

# **MICROCONTROLADORES DE 32 BITS**

## **COLDFIRE V1/ FAMILIA JM**

**DIEGO ALEJANDRO MÚNERA HOYOS**

**Revisión técnica:**

0.1

# Contenido

**Introducción**

**Lista de Figuras**

**Lista de Ejemplos**

## PARTE I Conceptos Generales

**Objetivos**

**Contenidos**

### **Capítulo 1. Historia de los microcontroladores**

#### **1.1. Evolución de las máquinas**

Historia

La memoria

Los módulos o periféricos

Lenguajes de programación

#### **1.2. El futuro**

#### **1.3. Referencias**

#### **1.4. Preguntas**

### **Capítulo 2. Arquitecturas**

#### **2.1. Von Newman**

#### **2.2. Harvard**

#### **2.3. Otros conceptos y otras arquitecturas**

CISC y RISC

SISD, SIMD, MISD y MIMD

VLIW

Superescalar

Paralelas

Predicción y ejecución especulativa

#### **2.4. Referencias**

#### **2.5. Preguntas**

### **Capítulo 3. Modos de direccionamiento y tipos de instrucciones**

#### **3.1. Los direccionamientos**

Directo

Inmediato

Indirecto

De *Stack Pointer*

Con *offset*

A base

Escalado

Mixtos (base+ índice \* escalar + desplazamiento)

Relativo

Salto absoluto

Bit

Memoria a memoria

### **3.2. Lenguaje *assembler***

### **3.3. Tipos de instrucciones**

Formato de una instrucción

Movimiento de datos

Aritméticas y lógicas

De control

Manipulación de bits

Rote y desplazamiento

Salto relativo

Salto absoluto

Especiales

### **3.4. Directivas o pseudo-instrucciones**

De definición

De origen

Otras

### **3.5. Referencias**

### **3.6. Preguntas**

## **Capítulo 4. Elementos y herramientas básicas de desarrollo**

### **4.1. Consideraciones de diseño con microcontroladores**

El diseño

Limitación de memoria y velocidad de ejecución

Algoritmos y diagramación previa

Las banderas de usuario como marca de eventos

### **4.2. La codificación**

### **4.3. El ensamblaje o compilación**

### **4.4. La depuración**

La puesta a punto

### **4.5. Herramientas de desarrollo disponibles en el mercado**

Spyder

Multitink

CyclonePro

DEMOJM

### **4.6. Otros aspectos a tener en cuenta en el diseño con microcontroladores**

El RESET

El Reloj

El circuito impreso (PCB)

Limitaciones eléctricas y del ambiente

Ruido (EMI – EMC)

### **4.7. Referencias**

### **4.8. Preguntas**

## **PARTE II**

### **Arquitectura de los microcontroladores de 32 bits ColdFire® V1**

#### **Objetivo**

#### **Contenidos**

##### **Capítulo 5. La familia de los Microcontroladores ColdFire® V1**

###### **5.1. Introducción a la arquitectura ColdFire® V1 de 32 bits**

Características generales

Arquitectura del núcleo

Modos de operación

Modelo de programación

###### **5.2. Organización de la memoria**

###### **5.3. Procesamiento de excepciones**

###### **5.4. La pila y el puntero a pila**

###### **5.5. Referencias**

###### **5.7. Preguntas**

##### **Capítulo 6. Modos de Direcciónamiento y Juego de Instrucciones**

###### **6.1. Modos de direcciónamiento**

###### **6.2. Resumen del juego instrucciones**

###### **6.3. Ejercicios con direcciónamientos e instrucciones**

###### **6.4. Referencias**

###### **6.5. Preguntas**

##### **Capítulo 7. Migración de 8 a 32 bits**

###### **7.1. Una breve descripción de la familia de 8 bits HCS08**

###### **7.2. Concepto FLEXIS**

###### **7.3. Cuidados en la migración**

###### **7.4. Referencias**

###### **7.5. Preguntas**

## **PARTE III**

### **Ambiente de programación, herramientas de desarrollo**

#### **Objetivo**

#### **Contenidos**

##### **Capítulo 8. Ambiente de programación y herramienta de desarrollo**

###### **8.1. Introducción al compilador CodeWarrior®**

###### **8.2. Descripción de la herramienta DEMOJM**

###### **8.3. Creación de proyectos en C**

###### **8.4. Familiarización con el ambiente integrado de desarrollo (IDE)**

El Editor de CodeWarrior®  
El Compilador de CodeWarrior®  
El Depurador de CodeWarrior®

### **8.5. El primer programa en C**

### **8.6. Herramienta de desarrollo DEMOJM**

Descripción del *hardware*  
Puesta a punto  
Ejemplo desde el concepto FLEXIS

### **8.7. Referencias**

### **8.8. Preguntas**

## **PARTE IV**

### **El Microcontrolador de 32 bits MCF51QE128**

#### **Objetivo**

#### **Contenidos**

### **Capítulo 9. Generalidades de la máquina MCF51JM128**

- 9.1. Diagrama en bloques**
- 9.2. Distribución de pines y conexión mínima**
- 9.3. Distribución de la memoria**
- 9.4. Modos de bajo consumo**
- 9.5. Referencias**
- 9.6. Preguntas**

### **Capítulo 10. El RESET, la máquina de excepciones y el controlador de interrupciones**

- 10.1. Las fuentes de RESET**
- 10.2. El controlador de interrupciones**
- 10.3. El Pin de Interrupción (IRQ: *Interrupt Request*)**
- 10.4. Ejercicio con la IRQ: Atención de una interrupción por flanco en el pin de IRQ**
- 10.5. Referencias**
- 10.6. Preguntas**

### **Capítulo 11. Reloj del sistema (MCG: *Multipurpose Clock Generator*)**

- 11.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**
- 11.2. Ejercicio con el MCG: Inicialización del reloj interno de la MCU y comprobación de la velocidad de trabajo**
- 11.3. Referencias**
- 11.4. Preguntas**

### **Capítulo 12. Los puertos de entrada y salida**

- 12.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**
- 12.2. La función KBI (*Keyboard Interrupts*)**

**12.3. Ejercicio con los puertos: Inicialización de una pantalla de cristal líquido (LCD) y migración de 8 a 32 bits.**

**12.4. Referencias**

**12.5. Preguntas**

**Capítulo 13. Funciones de Temporización (TPM: *Timer/Pulse Width Modulator Module*)**

**13.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**

**13.2. Ejercicios con el TPM:**

Funcionamiento del TPM como temporizador de propósito general

Funcionamiento del TPM como captura de evento de entrada (INPUT CAPTURE)

Funcionamiento del TPM como PWM (*Pulse Width Modulation*)

Funcionamiento del TPM como OUTPUT COMPARE

**13.3. El RTC (*Real Time Counter*)**

**13.4. Ejercicio con el RTC: Reloj digital inicializado por teclado**

**13.5. Referencias**

**13.6. Preguntas**

**Capítulo 14. Conversión A/D (ADC: *Analog to Digital Converter*)**

**14.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**

**14.2. Programa ejemplo: Medida de la temperatura utilizando un sensor LM35 y visualización en LCD**

**14.3. Referencias**

**14.4. Preguntas**

**Capítulo 15. Comparación analógica (ACMP: *Analog Comparator*)**

**15.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**

**15.2. Programa ejemplo: Monitoreo del voltaje en una celda de NiMh y control sobre la carga**

**15.3. Referencias**

**15.4. Preguntas**

**Capítulo 16. Comunicaciones seriales asíncronas (SCI: *Serial Communication Interface*)**

**16.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**

**16.2. Programa ejemplo: Comunicación con una PC vía Hyperterminal**

**16.3. Referencias**

**16.4. Preguntas**

**Capítulo 17. Comunicaciones seriales sincrónicas (SPI: *Serial Peripheral Interface / IIC: Inter.-Integrated Circuit*)**

**17.1. Diagrama en bloques, explicación del funcionamiento y registros asociados al módulo SPI**

**17.2. Diagrama en bloques, explicación del funcionamiento y registros asociados al módulo IIC**

### **17.3. Programas ejemplos:**

Comunicación SPI entre dos sistemas MCU

Lectura y escritura sobre una memoria serial IIC

### **17.4. Referencias**

### **17.5. Preguntas**

## **Capítulo 18. Comunicaciones seriales (USB: *Universal Serial Bus*)**

### **18.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**

### **18.2. Ejemplo con el módulo USB: Conexión de un mouse USB al DEMOJM**

(Comunicación como HOST)

### **18.3. Referencias**

## **Capítulo 19. Otros módulos del MCF51JM128**

### **19.1. Diagrama en bloques, explicación del funcionamiento y registros asociados al módulo CMT**

### **19.2. Diagrama en bloques, explicación del funcionamiento y registros asociados al módulo CAN**

### **19.3. Referencias**

# Introducción

Este texto está dirigido a estudiantes y docentes en las áreas de las Técnicas Digitales y Lenguajes de Programación, que desarrollen sobre Sistemas Embebidos a microcontroladores; así como a personas que participen en el desarrollo de nuevos productos en las empresas de manufactura (OEM: *Original Equipment Manufacturer*).

El objetivo fundamental del texto es la enseñanza de los conceptos básicos de los microcontroladores de Freescale de 32 bits, en particular el microcontrolador ColdFire 51JM128. El texto contiene una gran variedad de ejemplos y ejercicios prácticos, que ayudarán al lector a la asimilación inmediata de estos conceptos.

Para tal objetivo, el lector encontrará una serie de prácticas comprobadas al 100% y aplicadas sobre un modelo de desarrollo simple pero efectivo. Este modelo se apoya en las siguientes tres aspectos:

- **Metodología y disciplina de diseño:** Proporcionada por la experiencia del autor durante más de 20 años en el entorno de los sistemas embebidos a microcontrolador.
- **Recursos:** Documentación para el empleo de una herramienta de desarrollo de gran sencillez y bajo costo como el DEMOJM, para el montaje de las prácticas propuestas en el texto, sobre la plataforma CodeWarrior® 6.2 de Freescale.
- **Prácticas implementadas y propuestas:** Listado de los programas y diagramas de los circuitos, para los ejercicios realizados y propuestos.

Para la comprensión del texto no es necesario que el lector tenga amplio conocimiento sobre lenguajes de programación, dado que el modelo de desarrollo sobre las funciones básicas del lenguaje C.

A continuación se hace una breve descripción de la organización del libro.

**Parte I. Conceptos Generales:** Esta sección contiene una breve descripción histórica, actualidad y evolución de los microcontroladores. En el aparte encontrarán la definición de los componentes que conforman su arquitectura interna y externa. También, de una manera muy genérica, aparecen las definiciones de los diferentes modos de direccionamiento y los tipos de instrucciones que hacen a la MCU (*Microcontroller Unit*) una herramienta poderosa para resolver problemas en áreas tan importantes como el Control, Telecomunicaciones, Bioingeniería, entre otras. Al final del la PARTE I es presentado al lector una serie de elementos y herramientas adicionales, para el desarrollo con dispositivos MCU y recomendaciones para el diseño de circuitos con microcontroladores.

**Parte II. Arquitecturas Freescale de 8 y 32 bits:** El libro, como se dijo al principio del prólogo, enfatiza el uso de microcontroladores de 8 y 32 bits como elementos de desarrollo de prototipos o productos terminados. Las arquitecturas HCS08 y ColdFire®

V1 de Freescale trabajan el concepto de la migración de 8 a 32 bits, aportando ventajas como un muy alto ahorro de energía y herramientas de desarrollo comunes. Es entonces en este aparte en donde aparece el estudio de las arquitecturas mencionadas.

**Parte III. Conceptos de Programación:** Este aparte tiene como objetivo la introducción al lector sobre el ambiente de programación CodeWarrior®, en su versión 6.2. Otro objetivo importante es la presentación de la herramienta de desarrollo sobre la cual se basan las prácticas del libro, llamada DEMOJM.

**Parte IV. Los Microcontroladores MC9S08JM128 y MCF51JM128:** Corresponde a la parte final del texto y está orientada hacia los miembros de las familias HCS08 y ColdFire® V1 de Freescale. En la parte inicial hay una orientación al lector en el empleo de una herramienta de desarrollo de muy bajo costo (US\$99) sobre la cual se implementarán la gran mayoría de los ejercicios del texto, llamada DEMOQE128 de Pemicro. También, como una segunda alternativa, aparece una recomendación sobre el diseño de un sistema de desarrollo básico, que sería compatible con los ejercicios tratados. Más adelante se definen y se trabajan las arquitecturas y los módulos internos de los procesadores MC9S08QE128 y MCF51QE128. Para cada módulo considerado hay un ejercicio práctico, desarrollado en ANSI C y con algunos apartes de *Assembler*, que refuerza rápidamente al usuario sobre la programación y/o configuración del mismo. Al final del aparte, el lector podrá encontrar algunas recomendaciones para el trabajo con módulos no vistos con detalle.

Se espera que este esfuerzo sirva para formar en el lector una idea concreta del poder operativo de los microcontroladores de 8 y 32 bits, así como de la facilidad de migrar de una máquina a otra conservando bajos niveles de consumo de energía, utilizando herramientas de desarrollo comunes y de bajo costo.

El autor agradece a la compañía Freescale y en especial al ingeniero Armando Molano, por su valioso apoyo en el desarrollo de este texto.

**Diego Alejandro Múnera Hoyos.**

# **PARTE I**

## **CONCEPTOS GENERALES**

### **OBJETIVO**

Esta parte tiene como objetivo la presentación de los conceptos generales, necesarios para el desarrollo de prototipos y/o productos en el entorno de tecnologías embebidas en microcontroladores de 8 y 32 bits. En el Capítulo no se hace referencia a una tecnología en particular, más bien se tratan los conceptos y definiciones desde una perspectiva general.

### **CONTENIDOS**

#### **CAPÍTULO 1. Historia de los microcontroladores**

Presenta una lectura rápida y amena, que ubique temporalmente al lector sobre la evolución de los microcontroladores, la memoria, los módulos o periféricos y los lenguajes de programación para desarrollar en estos.

#### **CAPÍTULO 2. Arquitecturas**

En este capítulo se conceptualiza sobre las diferentes arquitecturas en las cuales están implementadas las unidades centrales de proceso (CPU), que conforman el núcleo (*core*) de los microcontroladores.

#### **CAPÍTULO 3. Modos de direccionamiento y tipos de instrucciones**

Pasa a ser uno de los capítulos más importantes del libro, debido a que comienza a definir la parte operativa y de programabilidad<sup>1</sup> que tienen los procesadores.

La manera como los procesadores operan sobre los datos y la forma como direccionan los elementos de almacenamiento, informa al usuario sobre su potencial de procesamiento y se convierte en un factor decisivo a la hora de seleccionar una máquina.

#### **CAPÍTULO 4. Elementos y herramientas básicas de desarrollo**

Es la parte filosófica y metodológica del texto, fundamentada en la experiencia de los autores. En este capítulo, el lector encontrará recomendaciones genéricas a la hora de enfrentar un diseño basado en sistemas embebidos con microcontroladores. Adicionalmente, el capítulo presenta las diferentes etapas y las herramientas que se utilizan en el desarrollo de sistemas con dispositivos MCU.

---

<sup>1</sup> Concepto de gran importancia, descrito por el matemático John Von Newman en la década de los 40 del siglo XX. Von Newman, se refería a la potencia de un procesador debido a su condición de poder ejecutar el código almacenado desde sus elementos de memoria.

# CAPÍTULO 1

## Historia de los Microcontroladores

### 1.1. Evolución de las máquinas

Historia

La memoria

Los módulos o periféricos

Lenguajes de programación

Herramientas de desarrollo

### 1.2. El futuro

### 1.3. Referencias

### 1.4. Preguntas

## 1.1. EVOLUCIÓN DE LAS MÁQUINAS

El microcontrolador es uno de los inventos más sobresalientes del siglo pasado. En la actualidad aporta soluciones al ser humano en comunicación, entretenimiento, salud, seguridad, confort, movilidad y controlabilidad en general. Existen más de 15.000 millones de soluciones, implementadas en el planeta, a base de microcontroladores.

El núcleo de un microcontrolador (MCU) es un microprocesador (CPU). Siendo este último un **dispositivo lógico secuencial utilizado en sistemas electrónicos digitales el cual realiza, mediante su característica de programabilidad, operaciones aritméticas, lógicas y de control**. En la actualidad, estas máquinas han alcanzado capacidades de manipulación del dato de hasta 32 bits, que los hace muy poderosos en cálculos matemáticos y lógicos.

Haciendo una corta definición, se podría decir que un microcontrolador es un: **dispositivo lógico secuencial utilizado en sistemas electrónicos digitales y análogos, el cual realiza mediante su característica de programabilidad, operaciones aritméticas, lógicas y utilizando sus dispositivos periféricos internos, operaciones de controlabilidad (*embedded controller*)**.

En el año 1965, el físico Gordon Moore publica un artículo en la revista Electronics Magazine, sobre lo que más tarde conoceríamos como la Ley de Moore (¡más que una ley es una profecía!). Esta ley predice el comportamiento del crecimiento en número de transistores de la arquitectura de una CPU, tal como aparece en la Figura 1.1. Dice Moore: “Esta predicción se mantendrá hasta el año 2011 y obedece más a razones comerciales que tecnológicas”.

La ley predice que cada 18 meses la cantidad de transistores contenidos en una CPU debe duplicarse, de lo contrario no será comercialmente viable.

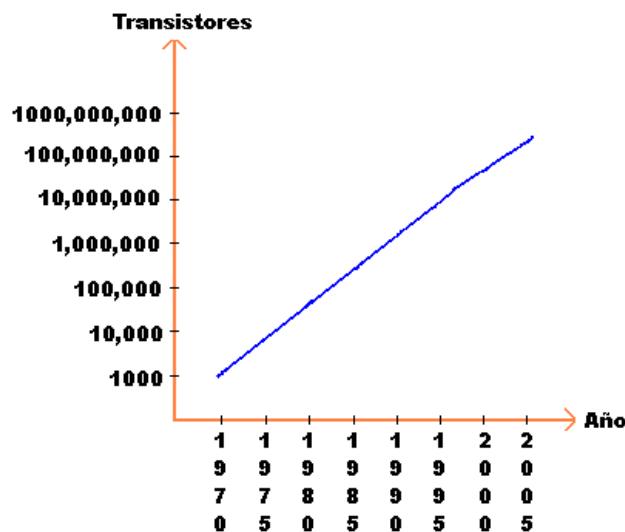


Figura 1.1. Grafico de Moore [2].

## **Historia**

Para la década de los años 70 (siglo XX), aparecen los primeros microcomputadores encapsulados en un chip, que más tarde se llamarían microcontroladores. Más que la representación tabulada de los diferentes modelos, fechas de aparición, características básicas y el nombre de las industrias dedicadas al desarrollo de los microcontroladores.

Compañías tan importantes como FREESCALE, RENESAS, ATMEL, RABBIT, DALLAS, INFINEON, INTEL, MICROCHIP, TOSHIBA, NEC, SILICON LABORATORIES, entre otras, mantienen una fuerte competencia en la innovación y producción sobre los dispositivos microcontroladores. La competencia se desarrolla en aspectos como la capacidad en bits, migración, velocidad, variedad en periféricos y bajo consumo, entre otros.

A continuación se resaltan algunos aspectos importantes en la evolución tecnológica de los MCU.

- **Capacidad en bits**

Ya se ha dicho que los microcontroladores crecieron al lado de los microprocesadores y en ese sentido se han desarrollado en 4, 8, 16 y hasta 32 bits.

- **Tecnología de programación**

Inicialmente, el programa almacenado residía en ROM de máscara y era el fabricante quien lo generaba. La desventaja de esta tecnología radicaba en los altos volúmenes que se tenían que producir, para que justificara su fabricación desde el punto de vista económico y el riesgo de cometerse algún error, factor por el cual se perdía una gran cantidad de dinero.

Más tarde aparecen las versiones OTP (*One Time Programmable*) de muy bajo costo, la programación es realizada por el usuario, pero con el cuidado de que sólo se puede hacer una vez. De tal manera que una vez programado el microcontrolador no había forma de revertir el proceso.

Con un costo mayor, pero más seguro, se presenta la alternativa de programación sobre una EPROM. Un gran inconveniente de esta tecnología era la demora en el borrado de los datos, debido a la larga exposición del *chip* ante lámparas de luz ultra violeta.

La anterior tecnología es mejorada por la EEPROM, que permite grabar y borrar eléctricamente los datos, pero con el inconveniente de la velocidad de procesamiento.

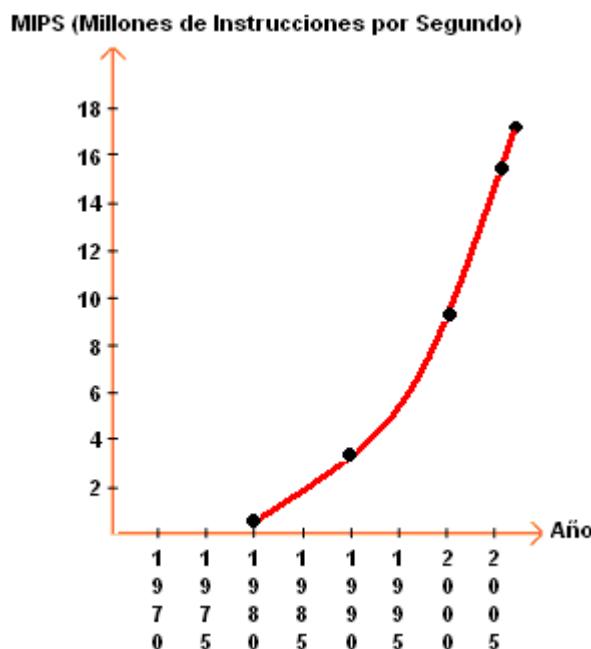
Finalmente, es la tecnología FLASH la que se impone en los mercados por su bajo costo, velocidad y facilidad de manipulación en la programación. Tiene la ventaja de poderse reprogramar miles de veces (aún más que la EEPROM) y su borrado es eléctrico, pero es importante tener en cuenta las versiones ROM y OTP como una

buenas alternativas para abaratar costos de fabricación y que hoy en día son bastante utilizadas en productos masivos.

- **Velocidad**

A diferencia de los microprocesadores, el objetivo fundamental de un microcontrolador no es el de ejecutar operaciones a velocidades de cientos de MHz, más bien, lo interesante sería ejecutar operaciones a la más alta velocidad con el más bajo consumo y un costo razonable. El objetivo anterior radica en que la mayoría de las aplicaciones con microcontroladores se orientan a dispositivos portátiles soportados por baterías.

La Figura 1.2 presenta la evolución en velocidad de los microcontroladores de 8 bits hasta la actual década. Observe la aceleración exponencial de la velocidad en la última década.



**Figura 1.2. Evolución de la velocidad para las máquinas de 8 bits.**

Otras máquinas con arquitecturas más complejas como es el caso del microcontrolador de 16 bits C8051F120<sup>2</sup>, que ejecuta operaciones a 100 MIPS con un reloj de 100MHz o el microcontrolador de 32 bits MCF5485<sup>3</sup>, que ejecuta operaciones a 308 MIPS con un reloj de 200MHz.

<sup>2</sup> Tomado del artículo:  
<http://electronicdesign.com/Articles/ArticleID/2096/2096.html>

<sup>3</sup> Tomado de:  
[http://www.freescale.com/webapp/sps/site/prod\\_summary.jsp?code=MCF548X&nodeId=0162468rH3YTL  
C00M93426](http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=MCF548X&nodeId=0162468rH3YTLC00M93426)

- **La memoria**

Debido a que es el código quien se convierte en el factor determinante en un desarrollo con microcontrolador, la cantidad de memoria dispuesta en este es un factor importante al momento de elegir la máquina.

La memoria generalmente es interna, pero existen máquinas con manejo de memoria externa y pueden acceder a varios MB, tanto en memoria de código como datos.

Es bien sabido que los sistemas soportados por un microcontrolador no requieren de grandes cantidades de memoria, tanto de código como de datos (FLASH-RAM). En este sentido los microcontroladores vienen equipados con memoria de código de hasta 1MB y de datos de hasta 128KB.

- **Los módulos o periféricos**

Gran variedad de periféricos orientados a los diferentes mercados tecnológicos como: las telecomunicaciones, el transporte, aparatos eléctricos de consumo masivo, entre otros; son implementados en los microcontroladores. Los usuarios encontrarán una gran variedad de periféricos, siendo los más comunes:

- Temporizadores (PWM, OUTPUT COMPARE, INPUT CAPTURE)
- Convertidores análogo a digital
- Entradas/salidas de propósito general
- Reloj de tiempo real
- Sistemas de protección de flujo de programa (WDT, COP)
- Puertos de comunicación serial asíncrona (UART, CAN)
- Puertos de comunicación serial sincrónica (IIC, SPI)
- Bus universal de comunicación serial (USB)
- Puerta trasera de depuración (BDM, JTAG)

Algunas máquinas de mayor desempeño involucran módulos más especializados como:

- Controladores para ETHERNET
- Unidades de generación y aceleración criptográfica
- Unidades de generación de números aleatorios
- Unidades de generación y verificación de código de redundancia cíclica
- Unidades de tratamiento de aritmética flotante (FPU)
- Unidades de multiplicación, acumulación y corrimiento (MAC)
- Unidades para manejo directo de memoria (DMA)

- **Lenguajes de programación**

Los lenguajes más populares para la programación de microcontroladores, en su orden de importancia, son:

- **Assembler:** Toda máquina deberá soportar programación en su lenguaje nativo, que se acerque de manera directa a su núcleo. Este lenguaje es llamado *assembler* y no debe ser ignorado por ningún programador, aún teniendo el mejor compilador en otros lenguajes. La razón de lo anterior radica en la posibilidad de desarrollar código con la mejor eficiencia, esto es, mayor velocidad, mayor aprovechamiento del recurso de memoria y más bajo consumo.

Afortunadamente muchas aplicaciones permiten la mezcla de *assembler* y otros lenguajes de mayor nivel, proporcionando al usuario la posibilidad de hacer código óptimo.

- **C/C++:** Por excelencia, es el lenguaje de la ingeniería debido a su estructura, portabilidad y aprovechamiento de los recursos de las máquinas procesadoras (CPU, MCU). Los compiladores de este lenguaje son cada vez más óptimos, de tal forma que acortan la brecha entre el lenguaje nativo de la máquina y el C/C++.
- **Basic:** Aunque no es muy popular para sistemas embebidos a microcontrolador, presenta una manera cómoda para programar los MCU sin la necesidad de mucho conocimiento de la máquina. La desventaja radica en el alto consumo de los recursos, por tratarse de un lenguaje de alto nivel y adicionalmente la pérdida del enfoque de la máquina que se está programando.

## 1.2. EL FUTURO

La evolución de los microcontroladores está ligada a la evolución de los microporcesadores y ha sido caracterizada por aspectos como el incremento de la velocidad de ejecución, una disminución del consumo de energía, un alto grado de miniaturización, un incremento en la capacidad de manipulación del dato y un aumento en la capacidad implementada de dispositivos de memoria y periféricos.

El futuro prevé una migración acelerada de las máquinas de 8 bits y 16, a las máquinas de 32 bits y superiores. El factor predominante es el fenómeno de mercado, que hace que la diferencia de precio entre la capacidad en bits sea cada día menor. De esta manera los usuarios se enfrentan a la tentación de tener más prestaciones por un precio razonable.

### **1.3. REFERENCIAS**

- Stallings, William. Organización y Arquitectura de Computadores. Ed. Prentice Hall, 5º ed. 2000.
- <http://www.intel.com/cd/corporate/techtrends/EMEA/spa/209840.htm>

### **1.4. PREGUNTAS**

- Realice una definición corta y clara para un microcontrolador.
- ¿Cuales son las capacidades en bits para los microcontroladores mencionados en este texto?
- ¿Por qué es importante el concepto de bajo consumo en un microcontrolador?
- ¿Cuál es la razón de la baja cantidad de memoria implementada en un microcontrolador?
- Enuncie tres módulos especializados que traen algunos microcontroladores.
- ¿Por qué no son muy populares los lenguajes de alto nivel para programar microcontroladores?
- Enuncie una razón de peso para migrar a máquinas de más alto número de bits.

# CAPÍTULO 2

## Arquitecturas

**2.1. Von Newman**

**2.2. Harvard**

**2.3. Otros conceptos y otras arquitecturas**

**2.4. Referencias**

**2.5. Preguntas**

## 2.1. VON NEWMAN

En el año de 1903 nace en Hungría Jhon Von Newman, uno de los más brillantes matemáticos de la era de la computación. Durante la Segunda Guerra Mundial participó como asesor en la construcción de la computadora ENIAC y más tarde de la UNIVAC, siendo su gran aporte el del concepto de programa almacenado o micro-código. A este señor (ver Figura 2.1) se debe la arquitectura en la que se fundamentan la mayoría de los microprocesadores y microcontroladores actuales.

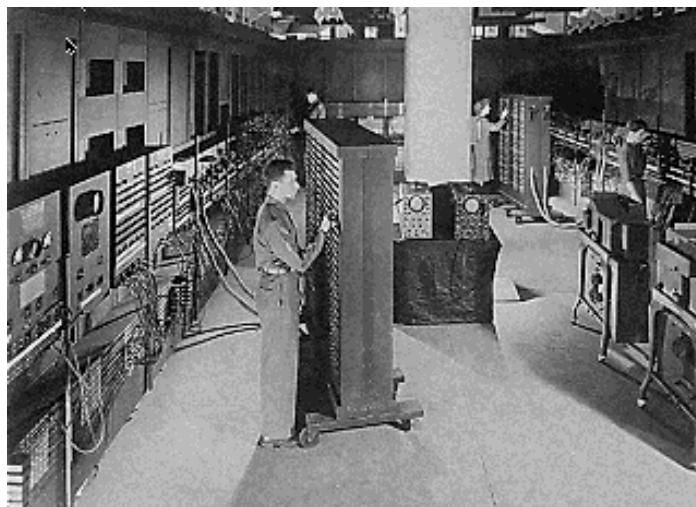


**Figura 2.1. El matemático Jhon Von Newman**

(Fuente: [http://www.dma.eui.upm.es/historia\\_informatica/Doc/personajes.htm](http://www.dma.eui.upm.es/historia_informatica/Doc/personajes.htm))

ENIAC (*Electronics Numerical Integrator and Calculator*), como aparece en la Figura 2.2, fue la primera computadora programable de propósito general, concebida para cálculos de balística para la Segunda Guerra Mundial. La computadora se terminó de desarrollar en 1946 cuando la guerra había pasado.

