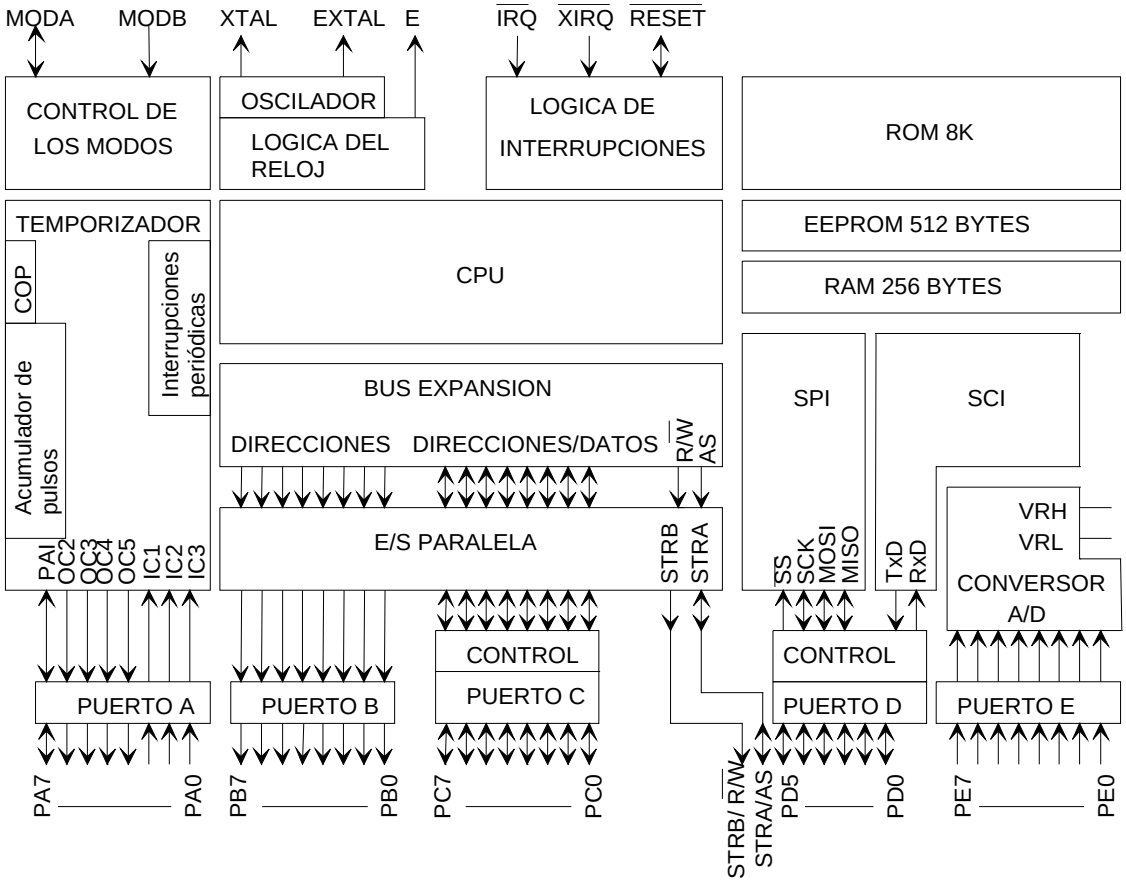
MICROCONTROLADOR MC68HC11

FUNDAMENTOS, RECURSOS Y PROGRAMACIÓN



CRISTINA DOBLADO ALCÁZAR
JUAN GONZÁLEZ GÓMEZ
ANDRÉS PRIETO−MORENO
JUAN JOSÉ SAN MARTÍN

MICROCONTROLADOR 68HC11:

FUNDAMENTOS, RECURSOS Y PROGRAMACIÓN

OBJETIVOS :

Con este libro se pretende cubrir el gran vacío que existe de información en castellano sobre el microcontrolador 68HC11. Esta basado en la experiencia que han ido adquiriendo los autores en el desarrollo de aplicaciones digitales. Por ello es eminentemente práctico, se ha evitado explicar con profundidad los mecanismos internos de funcionamiento centrándose sobre todo en los conocimientos básicos que hay que tener para poder utilizar todos los recursos internos. En el caso de buscar información detallada sobre algún recurso es aconsejable leer directamente el manual de referencia del microcontrolador 68hc11. Todos los ejemplos que aparecen han sido probados en la tarjeta entrenadora CT6811 desarrollada por los propios autores, pero son totalmente válidos para cualquier otra entrenadora.

Este texto se dirige a aquellas personas que ya tienen unos conocimientos básicos de programación en ensamblador y de sistemas digitales, para los cuales la comprensión del mismo se ve facilitada ya que no es objeto del libro explicar tales materias. Se ha usado un lenguaje coloquial para facilitar y amenizar la lectura, es muy común que los libros técnicos terminen siendo un plomo duro de roer y esa no es la intención de los autores.

Se espera que este libro sirva para fomentar el interés sobre este microcontrolador, el cual, por sus recursos internos y la facilidad de su programación, es muy usado tanto en la industria como en los centros de investigación. En el Massachusetts Institute of Technology (M.I.T) el 68HC11 se ha abierto un hueco y es ampliamente usado en el diseño de pequeños microbots autónomos, actividad que siguen de cerca los autores.

LOS AUTORES:

Cuando se empezó a escribir este texto los cuatro autores estudiaban los últimos cursos de la E.T.S.I Telecomunicación de la U.P.M. Se conocieron cuatro años antes al perseguir un objetivo común, el diseño y desarrollo de sistemas digitales autónomos, formando un equipo conocido con el nombre de 'GRUPO J&J'. Con un espíritu ingenieril empezaron a desarrollar sistemas modulares y de bajo coste que permitían construir sistemas más complejos como si de puzzles se tratara. Debido al interés de la gente se desarrolló un equipo entrenador de bajo coste basado en el microcontrolador 68HC11 y ahora se completa con este libro para satisfacer a todas aquellas personas, incluidos los propios autores, que lo han echado en falta en todo este tiempo.

El grupo durante 1997 centró su actividad en la construcción de pequeños robots, sistemas de tarjetas chip, y control remoto por internet. En 1998 este grupo empezó a formar parte de la empresa Microbótica continuando, entre otras actividades, con el diseño y construcción de sistemas digitales. Para las personas interesadas algunos de los trabajos se pueden ver en la dirección de WEB:

http://www.microbotica.es

y para contactar con ellos se puede utilizar la dirección de e−mail:

info@microbotica.es

Por último hay que agradecer a los integrantes del grupo que en su momento redactaron este libro, la labor realizada para hacer más sencillo el estudio y comprensión del microcontrolador MC68HC11.

Cristina Doblado Alcázar
Juan José San Martín
Andrés Prieto−Moreno Torres
Juan González Gómez

MICROBÓTICA

ÍNDICE

PROLOGO

A lo largo del siglo XX, la construcción de sistemas electrónicos ha exhibido un ritmo de crecimiento único; sus logros resultan impensables en cualquier otra rama de la tecnología. Una de las claves de este desarrollo ha sido la estandarización, miniaturización y abstracción de sus componentes, un proceso comenzó con la radiodifusión comercial, continua con la aparición de la televisión y recibe el impulso definitivo durante la Segunda Guerra Mundial. El tiempo transcurrido entre los relés, válvulas, transistores, circuitos impresos, circuitos integrados de densidades y tecnologías diversas, hasta llegar a los *chips* VLSI actuales es tan breve, que aún hoy es posible encontrar venerables ingenieros−dinosaurios que, a lo largo de su carrera profesional, han creado productos electrónicos sorprendentes con cada uno de estos dispositivos.

Uno de los componentes estándares más famosos han sido los pertenecientes a la "familia" TTL. La mayoría de los estudiante de Ingeniería y Ciencias Físicas los conocen: han sido obligados a perpetrar al menos un circuito digital con ellos. Estos dispositivos han ofrecido durante años una variedad de útiles funciones estandarizadas, magníficamente implementadas e ingeniosamente divididas en "rebanadas" de 4, 8 o más bits. Sin embargo, ya en la década de los ´70 el estilo de diseño TTL había entrado en crisis. En efecto, si bien la agrupación de estos componentes permitía al diseñador materializar cualquier producto imaginable, sus características finales en tamaño, consumo, fiabilidad y precio limitaban notablemente la posibilidad de ganar dinero mediante su comercialización. Adicionalmente, dado que cada producto debía diseñarse de manera artesanal, incluso la cantidad de ingenieros disponibles por aquella época para realizar estas tareas comenzó a ser insuficiente, constituyendo otro freno a las posibilidades de negocio.

Algunas personas comenzaron a preguntarse como salir de esta crisis. Por un lado, los avances en el proceso de integración permitían aumentar la complejidad de los circuitos integrados. Pero agregar complejidad a los dispositivos implicaba también aceptar que el campo de aplicación sería más restringido. Otra vez aparecían los aspectos económicos. La pregunta que flotaba en el aire era: ¿Cómo hacer un dispositivo lo suficientemente "grande" como para permitir la construcción de sistemas electrónicos lo suficientemente complejos, pero que a la vez que resultara lo suficientemente estándar para que su precio fuese lo suficientemente bajo? La respuesta a esta encrucijada la encontró Ted Hoff y la materializó Federico Fagin. Aparecía un componente paradigmático: el microprocesador.

Hoff y Fagin trabajaban en 1971 en una pequeña empresa dedicada al prometedor campo de la electrónica integrada, dos palabras cuyos apócopes daban nombre a la compañía: INTEL. Habían recibido el encargo de realizar un conjunto de *chips* para una calculadora electrónica. Pronto se dieron cuenta que con el *man−power* disponible (solo 4 ingenieros de diseño) no podrían terminar el trabajo a tiempo. Solo había una posibilidad de éxito: diseñar un único circuito, que fuera programable a la manera de los grandes computadores de la época. Así, el mismo *chip* podría ser usado en las diferentes tareas de la calculadora: leer el teclado, realizar las operaciones, exhibir los resultados, etc. con solo modificar su programa. El resto de la historia es conocido.

El microprocesador surge como el primer circuito integrado altamente complejo, totalmente estándar y relativamente fácil de utilizar. La combinación complejidad−programabilidad se ha extendido a dos famosos derivados del microprocesador: el Procesador Digital de Señal y el Microcontrolador. La fórmula también ha sido aplicada exitosamente en el otro componente−paradigma totalmente diferente: la FPGA. Este libro se centra uno de ellos, el microcontrolador.

El microcontrolador es uno de los componente actuales más "entretenidos", económicos y de mayor campo de aplicación. Por un precio cercano a 1K pesetas, se puede conseguir un *chip* que

integra una CPU, varios temporizadores, puertos de E/S, bloques de comunicación e incluso conversores analógico−digital. Solo le falta incluir una FPGA, al menos por ahora.... En lo que respecta al *software* de programación, es sencillo, potente, gratis, e ilimitado en la *www*. Más no se puede pedir.

El enfoque para la enseñanza de microcontroladores adoptado en este libro por el Grupo J&J es útil, eminentemente práctico y ligado al objetivo esencial del dispositivo: la construcción de productos electrónicos fiables en el mínimo tiempo posible. Para ello el texto incluye 43 ejemplos muy claros, pequeños programas que en conjunto cubren la mayoría de las aplicaciones de los microcontroladores y finalmente, 8 planos de circuitos. La cosa no queda allí, los *J&J* (que en realidad son, alfabéticamente, A&C&J&J) también han desarrollado una tarjeta de desarrollo, la CT6811 basada en el 68HC11 de Motorola, una interface de potencia y varias rutinas para su programación. No resulta extraño que estos productos hayan alcanzado cierta popularidad entre los aficionados a la mecatrónica−robótica y que el Grupo J&J se haya transformado en uno de los impulsores de esta disciplina en el área de Madrid.

Confío que este libro permitirá a los lectores aprender todos los trucos del tema en unas pocas horas.

Prof. Eduardo Boemo

Escuela Superior de Ingeniería Informática,
Universidad Autónoma de Madrid, 1998

1. INTRODUCCION

1.1. ¿Qué es un microcontrolador?

Un microcontrolador (MCU) es un circuito integrado que incorpora una unidad central de proceso (CPU) y una serie de recursos internos. La CPU permite que el microcontrolador pueda ejecutar instrucciones almacenadas en una memoria. Los recursos internos son memoria RAM, memoria ROM, memoria EEPROM, puerto serie, puertos de entrada/salida, temporizadores, comparadores, capturadores...

Se puede decir que es una evolución del microprocesador, al añadirle a este último las funciones que antes era necesario situar externamente con otros circuitos. El ejemplo típico esta en los puertos de entrada/salida y en la memoria RAM, en los sistemas con microprocesadores es necesario desarrollar una lógica de control y unos circuitos para implementar las funciones anteriores, con un microcontrolador no hace falta porque lo lleva todo incorporado, además en el caso de tener que ampliar el sistema ya ofrece recursos que facilitan esto.

En resumen, un microcontrolador es un circuito integrado independiente, que no necesita memoria ni puertos externos pues los lleva en su interior, que facilita la tarea de diseño y reduce el espacio, traduciéndose todo a una aplicación final más económica y fiable.

1.2. El microcontrolador 68HC11 de Motorola

En este libro se analiza el funcionamiento de los recursos internos del microcontrolador 68HC11 de Motorola. Existen otros muchos microcontroladores en el mercado, pero el 68HC11 destaca por sus recursos, simplicidad y facilidad de manejo.

Motorola describe al 68hc11 como un microcontrolador de 8−bits fabricado con tecnología HCMOS, con una frecuencia de bus de 2 Mhz y con una amplia lista de recursos internos. Es capaz de ejecutar todas las instrucciones del M6800 y M6801 y 91 más que se le han incorporado.

En la figura 1 (página siguiente) se muestran los modelos más importantes que componen la familia. La principal diferencia entre ellos es en la cantidad de RAM, ROM, EPROM y EEPROM.

Este texto se centra principalmente en el modelo A1, que es para el que se ha desarrollado la tarjeta CT6811. Los modelos A8 y A0 son muy similares, y la mayoría de los programas mostrados más adelante sirven también para ellos. Los recursos internos disponibles en el modelo A1 son:

- 256 bytes de memoria RAM
- 5 puertos de 8 bits, con pines de entrada, salida y de entrada/salida
- Conversor analógico−digital de 8 canales y 8 bits de resolución.
- Una UART para comunicaciones serie asíncronas (SCI)
- Un módulo de comunicaciones serie síncronas (SPI)
- 5 comparadores con salida hardware
- 3 capturadores de entrada
- Un acumulador de pulsos externos de 8 bits
- Temporizador principal de 16 bits
- Interrupciones en tiempo real
- 2 entradas de interrupciones externas
- Software en ROM para cargar un programa externo en la RAM interna

Muchos de los recursos no son accesibles simultáneamente. Por ejemplo, si se quiere utilizar memoria externa los puertos B y C se deben utilizar como bus de datos y direcciones. Por ello el 68hc11 se suele utilizar sin memoria externa, en caso de necesitar mayor memoria se suele recurrir a modelos de la familia que incorporan un mayor tamaño. En la CT6811 se puede sustituir el 68HC11A1 por el 68HC811E2 para pasar de 512 bytes de EEPROM a 2Kbytes, o por el 68HC11E9 para tener 12K de ROM grabable una vez.

DISPOSITIVO	RAM	ROM	EPROM	EEPROM	COMENTARIOS
MC68HC11A0	256	0	0	0	Temporizador de 16 bits 8 canales A/D SPI SCI
MC68HC11A1	256	0	0	512	
MC68HC11A7	256	8K	0	0	
MC68HC11A8	256	8K	0	512	
MC68HC11D0	192	0	0	0	Temporizador de 16 bits SPI SCI
MC68HC11D3	192	4K	0	0	
MC68HC711D3	192	0	4K	0	
MC68HC11EDO	512	0	0	0	Temporizador, SPI,SCI
MC68HC11E0	512	0	0	0	Temporizador de 16 bits 8 canales A/D SPI SCI
MC68HC11E1	512	0	0	512	
MC68HC11E8	512	12K	0	0	
MC68HC11E9	512	12K	0	512	
MC68HC711E9	512	0	12K	512	
MC68HC811E2	256	0	0	2K	Temporizador,SPI,SCI, 8 canales A/D
MC68HC11E20	768	20K	0	512	Temporizador de 16 bits, 8 canales A/D, SPI,SCI
MC68HC711E20	768	0	20K	512	
MC68HC11F1	1K	0	0	512	Bus no multiplexado, 8 canales A/D,SCI,SPI,44 CS

Figura 1: La familia 68hc11 de Motorola

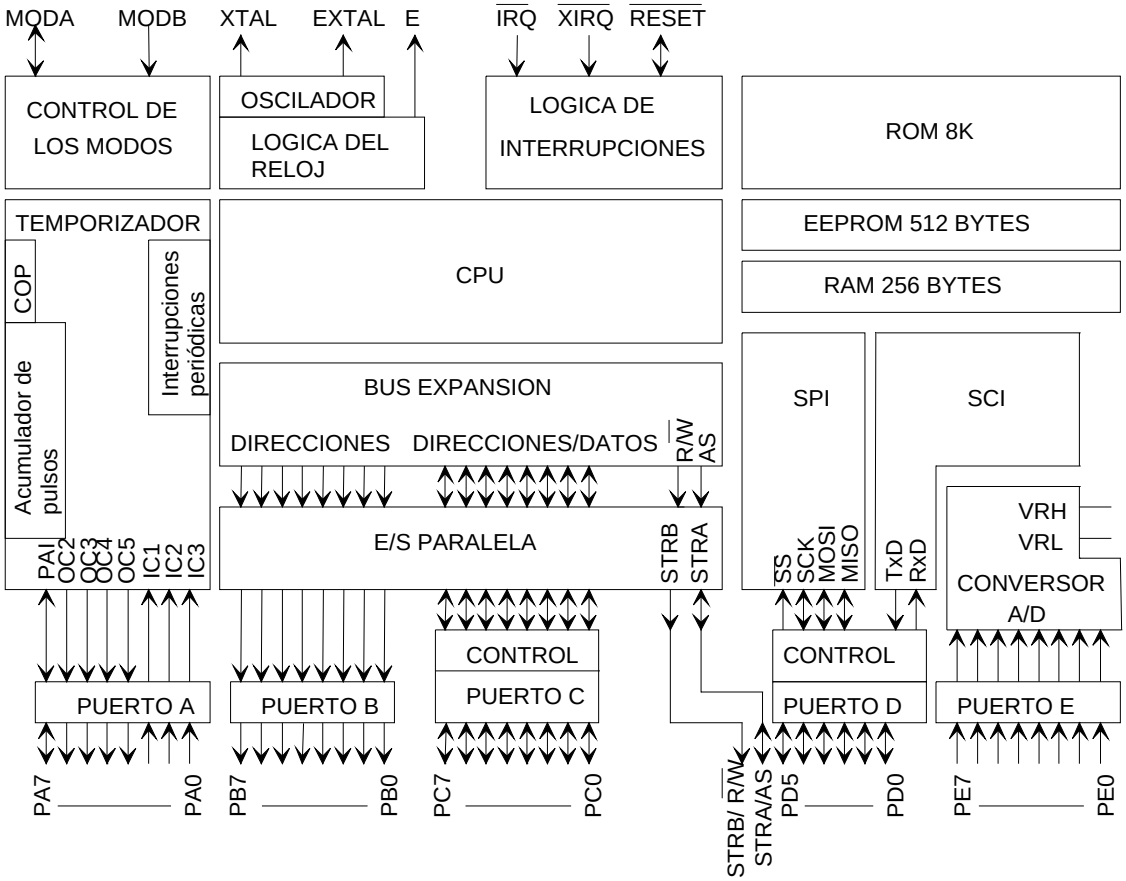
1.3. Tarjetas entrenadoras

Para desarrollar aplicaciones con el microcontrolador 68HC11 se necesita una tarjeta entrenadora. Estas tarjetas permiten cargar programas en la RAM interna del microcontrolador desde el PC. Una vez que el programa funciona correctamente se graba en la EEPROM interna o en una EPROM externa. Aquí se hace referencia a la tarjeta CT6811 del Grupo J&J. Esta entrenadora contiene lo mínimo que se necesita para poder trabajar con el 68HC11. Lo interesante es que puede servir para el desarrollo de aplicaciones (modo entrenador), y para usarla como producto terminado (modo autónomo).

La CT6811 funciona con una fuente de alimentación de 5 voltios, siendo también posible utilizar pequeños transformadores a 6 voltios, o incluso 4 pilas tamaño AA. La conexión al PC se realiza por el puerto serie y la tarjeta dispone de una serie de ’jumpers’ para configurar difrentes modos de trabajo.

1.4. Diagrama de bloques del 68HC11

2. ASPECTOS HARDWARE DEL 68HC11



Si bien Motorola advierte que este microcontrolador no ha sido diseñado para funcionar implantado en organismos vivos bajo ningún aspecto, así como en cualquier dispositivo que tenga relación directa con el sustento de la vida (marcapasos, etc...), su diseño hace hincapié en que se trate de un dispositivo "seguro".El que los microcontroladores se utilicen de forma mayoritaria en aplicaciones de control, obliga a estos dispositivos a prestar mucha atención a los sistemas de seguridad hardware.

En los sistemas de control profesionales hay que evitar cualquier inestabilidad ya que en su caso se puede producir una catástrofe. Sistemas de este tipo son los utilizados en la industria del automóvil, aeronáutica, sistemas de seguridad,etc....

Por esta razón fue necesaria la inclusión de sofisticados sistemas COP, que tiene por objetivo velar por la fiabilidad del chip. Las acciones que realiza son las de verificar que la ejecución de las instrucciones se mantiene en sincronía con el reloj, que este último mantiene la frecuencia correcta sin demasiadas variaciones, etc...

Asociado a su vez a temas de control y en particular el control en tiempo real, se encuentra el llamado Temporizador general (main timer), que no solo provee de interrupciones en tiempo real sino que gestiona todo el funcionamiento del integrado y sus posibles periféricos.

Para poder definir la CPU desde un punto de vista hardware se puede decir que se trata de una arquitectura LOAD/STORE (se trabaja con registros intermedios), y que por costumbre de su diseñador mantiene una arquitectura de memorias del tipo Princeton (las memorias de datos y código no están separadas, i8051...).

A partir de aquí es importante destacar que el micrcontrolador 68HC11, contiene una serie subsistemas internos que lo ponen entre los más completos del mercado actual.

2.1. El patillaje del MC68HC11

El MC68HC11 (MCU), dispone de 98 funciones de entrada y/o salida, las cuales se ven representadas por 52 pines en el caso de un encapsulado del tipo PLCC, o bien 48 para el del tipo DIP. Los diferentes encapsulados se muestran en la figura 3. Además, dependiendo del modelo MCU, se cuenta con diferentes tipos y tamaños de memorias (ROM, RAM, EPROM, EEPROM).

Debido al diseño de los circuitos internos del micro, muchas de las señales de salida son de **colector abierto**. El fabricante recomienda como resistencia de pull−up un valor de 4K7.

Cuando se monta un sistema digital basado en microcontrolador, existe siempre el peligro de que un mal diseño provoque no solo un mal funcionamiento sino un daño irreparable de los circuitos. Esto se debe a que a diferencia de la lógica digital habitual, los microcontroladores, en general, trabajan con varios tipos de señales, conversores A/D, salidas PWM, líneas de transmisión y un largo etcétera, lo que provoca que un mal conexionado pueda tener graves consecuencias.

Un mecanismo de protección frente a este tipo de conflictos es el adoptado por el 68HC11 donde varios de sus pines se encuentran dotados de circuitos internos de protección. Igualmente este tipo de soluciones tienen sus propias limitaciones por lo que nunca se debe bajar la guardia. Para saber más sobre estas protecciones acudir al manual de Referencia Técnica de Motorola.

Para facilitar la comprensión se clasifican todos los pines del microcontrolador en grupos de acuerdo a las funciones de los mismos, siendo estas agrupaciones las siguientes.

1. Alimentación: VDD, VSS.
2. Reloj: EXTAL, XTAL, E.
3. Reset: RESET.
4. Transmisión serie asíncrona: TxD, RxD.
5. Petición de interrupciones hardware: IRQ, XIRQ, IC1−3, PAI, STRA.
6. Modos de arranque: MODA, MODB.
7. Comparadores: OC1−5.
8. Capturadores: IC1−3, PAI.
9. Transmisión serie síncrona: SCK, MISO, MOSI, SS.
10. Puertos: PA0−7, PB0−7, PC0−7, PD0−3, PE0−3.
11. Conversores: AN0−7.
12. Buses: AD0−7, A8−15, AS, R/W.

Con esta clasificación, se intenta dar una vista general de todos los subsistemas hardware que conforman el microcontrolador y que tienen salida directa al exterior a través del encapsulado. Las descripciones son resumidas ya que en capítulos posteriores se detallan cada uno de los sistemas individualmente.

2.2. Pines de reloj

•**EXTAL** y **XTAL**: Son las conexiones de entrada para la introducción de una señal de reloj. El microcontrolador está diseñado para trabajar con osciladores de cristal de la forma que muestra la figura 2.

Esta frecuencia de reloj que es introducida en el MCU, es la encargada de regir el funcionamiento interno de los susbsisitemas que lo componen, por lo que no hay que confundirla con la señal de reloj de sus buses de datos ya sea internos o externos.

La velocidad máxima aconsejable por el fabricante está en torno a los 8Mhz. Es muy recomendable trabajar a esta frecuencia ya que de esta manera se consigue que el chip disponga de valores de velocidades para las transmisiones asíncronas compatibles con el estándar RS232c como los típicos 9600 baudios.

Para frecuencias altas, mayores de 1 MHz el circuito es el mostrado en la figura 3. Para trabajar a frecuencias más bajas es necesaria la inclusión de una resistencia más para lograr que la impedancia de salida aumente y no afecte mucho al MCU.

Los valores que da el fabricante para los componentes del circuito de reloj son: R=1−10MΩ y C1=C2=5−25pF. En el circuito de reloj de la tarjeta CT6811 los valores empleados son C1=C2=22pF y R=10MΩ. El valor del cristal es de 8 MHZ.

•**E**: Por este pin el MCU genera una señal de reloj, cuatro veces menor, que la que es introducida por medio del oscilador externo (EXTAL y XTAL). Esta señal, la utiliza el micro para gestionar todos los dispositivos periféricos del sistema., es decir, la señal E es la velocidad del bus. A modo de ejemplo, cabe notar que la tarjeta CT6811 dispone de una velocidad de bus de 2MHz.

Debido a la importancia de esta señal, y al uso que se hace de la misma se la volverá a citar más adelante.

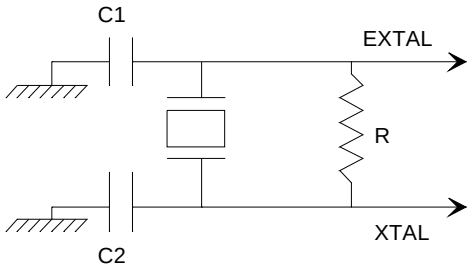
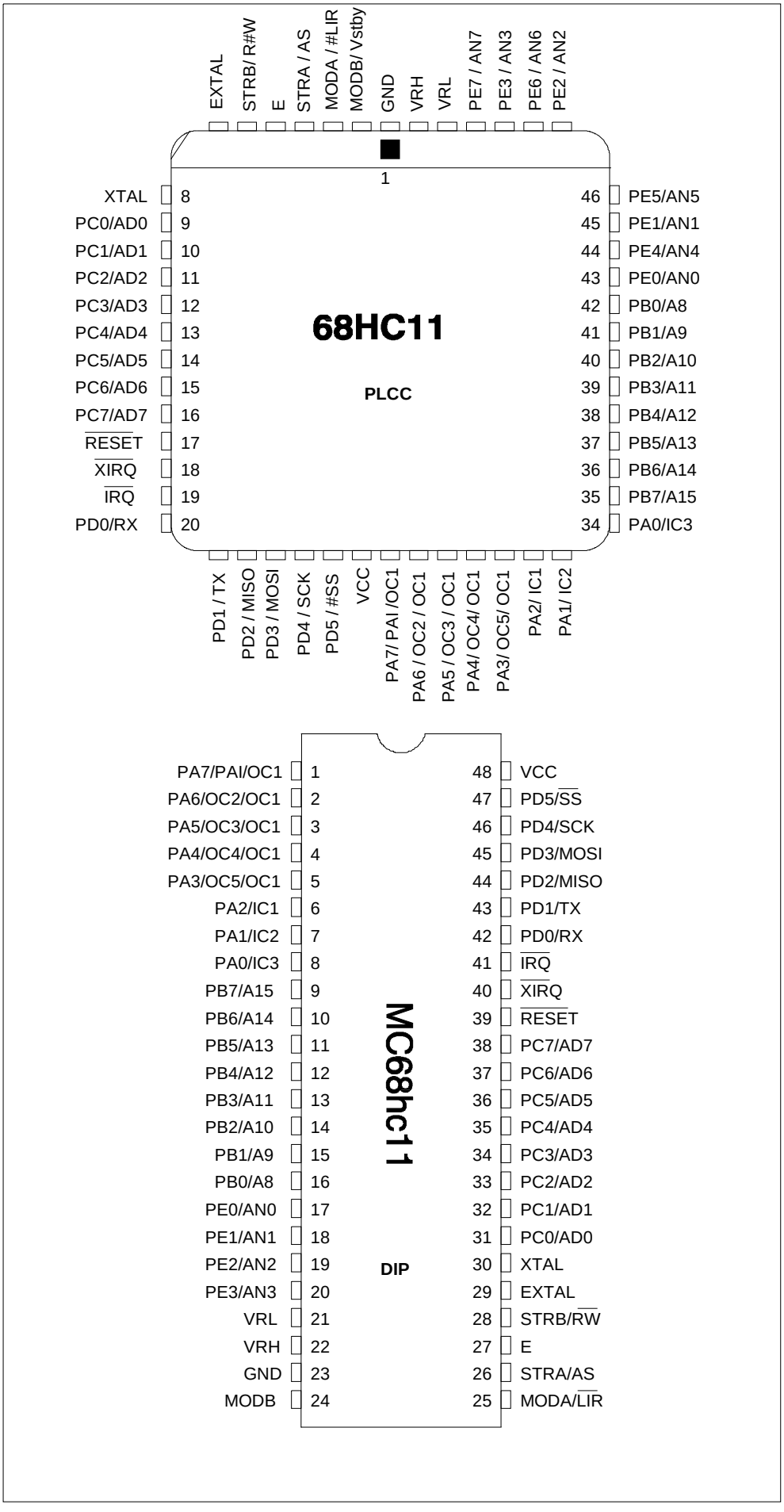


Figura 2: Circuito de reloj para frec. mayores de 1MHZ


EXTAL
STRB/ R#W
E
STRA / AS
MODA / #LIR
MODB/ Vstby
GND
VRH
VRL
PE7 / AN7
PE3 / AN3
PE6 / AN6
PE2 / AN2
1
XTAL 8
PC0/AD0 9
PC1/AD1 10
PC2/AD2 11
PC3/AD3 12
PC4/AD4 13
PC5/AD5 14
PC6/AD6 15
PC7/AD7 16
RESET 17
XIRQ 18
IRQ 19
PD0/RX 20
46 PE5/AN5
45 PE1/AN1
44 PE4/AN4
43 PE0/AN0
42 PB0/A8
41 PB1/A9
40 PB2/A10
39 PB3/A11
38 PB4/A12
37 PB5/A13
36 PB6/A14
35 PB7/A15
34 PA0/IC3
68HC11
PLCC
PD1 / TX
PD2 / MISO
PD3 / MOSI
PD4 / SCK
PD5 / #SS
VCC
PA7/ PAI /OC1
PA6 / OC2 / OC1
PA5 / OC3 / OC1
PA4/ OC4/ OC1
PA3/ OC5/ OC1
PA2/ IC1
PA1/ IC2
PA7/PAI/OC1 1
PA6/OC2/OC1 2
PA5/OC3/OC1 3
PA4/OC4/OC1 4
PA3/OC5/OC1 5
PA2/IC1 6
PA1/IC2 7
PA0/IC3 8
PB7/A15 9
PB6/A14 10
PB5/A13 11
PB4/A12 12
PB3/A11 13
PB2/A10 14
PB1/A9 15
PB0/A8 16
PE0/AN0 17
PE1/AN1 18
PE2/AN2 19
PE3/AN3 20
VRL 21
VRH 22
GND 23
MODB 24
48 VCC
47 PD5/SS
46 PD4/SCK
45 PD3/MOSI
44 PD2/MISO
43 PD1/TX
42 PD0/RX
41 IRQ
40 XIRQ
39 RESET
38 PC7/AD7
37 PC6/AD6
36 PC5/AD5
35 PC4/AD4
34 PC3/AD3
33 PC2/AD2
32 PC1/AD1
31 PC0/AD0
30 XTAL
29 EXTAL
28 STRB/RW
27 E
26 STRA/AS
25 MODA/LIR
MC68hc11
DIP

2.3. Pines de alimentación

•**VDD**: Es el pin de alimentación positiva, la cual debe ser el valor estándar de cinco voltios (el margen aceptado es de 4.5 a 5.5 voltios)

•**VSS**: Es la masa del MCU.

Para asegurar una buena robustez contra el ruido, es conveniente la conexión de un par de condensadores en paralelo entre VDD y VSS, con el fin de que estos anulen los posibles rizados provenientes de la fuente de alimentación y provocados por las conmutaciones internas del microcontrolador.

Dichos condensadores deben estar físicamente lo más cerca posible al chip. Los valores cedidos por el fabricante para dichos componentes son de 1 y 0.01 microfaradios.

2.4. Pines de reset

•**RESET**: Está señal, activa a nivel bajo, es bidireccional. El 68HC11 está preparado no sólo para recibir señales de "reset" por este pin sino que es el propio dispositivo el que es capaz de generar dicha señal para todos los periféricos que conformen el sistema digital.

De esta manera es posible que el diseñador trate a los subsistemas internos del microcontrolador como un conversor A/D, del mismo modo que a cualquier dispositivo externo como un display, ya que al recibir la señal de "reset, el MCU la transmitirá internamente al A/D, y generará la misma señal para el display.

Generar un "reset" admite una gran cantidad de diseños diferentes. Dependiendo del tipo de sistema digital conviene uno u otro.

Uno de los más utilizados en las tarjetas entrenadoras, es la conexión de un simple pulsador que al presionarlo habilite un camino de masa al pin E. A partir de aquí, las evoluciones son muy variadas.

En la figura 4 se muestra un ejemplo de un circuito de reset. Este circuito es el empleado en la tarjeta CT6811. Observar que se ha conectado el componente MC34064, de Motorola, en la entrada de reset del 68HC11. Este componente se emplea para que sólo lleguen tensiones de 0 ó 5 voltios a la entrada de "reset". Si no se emplea este componente, al conectar y desconectar la alimentación pueden aparecer tensiones transitorias entre 0 y 5 voltios que hagan que el micro ejecute instrucciones aleatorias, lo que provoca cosas inesperadas.

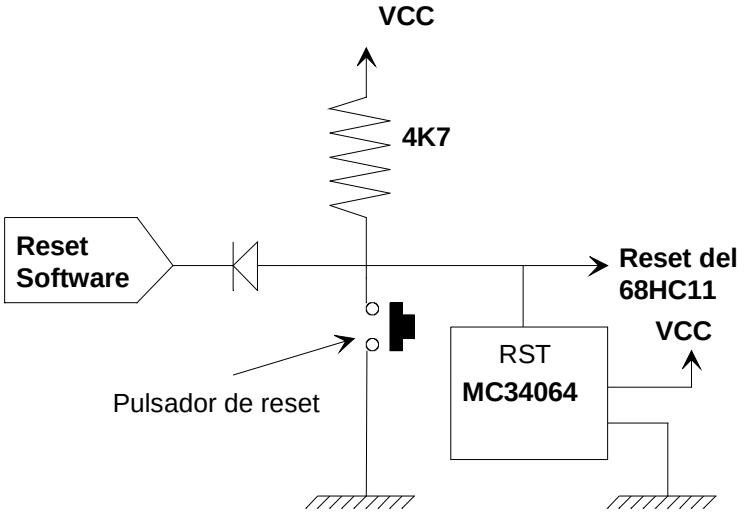


Figura 4: Circuito de reset empleado en la tarjeta CT6811

Cabe notar, que en este mismo circuito se dispone de una entrada de "reset software", esto brinda la posibilidad del control de la tarjeta, es este caso la CT6811, dese otro dispositivo que tenga acceso a ese pin como puede ser un PC.

Por último destacar que este pin es de colector abierto y para deshabilitarlo (que no sea un "reset" constante) debe tener una resistencia de pull−up de 4K7.

2.5. Pines de transmisión serie asíncrona

•**TxD** y **RxD**: Son los pines de transmisión y recepción de datos serie asíncronos. El 68HC11 trabaja con niveles TTL, por lo que será necesario un circuito de adaptación si se quiere comunicar con un dispositivo que utilice otro tipo de norma como por ejemplo un PC (norma RS−232C). Un integrado que realiza esta adaptación es el MAX232. En la figura 5 se muestra el diagrama de bloques de la conexión del 68HC11 a un PC. En la figura 6 se muestra cómo se debe configurar el MAX232 para que realice correctamente la función de adaptación. Los condensadores empleados son electrolíticos de 22µF y 25V.

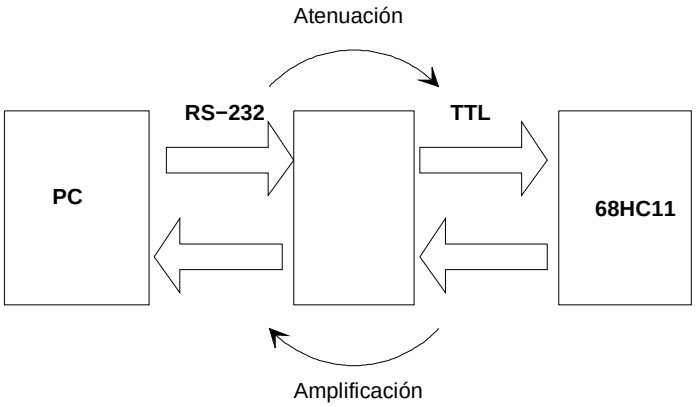


Figura 1: Funcionamiento del integrado MAX232

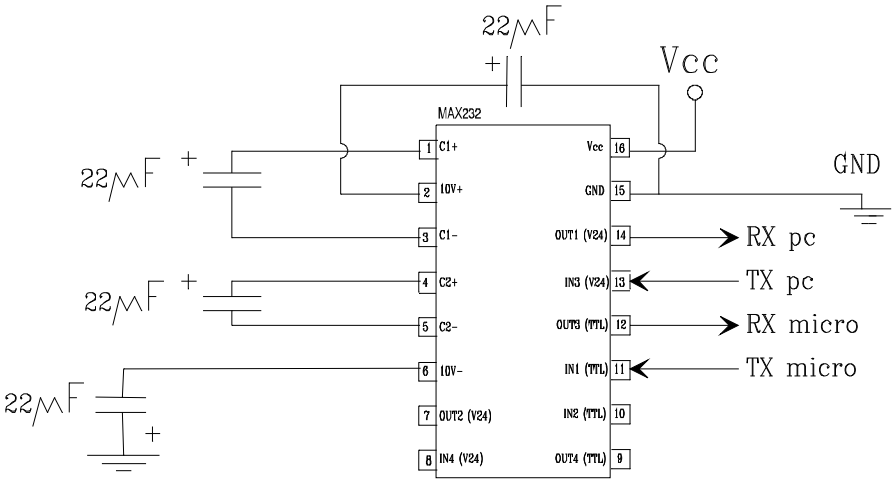


Figura 2: Configuración del MAX232 para conectar el 68HC11 a un PC (norma RS−232C)

Una de las ventajas de este dispositivo o similar es que no hacen uso de alimentaciones externas, diferentes a las habituales en estos sistemas (5V), evitando de esta manera aumentar la complejidad de los sistemas de alimentación.

2.6. Pines de los capturadores

•**IC1, IC2** e **IC3**: Estos tres pines representan las tres entradas de los tres capturadores independientes que incorpora el 68HC11. Mediante ellos, es posible recoger cualquier cambio de nivel que se produzca en la circuitería exterior. (Los capturadores son tratados en la sección 4.8).

•**PAI**: Este pin representa la entrada a un acumulador de pulsos, mediante el cual es posible medir la duración y contar flancos activos de señales externas al microcontrolador.

2.7. Pines de petición de interrupciones externas

•**IRQ**: Este pin provee al microcontrolador de una entrada de interrupción enmascarable, activa a nivel bajo, y de colector abierto lo que permite conectar varios dispositivos a la vez. Debe permanecer conectada a una resistencia de pull−up (4K7).