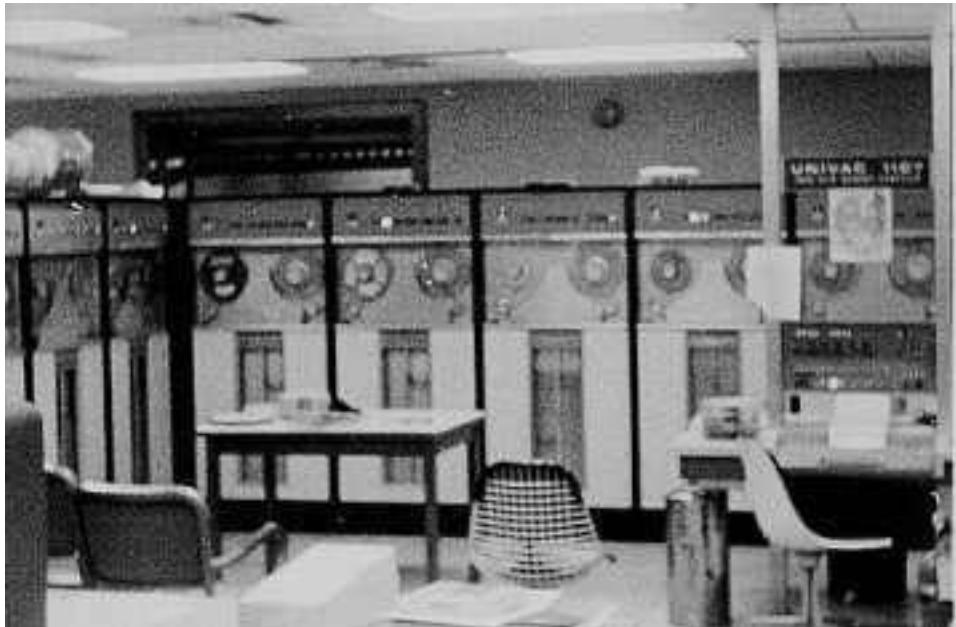
Esta máquina pesaba 30 toneladas, la base operativa era decimal, usaba 800 kilómetros de cable, tenía 17.000 tubos de vacío, realizaba 100.000 operaciones por segundo y consumía 150KWh.



**Figura 2.2. Computadora ENIAC**

(Fuente: <http://ei.cs.vt.edu/~history/ENIAC.Richey.HTML>)

La primera máquina en donde se utilizó el lenguaje *assembler* y el concepto de Programa Almacenado (desarrollado por Von Neuman) fue UNIVAC (*Universal Automatic Computer*), ver Figura 2.3. El concepto de Programa Almacenado se refiere a las instrucciones que ejecutará la máquina y que es llevado a la memoria de código.



**Figura 2.3. Computadora UNIVAC**

(Fuente: [http://www.dma.eui.upm.es/historia\\_informatica/Doc/personajes.htm](http://www.dma.eui.upm.es/historia_informatica/Doc/personajes.htm))

Estas máquinas eran programadas en lenguaje de máquina, que sería reemplazado más tarde por el lenguaje *assembler*, que elimina la incómoda y tediosa labor de digitar los programas en binario. Esta máquina fue usada en el censo de población de los Estados Unidos y utilizaba memoria magnética en cinta.

Aquí es importante mostrar la definición de Von Neuman, para entender como sus ideas han perdurado por todos estos años. Toma do textualmente de algunos apartes<sup>4</sup> sobre el diseño de Von Newman:

2.2. Primero: Como el dispositivo es, principalmente, un computador, tendrá que realizar las operaciones aritméticas elementales muy frecuentemente. Estas son: la suma, la resta, la multiplicación y la división: +, --, x, /. Es, por tanto, razonable, que contenga elementos especializados sólo en estas operaciones.

Debe observarse, sin embargo, que aunque este principio parece consistente, la manera específica de cómo se aplica requiere un examen cuidadoso [...]. En cualquier caso, tendrá que existir la parte de aritmética central que constituirá la primera parte específica: CA (*Central Arithmetical*).

---

<sup>4</sup> Tomado de las notas del matemático Jhon Von Newman.

2.3 Segundo: El control lógico del dispositivo, es decir, la secuenciación adecuada de las operaciones, debe ser realizado eficientemente por un órgano central de control. Si este dispositivo de control debe ser versátil, es decir, servir en lo posible para todo uso, hay que diferenciar entre las instrucciones específicas que definen un problema particular, y los órganos de control general que se ocupan de que se lleven a cabo dichas instrucciones, sean cuáles sean. Las primeras deben almacenarse en algún lugar; las otras deben representarse definiendo partes operativas del dispositivo. Con el control central nos referimos sólo a esta última función, y los órganos que la realizan forman la segunda parte específica: CC (*Central Control*).

2.4 Tercero: Cualquier dispositivo que realice secuencias largas y complicadas de operaciones (concretamente de cálculo), debe tener una memoria considerable [...].

Las instrucciones que gobiernan un programa complicado pueden constituir un material considerable, sobre todo si el código es circunstancial (lo cual ocurre en la mayoría de las situaciones). Este material debe tenerse en cuenta [...].

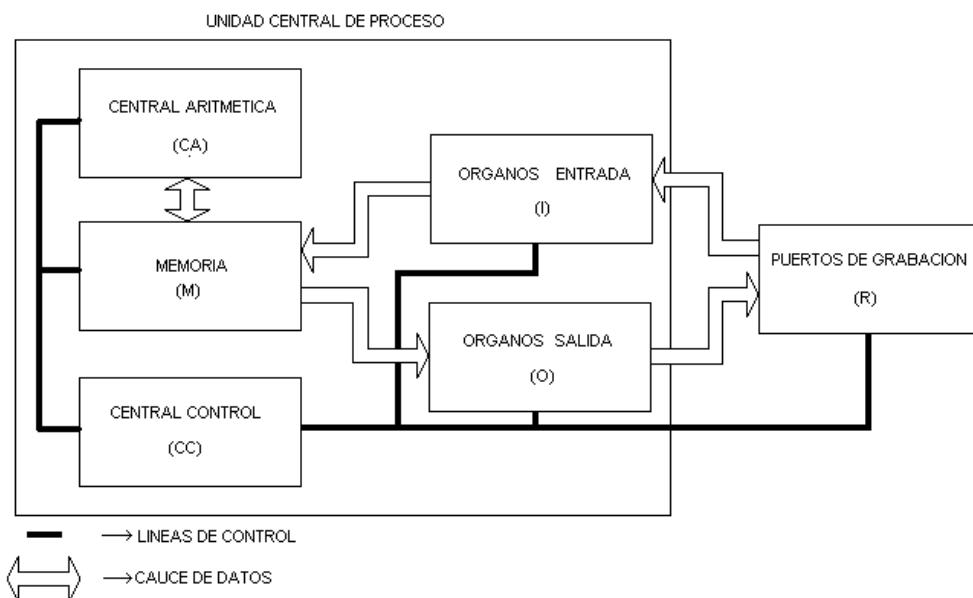
En cualquier caso, la memoria total es la tercera parte específica del dispositivo: M (*memory*).

2.6 Las tres partes específicas, CA, CC y M, corresponden a las neuronas asociativas del sistema nervioso humano. Quedan por discutir los equivalentes a las neuronas sensoriales o aferentes y las motoras o eferentes. Estos son los órganos de los dispositivos de entrada y salida [...].

El dispositivo tiene que estar dotado con la habilidad de mantener contacto de entrada y salida (sensorial y motor) con medios específicos de este tipo (cf.1.2): el medio será llamado el medio de grabación exterior del dispositivo: R (*Recording*) [...].

2.7 Cuarto: El dispositivo tiene que tener órganos para transferir [...] información a partir de R a sus partes específicas C y M, ver figura 1.7. Estos órganos forman su entrada, la cuarta parte específica: I (*Input*). Veremos que lo mejor es hacer todas las transferencias a partir de R (mediante I) hasta M, y nunca directamente a partir de C [...].

De este diseño aparecen los primeros diagramas de bloques que conformarían la máquina Von Newman (Figura 2.4).



**Figura 2.4 Arquitectura primaria Von Newman.**

Continúa Von Newman:

2.8 Quinto: El dispositivo tiene que tener órganos para transferir [...] información a partir de sus partes específicas C y M hacia R. Estos órganos forman su salida, la quinta parte específica: O (Output). Veremos que es mejor, de nuevo, hacer todas las transferencias a partir de M (mediante O) a R, y nunca directamente a partir de C [...].

Es claro que Von Newman define, de manera nuclear y salvo raras excepciones, la arquitectura de todos los computadores de hoy en día, la Figura 2.5. presenta de una manera más evolucionada el diseño del matemático.

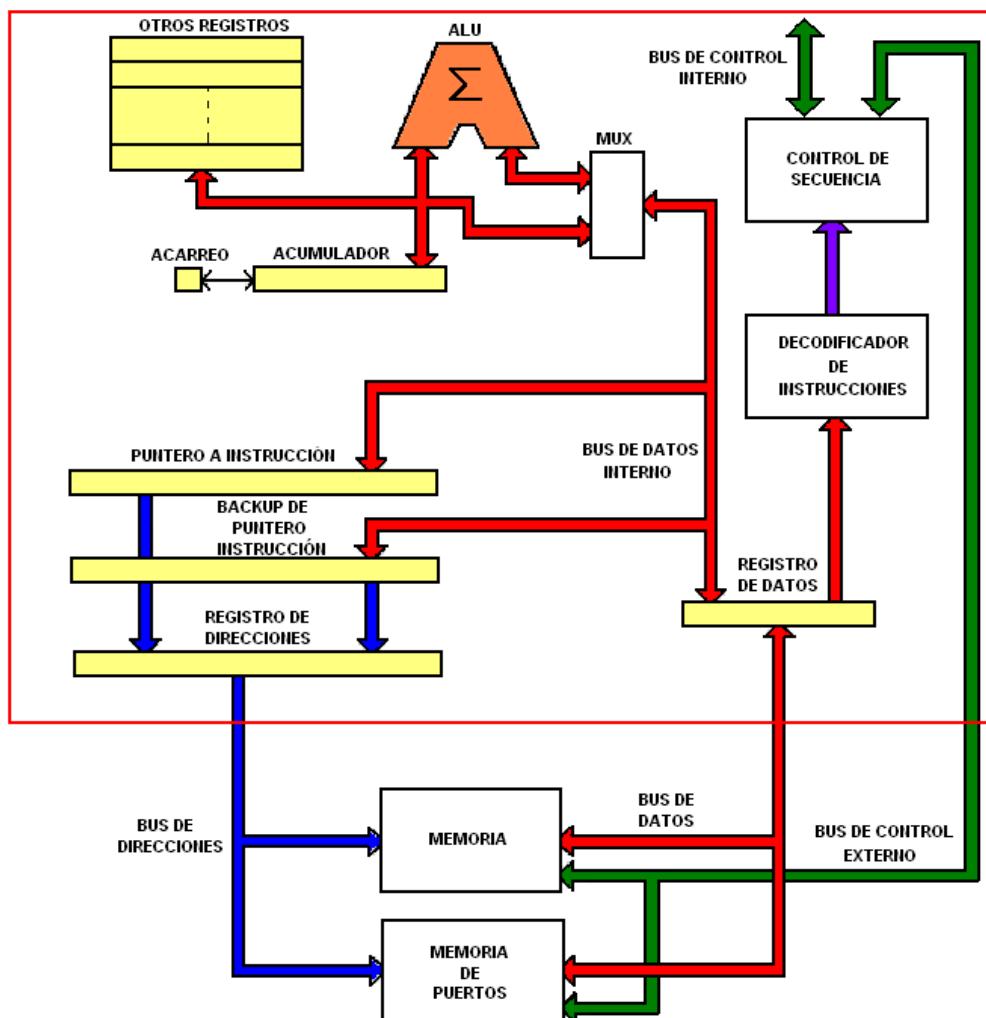


Figura 2.5. Arquitectura Von Newman evolucionada.

## 2.2. HARVARD

La arquitectura Harvard diseñada por el señor Howard Aiken, graduado de la Universidad de Harvard y desarrollada en 1944 durante el proyecto Mark I, tiene como característica más importante tener los datos y las instrucciones en buses separados. De esta manera se puede ejecutar en forma paralela una instrucción con su respectivo dato, haciendo más corto el tiempo de ejecución.

El núcleo de la CPU está conectado a dos memorias por intermedio de dos buses separados. Una de las memorias contiene solamente las instrucciones del programa, y es llamada Memoria de Programa. La otra memoria sólo almacena los datos y es llamada Memoria de Datos (Figura 2.6). Ambos buses son totalmente independientes y pueden ser de distintos tamaños.

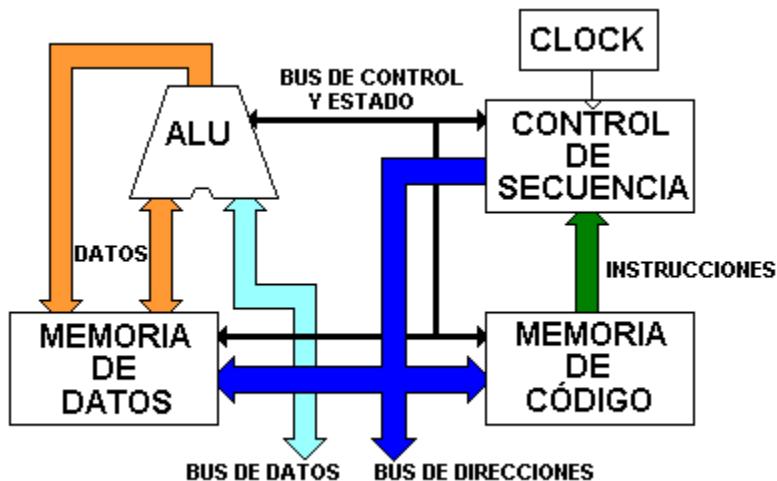


Figura 2.6. Arquitectura Harvard

Para un procesador implementado sobre tecnología Harvard, bajo RISC (*Reduced Instrucción Set Computer*), el set de instrucciones y el bus de la memoria de programa pueden diseñarse de manera tal que todas las instrucciones tengan la misma longitud y puedan ser ejecutadas en un ciclo de máquina.

Además, como los buses son independientes, la CPU puede estar accediendo a los datos para completar la ejecución de una instrucción, y al mismo tiempo estar leyendo la próxima instrucción a ejecutar. Se puede observar claramente que las principales ventajas de esta arquitectura son:

- El tamaño de las instrucciones no esta relacionado con el de los datos, y por lo tanto puede ser optimizado para que cualquier instrucción ocupe una sola posición de memoria de programa, logrando así mayor velocidad y menor longitud de programa.
- El tiempo de acceso a las instrucciones puede superponerse con el de los datos, logrando una mayor velocidad de operación.

Una pequeña desventaja de los procesadores con arquitectura Harvard, es que deben poseer instrucciones especiales para acceder a tablas de valores constantes que pueda ser necesario incluir en los programas, ya que estas tablas se encontrarán físicamente en la memoria de programa (por ejemplo en la EPROM de un microprocesador).

## 2.3. OTROS CONCEPTOS Y OTRAS ARQUITECTURAS

En la búsqueda de ejecutar procesos cada vez más rápidos, programas potentes cargados de multiplicidad de funciones, manipulación de grandes volúmenes de información, bajo consumo de energía y ocupación de pequeños espacios, las compañías que desarrollan microprocesadores y microcontroladores están permanentemente buscando nuevas arquitecturas. Es de anotar que las arquitecturas de actualidad tienen su fundamento en la arquitectura madre, la arquitectura Von Newman, y en los elementos más importantes de la arquitectura Harvard.

- **Conceptos de CISC y RISC**

Se hace necesario mencionar dos conceptos bastante relacionados con las arquitecturas de actualidad, como lo son RISC y CISC, en donde:

- **RISC (Reduced Instruction Set Computer):** Computador con juego reducido de instrucciones. Generalmente este tipo de instrucciones son ejecutadas en un ciclo de la máquina, utilizando modos de direccionamiento simples e instrucciones sencillas. El concepto de segmentación (ejecución de varias instrucciones en el mismo ciclo de máquina) es más fácil aplicarlo a las instrucciones RISC debido a que éstas tienen un ancho en bits constante.
- **CISC (Complex Instruction Set Computer):** Computador con juego complejo de instrucciones. Un procesador cuyo núcleo está basado en el concepto CISC en su repertorio de instrucciones, no necesita de compiladores costosos ni complejos en mejora de sus prestaciones. El enfoque de este concepto es el desarrollo de lenguajes de alto nivel (HLL: *High Level Language*). Este concepto también configura programas más cortos y de mejor aprovechamiento de la memoria. Se ha visto que algunos algoritmos, de moderada y alta complejidad, se desarrollan mucho más rápido en una máquina CISC que en una RISC.

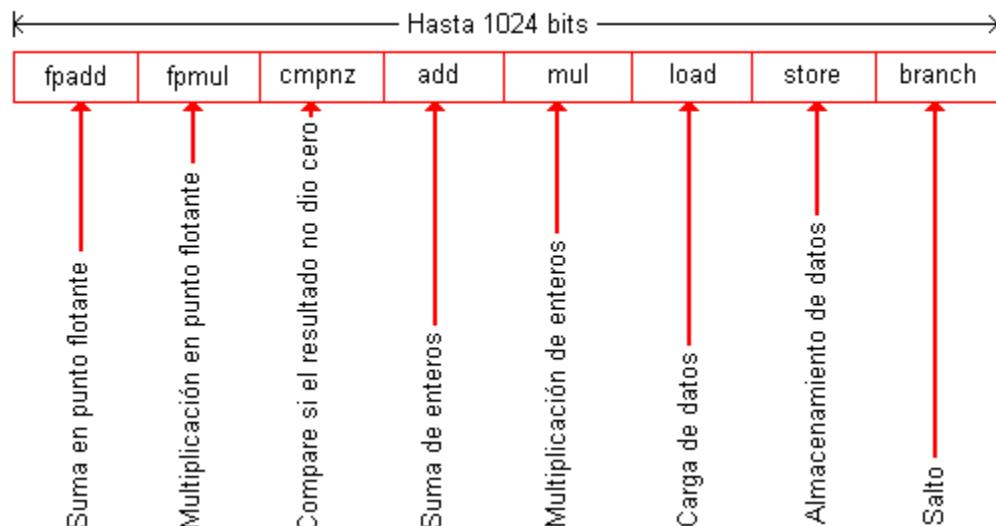
Es frecuente encontrar un sin número de artículos que hablan bien o mal de RISC y de CISC, pero a estas alturas se concluye que lo mejor es tomar lo bueno de un concepto y del otro, como lo han hecho muchos fabricantes de microprocesadores y que han tenido gran éxito.

- **Concepto VLIW (Very Large Instruction Word)**

Acogiéndose más al concepto RISC, en cuanto al tamaño constante de sus instrucciones, pero tomando elementos como el de realizar tareas complejas, propias

de la técnica CISC, aparece la arquitectura VLIW con instrucciones cuyo ancho va desde los 64 bits hasta los 1024 bits.

El formato de una instrucción VLIW se muestra en la Figura 2.7, en donde se puede ver la ejecución de varios procesos al mismo tiempo.



**Figura 2.7 Formato de instrucción VLIW**

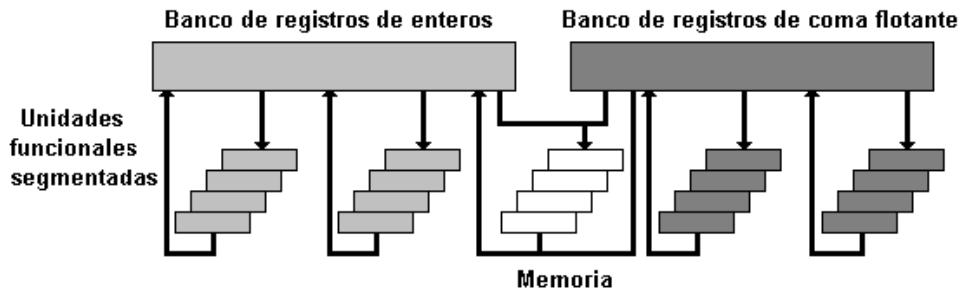
A continuación se hace una breve descripción de las arquitecturas que actualmente se están aplicando a las máquinas modernas y de otras que aún se encuentran en etapa de desarrollo:

- **Concepto de arquitectura superescalar**

El fundamento del proceso superescalar, nacido en el año 1987 y derivado del procesamiento normal escalar, es el canal o cauce de instrucciones (*PIPELINE*) y toma elementos del concepto RISC, pero también es posible implementarla sobre el concepto CISC.

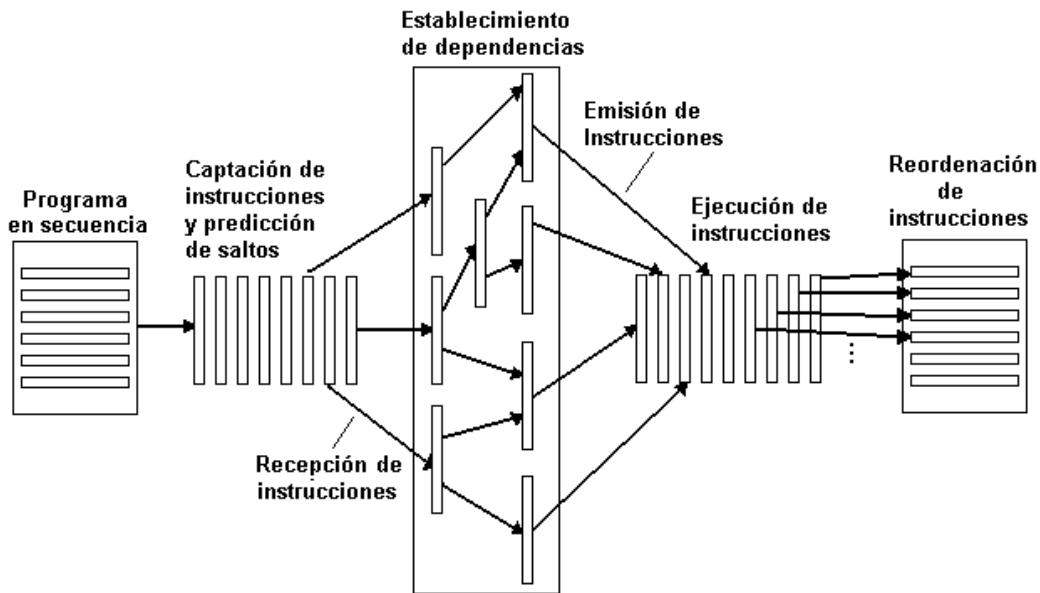
Un cauce de instrucciones consta de múltiples etapas de proceso distribuidas a lo largo de los ciclos de máquina, en donde varias instrucciones pueden ser ejecutadas a la vez. Si existen varios cauces de instrucciones se puede concluir que existe cierto nivel de ejecución en paralelo (ver Figura 2.8).

Para lograr esta técnica es necesario que el sistema, en tiempo de compilación, organice el orden en el que deben ser ejecutadas las instrucciones (ver Figura 2.9) y de esta forma evitar las dependencias entre instrucciones.



**Figura 2.8. Arquitectura superescalar.**

(Fuente: Stallings, William. Organización y arquitectura de computadores. 5ta ed. 2000)



**Figura 2.9. Ordenamiento de instrucciones para ejecución óptima.**

(Fuente: Stallings, William. Organización y arquitectura de computadores. 5ta ed. 2000)

Lo anterior quiere decir, que si la entrada de una instrucción depende de la salida de una instrucción precedente, la segunda instrucción se ve amarrada y no podría ejecutarse hasta tanto no se procesen los datos que requieren.

**Ejemplo:** Listado de un código ejecutado en una máquina tradicional Harvard:

- Paso 1:** sume los registros R1, R2 y almacene el resultado en R3
- Paso 2:** substraiga R3 de R4 y almacene el resultado en R5
- Paso 3:** sume los registros R6, R7 y almacene el resultado en R8
- Paso 4:** almacene R1 en R9.

Para una máquina de procesamiento superescalar, el ejemplo anterior en arquitectura RISC, se podría ejecutar así:

**Paso 1:** (las siguientes instrucciones se hacen simultáneamente)

- Sume los registros R1, R2 y almacene el resultado en R3.
- Sume los registros R6, R7 y almacene resultado en R8.
- Almacene R1 en R9.

**Paso 2:** Substraiga R3 de R4 y almacene el resultado en R5.

Se puede ver como las operaciones de los pasos 1, 3 y 4 se pueden ejecutar simultáneamente, debido a que no existe interdependencia entre ellas.

Una explicación gráfica del proceso *pipeline* simple y superescalar se presenta en la Figura 2.10.

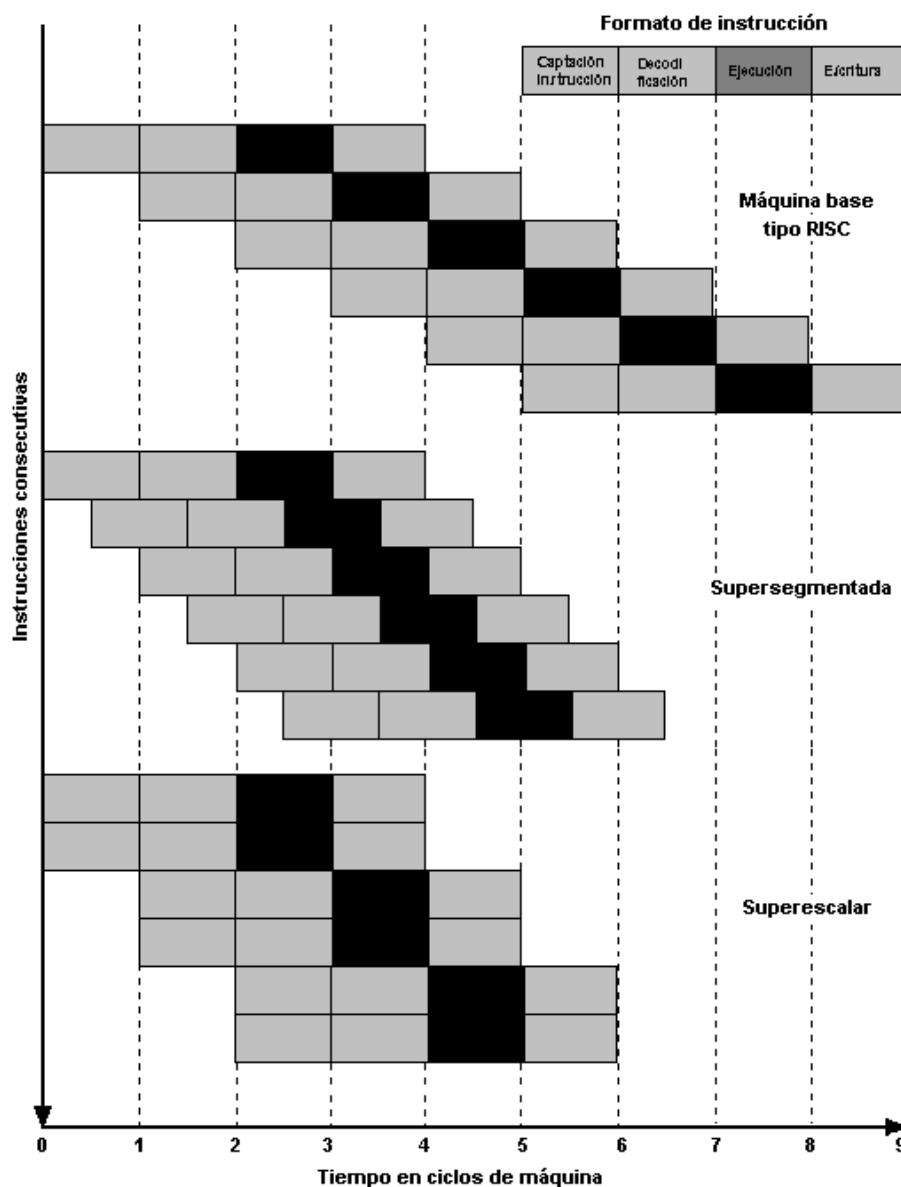
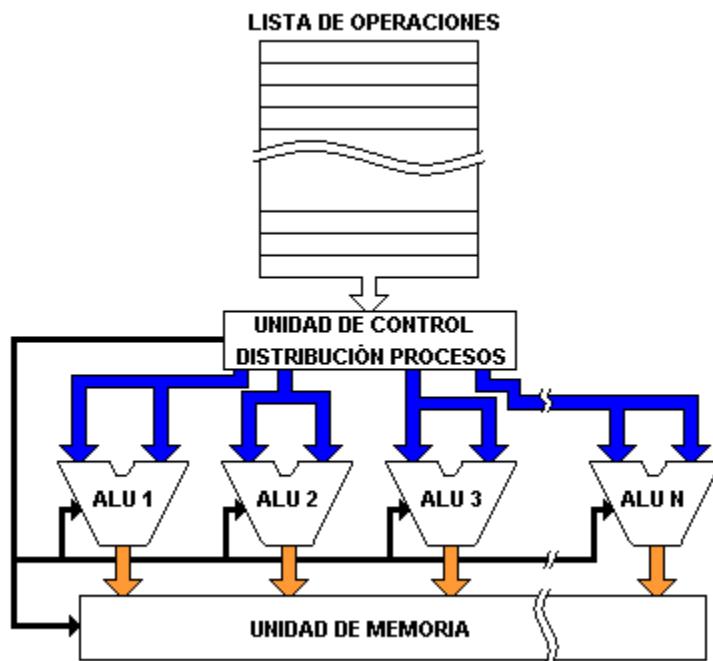


Figura 2.10. Comparación de procesos RISC, superescalar y supersegmentazo [1].

- **Concepto de procesadores vectoriales o paralelos**

La característica más importante de un procesador vectorial es la de contener varias ALU con sus respectivos cauces de datos, y con el empleo de una sola unidad de control.

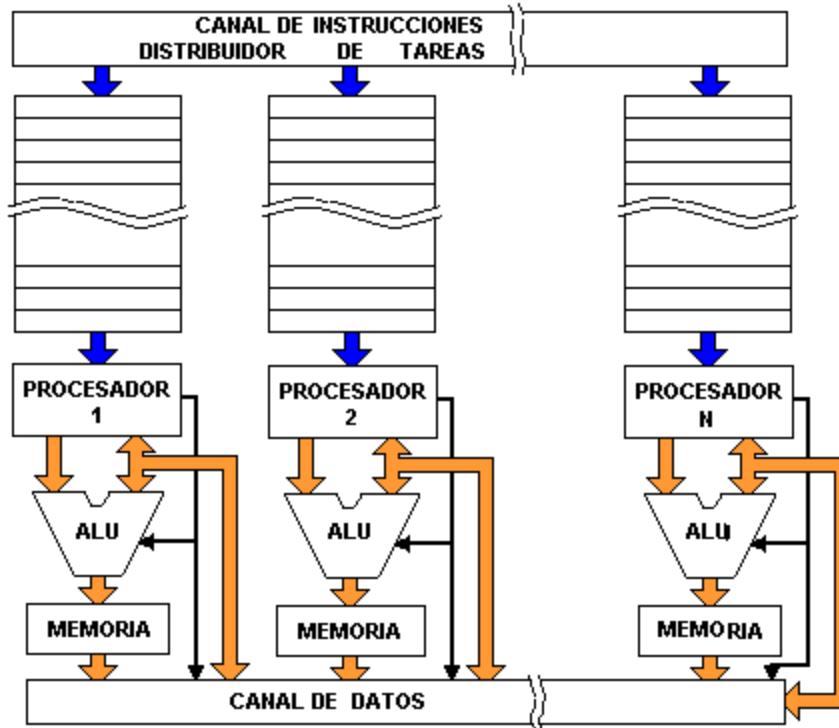
El controlador encausa varias operaciones hacia las diferentes ALU, para que éstas sean ejecutadas en paralelo. Si a lo anterior se le aplica la segmentación de instrucciones, se obtendrían máquinas con operaciones muy potentes y de velocidades altas (ver Figura 2.11).



**Figura 2.11. Procesador vectorial.**

La técnica vectorial de mayor actualidad es la de implementar sistemas con varios microprocesadores en paralelo, para esto se recomienda dividir los programas en subtareas, que se ejecutarán en paralelo.

La técnica anterior va acompañada de una sincronización muy eficiente, que coordina cada tarea (ver Figura 2.12).



**Figura 2.12. Procesador vectorial con varios procesadores paralelos**

- **Conceptos de clasificación SISD, SIMD, MISD y MIMD**

Existe una clasificación más evolucionada de las arquitecturas y está basada en el cauce de las instrucciones y de cómo se manipulan los datos. Estos conceptos aparecen en el año de 1966 y fueron propuestos por el señor Michael Flynn<sup>5</sup>.

- **SISD (Single Instruction Single Data)**

Reúne todas las arquitecturas que ejecutan una sola instrucción por ciclo de máquina y operan sobre un único dato.

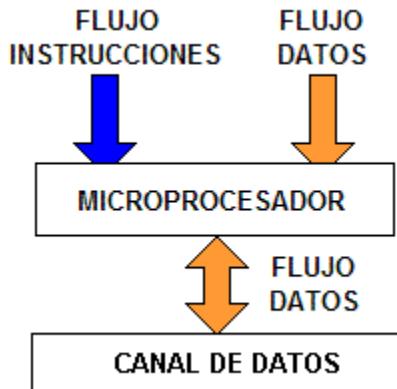
Dentro de esta clasificación se encuentran la mayoría de los procesadores de la década comprendida entre los años 1980-1990, y en la actualidad los microprocesadores que se encuentran en el interior de microcontroladores de aplicación industrial.

En esta clasificación podemos incluir los primeros PC, estaciones de trabajo y los conceptos de segmentación, superescalares y prebúsqueda de instrucciones.

La Figura 2.13 muestra un diagrama en bloques del flujo operativo de una máquina SISD típica.

---

<sup>5</sup> Concepto conocido como Taxonomía de Flynn.

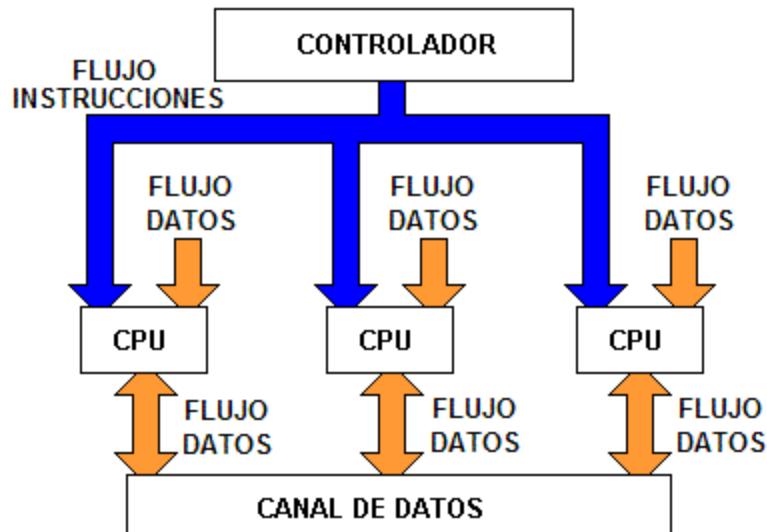


**Figura 2.13. Procesador SISD.**

- **SIMD (Single Instruction Multiple Data)**

Dentro de esta arquitectura encajan la mayoría de las máquinas actuales. El método consiste en aplicar una instrucción a un grupo de datos, como por ejemplo movimientos de datos de memoria a memoria, operaciones sobre vectores o matrices de datos, entre otras.

En esta clasificación podemos involucrar los computadores vectoriales y matriciales (ver Figura 2.14).



**Figura 2.14. Procesador SIMD.**

- **MISD (Multiple Instruction Single Data)**

Es la menos popular y su escasa aplicación se orienta a procesos, en donde sobre un único dato se aplican multiplicidad de operaciones como en las matrices sistólicas (ver Figura 2.15).

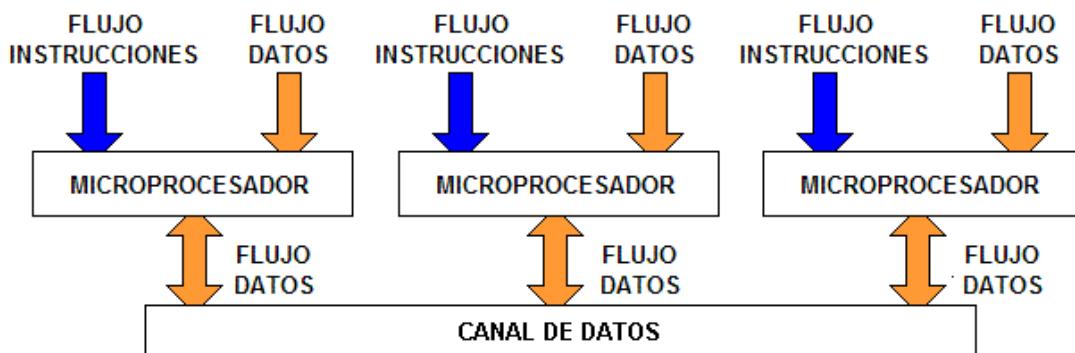


**Figura 2.15. Procesador MISD.**

- **MIMD (Multiple Instruction Multiple Data)**

Este tipo de arquitectura se está afianzando cada vez más y está íntimamente ligada a procesadores paralelos. La técnica consiste en hacer múltiples operaciones sobre grandes cantidades de datos como se hace en sistemas de memoria compartida y memoria privada (ver Figura 2.16).

La técnica MIMD multiplica recursos dentro de un procesador o utiliza varios de manera paralela.



**Figura 2.16. Procesador MIMD.**

- **Predicción y ejecución especulativa**

Un predictor de saltos es un mecanismo *hardware* utilizado en los procesadores que utilizan segmentación de la unidad de proceso, para reducir ciclos de parada en el *pipeline*.

Los saltos condicionales introducen retardo en los procesadores, ya que normalmente no se evalúa la condición del salto hasta pasadas varias etapas. Lo anterior hace que se tenga que parar el cauce o que se puedan introducir instrucciones en el *pipeline*, que no deben de ser ejecutadas. Teniendo que convertirse posteriormente en instrucciones de no operación (NOP) y decrementando así el rendimiento del procesador.

La predicción es posible anotando el comportamiento del programa en saltos anteriores y tratando de aprender la ruta del flujo de programa, para aplicar métodos binarios o estadísticos y así tomar la decisión correcta del salto.

La ejecución especulativa es la ejecución de código por parte del procesador, que no tiene por qué ser necesariamente *a priori*.

La ejecución especulativa no es más que una optimización y sólo es útil cuando la ejecución previa requiere menos tiempo y espacio que el que requeriría la ejecución posterior. Este ahorro es lo suficientemente importante como para compensar el esfuerzo gastado en caso de que el resultado de la operación nunca llegue a usarse.

Los procesadores modernos, que hacen uso de un *pipeline*, usan la ejecución especulativa, para reducir el costo computacional de las instrucciones de salto condicional. Cuando el procesador se encuentra con una instrucción de salto condicional, el procesador intenta predecir en donde es más probable que se salte (tema visto en párrafos anteriores) e inmediatamente comienza a ejecutar el código que empieza en esa área.

Si *a posteriori* se demuestra que la predicción fue errónea, todo lo ejecutado después del salto se descarta. Esta ejecución prematura sale realmente rentable, en el momento en que se evite que el *pipeline* se detenga para conocer cual sería la próxima instrucción a ejecutar.

## **2.4. REFERENCIAS**

- [1] Stallings, William. Organización y Arquitectura de Computadores. Ed. Prentice Hall, 5º ed. 2000.

## **2.5. PREGUNTAS**

- ¿A quién se le considera el Padre de la Computación?
- Describa el concepto de Programa Almacenado.
- Dibuje un diagrama de bloques que represente la máquina de Von Newman.
- ¿Cuál es la diferencia fundamental entre una máquina con arquitectura Von Newman y arquitectura Harvard?
- Para las arquitecturas Harvard y Von Newman, ¿cuál sería la afinidad con los conceptos CISC y RISC?
- Enuncie dos características importantes de la arquitectura VLIW.
- ¿Qué es un cauce de instrucciones (*pipeline*)?
- Defina brevemente SISD, SIMD, MISD y MIMD.

# CAPÍTULO 3

## Modos de Direcciónamiento y Tipos de Instrucciones

- 3.1. Los direccionamientos
- 3.2. Lenguaje *assembler*
- 3.3. Tipos de instrucciones
- 3.4. Directivas y/o pseudo-instrucciones
- 3.5. Referencias
- 3.6. Preguntas

### 3.1. MODOS DE DIRECCIONAMIENTO

Los diferentes modos de direccionamiento que se explicarán en éste aparte no hacen referencia a ninguna máquina en particular y se utilizará un *assembler* genérico. Aquí, se tratarán de explicar la mayoría de direccionamientos para las máquinas existentes en el mercado.

También como norma para el usuario, el formato de las instrucciones usadas para los diferentes tipos de direccionamiento será:

**ETIQUETA: INSTRUCCIÓN FUENTE,DESTINO ;COMENTARIO**

A la dirección resultante de un direccionamiento se le llamará la **Dirección Efectiva** (DE), también llamada dirección resultante del direccionamiento.

- **Direccionamiento de registro**

Se utiliza para operaciones o movimientos de información entre registros de la CPU, por ejemplo:

MOVER R1 , R0 ;Mueva al registro R0 el contenido del  
;registro R1 (ver Figura 3.1)



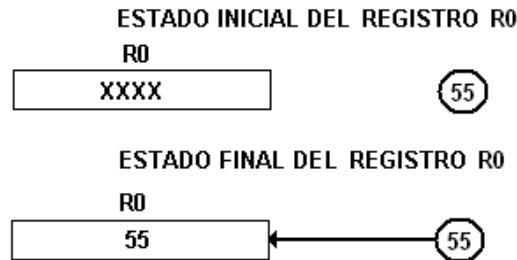
**Figura 3.1. Direccionamiento de registro.**

- **Direccionamiento inmediato**

Cuando se requiera operar o almacenar una cantidad constante en registros de la CPU, localidades de memoria o dispositivos periféricos.

MOVER #55 , R0 ;Almacene la constante 55 en el registro R0.  
;El símbolo “#” indica que la cantidad 55 es  
;una cantidad constante (ver Figura 3.2).

**NOTA:** Existen muchas formas de representar una constante y depende del ensamblador que se utilice. Algunos no anteponen un símbolo especial, para indicar que una cantidad es una constante (Ejm: A9H, para indicar la constante hexadecimal A9).



**Figura 3.2 Direccionamiento inmediato**

El error más común en este tipo de direccionamiento es tratar de almacenar en una constante algún tipo de información, por ejemplo:

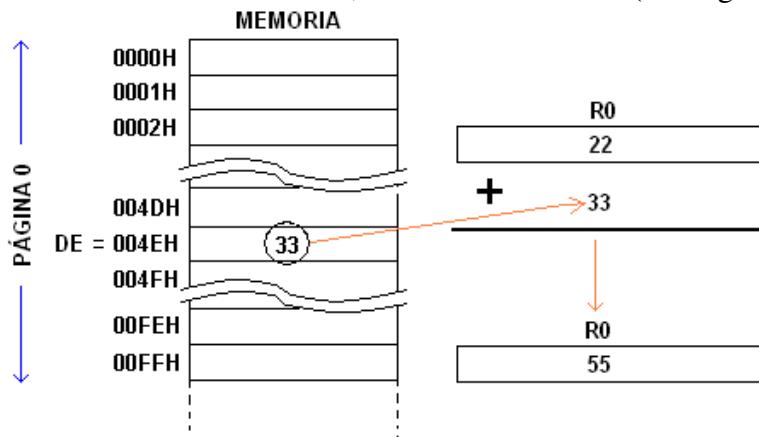
MOVER R5 , #0x1C78 ;Error: el contenido del registro R5 no puede ser almacenado dentro de una constante

- **Direccionamiento directo a registro**

En algunas máquinas se limita éste direccionamiento a la página cero del mapa de memoria. La página cero es el área de la memoria comprendida entre la dirección 00H hasta la dirección FFH, es decir, las primeras 256 celdas.

El direccionamiento consiste en intercambiar u operar información entre los registros de la máquina y la memoria. El direccionamiento directo es, por excelencia, uno de los más rápidos en ejecución que tienen las máquinas. Por ejemplo:

ADICIONE (0x004E) , R2 ;Sume el contenido del registro R2 y el contenido  
;de la celda 004E y el resultado almacénelo en el  
;registro R2. Los "( )" indican que la cantidad  
;0x004F no es una constante, sino una dirección de  
;la memoria. Es importante entender que la  
;dirección 0x004E pertenece a la página 0 de la  
;memoria del sistema (ver Figura 3.3).



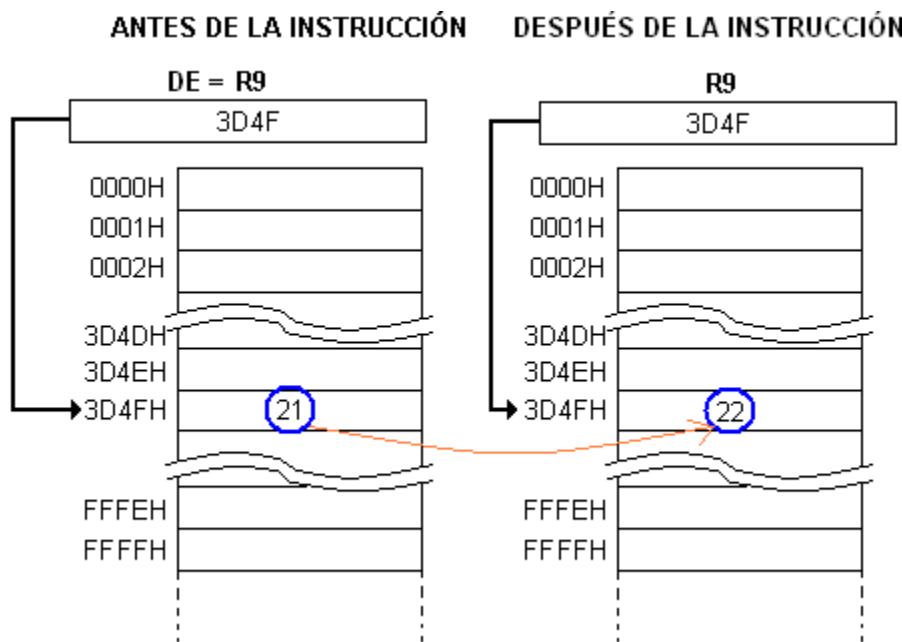
**Figura 3.3. Direccionamiento directo a registro**

- **Direccionamiento indirecto**

En éste direccionamiento se utiliza un registro como apuntador a la localidad de memoria o dispositivo E/S a ser accesado. La ventaja del direccionamiento es la movilidad del puntero mediante operaciones matemáticas de incremento y decremento sobre éste.

La simbología que distingue este direccionamiento es la de encerrar al puntero entre paréntesis "( )". Por ejemplo:

INCREMENTE (R9) ;Incremente el contenido de la celda apuntada por el ;puntero R9 (ver Figura 3.4).



**Figura 3.4. Direccionamiento indirecto.**

El error más común en este tipo de direccionamiento es olvidar los paréntesis, por ejemplo:

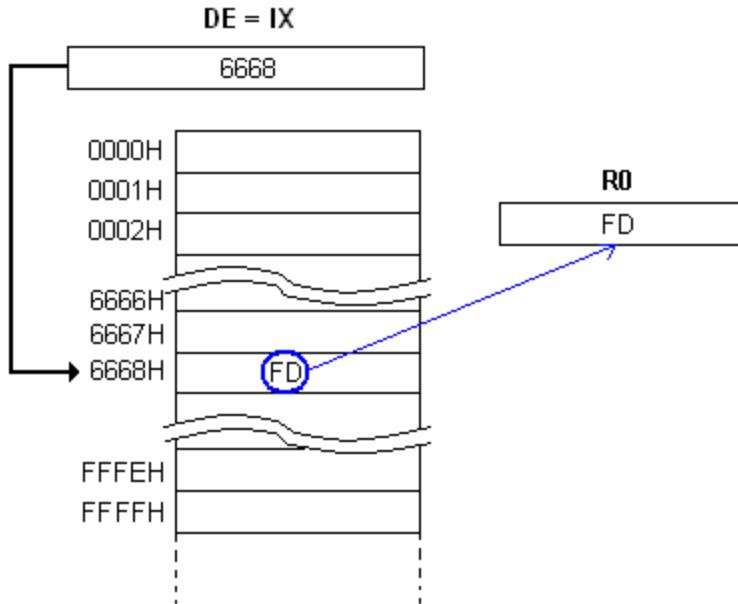
DECREMENTE R9 ;Error: si lo que se quiere es decrementar el ;contenido de la celda apuntada por R9, la ;instrucción está decrementando el contenido del ;registro R9.

- **Direccionamiento indexado**

Todo tipo de direccionamiento indexado utiliza uno de los registros como puntero índice. Existen diferentes modos del direccionamiento indexado y a continuación se presentan:

- **Indexado sin desplazamiento (offset):** Hace referencia al contenido de la localidad de memoria apuntada por el registro índice en particular sin adicionar ningún desplazamiento, por ejemplo:

ALMACENE (IX) , R0 ;Almacene en el registro R0 el contenido de  
;la localidad de memoria apuntada por el  
;índice IX (ver Figura 3.5).



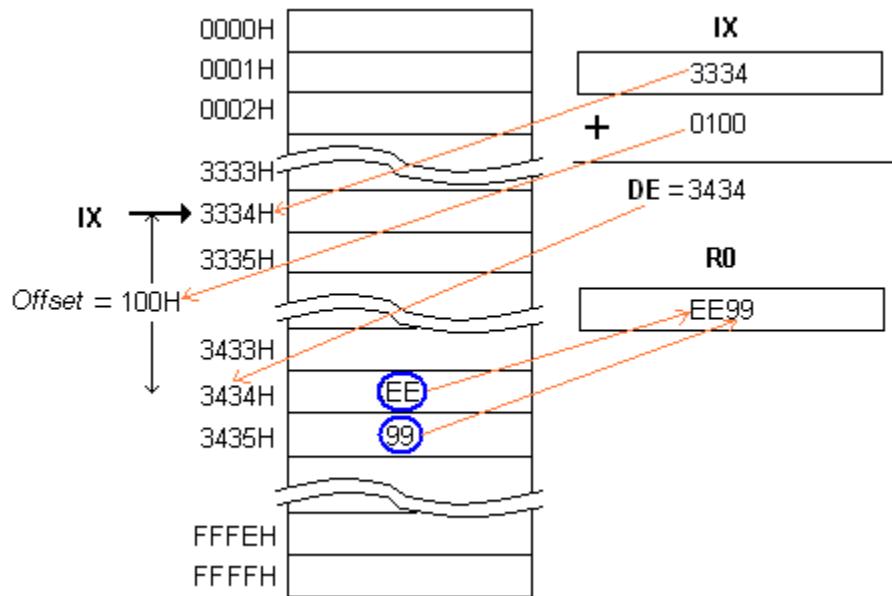
**Figura 3.5. Direccionamiento indexado sin offset.**

- **Indexado con desplazamiento – relativo a registro:** Hace referencia al contenido de la localidad de memoria apuntada por el registro índice, o base en particular, más un desplazamiento.

El desplazamiento generalmente se aplica en aritmética signada como complemento a dos. Lo anterior quiere decir que pueden haber desplazamientos negativos y positivos (hacia delante o atrás en la memoria), por ejemplo:

**NOTA:** Para este tipo de direccionamiento y para los demás, se asumirá la notación *little endian* (el más bajo en memoria quedará almacenado en la parte alta del destino).

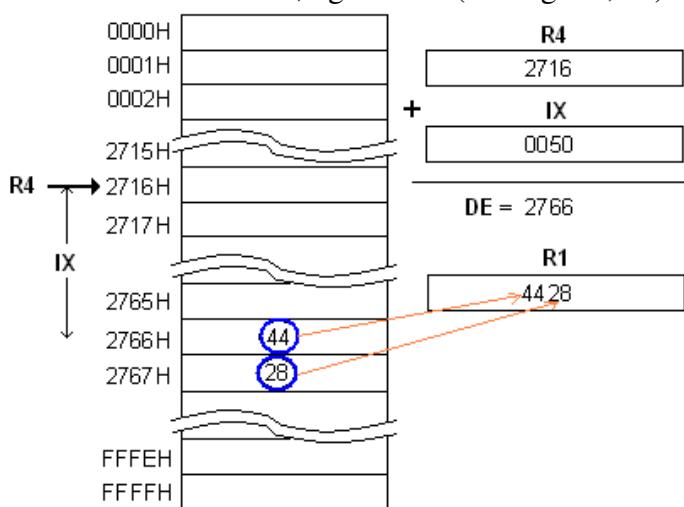
MOVER (IX + 0x100) , R3 ;Almacene en el registro R0 el contenido de  
;las celdas de memoria apuntadas por el  
;índice IX más 100 hexa-posiciones (ver  
;figura 3.6).



**Figura 3.6. Direccionamiento indexado con offset.**

- **Indexado a base más índice:** Utiliza como apuntadores un registro base y un registro índice, de tal manera que la celda a ser accesada se encuentra en la localidad de memoria, resultante de sumar el contenido de los dos punteros. Por ejemplo:

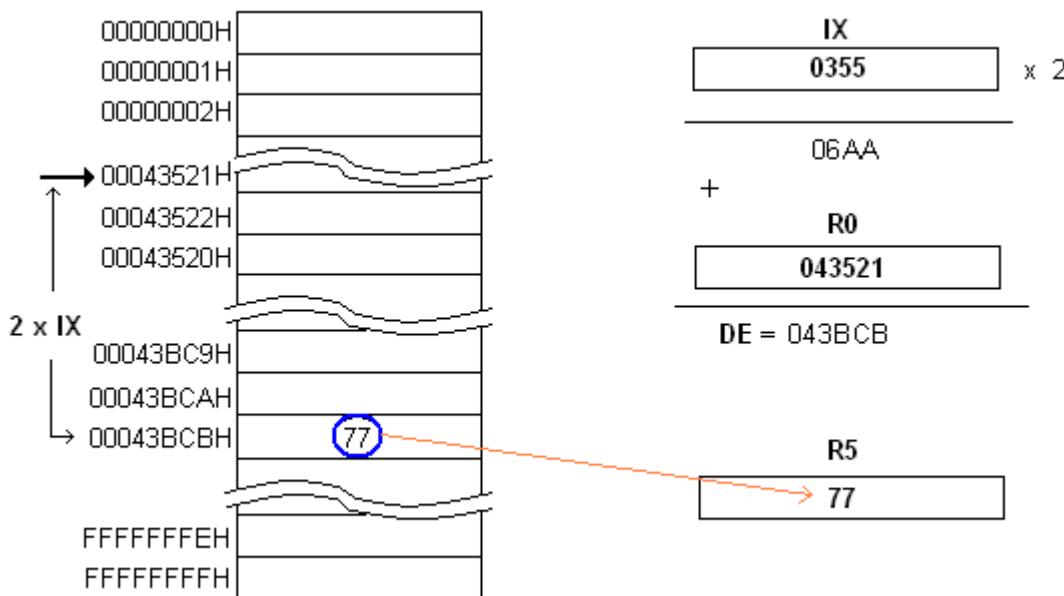
MOVER (R4 + IX), R1 ;Almacenar el contenido de las celdas apuntadas por ;el registro base R4 más el índice IX, dentro del ;registro R1 (ver Figura 3.7).



**Figura 3.7. Direccionamiento a base más índice.**

- **Indexado a base con índice y escalamiento:** Este tipo de direccionamiento es muy poderoso debido a la estructura de punteros que utiliza y a las propiedades de escalamiento y desplazamiento, para formar la dirección efectiva. Por ejemplo:

MOVER (R0 + 2\*IX) , R5 ;Copia en R5 el contenido de la celda apuntada por ;el registro base R0 más el contenido del registro ;índice IX multiplicado por 2 (ver Figura 3.8).



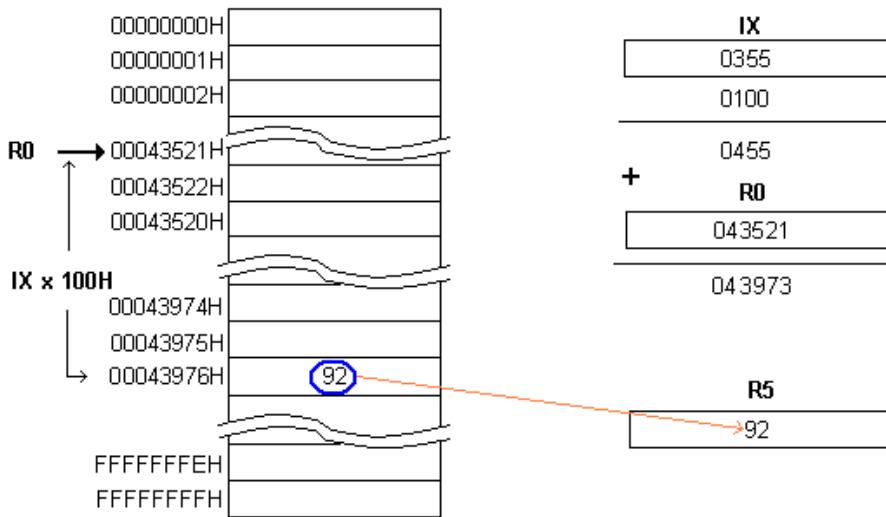
**Figura 3.8. Direccionamiento a base con índice y escalamiento.**

- **Relativo a base más índice con desplazamiento:** Hace referencia al contenido de la localidad de memoria apuntada por el registro índice más la base y a todo esto se suma un desplazamiento. Este desplazamiento generalmente se aplica en aritmética signada como complemento a dos. Quiere decir que puede haber desplazamientos negativos y positivos, por ejemplo:

MOVER [R0 + IX + 0x100] , R5 ;Almacene en el registro R5 el contenido de ;la celda de memoria apuntada por la suma ;del registro base R0 y el índice IX, más un ;offset de 0x100 (ver Figura 3.9).

- **Direccionamiento de salto relativo condicional**

La verdadera “inteligencia” de las máquinas radica en la toma de decisiones, lo que en un lenguaje C llamaríamos las sentencias IF-ELSE. Un direccionamiento relativo condicional evalúa una bandera de la máquina, lógicamente después de haberse ejecutado una operación aritmética, lógica o de control.



**Figura 3.9. Direccionamiento relativo a base más índice con desplazamiento.**

Los saltos relativos tienen dos posibilidades, la primera evalúa si la condición es verdadera y se ejecuta el salto; la segunda es tomada por descarte y el programa continúa con su curso normal.

De cualquier manera el contador de programa es trasladado a la dirección del salto y el programa se sigue ejecutando a partir de la nueva dirección.

El salto que se realiza es limitado y obedece a un salto con dirección, es decir, la máquina puede saltar hacia delante (incrementando en memoria) o hacia atrás (decrementando en memoria). La magnitud del salto está comprendida en un número de 8 o 16 bits y trabaja en aritmética signada como complemento a dos. Por ejemplo:

SALTE SI ACARREO 59H	;Salte 59H posiciones hacia delante si la ;bandera de acarreo está en “1”.
----------------------	---

Para evitar tener que calcular cuantas celdas hacia adelante o atrás debe saltar la máquina, los ensambladores (compiladores) permiten referenciar el sitio del salto usando etiquetas. Por ejemplo, la instrucción anterior se podría expresar:

SALTE SI ACARREO LOOP	; Salte a LOOP si la bandera de acarreo está ;en “1”.
-----------------------	--

La única condición es que LOOP no se encuentre por fuera de la magnitud signada del salto..

La ventaja de este tipo de saltos es la rapidéz, debido a que se hace utilizando direccionamientos cortos (8 o 16 bits); la desventaja radica en la poca longitud del salto (ver Figura 3.10).

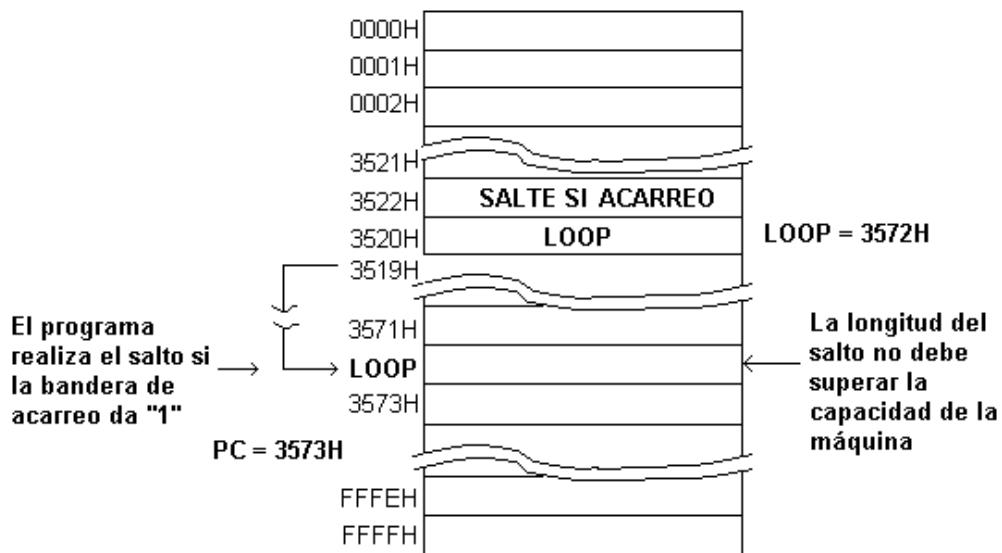


Figura 3.10. Direccionamiento de salto relativo condicional.

- **Direccionamiento de salto absoluto**

Son saltos por definición incondicionales con capacidades de salto mucho mayores a los relativos. Por ejemplo:

SALTE	INICIO	;salte a INICIO incondicionalmente (ver ;Figura 3.11.
-------	--------	--

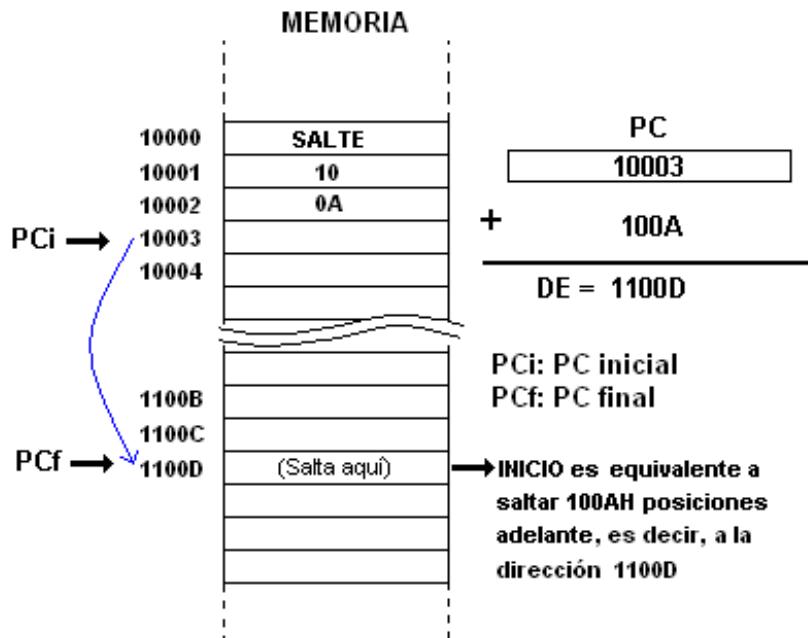


Figura 3.11. Direccionamiento de salto cercano.