•**XIRQ**: Del mismo modo que el pin anterior, este permite producir todas las peticiones de interrupción con la salvedad de que estas son no enmascarables. Uno de los usos más comunes de esta entrada es para activar alguna rutina de servicio referente a caída de la alimentación del sistema, o cualquier otra función de carácter prioritario como esta última.

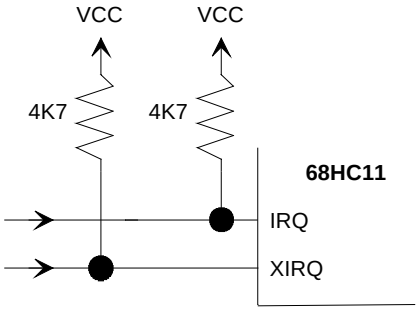


Figura 3: Peticiones de interrupción externas

•**IC1, IC2 e IC3**: Si bien su funcionamiento "normal", como capturadores de entrada, se encuentra explicado en la sección 4.8, cabe destacar la posibilidad que presentan estos pines para ser utilizados como entradas de interrupciones externas. A partir de ellas es posible construir varios niveles de petición, y no tener que agregar demasiado hardware externo (un decodificador de prioridad). Estos "capturadores" configurados adecuadamente, generan una interrupción cada vez que perciben de la entrada, un cambio de nivel (según configuración).

•**PAI** y **STRA**: Estos dos pines, al igual que los anteriores presentan esta misma propiedad (detectores de flancos), pero tienen la salvedad de ser menos versátiles, ya que por ejemplo al utilizar STRA para el manejo de interrupciones, se limita el MCU al modo NORMAL (no extendido, *Single−Chip)*, al no poder conectar memorias externas, etc..,ya que se pierde el pin AS (validación de dirección).

2.8. Pines de configuración de los modos de arranque

•**MODA** y **MODB**: Estas dos señales son tenidas en cuenta por el microcontrolador únicamente en el momento del arranque del sistema. Según el nivel al que se encuentren, el 68HC11 se configurará en alguno de los 4 modos de que dispone. Mediante software es posible cambiar de un modo a otro (ver registro HPRIO, sección 5.3.4).

MODA	MODB	Modo de arranque
0	0	Especial Bootstrap
0	1	Normal
1	0	Especial extendido (Test)
1	1	Normal extendido

La tarjeta CT6811 permite la selección del modo de arranque mediante unos conmutadores de configuración.

2.9. Pines de los comparadores

− **OC2, OC3, OC4** y **OC5**: Estos cuatro pines representan las respectivas salidas hardware de cuatro de los cinco comparadores internos de los que está provisto el microcontrolador. Una vez comparado un valor previamente almacenado en un registro de 16 bits con el valor del temporizador principal, en el caso de producirse la igualdad y cuando las salidas hardware estén habilitadas, el pin correspondiente al comparador activado muestra a la salida el nivel para el cual esté configurado. Los comparadores se tratan en profundidad en la sección 4.7.

− **OC1**: Si bien posee un solo nombre, aquí se ven representados nada menos que cinco pines de salida. Del mismo modo que en el caso anterior se trata de un comparador, pero que su salida hardware no es de un solo canal, sino de cinco!. (Más información en sección 4.7.4)

2.10. Pines de transmisiones serie síncrona

•**SCK**: Este pin se refiere a la señal de reloj que comanda cualquier transmisión síncrona. Puede ser tanto de salida como de entrada, según que el microcontrolador trabaje como maestro o esclavo respectivamente.

•**MISO y MOSI**: (Master In Slave Out) (Master Out Slave In), estos dos pines son las dos vías por donde van a fluir los datos. Para el caso en que se configure el 68HC11 para trabajar como Maestro frente a otro dispositivo, el cual hace de esclavo, el pin MOSI es el que funciona como salida de datos desde el maestro al esclavo y el pin MISO es la vía contraria, es decir por donde el maestro recibe los datos del esclavo.. En la figura 8 se muestra el estado de los pines del SPI cuando el 68HC11 funciona como maestro y como esclavo.

Mientras que si el MCU se usa como Esclavo, los datos del Maestro se reciben por el MOSI y tienen su vía contraria por MISO. Ver figura.

Para el caso en que a lo largo de las transmisiones el 68HC11 varíe su papel de maestro o esclavo, los pines cambiarán su función de forma automática.

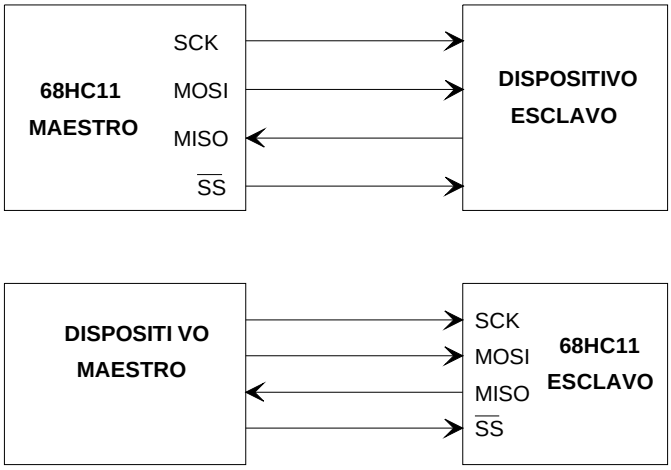


Figura 4: El 68HC11 funcionando como maestro y como esclavo

−**SS**: Señal que se activa a nivel bajo y que tiene diferentes funciones según se haya configurado el micro como Maestro o Esclavo.

68HC11 como esclavo: Cuando SS=0 se activa el sistema de transferencias síncronas, en caso contrario la señal SCK no es tenida en cuenta y MISO está en estado de alta impedancia.

68HC11 como maestro: SS no repercute dentro de la transmisión propiamente dicha y puede ser utilizada como detección de errores, o de propósito general. Normalmente se utiliza para activar al esclavo, aunque cualquier bit de cualquier otro puerto sirve para esta función.

2.11. Pines de los puertos de E/S

La integración y la potencia de este microcontrolador obliga a un alto grado de multiplexación de las funciones que soporta, por tanto, si bien la cantidad de puertos existentes en el chip es grande y variada, estos se ven a veces desactivados ya que son utilizados para otros fines como comparadores, capturadores, control de transmisiones asíncronas, etc...de tal manera que el número de puertos netos, puede hasta resultar escaso para algunas aplicaciones.

Esto, que siempre es dependiente del sistema en cuestión, tiene una sencilla solución ya que el 68HC11 está preparado para soportar diferentes tipos de expansiones.

Estructura general de los puertos.

Puerto	A	B	C	D	E
Bit 0	entrada	salida	bidireccional	entrada	entrada
Bit 1	entrada	salida	bidireccional	salida	entrada
Bit 2	entrada	salida	bidireccional	bidireccional	entrada
Bit 3	salida	salida	bidireccional	bidireccional	entrada
Bit 4	salida	salida	bidireccional	bidireccional	entrada
Bit 5	salida	salida	bidireccional	bidireccional	entrada
Bit 6	salida	salida	bidireccional	−	entrada
Bit 7	bidireccional	salida	bidireccional	−	entrada

2.12. Pines de los buses

Cuando el microcontrolador se configura para funcionar en el *modo extendido*, es decir, con la capacidad de redireccionar 64K de memoria, se ve obligado a generar un bus de direcciones, uno de datos y uno de control. De esta manera, el 68HC11 deja que su CPU interna tenga acceso al exterior, por lo que a partir de aquí, de alguna manera, el microcontrolador comienza a funcionar como un microprocesador .

Dicho "nuevo" microprocesador tiene una potencia razonable regida por un bus de datos de 8 bits, y un bus de direcciones de 16 bits, por lo que su espacio de direccionamiento es de 64Kbytes de memoria plana.

Este mapa de memoria es plano y comparte espacio con los puertos que se incorporen a nuestro sistema y todos los registros internos de MCU.

En caso de que se superpongan registros internos del microcontrolador con dispositivos externos como pueden ser secciones de memoria, etc... el gestionador de bus, da prioridad a los internos, dejando de lado los restantes.

Mediante la utilización de la totalidad del PUERTO B, el micro lleva al exterior la PARTE ALTA DEL BUS DE DIRECCIONES, y por medio del PUERTO C se presenta al exterior de forma multiplexada la PARTE BAJA DEL BUS DE DIRECCIONES, y el BUS DE DATOS. Esta multiplexación obliga a tener que agregar un hardware adicional que permita poder separar ambos buses. Para esto se puede utilizar un registro tipo latch triestado (para no cargar el circuito) ya sea por flanco (74374) o por nivel (74373, recomendado por el fabricante). Con este latch se intercepta la salida del puerto c de tal forma que se capture la parte baja del bus de direcciones, y no se solape con el dato.

Ahora bien, esta multiplexación viene regida por una señal perteneciente al bus de control llamada E . Por este pin se va a obtener una señal de reloj que es la que se entrega a los periféricos y que por tanto comanda el **ciclo de bus** del sistema, de tal forma que cuando dicha señal E se encuentra a nivel bajo, por el puerto B se direcciona la parte alta del bus A, y por el puerto C la parte baja, los cuales serán capturados en el latch. Finalmente, cuando E pase, en la segunda parte del ciclo del bus, al nivel alto, el puerto C presenta a su salida al bus de datos completando de esta forma la demultiplexación. En la figura 9 se muestra cómo realizar esta demultiplexación.

- **ADO−AD7**: Señales de los buses (A y D) multiplexadas.
- **A8−A15**: Señales de la parte alta del bus de direcciones (A).
- **AS**: Señal de validación de la dirección puesta en el bus A, la cual es muy útil para validar la captura del latch, ya que al activarse informa que el bus de direcciones está completo.(AS pertenece al bus de control)

•**R/W**: Señal de Lectura (nivel alto) y Escritura (nivel bajo), la cual se comporta de manera idéntica a cualquier microprocesador. (R/W pertenece al bus de control)

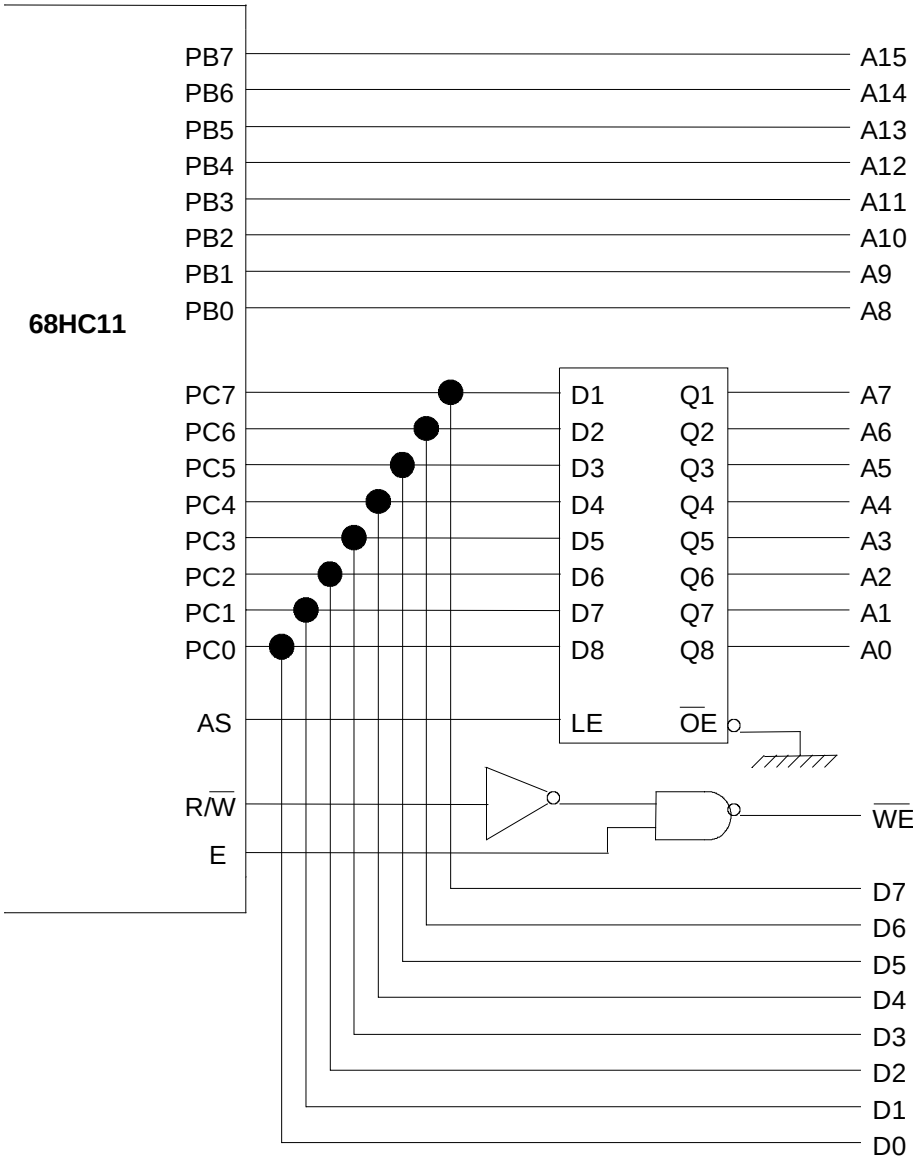


Figura 5: Demultiplexación del bus de datos y direcciones

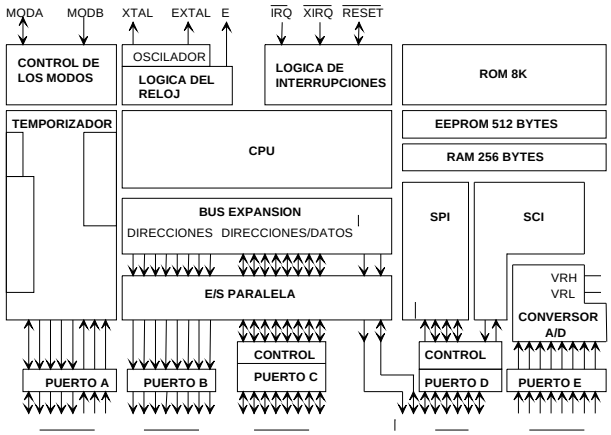
2.13. Pines de los conversores A/D

•**AN0−7**: Este sistema es uno de los más populares a la vez que más delicado. Se basa en un conversor Analógico Digital de 8 bits que cuenta con 8 canales a los cuales puede acceder de 4 en 4. Las velocidades de muestreo están sujetas a la velocidad de reloj del microcontrolador, a la vez que las señales de referencia se pueden fijar externamente, es decir, disponer al micro para que convierta a digital señales analógicas que varíen entre 0 y +6 voltios.

Es recomendable la utilización precavida del sistema ya que una entrada de tensión fuera de los niveles de referencia prefijados provoca un corto interno y la alta probabilidad de que se destruya, al menos, el canal en cuestión.

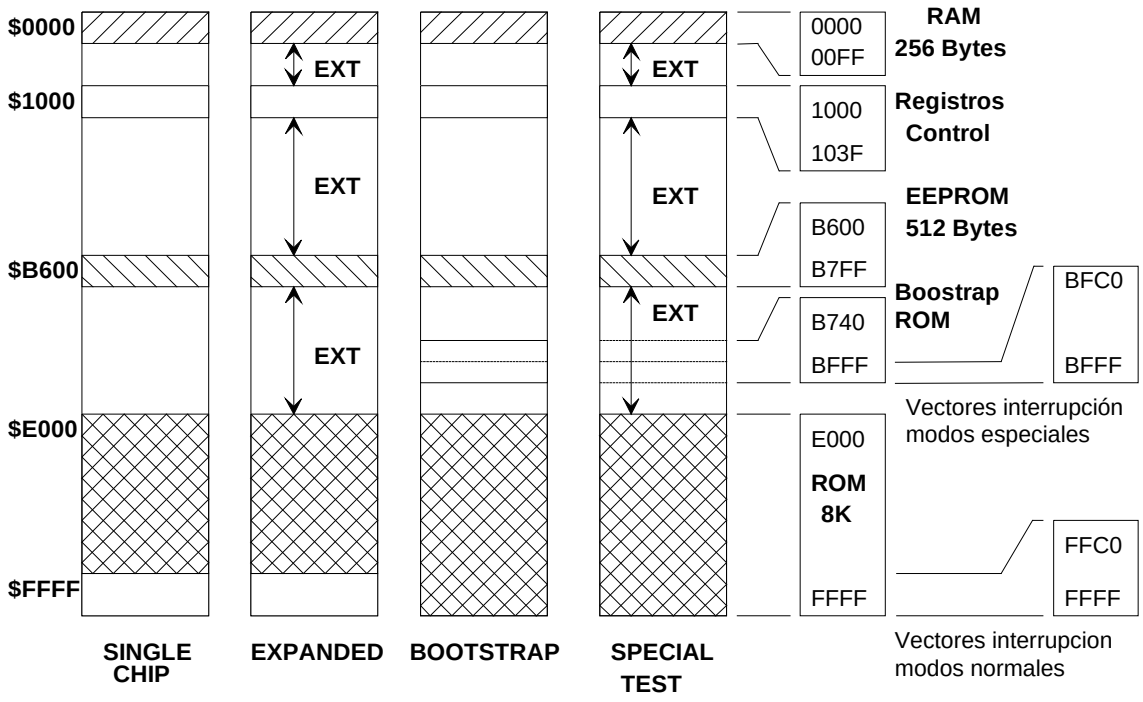
Por último cabe destacar, que dependiendo del rango de tipo de señales analógicas que se esperan recibir, es siempre aconsejable el uso de filtros que acondicionen las mismas para una mejor conversión.

3. PROGRAMACIÓN DE LA CPU DEL 68HC11



3.1. Los modos de funcionamiento del 68HC11

El 68HC11 puede funcionar en 4 modos diferentes: Single chip, expanded, bootstrap y special test. En cada modo se dispone de un mapa de memoria diferente, como se muestra en la figura 10.



- **Single chip:** En este modo de

Figura 10: Configuración del mapa de memoria para los diferentes modos de funcionamiento del 68HC11.

funcionamiento, el mapa de memoria del 68HC11 está constituido por la memoria RAM, la memoria EEPROM, los registros de control y la memoria ROM. Este modo está pensado para funcionar cuando existe un programa grabado en la ROM, de tal manera que al arrancar se comience a ejecutar el programa indicado por los vectores de interrupción que se encuentran en ROM

- **Expanded**: Además del mapa de memoria del modo single chip, es posible acceder al resto de las posiciones de memoria conectando memorias externas. El precio a pagar es que se pierden dos puertos de E/S, el puerto B y C, que se utilizarán como bus de datos y direcciones. En este modo se puede utilizar la memoria ROM interna, pero también es posible deshabilitar esta ROM y acceder a memoria externa y con ello a los vectores de interrupción que se encuentren en esa memoria externa.

- **Bootstrap:** Este modo difiere del single chip en que los vectores de interrupción no se encuentran en la memoria ROM de 8K sino que se encuentran en otra memoria ROM, llamada ROM de arranque. Al arrancar en este modo, automáticamente comienza a ejecutarse el programa BOOTSTRAP que se encuentra en ROM.

- **Special test**: Igual que el modo Bootstrap con la salvedad de que se puede acceder a memoria externa. Este modo se utiliza para realizar pruebas de fábrica. En este modo especial se tiene acceso a determinados registros de control que en otros modos están protegidos.

Los modos de funcionamiento se pueden configurar de dos formas diferentes: configuración hardware y configuración software. La configuración hardware consiste en colocar unos determinados niveles lógicos en las patas **moda y modb** (apartado 2.8) del 68HC11. Al realizar un reset del 68HC11 el micro arrancará en el modo especificado por las tensiones de las patas moda y modb. La configuración software del modo de funcionamiento se basa en la modificación del registro HPRIO situado en la dirección $103C. Este registro se describe con detalle en la sección 5.3.4.

3.2. Registros de la CPU

La CPU del MCU dispone de 2 registros acumuladores de 8 bits, que se unen para formar el registro D de 16 bits, siendo el acumulador A la parte alta y el acumulador B la parte baja. Además dispone de 2 registros para direccionamiento indexado X,Y ambos de 16 bits. El puntero de pila y el contador de programa son también de 16 bits, lo que permite que la longitud máxima de un programa sea de 64Kbytes, que es el espacio máximo direccionable por el MCU. El registro CCR es el llamado registro de estado, que contiene unos bits de especial importancia que reflejan el estado de la CPU.

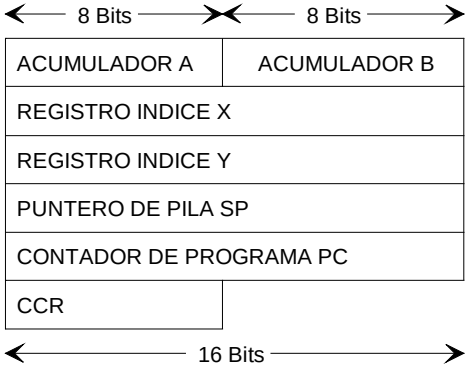


Figura 11: Registros del 68HC11

El **puntero de pila** debe ser inicializado por el usuario. La pila "crece" desde direcciones altas hacia direcciones bajas, por lo que al introducir un elemento en la pila, SP se decrementa en 1 ó 2 bytes dependiendo del tamaño del dato metido en la pila. Al sacar un elemento de la pila, SP se incrementa.

El contador de programa **PC** se va incrementado según se van ejecutando las instrucciones. Por tanto, los programas se ejecutan desde direcciones bajas a altas y la pila crece de direcciones altas a bajas. Es importante dar a SP un valor "seguro" de tal manera que la pila no se solape con el código, si es que el código se encuentra en RAM.

El registro **CCR** es de 8 bits. Cada bit tiene una letra asignada y representa una situación diferente del estado de la CPU. Se presenta en la figura 12.

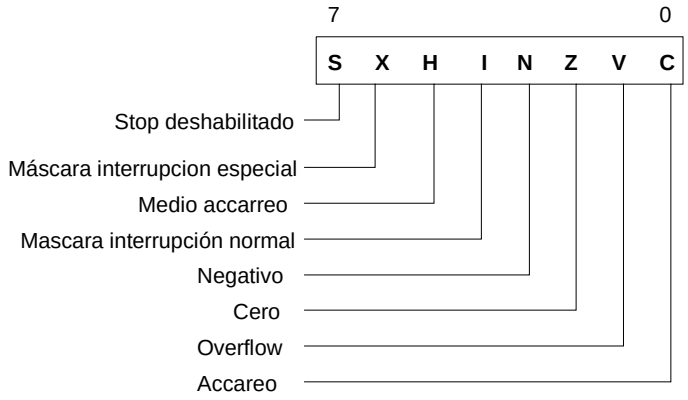


Figura 12: Registro CCR

3.3. MODOS DE DIRECCIONAMIENTO.

Existen 6 modos de direccionamiento distintos. Los modos de direccionamientos son distintas formas que tiene la CPU de acceder a los datos que están en memoria.

3.3.1. Direccionamiento inmediato

El dato al que se hace referencia se encuentra "dentro" de la instrucción, no es necesario acceder a memoria. El dato puede ser de 1 ó 2 bytes. Este modo de direccionamiento se indica mediante el signo #.

Ejemplo: **LDAA #08** Esta instrucción carga el valor decimal 8 en el acumulador A. El valor 08 se encuentra a continuación del código de instrucción de LDAA. La instrucción LDAA tiene el código $86, por tanto, en memoria esta instrucción queda representada mediante los valores: (Se supone que esta instrucción comienza en la dirección $0000).

Dir. memoria	contenido	nemónico
0000	$86	LDAA
0001	$08	#8

3.3.2. Direccionamiento extendido

El dato se encuentra en la dirección de memoria especificada. El dato puede estar en cualquier posición de la memoria dentro del límite de las 64Kb, por lo que la dirección ocupa 2 bytes.

Ejemplo: **LDAA $FC00** Esta instrucción carga en el acumulador el contenido de la dirección $FC00. La dirección del dato se almacena después del código de la instrucción y ocupa 2 bytes. Las instrucciones con este modo de direccionamiento ocupan 3 bytes (1 byte para el código de la instrucción y 2 bytes para la dirección).

3.3.3. Direccionamiento directo

Este modo de direccionamiento es exactamente igual que el anterior con la salvedad de que sólo actúa con direcciones comprendidas entre $00−$FF (256 primeros bytes de la memoria). La utilidad de este modo es que sólo necesita 1 byte para especificar la dirección del dato, con lo que se ahorra espacio y tiempo.

Ejemplo: **(1) LDAB $FC00**
(2) LDAB $05

Ambas instrucciones cargan en el acumulador B el contenido de la dirección especificada. En el caso de la instrucción 1, se utiliza direccionamiento extendido y la instrucción ocupa 3 bytes. En el caso de la instrucción 2, como la dirección es menor que $FF se utiliza direccionamiento directo y la instrucción ocupa 2 bytes.

Se saca una conclusión muy importante al comparar ambos modos de direccionamiento: siempre que sea posible conviene usar el direccionamiento directo, es decir, **"SITUAR LAS VARIABLES EN LAS DIRECCIONES BAJAS DE LA MEMORIA, EN EL ESPACIO $00−$FF".** De esta manera, todas las instrucciones que hagan referencia a variables, utilizarán direccionamiento directo y se ahorrarán muchos bytes de memoria.

¿Quién decide el modo de direccionamiento?. Es el propio ensamblador. Al encontrarse el ensamblador con la instrucción 1 (LDAB $FC00) sabe que la dirección es superior a $FF y que debe utilizar direccionamiento extendido. Al encontrarse con la instrucción 2, la dirección es menor que $FF y por tanto utiliza direccionamiento directo ahorrando un byte.

3.3.4. Indexado

Este modo de direccionamiento se utiliza para acceder a TABLAS (Arrays, cadenas....) en la memoria. El dato se busca de la siguiente forma: Se toma la dirección del registro índice (X ó Y), se le suma un desplazamiento (offset) de 8 bits y el contenido de esa dirección es el dato buscado.

Este modo de direccionamiento se especifica colocando como argumentos en la instrucción un offset, una coma y el registro índice.

Ejemplo: **LDAB 5,X** Esta instrucción carga en el acumulador B el contenido de la dirección especificada por X más un offset de 5. Se simboliza de la siguiente manera:
dir = (X) + 5 , siendo dir la dirección que contiene el dato.

Para el caso particular de tener un offset de 0, es decir, que se quiera acceder a la dirección contenido en X, se puede especificar de las siguientes maneras:

```
LDAB  0,X
LDAB  ,X
LDAB  X
```

Las 3 instrucciones son equivalentes.

Este modo de direccionamiento es muy interesante porque permite acceder a cualquier dirección de memoria ($0000−$FFFF) como en el direccionamiento extendido pero las instrucciones sólo ocupan 2 bytes, como en el direccionamiento directo. Además, el registro X se puede variar (incrementar, decrementar...) con lo que se obtiene una gran flexibilidad a la hora de acceder a tablas de datos.

Para acceder a los **registros de configuración** del microcontrolador, que se encuentran en las direcciones $1000−$103F, es conveniente utilizar este modo de direccionamiento porque así las instrucciones ocupan menos bytes. En el registro índice se introduce la dirección $1000 correspondiente al comienzo de los registros de configuración del microcontrolador, y sólo es necesario especificar el desplazamiento:

Instrucciones	**Comentarios**	**Tamaño instrucción**
LDX $1000	; X = $1000	(3 Bytes)
STAA 1,X	; Meter A en dir. $1001	(2 Bytes)
STAB $2C,X	; Meter B en dir. $102C	(2 Bytes)
STAA $10,X	; Meter A en dir. $1010	(2 Bytes)

Para realizar esto mismo con direccionamiento extendido sería:

```
STAA $1001      ; Meter A en dir. $1001  (3 Bytes)
STAB $102C      ; Meter B en dir. $102C  (3 Bytes)
STAA $1010      ; Meter A en dir. $1010  (3 Bytes)
```

Para acceder a 3 registros de control distintos, ambos trozos de código ocupan 9 bytes de memoria. Pero si se pretende acceder a más de 3 registros de control, que suele ser lo más habitual, se ahorra memoria utilizando direccionamiento indexado.

El offset aplicado es de 8 bits y **sin signo** por lo que el offset máximo es de 256 bytes.

3.3.5. Direccionamiento relativo

Este modo de direccionamiento sólo se utiliza con las instrucciones de **bifurcación.** Estas indican a la CPU que realice un salto de tantos bytes hacia adelante o hacia atrás. El desplazamiento **tiene signo** y es de un byte por lo que las bifurcaciones sólo se pueden hacer de 128 bytes hacia atrás ó 127 bytes hacia adelante:

```
Ejemplo:    Bucle   ....
                    ....
                    BNE bucle
```

El programador no necesita calcular el salto a efectuar ,lo realiza automáticamente el ensamblador. Sin embargo, es importante saber que los saltos con instrucciones BR sólo se pueden realizar hacia posiciones de memoria que estén a menos de 128 bytes por debajo y a menos de 127 bytes por arriba. Si se sobrepasa el límite el ensamblador dará un mensaje de error.

```
Ejemplo:    Bucle   Inst1          ; Dirección 0
                    ......
                    ......
                    BNE bucle      ; Dirección 200
```

Este programa daría error puesto que el salto que la bifurcación realiza es de más de 128 bytes hacia arriba. La ventaja es que todas las instrucciones BR (BRANCH) ocupan 2 bytes. La desventaja es que sólo se pueden hacer bifurcaciones relativamente cortas. Para realizar bifurcaciones a cualquier posición de la memoria se utilizan las instrucciones JUMP (JMP y JSR) , que ocupan 3 bytes.

3.3.6. Direccionamiento inherente

Los operandos se encuentran en registros de la CPU. Por el código de la instrucción la CPU sabe de qué registros se trata.

```
Ejemplo:    CÓDIGO      MNEMÓNICO
              $1B          ABA          ; A:=A+B
              $5C          INCB         ; B:=B+1
              $08          INX          ; X:=X+1
```

La primera instrucción, de código $1B, suma el contenido del acumulador A y B y el resultado lo introduce en el acumulador A. La segunda incrementa el acumulador B y la tercera el registro de índice X. Estas instrucciones sólo ocupan un byte y por ello conviene abusar de ellas.

3.4. JUEGO DE INSTRUCCIONES

Las instrucciones se dividen en distintos grupos. Todas las instrucciones tienen dos campos: uno es el mnemónico y el otro es el dato o la dirección a la que hace referencia la instrucción. Este campo es opcional.

3.4.1. Instrucciones de carga, almacenamiento y transferencia

- **CARGA:** Estas instrucciones permiten introducir un nuevo valor en los registros, leer una posición de memoria, un puerto etc...

LDAA : Introducir un dato de 8 bits en el acumulador A

```
Ej.     LDAA #30          ; A:=30 (Direccionamiento inmediato)
        LDAA $1000        ; Introducir en A el contenido de la dirección $1000.
```

LDAB : Introducir un dato de 8bits en el acumulador B

LDD : Introducir un dato de 8 ó 16 bits en el doble acumulador D (Formado por A y B yuxtapuestos)

```
Ej.   LDD #$FFCC        ; D:=$FFFF −−> A:=$FF; B:=$CC
      LDD #$10;         ; D:=$0010 −−> A:=$00; B:=$10
```

LDX : Introducir un dato de 16 bits en el registro índice X

```
Ej.   LDX #$1000        ; X:=$1000
      LDX 3,Y           ; Meter en X el contenido de la dirección Y+3
      LDX 5,X           ; Meter en X el contenido de la dirección X+3.
```

LDY :Introducir un dato de 16 bits en el registro de índice Y

LDS :Introducir un dato de 16 bits en el SP (puntero de pila). Esta instrucción hay que utilizarla al menos una vez en nuestros programas para inicializar la pila.

Ej. LDS #$FC00 ;Inicializar la pila a partir de la dirección $FC00 hacia abajo

CLRA :Borrar el contenido del acumulador A. Esta instrucción hace lo mismo que LDAA #0, con la diferencia de que el direccionamiento es inherente y sólo ocupa 1 byte, mientras que LDAA #0 ocupa 2 bytes.

CLRB :Borrar el contenido del acumulador B.

- **ALMACENAMIENTO**: Estas instrucciones permiten alterar una posición de memoria, un puerto, registros internos, etc.

STAA :Almacenar el acumulador A en una dirección de memoria.

```
Ej.   STAA  $1000       ; Mandar el acumulador por el puerto A
```

STAB :Almacenar el acumulador B.
STD :Almacenar el doble acumulador D. (16 bits)
STX :Almacenar el registro de índice X(16 bits)
STY :Almacenar el registro de índice Y (16 bits)
STS :Almacenar el puntero de pila SP.

CLR :Poner a cero el contenido de una dirección de memoria.

```
Ej.   CLR  $1000        ; Mandar un 0 por el puerto A
```

- **TRANSFERENCIAS**: Permiten transferir datos entre registros y registros y memoria. El direccionamiento es inherente por lo que no es necesario especificar dirección. Con el mnemónico basta.

PSHA :Introducir el acumulador A en la pila. Se introduce A en la dirección especificada por SP. SP se decrementa en 1

PSHB :Introducir el acumulador B en la pila

PSHX :Introducir el registro de índice X en la pila. Se introduce X en la pila. SP se decrementa en 2 unidades puesto que X es de 16 bits.

PSHY :Introducir el registro de índice Y en la pila.

PULA :Sacar A de la pila. Se decrementa SP en una unidad y se introduce en A el dato contenido en la posición apuntada por SP.

PULB :Sacar B de la pila.

PULX :Sacar X de la pila. SP se incrementa en 2 unidades ya que X es de 16 bits.
PULY :Sacar Y de la pila.
TAB :Introducir el valor de A en B

TBA :Introducir el valor de B en A
TSX :Introducir el valor de SP en X
TSY :Introducir el valor de SP en Y
TXS :Introducir el valor de X en SP
TYS :Introducir el valor de Y en SP
XGDX :Intercambiar el registro D con el X
XGDY :Intercambiar el registro D con el Y

3.4.2. Instrucciones aritméticas

- **SUMAR**

ADDA :Añadir un dato al acumulador A
Ej. ADDA #5 ; Sumar 5 al acumulador.
ADDA $C000 ;Sumar el contenido de la dir. $C000 al acumulador
ADDB :Añadir un dato al acumulador B
ADDD :Añadir un dato al doble acumulador D
ADCA :Añadir al acumulador A un dato y el contenido del acarreo.
ADCB :Añadir al acumulador B un dato y el contenido del acarreo.
ABA :Sumar el acumulador A y B y poner el resultado en A
ABX :Sumar B y X y poner resultado en X
ABY :Sumar B y Y y poner resultado en Y
INCA :Incrementar el acumulador A
INCB :Incrementar el acumulador B

INC :Incrementar el contenido de una dir. de memoria.
Ej. INC 2,Y ; Sumar una unidad al byte que se encuentra en la dir. Y+2.

INX :Incrementar registro X
INY :Incrementar registro Y
INCS :Incrementar puntero de pila SP.

- **RESTAR**

SUBA :Restar un dato al acumulador A.
Ej. SUBA #$2C ; Restar $2C al acumulador.

SUBB :Restar un dato al acumulador B.
SUBD :Restar un dato al doble acumulador D.
SBCA :Restar al acumulador A un dato y el contenido del acarreo.
SBCB :Restar al acumulador B un dato y el contenido del acarreo.
DECA :Decrementar acumulador A
DECB :Decrementar acumulador B
DEC :Decrementar byte de una dir. de memoria.
DEX :Decrementar registro X.
DEY :Decrementar registro Y.
DES :Decrementar puntero de pila SP.

- **COMPARACIONES**

CMPA :Comparar acumulador A con un dato. Se activan los bits correspondientes del registro de status. Los bits que se activan son el Z(Cero) y el N (negativo).
Ej. CMPA #10 ; Comparar Acumulador A con 10.

CMPB :Comparar Acumulador B con un dato.
CPD :Comparar doble acumulador D con un dato.
CPX :Comparar registro X con un dato.
CPY :Comparar registro Y con un dato.
CBA :Comparar A con B.

- **COMPLEMENTO A DOS**

  Estas instrucciones permiten obtener números negativos en formato de complemento a dos.

  NEG :Complementar a dos un byte de la memoria.
  NEGA :Complementar a dos el acumulador A
  NEGB :Complementar a dos el acumulador B

- **MULTIPLICACIONES Y DIVISIONES**

  MUL :Se multiplican A y B y el resultado se introduce en el doble acumulador D
  IDIV :Se divide D entre X y el resultado se guarda en X. El resto se guarda en D.

3.4.3. Operaciones lógicas y de manipulación de bits

ANDA :Se realiza una operación AND lógica entre el registro A y la memoria. El resultado se almacena en el acumulador A.

ANDB :Idem pero con el acumulador B.

ORAA :Se realiza una operación OR lógica entre el acumulador A y la memoria. El resultado se almacena en el acumulador A.

ORAB :Idem pero con el acumulador B.

EORA :Realizar un or−exclusivo (XOR) entre el registro A y memoria y almacenar resultado en acumulador A.

EORB :Idem pero con el acumulador B.
COMA :Se realiza el complemente a uno de A.
COMB :Se realiza el complemento a uno de B.

BITA :Esta instrucción sirve para comprobar si determinados bits de una posición de memoria están activados o no. Se realiza un AND lógico entre el acumulador y la posición de memoria pero no se altera ninguna de las dos. El resultado queda reflejado en el bit Z del registro CCR.

Ejemplo: Queremos comprobar si los bits 0 y 1 del puerto A están ambos activados:

```
LDAA #$03      ; Meter el valor $03 (00000011 en binario) en A
BITA PORTA     ; Comprobar bits 0 y 1.
BEQ subrutina  ; Saltar si ambos bits son cero.
```

BITB :Idem que BITA pero con el acumulador B.

BCLR :Poner a cero los bits especificados de una posición de memoria. La sintaxis es: BCLR operando máscara. El operando es una posición de memoria a la que se puede acceder mediante cualquiera de los modos de direccionamiento. Máscara es un byte, cuyos bits a uno se corresponden con los bits del operando que se quieren poner a cero. Por ejemplo, se quiere poner a cero los bits 0 y 1 del puerto A: **BCLR PORTA,X $03.**

¡ Sólo está permitido el direccionamiento indexado!

BSET :Lo mismo que BCLR pero los bits en vez de ponerse a cero se ponen a 1.

BRCLR: Esta instrucción es muy útil y un poco diferente del resto porque tiene 3 parámetros. Se bifurca a la dirección especificada si unos bits determinados están a cero. La sintaxis es: **BRCLR** *operando máscara dirección*. Se realiza un AND lógico entre el operando y la máscara y se bifurca si la operación da como resultado cero, es decir, si todos los bits indicados estaban a cero.

Un ejemplo muy típico es un bucle de espera hasta que se active un bit de una posición de memoria:

```
        LDX    #$1000
wait    BRCLR 0,X $80 wait
```

El bucle se repite mientras el bit 7 del puerto A sea distinto de cero. En cuanto se ponga a cero se sale del bucle. **Esta instrucción sólo permite direccionamiento indexado!!**

BRSET :Igual que BRCLR pero se salta cuando los bits indicados se ponen a 1.

3.3.4. Desplazamientos y rotaciones

•Desplazamientos aritméticos:

ASL :Desplazamiento aritmético a la izquierda de un operando en memoria.
ASLA :Desplazamiento aritmético a la izquierda del acumulador A.
ASLB :Idem pero con el acumulador B.
ASLD :Idem pero con el doble acumulador D.
ASR :Desplazamiento aritmético a la derecha de un operando en memoria
ASRA :Desplazamiento aritmético a la derecha del acumulador A.
ASRB :Idem con el acumulador B

•Desplazamientos lógicos:

LSR :Desplazamiento lógico a la derecha de un operando en memoria
LSRA :Desplazamiento lógico a la derecha del acumulador A
LSRB :Idem con el acumulador B.
LSRD :Idem con el acumulador D.

•Rotaciones:

ROL :Rotación a la izquierda de un operando en memoria
ROLA :Rotación a la izquierda del acumulador A
ROLB :Rotación a la izquierda del acumulador B
ROR :Rotación a la derecha de un operando en memoria
RORA :Rotación a la derecha del acumulador A
RORB :Rotación a la izquierda del acumulador B

3.4.5. Bifurcaciones y saltos

- **Bifurcaciones** :Las bifurcaciones (instrucciones BRANCH) se diferencian de los saltos en que se realizan mediante direccionamiento relativo por lo que sólo se pueden utilizar para saltar 128 bytes hacia atrás o 127 bytes adelante. Las bifurcaciones condicionales bifurcan a la dirección especificada cuando se da una determinada condición en el registro de estado CCR.