- **Direccionamiento de puntero a pila (SP: stack pointer) sin desplazamiento**  
Utiliza el puntero de pila como apuntador a memoria. Este direccionamiento es muy rápido debido a las propiedades de decodificación y movimiento del puntero a pila. Por ejemplo:

ALMACENE R0 , (SP) ;almacene el registro R0 en la celda de memoria  
;apuntada por el registro SP (ver Figura 3.12)

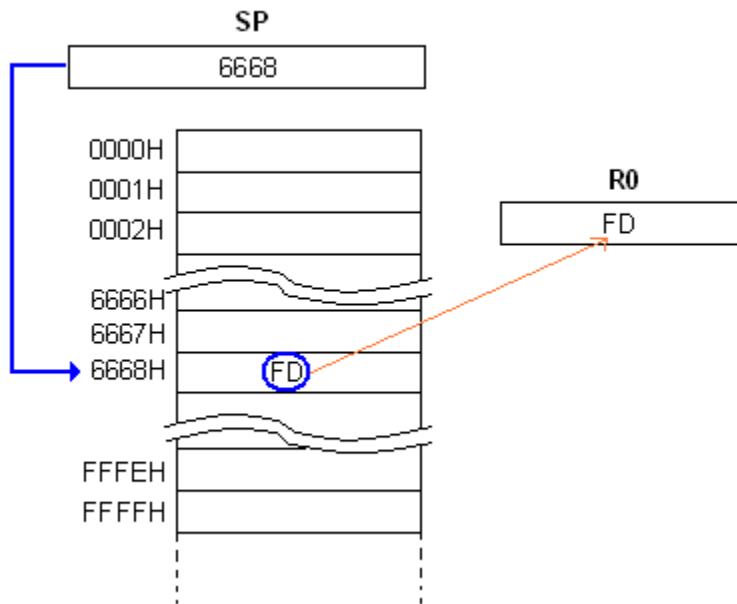


Figura 3.12. Direccionamiento de pila sin offset.

- **De puntero a pila con desplazamiento**  
Manipula el dato desde o hacia la dirección apuntada por el puntero de pila más un desplazamiento. Por ejemplo:

MOVER 0x1023 (SP) , R1 ;Almacene en el registro R1 el contenido de la celda  
;apuntada por SP más 0x1023 posiciones (ver  
;Figura 3.13)

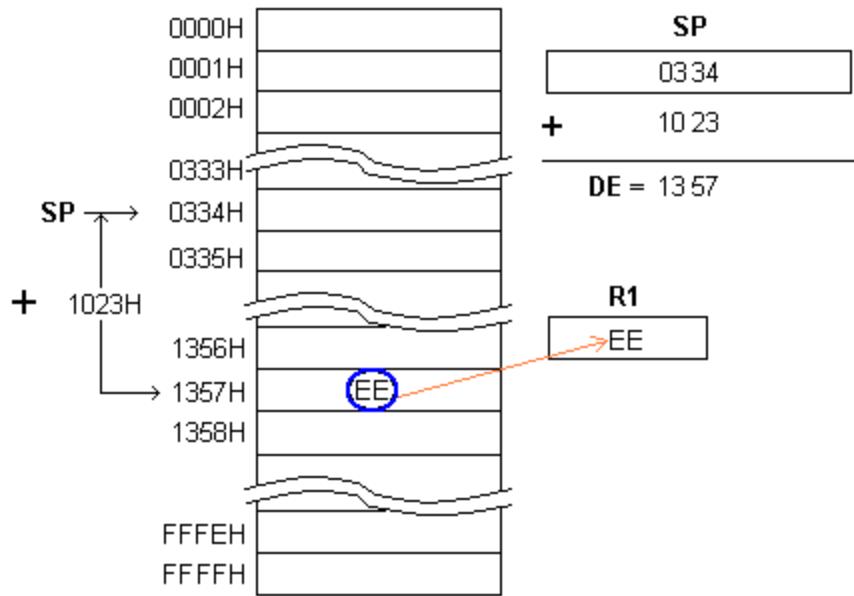


Figura 3.13. Direccionamiento de pila con *offset*.

- **Direccionamiento de memoria a memoria**

Lo interesante de este tipo de direccionamiento es que no utiliza registros de la máquina para manipular la información. Este direccionamiento trabaja directamente con las localidades de memoria, proporcionando un ahorro en memoria de programa y un aumento en la velocidad de ejecución. Ejemplos:

- **De inmediato a directo:** Se almacenan constantes en las posiciones de memoria correspondientes a la página cero, por ejemplo:

MOVVER #0x3D , 0x00FE ;almacene la constante 0x3D en la localidad de memoria con nombre 0x00FE (ver figura 3.14)

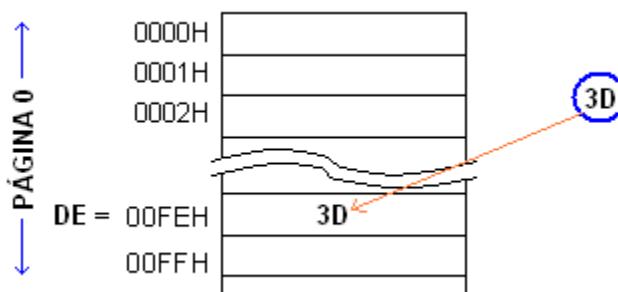


Figura 3.14. Direccionamiento inmediato a directo.

- **De directo a directo:** Se accesa directamente información entre celdas de memoria, sin necesidad de utilizar registros de la máquina. Por ejemplo:

MOVVER 0x0037 , 0x0051 ;Almacene el contenido de la localidad de memoria

;0x0037 en la localidad 0x0051 (ver Figura 3.15).

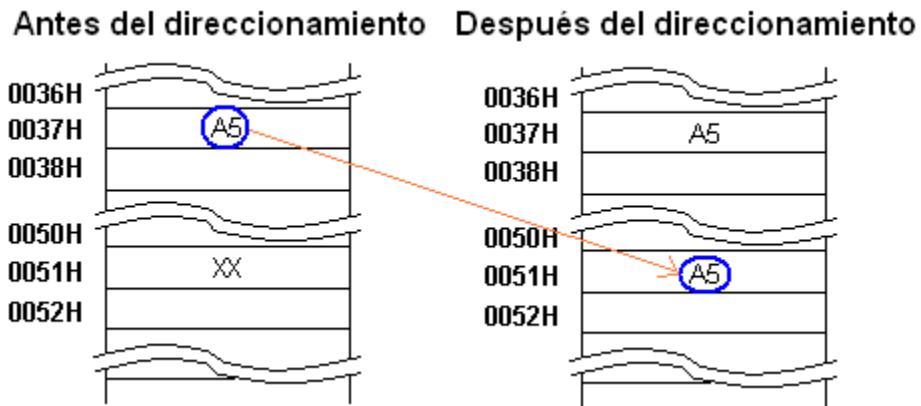


Figura 3.15. Direccionamiento directo a directo.

- **De indexado con postincremento a directo**

Se copia información de la memoria apuntada por el registro índice, en página cero, a la localidad de memoria, en página cero. Al final se incrementa la posición del índice. Por ejemplo:

MOVER IX+, 0x0002 ;almacene el contenido del registro apuntado por el  
;índice IX en la localidad 0x0002, y después  
;incremente el puntero IX (ver Figura 3.16).

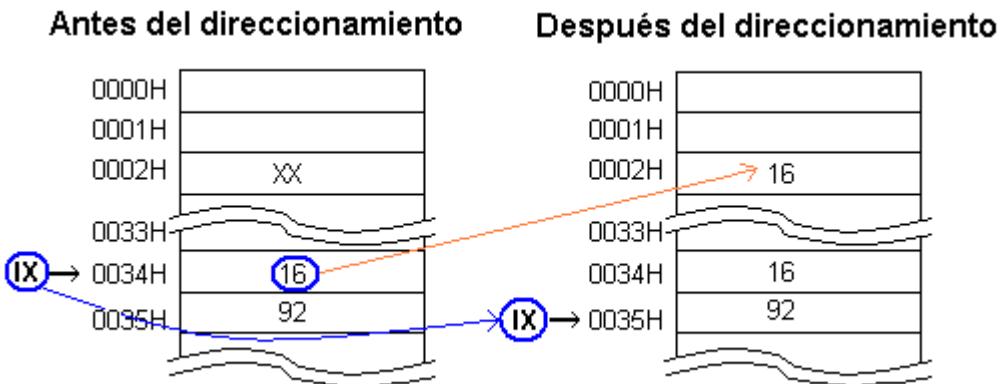


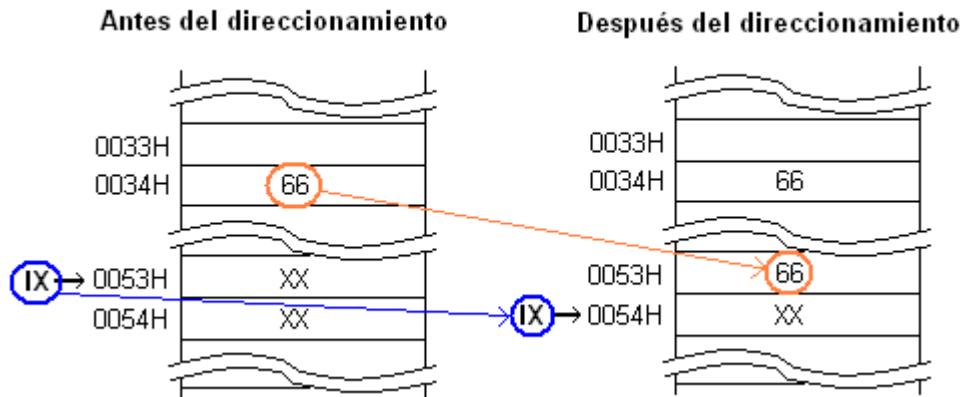
Figura 3.16. Direccionamiento indexado con postincremento a directo.

- **De directo a indexado con postincremento**

Se copia información de la localidad de memoria, en página cero, a la localidad de memoria apuntada por el índice. Finalmente, se incrementa el índice. Por ejemplo:

MOVER 0x0034, IX+ ;almacene el contenido de la celda 0x0034 en la  
;localidad de memoria apuntada por el índice IX y

;luego incremente el puntero (ver Figura 3.17)



**Figura 3.17. Direccionamiento directo a indexado con postincremento.**

- **Direccionamiento Inherente o implícito**

Este tipo de direccionamiento no especifica operandos en la instrucción, porque inherente a la instrucción va el operando o la acción concreta a ejecutar. Por ejemplo:

INCREMENTAR EL ACUMULADOR

;incrementa el acumulador de la  
;máquina  
;no opere

NOP

### 3.2. LENGUAJE ASSEMBLER

El *assembler* o mal llamado “ensamblador” es el lenguaje más cercano al núcleo de cada máquina, si se descarta el lenguaje de máquina como un método simple y directo de programar la tareas que ésta debe ejecutar.

El ensamblador está desarrollado en mnemónicos que tienen una interpretación directa en el código de máquina de la CPU y que lo que convierte en el lenguaje más rápido y eficiente, para aquellas aplicaciones en donde es necesario optimizar al máximo los recursos de la máquina (memoria, velocidad ,consumo, etc).

Cada núcleo de máquina tiene un assembler particular y se puede decir que único. En este sentido el *assembler* es un lenguaje de poca preferencia por el programador, pero que tarde o temprano tendrá que aprender si de optimización de código se trata. De hecho, en este texto se hace una especial recomendación en su uso y mezcla con el lenguaje C/C++.

Por lo general las máquinas con arquitectura Von Newman, bajo un concepto CISC, traen un amplio juego de instrucciones en *assembler*. Estas instrucciones toman varios ciclos de máquina en ejecutarse, pero pueden realizar operaciones medianamente complejas.

Las máquinas con arquitectura Harvard, bajo un concepto RISC, traen un juego reducido de instrucciones en *assembler*. Por lo general toman un ciclo de máquina en ejecutarse y realizan operaciones simples.

El formato genérico para introducir una instrucción en *assembler* es:

**ETIQUETA INSTRUCCIÓN DESTINO,FUENTE ;COMENTARIO**

En donde:

- **ETIQUETA:** referencia o dirección donde se encuentra esa línea de instrucción respecto de la memoria de código. Este campo sólo aparece si es necesario hacer referencia a un nombre para identificar el lugar.
- **INSTRUCCIÓN:** comando en lenguaje ensamblador, también llamado código de operación.
- **DESTINO:** Lugar de memoria o dispositivo E/S a donde será aplicado el direccionamiento o destino de la información.
- **FUENTE:** Lugar de memoria o dispositivo E/S de donde se toma la información.
- **COMENTARIO:** explicación corta de lo que se hace en la línea de programa.

**NOTA:** El DESTINO y la FUENTE pueden aparecer en diferente posición, dependiendo de la máquina a trabajar o podrían no aparecer, dependiendo del tipo de instrucción.

### 3.3. TIPOS DE INSTRUCCIONES

La relación entre los modos de direccionamiento y los tipos de instrucciones es de multiplicidad, es decir, existen muchos modos de direccionamiento que se acomodan a un tipo de instrucción, y muchos tipos de instrucciones que se acomodan a un tipo de direccionamiento.

Dependiendo de la arquitectura de la máquina, el juego de instrucciones puede ser muy amplio, moderado o justo. Existen máquinas cuyo juego de instrucciones supera las 300, otras son menores a 200 y existen máquinas con un juego de instrucciones menor a 40.

Existe un formato binario para representar las instrucciones llamado, código de operación (*opcode*). El formato puede estar formado desde 1 hasta 6 bytes, dependiendo de la complejidad de la operación que realiza la instrucción.

A continuación se definen los tipos más comunes de instrucciones y que están orientadas hacia una máquina genérica.

- **Instrucciones de movimiento de datos**

Las instrucciones de movimiento de datos, llevan y traen datos desde o hacia la memoria, registros o dispositivos E/S. Las instrucciones de movimiento de datos se usan para intercambiar, copiar y mover datos escalares con un ancho de 8, 16, 32 y hasta 64 bits; así como arreglos vectoriales, matriciales y cadenas de caracteres. Dentro de este grupo se podrían destacar:

- **Instrucciones de movimiento de datos:** Estas instrucciones copian datos desde una fuente hacia un destino. Dependiendo del modo de direccionamiento aplicado, el dato estará en un lugar específico o podrá ser una constante.
- **Instrucciones de movimientos con la pila:** Este tipo de instrucciones sirven para ingresar o extraer datos hacia o desde la pila. Son instrucciones muy rápidas por estar asociadas al mecanismo de la pila y el puntero a pila. Son ampliamente usadas en los llamados a funciones o atenciones a eventos de interrupción.
- **Instrucciones de intercambio de datos y movimiento de bloques:** Intercambian los contenidos de registros o memoria de una máquina de manera individual o grupal. Son ampliamente utilizadas en la transferencia de bloques de información.
- **Instrucciones aritméticas y lógicas**

Si algo sabe hacer bien un procesador es sumar. Sumando, desplazando y complementando se obtienen las demás operaciones aritméticas en una máquina. Máquinas más poderosas contienen unidades como las unidades multiplicadoras y de acumulación (MAC) y unidades de operación en punto flotante (FPU).

Las principales operaciones aritméticas son la suma, resta, multiplicación, división y algunas adicionales como la comparación, negación, el incremento y el decremento. A continuación se describen brevemente las operaciones aritméticas más comunes.

- **La suma:** Existen dos maneras de sumar en un microprocesador, a saber:
  - **Suma sin acarreo:** Para este tipo de suma no se tiene en cuenta un posible acarreo de entrada. Las cantidades son sumadas bit a bit y el resultado se guarda en el registro que se especifique, siendo el más usual el acumulador de la CPU. Si el resultado sobrepasa la capacidad de almacenamiento del registro destino, se genera un acarreo que se guarda en la respectiva bandera.

```
MOVER 0x100F , R0 ;almacene en el registro R0 la cantidad 0x100F  
SUMAR 0xFD03 , R0 ;adicone a R0 la cantidad 0xFD03 y el resultado  
;almacénelo en R0
```

El resultado de esta operación, asumiendo a R0 como un registro de capacidad en 16 bits será:

R0 = 0D12H y la bandera de acarreo C = 1

- **Suma con acarreo:** Para este tipo de suma se tiene en cuenta un posible acarreo de entrada. La instrucción es bastante útil para sumas con máquinas cuya capacidad operativa excede el ancho de su palabra de dato, de tal manera que tendrían que partir la suma en sumas parciales. Por ejemplo:

Para una máquina de 8 bits de dato, hacer la suma de las cantidades 4D28H (operando 1) y 7FF3H (operando 2) no se podría hacer directamente. Entonces es necesario partir la operación en dos partes. En la primera parte se sumarían los bytes de menor peso de cada operando (28H + F3H), esta suma se haría del tipo sin acarreo, porque no es necesario asumir un acarreo de entrada. La segunda parte sería hacer una suma con acarreo de los bytes altos de los operandos (4DH + 7FH), de esta manera se tendría en cuenta un posible acarreo arrojado por la primera suma para llevarlo a la siguiente suma. Un programa en *assembler* para una máquina genérica sería:

```

MOVER 0x28 ,R0          ;cargue el registro R0 con la parte
                           ;baja del operando 1
SUME    0xF3 , R0        ;sume sin acarreo la parte baja del
                           ;operando 2 al registro R0 y guarde
                           ;resultado en R0
MOVER R0 , (0x0041)      ;almacene el resultado bajo de la
                           ;operación en la celda 0x0041
MOVER 0x4D , R0          ;cargue el registro R0 con la parte
                           ;alta del operando 1
SUME CON ACARREO 0x7F , R0 ;adicone con acarreo la parte alta del
                           ;operando 2
MOVER R0 , (0x0040 , R0   ;almacene resultado alto de la
                           ;operación en la celda 0x0040

```

Con la operación de suma sin acarreo:

$$\begin{array}{r}
 & 28H \\
 + & \underline{F3H} \\
 & 1BH \quad y \quad C = 1
 \end{array}$$

Con la operación de suma con acarreo:

$$\begin{array}{r}
 & 1 \\
 & 4DH \\
 + & \underline{7FH} \\
 & CDH \quad y \quad C = 0
 \end{array}$$

De tal manera que el resultado final es igual a: CD1BH.

- **El incremento:** Incrementar es sumar uno a la cantidad, sea un registro o una localidad de memoria.
- **La resta:** Al igual que la suma, en la resta también se tienen las dos operaciones, con y sin acarreo.
  - **Resta sin acarreo:** El resultado de esta resta es almacenado en el registro destino, que generalmente es el acumulador de la CPU. Si la operación causa un sobreflujo, esto afectará las banderas de acarreo, sobreflujo, signo y probablemente acarreo intermedio. Por ejemplo:

Si se resta, en una máquina de 8 bits, de 1F3DH la cantidad 2470H, el resultado de restar la cantidad 70H de la cantidad 3DH arroja como resultado la cantidad CDH, y C = 1 (acarreo que no se tendría en cuenta para la siguiente operación, de hacerse con resta sin acarreo). Seguidamente, al restar de 1FH la cantidad 24H el resultado es FBH. Finalmente el resultado de la operación debió haber sido FACDH y no FBCDH, por la razón de no haberse tenido en cuenta el acarreo producido en la primera resta parcial, todo esto debido al empleo de la operación de resta sin acarreo.

$$\begin{array}{r}
 3DH \\
 - 70H \\
 \hline
 \text{CDH y C = 1}
 \end{array}
 \quad
 \begin{array}{r}
 1FH \\
 - 24H \\
 \hline
 \text{FBH y C = 1}
 \end{array}$$

Resultado total = FBCDH y C = 1

- **Resta con acarreo:** A diferencia de la resta sin acarreo, para esta operación se tiene en cuenta el acarreo generado anteriormente. El resultado de la operación anterior daría como resultado FACDH, y C = 1, el cual concuerda con el resultado verdadero.

$$\begin{array}{r}
 3DH \\
 - 70H \\
 \hline
 \text{CDH y C = 1}
 \end{array}
 \quad
 \begin{array}{r}
 1FH \\
 - 24H \\
 \hline
 \text{FAH y C = 1}
 \end{array}$$

Resultado total = FACDH y C = 1

La operación anterior causa, también, que banderas como la de signo y sobreflujo se activen.

- **El decremento:** Decrementar es restar uno a la cantidad, sea un registro o una localidad de memoria.
- **La multiplicación:** Hacer una multiplicación en un procesador de 8 bits, de las décadas de los 70 y 80 significaba un gran gasto de operaciones de suma, resta y desplazamiento en forma iterativa. Con consumos de cientos de ciclos de máquina. En la actualidad las máquinas pueden hacer una multiplicación en 4 ciclos de máquina o menos, con instrucciones específicas para tal propósito.
- **La división:** Como la multiplicación, una división en un procesador de 8 bits, de las décadas de los 70 y 80 significaba un gran gasto de operaciones de suma, resta y desplazamiento en forma iterativa.

Normalmente, en los procesadores actuales no está implementada una división inmediata por problemas que se comentarán a continuación. Por lo general el dividendo duplica en capacidad al divisor, por ejemplo de 16 en 8 ó 64 en 32, tampoco hay un comportamiento predecible de las banderas y adicionalmente existe el peligro de dividir por cero.

El desbordamiento que se produce en una división, también genera resultados conflictivos, por ejemplo, si en una máquina con acumulador de 8 bits se va a dividir la cantidad 0x4567 entre el número 5, el cociente de dicha división no se puede almacenar en el acumulador de la CPU, porque provocaría un desbordamiento.

Debido a tanto inconveniente en la implementación de la división, se deja a control del programador hacer más manejables los algoritmos de división para el usuario final de la aplicación.

- **Operaciones en BCD (*Binary Coded Decimal*):** La aplicación práctica de operar con números BCD, es poder representar cantidades decimales como agrupaciones de 4 bits, haciendo referencia a las unidades, decenas, centenas, miles,..., etc., así como las partes fraccionarias en décimas, centésimas, etc.

Otra ventaja radica en la visualización de cantidades en dispositivos llamados *display* de siete segmentos (ver Figura 3.18), de tal manera que la presentación de resultados binarios, mediante números BCD, sería directa.

La operación más popular para el tratamiento de cantidades BCD es la de AJUSTE DECIMAL, la cual es aplicada una vez se haya realizado la operación aritmética sobre la cantidad BCD. Por ejemplo:

Si al número BCD 95 (10010101) se le va a sumar el número BCD 27 (00100111), el resultado nos arroja el número BCD 122 (000100100010), para

que esto sea posible en un procesador, es necesario hacer el ajuste decimal. Este ajuste se hace de acuerdo a la Tabla 3.1 que se detalla a continuación:

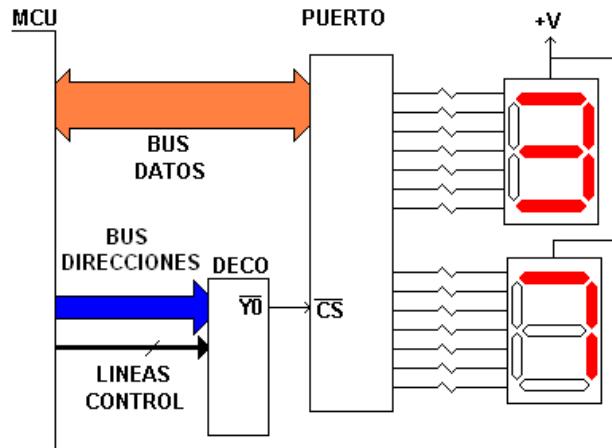


Figura 3.18. Aplicación operaciones BCD.

Tabla 3.1. Ajuste BCD según operación.

Operación anterior	Estado Bandera C	Valor de Bits 4 - 7	Estado bandera H	Valor Bits 0 - 3	Número a sumar	Estado Final del C
SUMA	0	0 - 9	0	0 - 9	00	0
	0	0 - 8	0	A - F	06	0
	0	0 - 9	1	0 - 3	06	0
	0	A - F	0	0 - 9	60	1
	0	9 - F	0	A - F	66	1
	0	A - F	1	0 - 3	66	1
	1	0 - 2	0	0 - 9	60	1
	1	0 - 2	0	A - F	66	1
	1	0 - 3	1	0 - 3	66	1
RESTA	0	0 - 9	0	0 - 9	00	0
	0	0 - 8	1	6 - F	FA	0
	1	7 - F	0	0 - 9	A0	1
	1	6 - F	1	6 - F	9A	1

Entonces para la operación anterior:

$$\begin{array}{r}
 10010101 \rightarrow 95 \text{ BCD} \\
 + 00100111 \rightarrow 27 \text{ BCD} \\
 \hline
 10111100 \rightarrow \text{Según la tabla se} \\
 + 01100110 \rightarrow \text{debe sumar 66 BCD} \\
 \hline
 100100010 \rightarrow 122 \text{ BCD}
 \end{array}$$

- **Operaciones lógicas:** Las operaciones lógicas se ejecutan bit a bit, proporcionando control de datos a bajo nivel, estas operaciones modifican las banderas de la CPU. Las operaciones lógicas más importantes son:

- **AND:** Es el producto lógico, bit a bit, sobre dos cantidades, la operación es muy utilizada para chequeo de bits contra una máscara o dato de chequeo.
- **OR inclusiva:** Esta operación realiza la suma inclusiva de dos cantidades, al igual que la operación AND, también es muy utilizada con una máscara o dato de chequeo.
- **OR Exclusiva (XOR):** Hace la suma exclusiva, bit a bit, de dos cantidades. La OREX sirve como función comparadora de cantidades. Si se hace la XOR de un registro sobre el mismo sería equivalente a borrar su contenido.
- **Negación NOT:** La operación de inversión lógica es la NOT y realiza el complemento a “1” de una cantidad.
- **Negación NEG:** La operación de inversión aritmética es NEG y realiza el complemento a dos de una cantidad. Recordar que el complemento a dos es la NOT de la cantidad más “1”.

- **Instrucciones de salto sin condición**

Estas instrucciones permiten que el curso normal del programa ramifique hacia otra dirección, en donde se continuará el proceso, sin evaluar ningún tipo de condición.

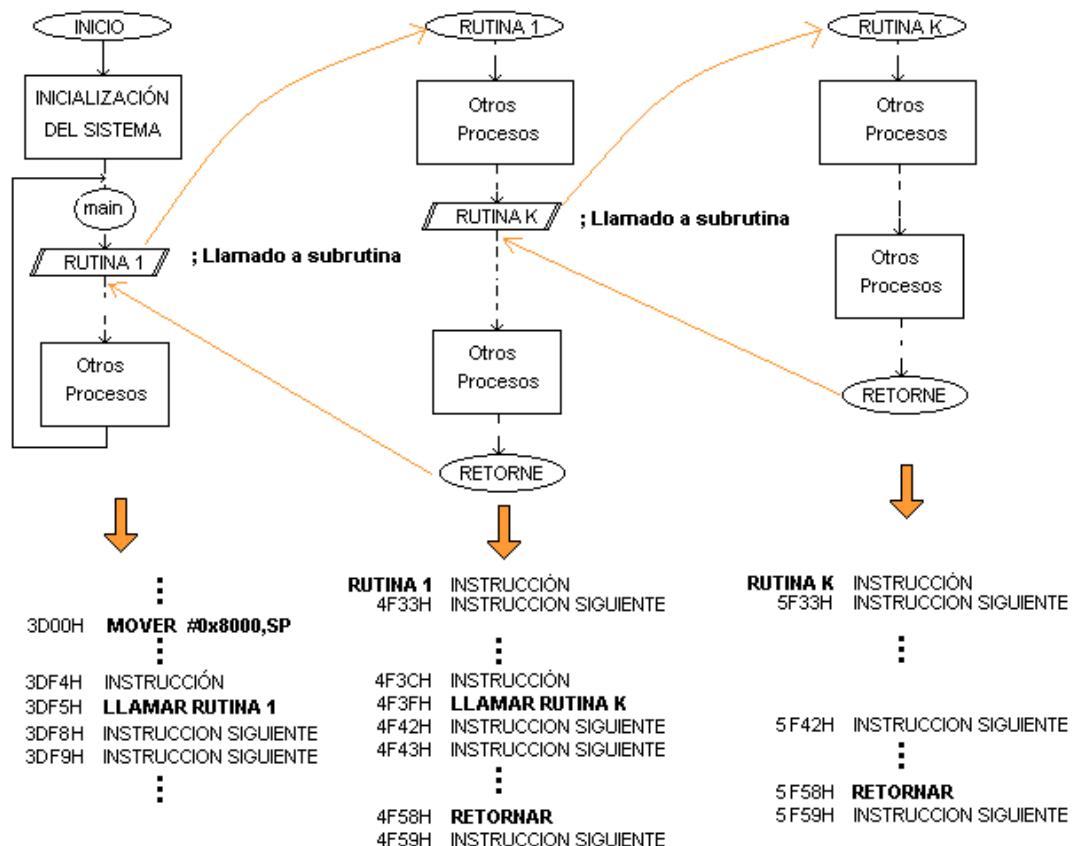
- **Instrucciones de llamados a subrutinas**

Son el equivalente en C/C++ a los llamados a funciones, y trasladan el flujo de programa a una dirección especial donde se encuentra un procedimiento compacto, que atiende la necesidad del proceso. Estos procedimientos son reutilizables y no es necesario volverlos a digitar cada vez que se necesitan, de esta manera ahorran espacio de memoria.

El único inconveniente para llamar subrutinas, es que a la máquina le toma un tiempo ejecutar el llamado y retornar de éste. El puntero de pila juega un papel importante ante un llamado a subrutina.

Es en la pila donde se guarda la dirección de retorno del programa después de haberse atendido el proceso de la subrutina, para de esta manera devolver el flujo normal del programa en ejecución.

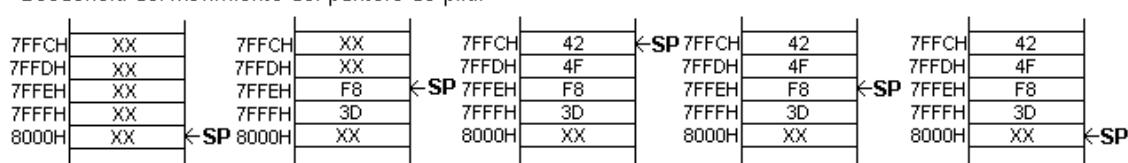
Para retornar de una subrutina, existe una instrucción especial que le indica a la máquina que deberá seguir con el flujo normal del programa. La Figura 3.19 muestra como se mueve la pila, el puntero a la pila y el contador de programa ante el llamado de una subrutina.



El movimiento del puntero de pila y el puntero a instrucción se podría tabular, de manera que en la primera fila aparece la instrucción en ejecución y en la segunda fila el ANTES y DESPUÉS de la ejecución. En la fila 3 aparece la dirección del PC y en la fila 4 la de SP.

	CALL RUTINA 1		CALL RUTINA K		RETURN RUTINA K		RETURN RUTINA 1	
	ANTES	DESPUÉS	ANTES	DESPUÉS	ANTES	DESPUÉS	ANTES	DESPUÉS
<b>PC</b>	3DF8H	4F33H	4F42H	5F33H	5F58H	4F43H	4F59H	3DF9H
<b>SP</b>	8000H	7FFEH	7FFEH	7FFCH	7FFCH	7FFEH	7FFEH	8000H

Secuencia del movimiento del puntero de pila:



**Figura 3.19. Movimiento del SP y el PC ante instrucción de llamado a subrutina.**

- **Instrucciones de control de bajo consumo**

Este tipo de instrucciones llevan a la máquina a un estado de no operación o hibernación. La importancia de estos estados está en poner a la máquina a consumir la menor cantidad de energía de la fuente de alimentación. La aplicación más directa es para los sistemas que se encuentran soportados por baterías.

En la mayoría de dispositivos “inteligentes”, como celulares, calculadoras, computadoras de bolsillo, relojes, etc, la batería juega un papel importante y mientras más dure mejor para el usuario final.

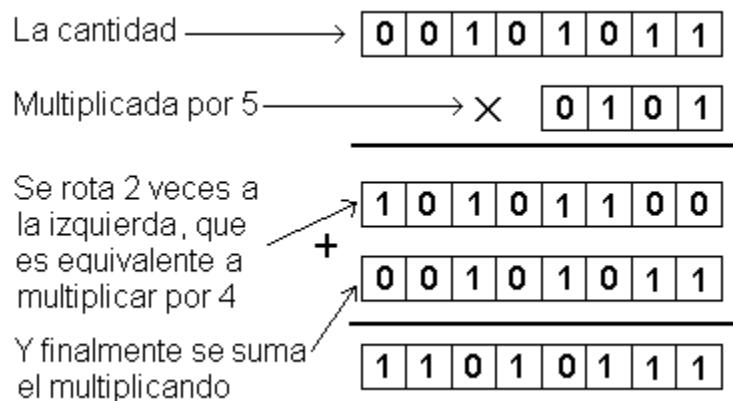
Es por lo anterior que las máquinas deben de estar dotadas de instrucciones para entrar en modos de bajo consumo.

- **Instrucciones para manipulación de bit**

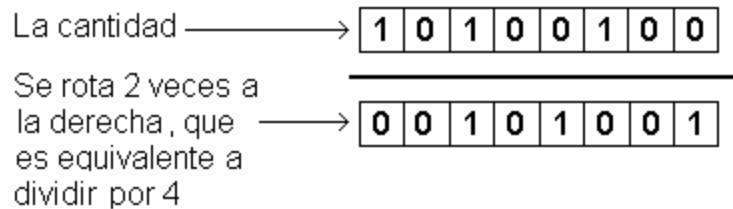
Se utilizan para poner en “1”, “0” o chequear el estado de un bit en particular. Las instrucciones de manipulación de bits ayudan al procesador a establecer eventos de bits, como las banderas generadas y manipuladas por el programador. También son usadas para el encendido, apagado y el chequeo del estado señales, de dispositivos de entrada y salida.

- **Instrucciones de rote y desplazamiento**

Un simple algoritmo de multiplicación podría implementarse haciendo que los bits de un registro rotén hacia la derecha (ver Figura 3.20) y conociendo cuantas veces es el número a dividir múltiplo de 2. Finalmente, si el número no es un múltiplo de 2, se harían ajustes con sumas sucesivas. Un algoritmo para la división se haría de forma similar, pero rotando hacia la derecha (ver Figura 3.21) y ajustando con restas sucesivas.



**Figura 3.20. Multiplicación por rote de bits.**



**Figura 3.21. División por rote de bits.**

En las comunicaciones seriales es necesario transmitir y recibir bit a bit la información contenida en un dato. Es por esto que éste tipo de instrucciones se hacen de gran utilidad y mediante un pequeño algoritmo se implementarían los UART (*Universal Asynchronous Receiver Transmitter*), tan necesarios en las comunicaciones seriales.

Es importante anotar que la bandera de acarreo juega un papel importante en este tipo de instrucciones, dando información sobre el desbordamiento de las cantidades o si es del caso el valor del bit en cuestión. Algunas de las instrucciones más importantes de éste género son:

- **Instrucciones de desplazamiento**

Estas a su vez se dividen en desplazamientos lógicos y aritméticos. Los desplazamientos lógicos no tienen en cuenta el signo de la operación, generalmente cuando se desplaza un número a izquierda o derecha, se inserta un cero por el bit extremo al desplazamiento (ver Figura 3.22).

Antes del desplazamiento

0	0	1	1	0	1	0	1	← 0
---	---	---	---	---	---	---	---	-----

Después del desplazamiento

0	1	1	0	1	0	1	0	← 0
---	---	---	---	---	---	---	---	-----

**Figura 3.22. Desplazamiento a la izquierda.**

Los desplazamientos aritméticos tienen en cuenta el signo de la cantidad (ver Figura 3.23), dejando éste como pivote y desplazando el resto de bits.

Antes del desplazamiento

1	0	1	1	0	1	0	1	0	1	1	0	1	0	1	0	← 0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

Después del desplazamiento

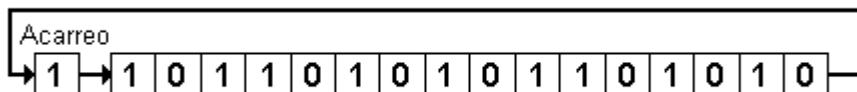
1	1	1	0	1	0	1	0	1	1	0	1	0	1	0	0	← 0
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

El bit de signo se conserva

**Figura 3.23. Desplazamiento signado.**

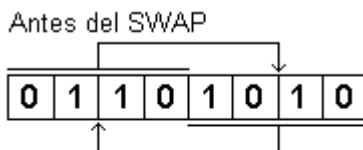
- **Instrucciones de rote**

Al igual que las de desplazamiento, las instrucciones de rote desplazan bits de izquierda a derecha y viceversa. Los rotes, a diferencia de los desplazamientos, realimentan los bits desplazados en forma circular, y en la mayoría de los casos se utiliza el bit de acarreo como parte del rote. En la figura 3.24 se ilustra un mecanismo de rote.



**Figura 3.24 Rote a la derecha**

Hay casos de rotes para *nibbles* y bytes llamados SWAP, los cuales consisten en intercambiar los *nibbles* (bytes) inferior y superior de un registro (ver Figura 3.25).



**Después del SWAP**

1	0	1	0	0	1	1	0
---	---	---	---	---	---	---	---

**Figura 3.25. Intercambio de nibbles.**

### 3.4. DIRECTIVAS Y/O PSEUDO-INSTRUCCIONES

Las directivas son instrucciones propias del ensamblador o compilador de una máquina. Estas pseudo instrucciones ayudan a controlar el proceso de ensamblado y la forma como un operando va a ser interpretado. También sirven al programador para hacer más fácil el entendimiento de las variables, direcciones y constantes que intervienen en un programa hecho en lenguaje de bajo y mediano nivel.

Las máquinas no entienden las directivas, porque no forman parte del juego de instrucciones que éste ejecuta. Las directivas se ejecutan en tiempo de compilación o ensamble.

Las directivas más comunes y útiles son:

- **Directivas de definición de datos:** Este tipo de instrucciones se emplean para ubicar datos en secuencia y asignarles un tipo numérico. Las más representativas en su género son:
  - **DB (Define Byte):** Define un dato tipo byte o una secuencia de bytes en la memoria, generalmente memoria de código. Por ejemplo:

```
DB 23,PEPE,$FF,00,"C" ;todas estas cantidades son definidas como  
;tipo byte.
```

- **DW (Define Word):** igual que DB pero para datos como palabras. Por ejemplo:

```
DW $F43D,1234,PEPA,"OK"
```

- **DD/DL (Define Double/Define Long):** Igual que DB pero para una cantidad tipo long. Por ejemplo:

```
DD LONG,00FF34DDH,"HOLA"
```

- **Directivas de dirección inicial de programa:** Establecen la dirección a partir de la cual el ensamblador establecerá una serie de instrucciones o datos. Por ejemplo:

- **ORG (origen) / END (fin):** La directiva ORG establece el origen a partir del cual se ubicará un proceso o código de programa y la directiva END marcará el final del proceso de ensamblaje, por ejemplo:

```
MAIN      ORG          0x8000  
          MOVER        R0,R1  
          ADICIONAR   #34,R0  
          .....  
          .....  
ENDMAIN    END
```

En éste código, las instrucciones serán ubicadas y ensambladas a partir de la dirección 0x8000. Es necesario entender que la etiqueta MAIN es equivalente a la dirección 0x8000.

- **SEGMENT (segmento) / ENDS (fin de segmento):** asigna un segmento de memoria y determina a donde se cierra el segmento. Por ejemplo:

SEGDATOS	SEGMENT
DATOS8 DB	\$34,\$66,\$FEen ,,\$55
DATOS16 DB	\$FF3D,”ab”,6784
FINSEG ENDS	

Para éste ejemplo las etiquetas SEGDATOS, DATOS8, DATOS16 y FINSEG, determinan un lugar en la memoria a partir de la ubicación del segmento a donde se almacenarán una serie de datos.

- **Directiva de asignación (EQU):** La directiva EQU se utiliza para asignar a un nombre o etiqueta un valor numérico, su utilidad radica en que es mucho más fácil hacer referencia a un nombre que a una cantidad numérica. Por ejemplo:

MOTOR EQU 1	;MOTOR es el bit 1 de un puerto de salidas
PULSADOR EQU 2	;PULSADOR corresponde al bit 2 de un
	;puerto de entrada.

### 3.5. REFERENCIAS

- Múnera, Diego. Introducción A los Microprocesadores. Ed. Universidad Pontificia Bolivariana. Medellín, 2005.

### 3.6. PREGUNTAS

- ¿Qué es una dirección efectiva (DE)?
- Asumiendo un formato de instrucción INSTRUCCIÓN FUENTE , DESTINO, ¿Qué tipo de direccionamiento hay en cada línea?:
  - ADICIONAR            #0x3FD , R3            ;R3 es un registro de la máquina
  - MOVER                R2 , R6                ;R2 y R6 son registros de la máquina
  - INC                    (IY)                    ;IY es un registro índice de la máquina
  - SALTAR               TEXTO
  - SALTAR SI CERO     LOOP
  - MOVER                R3 , (IX + 0x67FD)    ;R3 es un registro de la máquina
  - MOVER                R1 , (R2 + 4\*IY)        ;R1, R2 e IY son registros de la máquina
  - PARAR
- Asumiendo un formato de instrucción INSTRUCCIÓN FUENTE , DESTINO, describir ¿qué tipo de error hay en cada línea?
  - MOVER               IX , #0xFF3C            ;IX es un registro de la máquina
  - INC                   (IY)                    ;incrementar la dirección del puntero
  - ADD                   #2 , R4                ;adicionar 2 al contenido de la celda apuntada por R4
- Enuncie cuatro tipos de instrucciones que puede ejecutar un procesador.
- ¿Qué es una directiva?
- ¿Puede un microcontrolador ejecutar una directiva? Justifique su respuesta.

# CAPÍTULO 4

## Elementos y Herramientas Básicas de Desarrollo

- 4.1. Consideraciones de diseño con microcontroladores**
- 4.2. La codificación**
- 4.3. El ensamblaje o compilación**
- 4.4. La depuración**
- 4.5. Herramientas de desarrollo disponibles en el mercado**
- 4.6. Otros aspectos a tener en cuenta en el diseño con microcontroladores**
- 4.7. Referencias**
- 4.8. Preguntas**

## 4.1. CONSIDERACIONES DE DISEÑO CON MICROCONTROLADORES

La base de un buen producto es la concepción inicial que de él se tenga, antes de hacerlo material.

El mantenimiento, actualización y diseño de programas (*software*) en las industrias, es un proceso demorado y que consume una gran cantidad de recursos. Entre éstos, uno de los más representativos es la mano de obra del ingeniero de desarrollo.

Un programador sin método y disciplina, genera productos difíciles de actualizar y mantener. Muchas veces las empresas prefieren comenzar desde cero, antes de retomar el diseño dejado por un programador de este estilo. Es por esto que en este texto se plantea una metodología, sencilla pero eficaz, para hacer desarrollos con microcontroladores; tratando en lo posible que los desarrollos se hagan teniendo en cuenta el mantenimiento y las actualizaciones de los programas a futuro.

Los circuitos (*hardware*) deberán estar diseñados bajo norma e implementados con componentes de buena calidad. La secuencia de consideraciones recomendadas para enfrentar un diseño se plantean a continuación y están orientadas a la programación en lenguaje *ASSEMBLER*.

- **Entendimiento del problema**

Por lo general un cliente no expresa bien, en los términos que un técnico o ingeniero quisiera oír, las especificaciones del producto que necesita. Sea por que falta información o porque no comprende la magnitud del diseño en sí.

El diseñador no sólo debe entender la necesidad que se le plantea, sino también tratar de completar o ajustar los requerimientos del cliente.

Entender y acotar el problema que se plantea es fundamental para garantizar el buen término del proyecto a realizar y de esta manera evitar los costosos reprocesos.

- **Requerimientos de entrada y salida involucradas en el sistema**

Consiste en hacer una lista del tipo de variables de entrada y salida que son necesarias en el diseño, como se muestra en la Tabla 4.1.

En esta lista deben figurar características mínimas, como: **NOMBRE** asignado a la variable en el proceso, **TIPO** de variable, el pin o los **PINES** que ocupa, el **RANGO** y la **MAGNITUD** en la que está contenida la variable y una breve **DESCRIPCIÓN** de la variable.

**Tabla 4.1. Tipos de variable.**

NOMBRE	TIPO	PINES	RANGO MAGNITUD	DESCRIPCIÓN
PULSADOR	Digital - Out	Bit 0 puerto A	0 – 5 VCD	Pulsador de inicio de secuencia
TEMPERATURA	Análoga - In	Bit 2 puerto B	0 – 4.35 V	Medida de la temperatura del horno
BUS DATOS DISPLAY	Digital - Out	Bit 0 a bit 7 puerto C	0 – 5 VCD	Bus de datos del display LCD
E	Digital - Out	Bit 0 puerto E	0 – 5 VCD	Señal de habilitación del LCD
RS		Bit 1 puerto E	0 – 5 VCD	Señal de Control/dato del LCD

- **Sistemas de alimentación de los circuitos**

Diseño de las fuentes de alimentación y filtros, considerando la disipación de los dispositivos de mayor consumo y sistemas de refrigeración.

El diseñador debe tratar de estandarizar en las diferentes fuentes de alimentación, en cuanto a las tensiones de los sistemas electrónicos, los tipos de conectores, la adecuada puesta a tierra de los sistemas eléctricos, el apantallamiento de las señales, la separación eléctrica de las diferentes fuentes de alimentación y la codificación del cableado.

El diseñador debe considerar el cálculo y la instalación de sistemas de disipación y refrigeración en las fuentes de alimentación.

- **Consideraciones de ruido por EMI (*Electro Magnetic Interference*)**

Todo equipo electrónico, y particularmente los equipos que manipulan señales de radio frecuencia, conmutación electrónica, entre otras, sufren el inevitable mal de ruido por interferencia electromagnética.

Las interferencias pueden ser tan leves como un simple ruido de “crispeteo” en una señal de audio o tan catastróficas como la perturbación de todo un sistema de vuelo en una aeronave y una desafortunada colisión.

La señal de 60Hz (50Hz) de la red de alimentación es el factor más común, que perturba los sistemas electrónicos. Hoy en día los 220Hz, generados por la conmutación de la fuente de los sistemas celulares, se constituyen en una de las altas fuentes contaminantes para los circuitos electrónicos.

En los ambientes industriales se tendrán fuentes adicionales de ruido, como el ruido por la conmutación en dispositivos de potencia (tiristores), Armónicas producidas por los motores y transformadores en las plantas productivas. Todas estas involucran señales fuertes para frecuencias de decenas de KHz.

Las conmutaciones digitales producen ruido de alta frecuencia del orden de MHz, que podrían perturbar a otros sistemas. Las etapas de RF de sistemas de radio, audio y televisión, son otros agentes altamente contaminantes y contaminados.

Para minimizar el efecto de estas fuentes de ruido, se recomienda el cálculo y la instalación de:

- Inductores de choque (ferritas), a la entrada de las líneas de potencia de los circuitos electrónicos.
- Las tierras.
- Blindaje y/o apantallamiento de los circuitos, usando recintos, cables y mallas especiales.
- Filtros para desacople y *bypass*.
- Filtros, para las diferentes etapas de circuitos del sistema.
- Adecuados circuitos impresos (PCB).

- **Conocimiento de las variables a manipular**

Este es un factor determinante para la seguridad que brinda un sistema al usuario. Las variables más importantes a considerar en un diseño son:

- **Variables vitales:** Cuando se desarrollan aplicaciones orientadas a sustentar, compensar, monitorear o reemplazar órganos de los seres vivos, transporte de los mismos, mecanismos para la diversión, máquinas de producción, herramientas y cualquier otro dispositivo que entre en contacto con seres humanos y animales, es necesario poner especial cuidado en la seguridad que el sistema debe brindar a éstos.

El diseñador deberá tener a la mano las normas locales o internacionales sobre las magnitudes eléctricas peligrosas, que atenten contra la integridad de los seres vivos. Deberá incluir en el diseño, las restricciones y los anuncios de seguridad para quienes manipulan los circuitos.

Cantidad de memoria de código y datos, necesarios para el buen funcionamiento del sistema y su crecimiento a futuro. Los sistemas que manipulan variables vitales deben tener alto grado de redundancia en sus sistemas “inteligentes”, actuadores y sensores, así como el soporte de batería o *backup* energético.

También se debe dejar la posibilidad de la operación manual del sistema, considerando que el automatismo puede fallar en cualquier momento.

- **Variables de valor:** En aplicaciones que comprometen objetos valiosos como el dinero, los medios de pago virtuales, seguridad de locales e industrias y cualquiera otra manifestación del término valor.

- **Variables ambientales:** Todo lo relacionado con la contaminación del planeta, preservación de recursos naturales, etc.
- **Diagramas y algoritmos**

Los diagramas son importantes porque universalizan la programación y sirven como herramienta para representar en forma gráfica el desarrollo de un algoritmo. Cuando los sistemas presentan un mediano o alto grado de complejidad, es altamente recomendado hacer un diagrama con el fin de agilizar el proceso de codificación y tener mayor certeza en el buen funcionamiento del programa.

Un algoritmo es un conjunto de pasos, procedimientos o acciones que nos permiten alcanzar un resultado o resolver un problema y constituye la parte central de un diagrama. En la vida cotidiana el ser humano aplica algoritmos en forma inconsciente. Por ejemplo, cuando se hace un plato de comida se repiten una serie de pasos y procedimientos que llevan a un resultado, en la mayoría de los casos, positivo.

A continuación se definen algunas herramientas y métodos para implementar diagramas de flujo y generar algoritmos:

- **Simbología ISO y/o ANSI para hacer diagramas de flujo:** Los símbolos básicos, que se utilizan en este texto para plantear algoritmos son presentados en la Tabla 4.2.

**Tabla 4.2. Simbología ISO para diagramación.**

SÍMBOLO	COMENTARIO
	Inicio o final de un procedimiento
	Líneas conectoras
	Conectores entre páginas
	Ciclo de decisión
	Entrada o salida de información
	Llamado a subrutina o proceso
	Operación o procedimiento

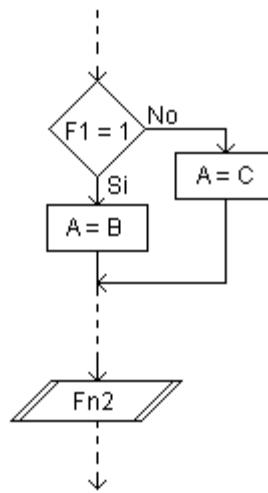
Para implementar desarrollos con la simbología propuesta, es necesario considerar algunas reglas mínimas:

- Todo diagrama de flujo debe tener un INICIO y un FIN, pero este FIN deberá indicar dónde continúa el sistema (ver Figura 4.1). Esta regla indica que el FIN no es funcional, sino de representación gráfica e indica el final del proceso.



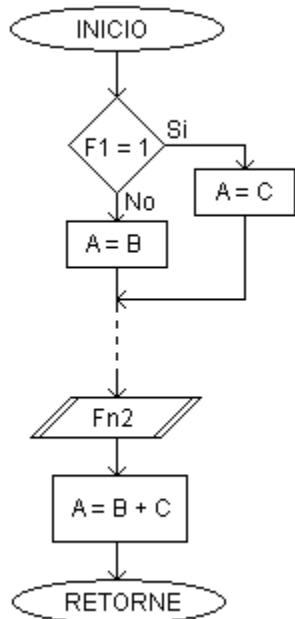
**Figura 4.1. Inicio y fin.**

- Las líneas conectoras deben hacerse en sentido horizontal y vertical, e indicar la dirección del flujo mediante una flecha (ver Figura 4.2).



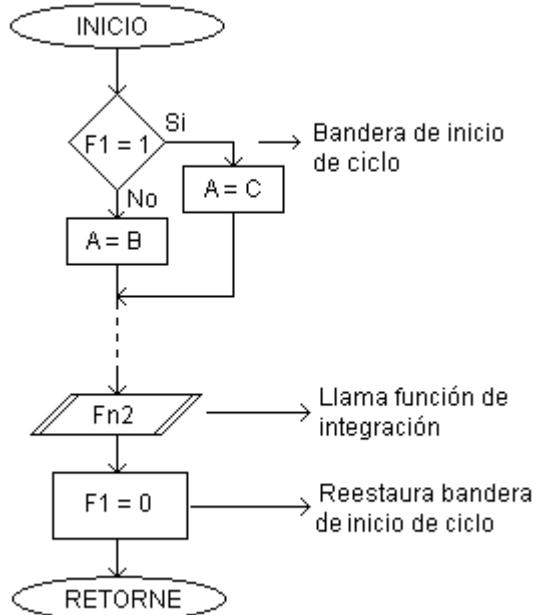
**Figura 4.2 Líneas conectoras.**

- Los diagramas se construyen tipo *left to right* y *top to down* (ver Figura 4.3).



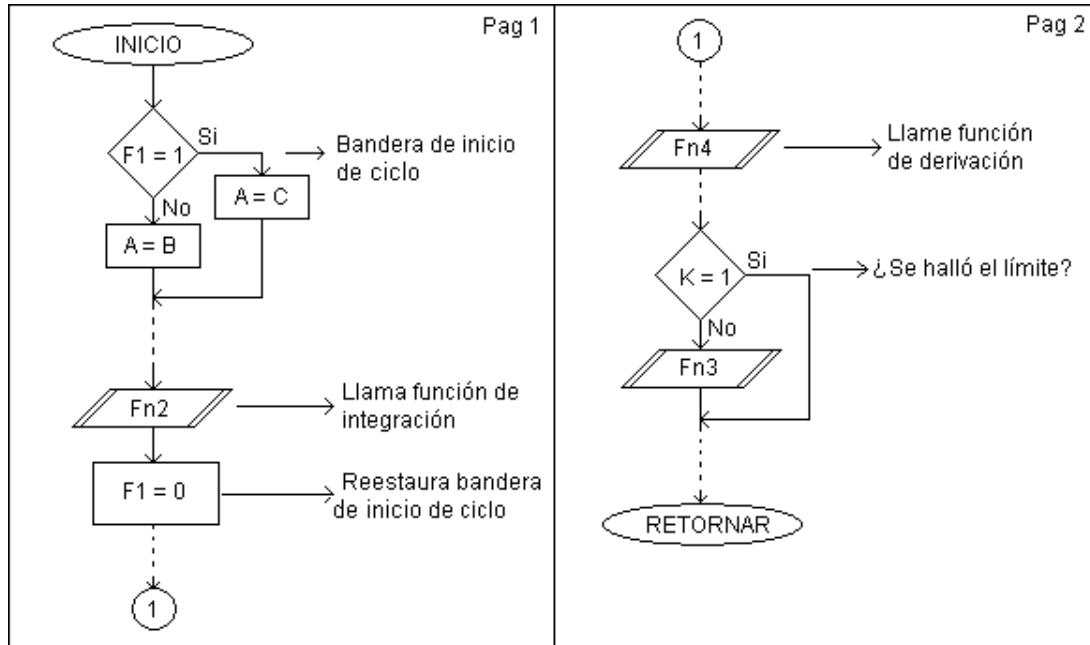
**Figura 4.3 Diagrama *top/down left/right*.**

- Acompañar los diagramas de comentarios que expliquen la acción principal de cierta parte del proceso (ver Figura 4.4).



**Figura 4.4 Uso de comentarios en los diagramas.**

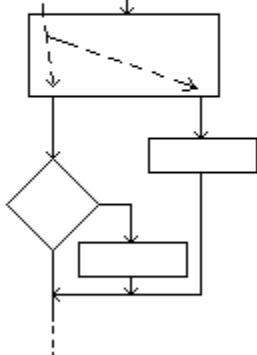
- Enumerar las hojas y utilizar conectores cuando el diagrama ocupe más de una página (ver Figura 4.5).



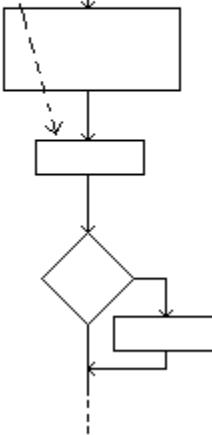
**Figura 4.5 Cambio de página.**

- Evitar los siguientes errores (ver Figuras 4.6 a, b y c):

a) Forma incorrecta: en un proceso no se pueden derivar más de una tarea.



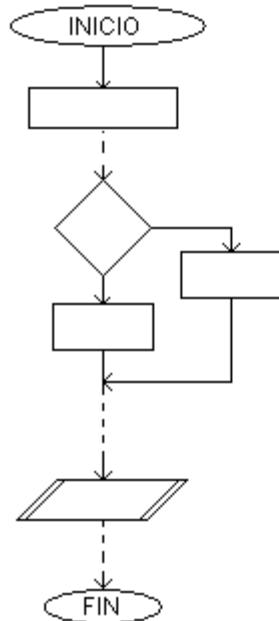
Forma correcta: trasladar un proceso como se muestra en la figura



**Figura 4.6\_a**

b) Forma incorrecta:

Un programa o un proceso no tiene fin, siempre debe fluir, a menos que el sistema sea desactivado



Forma correcta:

Si se trata del programa principal

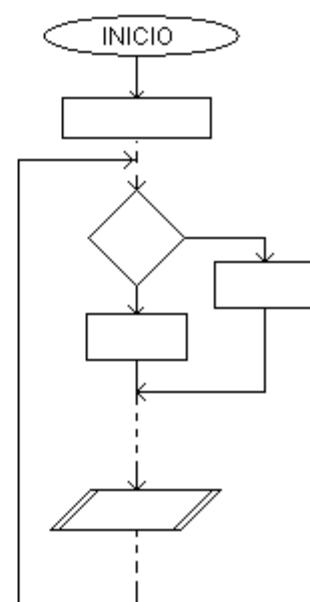
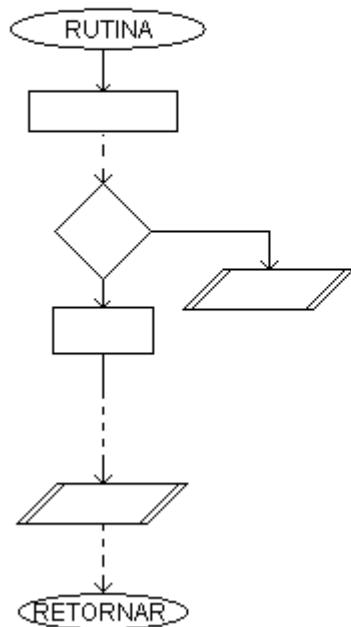


Figura 4.6\_b

c) Forma incorrecta:

Todo proceso debe mostrar continuidad



Forma correcta:

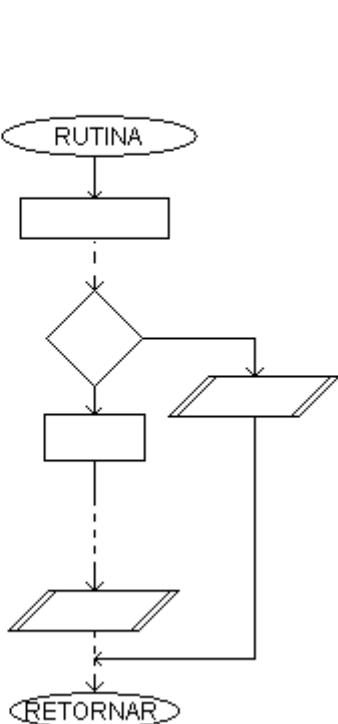


Figura 4.6\_c

- **Algoritmo**

Existen muchas técnicas para desarrollar algoritmos en bajo nivel, pero en este texto se plantea una técnica bastante sencilla y efectiva. Es en el algoritmo en donde el programador debe poner mayor cuidado a la hora de diseñar programas, teniendo en cuenta que éstos conforman la base operativa del diseño. Un algoritmo es eficiente cuando:

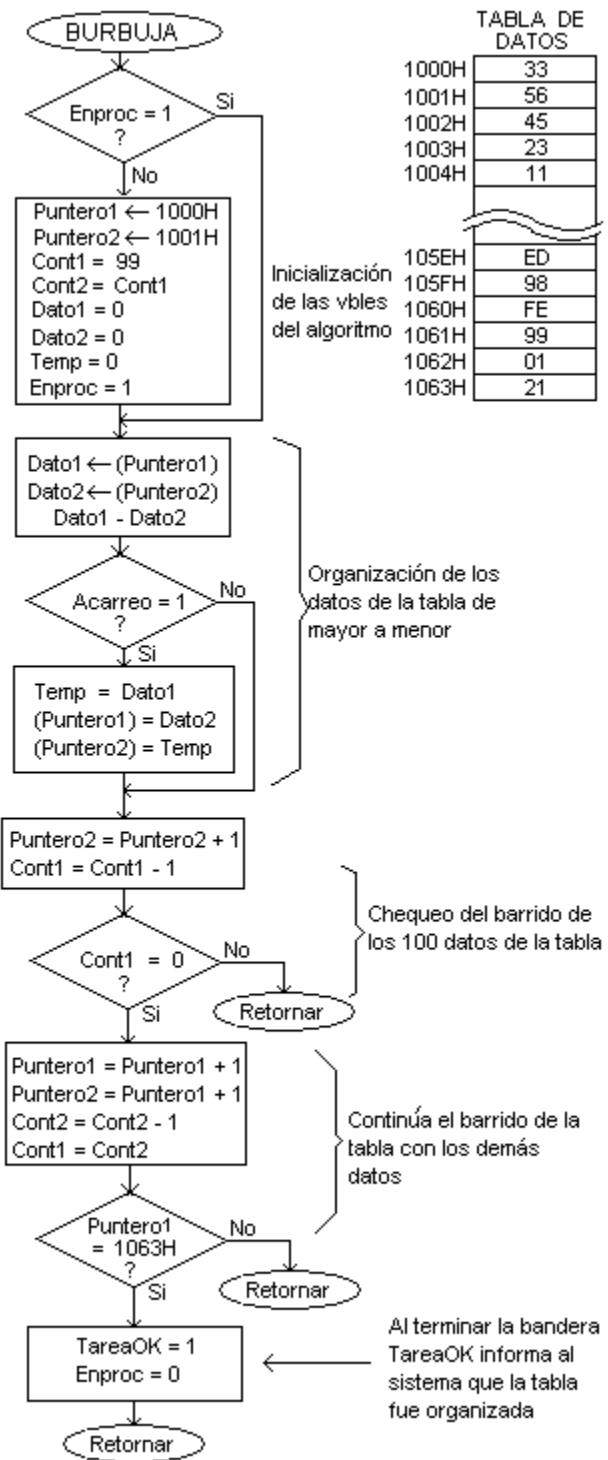
- Tiene pasos claros y precisos.
- Es determinístico o sea que para una entrada de datos idénticos debe arrojar siempre el mismo resultado.
- Es finito, es decir, su longitud es acotada por más complejo que sea.
- Utiliza modos de direccionamiento adecuados para las operaciones.
- Hace buen uso del recurso de la memoria.
- Es el más corto y óptimo en su desempeño.
- No hace perder tiempo innecesario a la CPU.
- Trata bien las variables que necesita en sus operaciones.
- Está bien documentado y es fácil de mantener.