BCC :Bifurcación si acarreo está a cero
BCS :Bifurcación si acarreo está a uno
BEQ :Bifurcar si el resultado a sido cero (Z=1)
BGE :Bifurcar si mayor o igual (Signo)
BGT :Bifurcar si mayor que (Signo)
BHI :Bifurcar si mayor que (Sin signo)
BHS :Bifurcar si mayor o igual (Sin signo)
BLE :Bifurcar si menor o igual (Signo)
BLO :Bifurcar si menor (Sin Signo)
BLS :Bifurcar si menor o igual (Sin signo)
BLT :Bifurcar si menor (Signo)
BMI :Bifurcar si negativo (N=1)
BNE :Bifurcar si no igual (Z=0)
BPL :Bifurcar si positivo (N=0)

BVC :Bifurcar si overflow está a cero (V=0)
BVS :Bifurcar si overflow está a uno (V=1)
BRA :Bifurcar (Salto incondicional)
BSR :Llamar a una subrutina (incondicional)

- **Saltos** Los saltos se pueden realizar a cualquier dirección de memoria.

JMP :Salto incondicional
JSR :Salto incondicional a una subrutina

3.4.6. Instrucciones de modificación del CCR

Estas instrucciones alteran los bits del registro de estado CCR.
CLC :Poner a cero el bit de acarreo
SEC :Poner el bit de acarreo a uno
CLI :Poner el bit de interrupciones a cero. Las interrupciones se permiten.
SEI :Poner el bit de interrupciones a uno. Las interrupciones se inhiben.
CLV :Poner el bit de overflow a cero
SEV :Poner el bit de overflow a uno
TAP :Mover el Acumulador A al registro CCR
TPA :Mover el CCR al acumulador A

3.4.7. Otras instrucciones

RTS :Retornar de una subrutina
RTI :Retornar de una interrupción
SWI :Interrupción Software
WAI :Esperar hasta que ocurra una interrupción
NOP :No operación. No hace nada salvo consumir un ciclo de reloj.
STOP :Parar el reloj.

3.5. INTERRUPCIONES

Las interrupciones son señales generadas interna o externamente al microcontrolador que provocan que la CPU deje de ejecutar el programa en curso y ejecute una rutina específica para atender a la interrupción. Una vez ejecutada la rutina de servicio de la interrupción, la CPU continúa con el programa que estaba ejecutando antes de producirse la interrupción.

Las direcciones de las rutinas de servicio de las interrupciones se encuentran en una tabla en memoria, denominada tabla de vectores de interrupción. Existen dos tablas de vectores de interrupción según el modo de funcionamiento del MCU. Si el modo de funcionamiento es el especial de arranque, la tabla se encuentra en memoria ROM en las direcciones $BFD6−$BFFE. Si funciona en modo normal o extendido, la tabla se encuentra en las direcciones $FFC0−$FFFE. En la tabla hay 21 vectores, y cada vector contiene una dirección de memoria que ocupa 2 bytes, por tanto las tablas de vectores ocupan 21*2 = 42 bytes. En la figura 13 se muestra la tabla de vectores de interrupción.

3.5.1. Interrupción de RESET:

La interrupción de RESET es una interrupción especial. Ocurre cada vez que se recibe un nivel bajo en el pin de RESET del MCU (cada vez que se pulsa el botón de reset). Al producirse esta interrupción, la CPU toma de la tabla de vectores de interrupción la dirección de la rutina que tiene que empezar a ejecutar. Si el MCU funciona en modo especial de arranque, se empezará a ejecutar un programa en ROM, llamado BOOTSTRAP, que permite cargar un programa cualquiera procedente del exterior en la memoria RAM. Si el MCU funciona en modo normal o expandido, se ejecuta la rutina indicada por su correspondiente vector de interrupción.

Nada más producirse el reset, las interrupciones se inhiben y el contenido de los registros queda indeterminado. La memoria RAM se mapea en las direcciones $0000−$00FF del mapa de memoria, y los registros de control se sitúan en las direcciones $1000−$103F. Las memorias ROM y EEPROM quedan configuradas de la misma manera que lo estaban antes del reset puesto que su registro de configuración no se borra al eliminar la alimentación (es un registro tipo EEPROM). La mayoría de los periféricos del MCU (SCI, SPI, Puertos, temporizadores...) sufren algún cambio en sus registros.

3.5.2. Tipos de interrupciones:

Existen 3 tipos de interrupciones: las interrupciones enmascarables, las no enmascarables y las interrupciones software.

- **Las interrupciones enmascarables**, como su nombre indica, se pueden enmascarar, es decir, inhibir sin más que actuar sobre el bit I del CCR. (I=0−−> se permiten interrupciones; I=1−−> Interrupciones inhibidas). Con la instrucción CLI, el bit I del CCR se pone a cero y se permiten las interrupciones. Con SEI, I se pone a 1 y se inhiben las interrupciones.

- **Las interrupciones no enmascarables** no se pueden inhibir. Son las interrupciones correspondientes a un fallo en el MCU, instrucción ilegal, RESET y la interrupción externa XIRQ.

- **Las interrupciones software** son las producidas por el propio programador en unos instantes totalmente conocidos. Sólo existe una interrupción software que se produce con la instrucción SWI. (equivalente al TRAP del 68000). La CPU al leer esta instrucción actúa como si de una interrupción normal se tratase. Estas interrupciones son no enmascarables porque de lo contrario la cpu se colgaría:

```
SEI          ; Inhibir interrupciones
SWI          ; ¡¡ Nunca se ejecutaría !!
```

Si se pudiesen enmascarar, la CPU nunca ejecutaría la instrucción SWI y quedaría colgada hasta que se "resetease".

También es posible clasificar las interrupciones en internas y externas.

•**Las interrupciones internas** son las producidas por circuitos o periféricos integrados dentro del propio microcontrolador.

•**Las interrupciones externas** son las producidas por circuitos o periféricos externos al MCU. Existen 2 entradas de interrupciones externas:

IRQ es una entrada de interrupciones enmascarables.
XIRQ es la entrada de interrupciones no enmascarables.
IC1, IC2, IC3, PAI y STRA pueden considerarse como interrupciones externas especiales.

3.5.3. Prioridad de las interrupciones:

No todas las interrupciones tienen la misma prioridad. Si se producen dos interrupciones a la vez, primero se ejecuta la de mayor prioridad y después la de menor. Por ejemplo, si se producen a la vez la interrupción de RESET y una procedente del puerto de comunicaciones SERIE (SCI), ¿a qué interrupción se atiende antes?. Lógicamente la de RESET tiene una prioridad mayor y se procedería a la reinicialización del sistema.

Las interrupciones no enmascarables (RESET, XIRQ etc...) tienen la máxima prioridad. Las demás interrupciones tienen una prioridad determinada por el hardware, pero es posible hacer que la prioridad de una determinada interrupción sea máxima. Para ello basta con escribir un cierto valor en los bits 0−3 del registro HPRIO ($103C). **Sólo se puede cambiar la prioridad estando las interrupciones inhibidas**. En el apéndice D se encuentra información relacionada con el registro HPRIO y las prioridades de las interrupciones.

3.5.4. Proceso de interrupción:

La CPU del microcontrolador está ejecutando un programa. De repente aparece una interrupción. Si las interrupciones están permitidas (Bits X e I del CCR) se introducen todos los registros en la pila, se inhiben las interrupciones y se obtiene de la tabla de vectores de interrupción la dirección a la que tiene que bifurcar la CPU. Las interrupciones se inhiben para que no se produzcan anidamientos de interrupciones, es decir, que mientras se está atendiendo a una interrupción, por defecto no se atienda a otra. El usuario por supuesto puede activar las interrupciones en una rutina de servicio de una interrupción, con lo que provocaría un anidamiento de interrupciones (No es recomendable anidar interrupciones).

Si se está ejecutando la rutina de servicio de una interrupción enmascarable y se produce una interrupción no enmascarable, como esta última tiene una prioridad mayor, la CPU procede a atenderla dejando a la anterior "congelada". Cuando termina con la no enmascarable vuelve con la enmascarable.

Todas las rutinas de servicio de interrupción deben acabar con **la instrucción RTI** que indica a la CPU que la interrupción ha sido atendida y que se puede retornar al programa anterior: La CPU toma de la pila todos los registros que previamente había guardado y continúa ejecutando el programa que había sido interrumpido.

En el microcontrolador existen muchos periféricos integrados que pueden producir interrupciones en cualquier momento. Para generar una interrupción activan un bit de un determinado registro del periférico. Este bit indica a la CPU que el periférico en cuestión ha solicitado interrupción. Cuando la CPU pasa a ejecutar la rutina de servicio del periférico, lo primero que debe hacer es DESACTIVAR ESTE BIT. Si no se hace, al retornar de la interrupción con RTI, el bit seguirá activo y la CPU lo interpretará como una nueva interrupción, con lo que se vuelve a ejecutar la rutina de servicio. Así permanecería la CPU en un bucle infinito. En algunos periféricos este bit se desactiva automáticamente al ser ejecutada su rutina de interrupción, pero en otros no, como por ejemplo la unidad de comunicaciones serie asíncronas (SCI). Por tanto, **lo primero que hay que hacer en una rutina de servicio de interrupciones es desactivar el bit de interrupción correspondiente**. Según el periférico de que se trate, este bit se desactivará de una forma u otra. En algunos casos se puede desactivar escribiendo directamente un valor en el registro en que está contenido. En otros casos es necesario ejecutar una serie de instrucciones para que se borre. Por ejemplo, suponiendo que se esta diseñando la rutina de servicio del SCI, ésta rutina se ejecuta cada vez que se recibe un carácter por este puerto. Lo primero que hay que hacer es leer el registro de estado y a continuación el dato recibido. De esta manera el bit de interrupción se desactiva y la rutina puede terminar normalmente con RTI sin que pasen "cosas raras".

3.5.5. Vectores de interrupción

En la figura 13 se muestran las tablas de vectores de interrupción para los modos single chip, expanded y bootstrap. En los modos single chip y expanded, los vectores de interrupción están situados a partir de la dirección

FFD6. En el modo bootstrap los vectores se encuentran en memoria ROM a partir de la dirección BFC0. Puesto que estos vectores están en ROM su valor no se puede cambiar. Estos vectores apuntan a diferentes zonas de la RAM. Es allí donde se debe realizar el salto a la subrutina de atención a la interrupción.

En la tabla de la figura 13 puede verse a la izquierda la causa de la interrupción. A la derecha están las direcciones de los vectores de interrupción para los distintos modos de funcionamiento. La columna de la derecha del todo, indica el contenido del vector de interrupción del modo bootstrap, es decir, indica la dirección en RAM a la que se bifurca cuando ocurre una determinada interrupción. Por ejemplo, si se produce la interrupción correspondiente al SCI, la CPU pasa a ejecutar el código que se encuentra a partir de la dirección $00C4. Para el caso del modo extendido no se especifica el contenido del vector porque se
puede cambiar por software, según los valores colocados en esas posiciones de memoria.

FUENTE DE INTERRUPCION	**DIRECCION DEL VECTOR**		
	SINGLE CHIP/ EXPANDED	**BOOTSTRAP**	
		Dirección ROM	**Saltar a la direccion**
Interrupción del SCI	FFD6	BFD6	00C4
Interrupción del SPI	FFD8	BFD8	00C7
Acumulador de pulsos	FFDA	BFDA	00CA
Overflow en acumulador de pulsos	FFDC	BFDC	00CD
Overflow del temporizador	FFDE	BFDE	00D0
Comparador de salida 5	FFE0	BFE0	00D3
Comparador de salida 4	FFE2	BFE2	00D6
Comparador de salida 3	FFE4	BFE4	00D9
Comparador de salida 2	FFE6	BFE6	00DC
Comparador de salida 1	FFE8	BFE8	00DF
Capturador de entrada 3	FFEA	BFEA	00E2
Capturador de entrada 2	FFEC	BFEC	00E5
Capturador de entrada 1	FFEE	BFEE	00E8
Interrupción de timpo real	FFF0	BFF0	00EB
Puerto paralelo/IRQ	FFF2	BFF2	00EE
XIRQ	FFF4	BFF4	00F1
Interrupción software	FFF6	BFF6	00F4
Código de instrucción ilegal	FFF8	BFF8	00F7
Fallo del sistema	FFFA	BFFA	00FA
Fallo en el monitor del reloj	FFFC	BFFC	00FD
Interrupción de RESET	FFFE	BFFE	BF40

Figura 13: Tabla con los vectores de interrupción para los diferentes modos.

4. RECURSOS DEL 68HC11 Y SU PROGRAMACIÓN

4.1. MAPA DE MEMORIA

El MCU direcciona 64Kbytes de memoria, parte de esta memoria se encuentra dentro del MCU y el resto se puede implementar mediante chips de memoria externos al MCU. Según el modelo de microcontrolador empleado (A8, A0, A2, E9...) se dispondrá de más o menos recursos de memoria. La memoria RAM se sitúa por defecto a partir de la dirección $0000 hasta la $00FF (256 bytes de RAM).

De la dirección $1000 hasta la $103F se encuentran situados los registros de control del MCU. Estos registros son células de memoria RAM o EEPROM (no volátiles). Las células EEPROM sólo se pueden escribir bajo unas circunstancias especiales, y siempre que el MCU esté en modo especial.

Entre las direcciones $B600−$B7FF se sitúan 512 bytes de memoria EEPROM que no están disponibles en el modelo A0. En los modos especiales se activa una memoria ROM en las direcciones $BF40−$BFFF que contiene el programa BOOTSTRAP usado para cargar programas externos y una tabla con los vectores de interrupción.

Entre las direcciones $E000−$FFFF se encuentra la memoria ROM, que salvo ocasiones muy especiales siempre está desconectada. En esta ROM el fabricante puede grabar cualquier aplicación que el usuario quiera (y que pueda pagar!!).

Desde la dirección $FFC0 hasta el final se encuentra la tabla de vectores de interrupción del modo normal.

En la figura 14 se muestra el **mapa de memoria** por defecto. Configurando algunos registros de control adecuadamente se puede "remapear" el sistema para ajustarlo a las necesidades de la aplicación. Por ejemplo, la memoria RAM se puede situar en cualquier otra parte (con alguna restricción) dentro de los 64Kbytes del mapa de memoria. Lo mismo ocurre con los registros de control.

El registro que permite cambiar las direcciones de la memoria y de los registros de control se denomina INIT y se encuentra en la dirección $103D.

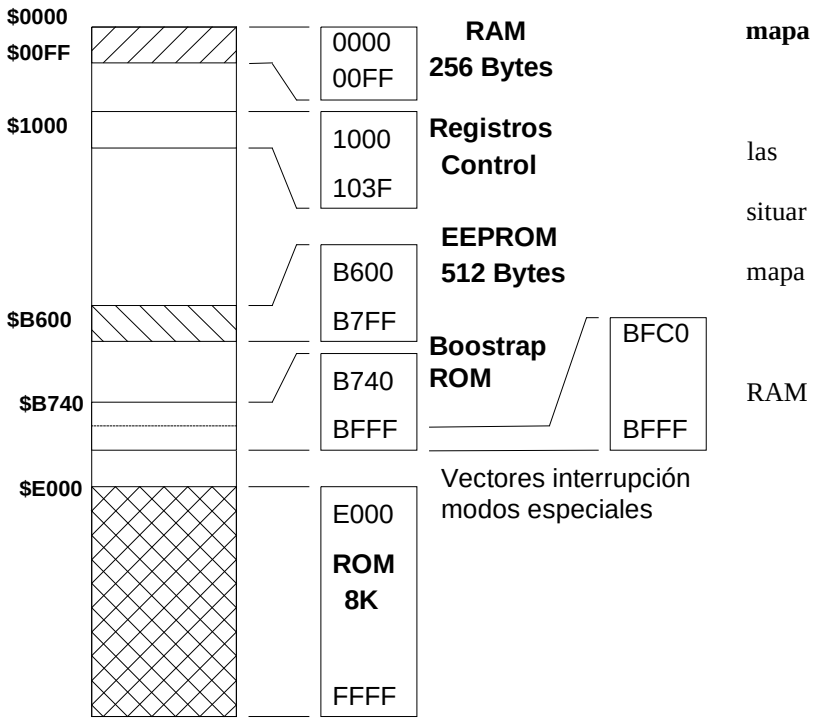


Figura 14: Mapa de memoria en el modo BOOTSTRAP

Registro INIT:

7	6	5	4	3	2	1	0
RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0

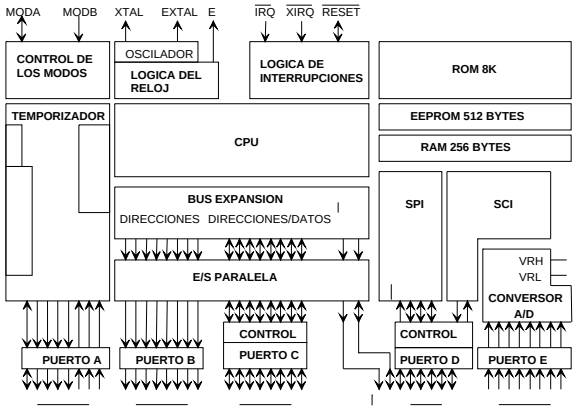
Las direcciones de memoria están formadas por 2 bytes, es decir, direcciones de 16 bits. Los bits del 7 al 4 del registro INIT permiten establecer los 4 bits de mayor peso de la dirección en la que se va a mapear la memoria RAM. Los 12 bits restantes no se pueden fijar. Los bits del 3 al 0 hacen lo mismo pero con la dirección de comienzo de los registros de control. El mapa de memoria queda dividido en 16 páginas de 4KBytes cada página (16*4KB = 64Kb). Tanto la RAM interna como los registros de control se pueden situar en cualquiera de estas 16 páginas:

$0000 (Situación de la RAM por defecto),
$1000 (Situación de los registros de control por defecto),

$2000, $3000, $4000, $5000 ... ... $D000, $E000 y $F000.

Debido a la posibilidad de remapear, pueden surgir conflictos al encontrarse 2 recursos internos o externos en las mismas posiciones de memoria. El MCU resuelve esto adjudicando unas prioridades. Si se mapean la RAM, los registros de control y un dispositivo externo en la zona de la ROM (A partir de la dirección $E000 en adelante) el MCU asigna la máxima prioridad a los registros de control, después a la RAM, después la ROM y finalmente el recurso exterior. En este caso, si se intentase acceder a las direcciones $E000−$E03F se seleccionarían los registros de control. A partir de la $E03F ya no existirían los registros de control y se activaría la memoria RAM. A partir de la dirección en la que la RAM se acabase (sólo son 256 bytes de RAM) se accedería a la ROM. Al dispositivo exterior nunca se podría acceder a no ser que se mapease fuera de la zona destinada a la ROM o que se desactivase la ROM.

4.2. PUERTOS DE ENTRADA/SALIDA



Existen 5 puertos de 8 bits disponibles: Puerto A, B, C, D y E. Además de comportarse como puertos normales, sus pines están compartidos con alguno de los recursos internos.

4.2.1. PUERTO A

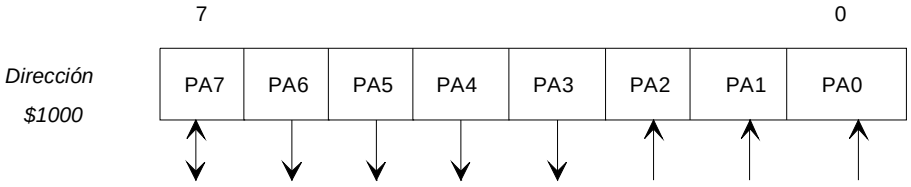


Figura 15: Bits del puerto A

El puerto A dispone de 3 pines de entrada, 4 pines de salida y uno configurable como entrada o como salida. Se encuentra mapeado en memoria en la dirección $1000. Los pines del puerto A están compartidos por otros recursos: comparadores, acumulador de pulsos y capturadores. En la figura 15 se muestran todas las funciones que están asignadas a cada pin del puerto A.

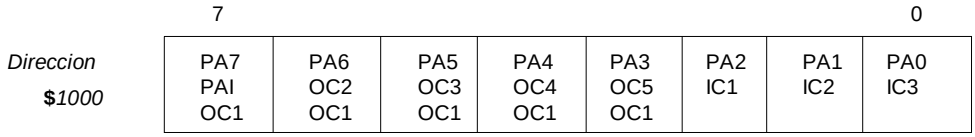


Figura 16: Recursos que utilizan cada bit del puerto A

PA0−PA2	Bits de entrada
PA3−PA6	Bits de salida
PA7	Bidireccional
OCx	Salidas de los comparadores
ICx	Capturadores de entrada
PAI	Acumulador de oulsos

Por defecto los recursos internos asociados a los pines del puerto A están "desconectados". El puerto A se comporta como un puerto normal en el que si se escribe un valor en la dirección $1000 se reflejará en los correspondientes pines de salida y si se lee un valor, se hará de los pines de entrada.

El pin 7 se puede configurar tanto para entrada como para salida cambiando el bit 7 del registro **PACTL** ($1026). Un cero en este bit indica entrada y un uno salida. Por defecto está configurado como entrada.

A continuación se proponen una serie de programas que usan el puerto A, en concreto encienden y apagan el LED situado en el bit PA6. Para ello se debe disponer de la tarjeta entrenadora CT6811, en caso contrario se deberá conectar un LED al bit PA6 para que los programas tengan efecto exterior.

−EJEMPLO 1: Activar el bit PA6 del puerto A

Este es el ejemplo más simple posible. Simplemente se activa el bit 6 del puerto A. Si se prueba en la tarjeta CT6811 se podrá ver cómo se enciende el led conectado a este bit.

```
; +-------------------------------------------------------------------------+
; | LEDON.ASM (C)GRUPO J&J. Febrero 1997                                    |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Simplemente se enciende el led de la tarjeta CT6811.                 |
; +-------------------------------------------------------------------------+

        ORG $0000

        LDAA #$40       ; Almacenar en el acumulador A el valor $40
        STAA $1000      ; Enviar el valor $40 a la posición de memoria $1000

inf     BRA inf         ; Bucle infinito

        END
```

−EJEMPLO 2: Activar intermitentemente el bit 6 del puerto A

El siguiente ejemplo hace activarse y desactivarse el bit 6 del puerto A. Si se ejecuta en la tarjeta CT6811 se puede ver como parpadea el led.

```
; +-------------------------------------------------------------------------+
; | LEDP.ASM (C) GRUPO J&J. Febrero 1997                                    |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Simplemente se enciende y se apaga el led de la tarjeta CT6811.      |
; +-------------------------------------------------------------------------+

        ORG $0000

comienzo
        LDAA $1000
        EORA #$40         ; Cambiar de estado el bit PA6
        STAA $1000

        LDY #$FFFF        ; Realizar una pausa
dec     DEY
        CPY #0
        BNE dec

        BRA comienzo      ; Repetir el proceso

        END
```

−EJEMPLO 3: Utilización del bit 7 del puerto A como salida

El bit 7 del puerto A es un bit que puede ser configurado para entrada o para salida. Por defecto está configurado como entrada. En este ejemplo se configura para salida y se pone a ’1’, de tal forma que si se conecta un led se puede ver cómo se enciende.

```
; +-------------------------------------------------------------------------+
; | PA7.ASM (C) GRUPO J&J. Febrero 1997                                     |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Se configura el bit 7 del puerto A para salida y se activa con       |
; |    un ’1’.                                                              |
; +-------------------------------------------------------------------------+
```

```
PACTL   EQU  $26

        ORG $0000

        LDX #$1000
        LDAA #$80       ; Poner bit 7 del registro PACTL a '1' para
        STAA PACTL,X    ; configurar el bit 7 del puerto A como salida.

        LDAA #$80
        STAA $1000      ; Activar el bit 7 del puerto A

inf     BRA inf         ; Bucle infinito

        END
```

−EJEMPLO 4:

En este ejemplo se refleja el estado del bit PA0 (bit de entrada) sobre el bit PA6 de salida. Si la entrada PA0 se pone a '1', el bit de salida PA6 se pondrá a '1'. Lo mismo con el estado '0'.

```
; +-------------------------------------------------------------------------+
; | PA0.ASM  (C) GRUPO J&J. Febrero 1997                                    |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |  Programa ejemplo de la lectura del bit PA0. Este estado se reflejará   |
; | sobre el bit de salida PA6.                                             |
; +-------------------------------------------------------------------------+

         ORG $0000
comienzo
        LDAA $1000      ; Leer puerto A
        ANDA #$01       ; Dejar en acumulador A el estado del bit PA0
        CMPA #$00       ; ¿PA0 está a 0?
        BEQ activa_pa6  ; Si--> Desactiva PA6
        LDAA #$40       ; NO--> Activa PA6.
        STAA $1000      ;
        BRA comienzo

activa_pa6
        LDAA #$00
        STAA $1000
        BRA comienzo

        END
```

4.2.3. PUERTO B

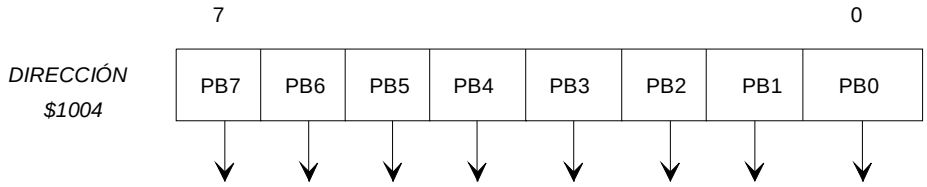


Figura 17: El puerto B

Sus 8 bits son de salida. En el modo no expandido del MCU se comporta como un puerto de salida (PBx). En el modo expandido se utiliza para mandar el byte alto del bus de direcciones (Ax). Su dirección es la $1004. En la figura 18 se muestran todos los usos de los bits del puerto B.

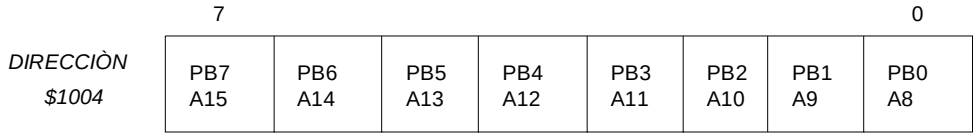


Figura 68: Todos los usos de los bits del puerto B

−EJEMPLO 1: Activación del bit PB0.

```
; +-------------------------------------------------------------------------+
; | PB0.ASM   (C) GRUPO J&J. Marzo 1997                                     |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
```

```
; ¦    Se activa el bit PB0 del puerto B.                                    ¦
; +-----------------------------------------------------------------------+

PORTB   EQU $1004       ; Dirección del puerto B

        ORG $0000


        LDAA #$01
        STAA PORTB      ; Activar el bit PB0 del puerto B

inf     BRA inf         ; Bucle infinito

        END
```

−EJEMPLO 2: Activación rotatoria de los bits del puerto B.

En este ejemplo se activa cada vez un bit del puerto B. Se comienzo con el bit PB0, después el PB1... hasta llegar al PB7. Una vez activado este bit se vuelve a la situación inicial.

```
; +-----------------------------------------------------------------------+
; ¦ PUERTOB.ASM   (C) GRUPO J&J. Marzo 1997                               ¦
; ¦-----------------------------------------------------------------------¦
; ¦  Programa ejemplo para ser ejecutado en la tarjeta CT6811.            ¦
; ¦  Este programa se debe cargar en la RAM interna del 6811.             ¦
; ¦                                                                       ¦
; ¦    Activación rotativa de los bits del puerto B                       ¦
; +-----------------------------------------------------------------------+

PORTB   EQU $1004       ; Dirección del puerto B

        ORG $0000

comienzo
        LDAA #$01
siguiente
        STAA PORTB      ; Activar el bit del puerto B que toca
        BSR pausa       ; Pausa
        ROLA            ; Rotar acumulador A hacia la izquierda
        CMPA #0         ; ¿Se ha hecho la ultima rotación ?
        BNE siguiente   ; No --> continuar
        BRA comienzo    ; Si --> Volver a comenzar

pausa   LDY #$FFFF
wait    DEY
        CPY #0
        BNE wait
        RTS
        END
```

4.2.5. PUERTO C

Es un puerto de entrada/salida. En el modo no expandido sus 8 bits pueden actuar como entradas o salidas independientes, según cómo se configuren los bits en el registro **DDRC** ($1007). Un cero en un bit del registro DDRC configura el pin correspondiente para entrada. Un uno lo hace para salida. La dirección del puerto C es $1003.

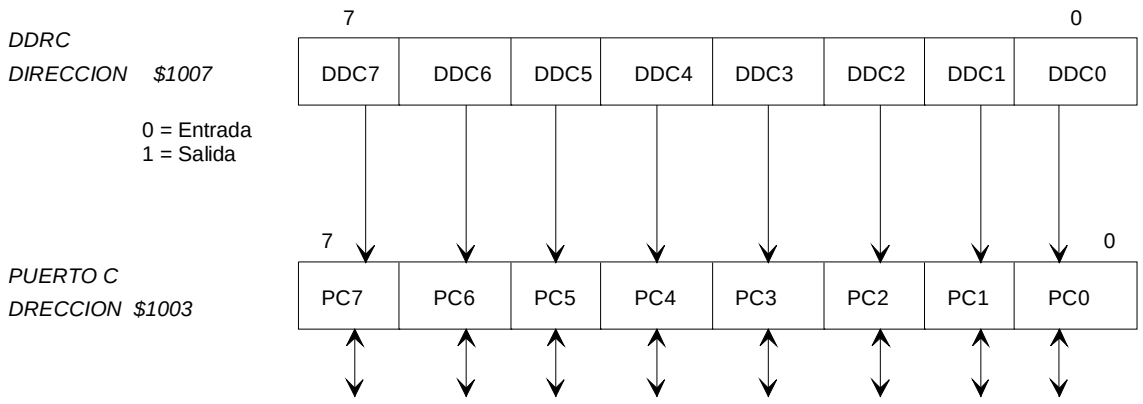


Figura 19: Puerto C y registros de configuración del puerto C (DDRC)

Si el microcontrolador está funcionando en el modo expandido, el puerto C (PCx) actúa como parte baja del bus de

direcciones (Ax) multiplexada con el bus de datos (Dx). En la figura 20 se muestran todos los usos de los pines del puerto C.

	7							0
DIRECCION *$1003*	PC7 A7 D7	PC6 A6 D6	PC5 A5 D5	PC4 A4 D4	PC3 A3 D3	PC2 A2 D2	PC1 A1 D1	PC0 A0 D0

Figura 20: Usos de los bits del puerto C

−EJEMPLO 1:

En este ejemplo se configuran los 4 bits de menor peso del puerto C para entrada y los 4 bits de mayor peso para salidas. El estado de las entradas se refleja sobre los bits de salida.

```
; +-------------------------------------------------------------------------+
; | PUERTOC.ASM   (C) GRUPO J&J. Marzo 1997                                 |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Los 4 bits de menor peso del puerto C se configuran para entrada     |
; |    y los 4 bits de mayor peso del puerto C se configura para salida.    |
; |    El estado de la entrada se refleja sobre los bits de salida.         |
; +-------------------------------------------------------------------------+

PORTC   EQU $03         ; Dirección del puerto C
DDRC    EQU $07         ; Configuración del puerto C

        ORG $0000

        LDX #$1000
        LDAA #$F0       ; Configurar puerto c:
        STAA DDRC,X     ; Bits 0,1,2 y 3 para entrada. Resto salidas

repite
        LDAA PORTC,X    ; Leer puerto C
        ANDA #$0F       ; Quedarse con los bits de entrada
        ROLA
        ROLA
        ROLA
        ROLA
        STAA PORTC,X    ; Escribir bits de entrada sobre los bits de salida
        BRA repite

        END
```

4.2.7. PUERTO D

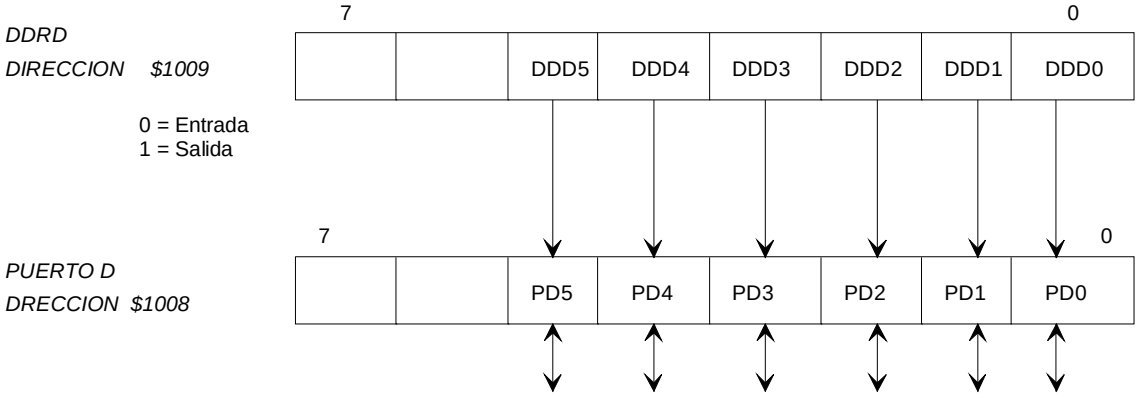


Figura 21: Puerto D y su configuración

Igual que el puerto C, es un puerto de entrada/salida en el que se pueden configurar sus bits independientemente para entrada o salida. El registro de configuración **DDRD** se encuentra en la dirección $1009. Unos se corresponden con salidas y ceros con entradas. El puerto está mapeado en la dirección $1008. Este puerto es de sólo 6 bits. Está compartido con el SPI y el SCI.

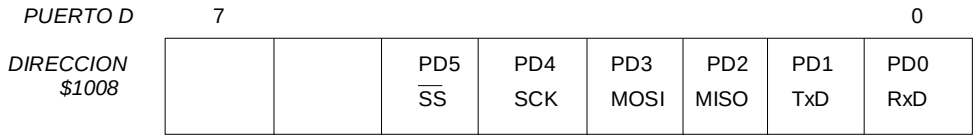


Figura 22: Otros usos de los bits del puerto D

Los bits 0 y 1 están compartidos con el puerto serie. Por ello, para poder ser utilizados es preciso que el SCI esté desactivado. Los bits 2,3,4 y 5 están compartidos por el SPI. Para que funcionen es necesario asegurarse que el SPI está desactivado.

Una de las características del puerto D es que sus salidas pueden ser normales o de **colector abierto.** Esto se configura con el bit 5 del registro **SPCR** ($1028). Si este bit está a 0, las salidas serán normales. Si está a 1, las salidas estarán en colector abierto. Por defecto estan en colector abierto.

En la tarjeta CT6811 se debe seleccionar el modo con salidas en colector abierto puesto que existen resistencias de pull−up colocadas en algunas salidas. No se deben configurar como salidas normales.

−EJEMPLO 1:

Lectura del estado del bit PD2 y escritura sobre el bit PD3. El estado del bit de entrada PD2 se refleja sobre el estado del bit de salida PD3.

```
; +-------------------------------------------------------------------------+
; | PUERTOD.ASM   (C) GRUPO J&J. Marzo 1997                                 |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Se configura el bit 2 del puerto D como entrada y el bit 3 como      |
; |    salida. El estado del bit de entrada se refleja sobre el bit de      |
; |    salida.                                                              |
; +-------------------------------------------------------------------------+

PORTD   EQU $08         ; Dirección del puerto D
DDRD    EQU $09         ; Configuración del puerto D

        ORG $0000

        LDX #$1000

        LDAA #$08       ; Configurar puerto D:
        STAA DDRD,X     ; Bit 3 para salida. Resto entradas.
repite
        LDAA PORTD,X    ; Leer puerto D
        ANDA #$04       ; Quedarse con el PD2
        ROLA
        STAA PORTD,X    ; Escribir estado del bit 2 sobre el bit 3
        BRA repite

        END
```

4.2.9. PUERTO E

Es un puerto de 8 bits de entrada (PEx). Está situado en la dirección $100A. Comparte pines con los 8 canales del conversos A/D (ANx). Para poder utilizar el puerto E como un puerto normal es preciso que el conversor A/D interno del 68HC11 esté desconectado. Por defecto lo está.

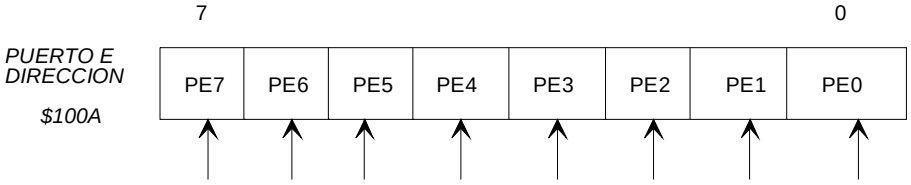


Figura 23: PUERTO E

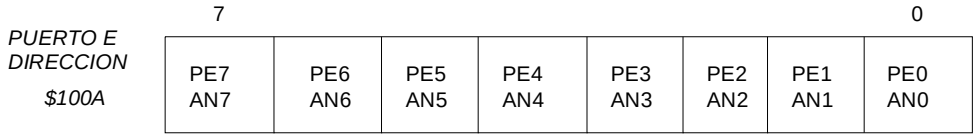


Figura 24: Otros usos de los bits del puerto E

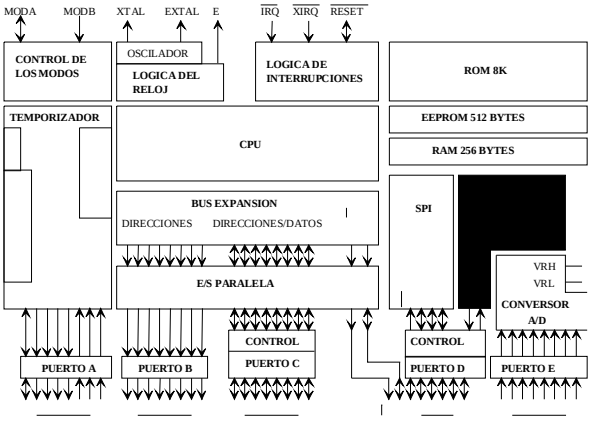
−EJEMPLO 1:

Reflejo del estado del bit PE0 sobre el bit PA6. En la tarjeta CT6811 el bit PA6 está conectado a un led. Por ello, cuando se reciba un ’1’ por el bit 0 del puerto E se encenderá el LED. Cuando se reciba un ’0’ se apagará el LED.

```
; +-------------------------------------------------------------------------+
; | PUERTOE.ASM   (C) GRUPO J&J. Marzo 1997                                 |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    El estado del bit 0 del puerto E se refleja sobre el led de la       |
; |    tarjeta CT6811.                                                      |
; +-------------------------------------------------------------------------+

PORTA   EQU $00         ; Dirección del puerto A
PORTE   EQU $0A         ; Dirección del puerto E

        ORG $0000

        LDX #$1000
repite
        LDAA PORTE,X    ; Leer puerto D
        ANDA #$01       ; Quedarse con el PE0
        CMPA #0         ; ¿Es ’0’?
        BEQ apaga_led   ; Si--> apagar el led
        LDAA #$40       ; No--> encender el led
        STAA PORTA,X
        BRA repite

apaga_led
        CLRA
        STAA PORTA,X
        BRA repite

        END
```

4.3. TRANSMISIONES SERIE

4.3.1. Introducción



El microcontrolador 68HC11 dispone de una unidad de comunicaciones serie (SCI) que permite realizar comunicaciones asíncronas a distintas velocidades y con paquetes de 8 y 9 bits. (8 ó 9 bits de datos).

Los paquetes mandados contienen un bit de comienzo en la cabeza (Bit de start), 8 ó 9 bits de datos y un bit de stop al final.

Para configurar los parámetros de la comunicación, enviar y recibir datos y comprobar el estado de la transmisión, el SCI dispone de 5 registros mapeados en memoria.

Además el SCI dispone del modo especial de funcionamiento WAKE−UP, usado en sistemas multireceptores pero de uso poco frecuente. En este capítulo se describen las funciones básicas de los bits que actúan sobre este modo, pero al lector interesado se le recomienda leerse el capítulo 9 del manual de referencia del 68hc11 de Motorola.

4.3.2. Unidad de transmisión y unidad de recepción

El SCI está formado por una unidad de transmisión y una unidad de recepción que son totalmente independientes, lo que permite que las comunicaciones sean bidireccionales, es decir, se puede transmitir y recibir a la vez (Modo Full−duplex).

La unidad de transmisión está formada por un registro de desplazamiento con carga en paralelo, llamado **registro de transmisión**. Al introducir un valor en este registro, comienza a desplazarse hacia la derecha el contenido del registro, enviando los bits por la línea serie, a una velocidad configurable por el usuario.

De la misma manera la unidad de recepción dispone de otro registro, **registro de recepción** que recibe los bits en serie y los va desplazando hasta obtener un dato en paralelo que puede ser leído.

Tanto el registro de transmisión como el de recepción están mapeados en la misma dirección de memoria. Al escribir en esa dirección de memoria, el dato se cargará en el registro de transmisión. Al efectuar una lectura, el dato se leerá del registro de recepción. Ambos registros comparten la misma dirección física de memoria pero se trata de dos registros diferentes. Puesto que ambos registros comparten dirección física, se les ha asignado un único nombre: **registro de datos (SCDR)**, y está situado en la dirección $102F.