Las partes más representativas, que conforman un algoritmo son:

- Variables o parámetros de entrada: son todas aquellas variables que intervienen en el desarrollo del algoritmo y que contienen las cantidades a ser operadas.
- Condiciones de entrada: condiciones específicas cuando se llama el algoritmo, de tal manera que desvén el flujo del proceso dependiendo del estado de la condición.
- Cuerpo de inicialización: parte del algoritmo donde se inicializan las variables que intervienen en éste, como registros de almacenamiento temporal, banderas, contadores, etc.
- Cuerpo principal del algoritmo: es el desarrollo de la parte “inteligente” del algoritmo. En esta parte también es posible que se generen variables de salida para el sistema.
- Variables o parámetros de salida: Son aquellas variables que entrega el algoritmo al sistema, como resultado de operación.

No todo algoritmo contiene las partes anteriormente descritas, pero se encuentra que la estructura planteada es bastante típica y se podría asumir como la regla más general. La Figura 4.7 muestra la estructura de un algoritmo con las partes anteriormente descritas.

El algoritmo organiza 100 números de mayor a menor, utilizando direccionamiento indirecto. La forma como está concebido el algoritmo se fundamenta en el principio de no permitir iteraciones internas, de tal manera que bloquen la máquina y se descuide el resto del sistema (este concepto se ampliará más adelante).



**Figura 4.7. Algoritmo para organizar de mayor a menor 100 datos en una tabla.**

En este algoritmo son utilizadas las siguientes variables:

- **Puntero1:** Puntero cabeza a tabla de datos.
- **Puntero2:** Puntero de desplazamiento secundario, para comparar con el dato cabeza.
- **Cont1 y Cont2:** Contadores de eventos.
- **Dato1 y Dato2:** Registros de almacenamiento temporal.
- **Temp:** Registro de propósito general
- **Enproc y TareaOK:** Banderas tipo bit.

- **Ejemplos de diagramación**

Los siguientes ejemplos muestran el manejo de la simbología para diagramas de flujo, así como la demostración de cómo enfrentar un problema hasta convertirlo en un algoritmo.

- **Ejemplo 1:** Utilizando diagramas de flujo, diseñar un algoritmo que busque en una tabla (dirección = TABLA) de 1000 datos (**CONTDATOS**), cuántas veces (**CONT69**) se encuentra el número 69. Si el número de veces que está éste número supera la cantidad 15, el sistema deberá generar una alarma (**ALARMA**).

Las variables de entrada al sistema y su valor inicial sería:

- Dirección de la **TABLA** donde se encuentran los datos inicializada an 1000H.
- Contador de datos **CONTDATOS**, inicializada en 1000.
- Dirección del **PUNTERO** para barrer la tabla de datos, inicializado en 1000H.

Las variables de salida e internas con su valor inicial:

- Variable **CONT69**, como contador de número de veces que se encuentra el 69H. Inicializado en 0.
- **ALARMA** del sistema, cuando se encuentran 15 o más números 69H. Inicializada en 0.

Luego viene la parte “inteligente” del algoritmo, en donde se extrae el primer dato de la tabla mediante el mecanismo de direccionamiento indirecto con **PUNTERO**. El dato es comparado con la cantidad 69H y de haber coincidencia se incrementa la variable **CONT69**.

Debido a que existe la condición de poner una alarma cuando **CONT69** llegue al valor de 15, es necesario hacer una segunda comparación.

Seguido a esto se debe decrementar la variable **CONTDATOS**, para controlar en el sistema el momento en el que se han comparado los 1000 datos de la tabla. Lo anterior se hace examinando el estado de la bandera Z (CERO) de la máquina.

El procedimiento se repite incrementando la variable **PUNTERO** y haciendo las comparaciones descritas anteriormente.

Note que el algoritmo no termina cuando la cantidad de números 69H ha llegado a 15, esto se deduce del hecho de que no se especifica en la definición inicial del problema y más bien se entiende que es necesario seguir comparando.

Finalmente el algoritmo debe entregar al sistema los siguientes parámetros:

- La variable **CONT69**.
- El estado de la variable **ALARMA**.

La Figura 4.8 muestra el resultado del diseño de éste algoritmo, asociado a un procedimiento principal (*main*).

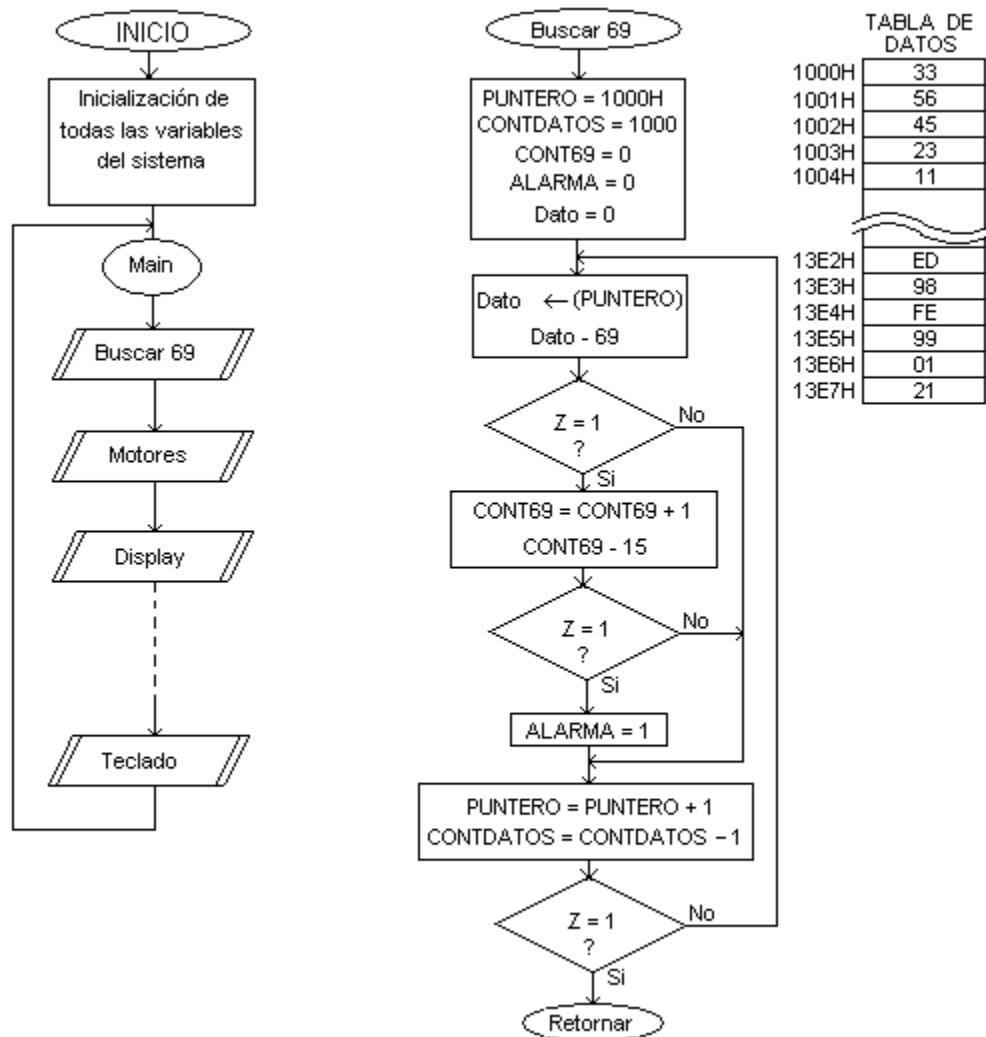


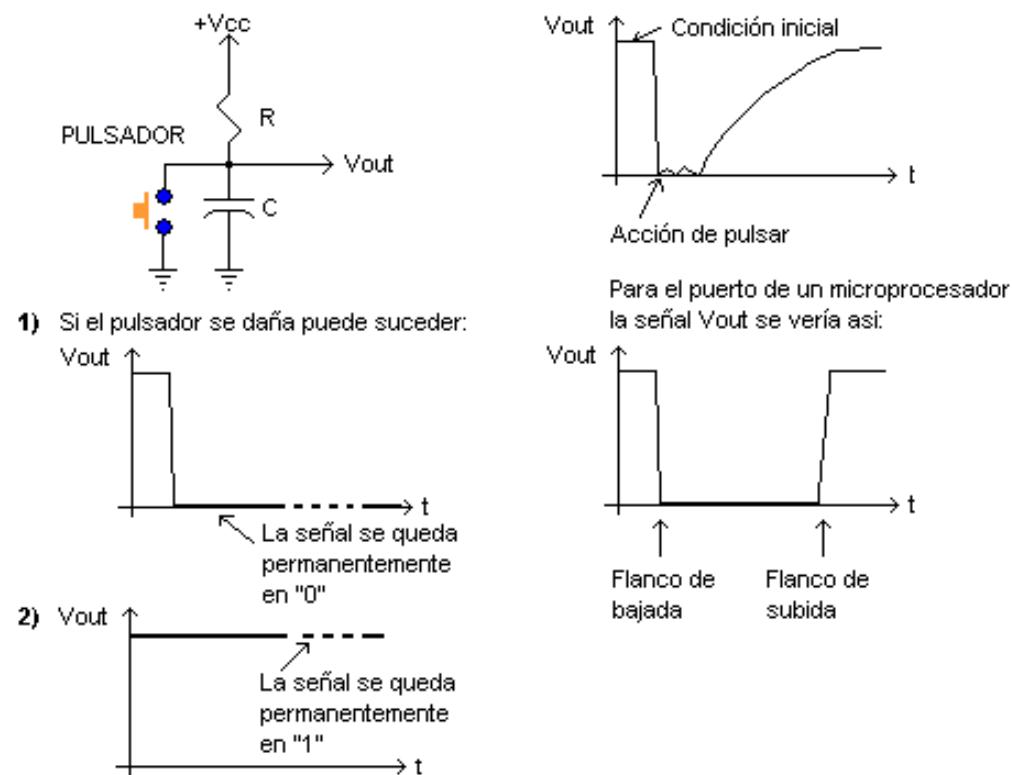
Figura 4.8. Programa para buscar dato en tabla.

- **Ejemplo 2:** Diseñar utilizando diagramas de flujo un algoritmo para detectar un flanco de bajada de un pulsador.

La condición inicial para éste algoritmo es garantizar que la señal del pulsador (ver Figura 4.9) se encuentre en el estado lógico “1”.

El algoritmo podría fallar si el pulsador o el pin del puerto se dañan, colocando un “0” permanentemente y el sistema podría caer en iteraciones infinitas.

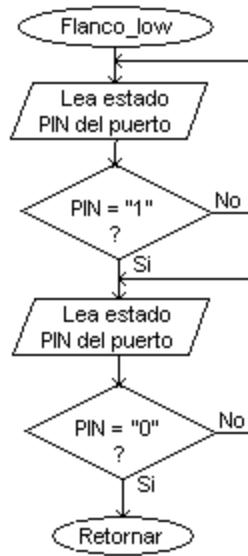
Más adelante es propuesto un algoritmo evolucionado, que supera el inconveniente planteado.



**Figura 4.9. Condiciones para lectura de flanco.**

Este algoritmo no tiene entradas de registro o memoria, pero si produce una salida que indica al programa principal que el flanco se ha presentado.

Una vez se ha garantizado el estado inicial lógico “1”, el algoritmo se queda esperando que la señal vaya al estado lógico “0” y de esta manera marcar el evento de flanco de bajada detectado (ver Figura 4.10).



**Figura 4.10. Detección de flanco de bajada.**

Para estos dos ejemplos, existe el problema de tener que dejar el microprocesador esperando eventos o iterando hasta que el algoritmo haya sido resuelto. Para evitar esta pérdida innecesaria de tiempo se observa a continuación, el uso de las banderas como marca de eventos del sistema.

- **Las banderas del usuario**

Las banderas nos indican sucesos importantes, acontecidos durante el desarrollo de un programa. Las banderas de usuario son bits pertenecientes a registros, generalmente del tipo RAM.

Una bandera marca un evento logrado dentro de un algoritmo, de tal manera que ciertos procesos ya alcanzados no se repitan y se pueda devolver el flujo de programa al microprocesador, evitando la pérdida innecesaria de tiempo.

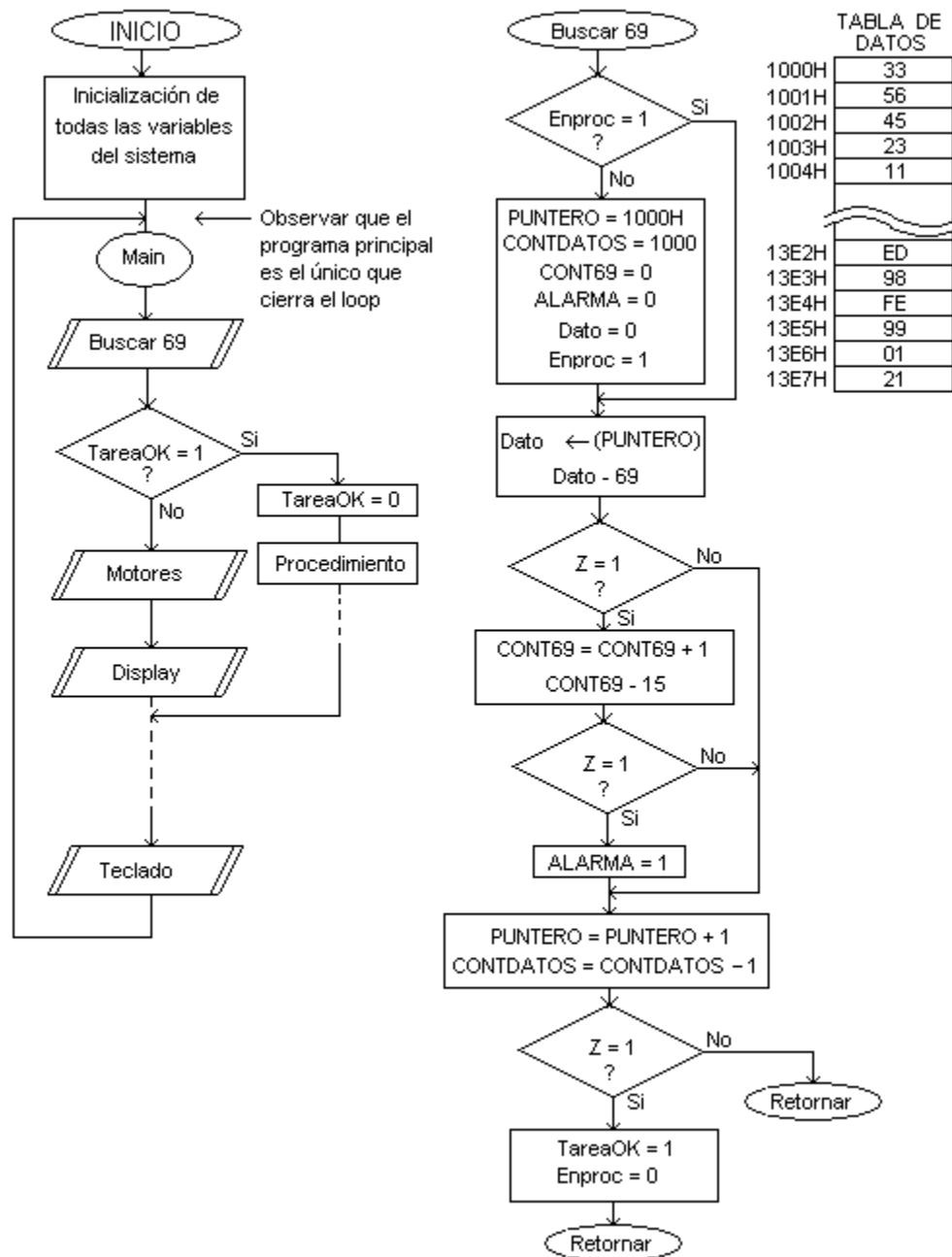
Por convención, las banderas son activas en el estado lógico “1” y deben ser inicializadas en la parte del programa que las usa. Las banderas deben restaurarse una vez han cumplido su servicio. Para la demostración del uso de las banderas se repetirán los ejercicios del numeral anterior.

- **Ejemplo1 mejorado:** La Figura 4.11 muestra la variación del ejercicio para encontrar los números 69H.

Observe como las banderas **Enproc** y **TareaOK** facilitan que el algoritmo no se quede iterando hasta analizar los 1000 datos, sino que permiten al programa estar atendiendo otros procesos y rutinas, como se ilustra en el *loop* principal del programa.

También es importante entender cómo las banderas son inicializadas y activadas de manera efectiva durante el proceso.

- **Ejemplo2 mejorado:** La Figura 4.12 muestra la variación del ejercicio para la detección de un flanco de bajada.



**Figura 4.11. Programa mejorado, para buscar dato en tabla con uso de banderas.**



**Figura 4.12. Programa mejorado, para detección de flanco de bajada.**

En este ejercicio se puede observar el trabajo de la bandera **Ya\_en\_1**, para retornar la máquina mientras no se haya dado la condición de señal en “1”. Cuando se cumpla esta condición, basta con esperar que la señal caiga (“0”) para validar el evento de flanco con el bit **Flanco\_OK**. La bandera **Flanco\_OK** es externa al algoritmo.

Observe que si el pulsador se daña, sea por desconexión o por cortocircuito, el sistema no queda bloqueado. Otra alternativa sería leer el pulsador como una entrada de interrupción y así eliminar el evento de hacer *polling* sobre la señal de entrada..

## 4.2. LA CODIFICACIÓN

Codificar es escribir un programa en algún lenguaje de programación, partiendo de un diagrama de flujo o de estado y en el peor de los casos de una idea (programación tipo “chorizo”).

La codificación de bajo nivel utiliza el *assembler* como lenguaje primario, debido a la cercanía del código al lenguaje de máquina de la CPU. Pero la codificación no es solamente la escritura de programas en mnemónicos, debido a que codificar implica tener orden y técnica. La idea es hacer que los programas sean fáciles de entender, mejorar y mantener. Todo programa en mnemónicos tiene una estructura, como se vió en el aparte 3.1. A continuación se presenta un ejemplo hecho para la máquina ColdFire® V1.

Sin importar lo que hace el programa ejemplo, se pueden distinguir claramente los cuatro campos de la estructura planteada en el capítulo anterior. Los comentarios que anteceden a las líneas de código son necesarios debido a que el lenguaje *assembler* da poca información de lo que se pretende hacer.

El encabezado principal es necesario, porque en éste se pueden distinguir datos tan importantes como el autor, fechas de creación, versiones, entre otros. También es importante reconocer los encabezados de cada subprocesso, como se detalla en la rutina de atención a la interrupción del temporizador allí codificada.

```
/*
***** PROGRAMA PARA PONER EN PRÁCTICA LOS MODOS DE DIRECCIONAMIENTO E INSTRUCCIONES *****
/*
/* Fecha: Noviembre 05 de 2007 */
/* Elaboró: Diego Múnera */
/* Versión: 1.0 */
/* Descripción: Este programa utiliza diferentes tipos de direccionamientos e instrucciones, para ilustrar lo visto en clase. Adicionalmente se atiende una interrupción por sobre flujo del TPM1. Es importante entender que el programa está hecho en Assembler, pero sobre un cuerpo de C.
*/
//INCLUDE PARA INTERRUPCIONES
#include <hidef.h> //INCLUDE PARA DECLARACIÓN DE REGISTROS DEL MCU
#include "derivative.h"

void main(void) {
    asm(
        //HABILITE INTERRUPCIONES GLOBALES:
        MOVE.W SR,D0          //LEA EL RESISTRO DE ESTADO SR
        AND.L #0xF8FF,D0        //ACLARA LA BANDERA I
        MOVE.W D0,SR            //ACTUALIZA REGISTRO SR

        //INHIBICIÓN DEL COP Y HABILITAR PIN DE RESET:
        MOVE.L #SOPT1,A0         //PUNTERO A REGISTRO SOPT1
        MOVE.B #0x17,(A0)        //INHIBE EL COP Y HABILITA PIN DE RESET

        //INICIALIZACIÓN PUERTO H:
        MOVE.L #PTHD,A0          //PUNTERO A REGISTRO PTHD
        MOVE.B #0x00,(A0)        //INICIALIZACIÓN DE PINES PUERTO H EN 0
        MOVE.L #PTHDD,A0          //PUNTERO A REGISTRO PTHDD
        MOVE.B #0x01,(A0)        //PIN 0 DEL PUERTO H COMO SALIDA
        MOVE.L #PTHPE,A0          //PUNTERO A REGISTRO PTHPE
        MOVE.B #0x02,(A0)        //HABILITA PULLUP PIN 1 PUERTO H
        MOVE.L #PTHD,A0          //PUNTERO A REGISTRO PTHD
        BSET.B #0,(A0)           //PIN 0 DEL PUERTO H EN 1

        //INICIALIZACIÓN DEL MÓDULO TPM1 PARA GENERAR UNA INTERRUPCIÓN EN EL SOBREFLUJO:
    );
}
```

```

MOVE.L #TPM1SC,A0          //PUNTERO A REGISTRO TPM1SC
MOVE.B #0x08,(A0)           //UTILIZA RELOJ DE BUS INTERNO Y PREESCALER DE 8
MOVE.L #TPM1MODH,A0         //PUNTERO A REGISTRO TPM1MODH
MOVE.B #0x09,(A0)           //INICIALIZA PARTE ALTA MÓDULO DE CONTEO
MOVE.L #TPM1MODL,A0         //PUNTERO A REGISTRO TPM1MODL
MOVE.B #0x06,(A0)           //INICIALIZA PARTE BAJA MÓDULO DE CONTEO
MOVE.L #TPM1SC,A0           //PUNTERO A REGISTRO TPM1SC
BSET.B #6,(A0)              //HABILITE INTERRUPCIÓN POR SOBREFLUJO DEL TPM1

//ESCRITURA DE BLOQUE DE 100 CELDAS DE RAM UTILIZANDO DIRECCIONAMIENTO INDIRECTO CON
//POSTINCREMENTO:
MOVE.L #100,D1               //PREPARA CICLO DE 100 ITERACIONES
MOVE.L #0x00800000,A1         //INICIALIZA PUNTERO A TABLA
CLR.L D2                     //NÚMERO A ALMACENAR

LOOP1:MOVE.B D2,(A1)+
ADD.L #1,D2                  //INCREMENTA NÚMERO A ALMACENAR
SUB.L #1,D1                  //DECREMENTA CONTADOR DE ITERACIONES
BNE LOOP1                    //SALTE SI NO ES CERO

//EJEMPLO CON DIRECCIONAMIENTO INDIRECTO CON ÍNDICE ESCALADO Y DESPLAZAMIENTO DE 8 BYTES:
//A2: PUNTERO INDIRECTO, D2: PUNTERO ÍNDICE, D1: CONTADOR DE ITERACIONES Y D5 DESTINO DE DATOS
MOVE.L #0x00800000,A2         //A2 APUNTA AL INICIO DE LA RAM
CLR.L D2                     //ACLARA PUNTERO ÍNDICE
CLR.L D5                     //ACLARA ALMACÉN DE DATOS
MOVE.L #10,D1                 //PREPARA CICLO DE 10 ITERACIONES

LOOP2:MOVE.B 0x08(A2,D2*4),D5 //MUEVE INFORMACIÓN
ADD.L #1,D2                  //INCREMENTA ÍNDICE
SUB.L #1,D1                  //DECREMENTA CONTADOR DE ITERACIONES
BNE LOOP2                    //SALTE SI NO ES CERO

)
for(;;) {                   //NO SE PUEDE SALIR DEL MAIN
}

/*********************************************
/*           FUNCIÓN DE ATENCIÓN A INTERRUPCIÓN POR TPM1           */
/*********************************************
interrupt VectorNumber_Vtpm1ovf void ISR_TIMER_OVF (void){
asm(
    MOVE.L #TPM1SC,A0          //PUNTERO A REGISTRO TPM1SC
    BCLR.B #7,(A0)              //ACLARE BANDERA POR SOBREFLUJO DE TPM1
)
}

```

De esta manera se distinguen tramos importantes del programa, orientando al lector a visualizar fácilmente partes importantes y compactas del mismo. Como parte de la disciplina de codificación se puede observar la estricta tabulación entre los campos. Se recomienda al lector asumir éstas técnicas de decodificación, aún aplicadas a otros lenguajes de programación como el C/C++, Visual Basic, entre otros.

#### 4.3. EL ENSAMBLE, COMPILACIÓN O TRADUCCIÓN

Pasar de los mnemónicos de las instrucciones a código de máquina es un proceso que para los programas en *assembler* se llama ensamblar, para los programas en C/C++ se llama compilación y para lenguajes de mayor nivel se llama traducción.

El ensamblador proporciona una interfase entre el programador y la arquitectura del microprocesador, para resolver los algoritmos de programación. El código resultante después de ensamblar un programa se llama código objeto.

Normalmente, un ensamblador ejecuta dos pasos en el ensamblaje de un programa. El primer paso consiste en armar una tabla con los mnemónicos de las instrucciones del programa, las equivalencias entre las etiquetas y los valores numéricos y una revisión de sintaxis del programa en *assembler*.

El segundo paso, consiste en verificar la validez de las instrucciones anotadas en la tabla anterior, verificar las direcciones asignadas en el programa contra las del mapa de memoria de la máquina y traducir instrucción por instrucción del programa almacenado.

Finalmente el ensamblador genera un reporte de los errores encontrados, tales como:

- Etiquetas duplicadas.
- Direcciones de memoria no implementadas.
- Instrucciones inválidas.
- Errores de sintaxis.
- Operandos fuera de margen.

Una vez superado el proceso de ensamblaje, el resultado que arroja el ensamblador es la generación de archivos tales como:

- **Archivo .OBJ:** Este archivo almacena el binario puro que interpreta la CPU para la ejecución del código. Los archivos .OBJ son la base de partida para generar el programa con el que finalmente se programará la memoria de código de la CPU o en el caso de un microcontrolador, su memoria interna.
- **Archivo .MAP:** Informa sobre la distribución y utilización de la totalidad de los recursos de la máquina, como la memoria y los periféricos.
- **Archivo .PRN o .LST:** Es un listado completo del programa en *assembler* con identificación de direcciones, mnemónicos y código de máquina para las operaciones.
- **Archivo .HEX o .S19 o .BIN:** es el archivo binario como lo interpreta una CPU, pero dentro de un formato elaborado por el fabricante del dispositivo a programar.

#### **4.4. LA DEPURACIÓN (Simulación en frío y caliente)**

La simulación es la prueba lógica que se hace sobre un programa, tratando de hacer que la máquina ejecute los procesos a una velocidad controlada por el programador y permitiendo la visualización de aquellas variables de interés. Ningún proceso de simulación predice con exactitud el comportamiento de un sistema y tampoco da garantía del exitoso funcionamiento de éste.

Existen varias formas de probar o simular un programa. La más económica es la prueba de escritorio, la cual se hace en el papel mediante un seguimiento de las variables de interés a lo largo de los diagramas de flujo. Este procedimiento es recomendable para pequeños algoritmos, ante la carencia de un medio de cómputo.

Otra forma de simular es mediante la ejecución de un *software* de simulación, que hace sumamente fácil el seguimiento de la lógica del programa y las restricciones de temporización del mismo. Los simuladores, generalmente, vienen incluidos en los paquetes de desarrollo ofrecidos por el proveedor de la máquina.

Los simuladores pueden ser en frío o en caliente (*in circuit*). Los simuladores en frío no permiten la conexión entre el *software* y el *hardware* del sistema, limitando la comprobación del buen desempeño del sistema entre un 60% y un 80%. El problema de la simulación en frío es que no predice el comportamiento de las variables físicas del sistema (eventos temporales, eléctricos, ruido, consumos, etc).

Los simuladores en caliente (ver Figura 4.13), permiten una comunicación entre la aplicación (*software* de simulación) y el sistema (*hardware*). Generalmente esta comunicación se hace utilizando un puerto serial de la máquina, a velocidades que oscilan entre 9600 baudios y 115000 baudios o un puerto USB de alta velocidad.

Una simulación en caliente garantiza entre un 80% y 95% el éxito de funcionamiento del sistema, el problema consiste en que se pierde la noción de “tiempo real” debido a que las acciones entre el programa y el circuito llevan el retardo de las comunicaciones seriales.



**Figura 4.13 Simulación en caliente.**

La mayoría de los simuladores tienen las siguientes características mínimas:

- **Ejecución paso a paso de las instrucciones**  
Permite ejecutar en forma secuencial y pausada las diferentes instrucciones del programa, permitiendo al usuario verificar la lógica y comportamiento de los algoritmos.
- **Ejecución rápida y ejecución continua**  
Cuando el programa está bastante maduro, se pueden adelantar procesos ya comprobados de una manera más ágil. La ejecución rápida permite que los movimientos del flujo del programa sean perceptibles. Por el contrario, la ejecución continua no permite que se visualicen los movimientos de la máquina durante la simulación y se conozca al detalle cada subprocesso (subrutina). Es necesario que el usuario detenga el proceso para visualizar los cambios en las variables del sistema.
- **Permiten colocar puntos de ruptura (*break points*) y verificación (*mark points*)**  
Un punto de ruptura es el lugar donde la CPU se detiene en el proceso de ejecución rápida o continua. Los *break points* son importantes porque permiten al programador establecer paros controlados de la ejecución del programa, con miras a la verificación de algunas variables del sistema o para adelantar procesos que ya han sido probados. Es posible establecer la cantidad de puntos de ruptura que el usuario estime conveniente, así como removérselos. Los puntos de verificación (*mark points*) se establecen de acuerdo a una condición especial del usuario, como por ejemplo la coincidencia de una dirección o del valor de un dato.
- **Visualización de todas las variables involucradas en el sistema**  
Permite mediante el uso de ventanas visualizar cualquier tipo de variable, en el formato más adecuado (binario, hexadecimal, octal o decimal) y adicionalmente modificar el valor de las mismas.
- **Visualización de los pines de control y puertos de la máquina**  
Estado lógico de pines de puertos, tanto de entrada como salida.
- **Ubicación de estímulos en la ejecución**  
Permite estimular variables de memoria o puertos, utilizando un archivo especializado que contiene los diferentes estímulos, lógicos, y el momento en el que deben ser ingresados al sistema.
- **Cálculo de tiempos**  
En la simulación, es posible medir el tiempo utilizando los ciclos de máquina o de reloj que toma un proceso. Se pueden hacer medidas del ancho de un pulso, la frecuencia de una señal o cuánto retardo introduce un proceso al programa. De ésta manera el programador conoce la variable tiempo y puede tomar decisiones, para que ésta no afecte el correcto funcionamiento del sistema.

- **La puesta a punto**

La puesta a punto involucra la integración del programa con el sistema. Cualquier falla que se detecte en éste proceso implica una revisión y modificación de los pasos anteriores. Muchas veces los cambios implican devolverse hasta el proceso de diseño de los algoritmos, es en éste punto donde se visualiza la importancia de tener un método y una disciplina en la programación. Lo anterior ayuda a que los cambios que se necesiten en los pasos anteriores no sean traumáticos y demanden gran cantidad de tiempo de desarrollo.

#### 4.5. OTRAS HERRAMIENTAS DE PROGRAMACIÓN Y SIMULACIÓN

Existen otras herramientas para ayudar al proceso de diseño de sistemas microprocesados y/o microcontrolados. Algunas de ellas sirven para verificar que la lógica y el *hardware* interactúen apropiadamente, para desarrollar los algoritmos en lenguaje de mayor nivel o como herramientas de programación de los dispositivos que contienen el código. A continuación se mencionan algunas de ellas y en el Capítulo 8 se detallan las herramientas CodeWarrior 6.1 y DEMOJM, que son las herramientas a usar en los ejercicios de este texto.