Una vez configurado el SCI adecuadamente, enviar y recibir datos es una tarea muy sencilla: basta con leer o escribir en el registro de datos.

Ejemplo. **Para enviar datos:**

```
LDX  #$1000     ; Utilizar X para acceder a memoria
LDAA #dato      ; Cargar en el acumulador A el dato a enviar (Dato de 8 bits)
STAA $2F,X      ; Enviar dato por el puerto serie.
```

Para recibir datos:

```
LDX  #$1000
LDAA $2F,X      ; Cargar en acumulador A el dato recibido.
```

No obstante, no hay que dejarse engañar por la aparente simplicidad. Por ejemplo, el siguiente trozo de código no funcionará correctamente:

```
LDX  #$1000
LDAA #dato1     ; Cargar dato1 en el acumulador A
STAA $2F,X      ; Mandar dato1
LDAA #dato2     ; Cargar dato2 en el acumulador A
STAA $2F,X      ; Mandar dato2
```

El problema está en que el programa se ejecuta más rápidamente que lo que tarda el SCI en enviar el dato. El dato1 se empieza a mandar. El MCU continúa ejecutando las siguientes instrucciones mientras el dato se sigue

enviando. Cuando se escribe dato2 en el registro de datos, todavía no se ha terminado de enviar el dato1, por lo que el primero es "machacado". Debido a esto, existe un bit en un registro del SCI que se pone a 1 cuando el registro de transmisión está vacío e indica que el dato que se había escrito antes ya se ha enviado.

4.3.3. Registros del SCI.

El SCI dispone de 5 registros mapeados en memoria. Estos registros son los siguientes:

REGISTRO	DIRECCIÓN	DESCRIPCIÓN
BAUD	$102B	Registro de velocidad
SCCR1	$102C	Registro de control 1
SCCR2	$102D	Registro de control 2
SCSR	$102E	Registro de estado
SCDR	$102F	Registro de datos

•Registro de velocidad (BAUD $102B)

7	6	5	4	3	2	1	0
TCLR	**0**	**SCP1**	**SCP0**	**RCKB**	**SCR2**	**SCR1**	**SCR0**

El registro de velocidad permite configurar la velocidad (en baudios) de la comunicación. Los bits 4 y 5 (SCP0 y SCP1) determinan la máxima velocidad en baudios. Esta velocidad depende del cristal que se haya conectado al microcontrolador. Lo habitual es colocar un cristal de 8Mhz. Para este cristal se tiene la siguiente tabla:

SCP1	**SCP0**	**Baudios**
0	0	125000
0	1	41667
1	0	31250
1	1	9600

Para otros valores del cristal se tienen velocidades totalmente distintas. Para más información acudir al manual del microcontrolador. **En la tarjeta CT6811 se utiliza un cristal de 8MHZ**

Una vez determinada la velocidad máxima en baudios, con los bits 0,1 y 2 (SCR0, SCR1 y SCR2) se divide la velocidad máxima por un valor:

SCR2	**SCR1**	**SCR0**	**Dividir vel. máx. entre**
0	0	0	1
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

De esta manera, si por ejemplo se selecciona 9600 baudios como velocidad máxima (SCP1=1 ; SCP2=1), al dividir entre 8 se obtiene una velocidad de 1200 baudios.

Para un cristal de 8 Mhz las velocidades que se consiguen son las que se encuentran resumidas en la figura 25 Los bits 3 y 7 (RCKB y TCLR) están reservados para pruebas del fabricante, deben estar a cero siempre.

SCP1	SCP2	SCR2	SCR1	SCR0	BAUDIOS
0	0	0	0	0	125000
0	0	0	0	1	62500
0	0	0	1	0	31000
0	0	0	1	1	15625
0	0	1	0	0	7812’5
0	0	1	0	1	3906
0	0	1	1	0	1953
0	0	1	1	1	977
0	1	0	0	0	41666
0	1	0	0	1	20833
0	1	0	1	0	10417
0	1	0	1	1	5208
0	1	1	0	0	2604
0	1	1	0	1	1302
0	1	1	1	0	651
0	1	1	1	1	326
1	0	0	0	0	31250
1	0	0	0	1	15625
1	0	0	1	0	7812.5
1	0	0	1	1	3906
1	0	1	0	0	1953
1	0	1	0	1	977
1	0	1	1	0	488
1	0	1	1	1	244
1	1	0	0	0	9600
1	1	0	0	1	4800
1	1	0	1	0	2400
1	1	0	1	1	1200
1	1	1	0	0	600
1	1	1	0	1	300
1	1	1	1	0	150
1	1	1	1	1	75

Figura 25: Velocidades de transmisión serie que se pueden alcanzar con un cristal de 8MHZ

Registro de control 1 (SCCR1 $102C)

7	6	5	4	3	2	1	0
R8	**T8**	**0**	**M**	**WAKE**	**0**	**0**	**0**

El bit 4 (M) permite configurar el SCI para utilizar 8 ó 9 bits de datos:
M=1 −−> 9 bits de datos.
M=0 −−> 8 bits de datos.

Lo normal es utilizar 8 bits de datos. Para comunicarse con un PC el bit se deberá poner a cero puesto que el PC no permite más de 8 bits de datos. Por defecto está a cero. Si se quieren transmitir 9 bits de datos, el noveno bit se escribe en el bit 6 del SCCR1 (T8) y los 8 bits menos significativos se escriben en el registro de datos. Análogamente, si se quiere recibir un dato de 9 bits, el bits más significativo (el noveno) se sitúa en el bit 7 del registro SCCR1 (R8) y los 8 bits restante en el registro de datos.

El SCI tiene un modo de funcionamiento especial (modo WAKEUP) para aumentar la eficiencia en sistemas multireceptores. Es el modo en el cual el receptor se queda con las interrupciones inhibidas esperando un evento hardware externo, (asociado a la línea de recepción), que le devuelva al estado activo con interrupciones. El evento externo puede ser de dos tipos y se selecciona con el bit 3 (WAKE). Si WAKE=1 entonces se espera hasta detectar una marca de dirección, si por el contrario WAKE=0 se espera hasta detectar que la línea de recepción esta vacía. El modo WAKE−UP se selecciona mediante software activando el bit 1 del registro de control 2 (RWU).

Los bits 0,1,2 y 5 no se usan y permanecen siempre a cero.

•Registro de control 2 (SCCR2 $102D)

7	6	5	4	3	2	1	0
TIE	**TCIE**	**RIE**	**ILIE**	**TE**	**RE**	**RWU**	**SBK**

Este registro de control 2 es el principal, con él se configura la salida al exterior del circuito de comunicaciones serie. Esta salida se corresponde con los pines emparejados a los bits 0 y 1 del puerto D. Por tanto, esos pines tienen dos funciones: ser utilizados como bits 0 y 1 del puerto D o como señales Tx y Rx para comunicaciones serie. Mediante el bit 3 del registro de control 2 (TE) se activa o desactiva el transmisor del pin correspondiente. Si TE=1 (Transmitter enabled) el transmisor está activo y el bit 1 del puerto D se desactiva. No se puede utilizar el bit PD1 del puerto D. Si TE=0 el transmisor se desconecta y no se mandan datos serie. El pin funciona como bit 1 del puerto D.

Análogamente, con el bit 2 del registro de control 2 (RE) se activa o desactiva el receptor, que comparte pin con el bit 0 del puerto D. Si RE=1 (Receive Enable) el pin funciona para recibir datos serie del exterior. El bit 0 del puerto D deja de funcionar. Si RE=0 se desconecta el receptor y el pin funciona como bit 0 del puerto D.

En resumen, siempre que se quieran transmitir y recibir datos es imprescindible activar los bits RE y TE del registro de control 2. Una vez activados estos bits, cualquier lectura/escritura sobre el puerto D no se reflejará en los pines 0 y 1 del puerto D, como si no existiesen.

Los bits 4,5,6 y 7 (ILIE, RIE, TCIE, TIE) son máscaras que permiten que ciertas interrupciones ocurran o no. TCIE y TIE corresponden a dos interrupciones del transmisor y ILIE y RIE a dos interrupciones del receptor. En el transmisor, cada vez que el registro de transmisión se vacía, es decir, cada vez que se termina de mandar un dato y por tanto se puede mandar un dato nuevo, se genera una interrupción. Esta interrupción se enmascara con el bit TIE. Si TIE=1 la interrupción de transmisión está permitida y para TIE=0 está desactivada. El bit de estado asociado es TDRE, cuando TDRE=1 indica que se ha recibido un dato y si el bit TIE está a 1 entonces se produce la interrupción.. También hay otra interrupción que indica cuando se ha quedado vacía la línea de transmisión después de mandar el dato, esta se puede enmascarar con el bit TCIE. Si TCIE=1 la interrupción queda permitida, si TCIE =0 está desactivada. El bit de estado asociado es TC y el funcionamiento es análogo al anterior.

En el receptor también existe una interrupción que aparece cada vez que se ha recibido un dato, esta se enmascara con el bit RIE del registro de control 2. Con RIE=1 se permite y con RIE=0 no. El bit de estado asociado es RDRF. La otra interrupción se activa cuando la línea de recepción está vacía, el bit ILIE se encarga de enmascarla. Igual que antes, ILIE=1 la permite y ILIE=0 la desactiva. El bit de estado asociado es IDLE.

Cuando se activa el bit 1, (SBK=1), se mandan señales de BREAK indefinidamente hasta que el bit se desactive. Las señales de BREAK se caracterizan porque se envía todo ceros por la línea serie, no sólo son cero los bits de datos, sino que también se hace cero el bit de stop que siempre vale 1.

Si el bit 1 (RWU) es igual a 1 significa que el modo especial WAKE−UP esta activo. En este modo el receptor esta aletargado con las interrupciones inhibidas, esperando una condición hardware que lo despierte. Dicha condición depende del valor que tenga el bit 3 (WAKE) del registro de control 1 (SCCR1). Generalmente es el

programa quien pone este bit a 1 y la CPU quien lo desactiva. Para más información se recomienda leer el Capítulo 9 del manual de referencia del 68hc11 de Motorola.

•Registro de estado (SCSR $102E)

7	6	5	4	3	2	1	0
TDRE	**TC**	**RDRF**	**IDLE**	**OR**	**NF**	**FE**	**0**

El registro de estado es de sólo lectura y permite comprobar el estado del SCI. El bit 7 (TDRE) se pone a 1 cada vez que se ha terminado de enviar un carácter y por tanto el registro de transmisión está vacío y listo para enviar el siguiente carácter. Siempre que se vaya a transmitir un dato hay que asegurarse que este bit está a 1, porque de lo contrario "se machaca" el dato que se está enviando.

El bit 6 (TC) se pone a 1 cada vez que se ha enviado un carácter y la línea de transmisión se ha quedado vacía 'IDLE'. Este bit te ofrece mayores garantías cuando se quiere saber si la transmisión ha terminado completamente.

El bit 5 (RDRF) se pone a 1 cuando se ha recibido un dato nuevo en el registro de recepción. Por tanto, cuando se reciben datos hay que esperar a que este bit se ponga a 1.

El bit 4 (IDLE) se pone a 1 cuando detecta que la línea de recepción se ha quedado vacía (IDLE).
Si el bit RWU del registro de control 2 esta activo entonces este bit estará inhibido.

El bit 3 (OR) se pone a 1 cuando se ha recibido un carácter por el puerto serie y el anterior dato recibido todavía no se ha leído. Cuando ocurre este error el dato que se pierde es el que se acaba de recibir.

El bit 2 (NF=Noise Flag) se activa cuando se ha detectado un error en el dato que se acaba de recibir. La activación de este bit no produce interrupción, será el software quien se preocupe de examinar este bit después de recibir un dato para saber si es válido o no.

El bit 1 (FE) se activa cuando se ha detectado un error en la trama enviada. Al recibir un dato se comprueba que el bit de stop este a nivel alto. Si esto no se cumple se pone este bit (FE) a nivel alto (1).

El bit 0 no se utiliza.

Todos estos bits se ponen a cero automáticamente cuando se lee el registro SCSR y a continuación se lee del registro de datos (SCDR).

•Registro de datos (SCDR $102F)

Este registro tiene una doble función. Si se escribe en él, el dato se manda al registro de transmisión en la unidad de transmisión del SCI para ser enviado con la velocidad y configuración establecida. Al leerlo se obtiene el valor que tiene el registro de recepción interno de la unidad de recepción del SCI.

4.3.4. Ejemplos de programación del SCI

A la hora de programar las rutinas de comunicaciones serie se pueden seguir dos caminos distintos: Espera activa o interrupciones. Al utilizar espera activa todas las interrupciones están inhibidas. Es el propio software el que debe determinar cuándo se puede enviar un dato y cuándo hay un dato recibido listo para ser leído. Esto se realiza "explorando" el registro de estado del SCI. Cuando se detecta que ha ocurrido algún suceso se actúa en consecuencia. Programar mediante espera activa tiene la ventaja de que conceptualmente es muy sencillo y los programas son fáciles de realizar.

Mediante interrupciones se consigue liberar a la CPU de trabajos inútiles. ¿Por qué estar perdiendo tiempo en comprobar el registro de estado del SCI cuando puede ser el propio SCI el que nos avise de que algo ha ocurrido? Por ejemplo, cada vez que llega un dato nuevo se activa una interrupción. La CPU deja de hacer lo que estaba haciendo y pasa a atender la interrupción. Se lee el carácter que ha llegado y se continúa con lo que estaba haciendo. Las interrupciones tienen el inconveniente de que son más complicadas de entender y los programas son un poco más complejos, pero a cambio el programa es más "eficaz".

−EJEMPLO 1: Configuración del SCI.

L

En este ejemplo se activa el emisor y receptor del SCI. Se configura a una velocidad de 9600 baudios y se utilizan 8 bits de datos. Siempre que se quiera configurar el SCI habrá que realizar esas operaciones. El programa principal simplemente envía los caracteres 'A' y 'B' por el SCI.

```
; +-------------------------------------------------------------------------+
; | SCICONF.ASM (C) GRUPO J&J. Febrero 1997                                 |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Se configura el SCI a 9600 baudios y 8 bits de datos. Se envían los  |
; |  caracteres 'A' y 'B' por el puerto serie.                              |
; +-------------------------------------------------------------------------+

; Registros del SCI

BAUD    EQU  $2B
SCCR1   EQU  $2C
SCCR2   EQU  $2D
SCSR    EQU  $2E
SCDR    EQU  $2F


; ------- CONFIGURACIÓN DEL SCI ----------

        LDX #$1000       ; Para acceder a registros del SCI

        LDAA #$30
        STAA BAUD,X     ; Velocidad transmisión: 9600 baudios
        LDAA #$00
        STAA SCCR1,X    ; 8 bits de datos
        LDAA #$0C
        STAA SCCR2,X    ; Inhibir interrupciones SCI.
                        ; Activar transmisor y receptor del SCI

;---- BUCLE PRINCIPAL ----

bucle
        LDAA #'A'
        BSR enviar
        BSR pausa
        LDAA #'B'
        BSR enviar
        BSR pausa
        BRA bucle

pausa
        LDY #$FFFF
wait    DEY
        CPY #0
        BNE wait
        RTS

;+-----------------------------------------------------------+
;| Enviar un carácter por el puerto serie (SCI)              |
;| ENTRADAS: El acumulador A contiene el carácter a enviar   |
;| SALIDAS: Ninguna.                                         |
;+-----------------------------------------------------------+
enviar  BRCLR SCSR,X $80 enviar
        STAA SCDR,X
        RTS

        END
```

Lo importante de este ejemplo es la configuración. No obstante, en el resto de los ejemplos el SCI no se va a configurar puesto que esto ya se hace **al arrancar en modo bootstrap**. Por tanto si se está trabajando con la tarjeta CT6811 tanto en modo entrenador como autónomo, no es preciso realizar la configuración anterior. La velocidad de comunicaciones utilizada en los siguientes ejemplos es la que se configura por defecto en 68HC11 al arrancar en bootstrap. Esta velocidad es de 7680 baudios.

−EJEMPLO 2: Eco por el puerto serie.

En el siguiente ejemplo se realiza un eco por el puerto serie. Todo lo que se reciba por el puerto serie del 68HC11 se vuelve a enviar. Si se utiliza el programa MCBOOT en el PC se ve que todo lo tecleado aparece nuevamente escrito en la pantalla del PC.

```
; +-------------------------------------------------------------------------+
; | ECO.ASM  (C) GRUPO J&J. Febrero 1997                                    |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Se hace eco por el puerto serie del 68HC11.                          |
```

```
; +--------------------------------------------------------------------------+

; Registros del SCI

BAUD    equ  $2B
SCCR1   equ  $2C
SCCR2   equ  $2D
SCSR    equ  $2E
SCDR    equ  $2F

        ORG $0000

        LDX #$1000       ; Para acceder a registros del SCI

bucle   BSR leer_car     ; Esperar hasta que llegue un carácter por el SCI
        BSR enviar       ; Enviar el carácter recibido
        BRA bucle


;+-----------------------------------------------------------+
;¦ Rutina par leer un carácter del puerto serie (SCI)        ¦
;¦ La rutina espera hasta que llegue algún carácter          ¦
;¦ ENTRADAS: Ninguna.                                        ¦
;¦ SALIDAS: El acumulador A contiene el carácter recibido    ¦
;+-----------------------------------------------------------+
leer_car BRCLR SCSR,X $20 leer_car   ; Esperar hasta que llegue un carácter
         LDAA SCDR,X
         RTS


;+-----------------------------------------------------------+
;¦ Enviar un carácter por el puerto serie (SCI)               ¦
;¦ ENTRADAS: El acumulador A contiene el carácter a enviar    ¦
;¦ SALIDAS: Ninguna.                                         ¦
;+-----------------------------------------------------------+
enviar  BRCLR SCSR,X $80 enviar
        STAA SCDR,X
        RTS

        END
```

Lo importante de este ejemplo son las rutinas de enviar y recibir datos por el SCI. Las dos rutinas tienen la misma forma de actuación. Primero se espera a que se active el bit correspondiente en el registro SCSR y después se envía o recibe el carácter.

−EJEMPLO 3: Envío de cadenas por el SCI.

En este ejemplo se envía una cadena por el SCI cada vez que se recibe un carácter. Si se ejecuta el MCBOOT en el PC el resultado será que aparecerá la cadena ’hola como estas..." en la pantalla cada vez que se pulse una tecla.

```
; +--------------------------------------------------------------------------+
; ¦ SCICAD.ASM (C) GRUPO J&J. Febrero 1997                                   ¦
; ¦----------------------------------------------------------------------¦
; ¦  Programa ejemplo para ser ejecutado en la tarjeta CT6811.             ¦
; ¦  Este programa se debe cargar en la RAM interna del 6811.              ¦
; ¦                                                                        ¦
; ¦    Se envía una cadena por el puerto serie al pulsarse una tecla.      ¦
; +--------------------------------------------------------------------------+

; Registros del SCI

BAUD    equ  $2B
SCCR1   equ  $2C
SCCR2   equ  $2D
SCSR    equ  $2E
SCDR    equ  $2F

        ORG $0000

        LDX #$1000       ; Para acceder a registros del SCI

bucle   BSR leer_car     ; Esperar a que llegue un carácter por SCI
        LDY #hola        ; Meter en Y la dirección de la cadena hola
        BSR send_cad     ; Enviar la cadena por el puerto serie
        BRA bucle

;+-----------------------------------------------------------+
;¦ Rutina par leer un carácter del puerto serie (SCI)        ¦
;¦ La rutina espera hasta que llegue algún carácter          ¦
;¦ ENTRADAS: Ninguna.                                        ¦
;¦ SALIDAS: El acumulador A contiene el carácter recibido    ¦
;+-----------------------------------------------------------+
leer_car BRCLR SCSR,X $20 leer_car   ; Esperar hasta que llegue un carácter
        LDAA SCDR,X
        RTS

;+-----------------------------------------------------------+
;¦ Enviar un carácter por el puerto serie (SCI)               ¦
;¦ ENTRADAS: El acumulador A contiene el carácter a enviar    ¦
```

```
;¦ SALIDAS: Ninguna.                                          ¦
;+-------------------------------------------------------------+
enviar  BRCLR SCSR,X $80 enviar
        STAA SCDR,X
        RTS

;+-------------------------------------------------------------+
;¦ Enviar una cadena de caracteres por el puerto serie.       ¦
;¦ La cadena debe terminar con el carácter 0                  ¦
;¦ ENTRADAS: Registro Y contiene dirección cadena a enviar    ¦
;¦ SALIDAS: El acumulador A contiene el carácter recibido     ¦
;+-------------------------------------------------------------+
send_cad LDAA 0,Y        ; Meter en A el carácter a enviar
         CMPA #0         ; ¿Fin de la cadena?
         BEQ fin         ; Si--> retornar
         BSR enviar      ; NO--> enviar carácter.
         INY             ; Apuntar a la sig. posición de memoria
         BRA send_cad    ; Repetir todo
fin      RTS
;+--------+
;¦ DATOS  ¦
;+--------+
hola    FCC "Hola como estas.."
        FCB 0

        END
```

−EJEMPLO 4: Recepción por interrupciones.

En este ejemplo se envían constantemente los caracteres ’A’ y ’B’ por el puerto serie mediante espera activa. Cada vez que se recibe un carácter se ejecuta la rutina de atención mediante interrupciones. Esta rutina simplemente lee el carácter recibido y si es igual al carácter ’A’ cambia el led de la CT6811 de estado. Si no es el carácter ’A’ se ignora.

```
; +-------------------------------------------------------------------------+
; ¦ SCIINT.ASM (C) GRUPO J&J. Febrero 1997                                  ¦
; ¦-------------------------------------------------------------------------¦
; ¦  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              ¦
; ¦  Este programa se debe cargar en la RAM interna del 6811.               ¦
; ¦                                                                         ¦
; ¦    Se envía una cadena por el puerto serie al pulsarse una tecla.       ¦
; +-------------------------------------------------------------------------+

; Registros del SCI

BAUD    equ  $2B
SCCR1   equ  $2C
SCCR2   equ  $2D
SCSR    equ  $2E
SCDR    equ  $2F

       ORG $0000

        LDX #$1000
        BSET SCCR2,X $20  ; Activar interrupción de recepción de datos del SCI
        CLI               ; Permitir las interrupciones
bucle
        LDAA #’A’
        BSR enviar
        LDAA #’B’
        BSR enviar
        BRA bucle

;+-------------------------------------------------------------+
;¦ Enviar un carácter por el puerto serie (SCI)               ¦
;¦ ENTRADAS: El acumulador A contiene el carácter a enviar    ¦
;¦ SALIDAS: Ninguna.                                          ¦
;+-------------------------------------------------------------+
enviar  BRCLR SCSR,X $80 enviar
        STAA SCDR,X
        RTS

;+-------------------------------------------------------------+
;¦ Rutina de servicio de las interrupciones del SCI           ¦
;¦ Se determina la causa de interrupción del SCI y se salta   ¦
;¦ a la rutina correspondiente.                               ¦
;+-------------------------------------------------------------+
sci
        BRSET SCSR,X $80 recibir    ; Si se ha recibido un carácter
                                    ; saltar a la Rutina correspondiente.
        RTI

;+-------------------------------------------------------------+
;¦ Rutina de atención de la interrupción DATO RECIBIDO del    ¦
;¦ SCI. Cada vez que se recibe un carácter se ejecuta esta    ¦
;¦ rutina de atención.                                        ¦
;+-------------------------------------------------------------+
recibir
        LDAA SCDR,X     ; Leer dato recibido
        CMPA #’A’       ; ¿Se ha recibido el carácter ’A’?
        BNE fin         ; NO --> Retornar
```

```
        LDAA $1000      ; Cambiar de estado el LED
        EORA #$40
        STAA $1000
fin                     ; Retorno rutina de servicio
        RTI
;+-----------------------------------------------------------+
;¦ Vector de interrupción del SCI. En cuanto se produce      ¦
;¦ alguna interrupción del SCI se salta a la rutina sci      ¦
;+-----------------------------------------------------------+
        ORG $00C4
        JMP sci
         END
```

Obsérvese que en el bucle principal simplemente se envían los caracteres 'A' y 'B', no se tratan los caracteres recibidos. Esto se realiza 'automáticamente' mediante interrupciones. Al ejecutar este programa se puede hacer una idea de la potencia de las interrupciones.

4.3.5.− Mecanismo de interrupciones del SCI.

Existen 4 causas de interrupción en el SCI, al aparecer alguna de estas 4 se activa su bit correspondiente en el registro de estado. El SCI sólo tiene asociado un vector de interrupción por lo que sólo existe una rutina de servicio . Esta rutina de servicio tiene que leer el registro de estado (SCSR) y detectar cuál de las 4 causas de interrupción ha ocurrido. Según la cual bifurcará a una rutina o a otra. Finalmente se retornará con RTI. Ver ejemplo número 4 del apartado anterior.

Es importante hacer notar que siempre que haya un bit a 1 en el registro de estado, la CPU lo interpretará como una petición de interrupción y llamará a la rutina de servicio del SCI. Por tanto es **MUY IMPORTANTE** poner el bit correspondiente a cero para que al terminar la rutina de servicio la CPU no se vuelva a "interrumpir". Esto último es un punto muy importante. Supongaose que se produce una interrupción en el SCI porque se ha recibido un carácter. El bit 5 del registro de estado se pondrá a 1 indicando el evento. La CPU detecta que este bit está activado y pasa a ejecutar la rutina de servicio del SCI. Si esta rutina no pone a cero este bit, al terminar con RTI el bit seguirá activado y la CPU lo tomará como una nueva interrupción que se ha producido, con lo que se volverá a ejecutar la rutina de servicio. Así permanecerá la CPU en un bucle infinito ejecutando constantemente la rutina de servicio del SCI hasta que se produzca un reset externo.

Los bits del registro de estado no se pueden poner a cero directamente. Para ponerlos a cero es preciso realizar algún tipo de acción como es leer otros registros. Estos bits se ponen a cero cuando se lee el SCSR y a continuación se lee el registro de datos SCDR.

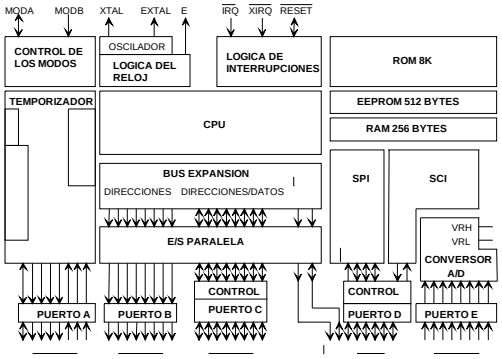
La siguiente tabla muestra las causas de interrupción , el bit del registro de estado asociado a cada una y el bit del registro de control 2 que las enmascara.

Motivo de interrupción	Bit de estado	Máscara interrupción
Registro de transmisión vacío	TDRE	TIE
Registro de transmisión vacío y línea transmisión vacía	TC	TCIE
Dato recibido	RDRF	RIE
Dato recibido y línea recepción vacía (IDLE)	IDLE	ILIE

4.4. TRANSMISIONES SERIE SINCRONAS

4.4.1. Introducción

La unidad SPI (Serial Peripheral Interface) es uno de los potentes periféricos que lleva incorporado el microcontrolador. Se trata fundamentalmente de un sistema de comunicaciones serie síncrono de alta velocidad. Si bien el SPI puede ser utilizado simplemente como un pequeño puerto con señales de entrada y salida, su uso más frecuente es el de comunicar varios dispositivos entre sí, ya sean simples periféricos u otros microcontroladores.



Para realizar las comunicaciones, el MCU permite seleccionar entre dos modos de funcionamiento: el **modo maestro** y **el modo esclavo**. Cuando se realizan redes de comunicaciones (entre dos o más dispositivos) solamente está permitido la existencia de un solo maestro, mientras que la de esclavos está indefinida. Realmente su número viene dado por las necesidades del sistema en desarrollo. Si bien las virtudes del modo maestro se comentarán más adelante hay que hacer notar que debido a la forma de configurar los SPI, es posible realizar cambios de turno o control, es decir, hacer que un MCU que antes era esclavo ahora sea maestro y viceversa.

La potencia de la unidad llega al límite al permitir transmisiones full duplex (en ambos sentidos simultáneamente) entre un maestro y un esclavo. A partir de aquí, es posible realizar desde una simple comunicación unidireccional entre el MCU y un periférico hasta construir enlaces jerárquicos complejos entre MCUs y/o periféricos.

4.4.2. Protocolo

Cuando el maestro tiene que mandar un mensaje a uno o varios esclavos debe proceder a realizar una selección de los mismos, trabajando al igual que si se tratara de un chip−enable. De esta forma, al ser activado el esclavo, recibe el dato manteniendo el sincronismo gracias a una señal de reloj conjunta. Es posible que cuando un esclavo sea activado con el fin de recibir un dato, este desee enviar una trama de respuesta al maestro. Esto será posible mientras su línea de activación la mantenga el maestro, de modo que si es necesario la transmisión se efectuará simultáneamente en los dos sentidos.

Existen cuatro líneas básicas asociadas a la unidad SPI mediante las cuales es posible montar los diferentes enlaces:

MOSI (Master Out, Slave In) esta es la línea por donde circulan los datos que el maestro quiere enviar a los esclavos, por tanto será la señal de salida de datos de la unidad que funcione como maestro y la señal de entrada de datos para los esclavos.

MISO (Master In, Slave Out) por esta línea viajarán los datos que sean enviados desde algún esclavo hacia el maestro, de esta forma será una señal de entrada para el maestro y las respectivas salidas para los esclavos.

SCK (Serial Clock) representa la señal de reloj que se producen las comunicaciones. Si bien es que cada unidad configure software la velocidad es previsible pensar que en comunicación sólo podrá prevalecer una, siendo esta la del maestro (virtudes de los "masters"...). Por tanto, para los esclavos representará una señal de entrada mientras que para el maestro será una salida.

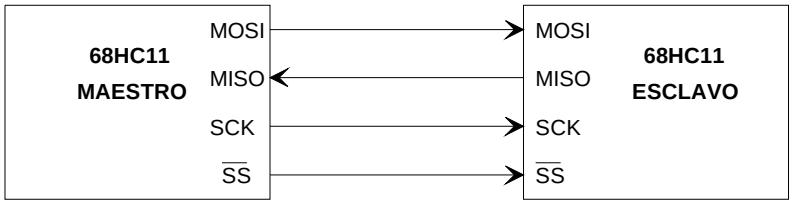


Figura 26: Comucación de dos 68HC11 mediante el SPI

SS (Slave Select) está línea tiene una funcionalidad muy concreta en las unidades esclavas ya que representa sus respectivas entradas de chip−enable. Cuando la unidad se configura como maestra puede utilizarse para diferentes fines que luego se comentarán.

4.4.3. Dispositivos SPI

Mediante software se puede seleccionar la velocidad, la polaridad y la fase de la señal de reloj que maneja el sincronismo de tal manera que es un sistema compatible con muchos de los periféricos que permiten entrada serie directa (puertos, lcd, teclados, dac, adc, etc...).

Igualmente hay que destacar que la construcción de sistemas que soporten las comunicaciones síncronas con el MCU no es una tarea ni mucho menos ardua, ya que un simple registro de desplazamiento con un correcto cableado se puede considerar como un puerto.

Es notable el ahorro de líneas de conexión que se genera en comparación con un puerto estándar, donde es necesario como mínimo el cableado del bus de datos.

4.4.4. Enlaces

Para activar los SPI esclavos es necesario disponer del control sobre las líneas respectivas SS. Para una comunicación del tipo MCU−MCU o simplemente MCU−periférico, este control se solventa de una manera sencilla ya que es posible conectar directamente la línea SS del maestro trabajando como entrada/salida a la entrada de selección del esclavo (unir los SS). Un ejemplo de conexión entre dos 68HC11 se muestra en la figura 26.

En la necesidad de realizar comunicaciones entre varios dispositivos, como por ejemplo un MCU con varios periféricos y/o con varios MCUs, la opción habitual de diseño se encuentra en montar un hardware externo que se encargue de ir recibiendo las ordenes del maestro y activando los esclavos correspondientes. La generación de los "enables" puede resultar muy sencilla utilizando un propio puerto de MCU, o compleja cuando el sistema de comunicaciones así lo requiera. Esta forma de realizar los enlaces entre los SPIs dota al mismo de la capacidad de soportar redes de comunicaciones que incorporen protocolos basados en máscaras, seguridad hardware, etc...

En la figura 27 se muestra un esquema de la conexión de un 68HC11 con varios periféricos a través del SPI. Los periféricos pueden ser puertos de E/S, conversores A/D y D/A, otros 68HC11...

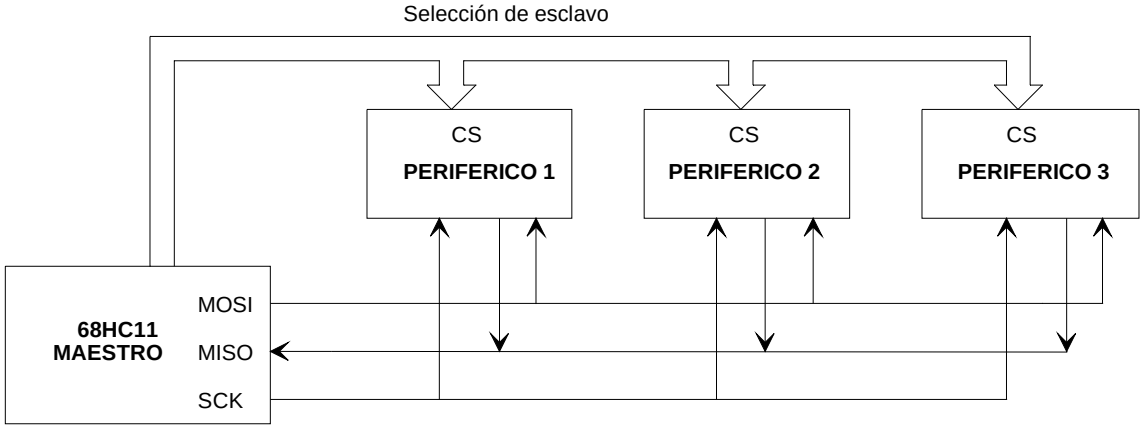


Figura 27: Conexión del 68HC11 a varios periféricos a través del SPI

4.4.5. Registros del SPI

Mapeados en la memoria del 68HC11 se dispone de tres registros:

REGISTRO	DIRECCIÓN	DESCRIPCIÓN
CONTROL	$1028	Registro de control
ESTADO	$1029	Registro de estado
DATOS	$102A	Registro de datos

−Serial Peripheral Control Register (SPCR $1028):

7							0
SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0

SPIE: Serial Peripheral Interrupt Enable.
1= Habilita las interrupciones del SPI.
0= Deshabilita las interrupciones del SPI.
SPE: Serial Peripheral System Enable.

1= Activa el SPI.
0= Desactiva el SPI.

DWOM: Port D Wire−Or Mode Option
1= La totalidad del puerto D actúa con salidas en colector abierto.
0= La totalidad del puerto D actúa con salidas cmos.

MSTR: Master Mode Select.
1= Configurado el SPI en modo Maestro.
0= Configurado el SPI en modo Esclavo.

CPOL: Clock Polarity.
1= El reloj se mantiene a nivel alto mientras no existan datos a transmitir.
0= El reloj se mantiene a nivel bajo mientras no existan datos a transmitir.

CPHA: Clock Phase.
Mirar figura 29 para ver la temporización.

SPR1, SPR0: SPI Clock Rate Select.
Permiten seleccionar la velocidad de transmisión según la siguiente tabla.

SPR1	**SPR0**	Reloj interno **E** dividido por:
0	0	2
0	1	4
1	0	16
1	1	32

−Serial Peripheral Status Register (SPSR $1029).

SPIF	WCOL	0	MODF	0	0	0	0

SPIF: SPI Transfer Complete Flag.
1= Se pone a uno cuando ha finalizado una transmisión entre el MCU y un periférico (u otro MCU).
0= Se pone a cero realizando una lectura sobre el registro seguido de un acceso al registro de datos SPDR.

WCOL: Write Collision.
1= Automáticamente se activa cuando se ha intentado escribir en el registro de datos antes de que la transmisión hubiese terminado. Es decir, cuando una vez colocado el byte a transmitir, y durante su transferencia se ha sobrescrito en dato con un nuevo byte a transferir.
0= Se pone a cero al realizar una lectura de este registro seguido de un acceso al registro de datos.

MODF:Mode Fault.
1= Se pone a uno cuando un MCU trabajando como maestro ha querido ser seleccionado como si este fuera esclavo.
0= Se pone a cero realizando una lectura del registro seguido de un acceso al registro de datos.

−Serial Peripheral Data I/O Register (SPDR $102A). Registro de datos.

Este registro es el utilizado para transmitir o recibir datos del bus serie que conforma el SPI. Una escritura en este registro cuando previamente se ha configurado el sistema como maestro iniciará una transferencia de un byte. De la misma manera las unidades configuradas como esclavas podrán leer los datos recibidos en sus respectivos registros.

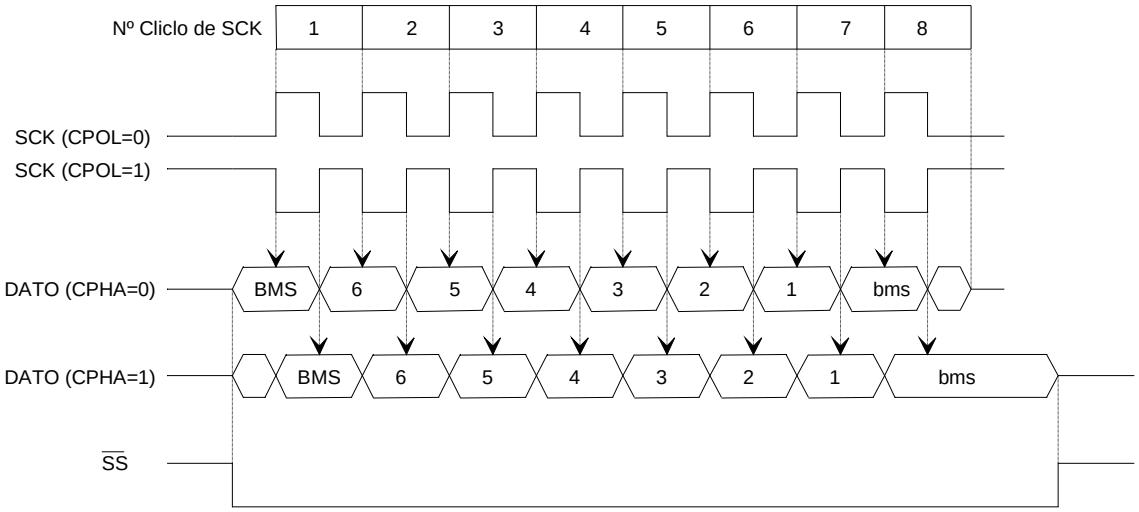


Figura 28: Cronograma del envío de un byte por el SPI

4.4.6. Ejemplo de programación del SPI

Se trata de un enlace entre dos microcontroladores a través del canal síncrono. Los dos supuestos nodos (micros) se encuentran a su vez conectados a sendos Pcs vía serie asíncrona estándar, de tal forma que al presionar una tecla en uno de ellos (Maestro), el 68HC11 asociado recibe el dato transmitiéndolo vía SPI al otro para que este a su vez lo transmita al otro Pc (Esclavo). De esta manera los dos ordenadores quedan virtualmente conectados por el SPI. La red queda conformada por dos enlaces asíncronos entre los micros y los Pcs respectivos, y por un enlace síncrono entre los propios microcontroladores.

El programa ejemplo se ha probado conectando dos tarjetas CT6811 como se muestra en la figura 29.