- **El emulador:** A diferencia de un simulador, el emulador permite la prueba del *software* y *hardware* de un sistema, en tiempo real. Esta importante herramienta permite emular pin a pin el procesador como si éste estuviera directamente conectado a un sistema, pero con las ventajas de un simulador en caliente.

Generalmente un emulador está compuesto de un *hardware* y un programa con iguales o mejores características que un simulador en caliente. El *hardware* se inserta directamente al sitio que ocupa el microprocesador a emular.

Para diseños electrónicos complejos, donde se requiere la intervención de un gran número de especialistas, el empleo de emuladores es bastante recomendado. El emulador es una herramienta útil no sólo en las etapas de depuración, sino como herramienta de productividad en la reducción de los retrasos originados en el desarrollo de nuevos productos.

- **Compiladores a C y otros:** Se dice que el lenguaje C/C++ es el lenguaje de los ingenieros, que desarrollan sobre sistemas embebidos a microcontroladores. Esto es cierto, siempre y cuando no se olvide que el *assembler* es el lenguaje más rápido y eficiente en el uso de recursos de la CPU, a la hora de optimizar procesos. Lo ideal sería combinar las bondades del *assembler* y el C/C++, utilizando compiladores a C/C++, es posible mezclar los dos lenguajes y lograr desarrollos óptimos. Los compiladores a C para microprocesadores y microcontroladores, traen los beneficios de los ensambladores y simuladores en una sola aplicación.

También, se deben resaltar aquellas ventajas relacionadas con la programación en lenguaje C, como instrucciones poderosas, aprovechamiento de la estructura del C, definición y manipulación de variables, portabilidad, etc.

El programador deberá tener especial cuidado a la hora de hacer su programa, cuidando de no consumir tiempo, energía y memoria innecesariamente.

- **Los programadores**

Un programador es un equipo electrónico al cual se le entrega un programa, en formato binario, con información del código que ejecutará un microprocesador o microcontrolador.

Los programadores pueden “quemar” la memoria de código, sea del tipo EPROM, EEPROM o FLASH; o si es del caso el procesador directamente. Los programadores actuales programan un gran número de éstos dispositivos y son llamados, en su mayoría, programadores universales industriales. También permiten hacer depuración y simulación en caliente.

La Figura 4.14 muestra un programador del tipo industrial para grandes volúmenes de producción y diferentes dispositivos de las familias ColdFire®, Power PC®, HC08, HCS08, HC12 y HCS12 de Freescale.



**Figura 4.14. Programadores industriales de Pemicro®.**  
**(Fuente: [www.pemicro.com](http://www.pemicro.com))**

Estos programadores permiten hacer programación y depuración, sin necesidad de estar conectado a un PC (*Stand Alone*).

El programador Spyder 08 de Freescale (ver Figura 4.15), permite hacer depuración para procesadores de 8 bits a muy bajo costo.

Otra herramienta muy popular, que se puede utilizar para las máquinas de 8, 16 y 32 bits (para microcontroladores de Freescale) es el programador USBMULTILINK de Pemicro, para bajo volumen (ver Figura 4.16).



**Figura 4.15. Programador Spyder 08 de Freescale®.**  
(Fuente: [www.freescale.com](http://www.freescale.com))



**Figura 4.16. Programador de bajo volúmen de Pemicro®.**  
(Fuente: [www.pemicro.com](http://www.pemicro.com))

#### 4.6. OTROS ASPECTOS A TENER EN CUENTA EN EL DISEÑO CON MCU'S

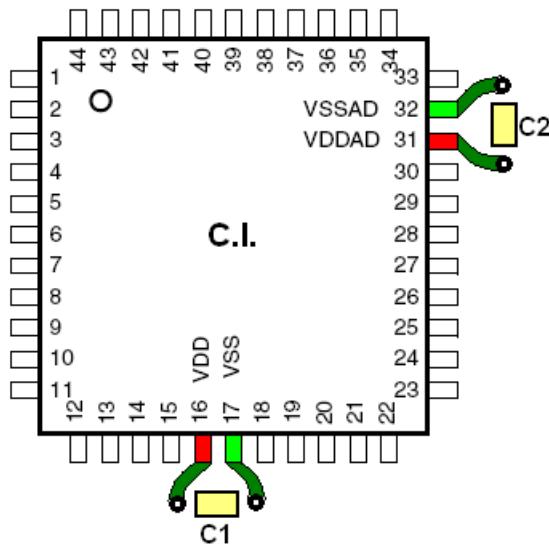
Las siguientes recomendaciones son importantes en el momento de diseñar circuitos, en donde se involucren microcontroladores y están orientadas, en gran medida, al diseño del PCB (*Printed Circuit Board*).

Toda medida que se tome para minimizar los efectos de la EMI (ElectroMagnetic Interference) y fortalecer el aspecto EMC (ElectroMagnetic Compatibility), es bienvenida en los diseños que involucran microcontroladores. El diseñador debe ser consciente sobre el hecho de que es imposible anular completamente las fuentes de ruido, pero si es posible mitigar notablemente sus efectos. A continuación se describen algunas técnicas

para combatir el los efectos de ruido sobre circuitos que contienen microcontroladores y están implementados sobre un PCB.

- **Ruido por acople de los circuitos integrados a la fuente:** Una de las principales vías de penetración de la EMI son los malos acoplos entre circuitos. Este efecto se produce debido a los transientes de tensión en los momentos de suicheo de la lógica. Para minimizar este efecto es recomendable ubicar un condensador en paralelo con la fuente de alimentación del circuito integrado, lo más cercano posible a los pines de la fuente (ver Figura 4.17).

Como un efecto adicional, estos capacitores mantienen una distribución apropiada de las fuentes de tal manera que permiten asegurar una tensión homogénea en los diferentes puntos del PCB.



**Figura 4.17. Ubicación de condensadores de desacople.**

Para el cálculo del capacitor dedesacople es importante considerar la frecuencia de conmutación de las señales más rápidas en el sistema, que por lo general están asociadas al reloj del circuito o con las señales de la lógica en sus cambios de bajo a alto o versus, y también están asociados a las impedancias de los circuitos.

La ecuación para el cálculo de los condensadores de desacople es, como criterio de capacitancia máxima es:

$$C_{max} = tr / 3.3 \times R_{th}$$

Donde:

**tr** = Tiempo de conmutación típico de la lógica o período del reloj del sistema.

**Rth** = Resistencia equivalente de Thevenin, teniendo en cuenta la resistencia de carga y la resistencia serie de la fuente de alimentación.

Suponiendo que se está trabajando con una lógica con conmutaciones de 20 ns y una resistencia equivalente de Thevenin de 1 Ω, el condensador de desacople recomendado sería de:

$$C_{max} = 20 \text{ ns} / 3.3 \times 1$$

$$C_{max} = 6.06 \text{ nF}$$

Como es un criterio máximo, se podría considerar un valor comercial cercano y sería de 4.7 nF (472).

**NOTA:** Observe que por lo general los condensadores más usados como desacople son de 0.1 uF o 0.01 uF, pero el autor recomienda hacer el cálculo con el objetivo de no instalar un condensador que poco servirá para mitigar el ruido por alta frecuencia.

- **Ruido por mal tratamiento de las tierras:** En lo posible, es necesario evitar largos caminos de retorno de tierra en los circuitos. La Figura 4.18 muestra una mala puesta a tierra de los diferentes circuitos de un sistema, en donde las corrientes de retorno se suman por cada derivación. El retorno a tierra de la señal de alimentación de los dispositivos depende de su distancia al nodo fuerte de tierra y en las derivaciones que de éste se hagan, en esa medida se formarán *loops* de impedancias, que forman un ambiente especial para la EMI.

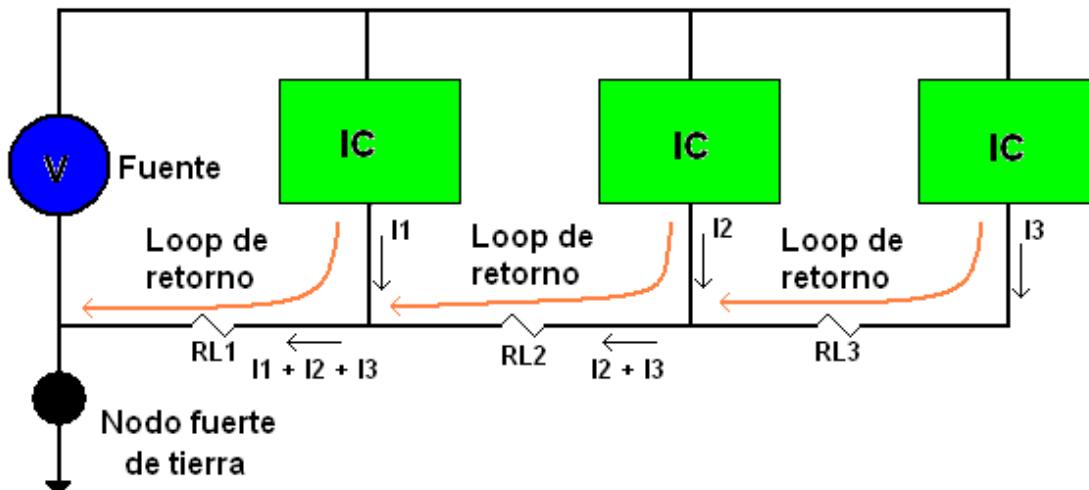


Figura 4.18. Circuito con malos retornos de tierra.

En la Figura 4.19 son presentadas varias soluciones para el problema del retorno a tierra. La idea es acortar los caminos de retorno de corrientes a tierra y buscar la forma más directa de llegar al nodo fuerte de tierra.

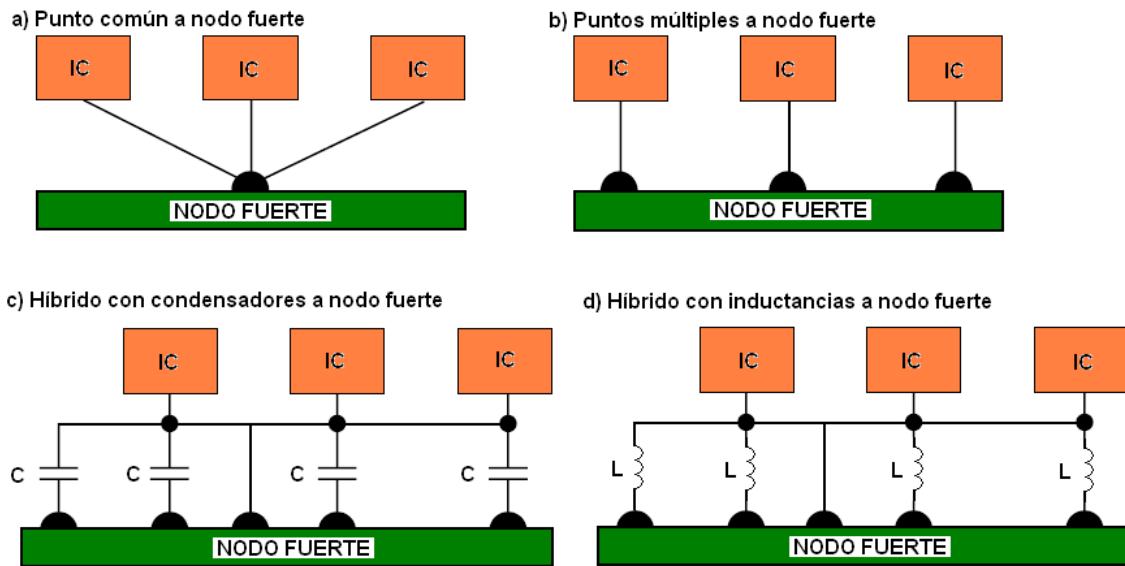


Figura 4.19. Circuitos recomendados para los retornos a tierra.

- **Ruido por mezcla de diferentes tipos de circuitos:** Para sistemas donde se combinen diferentes tipos de circuitos, como: análogos, digitales, de potencia, RF, entre otros, es recomendable hacer una separación física de los mismos. La Figura 4.20 ilustra sobre la separación de circuitos dependiendo de su tipo.

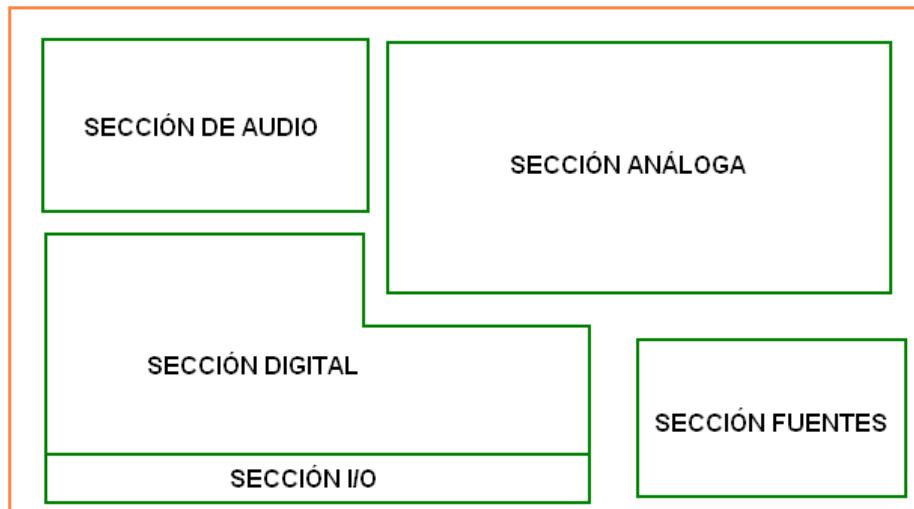
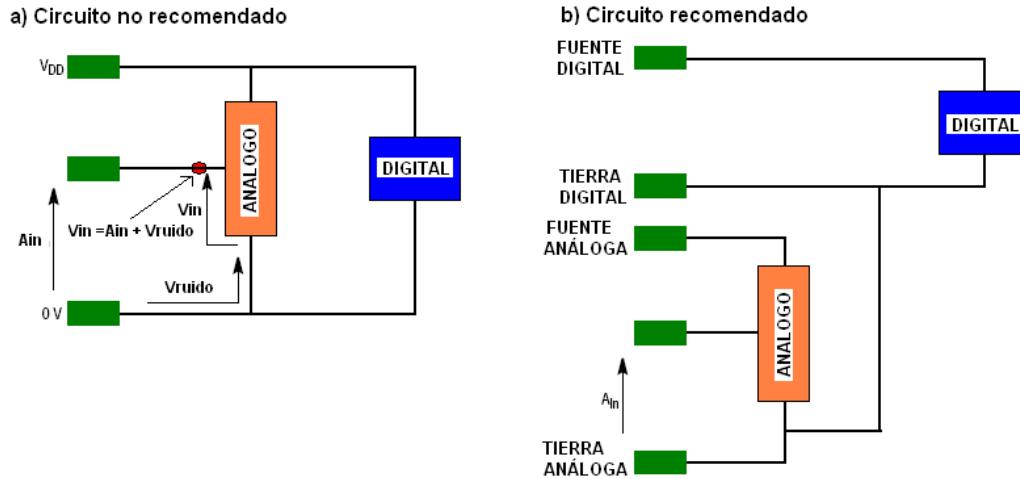


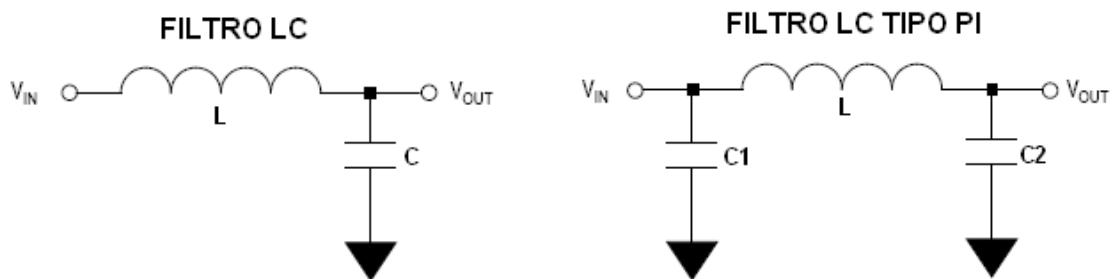
Figura 4.20. Separación de circuitos por tipo.

La separación de fuentes entre los diferentes tipos de circuitos es un factor importante. La Figura 4.21 muestra la recomendación en la separación de las fuentes.



**Figura 4.21. Separación de fuentes.**

La separación de fuentes mitiga la aparición de ruido debido a los circuitos de alimentación y digitales, dentro de los circuitos analógicos. Cuando no hay forma de independizar las fuentes la recomendación es adicionar filtros tipo LC, que eliminan componentes de alta frecuencia generada por conmutaciones. La Figura 4.22 ilustra sobre algunos tipos de filtros LC recomendados para eliminar ruido en fuentes compartidas.



**Figura 4.22. Filtrado LC para fuente de alimentación común.**

En donde la frecuencia de corte del filtro (sencillo) estaría en:

$$f_c = 1/(2 * \pi \sqrt{LC})$$

Esta frecuencia sería el corte a partir del cual el filtro atenuaría las componentes de ruido del sistema de alimentación. Una vez calculados los valores para el filtro, la

Figura 4.23 ilustra cómo se conecta para un sistema que comparte la fuente de alimentación.

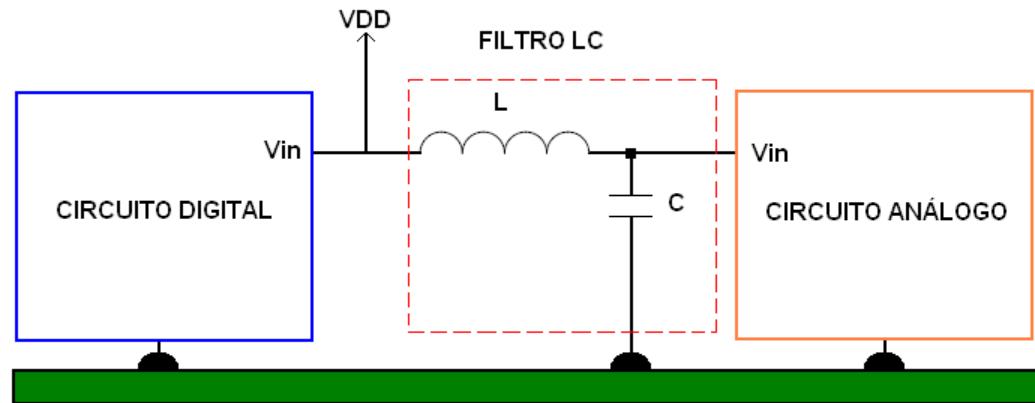


Figura 4.23. Ubicación Filtro LC para fuente de alimentación común.

En cuanto a las rutas para las tierras (para el caso de la alimentación simple), serían como las planteadas en la Figura 4.24.

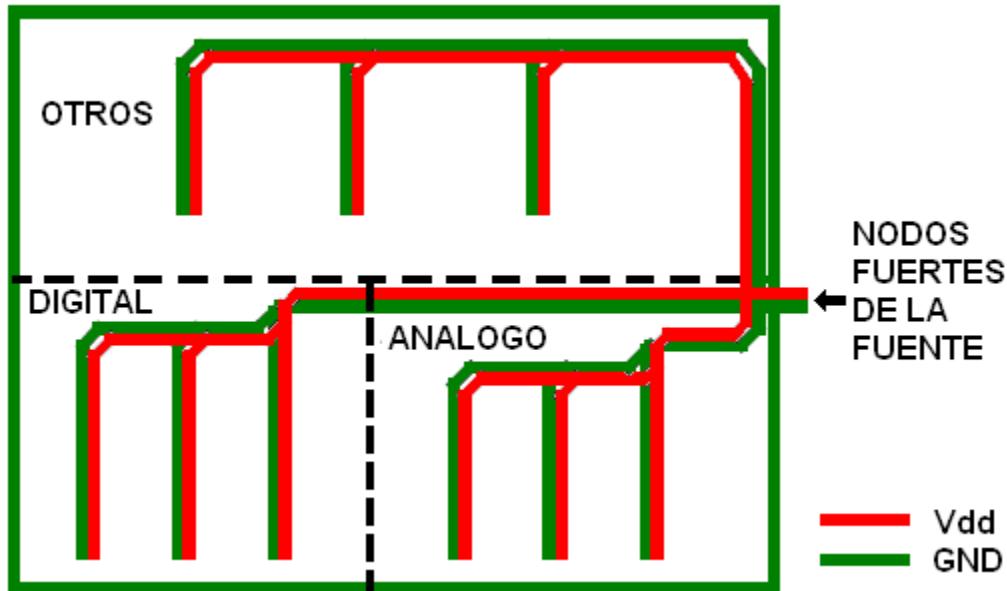
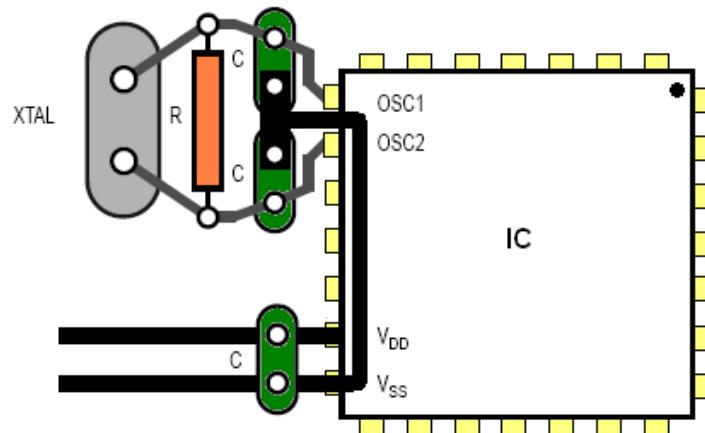


Figura 4.24. Ruteo de la tierra y la alimentación.

Otras recomendaciones importantes en el diseño de los PCB (*Printed Circuit Board*) serán presentadas a continuación.

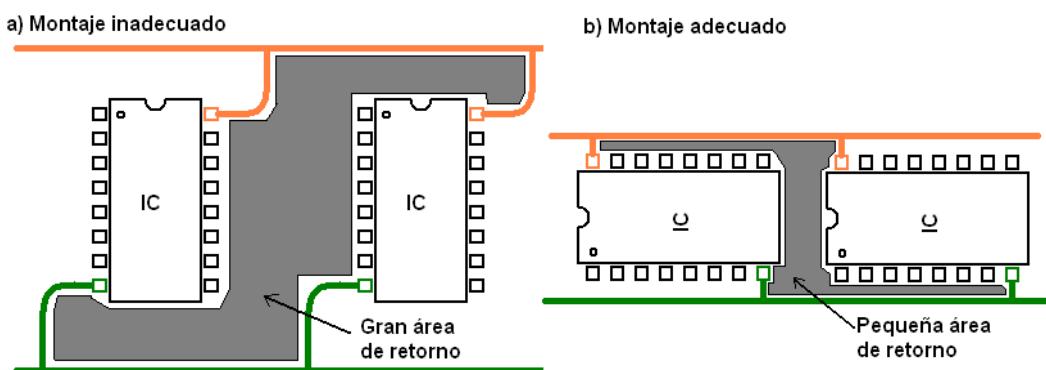
- **Montaje de reloj externo:** Sistemas con necesidad de implementación de reloj externo deberán ser tratados con especial cuidado, evitando ubicar el elemento oscilador muy alejado del punto de aplicación al MCU y evitando la ortogonalidad en los caminos del circuito de reloj . La Figura 4.25 ilustra sobre un montaje adecuado para un cristal de cuarzo como fuente de reloj de una MCU.



**Figura 4.25. Montaje en PCB de un reloj a cristal.**

- **Minimizar el área de retorno (loop) en los PCB :** Un área grande de retorno de las corrientes de la fuente hacia tierra, permite que fenómenos como las descargas por estática (ESD) o la formación de campos electro-magnéticos en la superficie del PCB.

La Figura 4.26 ilustra sobre un montaje con un área no apropiada de retorno y un circuito con un área adecuada de retorno.



**Figura 4.26. Área de retorno de corrientes de fuente.**

- **Distancia mínima de los puntos de referencia entre tierra y chasis:** Para el montaje mecánico del PCB y una superficie metálica, suelen usarse tornillos de fijación. Estos tornillos deben unir eléctricamente el potencial de tierra (GND) del circuito y el chasis (CHASSIS GND), siempre y cuando el sistema esté adecuadamente diseñado. Una recomendación es la de distribuir los tornillos de tal forma que la distancia mínima entre tornillos adyacentes no exceda los  $\lambda/20$  de la frecuencia más alta en el sistema.

**Ejemplo:** Para un circuito cuya señal de frecuencia más alta se espera que sea de 200 MHz, hallar la distancia mínima entre puntos de sujeción aplicando el criterio anterior.

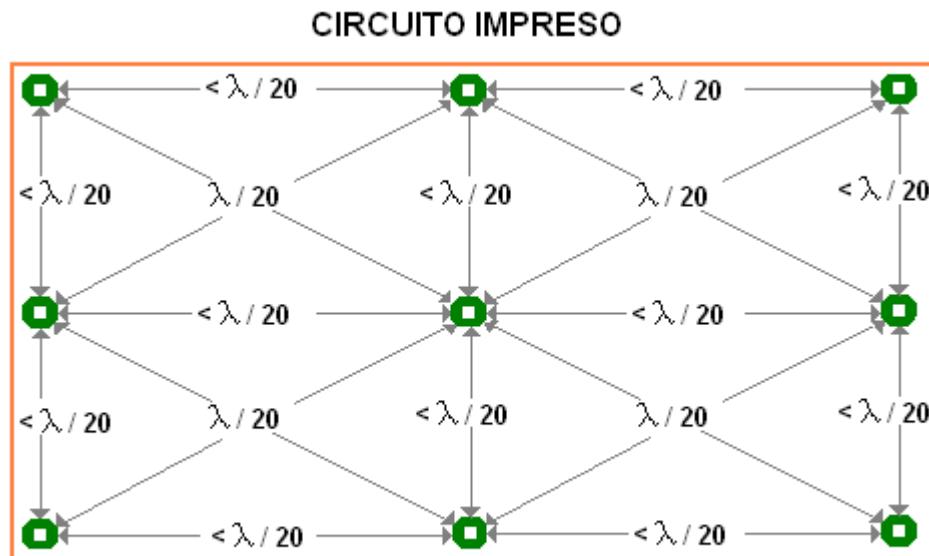
Sabiendo que  $\lambda = c / f$ , entonces, la longitud de onda sería de:

$$\lambda = 300000000 \text{ [km / seg]} / 200 \text{ MHz} = 1.50 \text{ mt}$$

Entonces, aplicando el criterio de  $\lambda/20$ :

$$\lambda / 20 = 7.5 \text{ cm.}$$

La Figura 4.27 ilustra una distribución apropiada de los puntos de contacto entre tierra y chasis, para el circuito del ejemplo.



Donde:  $\lambda / 20 = 7.5 \text{ cm}$

**Figura 4.27. Distribución de puntos de contacto GND – CHASSIS GND.**

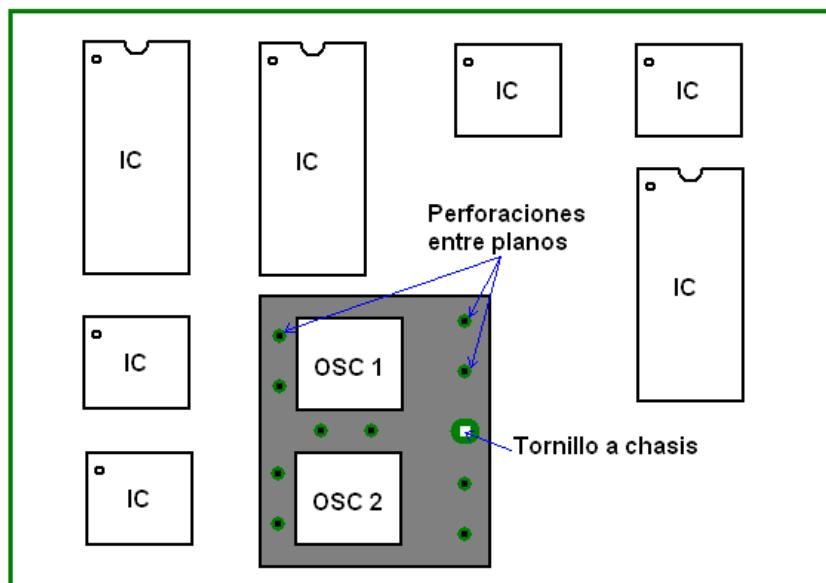
Para sistemas con señales de muy alta frecuencia el criterio de  $\lambda/20$  no es recomendado, debido a que las distancias podrían ser inferiores a 1 cm. La recomendación aquí sería el diseño de planos de tierra, anillos o mallas sobre el PCB. probablemente las capas del PCB deberán ser dos o más, con el fin de poder ubicar planos de tierra que sirvan como apantallamiento para las señales perturbadoras.

- **Planos de imagen:** Toda área de cobre, sea la tierra, voltaje o chasis, puede formar un potencial plano que suministra una baja impedancia para las corrientes de RF en su retorno a tierra (retorno de flujos). Esta técnica es bastante apropiada para combatir la emisiones por EMI.

Pero no siempre estos planos son efectivos para sistemas que manipulan frecuencias altas y se presenta el fenómeno conocido como efecto piel. El efecto piel se refiere a la corriente que circula por la capa más superficial de un material (piel) y que no circula por el medio del material o conductor. Como resultado de este fenómeno se pueden producir radiaciones EMI.

Una solución al problema sería establecer planos paralelos al plano de imagen, teniendo cuidado de no cruzar líneas por éste.

- **Ubicación de planos de tierra:** La idea es capturar los flujos de RF causados por los circuitos de oscilación como relojes, *buffers*, *drivers*, etc. El plano deberá ser localizado en el lado de los componentes y conectado directamente al nodo fuerte de tierra del sistema. La conexión deberá garantizar que al menos la tierra del oscilador y dos vías adicionales deberán establecerse con el nodo de tierra (ver Figura 4.28).



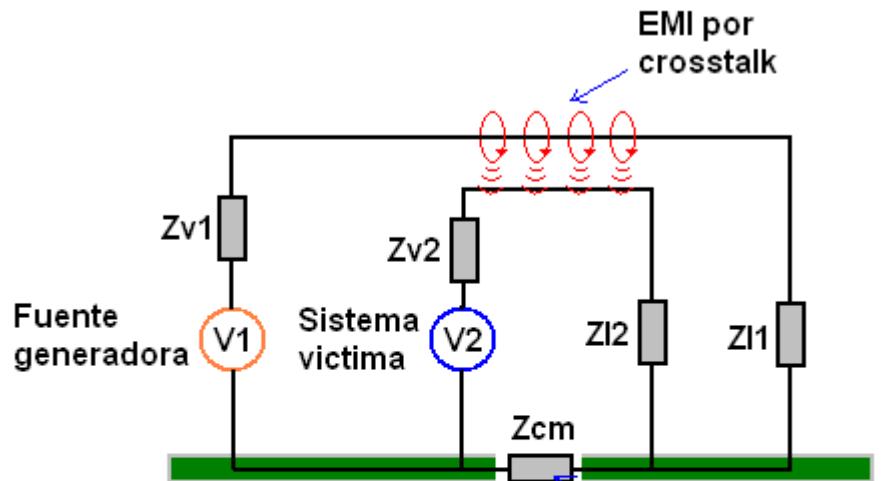
**Figura 4.28. Ubicación de planos de tierra.**

Como recomendaciones, el usuario deberá garantizar:

- No cruzar caminos por el plano de tierra en el lado de componentes, ni por el plano paralelo de tierra principal.
  - El plano de tierra no deberá tener capa de *anti-solder*.
- **Evitando la inducción entre conductores (*crosstalk*):** Este fenómeno se presenta al formarse campos electromagnéticos entre conductores, causado por corrientes en el circuito.