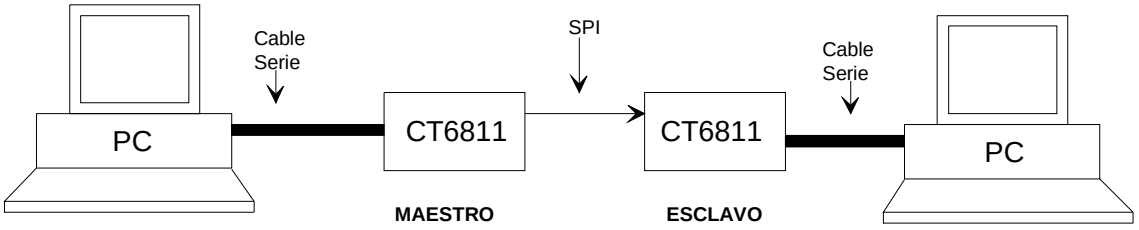


Figura 29: Montaje realizado para probar el funcionamiento del SPI

PROGRAMA MAESTRO:

```
; +---------------------------------------------------------------------+
; | MAESTRO.ASM (C) GRUPO J&J. Abril 1997                               |
; |---------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.          |
; |  Este programa se debe cargar en la RAM interna del 6811.           |
; |                                                                     |
; |    Se envía el carácter recibido por el SCI a través del SPI.       |
; +---------------------------------------------------------------------+
; Configuración del MAESTRO.

; DDRD:      .       .      !ss     sck     mosi    miso    txd     rxd
;            0       0       1       1       1       1       0       0

; El ss se desactiva, en caso contrario, se activaría con una entrada a nivel
; bajo => D/P
; La salida será mosi, y miso da igual su valor

; SPCR:     spie     spe     dwom    mstr    cpol    cpha    spr1    spr0
;            0        1       0       1       0       0       0       0
; Desactivamos las interrupciones, activamos el SPI, salida con drivers,
; configurado como maestro, formato de señal y alta velocidad.

; SPSR:     spif     wcol     .      modf    .       .       .       .
;            x
; Leemos el flag spif que se pone a uno cuando se termina la transmisión.

; Constantes:

PORTD   equ       $08     ; Registro de datos del puerto D.
BLOQ    equ       $F000   ; Comienzo de la zona de datos a transmitir.
TAM     equ       $1FFF   ; Tamaño del buffer a transmitir.

; Registros del SPI.
```

```
DDRD    equ       $09     ; Registro de control de dirección de datos.
SPCR    equ       $28     ; Registro de control del SPI.
SPDR    equ       $2A     ; Registro de datos del SPI.
SPSR    equ       $29     ; Registro de estado.

; Registros del SCI.

BAUD    equ       $2B
SCCR1   equ       $2C
SCCR2   equ       $2D
SCSR    equ       $2E
SCDR    equ       $2F

; Comienzo.

            ORG $0000
            LDX #$1000
            LDAA #$00           ; Manda algún valor conocido al
            STAA PORTD,X        ; puerto.

; Configura la unidad SPI.

            LDAA #$3C
            STAA DDRD,X
            LDAA #$50
            STAA SPCR,X

; Espera una tecla del PC para empezar...

Bucle       BSR leer_car
            STAA SPDR,X                 ; Enviar la tecla apretada por el SPI
WaitFlag    BRCLR SPSR,X $80 WaitFlag   ; Espera el fin de la trans.
            BRA Bucle                   ; Repetir indefinidamente

;+-----------------------------------------------------------+
;¦ Rutina par leer un carácter del puerto serie (SCI)        ¦
;¦ La rutina espera hasta que llegue algún carácter          ¦
;¦ ENTRADAS: Ninguna.                                        ¦
;¦ SALIDAS: El acumulador A contiene el carácter recibido    ¦
;+-----------------------------------------------------------+
leer_car BRCLR SCSR,X $10 leer_car   ; Esperar hasta que llegue un carácter
        LDAA SCDR,X
        RTS

        END
```

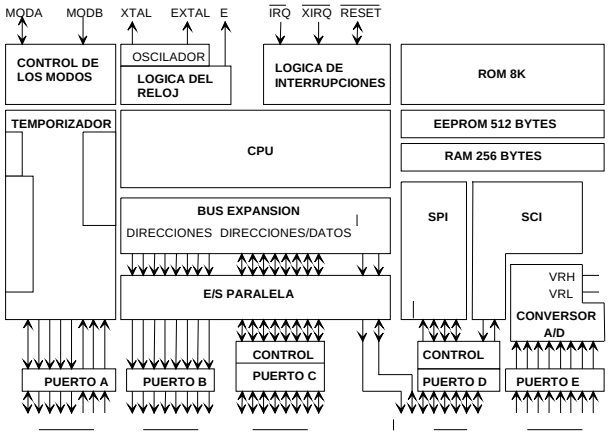
PROGRAMA ESCLAVO:

```
; +-------------------------------------------------------------------------+
; ¦ ESCLAVO.ASM (C) GRUPO J&J. Abril 1997                                   ¦
; ¦---------------------------------------------------------------------    ¦
; ¦  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              ¦
; ¦  Este programa se debe cargar en la RAM interna del 6811.               ¦
; ¦                                                                         ¦
; ¦    Se envía por el SCI todo lo que se recibe por el SPI.                ¦
; +-------------------------------------------------------------------------+

; Programa de prueba para la transmisiones sincronas. MISO/MOSI.
; Grupo J&J. Mayo 1996.
; Configuración del ESCLAVO. -> Simulador de la LCA.

; DDRD:      .       .      !ss     sck     mosi    miso    txd     rxd
;            0       0       1       1       1       1       0       0
; El ss puede tener cualquier valor, ya que se activara por el maestro.
; sck es la entrada de reloj, da igual su valor.
; La entrada será mosi (da igual su valor), y miso es la salida.
;

; SPCR:     spie     spe     dwom    mstr   cpol    cpha    spr1    spr0
;            0        1       0       0      0       0       0       0
; Desactivamos las interrupciones, activamos el SPI, salida con drivers,
; configurado como esclavo, formato de señal (igual a la del maestro).

; SPSR:     spif     wcol     .      modf    .       .       .       .
;            x
; Leemos el flag spif que se pone a uno cuando se termina la transmisión.

; Constantes:

PORTD   equ       $08     ; Registro de datos del puerto D.
BLOQ    equ       $F000   ; Comienzo de la zona de datos a transmitir.
TAM     equ       $1FFF     ; Tamaño del buffer a transmitir.

; Registros del SPI.

DDRD    equ       $09     ; Registro de control de direcc de datos.
SPCR    equ       $28     ; Registro de control del SPI.
SPDR    equ       $2A     ; Registro de datos del SPI.
```

```
SPSR    equ       $29     ; Registro de estado.

; Registros del SCI.

BAUD    equ       $2B
SCCR1   equ       $2C
SCCR2   equ       $2D
SCSR    equ       $2E
SCDR    equ       $2F

; Comienzo.

            ORG $0000
            LDX #$1000

; Configura la unidad SPI.

            LDAA #$3C
            STAA DDRD,X
            LDAA #$43           ;40 Max, 43 Min
            STA SPCR,X

; Comienza la recepción.

WaitFlag    BRCLR SPSR,X $80 WaitFlag     ; Espera el flag.
            LDAA SPDR,X                    ; Leer carácter del SPI
            BSR enviar                    ; Enviarlo por el SCI
            JMP WaitFlag

;+-----------------------------------------------------------+
;| Enviar un carácter por el puerto serie (SCI)              |
;| ENTRADAS: El acumulador A contiene el carácter a enviar   |
;| SALIDAS: Ninguna.                                         |
;+-----------------------------------------------------------+
enviar  BRCLR SCSR,X $80 enviar
        STAA SCDR,X
        RTS

        END
```

4.5. TEMPORIZADOR PRINCIPAL

4.5.1. Introducción



El temporizador está formado por un registro de 16 bits que se está incrementando constantemente. Arranca con el valor 0, se va incrementando con cada pulso de reloj y cuando llega a $FFFF se produce una interrupción de overflow, pasa a valer nuevamente 0 y continúa contando.

El valor del temporizador se puede leer en cualquier momento pero no escribir. La cuenta por tanto no se para.

La señal de reloj que incrementa el temporizador se obtiene dividiendo la frecuencia de la señal de reloj E por uno de estos valores: 1,4,8 ó 16.

El registro que contiene el valor del temporizador es el **TCNT** y se encuentra en las direcciones $100E (Byte mayor peso) y $100F (Byte de menor peso). Para leer su valor basta con utilizar cualquiera de los registros de 16 bits, por ejemplo el Y:

```
LDY $100E     ; Cargar en el registro Y el valor del temporizador
```

Para dividir la señal de reloj que incrementa el temporizador se tienen que introducir unos determinados valores en los bits 0 y 1 (PR0 y PR1) del registro **TMSK2** ($1024). Para un cristal de 8MHZ:

PRI	PR0	Tiempo total	Resolución
0	0	32.77ms	500ns
0	1	131.1ms	2us
1	0	261.1ms	4us
1	1	524.3ms	8us

El tiempo total es el invertido por el temporizador en dar una vuelta completa; es decir, partiendo de cero, es el tiempo que tarda en volver a cero. La resolución es el tiempo que tarda en pasar de un valor al siguiente.

La interrupción de overflow se utiliza para "extender" el contador. Por ejemplo, si se quiere que el temporizador cuente hasta el numero $3FFFF en vez de sólo hasta el $FFFF, cada vez que el contador llegue a $FFFF y se produzca la interrupción de overflow, se incrementa un byte en memoria que representa el byte más significativo del contador. De esta forma se consigue que el temporizador sea de 32bits (y de más bits...). Esta interrupción se habilita o deshabilita con el bit 7 del registro **TMSK2** ($1024). La bandera de interrupción es el bit 7 del registro **TFLG2** ($1025). Este bit se activa cada vez que se produce la interrupción de overflow y debe ser puesto a cero antes de abandonar la rutina de servicio.

El temporizador en sí no tiene casi utilidad de cara al programador. Pero si la tiene internamente, ya que sirve para controlar otros recursos muy importantes: Capturadores, comparadores, interrupciones en tiempo real, acumulador de pulsos....

Buscando una utilidad se destaca la obtención de números aleatorios. Cada vez que se quiera obtener un número aleatorio de 8 bits basta con leer la parte baja del valor del temporizador.

4.5.2. Los registros del temporizador

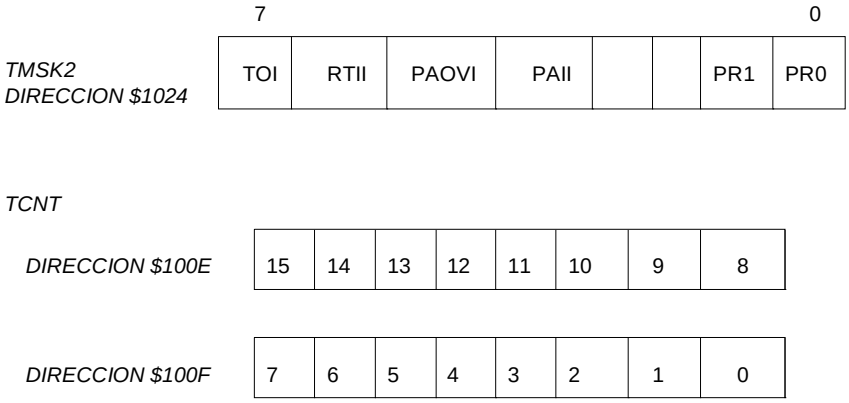


Figura 30: Registros del temporizador

4.5.3. Diagrama de bloques del temporizador

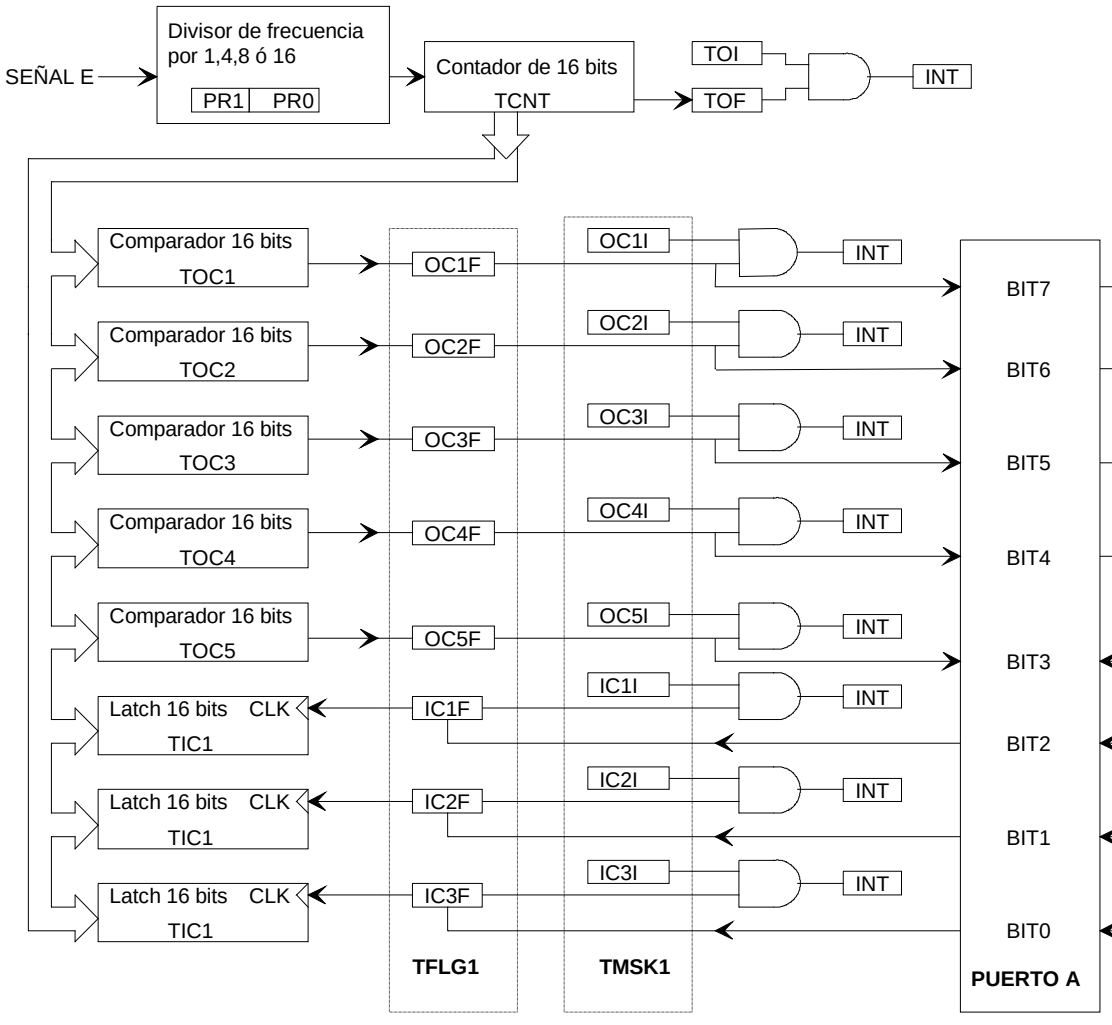


Figura 31: Diagrama de bloques del temporizador principal

En la figura 31 se muestra un esquema del sistema de temporización. La señal E, que es el reloj del sistema, se introduce primero por un divisor de frecuencia programable. Este divisor se modifica cambiando los bits PR0 y PR1 del registro **TMSK2**. La señal E se puede dividir por 1, 4, 8 ó 16. Se recuerda que para un cristal de 8 MHZ, la señal E es de 2 MHZ. En la tarjeta CT6811 se dispone de un cristal de 8MHZ. Por ello, en esta tarjeta las señales que se obtienen al dividir la señal E son de 2MHZ, 500 KHZ, 250 KHZ. y 125KHZ.

La señal E dividida se utiliza como señal de reloj para incrementar un contador de 16 bits. Este es el temporizador principal, cuyo valor se lee pero no se escribe. El 68hc11 lo utiliza internamente para proporcionar

otras funciones. Una de ellas son los comparadores. En este caso el valor del contador se introduce en los 5 comparadores que se pueden activar cuando el valor introducido es igual a uno previo que ha sido almacenado por software. La otra función son los capturadores de entrada. Cuando ocurre un determinado evento exterior el valor del contador es capturado por los 3 latches que constituyen los capturadores de entrada. Tanto los comparadores como los capturadores se tratan en más detalle en las secciones 4.7 y 4.8.

4.5.4. Ejemplo de programación del temporizador

−EJEMPLO 1: Lectura del temporizador**.**

En este ejemplo se muestra cómo leer el temporizador principal. El programa toma el bit más significativo y activa el led (bit PA6) según cómo esté el estado del bit más significativo del temporizador. Por defecto el temporizador funciona a su máxima frecuencia por lo que el led cambiará de estado cada 32.77ms. A esta frecuencia es casi inapreciable el parpadeo del led, pero fijándose detenidamente se observa que realmente existe.

```
; +--------------------------------------------------------------------------+
; | TIMER.ASM  (C) GRUPO J&J. Febrero 1997                                   |
; |--------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.               |
; |  Este programa se debe cargar en la RAM interna del 6811.                |
; |                                                                          |
; |    Ejemplo del temporizador principal. Simplemente se lee el valor       |
; | del temporizador principal y se modifica el estado del led de la         |
; | CT6811 en función del estado del bit más significativo del temporizador  |
; +--------------------------------------------------------------------------+

TMSK2   EQU $24
TCNT    EQU $0E
PORTA   EQU $00

        ORG $0000

        LDX #$1000

bucle
        LDD TCNT,X      ; Leer el valor del temporizador principal
                        ; A=Parte alta; B=Parte baja
        ANDA #$80
        CMPA #$80       ; Comprobar bit de mayor peso del temporizador
        BEQ apaga_luz   ; Si está activo--> Apagar led
        LDAA #$40       ; No activo--> Encender el led
        STAA PORTA,X
        BRA bucle

apaga_luz
        CLRA
        STAA PORTA,X
        BRA bucle

        END
```

−EJEMPLO 2: Modificación frecuencia del temporizador.

Se trata del mismo programa anterior pero ahora el temporizador se configura para trabajar a una frecuencia mucho menor. La señal E de divide por 16 por lo que la nueva frecuencia del temporizador es de 125Khz

```
; +--------------------------------------------------------------------------+
; | TIMER2.ASM (C) GRUPO J&J. Febrero 1997                                   |
; |--------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.               |
; |  Este programa se debe cargar en la RAM interna del 6811.                |
; |                                                                          |
; |    Ejemplo del temporizador principal. El programa hace lo mismo que     |
; | TIMER.ASM pero se modifica la frecuencia de funcionamiento del tempo-    |
; | rizador. El led cambiará de estado mucho más lentamente.                 |
; +--------------------------------------------------------------------------+

TMSK2   EQU $24
TCNT    EQU $0E
PORTA   EQU $00

        ORG $0000

        LDX #$1000
        BSET TMSK2,X $03  ; Dividir la señal E por 16

bucle
        LDD TCNT,X      ; Leer el valor del temporizador principal
                        ; A=Parte alta; B=Parte baja
        ANDA #$80
        CMPA #$80       ; Comprobar bit de mayor peso del temporizador
        BEQ apaga_luz   ; Si está activo--> Apagar led
        LDAA #$40       ; No activo--> Encender el led
        STAA PORTA,X
        BRA bucle

apaga_luz
        CLRA
        STAA PORTA,X
        BRA bucle

        END
```

−EJEMPLO 3: Interrupción de overflow del temporizador**.**

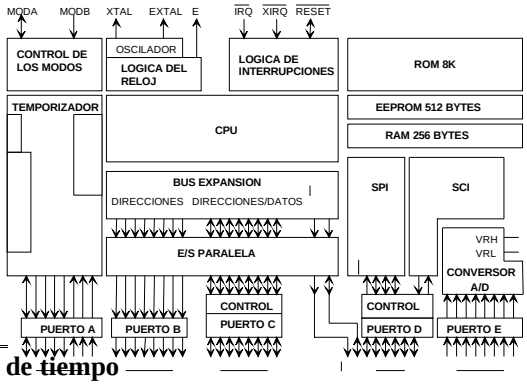
En este programa se utiliza la interrupción de overflow del temporizador para hacer parpadear el led de la tarjeta CT6811.

```
; +--------------------------------------------------------------------------+
; | OVERFLOW.ASM (C) GRUPO J&J. Febrero 1997                                 |
; |--------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.               |
; |  Este programa se debe cargar en la RAM interna del 6811.                |
; |                                                                          |
; |    Ejemplo del temporizador principal. Se utiliza la interrupción de     |
; | overflow del temporizador para hacer parpadear el LED de la CT6811.      |
; | Cada vez que el temporizador alcanza el valor $FFFF se cambia de esta-   |
; | do el led. Se configura el temporizador para que se produzca overflow    |
; | cada 524.3ms                                                             |
; +--------------------------------------------------------------------------+

TMSK2   EQU $24
TFLG2   EQU $25
TCNT    EQU $0E
PORTA   EQU $00

        ORG $0000

        LDX #$1000
        BSET TMSK2,X $03  ; Dividir la señal E por 16
        BSET TMSK2,X $80  ; Activar interrupción de overflow

        CLI               ; Permitir las interrupciones

inf     BRA inf


;--- Rutina de servicio de interrupción del overflow
overflow
        BSET TFLG2,X $80  ; Quitar flag de int. de overflow

        LDAA $1000        ; Cambiar de estado el led de la CT6811
        EORA #$40
        STAA $1000
        RTI
```

```
ORG $00D0
JMP overflow

END
```

La primera línea de la rutina de atención a la interrupción de overflow (BSET TFLG2,X $80) lo que hace es poner a cero el flag de interrupción de esta interrupción. Se hace con la instrucción BSET y no con BCLR porque para ponerlo a cero hay que escribir un 1 sobre este bit.

4.6. INTERRUPCIONES EN TIEMPO REAL



4.6.1. Introducción

La interrupción en tiempo real (Real Time Interrupt) es una interrupción que se produce cada ciertos instantes fijos de tiempo. El intervalo de tiempo comprendido entre 2 interrupciones de tiempo real consecutivas se puede configurar modificando los bits 0 y 1 del registro **PACTL** ($1026). Para un cristal de 8 MHZ se tiene que:

Bit 1	Bit 0	Intervalo de tiempo
0	0	4.096ms (4.1ms aprox)
0	1	8.192ms (8.2ms aprox)
1	0	16.384ms (16.4ms aprox)
1	1	32.768ms (32.77ms aprox)

Por ejemplo, configurando los bits 0 y 1 con el valor 1, cada 32.7ms se producirá una interrupción en tiempo real.

¿Para qué se utiliza esta interrupción? Tiene múltiples aplicaciones. Por ejemplo para programar relojes en tiempo real, cronómetros, calendarios, muestrear datos, etc.

El bit 6 del registro **TFLG2** se activa cada vez que se produce una interrupción en tiempo real. Este bit es el que está indicando a la CPU que hay una interrupción de tiempo real pendiente, por lo que la rutina de servicio de esta interrupción lo primero que debe hacer es poner este bit a cero. Si no se hace, al terminar la rutina la CPU detecta que este bit sigue activo y lo toma como si se hubiese producido otra interrupción. Esto produce que se vuelve a ejecutar la rutina de servicio y la CPU entra en un bucle infinito. El bit 6 del registro **TMSK2** ($1024) es el que habilita o deshabilita la interrupción de tiempo real.

El bit se pone a cero escribiendo un "1" en ese bit y ceros en los demás; es decir, escribiendo el valor $40 en el registro **TFLG2**. Esto se puede hacer utilizando STAA o BCLR.

Ejemplo:

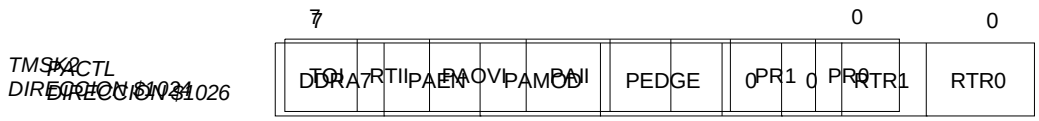
```
LDAA $40
STAA TFLG2,X
```

4.6.2. Los registros de las interrupciones en tiempo real

En la siguiente tabla se resumen los registros relacionados con las interrupciones de tiempo real:

PACTL	$1026	Bits 0 y 1 establecen período tiempo
TMSK2	$1024	Bit 6 Activar o desactivar interrupción.
TFLG2	$1025	Bit 6 interrupción pendiente.

En las siguientes figuras se muestran los registros relacionados con más detalle.



RTR1,RTR2 = Determinan el periodo de tiempo entre cada interrupción de tiempo real

TOI = Permiso de interrupción de overflow en timmer(1 permitida, 0 inhibida)

RTII = Permiso de interrupción de tiempo real

PAOVI= Permiso de interrupción de overflow en acumulador de pulsos

PR1,PR0 =Factor de división de la señal E

	7							0
TFLG2 *DIRECCION $1025*	TOF	RTIF	PAOVF	PAIF				

TOF = Flag de interrupción de overflow del temporizador

RTIF = Flag de interrupción de tiempo real

PAOVF= Flag de interrupción de overflow del acumulador de pulsos

PAIF = Flag de interrupción del acumulador de pulsos

4.6.3. Ejemplos de manejo de las interrupciones en tiempo real

−EJEMPLO 1: Interrupciones en tiempo real mediante espera activa.

Este programa hace cambiar de estado el led de la tarjeta CT6811 cada 32.7ms. Se realiza utilizando las interrupciones en tiempo real y espera activa. En realidad no se utilizan interrupciones, sino que se trabaja con el flag de interrupción. cuando se pone a 1 quiere decir que han pasado 32.7ms. Las interrupciones quedan deshabilitadas.

```
; +-------------------------------------------------------------------------+
; | RTI.ASM      (C) GRUPO J&J. Febrero 1997                                |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Ejemplo de las interrupciones en tiempo real. Cambiar el estado      |
; | del led cada 32.7ms. Se hace mediante espera activa.                    |
; +-------------------------------------------------------------------------+

TMSK2   EQU $24
TFLG2   EQU $25
PACTL   EQU $26
PORTA   EQU $00

        ORG $0000

        LDX #$1000

        BSET PACTL,X $03        ; Int. en tiempo real cada 32.7ms
bucle
main    BRCLR TFLG2,X $40 main  ; Esperar a que se active el flag

        BSET TFLG2,X $40        ; Poner a cero flag de interrupción
        LDAA PORTA,X
        EORA #$40
        STAA PORTA,X
        BRA bucle

        END
```

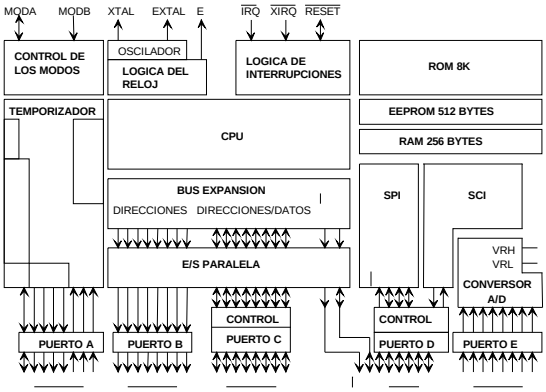
−EJEMPLO 2: Interrupción en tiempo real con interrupciones**.**

Mismo programa anterior pero realizado mediante interrupciones.

```
; +-------------------------------------------------------------------------+
; | RTII.ASM      (C) GRUPO J&J. Febrero 1997                               |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Ejemplo de las interrupciones en tiempo real. Cambiar el estado      |
; | del led cada 32.7ms. Se hace mediante interrupciones.                   |
; +-------------------------------------------------------------------------+

TMSK2   EQU $24
TFLG2   EQU $25
PACTL   EQU $26
PORTA   EQU $00

        ORG $0000

        LDX #$1000

        BSET PACTL,X $03        ; Int. en tiempo real cada 32.7ms
        BSET TMSK2,X $40        ; Habilitar int en tiempo real
        CLI                     ; Permitir las interrupciones

INF     BRA INF


;------- Rutina de atención a la interrupción en tiempo real
rti
        BSET TFLG2,X $40        ; Poner a cero flag de interrupción

        LDAA PORTA,X            ; Cambiar led de estado
        EORA #$40
        STAA PORTA,X
        RTI

        ORG $00EB
        JMP rti

        END
```

4.7. COMPARADORES

4.7.1. Introducción



Existen 5 comparadores independientes de 16 bits en el microcontrolador. El funcionamiento de cada comparador es el siguiente: Se introduce un valor en sus registros correspondientes. El temporizador principal se va incrementando. Cuando el valor del temporizador es igual al de algún comparador, se produce una interrupción (si están habilitadas) y se modifica opcionalmente el estado lógico de un pin del microcontrolador. Ver la figura 31, en la sección 4.5.3

Los registros de datos son los siguientes:

Nº comparador	Registro	Byte alto	Byte bajo
1	TOC1	$1016	$1017
2	TOC2	$1018	$1019
3	TOC3	$101A	$101B
4	TOC4	$101C	$101D
5	TOC5	$101E	$101F

Puesto que los comparadores son independientes, existe una interrupción para cada uno que se puede enmascarar mediante un bit situado en el registro de máscara de interrupciones **TMSK1** ($1022). El registro de interrupciones **TFLG1** ($1023) contiene las banderas de interrupción: los bits que se activan cuando se produce alguna de las interrupciones de los comparadores, bits que deben ser puestos a cero antes de terminar la rutina de servicio de interrupción..

Cuando el valor del comparador es igual al del temporizador, se produce una acción determinada en el pin sobre el que está actuando el comparador. Existen 4 acciones posibles que se seleccionan mediante 2 bits para cada comparador. Estos bits se encuentran en el registro TCTL1 ($1020). Las acciones posibles son: No afectar al pin correspondiente, poner el pin a ’1’, poner el pin a ’0’ y cambiar de estado el pin.

La salida hardware esta activa cuando al activarse el comparador se produce una acción sobre el pin de salida asociado. En caso contrario se dice que la salida hardware esta inhibida. En este sentido el comparador 1 es diferente al resto puesto que él puede actuar sobre los cinco pines de salida simúltaneamente. Los otros sólo actuan sobre su pin asociado.

Por defecto (Al hacer RESET) los comparadores se inicializan con los valores $FFFF y no realizan ninguna acción sobre los pines (Están desconectados). Con esta configuración, los bits 6,5,4 y 3 del puerto A se pueden utilizar como salida normal (El puerto A comparte estos pines con el comparador). Sin embargo, cuando los comparadores están configurados para salida hardware no es posible cambiar el estado del pin escribiendo un valor en el bit adecuado del puerto A.

La acción "Cambiar estado del pin" es muy interesante. Cada vez que el temporizador se iguala al valor del comparador, el estado del pin se cambia de valor lógico, es decir, si estaba a "1" pasa a "0" y viceversa. De esta manera se producen señales cuadradas de cualquier frecuencia muy fácilmente.

4.7.2. Los registros de los comparadores

En las siguientes figuras se muestran los registros de configuración y control de los comparadores.

TMSK1
DIRECCION $1022

7							0
OC1I	OC2I	OC3I	OC4I	OC5I	IC1I	IC2I	IC3I

TCTL1
DIRECCION $1020

OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5

OC1I-OC5I = Permiso de interrupción de cada comparador
IC1I-IC3I = Permiso de interrupción de cada capturador

TFLG1
DIRECCION $1023

OC1F	OC2F	OC3F	OC4F	OC5F	IC1F	IC2F	IC3F

OC1I-OC5I = Flag de interrupción de cada comparador
IC1I-IC3I = Flag de interrupción de cada capturador

OMx	OLx	Acción a realizar cuando se activa el comparador
0	0	No hay salida hardware. Los pines se utilizan a través del puerto A
0	1	Cambiar el estado del pin correspondiente
1	0	Poner a cero el pin correspondiente
1	1	Poner a uno el pin correspondiente

4.7.3. Los comparadores y el puerto A

Los 5 comparadores comparten pines con el puerto A. Si está activada la salida hardware de un comparador, automáticamente se desactiva el bit correspondiente del puerto A, y aunque se escriba en él, no tendrá efecto sobre el pin de salida. Para que la salida hardware esté desactivada se tienen que poner a cero los bits OMx y OLx del comparador, situados en el registro **TCTL1.**

7							0
PA7 PAI OC1	PA6 OC2 OC1	PA5 OC3 OC1	PA4 OC4 OC1	PA3 OC5 OC1	PA2 IC1	PA1 IC2	PA0 IC3

Direccion **$***1000*

Figura 32: Recursos que utilizan cada bit del puerto A

PA0−PA2	Bits de entrada
PA3−PA6	Bits de salida
PA7	Bidireccional
OCx	Salidas de los comparadores
ICx	Capturadores de entrada
PAI	Acumulador de oulsos

4.7.4. El comparador 1

El comparador 1 permite controlar 5 pines simultáneamente. Cada vez que el registro del comparador 1 es igual que el del temporizador se produce una salida hardware por los 5 pines más significativos del puerto A.

Mediante el registro **OC1M** ($100c) se configuran los pines que van a ser afectados por el comparador 1. Un 1 indica que el pin va a ser utilizado y un 0 indica que no. Los 5 bits más significativos del registro **OC1M** se corresponden con los pines de los 5 bits más significativos del puerto A. Cada vez que el temporizador principal se iguale al comparador 1, se manda por los pines activados un cierto valor que se encuentra en el registro **OC1D** ($100D). Los 5 bits más significativos de este registro coinciden con los 5 bits más significativos del puerto A.

Si los 5 pines están activos (Valor $F8 en **OC1M**) y el registro **OC1D** contiene el valor $A8 (10101000 en binario), al activarse el comparador se envían los bits 10101 por los pines de los bits 7,6,5,4 y 3 del puerto A. El bit 7 del puerto A sólo se puede utilizar si previamente se configura para funcionar en modo salida (Registro **PACTL**).

El utilizar el comparador 1 para controlar hasta 5 pines es muy útil para manejar motores paso a paso. El comparador 1 además se puede utilizar junto con los otros para controlar a la vez un pin. Esto permite generar pulsos de una duración mínima de 500nseg.

4.7.5. Ejemplos de manejo de los comparadores

−EJEMPLO 1: Utilización de un comparador para realizar pausas.

LX

En el siguiente ejemplo se utiliza el comparador 5 para realizar una pausa. La pausa se realiza mediante espera activa .Se hace parpadear el led.

```
; +-------------------------------------------------------------------------+
; | DELAY.ASM  (C) GRUPO J&J. Marzo   1997                                  |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Ejemplo de utilización del comparador 5 para realizar una pausa     |
; | múltiplo de 10ms exactos mediante espera activa.                        |
; +-------------------------------------------------------------------------+

TMSK1   EQU $22
TFLG1   EQU $23
TCTL1   EQU $20
TOC5    EQU $1E
TMSK2   EQU $24
TCNT    EQU $0E
PORTA   EQU $00

TIEMPO  EQU 20000  ; Número de tics de reloj necesarios para generar un
                   ; retraso de 10 ms. Cada tic de reloj son 500ns = 0.5microseg
                   ; 20000*0.5 = 10000microseg = 10mseg.

        ORG $0000

        LDX #$1000
        LDAA #$40     ; Encender el led.
        STAA PORTA,X

bucle   LDAA #50      ; Esperar 50 unidades de tiempo = 500mseg
        BSR delay

        LDAA PORTA,X  ; Cambiar de estado el led.
        EORA #$40
        STAA PORTA,X

        BRA bucle

;+---------------------------------------------------------------------+
;| Subrutina para de espera.                                           |
;| ENTRADAS: Acumulador A contiene el número de unidades de tiempo     |
;|    a esperar. Cada unidad de tiempo son 10ms.                       |
;+---------------------------------------------------------------------+

delay
        CMPA #0            ; ¿Queda alguna unidad de tiempo por esperar ?
        BEQ fin_delay      ; No--> terminar
        DECA               ; Si--> queda una unidad menos
        BSR delay10        ; Esperar una unidad de tiempo
        BRA delay          ; Repetir
fin_delay
        RTS

;+---------------------------------+
;| Subrutina para esperar 10ms.    |
;+---------------------------------+
delay10
        PSHA
        LDD TCNT,X             ; Escribir en comparador 5 el valor del
        ADDD #TIEMPO           ; temporizador principal más el número de
        STD TOC5,X             ; tics de reloj necesarios para una pausa
                               ; de 10ms.
        BSET TFLG1,X $08       ; Poner a cero flag del comparador 5
oc5     BRCLR TFLG1,X $08 oc5  ; Esperar a que se activa flag del comparador
        PULA
        RTS                    ; Terminar

        END
```

La subrutina delay10 realiza exactamente una pausa de 10ms utilizando el comparador 5. En este ejemplo no se han utilizado interrupciones. El funcionamiento de la rutina delay10 es el siguiente: primero se lee el valor de temporizador. A este valor se le añade el número correspondiente a un retraso de 10ms. El resultado se introduce en el comparador 5. El temporizador seguirá 'contando'. Cuando se iguale al valor que se había depositado en el comparador querrá decir que ha transcurrido el intervalo de tiempo que se quería. El flag del comparador 5 se activa. La subrutina delay10 espera a que este flag se active. Cuando se active este flag habrán transcurrido exactamente 10ms.

¿Qué valor hay que sumar al valor del temporizador para que se produzca una pausa de 10ms?. El temporizador se incrementa cada 500ns = 0.5 microsegundos. Se dice que un tic del temporizador tiene un periodo de 500ns. Para conseguir un retraso de 10ms se necesitará que transcurran 20000 tics (20000 * 0.5 microseg = 10000 microseg = 10 mseg). Con un valor de 2000 tics se obtendría un retraso de 1ms.

La subrutina ’delay’ simplemente llama a ’delay10’ un número de veces especificadas a través del acumulador A. Así, si se llama a ’delay’ con un valor A=50, se ejecutará 50 veces la subrutina ’delay10’ por lo que se logrará un retraso de 50*10mseg= 500mseg = 0.5 segundos.

−EJEMPLO 2: Temporización mediante interrupciones.

En el siguiente programa, el bucle principal enciende un led, activa las interrupciones del comparador 4 y ejecuta un bucle infinito. Mediante interrupciones se temporizan 2 segundos y se apaga el led. Lo importante de este programa es ver que el bucle principal podría estar realizando cualquier cosa. En este ejemplo simplemente ejecuta un bucle infinito, pero podría estar realizando cálculos mientras el led está siendo temporizado por las interrupciones del comparador 4.

```
; +-------------------------------------------------------------------------+
; | TEMPO.ASM  (C) GRUPO J&J. Marzo   1997                                  |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Ejemplo de utilización del comparador 4 para realizar temporiza-     |
; | ciones mediante interrupciones. El programa principal enciende el led,  |
; | activa la temporizaron y ejecuta un bucle infinito. Al cabo de 2        |
; | segundos el led se apagará.                                             |
; +-------------------------------------------------------------------------+

TMSK1   EQU $22
TFLG1   EQU $23
TCTL1   EQU $20
TOC4    EQU $1C
TMSK2   EQU $24
TCNT    EQU $0E
PORTA   EQU $00

TIEMPO  EQU 20000  ; Número de tics de reloj necesarios para generar un
                   ; retraso de 10 ms. Cada tic de reloj son 500ns = 0.5microseg
                   ; 20000*0.5 = 10000microseg = 10mseg.

        ORG $0000

        BRA inicio

cuenta  DB 0

inicio  LDX #$1000

        LDAA #$10
        STAA TMSK1,X  ; Permitir la interrupción del comparador 4

        LDAA #$40     ; Encender el led.
        STAA PORTA,X

        LDAA #200     ; Mantener el led encendido durante 2 segundos
        STAA cuenta

        CLI           ; Activar las interrupciones

inf     BRA inf

;+-----------------------------------------------------------------+
;| Rutina de servicio de interrupción del comparador 4             |
;+-----------------------------------------------------------------+
oc4
        BSET TFLG1,X $10        ; Poner a cero flag del comparador 4

        LDD TCNT,X
        ADDD #TIEMPO     ; Esperar 10 mseg
        STD TOC4,X

        LDAA cuenta
        CMPA #0          ; ¿Ha llegado la cuenta a 0?
        BEQ fin          ; Si--> Apagar el led
        dec cuenta
        RTI
fin
        CLRA
        STAA PORTA,X     ; Apagar el led
        BCLR TMSK1,X $10 ; Desactivar interrupción comparador 4
        RTI

        ORG $00D6
        JMP oc4

        END
```

−EJEMPLO 3: Generación de señales cuadradas mediante espera activa y salida hardware.

En este ejemplo se utiliza la salida hardware para cambiar el estado del pin asociado al bit PA6 del puerto A. Cada vez que se produzca una comparación el estado de este pin cambiará. De esta manera es muy fácil generar ondas cuadradas de muy diversas frecuencias de una forma muy fácil.

```
; +-------------------------------------------------------------------------+
; | ONDCUAD.ASM  (C) GRUPO J&J. Marzo   1997                                |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Ejemplo de utilización del comparador 2 con salida hardware para    |
; | generar una señal cuadrada de una frecuencia determinada.               |
; +-------------------------------------------------------------------------+
TMSK1   EQU $22
TFLG1   EQU $23
TCTL1   EQU $20
TOC2    EQU $18
TMSK2   EQU $24
TCNT    EQU $0E
PORTA   EQU $00

SEMIPERIOD EQU 60000

        ORG $0000

        LDX #$1000
        LDAA #$40     ; Activar salida hardware del comparador 2 para que
        STAA TCTL1,X  ; cambie el estado del pin en cada comparación
        LDAA #$40
        STAA PORTA,X  ; Encender el led

bucle
        LDD TCNT,X              ; Actualizar comparador 2
        ADDD #SEMIPERIOD
        STD TOC2,X

        BSET TFLG1,X $40        ; Poner a cero flag del comparador 2
oc2     BRCLR TFLG1,X $40 oc2   ; Esperar a que se activa flag del comparador
        BRA bucle
        END
```

En este ejemplo se genera una señal de frecuencia 1000000/(0.5*60000) = 33.33Hz. Cambiando el valor de la constante SEMIPERIOD se obtienen señales cuadradas de diferentes frecuencias. La constante SEMIPERIOD indica la mitad del periodo de la señal que se quiere generar. Puesto que cada tic del temporizador son 0.5microsegundos, el valor 2*SEMIPERIOD*0.5microseg da el valor del periodo de la señal en microsegundos que se genera. Con un valor de SEMIPERIOD=2500 la señal tiene una frecuencia de 400Hz.

Esta señal se puede usar para excitar un altavoz y alternando diferentes señales se genera música.

−EJEMPLO 4: Generación de señales cuadradas mediante interrupciones y salida hardware.

El programa hace lo mismo que el ejemplo anterior, pero esta vez la señal se genera mediante interrupciones. El bucle principal es un bucle infinito. Al generar la señal por interrupciones en el bucle principal se podría estar realizando cualquier operación mientras la señal se seguiría generando 'ella sola'.

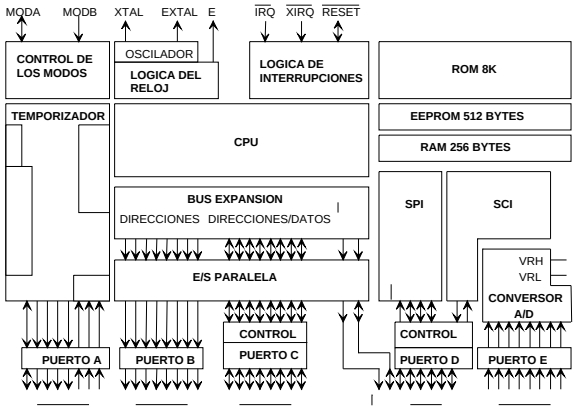
```
; +-------------------------------------------------------------------------+
; | ONDCUAD2.ASM  (C) GRUPO J&J. Marzo   1997                               |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Ejemplo de utilización del comparador 2 con salida hardware para    |
; | generar señales cuadradas de una frecuencia determinada mediante        |
; | interrupciones.                                                         |
; +-------------------------------------------------------------------------+

TMSK1   EQU $22
TFLG1   EQU $23
TCTL1   EQU $20
TOC2    EQU $18
TMSK2   EQU $24
TCNT    EQU $0E
PORTA   EQU $00
```

```
TIEMPO  EQU 60000

        ORG $0000

        LDX #$1000

        LDAA #$40     ; Activar salida hardware del comparador 2
        STAA TCTL1,X  ; Cambiar el estado del pin con cada comparación

        LDAA #$40
        STAA TMSK1,X  ; Permitir la interrupción del comparador 2

        LDAA #$40     ; Encender el led.
        STAA PORTA,X
        CLI           ; Activar las interrupciones

inf     BRA inf

;+--------------------------------------------------------------------+
;¦ Rutina de servicio de interrupción del comparador 2                ¦
;+--------------------------------------------------------------------+
oc2
        BSET TFLG1,X $40        ; Poner a cero flag del comparador 2

        LDD TCNT,X
        ADDD #TIEMPO            ; Actualizar comparador 2
        STD TOC2,X
        RTI

        ORG $00DC
        JMP oc2
        END
```

Este ejemplo podría ser la base de un órgano electrónico. Si se conecta un altavoz a la salida de comparador se podrán escuchar los tonos generados. Puesto que se generan por interrupciones, el bucle principal podría estar pendiente de las teclas que se pulsen. Según la tecla que se apriete, se modifica la frecuencia de la señal que se está generando por interrupciones.

4.8. CAPTURADORES DE ENTRADA

4.8.1. Introducción



Los capturadores de entrada permiten detectar el instante exacto en el que se ha producido un suceso. Cada capturador está formado por un registro de 16 bits, lógica de detección de flancos y lógica de generación de interrupciones.

Cada vez que por la entrada del capturador se detecta un flanco (de subida o bajada según cómo se configure) se almacena en el registro de 16 bits el valor del temporizador principal y se produce una interrupción. Existen 3 capturadores. sus registros se encuentran en las direcciones:

CAPTURADOR	DIRECCIONES	
	Byte alto	**Byte Bajo**
1	$1010	$1011
2	$1012	$1013
3	$1014	$1014

El registro de máscara de interrupciones se encuentra en **TMSK1** ($1022) y los bits 0,1 y 2 corresponden respectivamente a las interrupciones de los capturadores 3,2 y 1. En el registro de interrupciones **TFLG1** ($1023) se encuentran los bits que se activan cuando se ha producido alguna captura en alguno de los capturadores. Los bits 0,1 y 2 corresponden a los capturadores 3,2 y 1, igual que en el registro de máscara de interrupciones.

La captura se puede programar mediante 2 bits para cada capturador. Estos bits se encuentran en el registro **TCTL2** ($1021)

	7							0
Direccion **$***1000*	PA7 PAI OC1	PA6 OC2 OC1	PA5 OC3 OC1	PA4 OC4 OC1	PA3 OC5 OC1	PA2 IC1	PA1 IC2	PA0 IC3

Figura 33: Recursos que utilizan cada bit del puerto A

PA0−PA2	Bits de entrada
PA3−PA6	Bits de salida
PA7	Bidireccional
OCx	Salidas de los comparadores
ICx	Capturadores de entrada
PAI	Acumulador de oulsos

4.8.2. Registros de los capturadores

TMSK1
DIRECCION $1022
DIRECCION $1023

OC1I OC1F	OC2I OC2F	OC3I OC3F	OC4I OC4F	OC5I OC5F	IC1I IC1F	IC2I IC2F	IC3I IC3F

OC1I-OC5I = Permiso de interrupción de cada comparador
OC1I-OC5I = Flag de interrupción de cada comparador
IC1I-IC3I = Permiso de interrupción de cada capturador
IC1I-IC3I = Flag de interrupción de cada capturador

TCTL2
DIRECCION 1021

7							0
0	0	EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A

4.8.3. aplicaciones de los capturadores

EDGxB	EDGxA	ESTADO CAPTURADOR
0	0	Capturador deshabilidato
0	1	Captura de flancos de subida
1	0	Captura de flancos de bajada
1	1	Captura de flancos de subida y bajada

- Medir período y frecuencia de una señal cuadrada:

Capturando los instantes en que se producen dos flancos de subida consecutivos y realizando la resta se obtiene el período de la señal. Por supuesto también se puede hacer utilizando dos flancos de bajada consecutivos. Se configura el capturador para capturar flancos de subida por ejemplo. Se activan las interrupciones. Al llegar el primer flanco se almacena el valor del temporizador principal, se produce una interrupción y nuestra rutina de servicio salva en memoria este valor. Al llegar el segundo flanco de subida, se guarda el estado del temporizador principal y se ejecuta la rutina de servicio de interrupción que calcula la diferencia de tiempos y halla el período de la señal. Naturalmente sólo hay un rango de frecuencias que pueden ser detectadas. La máxima frecuencia teórica que se puede detectar depende de la resolución del reloj del temporizador principal. La mínima frecuencia depende del tiempo que tarda el temporizador en "dar la vuelta". Utilizando una señal de reloj de 8MHZ, la frecuencia máxima son 2MHZ y la mínima 2 Hz.

- Medir anchura de pulsos:

Midiendo los instantes de un flanco de subida y otro de bajada consecutivos se obtiene la anchura del pulso. Midiendo la anchura de los pulsos se puede detectar una señal digital no periódica, almacenarla en memoria y luego reproducirla exactamente igual.

- Contador de pulsos:

También es posible utilizar los capturadores como entradas de interrupción que se pueden configurar para activarse con flancos de subida o bajada. Esto es muy útil para contar pulsos, por ejemplo los pulsos provenientes de un encoder.

4.8.4. Ejemplos de utilización de los capturadores

−EJEMPLO 1**:** Interrupción en flanco de bajada.

En este ejemplo se utiliza el capturador para ’capturar’ flancos de bajada. Cada vez que se obtiene un flanco de bajada por el bit PA0 se cambiará el estado del led. Se hace mediante interrupciones.

```
; +---------------------------------------------------------------------+
; | CAP.ASM       (C) GRUPO J&J. Marzo   1997                           |
; |---------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.          |
; |  Este programa se debe cargar en la RAM interna del 6811.           |
; |                                                                     |
; |    Ejemplo de utilización del capturador de entrada 3 como entrada  |
; | de interrupciones configurada en flanco de bajada. Cada vez que se  |
; | obtenga un flanco de bajada por el pin PA0 se cambiará el estado del|
; | led                                                                 |
; +---------------------------------------------------------------------+
TMSK1   EQU $22
TFLG1   EQU $23
TCTL2   EQU $21
PORTA   EQU $00

        ORG $0000

        LDX #$1000
        LDAA #$02     ; Configurar capturador 3 para flanco de bajada
        STAA TCTL2,X  ;
        LDAA #$01
        STAA TMSK1,X  ; Permitir la interrupción del capturador 3
```

```
        CLI
inf     BRA inf

;+---------------------------------------------------------------+
;¦ Rutina de servicio de interrupción del capturador 3           ¦
;+---------------------------------------------------------------+
ic3     BSET TFLG1,X $01        ; Poner a cero flag del capturador 3
        LDAA PORTA,X
        EORA #$40               ; Cambiar led de estado
        STAA PORTA,X
        RTI

        ORG $00E2
        JMP ic3
        END
```

−EJEMPLO 2: Utilización del capturador 2 para contar flancos de bajada.

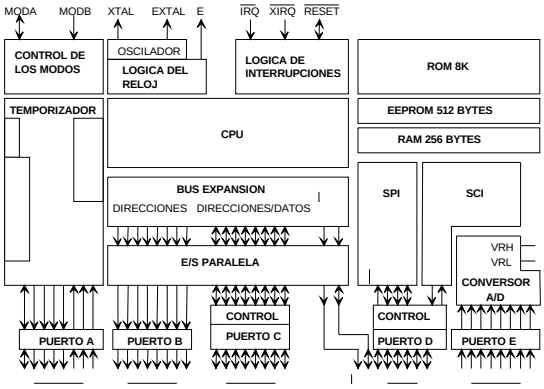
Cada vez que se reciban 5 flancos de bajada por el pin PA1 el led cambiará de estado.

```
; +-------------------------------------------------------------------------+
; ¦ CAP2.ASM       (C) GRUPO J&J. Marzo   1997                               ¦
; ¦-------------------------------------------------------------------------¦
; ¦  Programa ejemplo para ser ejecutado en la tarjeta CT6811.               ¦
; ¦  Este programa se debe cargar en la RAM interna del 6811.                ¦
; ¦                                                                          ¦
; ¦    Ejemplo de utilización del capturador de entrada 2 como entrada       ¦
; ¦ de interrupciones configurada en flanco de bajada. Cada vez que se       ¦
; ¦ obtienen 5 flancos de bajada se cambia el estado del led.                ¦
; +-------------------------------------------------------------------------+

TMSK1   EQU $22
TFLG1   EQU $23
TCTL2   EQU $21
PORTA   EQU $00

        ORG $0000

        LDX #$1000
        LDAA #$08     ; Configurar capturador 2 para flanco de bajada
        STAA TCTL2,X  ;
        LDAA #$02
        STAA TMSK1,X  ; Permitir la interrupción del capturador 2
        CLI
inf     BRA inf

;+---------------------------------------------------------------+
;¦ Rutina de servicio de interrupción del capturador 2           ¦
;+---------------------------------------------------------------+
nfb     DB 5            ; Número de flancos de bajada

ic2     BSET TFLG1,X $02        ; Poner a cero flag del capturador 2
        LDAA nfb
        CMPA #1         ; ¿Han ocurrido todos los flancos de bajada?
        BEQ actuar      ; Si --> Actuar
        DEC nfb         ; No --> salir
        RTI
actuar
        LDAA PORTA,X
        EORA #$40               ; Cambiar led de estado
        STAA PORTA,X
        LDAA #$05
        STAA nfb
        RTI

        ORG $00E5
        JMP ic2

        END
```

4.9. ACUMULADOR DE PULSOS

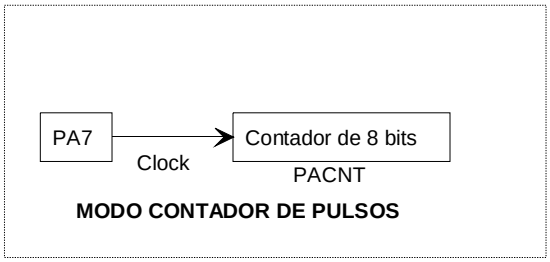
4.9.1. Introducción

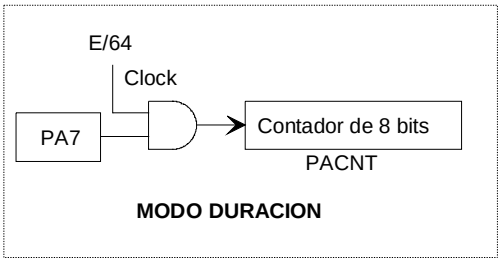
El acumulador de pulsos está formado por un contador de 8 bits, que puede ser leído/escrito en cualquier momento. El contador funciona en dos modos diferentes: **Modo cuenta de pulsos y modo duración**. En el primer modo la señal de reloj que incrementa el contador se toma del exterior de MCU, a través del pin asociado al bit 7 del puerto A. Cada vez que se detecta un flanco activo (de subida o bajada según se configure) se incrementa el contador en una unidad y se produce una interrupción (Si está permitida). Cuando el contador llega a $FF se produce una interrupción de overflow que permite extender el rango del contador y vuelve a comenzar desde cero.



El segundo modo funciona de manera diferente. La señal externa no pasa directamente a la entrada de reloj del contador sino que lo hace a través de una puerta AND con otra señal secundaria. Esta última es una onda cuadrada de frecuencia E/64, siendo E la frecuencia interna del MCU (que no es la del cristal externo). En el caso de tener un cristal de 8MHZ, la señal E tendrá una frecuencia de 2MHZ. Para que el contador se incremente tiene que estar la señal externa activa y la secundaria también. Este modo se denomina duración porque permite medir la duración que permanece activa la señal externa. Mientras esta este activa, la onda cuadrada aparecerá con una frecuencia E/64, incrementándose el contador en cada pulso.

La señal externa se introduce por el pin correspondiente al bit 7 del puerto A, que se puede configurar para ser entrada o salida. Para funcionar con el acumulador de pulsos es preciso que se configure como entrada.





4.9.2. Registros de control asociados al acumulador de pulsos

Los registros de control asociados con el acumulador son los siguientes:

PACNT
DIRECCION $1027

7	6	5	4	3	2	1	0

Contador de 8 bits. Se puede leer y escribir en cualquier momento

PACTL
DIRECCION $1026

7							0
DDRA7	PAEN	PAMOD	PEDGE	0	0	RTR1	RTR0

DDRA7 = Configuración de la dirección del bit 7 del puerto A

0 = Entrada
1 = Salida

PAEN = Bit de activación del acumulador de pulsos.

0 = Desactivado
1=Activado

PAMOD =Selección modo de funcionamiento del acumulador

0 = Modo cuenta de pulsos
1= Modo duración

PEDGE = Selección del flanco activo

0 = Flanco de bajada
1= Flanco de subida

RTR1,RTR2 = Determinan el periodo de tiempo entre cada interrupción de teimpo real

	7							0
TFLG2 *DIRECCION $1025*	TOF	RTIF	PAOVF	PAIF				

TOF = Flag de interrupción de overflow del temporizador

RTIF = Flag de interrupción de tiempo real

PAOVF= Flag de interrupción de overflow del acumulador de pulsos

PAIF = Flag de interrupción del acumulador de pulsos

4.9.3. Ejemplos de manejo del acumulador de pulsos

−EJEMPLO 1: Cuenta de 5 flancos de bajada.

En este ejemplo cada 5 flancos de bajada recibidos por el pin PA7 se cambia el estado del led.

```
; +-------------------------------------------------------------------------+
; | ACUM.ASM       (C) GRUPO J&J. Marzo   1997                              |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Ejemplo de utilización de acumulador de pulsos. Cada 5 flancos de    |
; | bajada en el pin PA7 se cambia el estado del led.                       |
; +-------------------------------------------------------------------------+

PACNT   EQU $27
PACTL   EQU $26
TMSK2   EQU $24
TFLG2   EQU $25
PORTA   EQU $00

        ORG $0000

        LDX #$1000

        CLR PACNT,X     ; Poner a cero el acumulador de pulsos

        LDAA #$40       ; Activar acumulador de pulsos
        STAA PACTL,X    ; Modo cuenta de pulsos. Flanco de bajada

        LDAA #$10       ; Permitir la interrupción del acumulador
        STAA TMSK2,X

        CLI
inf     BRA inf

;+-----------------------------------------------------------------+
;| Rutina de servicio de interrupción de acumulador de pulsos      |
;+-----------------------------------------------------------------+
ac
        BSET TFLG2,X $10        ; Poner a cero flag del acumulador

        LDAA PACNT,X            ; Leer el acumulador
        CMPA #$05               ; ¿Se han producido 5 flancos de bajada?
        BEQ actuar              ; Si actuar
        RTI                     ; No --> Terminar
actuar
        LDAA PORTA,X
        EORA #$40               ; Cambiar led de estado
        STAA PORTA,X
        CLR PACNT,X             ; Poner de nuevo a cero el acumulador de pulsos
        RTI

        ORG $00CA
        JMP ac

        END
```

−EJEMPLO 2: Interrupción de overflow del acumulador de pulsos.

En este ejemplo se cambia el estado del led cada vez que se produce un desbordamiento en el acumulador de pulsos. Los desbordamientos se producen cada 256 pulsos. Para hacer que ocurran sólo cada 5 pulsos se inicializa el acumulador de pulsos con el valor $FB.

```
; +-------------------------------------------------------------------------+
; | ACUM2.ASM       (C) GRUPO J&J. Marzo   1997                             |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Ejemplo de utilización de la interrupción de overflow del acumula-   |
; | dor de pulsos. Cada vez que se produce overflow se cambia el estado     |
; | del led. El overflow se produce cada 256 flancos de bajada. Para faci-  |
; | litar la prueba del programa, cada vez que se produce overflow se       |
; | coloca el valor $FB en el acumulador de pulsos para que se produzca     |
; | el overflow cada 5 pulsos.                                              |
; +-------------------------------------------------------------------------+

PACNT   EQU $27
PACTL   EQU $26
TMSK2   EQU $24
TFLG2   EQU $25
PORTA   EQU $00

        ORG $0000

        LDX #$1000

        LDAA #$FB       ; Inicializar acumulador de pulsos con $FA
        STAA PACNT,X

        LDAA #$40       ; Activar acumulador de pulsos
        STAA PACTL,X    ; Modo cuenta de pulsos. Flanco de bajada

        LDAA #$20       ; Permitir la interrupción de overflow del acumulador
        STAA TMSK2,X

        CLI
inf     BRA inf

;+-----------------------------------------------------------------+
;| Rutina de servicio de interrupción del overflow del acumulador  |
;+-----------------------------------------------------------------+
ovfac
        BSET TFLG2,X $20        ; Poner a cero flag del acumulador

        LDAA PORTA,X
        EORA #$40               ; Cambiar led de estado
        STAA PORTA,X

        LDAA #$FB
        STAA PACNT,X            ; Inicializar contador con valor $FA
        RTI

        ORG $00CD
        JMP ovfac

        END
```

4.10. LA INTERRUPCIÓN EXTERNA IRQ

4.10.1. Introducción

La entrada de interrupción IRQ se trata de una interrupción enmascarable. Puede ser activa a nivel bajo o en flanco de bajada, según cómo se configure en el registro **OPTION** ($1039). Si el bit 5 del registro **OPTION** está a '0', la IRQ queda configurada a nivel bajo. Si está a '1' se configura para flanco de bajada.

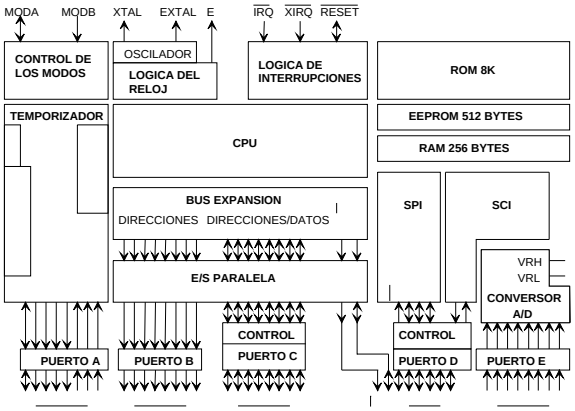
4.10.2. Ejemplo de utilización de la interrupción IRQ

Este ejemplo está pensado para ser ejecutado en la tarjeta CT6811. El pulsador se debe configurar para funcionar como entrada de IRQ. Esto se consigue situando el jumper JP4 en la posición de más a la izquierda. De esta forma cada vez que se apriete el pulsador se provoca una interrupción IRQ. El programa cambia de estado el led cada vez que recibe un flanco. En la rutina de interrupción se ha incluido un pequeño retardo para eliminar los 'rebotes' del pulsador.

```
; +-------------------------------------------------------------------------+
; | IRQ.ASM        (C) GRUPO J&J. Marzo   1997                              |
; |-------------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.              |
; |  Este programa se debe cargar en la RAM interna del 6811.               |
; |                                                                         |
; |    Ejemplo de utilización de la interrupción IRQ. Cada vez que se re-   |
; | cibe un flanco de bajada se cambia el estado del led.                   |
; +-------------------------------------------------------------------------+

OPTION  EQU $39
PORTA   EQU $00

        ORG $0000

        LDX #$1000

        BSET OPTION,X $20  ; Interrupción IRQ activa con flanco bajada
        CLI                ; Permitir las interrupciones

inf     BRA inf

;+-----------------------------------------------------------------+
;| Rutina de servicio de la interrupción IRQ.                      |
;+-----------------------------------------------------------------+
irq
        LDAA PORTA,X
        EORA #$40               ; Cambiar led de estado
        STAA PORTA,X

        LDY #$FFFF              ; Realizar una pausa anti-rebotes
wait    DEY
        CPY #0
        BNE wait

        RTI

        ORG $00EE
        JMP irq

        END
```

4.11. CONVERSOR ANALÓGICO−DIGITAL (A/D)

4.11.1. Introducción



El microcontrolador 68HC11 tiene una serie de conversores analógico digitales que son bastante útiles y que le han dotado de su gran popularidad. En este capítulo se va a indicar cómo usarlos , describiendo los distintos registros de control, y finalmente se propone un ejemplo.

Una característica del 68HC11 es que proporciona dos entradas (VRL, VRH) de referencia para las conversiones. VRH se corresponde con el valor máximo y VRL con el valor mínimo. Las tensiones de referencia deben ser fijadas por el usuario y estar en el rango (0v − 6v).

Cuidado con los límites pues aunque el *microcontrolador* permite superar Vcc es sumamente crítico con las tensiones negati ecir, evitar introducir tensiones que estén por debajo de GND, tanto en el nivel de referencia VRL, como en la entrada del conver

La placa CT6811 proporciona dos jumpers que permiten establecer los niveles de referencia a GND y VCC de tal manera que el usuario no tiene que preocuparse por realizar los circuitos de referencia salvo que quiera tener otros niveles. El 68hc11 con formato de 48 pines sólo dispone de cuatro entradas analógico digitales, mientras que el de 52 pines dispone de 8.

El tiempo que tarda en realizarse una conversión se corresponde con 32 ciclos de reloj. Dicho reloj es tomado como el E cuando la frecuencia de éste sea mayor de 750K o es tomado de una señal interna generada por un circuito RC cuando la frecuencia de E sea menor de 750khz. El oscilador RC[1] se activa situando a nivel alto el bit CSEL del registro OPTION.

Si se quiere información detallada sobre cómo se realiza la conversión interna y cuales son los circuitos equivalentes de entrada mirar el libro de referencia de motorola (capítulo 12).

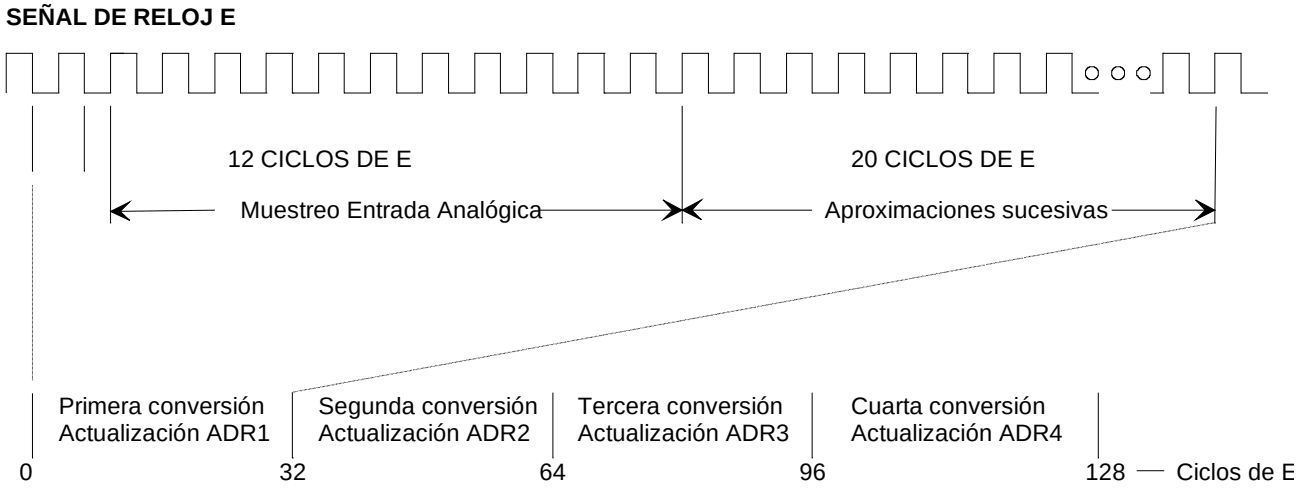


Figura 34: Secuencia de conversión

[1] El oscilador RC también se usará en ciertas condiciones para programar la EEPROM interna.

4.11.2. Registros del conversor A/D

− OPTION ($1039)

En este registro el bit más significativo es el **ADPU**. Indica si se activa o no el conversor. Al ponerlo a nivel alto el MCU entiende que se va a usar el conversor y empieza a cargar los condensadores utilizados para la conversión. Por ello hay que esperar un tiempo pequeño hasta poder utilizar correctamente el conversor[2]. Este retardo sólo ocurre al activar el conversor, luego únicamente afecta a la primera conversión.

En este registro hay otro bit (el número seis **CSEL**) que se encarga de seleccionar el reloj del conversor. Cuando la frecuencia del bus (reloj E) sea menor de 750Khz es recomendable utilizar otra frecuencia para la conversión. Esto se indica poniendo a nivel alto el bit **CSEL**. Al seleccionar esta opción el MCU genera automáticamente una propia desde un oscilador RC interno.(Esa frecuencia propia suele ser de 2Mhz).

Si el bit **CSEL** se deja a nivel bajo (0) se trabaja con la frecuencia del bus. Este modo tiene dos ventajas sobre el anterior. La primera de ellas se refiere al ruido en la conversión, que es menor que con el oscilador interno. La segunda de ellas se refiere a la velocidad. Al conectar el circuito RC interno, el MCU pierde un cierto tiempo en la sincronización de las lecturas y escrituras de la conversión.

7	6	5	4	3	2	1	0
ADPU	CSEL	IRQE	DLY	CME	0	CR1	CR0

Registro Option ($1039)

− ADCTL ($1030):

Al contrario que el registro anterior este es exclusivo del sistema conversor. De sus ocho bits sólo siete son válidos pues el bit número seis no tiene ninguna función.

7	6	5	4	3	2	1	0
CCF	0	SCAN	MULT	CD	CC	CB	CA

Registro ADCTL ($1030)

CCF: Bandera de conversión completa.

Este es un bit de sólo lectura. Cuando se ha producido una conversión completa los valores de esta se guardan en una serie de registros. Cuando los datos han sido guardados correctamente en los registros, este bit se pone a uno indicando una conversión completa. Para ponerlo a cero hay que escribir sobre el registro **ADCTL**, y una vez hecho esto empieza inmediatamente otra conversión.

Cuando el conversor esta funcionando en modo continuo, la puesta a nivel bajo de este bit hace que se abandone la conversión que se realiza en ese momento para empezar otra nueva. Por el contrario, en modo discreto, será necesario poner a nivel bajo el bit para que se realice otra conversión.

SCAN: Control de conversión continua.

Cuando este bit está a nivel bajo, las cuatro conversiones pedidas se realizan una sola vez. Los resultados se graban en los registros y hasta que no se indique, no vuelve a hacer otra conversión. (Se indica poniendo a nivel bajo el bit CCF).

Cuando este bit está a uno las conversiones se realizan continuamente. Los registros son actualizados cada vez que llegan los nuevos datos. (Es el modo SCAN o continuo)

[2]En el manual de referencia de Motorola no hay información sobre dicho tiempo de espera.

MULT: Canal múltiple / Único canal.

Cuando este bit está a cero el sistema realiza cuatro conversiones consecutivas sobre el mismo canal, guardando los resultados en los registros **ADR1−ADR4**. El canal sobre el que se realiza la conversión se especifica mediante los bits CD−CA del registro **ADCTL**. (Es el modo canal único)

Cuando el bit está a nivel alto las conversiones se realizan una en cada canal y los resultados se guardan en los correspondientes registros. En esta modalidad los bits CD y CC se encargan de elegir el grupo de cuatro canales que será muestreado. Esta opción no esta disponible en el formato del *microcontrolador* de 48 pines, ya que sólo tiene cuatro canales analógicos en lugar de ocho. (Es el modo canal múltiple).

CD,CC,CB,CA: Selección de canales.

Estos bits seleccionan los canales que van a ser operativos en las conversiones. En el modo múltiple (MULT=1) los bits CD y CC son los únicos que tienen efecto. Se encargan de seleccionar el grupo de cuatro canales que se quiere muestrear. Esta opción no tiene ninguna validez en la versión del 68HC11 de 48 pines pues sólo existe el primer grupo de canales.

En el modo único canal (MULT=0) hay que seleccionar un canal de entre los disponibles.(Cuatro u ocho según las versiones). Para ello mirar la tabla que se adjunta a continuación. Dicho canal se selecciona con los bits CD,CC,CB y CA.

Como consejo práctico para leer la tabla dejar siempre a nivel bajo el bit CD, cuando esta puesto a nivel alto se entra en los modos de funcionamiento especiales del *microcontrolador*, que no interesan. (Uno de esos modos es el factory testing del cual se puede encontrar más información en el manual de referencia de Motorola).

CD	CC	CB	CA	Señal del canal	Resultado en ADRx si MULT = 1
0	0	0	0	AD0 por bit 0 puerto E	ADR1
0	0	0	1	AD1 por bit 1 puerto E	ADR2
0	0	1	0	AD2 por bit 2 puerto E	ADR3
0	0	1	1	AD3 por bit 3 puerto E	ADR4
0	1	0	0	AD4 por bit 4 puerto E *	ADR1
0	1	0	1	AD5 por bit 5 puerto E *	ADR2
0	1	1	0	AD6 por bit 6 puerto E *	ADR3
0	1	1	1	AD7 por bit 7 puerto E *	ADR4
1	0	0	0	Reservado	ADR1
1	0	0	1	Reservado	ADR2
1	0	1	0	Reservado	ADR3
1	0	1	1	Reservado	ADR4
1	1	0	0	Vref hi**	ADR1
1	1	0	1	Vref low**	ADR2
1	1	1	0	Vref hi / 2**	ADR3
1	1	1	1	test**	ADR4

* : No disponible en la versión del micro de 48 pines.

** : Selección para el 'factory testing'

− ADR4−ADR1 ($1034−$1031) :

En estos registros se guardan los resultados de ocho bits de la conversión. Los registros son de lectura única y una vez rellenados con los datos válidos se activa la bandera CCF para indicar conversión finalizada. En el modo continuo esta bandera no tiene efecto, aunque la puesta a cero de este bit interrumpiría la conversión en curso para comenzar una nueva.

7	6	5	4	3	2	1	0
bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Registro ADRx ($103x)

Los registros **ADR1**, **ADR2**, **ADR3** y **ADR4** guardan los resultados; el bit más significativo del registro coincide con el bit más significativo de la conversión. La situación en el mapa de memoria de cada registro es

respectivamente $1031, $1032, $1033, y $1034. Aunque hay ocho canales solo hay cuatro registros para guardar los datos, esto hay que tenerlo muy en cuenta, ya que no se pueden muestrear ocho canales a la vez. Esta característica hace que la diferencia entre el formato de 48 pines y el de 52 pines no sea tan grande. La lógica de control de los circuitos del sistema hace que no se produzcan interferencias entre las lecturas por parte del software usuarios y las escrituras del MCU.

4.11.3. Programa ejemplo de manejo del conversor A/D

En este apartado se muestra un programa que sirve de ejemplo para la utilización del conversor. Se supone que se ha colocado la salida variable de un potenciómetro a la entrada de uno de los conversores. El potenciómetro se conecta entre 0 y 5 voltios, de tal forma que al variar el mando la tensión de salida oscila entre estos dos valores.

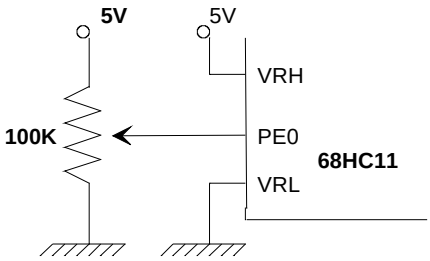


Figura 35: Conexión del potenciómetro al canal 1 del conversor A/D

El programa lee el valor que registra la salida del potenciómetro y la compara con 2.5 voltios. Si es mayor enciende el LED de la placa, en caso contrario lo apaga. Se supone que se trabaja con la CT6811 que ya lleva incorporado el LED, pero si se tiene otra tarjeta hay que ver el cambio de estado mirando la salida del pin PA6[3]. Además, como ya se ha mencionado, la CT6811 establece los niveles de referencia a VCC y GND mediante jumpers.

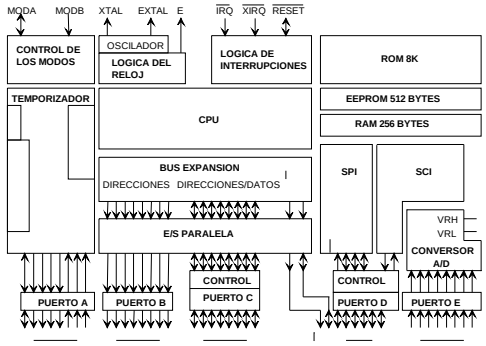
El canal analógico por el cual se introduce la señal del potenciómetro es el **PE0** (Primera entrada analógica), el resultado de la conversión se obtiene en **ADR1**. Se realiza una conversión continua de manera que cuando finaliza una conversión empieza inmediatamente la siguiente. El bucle se queda en este estado para siempre.

```
; +---------------------------------------------------------------------+
; | PAD.ASM.   (C) GRUPO J&J. MARZO 1997                                |
; |---------------------------------------------------------------------|
; |  Programa ejemplo para ser ejecutado en la tarjeta CT6811.          |
; |  Este programa se debe cargar en la RAM interna del 6811.           |
; |                                                                     |
; |    Ejemplo de utilización del canal 0 del conversor A/D. Cuando la  |
; |  tensión supera los 2.5 voltios se enciente el led de la placa.     |
; +---------------------------------------------------------------------+

;     Usaremos como entrada analógica el canal 1,  es  decir  la
; entrada PE0. Los niveles de referencia (VRH, VRL) serán (Vcc,
; GND)  respectivamente.  Es  decir los  jumpers JP1 y JP2 de  la
; CT6811 deben estar conectados.

        ORG $0000

OPTION  EQU $39
ADCTL   EQU $30
PORTA   EQU $00
ADR1    EQU $31

INICIO LDX  #$1000
       LDAA #$80
       STAA OPTION,X  ; encender el conversor
       LDAA #$20      ; configuración conversor:
       STAA ADCTL,X   ;   SCAN -> activo
                      ;   MULT -> inactivo
                      ;   ADR1 -> seleccionar primer canal

sigue  BRCLR ADCTL,X $80 sigue    ; espera a que termine conversión
          LDAA  ADR1,X            ; leer el resultado de la conversión
          CMPA  #$7F              ; comparar con la mitad (127 en decimal)
                                  ; que corresponde a 2.5v de entrada.
          BLO   menor             ; si es menor apagar el led
          LDAA  #$40              ; No--> encender el led
          STAA  PORTA,X
          BRA   sigue             ; Realizar la siguiente conversión

menor     CLRA
          STAA PORTA,X            ; Apagar el led.
          BRA  sigue              ; Realizar la siguiente conversión

          END
```

4.12. LA MEMORIA EEPROM

[3] En la placa CT6811 el LED esta conectado al pin PA6, es decir el bit6 del puerto A.

4.12.1. Introducción



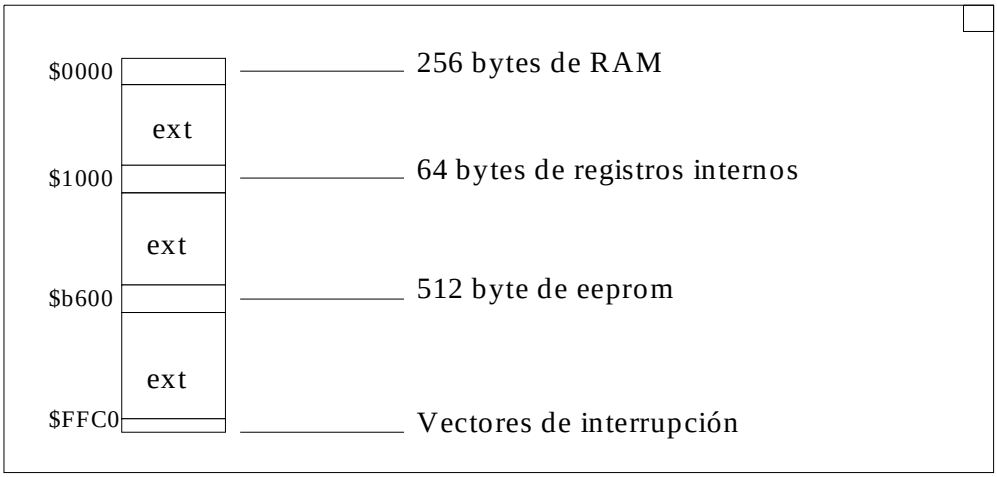
En algunos de los miembros de la familia 68HC11 se encuentra un recurso interno bastante útil a la hora de buscar un sistema autónomo barato y sencillo. La memoria EEPROM, con cual se ahorra la expansión externa de memoria.

Esto que suena tan bien tiene el inconveniente de la limitada capacidad de la EEPROM. Oscila entre 512bytes y 2kbytes según el modelo. Sin embargo para un gran número de aplicaciones esto no debe suponer ningún problema.

Número	EPROM	ROM	EEPROM	RAM	CONFIG	Comentarios
MC68HC11A8	−−−	8K	512	256	$0F	La familia construida en torno a este.
MC68HC11A1	−−−	−−−	512	256	$0D	A8 con la ROM desconectada
MC68HC11A0	−−−	−−−	−−−−	256	$0C	A8 con la ROM y EEPROM desactivada
MC68HC11E1	−−−	−−−	512	512	$0D	E9 con la ROM desactivada
MC68HC11E0	−−−	−−−	−−−	512	$0C	E9 con ROM y EEPROM desactivadas
MC68HC811E2	−−−	−−−	2K	256	$0F/$FF	A8 sin ROM y con EEPROM movible
MC68HC11E20	−−−	20K	512	768	−−−	A8 con mayor capacidad de memoria
MC68HC711E20	20K	−−−	512	768	−−−	Se puede grabar la ROM una sola vez
MC68HC11D3	−−−	4K	−−−	192	−−−	Versión de bajo coste
MC68HC11E9	−−−	12K	512	512	$0F	A8 con mayor RAM y 12K ROM
MC68HC711E9	12K	−−−	512	512	$0F	Se puede grabar la ROM una sola vez
MC68HC11F1	−−−	−−−	512	1K	$0F/$FF	Micro sin multiplexar
MC68HC11K4	−−−	24K	640	768	$FF	incorpora PWM, entre otras mejoras
MC68HC711K4	24K	−−−	640	768	$FF	Se puede grabar la ROM una sola vez
MC68HC11L6	−−−	16K	512	512	$0F	Como el E9 pero con mayor puertos
MC68HC711L6	16K	−−−	512	512	$0F	Versión del L6 con ROM grabable una vez

La EEPROM es una memoria que mantiene los datos aún desconectando la alimentación del sistema (memoria no volátil). Su ciclo de lectura es equivalente al de la ROM interna pero en escritura es más lenta. Es como una EPROM con la salvedad de que se borra eléctricamente; de esta forma se facilita su uso.

Este capítulo se refiere al 68HC11A1 que tiene 512bytes de EEPROM situados entre las direcciones $B600 − $B7FF. En otros modelos la posición de esta memoria será variable. (Por ejemplo en el 68HC11E2).



Mapa de memoria del 68HC11

La programación de la EEPROM se realiza a través del registro **PPROG**. Si no se va a usar se puede desactivar del mapa de memoria poniendo a cero el bit **EEON** del registro **CONFIG**. Más adelante se explica cómo realizar esta operación.

4.12.2. Registros de control de la EEPROM

Son dos los registros (PPROG, CONFIG) que actúan directamente sobre la EEPROM, aunque hay un tercero que interviene en algunos casos (OPTION).

− **Registro CONFIG**

El registro CONFIG se encarga, entre otras funciones, de activar o desactivar la EEPROM del mapa de memoria. Este registro es especial y está implementado con células EEPROM. En la sección 4.12.5 se estudia con más detalle. Si el bit **EEON** está a ’1’ significa que la EEPROM está activa, y si por el contrario **EEON** es cero entonces la EEPROM está inactiva. En el 68HC11A1 por defecto **EEON** está a ’1’.

7	6	5	4	3	2	1	0
0	0	0	0	NOSEC	NOCOP	ROMON	EEON

Registro CONFIG ($103F)

− **Registro PPROG**

Este registro se encarga de las operaciones realizadas sobre la EEPROM. Al hacer un reset se pone a cero y la EEPROM queda configurada para solo lectura.

7	6	5	4	3	2	1	0
ODD	EVEN	0	BYTE	ROW	ERASE	EELAT	EEPGM

Registro PPROG ($103B)

ODD : programa líneas impares (Sólo usado en modo TEST)
EVEN : programa líneas pares (Sólo usado en modo TEST)
BIT 5 : Siempre es cero
BYTE : Selección de borrado de un byte (Este bit tiene prioridad sobre el bit ROW)
’0’ = Se ha seleccionado modo de borrado de fila (ROW) o total (BULK)
’1’= Se ha seleccionado modo de borrado de un byte
ROW : Selección de borrado ROW (Borrado de una fila)
(Si el bit BYTE está a uno este bit no tiene significado)

'0'= selección de borrado total (BULK ERASE)
'1'= selección de borrado de una fila (ROW ERASE)

ERASE: Selección de la opción de borrado
'0'= Accesos a la EEPROM para lecturas o Programación
'1'= Accesos a la EEPROM para borrado

EELAT: Control sobre el latch de la EEPROM
'0'= La dirección de la EEPROM configurada para modo READ
'1'= La dirección y dato configurados para Programación / Borrado

EEPGM: Activación o desactivación del voltaje de programación de la EEPROM
'0'= Voltaje de programación ON.
'1'= Voltaje de programación OFF.

Con esta descripción de los bits del registro no se tiene una idea clara de lo que hay que hacer para programar la EEPROM. Esto no debe preocupar en este momento ya que que en los sucesivos apartados se aclara un poco más y al final se proporcionan todas las rutinas para operar sobre la EEPROM.

En versiones modernas se han introducido unas pautas de programación para evitar errores en la programación. Esta protección consiste en que los bits EELAT y EEPGM no se pueden modificar al mismo tiempo y además si cuando EELAT se está poniendo activo se realiza una escritura sobre una posición de memoria de la EEPROM esa escritura se ignora.

−Registro OPTION

En el registro OPTION se encuentra el bit CSEL que es necesario activar cuando la señal de reloj E sea menor de 1Mhz por razones que se explican en el siguiente apartado.

7	6	5	4	3	2	1	0
ADPU	CSEL	IRQE	DLY	CME	0	CR1	CR0

Registro Option ($1039)

4.12.3. Programación de la EEPROM

La programación de la EEPROM se realiza por medio del registro PPROG. Antes de ver las rutinas de control hay que estudiar el funcionamiento de la EEPROM, aunque sólo de manera superficial, lo mínimo para que se entiendan luego las rutinas de control.

En una EEPROM el estado de un byte borrado o en su caso 'no programado' tiene un valor de $FF, es decir sus bits a nivel alto. La programación permite pasar a '0' (nivel bajo) los bits que están a '1' (nivel alto), pero no al revés, si se quiere poner un bit a uno (estando previamente a cero) hay que borrar todo el byte primero. Si por el contrario se quiere pasar de '1' a '0' no hace falta borrar todo el byte primero. En resumen, siempre que hay que grabar un '1' en una celda EEPROM que previamente estaba a '0' hay que borrarla primero.

La programación requiere una sobrecarga de tensión, aunque no hay que preocuparse de esto puesto que la proporciona el *microcontrolador* de forma transparente al usuario. Lo que si que hay que tener en cuenta es que la velocidad de programación es más lenta que la velocidad de lectura, por lo tanto entre programación y programación hay que introducir unos retardos. Estos retardos son variables y dependen de la velocidad de reloj E.

Cuanto menor sea el valor de E (frecuencia de reloj) la efectividad de la sobrecarga disminuye[4]. Por esa razón el micro incorpora un pequeño oscilador RC interno que puede generar los tiempos de programación necesarios independientemente del reloj externo (E).

Motorola recomienda que cuando se trabaje con E<1Mhz activar el oscilador RC para programar la EEPROM. Para activar ese oscilador hay que actuar sobre el bit **CSEL** del registro **OPTION**[5]. En la siguiente tabla se presenta un esquema sobre los tiempos de programación.

E = 2 Mhz	Tiempo de escritura / borrado de 10ms
1 Mhz < E < 2 MHz	Tiempo de escritura / borrado entre (10ms − 20ms)

[4]La eficiencia de la sobrecarga disminuye con el incremento de tiempo necesario para borrar y grabar la posición de eeprom.

[5]Cuando se activa el bit CSEL hay que esperar 10ms hasta que se estabilice el circuito. Una vez que haya pasado ese tiempo podemos realizar operaciones sobre la EEPROM.
El oscilador RC también se utiliza en el sistema A/D cuando la E es pequeña.

E < 1 MHz	Activar oscilador interno y mantener tiempo de 20ms

Estos tiempos indican el retardo que hay que introducir entre dos programaciones consecutivas.. En la figura 36 se muestra una rutina que implementa un retardo.

```
RETARDO
            LDX #$0FFF   ; Cargar el valor $0FFF en el registro X
sigue       DEX          ; Decrementa registro X en una unidad
            BNE  sigue   ; Si no ha llegado a cero sigue en el bucle
            RTS          ; Cuando llega a cero sale de la subrutina
```

Figura 36: Implementación de un retardo software

La subrutina anterior se ha usado para simular un retardo de 10ms, por lo tanto si el usuario dispone de la CT6811, esta rutina le puede servir para establecer los retardos de programación de la EEPROM. Para obtener el valor preciso hay que analizar el número de ciclos que ejecuta la rutina y ver cuanto tiempo corresponde a un ciclo. (La CT6811 funciona con E=2Mhz por lo que su ciclo es de 500ns, DEX utiliza 3 ciclos y BNE utiliza 3. El bucle se ejecuta 4095 veces. El retardo es aproximadamente, ya que no se ha considerado el resto de instrucciones, de 12ms = 4095*6*500ns. Es decir el valor $0FFF introducido en X puede ser menor)[6].

4.12.4. Ejemplo de programación

ROM ARRAY					
B60F B60E B601	B600		B610 B611 ..... B61E	B61F	
B62F	B620		B630	B63F	
B64F	B640		B650	B65F	
B66F	B660		B670	B67F	
B68F	B680		B690	B69F	
B6AF	B6A0		B6B0	B6BF	
· ARRAY LEFT HALF ·			· ARRAY RIGHT HALF ·		
·	·		·	·	
·	·		·	·	
B78F	B780		B790	B79F	
B7AF	B7A0		B7B0	B7BF	
B7CF	B7C0		B7D0	B7DF	
B7EF B7EE B7E1	B7E0		B7F0 B7F1 ..... B7FE	B7FF	
COLUMN DECODERS AND SENSE AMPS					
			CHARGE PUMP		

Figura 37: Esquema de la estructura interna de la EEPROM

En la figura 37 se presenta un esquema de la estructura interna de la EEPROM. Este esquema es útil para entender los diferentes modos de borrado. Existen tres , borrado completo (BULK ERASE), borrado de una fila (ROW ERASE) y borrado de un byte (BYTE ERASE). Según cual sea la necesidad hay que emplear uno u otro. Por ejemplo si se desea borrar toda la EEPROM es más efectivo realizarlo en un solo paso, en vez de realizar 512 borrados de byte.

Para facilitar la comprensión se supone que nuestra señal de reloj es de 2MHz y que por tanto no es necesario activar el oscilador interno RC. Si hubiera que hacerlo (señal de reloj < 1 Mhz) habría que activar **CSEL** y esperar 10ms, luego se aplicarían las mismas rutinas que se presentan pero utilizando un retardo de 20ms.

− Lectura

Para leer la EEPROM el bit **EELAT** del registro **PPROG** debe estar a cero. Cuando este bit está a cero el resto de bits del registro dejan de tener significado o efecto. La lectura de la EEPROM se realiza como la de una memoria ROM normal.

[6]La operación DEX y DEY tienen una importante diferencia a la hora de ejecutarse. DEY utiliza cuatro ciclos de reloj mientras que DEX utiliza solo tres. Esto hay que tenerlo en mente pues el mismo bucle realizado con DEY sería más lento.

Al realizar un RESET sobre el *microcontrolador* el estado de este bit es cero, la EEPROM queda configurada para lectura.

− Programación

En la programación los bits **ROW** y **ERASE** del registro **PPROG** no se usan, estos sólo intervienen en el proceso de borrado. Se recuerda que para pasar de cero a uno hay que borrar previamente la celda.

```
      LDAA #DATO      ; Guardar dato en el acumulador A
      LDX #direccion  ; Guardar dirección en X
      JSR EPROG       ; Salto a la rutina de escritura
      ...
```

El código implementa una rutina que recibe como parámetro el dato y la dirección donde hay que escribirlo. El dato se pasa por el acumulador A y la dirección por el registro X.

```
EPROG
       LDAB   #$02
       STAB   $103B    ;poner EELAT a '1' (EPGM=0)
       STAA   0,X      ;situar el dato a grabar en la dirección indicada en RX
       LDAB   #$03
       STAB   $103B    ;poner a '1' EEPGM (EELAT=1). Se empieza a programar
       JSR    retardo  ;llamar a subrutina para esperar 10ms
       CLR    $103B    ;borrar PPROG, terminar proceso escritura y pasar modo lectura
       RTS             ;terminar subrutina y devolver el control
```

También hay que tener en cuenta cuál es la frecuencia del reloj E, ya que si esta es menor de 1Mhz el bit **CSEL** del registro **OPTION** debe ser activado ('1'), y luego esperar 10ms para empezar la programación.

− borrado de la EEPROM

En la figura 37 se muestra cómo la EEPROM se estructura en filas de 16 bytes cada una. Esta característica permite que se pueda escoger entre tres modos de borrado: borrado total (BULK ERASE), borrado de una fila (ROW ERASE) y borrado de un byte (BYTE ERASE).

BULK ERASE (BORRADO TOTAL)

Con esta opción se pueden borrar los 512 bytes de la EEPROM a la vez. Con esta operación el byte **CONFIG** que también esta realizado con celdas EEPROM no se ve afectado.

```
BTOTAL
       LDAB   #$06
       STAB   $103B    ;indicar modo de borrado total (Bulk erase)
       STAA   $b600    ;situar cualquier dato en una dirección de la EEPROM
       LDAB   #$07
       STAB   $103B    ;poner a '1' EEPGM (EELAT=1). Se empieza a borrar
       JSR    retardo  ;llamar a subrutina para esperar 10ms
       CLR   $103B     ;borrar PPROG, terminar proceso borrado y pasar modo lectura
       RTS             ;terminar subrutina y devolver el control
```

ROW ERASE (BORRADO DE UNA FILA)

Este método permite borrar una fila de la EEPROM. La utilidad radica en que una fila contiene 16 bytes por lo que se proporciona la opción de borrar bloques de bytes rápidamente, sin hacer un borrado total. El código que se propone a continuación acepta como dato la dirección del primer byte de la fila a borrar.

```
        LDX #dirección   ; guardar la dirección del primer byte de la fila en X
        JSR BFILA        ; saltar a la subrutina de borrado de fila
         ...
BFILA
       LDAB #$0E
       STAB $103B    ;indicar modo de borrado de fila (Row erase)
       STAA 0,X      ;situar cualquier dato en la dirección de la fila a borrar
       LDAB #$0F
       STAB $103B    ;poner a '1' EEPGM (EELAT=1). Se empieza a borrar
       JSR  retardo  ;llamar a subrutina para esperar 10ms
       CLR  $103B    ;borrar PPROG, terminar proceso borrado y paso modo lectura
       RTS           ;terminar subrutina y devolver el control
```

BYTE ERASE (BORRADO DE UN BYTE)

Con esta opción se borra un solo byte, sin afectar al resto. Al programa se le pasa como parámetro la dirección del byte a borrar.

```
        LDX #dirección   ; guardar la dirección del  byte a borrar en X
        JSR BBYTE        ; saltar a la subrutina de borrado de byte
         ...
BBYTE
       LDAB #$16
       STAB $103B      ;indicar modo de borrado de byte(byte erase)
       STAA 0,X        ;situar cualquier dato en la dirección del byte a borrar
       LDAB #$17
       STAB $103B      ;poner a '1' EEPGM (EELAT=1). Se empieza a borrar
       JSR  retardo    ;llamar a subrutina para esperar 10ms
       CLR  $103B      ;borrar PPROG, terminar proceso borrado y pasar modo lectura
       RTS             ;terminar subrutina y devolver el control
```

4.12.5. EL REGISTRO CONFIG

− **Introducción**

En el 68HC11 existe un registro de configuración implementado con celdas EEPROM que controla la presencia de ROM y EEPROM en el mapa de memoria y el sistema de WATCHDOG.

7	6	5	4	3	2	1	0
0	0	0	0	NOSEC	NOCOP	ROMON	EEON

Registro CONFIG($103F)

Los bits 7,6,5,4 no tienen efecto. En el modelo 68HC11Ex controlan la posición de la EEPROM

NOSEC: bit de desactivación del modo de seguridad.

Para tener esta opción hay que especificárselo a Motorola cuando programan la máscara ROM. Cuando la opción no se usa siempre se lee uno. Cuando se quiere poner el modo de seguridad en la RAM y EEPROM el bit NOSEC se debe programar a cero. De esa forma se activa el mecanismo anti−espía. En el cual se previene contra la selección del modo expandido multiplexado. Además si el micro se "resetea" estando en bootstrap y NOSEC=0 entonces la EEPROM, RAM y el registro CONFIG se borran antes de proseguir con el proceso de carga. Es decir la seguridad limita al micro a trabajar en modo single chip.

0 = Activa Seguridad
1 = Desactiva Seguridad.

NOCOP : COP Watchdog System Disable

El MCU incorpora un mecanismo contra errores que puedan aparecer al procesar instrucciones. Cuando se activa esta protección, el software se encarga de impedir que un contador interno llegue a cero. Si en algún momento ocurre esto, significa que no se están ejecutando más instrucciones y en consecuencia el MCU inicia una secuencia de reset. La frecuencia del contador del COP es ajustable, para ello hay que actuar sobre los bits **CR1** y **CR0** del registro **OPTION**. Por defecto esta protección esta quitada, el bit NOCOP = 1.Más información sobre este modo se encuentra en el manual de referencia de Motorola.

0 = Activada la protección WatchDog

XCI

1 = Desactivada la protección Watchdog

ROMON: Activación de la ROM interna

Cuando este bit se borra los 8K de ROM se desactivan, el espacio de memoria que ocupan se libera y la memoria es accesible desde el exterior. Esta opción es útil en el modo expanded.

En el modo single−chip la ROM siempre esta activada independientemente del estado de este bit.

0 = ROM desactivada.
1 = ROM activada.

Por defecto esta desactivada, ROMON a cero.

EEON : Activación de la EEPROM interna.

Cuando esta a cero, los 512bytes de EEPROM están desactivados, y la memoria es accesible desde el exterior. Por defecto, en los *microcontroladores* que tienen EEPROM interna, esta activada. Es decir EEON esta uno.

0 = EEPROM desactivada.
1 = EEPROM activada.

− Programación del registro CONFIG

El registro **CONFIG** esta implementado con células EEPROM. Por lo tanto para escribir sobre él hay que hacer lo mismo que se hace para los bytes de la EEPROM. Para borrar el **CONFIG** se realiza un ERASE BULK pero apuntando a la dirección del CONFIG. **Cuando se borra el registro CONFIG también se borra la EEPROM. Además el registro se programa estando en los modos 'Special test' o 'Bootstrap'.**

```
; primero borramos el registro
; después se programa el registro
        LDAB #$06
        STAB $103B        ;indicar modo de borrado de byte(byteerase)
PROGC   STAA $103f        ;situar cualquier dato en la dirección de CONFIG
        LDAB #$07         ; guardar el valor que tomará el registro CONFIG
        STAB $103B        ;poner a '1' EEPGM (EELAT=1). Se empieza a borrar
        JSR  retardo      ;llamar a subrutina para esperar 10ms
        CLR  $103B        ;borrar PPROG, terminar proceso borrado y pasar modo lectura
        LDAB #$03
        STAB $103B        ;poner a '1' EEPGM (EELAT=1). Se empieza a escribir
        JSR  retardo      ;llamar a subrutina para esperar 10ms
        CLR  $103B        ;borrar PPROG, terminar proceso borrado y pasar modo lectura
```

Modo de operar con el registro CONFIG

Este registro controla la inicialización del micro. Los cambios en este registro no tienen efecto hasta que no se realiza un Reset. Para ello el micro tiene un mecanismo que hace que al arrancarlo el registro **CONFIG** se copia en un Latch, de esa forma aunque se modifique el **CONFIG** sigue manteniendo la configuración inicial, pues está grabada en el Latch. Pero si ahora se hace un 'Reset' el nuevo valor del **CONFIG** se graba en el latch y la nueva configuración tiene efecto.

Para cambiar el **CONFIG** realizar los siguientes pasos.

a) Borrar el **CONFIG** (no hacer reset después de esto)
b) Programar el nuevo valor.
c) Hacer un reset para que el nuevo **CONFIG** tenga efecto.

5. LA COMUNICACIÓN ENTRE EL 68HC11 Y EL PC

5.1. INTRODUCCIÓN

Esta parte está enfocada a la realización de un software sobre el PC que trabaje con el 68HC11. La parte hardware ya está solucionada: se dispone de una tarjeta que es capaz de conectarse al PC y ejecutar programas. En esta parte se muestra cómo es posible a partir de esa tarjeta (por ejemplo la CT6811) construir herramientas software que la controlen.

5.2. EL PROGRAMA DE ARRANQUE BOOTSTRAP

5.2.1. Los modos de funcionamiento del micro

El 68HC11 puede funcionar en 4 modos diferentes, llamados: single chip, expanded, bootstrap y special test. Estos modos se configuran mediante los pines MODA y MODB del 68HC11. También es posible configurar estos modos por software, modificando unos bits de un determinado registro de control del 68HC11. El modo más empleado para arrancar es el modo bootstrap y es sobre el que trata el resto del capítulo.

5.2.2. El modo de funcionamiento bootstrap

El modo bootstrap es el más empleado a la hora de arrancar el 68HC11. Cuando la CT6811 funciona sin memoria externa, siempre arranca en este modo, independientemente de que se quiera utilizar la CT6811 en modo entrenador o autónomo.

Una vez inicializado el 68HC11 en este modo, comienza a ejecutar un programa que tiene almacenado en la ROM, llamado **bootstrap**. Los vectores de interrupción se encuentran también en memoria ROM y 'apuntan' a determinadas direcciones de la memoria RAM. Las interrupciones por defecto están inhibidas al inicializar el 68HC11.

5.2.3. Descripción del programa de arranque bootstrap

Una vez que el programa bootstrap comienza a ejecutarse lo primero que hace es configurar el 68HC11. Sin entrar en demasiados detalles, las configuraciones que realiza son las siguientes:

- Inicialización de la pila: Sitúa la pila al final de la RAM interna (Dirección $00FF)
- Configuración del SCI: La velocidad de transmisión se establece a 7812 Baudios y se habilitan el transmisor y receptor de comunicaciones.

Después de haberse configurado el 68HC11 se establece un protocolo entre el 68HC11 y el dispositivo conectado al SCI, que normalmente será un PC pero puede ser cualquier ente que disponga de comunicaciones serie asíncronas. Se supone que hay conectado un PC.

Sin entrar en detalles sobre el protocolo, que se comenta en las siguientes secciones, el resultado es que el programa bootstrap lee datos a través del puerto serie y los sitúa secuencialmente en la memoria RAM a partir de la dirección inicial $0000 hasta la dirección final $00FF. Una vez cargado el ultimo byte, se realiza un salto a la dirección $0000 y se comienza a ejecutar el programa cargado. La situación 'dentro' del 68HC11 es la siguiente. Ya no se está ejecutando el programa bootstrap, sino que el control lo tiene un programa que ha sido enviado desde el PC. Nuestro software comienza a ser ejecutado por el 68HC11.

5.2.4. Listado del programa bootstrap

```
; +--------------------------------------------------------------------------+
; ¦ Programa BOOTSTRAP. (C) MOTOROLA.                                        ¦
; +--------------------------------------------------------------------------+

; Offset de los registros de control utilizados

PORTD   EQU  $08
DDRD    EQU  $09
SPCR    EQU  $28
BAUD    EQU  $2B
SCCR1   EQU  $2C
SCCR2   EQU  $2D
SCSR    EQU  $2E
SCDAT   EQU  $2F
PPROG   EQU  $3B
TEST1   EQU  $3E
CONFIG  EQU  $3F

EEPSTR  EQU $B600       ; Comienzo de la EEPROM
EEPEND  EQU $B7FF       ; Final de la EEPROM

        ORG  $BF40

begin
        LDS  #$00FF      ; Inicializar la pila
        LDX  #$1000      ; Inicializar X para acceso indexado a registros
        BSET SPCR,X $20  ; Poner el puerto D en colector abierto

        LDAA #$A2        ; Establecer velocidad de comunicaciones
        STAA BAUD,X      ; (Divisor de velocidad 16)
                         ; Para un cristal de 8MHZ la velocidad configurada
                         ; es de 7812 Baudios

        LDAA #$0C
        STAA SCCR2,X      ; Habilitar transmisor y receptor

        BSET SCCR2,X $01  ; Enviar señal de BREAK

wait    BRSET PORTD,X $01 wait  ; Esperar hasta que se mande bit de start
        BCLR SCCR2,X $01        ; Ya no se envían más señales de BREAK

waitcar BRCLR SCSR,X $20 waitcar  ; Esperar a que llegue un carácter
        LDAA SCDAT,X              ; Leer carácter recibido

        BNE nocero      ; Si carácter recibido=$00 o BREAK saltar a la
        JMP $B600       ; memoria EEPROM

nocero  CMPA #$55       ; Si carácter recibido=$55, saltar al comienzo de
        BEQ STAR        ; la RAM. (Sólo utilizado para pruebas de fabrica)

        CMPA #$FF       ; Si carácter recibido=$FF, la velocidad de Tx actual
        BEQ baudok      ; es correcta.

        BSET BAUD,X $33 ; Establecer velocidad de 1200 baudios (Si cristal es
                        ; de 8MHZ

baudok  LDY #$0000               ; Inicializar puntero
waitdat BRCLR SCSR,X $20 waitdat  ; Esperar a que se reciba un dato
        LDAA SCDAT,X    ; Leer dato recibido
        STAA $00,Y      ; Almacenar dato en la RAM
        STAA SCDAT,X    ; Hacer eco del dato recibido
        INY
        CPY #$0100      ; ¿Se ha alcanzado el final de la RAM?
        BNE waitdat     ; NO --> Leer otro dato

STAR    JMP $0000       ; Ejecutar el programa cargado

; +--------------------------------------------------------------------------+
; ¦              VECTORES DE INTERRUPCIÓN DEL MODO BOOTSTRAP                 ¦
; +--------------------------------------------------------------------------+
        ORG $BFD4

        FDB  $00C4      ; SCI
        FDB  $00C7      ; SPI
        FDB  $00CA      ; Flanco subido en acumulador de pulsos
        FDB  $00CD      ; Desbordamiento en acumulador de pulsos
        FDB  $00D0      ; Desbordamiento del temporizador
        FDB  $00D3      ; Comparador 5
        FDB  $00D6      ; Comparador 4
        FDB  $00D9      ; Comparador 3
        FDB  $00DC      ; Comparador 2
        FDB  $00DF      ; Comparador 1
        FDB  $00E2      ; Capturador 3
        FDB  $00E5      ; Capturador 2
        FDB  $00E8      ; Capturador 1
        FDB  $00EB      ; Interrupción de tiempo real
        FDB  $00EE      ; IRQ
        FDB  $00F1      ; XIRQ
        FDB  $00F4      ; SWI
        FDB  $00F7      ; Código de instrucción ilegal
        FDB  $00FA      ; Fallo en el sistema COP
        FDB  $00FD      ; Monitor del reloj
```

```
FDB  #begin     ; Reset
END
```

5.3. DIALOGANDO CON EL 68HC11

5.3.1. El protocolo implementado en el bootstrap

A partir de ahora se toma como ejemplo la tarjeta CT6811, pero lo que se cuenta tiene validez para cualquier otra entrenadora. El PC esta conectado a la CT6811 y los switches de configuración están situados de tal forma que el 68HC11 está configurado en el modo bootstrap.

Al realizar un reset (software o hardware) de la CT6811, el 68HC11 comienza a ejecutar el programa bootstrap. Lo primero que hace es enviar una señal de **break** al PC y se queda esperando a recibir un byte de respuesta. El PC responde enviando un código, que puede ser alguno de los siguientes:

- Código $FF **:** El 68HC11 debe seguir configurado a la velocidad de 7812 baudios.
- Código $00 o BREAK**:** El 68HC11 debe 'saltar' a la memoria EEPROM
- Código $55**:** El 68HC11 debe saltar a la memoria $0000 (Sólo lo utiliza el fabricante)
- Cualquier otro código**:** El 68HC11 se configura para trabajar a 1200 baudios

Para cargar programas en la RAM interna del micro se debe enviar bien el código $FF o bien cualquier otro código diferente de $55 y $00.

Siguiendo la explicación, se supone que el 68HC11 ha recibido un carácter diferente de $55 y $00. Se queda esperando a que el PC comience a enviarle los bytes del programa a cargar. Cada byte que recibe, lo sitúa en la memoria RAM interna y hace un eco del byte recibido al PC. Cuando ha recibido el byte número 256, lo sitúa en la posición $FF, hace un eco de ese byte y realiza un salto a la posición $0000 para comenzar a ejecutar el programa cargado.

El protocolo seguido se ha representado gráficamente en la figura 38. En la figura 39 se ha dibujado el diagrama de flujo del programa bootstrap y también el diagrama de flujo de un programa en el PC para dialogar con el bootstrap.

5.3.2. Estado del micro una vez terminado el programa bootstrap.

Es importante conocer el estado del micro una vez que se ha ejecutado el programa bootstrap. Conociendo el estado, se evita volver a configurar algunos recursos del 68HC11 que ya están configurados, con lo que se ahorra código.

- La pila está inicializada en la dirección $00FF: Esta es la dirección más habitual. Si los programas utilizan esta pila, no es necesario inicializar el puntero de pila nuevamente.
- Transmisor y receptor de comunicaciones serie están activados
- Velocidad del sci configurada: 7812 ó 1200 baudios
- Puerto D configurado en colector abierto
- Registro X contiene el valor $1000

5.3.3. Velocidades de transmisión en el PC

El PC transmite sin ningún problema a la velocidad de 1200 Baudios, sin embargo no lo puede hacer a la velocidad de 7812. La velocidad más parecida que admite la UART del PC es de 7680 baudios. Esta velocidad es la utilizada en los programas del PC y da unos resultados totalmente satisfactorios.

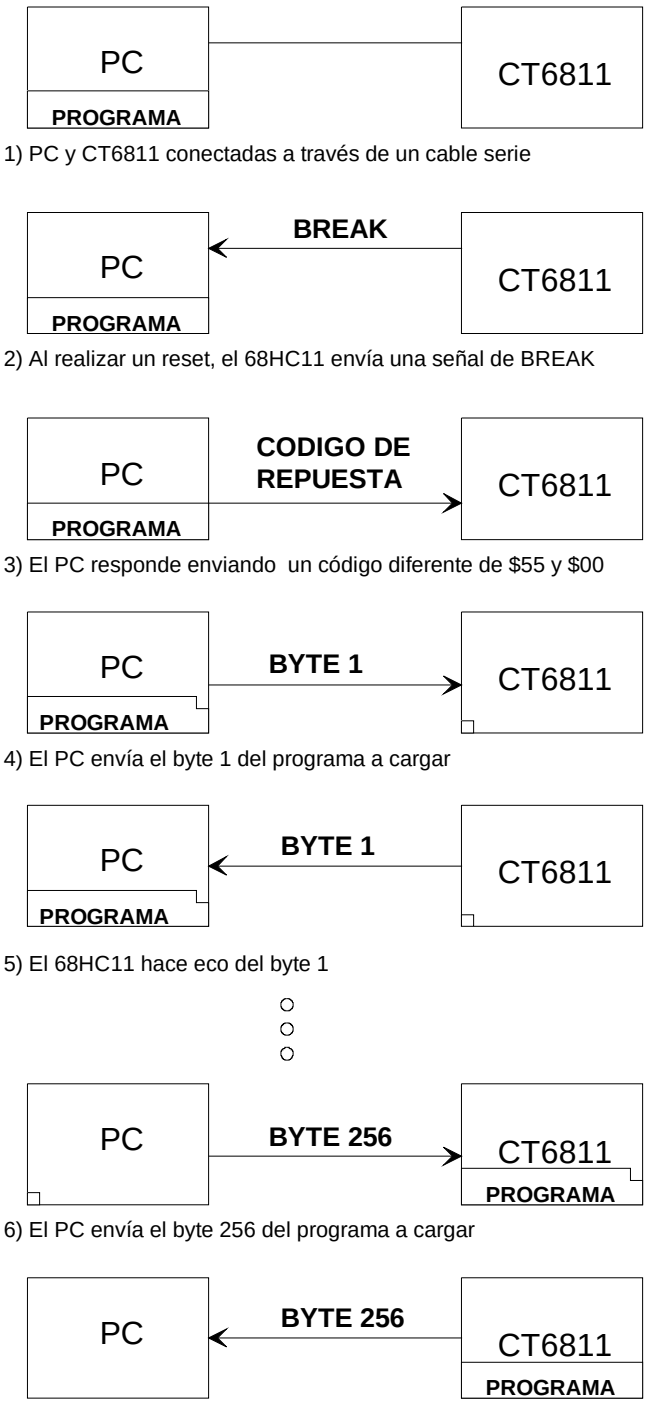


Figura 38: Protocolo empleado para 'dialogar' con el programa BOOTSTRAP.

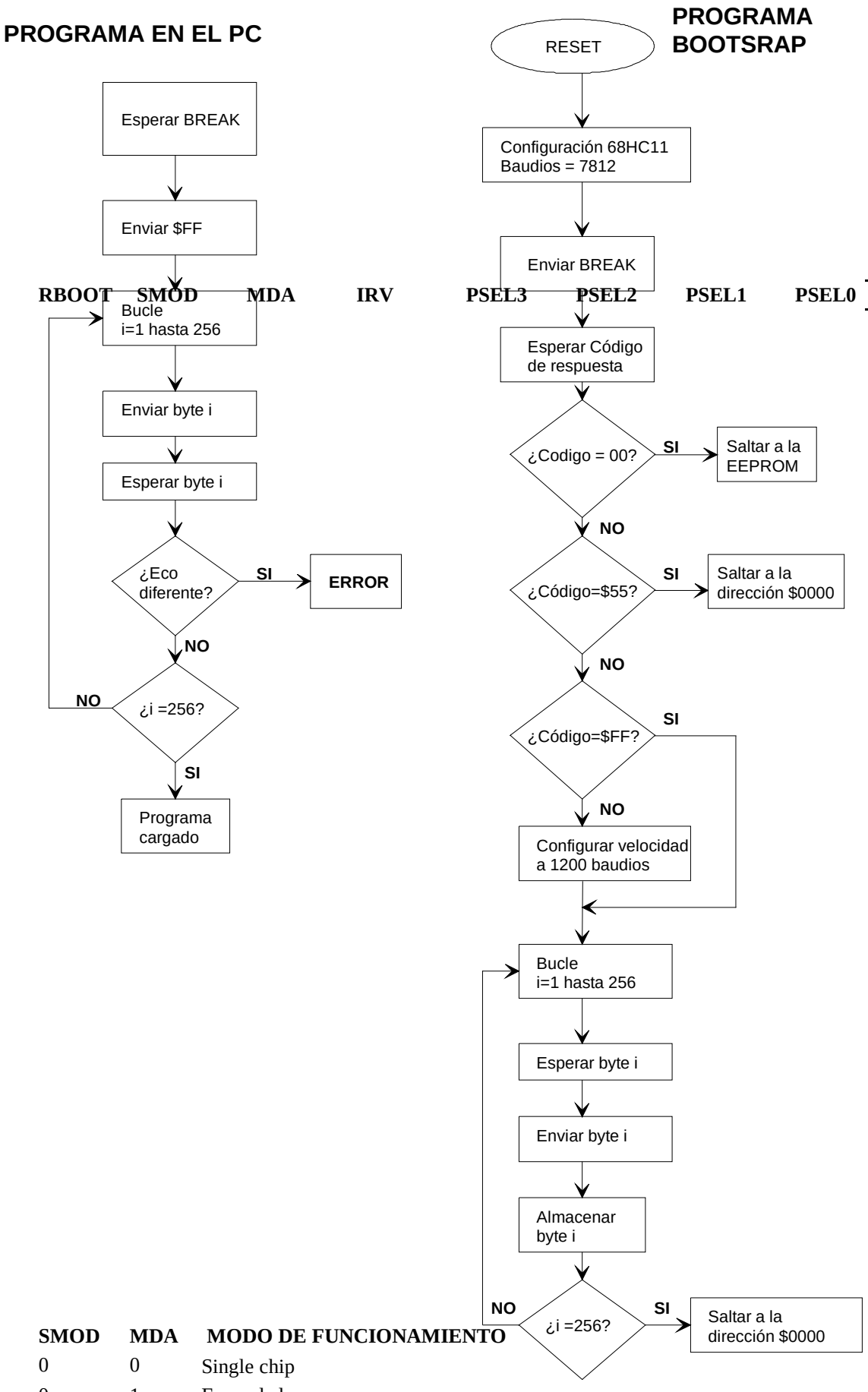


Figura 39: Diagramas de flujo

5.3.4. El registro HPRIO

El registro **HPRIO** se encuentra en la dirección $103C

RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0

RBOOT: Este bit está protegido y sólo se puede escribir si se esta en un modo especial (SMOD=1)

0 : Quitar el programa bootstrap del mapa de memoria

1 : Programa bootstrap presente en el mapa de memoria

SMOD: Indica si se esta en un modo especial o no. En los modos especiales existen privilegios especiales.

0 : Modo no especial

1 : Modo especial

MDA: Selección del modo.

SMOD	MDA	MODO DE FUNCIONAMIENTO
0	0	Single chip
0	1	Expanded
1	0	Special bootstrap

1	1	Special test

IRV = Internal Read Visibility
0 = No hay visibilidad de las lecturas internas a través del bus externo
1 = Los datos de las lecturas internas se dirigen al bus de datos externo

PSEL3−PSEL0 : Selección de prioridad de las interrupciones. (Sólo se pueden escribir si las interrupciones están deshabilitadas). La interrupción indicada pasará a tener la máxima prioridad sobre las demás

PSEL3	PSEL2	PSEL1	PSEL0	Aumentar prioridad a la fuente de interrupción:
0	0	0	0	Overflow del temporizador
0	0	0	1	Overflow en el acumulador de pulsos
0	0	1	0	Acumulador de pulsos
0	0	1	1	Transferencia completada en el SPI
0	1	0	0	Sistema SCI
0	1	0	1	Reservado
0	1	1	0	IRQ
0	1	1	1	Interrupción en tiempo real
1	0	0	0	Capturador de entrada 1
1	0	0	1	Capturador de entrada 2
1	0	1	0	Capturador de entrada 3
1	0	1	1	Comparador 1
1	1	0	0	Comparador 2
1	1	0	1	Comparador 3
1	1	1	0	Comparador 4
1	1	1	1	Comparador 5

El registro **HPRIO** es muy útil para pasar por software al modo expandido y para eliminar el bootstrap del mapa de memoria.

5.3.5. Carga de programas en la memoria externa

Teniendo la tarjeta CT6811 conectada a la tarjeta CT3216 se aumenta la RAM en 32Kbytes externos y también se añaden 16 Kbytes de EPROM externa. El software para trabajar en modo entrenador con este sistema tendría que funcionar de la siguiente manera:

1. La CT6811 debe arrancar en modo bootstrap. Inicialmente se carga un programa 'cargador' en la RAM interna, utilizando el protocolo definido por el programa bootstrap.

2. El programa cargador se ejecuta y espera a que se le envíe el programa que se quiere cargar en la RAM externa. Este programa cargador debe recibir cada byte del programa y situarlo en la dirección correcta de la memoria externa.

3. Finalmente el programa cargador debe saltar al programa cargado

5.4. EL FORMATO .S19 DE MOTOROLA

Los ficheros ejecutables que contienen el código máquina de las instrucciones en ensamblador pueden estar en varios formatos. Existen dos formatos principales: El formato .S19 de motorola y el formato .HEX de Intel. En este apartado se describe únicamente el formato .S19 que es el más habitual y es el que generan todos los compiladores para 68HC11.

Los archivos en formato .S19 están formados por líneas de dígitos hexadecimales. Cada línea comienza con los caracteres Sx, donde x puede ser el dígito 1 ó 9. Existen por tanto dos tipos de líneas, las que comienzan por S9 y las que comienzan por S1. Cada tipo de línea tiene unos campos diferentes:

- **Líneas del tipo S1:**

Número bytes línea	Dirección comienzo	Campo de datos	Checksum
1 Byte	2 BYTES	xxx Bytes	1 BYTE

Estas líneas constan de 4 campos. El primer campo indica el número de bytes que existen en la línea, sin contar con el byte de este campo ni tampoco el código S1. (Un byte son 2 dígitos hexadecimales). El segundo campo, constituido por 4 dígitos hexadecimales (2 bytes), indica la dirección de comienzo a partir de la cual se debe situar en memoria el campo de datos. El campo de datos contiene los códigos máquina de las instrucciones y los datos a almacenar en memoria. Su longitud no está determinada. Finalmente, el último campo está constituido por un byte de checksum para detectar errores en la transmisión.

- **Líneas del tipo S9:**

Número de bytes en la línea	4 dígitos de relleno	Byte de checksum
1 BYTE	2 BYTES	1 BYTE

Este tipo de líneas sólo aparecen para indicar el final de un archivo en formato .S19. El primer campo tiene la misma función que el de las líneas S1: Indica el número de bytes de la línea sin contar el byte de este campo ni los dígitos S9. El siguiente campo contiene 4 dígitos de relleno, que normalmente están a cero. El último es un byte de checksum.

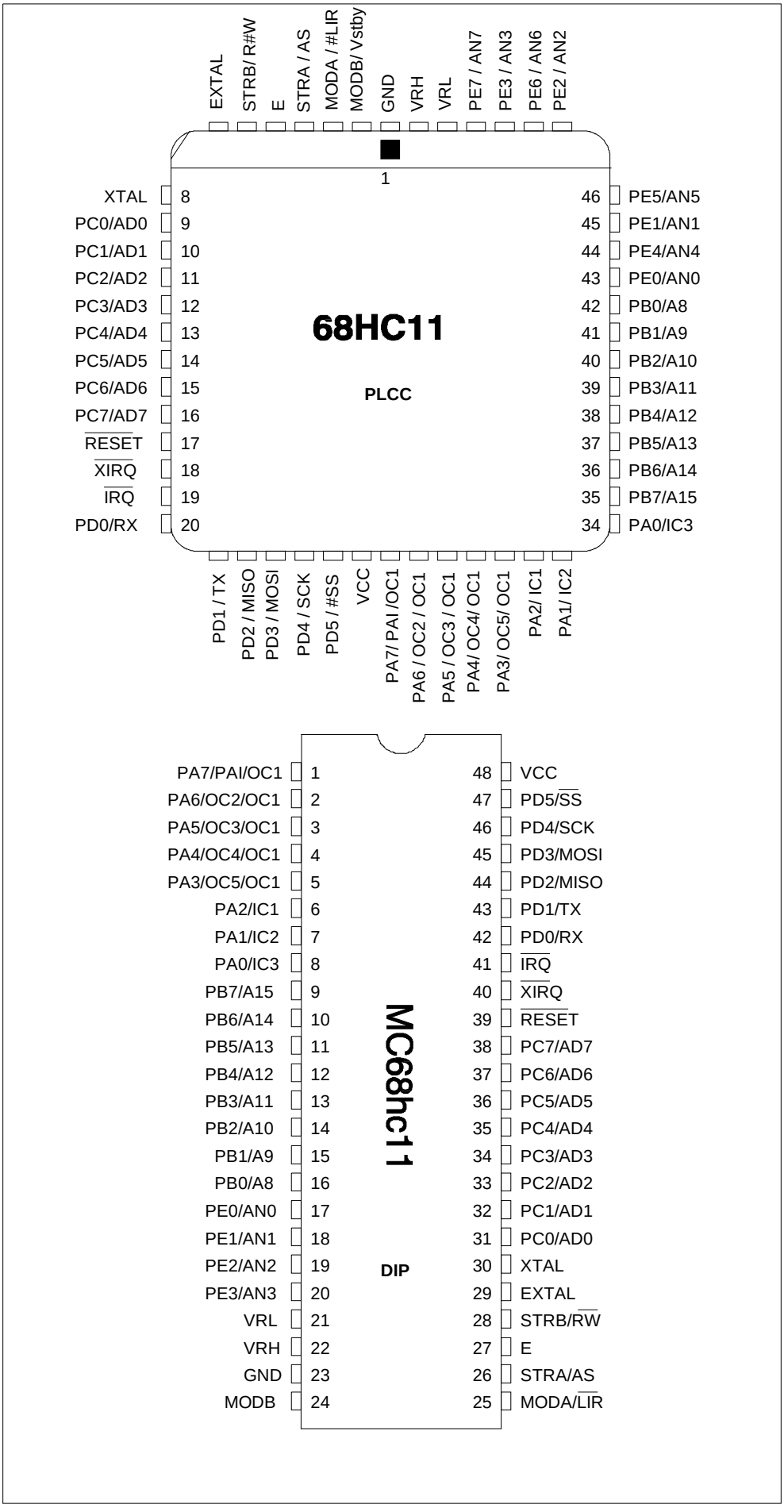
A continuación se presenta un ejemplo de un ejemplo de un archivo en formato .S19 formado por dos líneas del tipo S1 y una del tipo S9:

```
S11D8000CE10004FB710008D10814126FAF61000C8FFF710008D0920EE1F2E
S10A80201F2E80FCA72F397D
S9030000FC
```

Descomponiendo las líneas en campos, se obtiene:

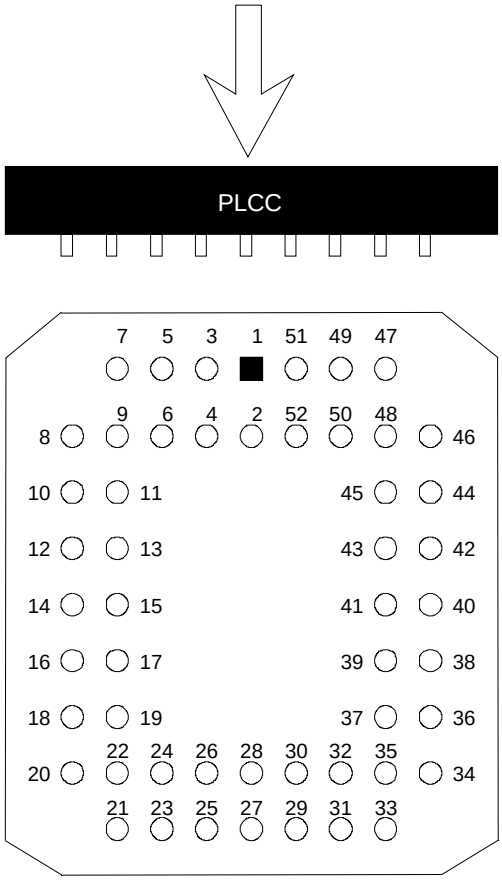
1D	8000	CE10004FB7100008D10814126FAF61000C8FFF710008D0920EE1F	2E
0A	8020	1F2E80FCA72F39	7D
03	0000	FC	

APÉNDICE A: PATILLAJE 68HC11

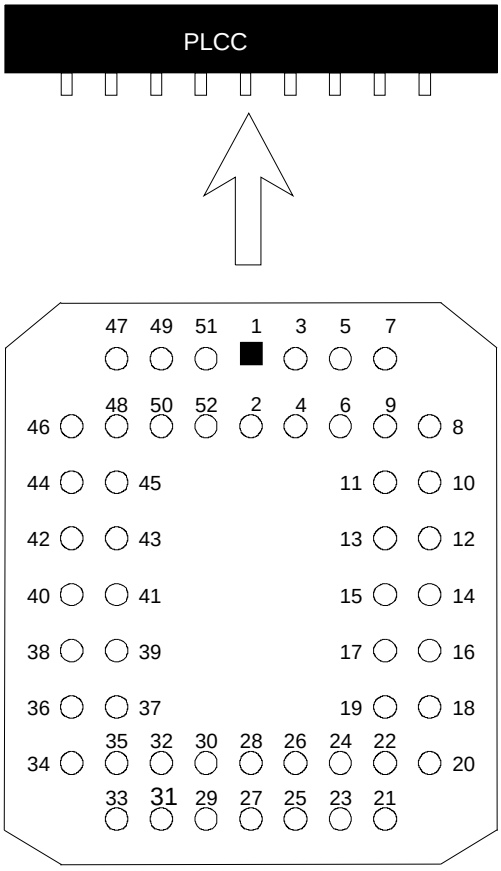

EXTAL
STRB/ R#W
E
STRA / AS
MODA / #LIR
MODB/ Vstby
GND
VRH
VRL
PE7 / AN7
PE3 / AN3
PE6 / AN6
PE2 / AN2
1
XTAL 8
PC0/AD0 9
PC1/AD1 10
PC2/AD2 11
PC3/AD3 12
PC4/AD4 13
PC5/AD5 14
PC6/AD6 15
PC7/AD7 16
RESET 17
XIRQ 18
IRQ 19
PD0/RX 20
46 PE5/AN5
45 PE1/AN1
44 PE4/AN4
43 PE0/AN0
42 PB0/A8
41 PB1/A9
40 PB2/A10
39 PB3/A11
38 PB4/A12
37 PB5/A13
36 PB6/A14
35 PB7/A15
34 PA0/IC3
68HC11
PLCC
PD1 / TX
PD2 / MISO
PD3 / MOSI
PD4 / SCK
PD5 / #SS
VCC
PA7/ PAI /OC1
PA6 / OC2 / OC1
PA5 / OC3 / OC1
PA4/ OC4/ OC1
PA3/ OC5/ OC1
PA2/ IC1
PA1/ IC2
PA7/PAI/OC1 1
PA6/OC2/OC1 2
PA5/OC3/OC1 3
PA4/OC4/OC1 4
PA3/OC5/OC1 5
PA2/IC1 6
PA1/IC2 7
PA0/IC3 8
PB7/A15 9
PB6/A14 10
PB5/A13 11
PB4/A12 12
PB3/A11 13
PB2/A10 14
PB1/A9 15
PB0/A8 16
PE0/AN0 17
PE1/AN1 18
PE2/AN2 19
PE3/AN3 20
VRL 21
VRH 22
GND 23
MODB 24
48 VCC
47 PD5/SS
46 PD4/SCK
45 PD3/MOSI
44 PD2/MISO
43 PD1/TX
42 PD0/RX
41 IRQ
40 XIRQ
39 RESET
38 PC7/AD7
37 PC6/AD6
36 PC5/AD5
35 PC4/AD4
34 PC3/AD3
33 PC2/AD2
32 PC1/AD1
31 PC0/AD0
30 XTAL
29 EXTAL
28 STRB/RW
27 E
26 STRA/AS
25 MODA/LIR
MC68hc11
DIP

APÉNDICE B:

NUMERACIÓN DEL ZÓCALO PLCC DE 52 PINES



PATILLAJE
VISTA SUPERIOR



PATILLAJE
VISTA INFERIOR

APÉNDICE C:

Resumen de los registros de control del 68HC11

	BIT 7	6	5	4	3	2	1	0	
$1000	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	**PORTA**
$1001									**sin usar**
$1002	STAF	STAI	CWOM	HDNS	OIN	PLS	EGA	INVB	**PIOC**
$1003	PC7	PC6	PC5	PC4	PC3	PC2	PC1	PC0	**PORTC**
$1004	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0	**PORTB**
$1005	PCL7	PCL6	PCL5	PCL4	PCL3	PCL2	PCL1	PCL0	**PORTCL**
$1006									**sin usar**
$1007	DDRC 7	DDRC6	DDRC5	DDRC4	DDRC3	DDRC2	DDRC1	DDRC0	**DDRC**
$1008	PD7	PD6	PD5	PD4	PD3	PD2	PD1	PD0	**PORTD**
$1009	DDRD 7	DDRD6	DDRD5	DDRD4	DDRD3	DDRD2	DDRD1	DDRD0	**DDRD**
$100A	PE7	PE6	PE5	PE4	PE3	PE2	PE1	PE0	**PORTE**
$100B	FOC1	FOC2	FOC3	FOC4	FOC5				**CFORC**
$100C	OC1M 7	OC1M6	OC1M5	OC1M4	OC1M3				**OC1M**
$100D	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3				**OC1D**
$100E	BIT 15							BIT 8	**TCNT high**
$100F	BIT 7							BIT 0	**TCNT low**
$1010	BIT 15							BIT 8	**TIC1 high**
$1011	BIT 7							BIT 0	**TIC1 low**
$1012	BIT 15							BIT 8	**TIC2 high**
$1013	BIT 7							BIT 0	**TIC2 low**
$1014	BIT 15							BIT 8	**TIC3 high**
$1015	BIT 7							BIT 0	**TIC3 low**

$1016	BIT 15							BIT 8	**TOC1 high**
$1017	BIT 7							BIT 0	**TOC1 low**
$1018	BIT 15							BIT 8	**TOC2 high**
$1019	BIT 7							BIT 0	**TOC2 low**
$101A	BIT 15							BIT 8	**TOC3 high**
$101B	BIT 7							BIT 0	**TOC3 low**
$101C	BIT 15							BIT 8	**TOC4 high**
$101D	BIT 7							BIT 0	**TOC4 low**
$101E	BIT 15							BIT 8	**TOC5 high**
$101F	BIT 7							BIT 0	**TOC5 low**
$1020	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5	**TCTL1**
$1021			EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A	**TCTL2**
$1022	OC1I	OC2I	OC3I	OC4I	OC5I	IC1I	IC2I	IC3I	**TMSK1**
$1023	OC1F	OC2F	OC3F	OC4F	OC5F	IC1F	IC2F	IC3F	**TFLG1**
$1024	TOI	RTII	PAOVI	PAII			PR1	PR0	**TMSK2**
$1025	TOF	RTIF	PAOVF	PAIF					**TFLG2**
$1026	DDRA 7	PAEN	PAMO D	PEDGE			RTR1	RTR2	**PACTL**
$1027	BIT 7							BIT 0	**PACNT**
$1028	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0	**SPCR**
$1029	SPIF	WCOL		MODF					**SPSR**
$102A	BIT 7							BIT 0	**SPDR**
$102B	TCLR		SCP1	SCP0	RCK	SCR2	SCR1	SCR0	**BAUD**
$102C	R8	T8		M	WAKE				**SCCR1**
$102D	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK	**SCCR2**
$102E	TDRE	TC	RDRF	IDLE	OR	NF	FE		**SCSR**
$102F	BIT 7							BIT 0	**SCDR**

$1030	CCF		SCAN	MULT	CD	CC	CB	CA	**ADCTL**
$1031	BIT 7							BIT 0	**ADR1**
$1032	BIT 7							BIT 0	**ADR2**
$1033	BIT 7							BIT 0	**ADR3**
$1034	BIT 7							BIT 0	**ADR4**
$1035									**sin usar**
$1036									**sin usar**
$1037									**sin usar**
$1038									**sin usar**
$1039	ADPU	CSEL	IRQE	DLY	CME		CR1	CR0	**OPTION**
$103A	BIT 7							BIT 0	**COPRST**
$103B	ODD	EVEN		BYTE	ROW	ERASE	EELAT	EEPGM	**PPROG**
$103C	RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0	**HPRIO**
$103D	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0	**INIT**
$103E	TILOP		OCCR	CBYP	DISR	FCM	FCOP	TCON	**TEST1**
$103F					NOSEC	NOCOP	ROMON	EEON	**CONFIG**

APENDICE D:

Descripción de todos los registros de control del 68HC11

ADCTL

REGISTRO DE STATUS/CONTROL DE CONVERSOR A/D

$1030	CCF		SCAN	MULT	CD	CC	CB	CA

- CCF = Conversión completa
- SCAN = Control del muestreo continuo
  - 0= Cuatro conversiones y parar
  - 1= Realizar conversiones continuamente
- MULT = Control del número de canales de conversión
  - 0= Conversión de un único canal
  - 1= Conversión de un grupo de cuatro canales
- CD − CA: Selección del los canales:

CD	CC	CB	CA	Señal del canal	Resultado en ADRx si MULT = 1
0	0	0	0	AD0 por bit 0 puerto E	ADR1
0	0	0	1	AD1 por bit 1 puerto E	ADR2
0	0	1	0	AD2 por bit 2 puerto E	ADR3
0	0	1	1	AD3 por bit 3 puerto E	ADR4
0	1	0	0	AD4 por bit 4 puerto E	ADR1
0	1	0	1	AD5 por bit 5 puerto E	ADR2
0	1	1	0	AD6 por bit 6 puerto E	ADR3
0	1	1	1	AD7 por bit 7 puerto E	ADR4
1	0	0	0	Reservado	ADR1
1	0	0	1	Reservado	ADR2
1	0	1	0	Reservado	ADR3
1	0	1	1	Reservado	ADR4
1	1	0	0	Vref hi	ADR1
1	1	0	1	Vref low	ADR2
1	1	1	0	Vref hi / 2	ADR3
1	1	1	1	test	ADR4

ADR1−ADR4

REGISTROS DE RESULTADOS DE CONVERSIÓN A/D

$1031	BIT 7							BIT 0	**ADR1**
$1032	BIT 7							BIT 0	**ADR2**
$1033	BIT 7							BIT 0	**ADR3**
$1034	BIT 7							BIT 0	**ADR4**

BAUD

Registro de control de los baudios del SCI

$102B	TCLR		SCP1	SCP0	RCKB	SCR2	SCR1	SCR0
RESET =	0	0	0	0	0	U	U	U

- TCLR = Uso exclusivo de test
- RCKB = Uso exclusivo de test
- SCP1,SCP0 = Divisor de velocidad
- SCR2−SCR0 = Selección de velocidad

CFORC

Registro para forzar los comparadores

$100B	FOC1	FOC2	FOC3	FOC4	FOC5			
RESET=	0	0	0	0	0	0	0	0

Escribir un ’1’ para forzar el comparador correspondiente

CONFIG

Registro de configuración y control

$103F					NOSEC	NOCOP	ROMON	EEON

- NOSEC = Deshabilitación del modo de seguridad
  - 0 = Modo de seguridad
  - 1 = No hay seguridad
- NOCOP = Deshabilitar el sistema COP
  - 0 = COP habilitado
  - 1 = COP deshabilitado
- ROMON = Habilitación de la memoria ROM
  - 0 = La ROM no está en el mapa de memoria
  - 1 = La ROM está en el mapa de memoria
- EEON = habilitación de la memoria EEPROM
  - 0 = La EEPROM no está en el mapa de memoria
  - 1 = La EEPROM está en el mapa de memoria

Este registro está implementado con bits del tipo EEPROM por lo que no será posible escribirlos igual que el resto de los registro. Para modificar los bits habrá que hacerlo de igual forma que para modificar la memoria EEPROM.

COPRST

Temporizador del circuito del COP

$103A	BIT 7							BIT 0

Escribir $55 y $AA para inicializar el temporizador del watchdog

DDRC

Dirección de los datos del puerto C

$1007	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0
RESET=	0	0	0	0	0	0	0	0

0 = Entradas
1 = Salidas

DDRD

Dirección de los datos del puerto D

$1009			DDD5	DDD4	DDD3	DDD2	DDD1	DDD0
RESET=	0	0	0	0	0	0	0	0

0 = Entradas
1 = Salidas

HPRIO

$103C	RBOOT	SMOD	MDA	IRV	PSEL3	PSEL2	PSEL1	PSEL0
RESET=	----	---	---	---	0	1	0	1

- RBOOT = Habilitación de la rom de arranque (Sólo se puede escribir si SMOD=1)
  0 = ROM de arranque no se encuentra en el mapa de memoria
  1 = ROM de arranque presente en el mapa de memoria

- SMOD: Indica si se esta en un modo especial o no. En los modos especiales existen privilegios especiales.
  0 : Modo no especial
  1 : Modo especial

- MDA**:** Selección del modo.

SMOD	MDA	MODO DE FUNCIONAMIENTO
0	0	Single chip
0	1	Expanded
1	0	Special bootstrap
1	1	Special test

- IRV = Internal Read Visibility

  0 = No hay visibilidad de las lecturas internas a través del bus externo
  1 = Los datos de las lectuas internas se dirigen al bus de datos externo

- PSEL3−PSEL0 : Selección de prioridad de las interrupciones. (Sólo se pueden escribir si las interrupciones están deshabilitadas). La interrupción indicada pasará a tener la máxima prioridad sobre las demás

PSEL3	PSEL 2	PSEL 1	PSEL 0	Aumentar prioridad a la fuente de interrupción:
0	0	0	0	Overflow del temporizador
0	0	0	1	Overflow en el acumulador de pulsos
0	0	1	0	Acumulador de pulsos
0	0	1	1	Transferencia completada en el SPI
0	1	0	0	Sistema SCI
0	1	0	1	Reservado
0	1	1	0	IRQ
0	1	1	1	Interrupción en tiempo real
1	0	0	0	Capturador de entrada 1
1	0	0	1	Capturador de entrada 2
1	0	1	0	Capturador de entrada 3
1	0	1	1	Comparador 1
1	1	0	0	Comparador 2
1	1	0	1	Comparador 3
1	1	1	0	Comparador 4
1	1	1	1	Comparador 5

INIT

Mapeado de la RAM y de los registros de control

$103D	RAM3	RAM2	RAM1	RAM0	REG3	REG2	REG1	REG0
RESET=	0	0	0	0	0	0	0	1

- RAM3−RAM0 = Mapeado de la memoria RAM
- REG3−REG0 = Mapeado de los registros de control

OC1D

Registro de datos del comparador 1

$100D	OC1D7	OC1D6	OC1D5	OC1D4	OC1D3			

Si OC1Mx esta activo, el dato situado en OC1Dx se envía al bit_x del Puerto A cada vez que ocurre una comparación correcta.

OC1M

Registro de mascara del comparador 1

$100C	OC1M7	OC1M6	OC1M5	OC1M4	OC1M3			
RESET=	0	0	0	0	0	0	0	0

Activar el bit correspondiente para permitir que el comparador 1 actúe sobre el bit del puerto A indicado.

OPTION

Opciones de configuración del sistema

$1039	ADPU	CSEL	IRQE	DLY	CME		CR1	CR0
RESET=	0	0	0	1	0	0	0	0

- APDU = Alimentación del conversor A/D
  - 0 = Conversor A/D no alimentado
  - 1 = Conversor A/D alimentado
- CSEL = Selección del reloj
  - 0 = Conversor A/D y EEPROM utilizan la señal de reloj E
  - 1 = Conversor A/D y EEPROM utilizan un reloj interno RC
- IRQE = Selección del flanco activo en interrupción IRQ (Time protected)
  - 0 = IRQ activa a nivel bajo
  - 1 = IRQ activa en flancos de baja
- DLY = Retraso en el arranque del oscilador al salir de una situación de STOP
  - 0 = No hay pausa
  - 1 = Pausa oblidada
- CME = Activación del monitor del reloj
  - 0 = No habilitado
  - 1 = Un paro o un descenso de velocidad en la señal de reloj provoca un reset
- CR1,CR0 = Selección del timeout del sistema COP

Para un cristal de 8MHZ:

CR0	CR1	Timeout
0	0	16.384 ms
0	1	65.536 ms
1	0	262.14 ms
1	1	1.049 s

PACNT

Contador del acumulador de pulsos

$1027	PCNT7	PCNT6	PCNT5	PCNT4	PCNT3	PCNT2	PCNT1	PCNT0

Este registro se puede leer y escribir.

PACTL

Control del acumulador de pulsos

$1026	DDRA7	PAEN	PAMOD	PEDGE			RTR1	RTR0
RESET=	0	0	0	1	0	0	0	0

- DDRA7 = Dirección de datos del pin 7 del puerto A
  - 0 = Entrada
  - 1 = Salida
- PAEN = Habilitación del contador de pulsos
  - 0 = Desactivado
  - 1 = Activado

- PAMOD = Modo de funcionamiento del acumulador de pulsos
  0 = Modo cuenta de pulsos
  1 = Modo duración
- PEDGE = Flanco de control del acumulador de pulsos
  0 = Flanco de bajada.
  1 = Flanco de subida
- RTR1,RTR0 = Periodo de activación de la interrupción de tiempo real

Para un cristal de 8 MHZ:

RTR1	**RTR0**	**Se produce la interrupción cada:**
0	0	4.10 ms
0	1	8.19 ms
1	0	16.38 ms
1	1	32.77 ms

PIOC

Registro de control de la E/S paralela

$1002	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB
RESET=	0	0	0	0	0	U	1	1

- STAF = Señal de validación A
  0 = Inactiva
  1 = Se activa con los flancos activos del pin STRA
- STAI = Permiso de interrupción de la señal de validación A
  0 = No se genera interrupción
  1 = Se solicita interrupción cuando STAF=1
- CWOM = Modo de colector abierto para el puerto C
  0 = Salida normal del puerto C
  1 = Salida en colector abierto
- HNDS = Seleción del modo de funcionamiento del protocolo
  0 = Modo de validación simple
  1 = Modo 'full handshake'
- OIN = Selección de entrada/salida
  0 = Entrada
  1 = Salida
- PLS = Selección del modo de pulsos para la salida STRB
  0 = STRB activo a nivel alto
  1 = Pulsos de STRB
- EGA = Selección del flanco activo para la señal STRA
  0 = Flanco de bajada
  1 = Flanco de subida
- INV = Inversión de la salida STRB
  0 =STRB Activo a nivel bajo
  1 = STRB Activo a nivel alto

PORTA

$1000	PA7/ PAI/ OC1	PA6/ OC2/ OC1	PA5/ OC3/ OC1	PA4/ OC4/ OC1	PA3/ OC5/ OC1	PA2/ IC1	PA1/ IC2	PA0/ IC3

RESET=	−−−	0	0	0	0	−−−−	−−−−	−−−

PORTB

$1004	PB7/ A15	PB6/ A14	PB5/ A13	PB4/ A12	PB3/ A11	PB2/ A10	PB1/ A9	PB0/ A8
RESET=	0	0	0	0	0	0	0	0

PORTC

$1003	PC7/ A7/ D7	PC6/ A6/ D6	PC5/ A5/ D5	PC4/ A4/ D4	PC3/ A3/ D3	PC2/ A2/ D2	PC1/ A1/ D1	PC0/ A0/ D0

PORTCL

$1005	PCL7	PCL6	PCL5	PCL4	PCL3	PCL2	PCL1	PCL0

PORTD

$1008			PD5/ SS#	PD4/ SCK	PD3/ MOSI	PD2/ MISO	PD1/ TxD	PD0/ RxD

PORTE

$100A	PE7/ AN7	PE6/ AN6	PE5/ AN5	PE4/ AN4	PE3/ AN3	PE2/ AN2	PE1/ AN1	PE0/ AN0

PPROG

Programación de la EEPROM

$103B	STAF	STAI	CWOM	HNDS	OIN	PLS	EGA	INVB
RESET=	0	0	0	0	0	0	0	0

- ODD : programa líneas impares (Solo usado en modo TEST)
- EVEN : programa líneas pares (Solo usado en modo TEST)
- BIT 5 : Siempre es cero
- BYTE : Selección de borrado de un byte (Este bit tiene prioridad sobre el bit ROW)
  ’0’ = Se ha seleccionado modo de borrado de fila (ROO) o total (BULA)
  ’1’= Se ha seleccionado modo de borrado de un byte
- ROO : Selección de borrado ROO (es decir de una fila)
  (Si el bit BYTE está uno este bit no tiene significado)
  ’0’= selección de borrado total (BULK ERASE)
  ’1’= selección de borrado de una fila (ROW ERASE)
- ERASE: Selección de la opción de borrado

'0'= Accesos a la EEPROM para lecturas o Programación
'1'= Accesos a la EEPROM para borrado

- EELAT: Control sobre el latch de la EEPROM
  '0'= La dirección de la EEPROM configurada para modo READ
  '1'= La dirección y dato configurados para Programación / Borrado
- EEPGM: Activación o desactivación del voltaje de programación de la EEPROM
  '0'= Voltaje de programación ON.
  '1'= Voltaje de programación OFF.

SCCR1

Registro 1 de control del SCI

$102C	R8	T8		M	WAKE			
RESET=	U	U	0	0	0	0	0	0

- R8 = Recibir el bit 8
- T8 = Transmitir el bit 8
- M = Configuración del formato de datos del SCI
  0 = 1 bit de start, 8 bits de datos y 1 bit de stop
  1 = 1 bit de start, 9 bits de datos y 1 bit de stop
- WAKE = Activación del SCI
  0 = Activación por línea libre
  1= Activación por marca

SCCR3

Registro 2 de control del SCI

$102D	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
RESET=	0	0	0	0	0	0	0	0

- TIE = Permitir interrupción de Listo para transmitir
- TCIE = Permitir interrupción de transmisión completa
- RIE = Permitir interrupción de carácter recibido
- ILIE = Permitir interrupción por línea libre
  0 = No interrupción
  1 = Interrupción permitida
- TE = Habilitación del transmisor
- RE = Habilitación del receptor
  0 = Off
  1 = On
- RWU = Control de la activación del receptor
  0 = Normal
  1 = Receptor dormido
- SBK = Enviar señal de BREAK

SCDR

Registro de datos del SCI

$102F	R7/T7	R6/T6	R5/T5	R4/T4	R3/T3	R2/T2	R1/T1	R0/T0

SCSR

Registro de Status del SCI

$102E	TDRE	TC	RDRF	IDLE	OR	NF	FE	
RESET=	1	1	0	0	0	0	0	0

- TDRE = Flag de listo para enviar
- TC = Flag de transmisión completa
- RDRF= Flag de caracter recibido
- IDLE = Flag de línea libre
- OR = Error de overrun
- NF = Error por ruido. Dato recibido puede estar mal
- FE = Error en la trama recibida.

SPCR

Registro de control del SPI

$1028	SPIE	SPE	DWOM	MSTR	CPOL	CPHA	SPR1	SPR0
RESET=	0	0	0	0	0	1	U	U

- **SPIE**: Serial Peripheral Interrupt Enable.
  1= Habilita las interrupciones del SPI.
  0= Deshabilita las interrupciones del SPI.
- **SPE**: Serial Peripheral System Enable.
  1= Activa el SPI.
  0= Desactiva el SPI.
- **DWOM**: Port D Wire−Or Mode Option
  1=La totalidad del puerto D actúa con salidas en colector abierto.
  0=La totalidad del puerto D actúa con salidas cmos.
- **MSTR**: Master Mode Select.
  1= Configurado el SPI en modo Maestro.
  0= Configurado el SPI en modo Esclavo.
- **CPOL**: Clock Polarity.
  1= El reloj se mantiene a nivel alto mientras no existan datos a transmitir.
  0= El reloj se mantiene a nivel bajo mientras no existan datos a transmitir.
- **CPHA**: Clock Phase.
  Se refiere a los dos formatos de sincronismos
- **SPR1, SPR0**: SPI Clock Rate Select.
  Permiten seleccionar la velocidad de transmisión según la siguiente tabla.

SPR1	**SPR0**	Reloj interno **E** dividido por:
0	0	2
0	1	4
1	0	16
1	1	32

SPDR

Registro de datos del SPI

CX

$102A	SPD7	SPD6	SPD5	SPD4	SPD3	SPD2	SPD1	SPD0

SPSR

Registro de status del SPI

$1029	SPIF	WCOL		MODF				
RESET=	0	0	0	0	0	0	0	0

- SPIF = Flag de listo para enviar
- WCOL = Flag de colisión en escritura
- MODF = Flag de error

TCNT

Temporizador principal

TCNT (high)

$100E	CNT15	CNT14	CNT13	CNT12	CNT11	CNT10	CNT9	CNT8

TCNT (low)

$100F	CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0

Registro de sólo lectura

TCTL1

Registro de control de los comparadores

$1020	OM2	OL2	OM3	OL3	OM4	OL4	OM5	OL5
RESET=	0	0	0	0	0	0	0	0

Omx	OLx	Acción a realizar al producirse una comparación
0	0	Ninguna acción
0	1	Cambiar de estado el pin correspondiente
1	0	Poner a cero el pin correspondiente
1	1	Poner a uno el pin correspondiente

TCTL2

Registro de control de los capturadores

$1021			EDG1B	EDG1A	EDG2B	EDG2A	EDG3B	EDG3A
RESET=	0	0	0	0	0	0	0	0

EDGxB	EDGxA	Configuración
0	0	Capturador deshabilitado
0	1	Capturar en flanco de subida
1	0	Capturar en flanco de bajada
1	1	Capturar en flanco de subida y bajada

TEST1

Registro de pruebas de fábrica

$103E	TILOP		OCCR	CBYP	DISR	FCM	FCOP	TCON
RESET=	0	0	0	0	−−−	0	0	0

- TIPOL = Prueba de instrucción ilegal
- OCCR = Mandar el registro CCR al puerto del temporizador
- CBYP= Timer divider Chain Bypass
- FCM = Forzar un error en el monitor del reloj
- FCOP = Forzar un fallo en el watchdog
- TCON = Configuración del test

TFLG1

Registro 1 de flags del temporizador

$1023	OC1F	OC2F	OC3F	OC4F	OC5F	IC1F	IC2F	IC3F
RESET=	0	0	0	0	0	0	0	0

- OC1F−0C5F = Flag del comparador 'x'
- IC1F−IC3F = Flag del capturador 'x'

Al escribir un '1' en los bits, se podrán a cero los correspondientes flags.

TFLG2

$1025	TOF	RTIF	PAOVF	PAIF				
RESET=	0	0	0	0	0	0	0	0

- TOF = Flag de overflow en el temporizador
- RTIF = Flag de la interrupción de tiempo real
- PAOVF = Flag de overflow en el acumulador de pulsos
- PAIF = Flag de flanco detectado en el acumulador de pulsos

TIC1−TIC3 **Registros de datos de los capturadores**

TIC1 (high)

$1010	IC115	IC114	IC113	IC112	IC111	IC110	IC19	IC18

TIC1 (low)

$1011	IC17	IC16	IC15	IC14	IC13	IC12	IC11	IC10

TIC2 (high)

$1012	IC215	IC214	IC213	IC212	IC211	IC210	IC29	IC28

TIC2 (low)

$1013	IC27	IC26	IC25	IC24	IC23	IC22	IC21	IC20

TIC3 (high)

$1014	IC315	IC314	IC313	IC312	IC311	IC310	IC39	IC38

TIC3(low)

$1015	IC37	IC36	IC35	IC34	IC33	IC32	IC31	IC30

TMSK1 **Registro de máscara de interrupción del temporizador**

$1022	OC1I	OC2I	OC3I	OC4I	OC5I	IC1I	IC2I	IC3I
RESET=	0	0	0	0	0	0	0	0

- OC1I−OC5I = Permitir interrupción del comparador 'x'

- IC1I−IC3I = Permitir interrupción del capturador 'x'
  0 = Interrupción inhibida
  1 = Interrupción permitida

TMSK2

$1024	TOI	RTII	PAOVI	PAII			PR1	PR0
RESET=	0	0	0	0	0	0	0	0

- TOI = Activación de la interrupción de overflow en el temporizador
- RTII = Activación de la interrupción de tiempo real
- PAOVI = Activación de la interrupción de overflow en acumulador de pulsos
- PAII = Activación de la interrupción del acumulador de pulsos
  0 = Interrupción inhibida
  1 = Interrupción permitida
- PR1,PR0 = Selección del factor de división de la señal de reloj del temporizador

PR1	PR0	Dividir E entre:
0	0	1
0	1	4
1	0	8
1	1	16

TOC1−TOC5

Registros de datos de los comparadores

TOC1 (high)

$1016	OC115	OC114	OC113	OC112	OC111	OC110	OC19	OC18

TOC1 (low)

$1017	OC17	OC16	OC15	OC14	OC13	OC12	OC11	OC10

TOC2 (high)

$1018	OC215	OC214	OC213	OC212	OC211	OC210	OC29	OC28

TOC2 (low)

$1019	OC27	OC26	OC25	OC24	OC23	OC22	OC21	OC20

TOC3 (high)

$101A	OC315	OC314	OC313	OC312	OC311	OC310	OC39	OC38

TOC3(low)

$101B	OC37	OC36	OC35	OC34	OC33	OC32	OC31	OC30

TOC4 (high)

$101C	OC415	OC414	OC413	OC412	OC411	OC410	OC49	OC48

TOC4(low)

$101D	OC47	OC46	OC45	OC44	OC43	OC42	OC41	OC40

TOC5 (high)

$101E	OC515	OC514	OC513	OC512	OC511	OC510	OC59	OC58

TOC5(low)

$101F	OC57	OC56	OC55	OC54	OC53	OC52	OC51	OC50

APÉNDICE E:
LISTADO DE LOS MNEMÓNICOS DEL 68HC11

ABA : Añadir el contenido del acumulador B al acumulador A
ABX: Añadir el contenido del acumulador B (sin signo) al contenido del registro X
ABY: Añadir el contenido del acumulador B (sin signo) al contenido del registro Y
ADCA: Añadir al acumulador A un dato y el bit de acarreo.
ADCB: Añadir al acumulador B un dato y el bit de acarreo.
ADDA: Añadir un dato al registro A
ADDB: Añadir un dato al registro B
ADDD: Añadir un dato de 16 bits al registro D
ANDA: Realizar una operación lógica AND entre un dato y el acumulado A. Resultado en A
ANDB: Realizar una operación lógica AND entre un dato y el acumulador B. Resultado en B
ASLA: Desplazar un bit a la izquierda el acumulador A.
ASLB: Desplazar un bit a la izquierda el acumulador B
ASLD: Desplazar un bit a la izquierda el acumulador D
ASRA: Desplazar un bit a la derecha el acumulador A
ASRB: Desplazar un bit a la derecha el acumulador B
BCC: Saltar si no hay acarreo
BCLR: Poner a cero bits de la memoria
BCS: Saltar si hay acarreo
BEQ: Saltar si igual
BGE: Saltar si mayor que o igual a cero
BGT: Saltar si mayor que cero
BHI: Saltar si es mayor
BHS: Saltar si mayor o igual
BITA: Comprobar el bit especificado del acumulador A
BITB: Comprobar el bit especificado del acumulador B
BLE: Saltar si menor que o igual a cero
BLO: Saltar si menor (Mismo que BCS)
BLS: Saltar si menor o igual
BLT: Saltar si menor que cero
BMI: Saltar si negativo
BNE. Saltar si no es igual
BPL: Saltar si es positivo
BRA: Salto incondicional
BRCLR: Saltar si los bits especificados están a cero
BRN: No saltar nunca (Equivalente a una operación NOP de 2 bytes)
BRSET: Saltar si los bits especificados están a uno
BSET: Poner los bits especificados a uno
BSR: Saltar a una subrutina
BVC: Saltar si no ha habido overflow
BVS: Saltar si ha habido overflow
CBA: Comparar el acumulador A con el B
CLC: Poner a cero bit de acarreo
CLI: Permitir las interrupciones
CLR: Poner a cero el contenido de memoria especificado
CLRA: Poner a cero el acumulador A
CLRB: Poner a cero el acumulador B
CLV: Poner a cero el bit de overflow
CMPA: Comparar el acumulador A con un dato
CMPB: Comparar el acumulador B con un dato
COMA: Complementar a uno el acumulador A
COMB: Complementar a uno el acumulador B
COM: Complementar a uno el contenido de memoria especificado
CPD: Comparar el registro D con un dato
CPX: Comparar el registro X con un dato
CPY: Comparar el registro Y con un dato
DAA: Ajuste decimal

CX

DEC: Decrementar una posición de memoria especificada
DECA: Decrementar el acumulador A
DECB: Decrementar el acumulador B
DES: Decrementar el puntero de pila SP
DEX: Decrementar el registro X
DEY: Decrementar el registro Y
EORA: Operación XOR entre un dato y el acumulador A
EORB: Operación XOR entre un dato y el acumulador B
FDIV: División
IDIV: División entera
INC: Incrementar el contenido de una posición de memoria
INCA: Incrementar el acumulador A
INCB: Incrementar el acumulador B
INS: Incrementar el puntero de pila SP
INX: Incrementar el registro X
INY: Incrementar el registro Y
JMP: Salto incondicional
JSR: Salto a una subrutina
LDAA: Cargar un dato en el acumulador A
LDAB: Cargar un dato en el acumulador B
LDD: Cargar un dato de 16 bits en el registro D
LDS: Cargar un dato de 16 bits en el puntero de pila SP
LDX: Cargar un dato de 16 bits en el registro X
LDY: Cargar un dato de 16 bits en el registro Y
LSL: Desplazamiento de un bit hacia la izquierda del contenido de una posición de memoria
LSLA: Desplazar el acumulador A un bit hacia la izquierda
LSLB: Desplazar el acumulador B un bit hacia la izquierda
LSLD: Desplazar el acumulador D un bit hacia la izquierda
LSR: Desplazaar el contenido de una posición de memoria un bit hacia la derecha
LSRA: Desplazar el acumulador A un bit hacia la derecha
LSRB: Desplazar el acumulador B un bit hacia la derecha
LSRD: Desplazar el registro D un bit hacia la derecha
MUL: Multiplicación sin signo
NEG: Complementar a dos el contenido de una posición de memoria
NEGA: Complementar a dos el contenido del acumulador A
NEGB: Complementar a dos el contenido del acumulador B
NOP: No operación
ORAA: Realizar la operación lógica OR entre un dato y el acumulador A
ORAB: Realizar la operación lógica OR entre un dato y el acumulador B
PSHA: Meter el acumulador A en la pila
PSHB: Meter el acumulador B en la pila
PSHX: Meter el registro X en la pila
PSHY: Meter el registro Y en la pila
PULA: Sacar el acumulador A de la pila
PULB: Sacar el acumulador B de la pila
PULX: Sacar el registro X de la pila
PULY: Sacar el registro Y de la pila
ROL:Rotación a la izquierda del contenido de una posición de memoria
ROLA: Rotar a la izquierda el acumulador A
ROLB: Rotar a la izquierda el acumulador B
ROR: Rotar a la derecha el contenido de una posición de memoria
RORA: Rotar a la derecha el contenido del acumulador A
RORB: Rotar a la derecha el contenido del acumulador B
RTI: Retorno de interrupción
RTS: Retorno de subrutina
SBA: Restar un dato al acumulador A
SBCA: Restar un dato y el bit de acarreo al acumulador A
SBCB: Restar un dato y el bit de acarreo al acumulador B
SEC: Poner a uno el bit de acarreo
SEI: Inhibir las interrupciones
SEV: Poner a uno el bit de overflow

STAA: Almacenar el acumulador A en una posición de memoria
STAB: Almacenar el acumulador B en una posición de memoria
STD: Almacenar el registro D
STOP: Para el reloj del sistema
STS: Almacenar el puntero de pila SP
STX: Almacenar el registro X
STY: Almacenar el registro Y
SUBA: Restar un dato al acumulador A
SUBB: Restar un dato al acumulador B
SUBD: Restar un dato al registro D
SWI: Provocar una interrupción software
TAB: Tansferir el acumulador A al acumulador B
TAP: Transferir el acumulador A al registro CCR
TBA: Transferir el acumulador B al acumulador A
TEST: Instrucción de test. Sólo se puede ejecutar en el modo de test
TPA: Transferir el registro CCR al acumulador A
TST:Comprobar si una posición de memoria está a cero
TSTA: Comprobar si el acumulador A está a cero
TSTB: Comprobar si el acumulador B está a cero
TSX: Transferir el puntero de pila al registro X
TSY: Transferir el puntero de pila al registro Y
TXS: Transferir el registro X al puntero de pila
TYS: Transferir el registro Y al puntero de pila
WAI: Esperar a que se produzca una interrupción
XGDX: Intercambiar los valores de los registro X y D
XGDY: Intercambiar los valores de los registros Y y D