El *crosstalk* aparece tanto en las fuentes como en los receptores ubicados en las cercanías del sistema, y es considerado como un problema de EMI al interior de éste. El fenómeno es no deseado y puede provenir de señales de reloj, datos, direcciones y entradas y salidas digitales (I/O).

Para que se presente la inducción son necesarios por lo menos tres conductores en el circuito (ver Figura 4.29), en donde dos líneas portan la señal y la tercera es la referencia del sistema. V1 es la señal portadora y V2 es la señal afectada por efecto de inducción por *crosstalk*.



Donde:

$Z_{v1}$ : impedancia de la fuente generadora

$Z_{v2}$ : señal afectada

$Z_{l1}$ : impedancia de carga de la fuente

$Z_{l2}$ : impedancia de la señal

$Z_{cm}$ : impedancia en modo común

Impedancia  
en modo  
común

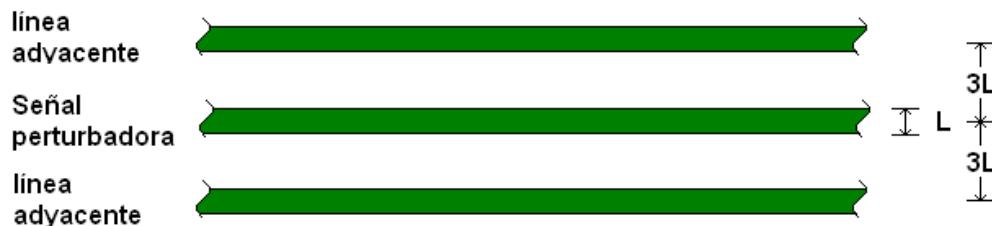
Figura 4.29. EMI por *crosstalk*.

La principal causa de éste fenómeno es la impedancia  $Z_{cm}$ , que se forma por el desbalance en modo común de  $V_1$  y  $V_2$ . Es por lo anterior que mientras más pequeña sea  $Z_{cm}$ , menor será el efecto de EMI causado por el *crosstalk*.

Para prevenir el efecto de *crosstalk*, existen una serie de recomendaciones de diseño del PCB y que a continuación se listan:

- Construir las tierras de acuerdo al tipo de familia de los circuitos integrados.
- Minimizar la distancia física entre componentes.
- Minimizar líneas que viajen en forma paralela, en especial si son de características diferentes (análogas, fuentes, digitales, etc).
- Localizar los componentes alejados de lineas de entrada y salida (I/O).
- Ajustar impedancias, ubicando terminales de ajuste (*matching*).
- **En líneas paralelas, proveer suficiente distancia para evitar la auto inducción.**
- Hacer que el cruce de rutas sea ortogonal para prevenir la formación de capacitancias de acople entre estas.
- Reducir al máximo la exagerada separación de los retornos de tierra, de las señales del sistema.
- Reducir la impedancia de los caminos y los niveles de las señales conducidas.

La técnica más recomendada, para reducir la inducción por *crosstalk*, es la resaltada en negrilla (**En líneas paralelas, proveer suficiente distancia para evitar la auto inducción**) y la distancia mínima se calcula como tres veces el ancho del camino que conduce la señal, medida de centro a centro (ver Figura 4.30).



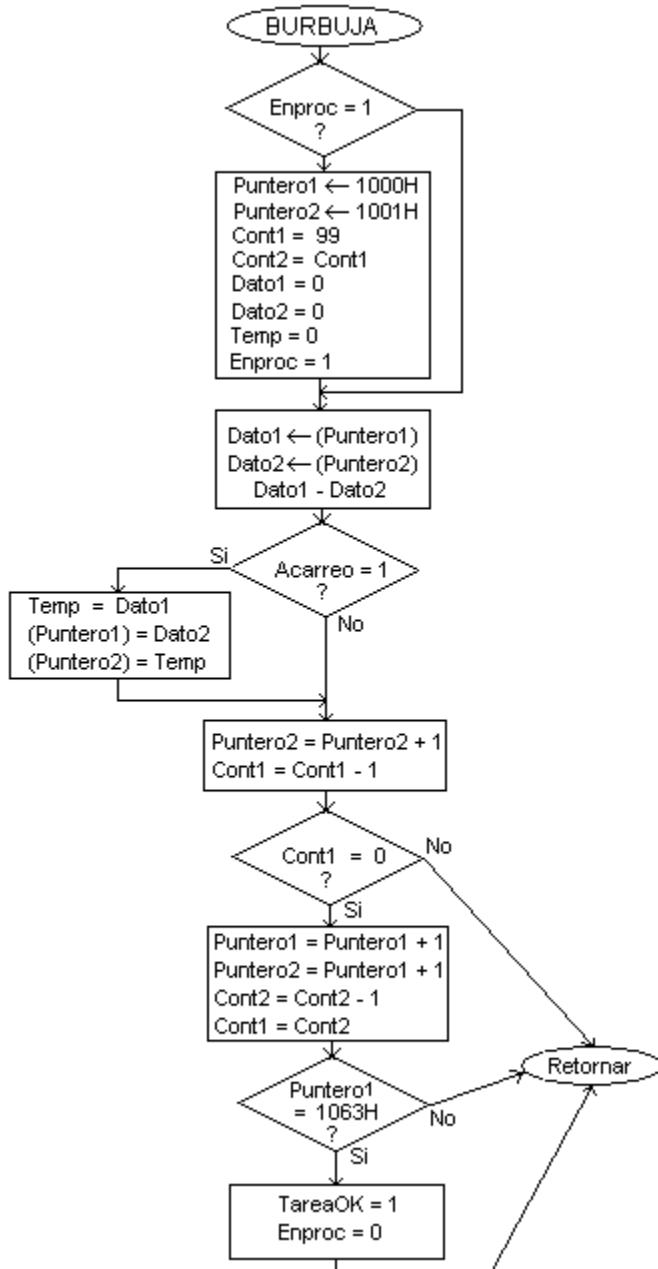
**Figura 4.30. Distancia mínima para minimizar *crosstalk*.**

## 4.7. REFERENCIAS

- Mardigian, Michel. EMI Troubleshooting Techniques. McGraw Hill. USA, 2000.
- Montrose, Mark. EMC and the Printed Circuit Board. IEEE. New York, 1999.
- Carr, Joseph. The Technician's EMI Handbook. Newnes. Boston, 2000.
- Ott, Henry. Noise Reduction Techniques in Electronic Systems. Wiley. New York, 1988.
- Glenewinkley, Mark. System Design and Layout Techniques for Noise Reduction in MCU-Based Systems. Freescale Semiconductor. Austin, 1995.

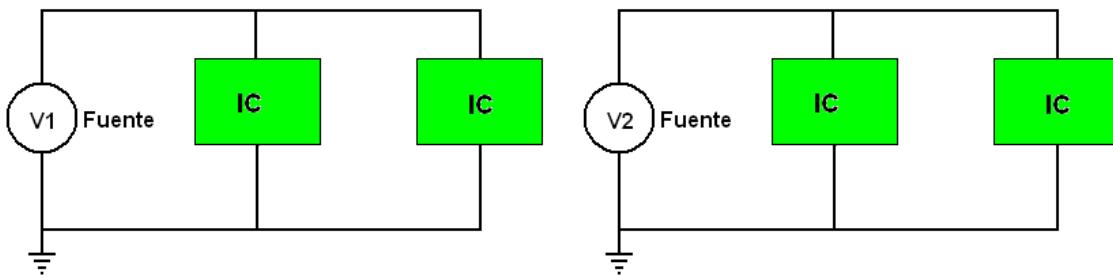
## 4.8. PREGUNTAS

- ¿Por qué no es recomendable un ingeniero que carezca de método para programación con sistemas embebidos a MCU?
- Enuncie al menos cuatro posibles causas de la presencia de EMI.
- Describa al menos cuatro posibles recomendaciones para mitigar el ruido por EMI.
- ¿Qué se considera una variable del tipo vital, en diseños con sistemas MCU?
- Defina algoritmo.
- Enuncie cinco razones para que un algoritmo se considere eficiente.
- Describa tres partes representativas que conforman un algoritmo.
- ¿Por qué son importantes las banderas de usuario en un algoritmo?
- Diseñar, utilizando diagramas de flujo, un algoritmo que cuente cuantos números primos (en 8 bits) hay en una lista de 1000 datos almacenados en memoria. El algoritmo no deberá iterar al interior del mismo.
- ¿Cuál es el lenguaje de más bajo nivel, utilizado para codificar programas en un sistema embebido a MCU?
- ¿Qué información, como mínimo, deberá ir en el encabezado del programa principal de un proyecto?
- ¿Cuál es la diferencia fundamental entre ensamblar, compilar y traducir?
- Describa los dos pasos que realiza un ensamblador.
- Enuncie tres tipos de error que puede detectar un ensamblador.
- ¿Qué contiene un archivo .PRN?
- ¿Cuál es la diferencia fundamental entre la depuración en frío y en caliente?
- ¿Qué ventajas tiene ejecutar un programa paso a paso, desde un simulador?
- ¿Cuál es la diferencia entre un punto de ruptura (breakpoint) y un punto de marca (markpoint)?
- Defina: proceso de puesta a punto (*set point*).
- ¿Cuál es la diferencia fundamental entre un simulador en caliente y un emulador?
- Desde el punto de vista de la forma, ¿Cuántos y cuales errores se pueden apreciar en el siguiente diagrama flujo? (Ver Figura 4.31).



**Figura 4.31.**

- ¿Cuál es la causa principal del fenómeno de ruido por acople, entre los circuitos integrados y la fuente del sistema?
- Para una frecuencia de conmutación de 100MHz, de las señales digitales de un circuito, calcular el capacitor de desacople necesario para mitigar este fenómeno. Asumir una impedancia de Thevenin de  $4\Omega$ .
- Para el circuito de la Figura 4.32, sugerir una conexión adecuada para las tierras. V1 y V2 son potenciales diferentes, pero referidos a la misma tierra.



**Figura 4.32.**

- ¿Cuál es la importancia de separar las fuentes para diferentes tipos de circuitos (digital, análogo, etc)?
- ¿Qué tipo de circuito se recomienda para lograr la separación de fuentes?
- ¿Qué fenómenos se producen en áreas de retorno muy grandes, en un PCB?
- Cuál es el criterio para la distancia mínima entre puntos de sujeción a la tierra de chasis?
- Para un circuito cuya señal de frecuencia más alta es 1GHz, calcular la distancia mínima entre puntos de sujeción a tierra.
- ¿Cuál es la idea de utilizar planos de tierra en los PCB?
- ¿Por qué se presenta el fenómeno de *crosstalk* en un sistema?
- ¿Cuál es la recomendación más importante para mitigar ruido por efecto de *crosstalk*?

# **PARTE II**

## **ARQUITECTURA DE LOS MICROCONTROLADORES DE 32 BITS COLDFIRE® V1**

### **OBJETIVO**

Esta parte tiene como objetivo fundamental el estudio de la arquitectura Freescale de 32 bits llamada ColdFire® V1, haciendo un especial énfasis en los modelos de programación y en los recursos que se dispone para desarrollar sobre estas máquinas.

La Compañía Freescale ha trabajado el concepto FLEXIS, mediante el cual es posible dar el salto de las máquinas de 8 bits a las de 32, sin mayores dificultades y con una compatibilidad del 100%. Es por esto que en esta parte aparece dicho concepto y en la parte III Capítulo 8.6, se hará la demostración de cómo hacer la migración.

### **CONTENIDOS**

#### **CAPÍTULO 5. La familia de los microcontroladores ColdFire® V1**

En este capítulo es presentada, de una manera resumida, la arquitectura de las máquinas ColdFire® V1. Aparece la definición de aspectos tan importantes como:

- El modelo de programación
- La memoria
- El proceso de excepciones
- La pila

#### **CAPÍTULO 6. Modos de direccionamiento y juego de instrucciones**

Con ejemplos elementales, son ilustrados los diferentes modos de direccionamiento y las instrucciones de las máquinas ColdFire® V1.

#### **CAPÍTULO 7. Migración de 8 a 32 bits**

De una manera muy resumida, es presentada la estrategia FLEXIS de Freescale. Resaltando aspectos como la compatibilidad de *hardware* y *software* entre las máquinas de 8 y 32 bits.

# CAPÍTULO 5

## La Familia de los Microcontroladores de 32 bits ColdFire® V1

### **5.1. Introducción a la arquitectura ColdFire® V1 de 32 bits**

- Características generales
- Arquitectura del núcleo
- Modos de operación
- Modelo de programación

### **5.2. Organización de la memoria**

### **5.3. Procesamiento de Excepciones**

### **5.4. La pila y el puntero a pila**

### **5.5. Referencias**

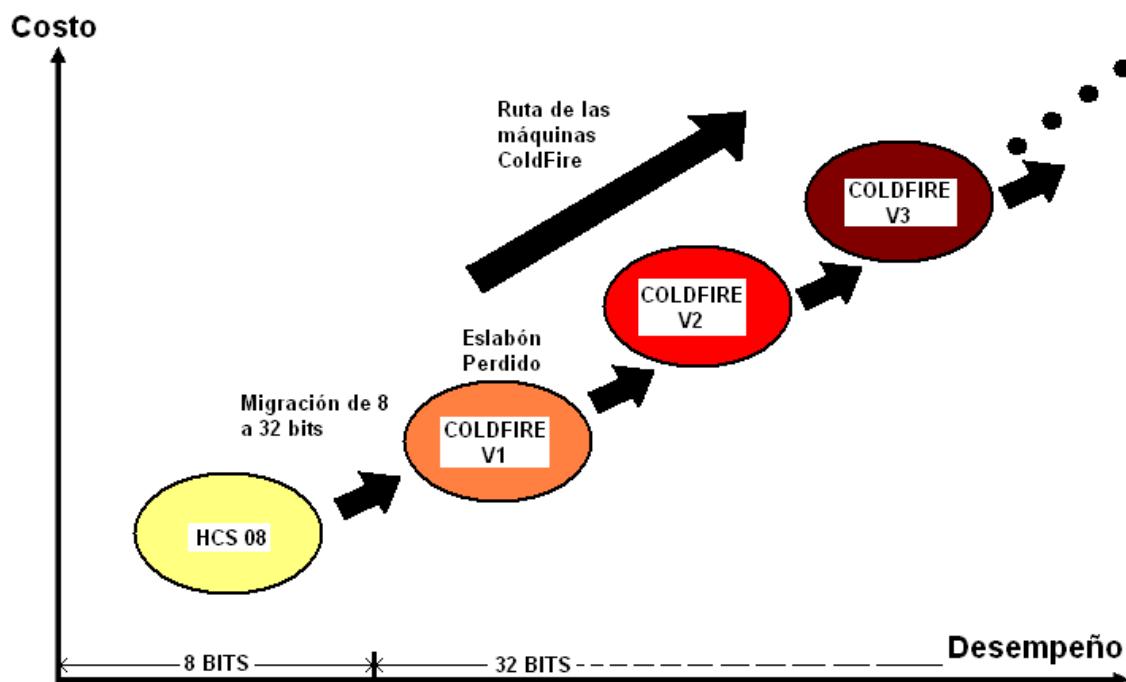
### **5.6. Preguntas**

## 5.1. INTRODUCCIÓN A LA ARQUITECTURA COLDFIRE® V1 DE 32 BITS

Como respuesta a la saturación de las capacidades y desempeño de las máquinas de 8 y 16 bits, para ciertas aplicaciones, las compañías ubicadas a la vanguardia del diseño de microcontroladores como Atmel, Freescale, Renesas, STMicroelectronics, Oki, entre otras, están desarrollando soluciones en 32 bits de bajo costo y muy bajo consumo, para atraer a los diseñadores.

Una de las opciones más fuertes es la llamada *Controller Continuum* de Freescale. Esta Compañía plantea la evolución migratoria desde las máquinas de 8 bits hacia las arquitecturas fuertes de 32 bits. Parte del proceso consistía en encontrar el eslabón perdido, al cual se le dedica gran parte de este texto y se refiere a los procesadores con arquitectura ColdFire® V1. La Figura 5.1 muestra la evolución planteada por Freescale.

**NOTA:** En éste capítulo aparecerá frecuentemente el concepto de **proceso de excepción**, que no es más que el mecanismo que utiliza el MCU para atender todo evento no esperado o programado, como las interrupciones y las fuentes que generan RESET. El concepto será ampliado y tratado en el aparte 5.3.



**Figura 5.1. Migración de 8 a 32 bits**

Inicialmente serán mantenidas las compatibilidades pin a pin de los circuitos integrados, así como las interfaces de los periféricos internos.

En este libro se estudiará la arquitectura interna de la máquina ColdFire® V1 y su plataforma, así como su arquitectura de programación, pero no será tratada a fondo su arquitectura de depuración debido al nivel y el enfoque del texto.

### Características generales

Los tres principios importantes en el desarrollo de las máquinas ColdFire® V1 son:

- **Su núcleo:** De tamaño reducido, la más baja disipación de potencia y el más amplio desempeño.
- **Periféricos y distribución de pines:** Compatibilidad total con las máquinas de 8 bits de la familia HCS08.
- **Segmentación (*Pipeline*):** Derivada de la familia ColdFire® V2, en forma simplificada. Esta característica le proporciona al núcleo un alto desempeño.

La máquina ColdFire® V1 viene implementada sobre una arquitectura de programación llamada ISA-C, como una derivación de las arquitecturas ISA-A e ISA-B de máquinas superiores. La modificación consiste en poder soportar los periféricos de las máquinas de 8 bits y así generar la compatibilidad con la familia HCS08, conjuntamente con la manipulación de datos en 8 y 16 bits.

La arquitectura ISA nace junto con el diseño de la máquina M68000, que está orientada al tratamiento de datos en 32 bits del tipo entero (*integer*) para lenguajes de alto nivel, destacándose por su baja complejidad y costo. Los aportes nuevos de la arquitectura ISA-C se pueden resumir en:

- Soporta tratamiento de datos tipo byte, word y long, sobre instrucciones de movimiento y comparación.

Ejemplo: La instrucción **MOVE**, que significa mover información desde una fuente hacia un destino, puede ser ejecutada de las siguientes 3 maneras:

**MOVE.B:** Mover un dato tipo byte.

**MOVE.W:** Mover un dato tipo word.

**MOVE.L:** Mover un dato tipo long.

- Posicionamiento de código independiente.
- Algunos tipos de operadores de manipulación de bits.

La mitad de los registros de propósito general Rn, divididos en: tipo dato (Dn) y tipo dirección (An), pasaron de ser 16 a 8. La arquitectura ISA-C utiliza desde D0 hasta D7 para los registros tipo dato y desde A0 hasta A7 para los registros tipo dirección.

La máquina ColdFire® V1 tiene total compatibilidad de pines respecto de las máquinas HCS08. En este sentido, existe un considerable sacrificio en la interfase de depuración respecto de las máquinas ColdFire® V2 y superiores. Al existir un sólo pin para la depuración (Interfase BDM), los procesadores ColdFire® V1 presentan los siguientes cambios:

- Se pasa de una depuración a tres pines del tipo *Full Duplex*, a un protocolo de un sólo pin del tipo *half duplex*. Este sacrificio redonda en una menor velocidad de depuración y una pérdida del concepto de “tiempo real”.

- La reducción del paquete de dato de depuración de 17 bits a 8 bits, hace que se modifique el protocolo clásico de depuración y se empleen técnicas de compensación para las capacidades de *trace* y *breakpoints* en los eventos de depuración.

Lo interesante de el sacrificio en el mecanismo de depuración es poder utilizar las mismas herramientas para las máquinas de 8 bits (familia HCS08) y 32 bits (familia ColdFire® V1), lográndose la migración al 100% de una tecnología a otra.

Otras características generales son:

- Modelo simplificado del modo supervisor<sup>6</sup>, conteniendo un manejo aparte de puntero a pila, un registro base para vectorización de eventos y un registro de configuración de la CPU.
- Soporta módulos opcionales<sup>7</sup> (no vienen con la versión inicial) como las de división de enteros, multiplicadoras/acumuladoras (MAC,EMAC), unidad de aceleración criptográfica (CAU), entre otras.
- Respuesta programable ante la ejecución de código ilegal o decodificación de direcciones de memoria no implementadas. Estos eventos se pueden tratar como de excepción o de generación de RESET.
- Hasta 50 MHz de velocidad de procesamiento del núcleo sobre una tecnología de 0.25 micrones.
- Con una marca de 0.85 Dhrystones<sup>8</sup>, 2.1 MIPS (*Millions of Instructions Per Second*) cuando se ejecuta en FLASH y 1.05 DMIPS (Dhystone MIPS) cuando se ejecuta en RAM.
- FLASH de dos ciclos de acceso, con bajo consumo de energía. El controlador de la FLASH permite acceso por especulación como técnica de reducción de tiempos y eficiencia en la ejecución del programa del usuario.
- RAM con un ciclo de acceso, implementada en la plataforma del procesador sobre un bus de alta velocidad.
- Controlador de interrupciones con las siguientes características:
  - Mapeado de los periféricos por fuera de la plataforma del módulo esclavo. Consiste de 64 bytes localizados al final de la memoria direccionable (0x(FF)FF\_FFC0 – 0x(FF)FF\_FFFF). El modelo de programación es accesado mediante el espacio declarado para los periféricos. El nivel de las interrupciones es codificado y el vector de atención es enviado directamente al núcleo del procesador.
  - El controlador soporta directamente 30 interrupciones de periféricos más 7 interrupciones por software (SWI).
  - Asociación fija entre las fuentes que requieren atención por interrupción y el nivel de interrupción con su prioridad de atención. Existen 30 interrupciones a las que se les asignan 7 niveles de importancia y 9 prioridades por nivel. El sistema acoge al 100% la estructura de prioridades de las máquinas de 8 bits de la familia

---

<sup>6</sup> Modo de puerta trasera (*background mode*).

<sup>7</sup> Se prevén a partir del 2008.

<sup>8</sup> Dhystone por segundo = reloj del procesador \* número de pasadas / tiempo de ejecución.

- HCS08. Es posible re-mapear hasta dos requerimientos de interrupción, hacia los niveles más altos de interrupción no enmascarable, asignando una prioridad.
- Se asigna un único número de vector<sup>9</sup> por cada fuente de interrupción, mediante la siguiente ecuación:

$$\text{Número del vector en ColdFire®_V1} = \text{Número del vector en HCS08} + 62$$

## Arquitectura del núcleo

Como se detalla en la Figura 5.2, la característica más importante del núcleo del procesador ColdFire® V1 es su segmentación de cauce (*Pipeline*). El núcleo está formado por dos estructuras independientes de *pipeline* con una interfase de bus unificada, para maximización del desempeño con una reducción del *hardware* del núcleo, lográndose una significativa reducción del costo<sup>10</sup>.

Las dos estructuras de segmentación de cauce son:

- **IFP (Instruction Fetch Pipeline):** Consiste de un cauce de dos niveles para la pre-búsqueda de las instrucciones que luego son ingresadas al OEP, como se define a continuación.
- **OEP (Operand Execution Pipeline):** Consiste de un cauce de dos niveles, en el cual se decodifica la instrucción, se capturan los operandos desde el IFP y se ejecuta la función asociada a la instrucción.

Ambas estructuras están desacopladas mediante el empleo de un *buffer*, que opera como una lista tipo FIFO, en donde el IFP actúa como un dispensador en la pre-búsqueda de las instrucciones que serán servidas al OEP, minimizando la pérdida de tiempo por paros en la espera de instrucciones.

En la pre-búsqueda realizada por el IFP, se pueden distinguir las siguientes etapas:

- **IAG (Instruction Address Generation):** Etapa encargada de decodificar la instrucción a ser ejecutada.
- **IC (Instruction fetch Cycle):** Etapa de generación de los ciclos de reloj necesarios para la búsqueda de la instrucción.
- **IB: (Instruction Buffer):** Etapa para el almacenamiento en la lista FIFO de las instrucciones en espera de ejecución.

La estructura de ejecución OEP está implementada sobre un flujo tipo RISC (*Reduced Instruction Set Computer*), soportada por un registro de archivo de lectura dual (RGF) conectado a la unidad aritmética y lógica (ALU). En la estructura son distinguibles las siguientes etapas:

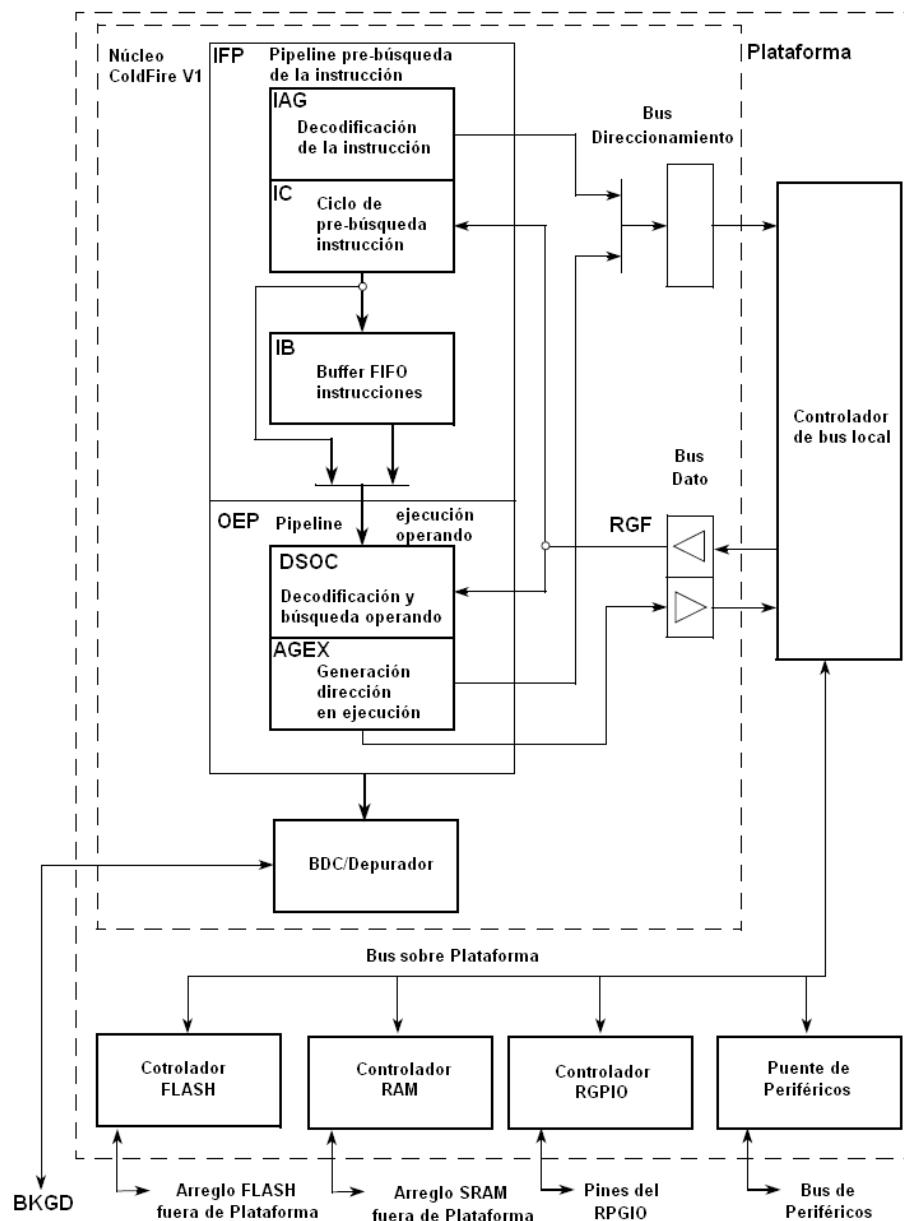
---

<sup>9</sup> Un vector es el lugar en la memoria de un ancho de 32 bits, que contiene la dirección en donde será atendido un evento de excepción. El procesador contiene una tabla de vectores cuya dirección origen es dinámica en vecindarios de 1MB.

<sup>10</sup> Es importante mantener siempre presente que la migración de 8 a 32 bits no debe representar un aumento significativo en el costo de las máquinas, de hecho se prevé la desaparición paulatina de los sistemas de 8 al igualarse el costo de los *chips*.

- **DSOC (Decode & Select Operand Cycle):** Ciclo de decodificación y selección de los operandos.
- **AGEX (Address Generation Execute Cycle):** Ciclo de generación de la dirección después de ejecución. Determina la dirección destino, como resultado de la operación ejecutada.

Con el objetivo de minimizar el costo en la fabricación del circuito integrado, el bus de direcciones del núcleo de las máquinas V1 ha sido reducido de 32 bits a 24 bits. De lo anterior se deduce que la máxima capacidad direccionable por los procesadores ColdFire® V1 es de 16 MB. La Figura 5.2 muestra un diagrama en bloques resumido de la arquitectura del núcleo de la máquina ColdFire® V1.



**Figura 5.2. Núcleo de la familia ColdFire® V1**

Otros bloques que conforman la arquitectura ColdFire® V1 son:

- BDC/Depurador (*Background Debug Controller*): Interconectado al mundo exterior vía pin BKGD. Este módulo permite hacer proceso de depuración elaborando operaciones de *trace* y *breakpoints*. Como ya se ha mencionado anteriormente, este texto trata superficialmente el tema.
- Controlador de la FLASH: Entendida como la memoria que contiene el programa de usuario y datos de sólo lectura.
- Controlador de la RAM: Memoria de datos, tanto de escritura como lectura.
- Controlador GPIO: Manipula un modulo de 16 bits de entradas y salidas (I/O) de propósito general, de acceso rápido y eficiente.

## Modos de operación

Las máquinas ColdFire® V1 tienen la capacidad de trabajar en distintos modos de funcionamiento (Ver tabla 5.1), dentro de los cuales se pueden mencionar:

- **Modo de depuración para desarrollo de código:** Manejado directamente por el módulo BDC (*Background Debug Controller*), el trabajo importante de este modo es el de poder analizar las operaciones del microcontrolador durante la ejecución del software del usuario. Por medio de este modo es posible descargar el *bootloader*<sup>11</sup> o la aplicación del usuario dentro de la memoria FLASH. Este modo también puede ser usado para borrar y reprogramar la FLASH después de que esta ha sido programada con anterioridad.
- **Modo seguro:** Mientras que el microcontrolador se encuentre en modo seguro, existen restricciones aplicadas a los comandos del depurador.
- **Modo RUN:** Es el modo normal de operación y el más común, porque es el modo en el que el usuario puede ejecutar su código. La máquina reconoce la solicitud de entrada al modo cuando el pin BKGD/MS es llevado a alto con el flanco de bajada de la señal interna de RESET. Este modo presenta las siguientes variaciones:
  - **Modo RUN normal:** Una vez la CPU ha salido del estado de RESET, carga el registro SR (*Status Register*)<sup>12</sup> y el registro PC (*Program Counter*)<sup>13</sup> con el contenido de las direcciones 0x(00)00\_0000 y 0x(00)00\_0004 de la memoria y ejecuta la primera instrucción apuntada por el PC. Es importante saber que la arquitectura ColdFire® V1 usa direccionamientos por *byte* en el modo *big endian*<sup>14</sup>.
  - **Modo RUN en bajo consumo (LPRUN: low Power RUN):** En este modo el regulador interno se lleva al estado de *standby* y de esta manera ubicar la CPU en

---

<sup>11</sup> Programa residente en memoria FLASH, que se ejecuta antes de la aplicación de usuario. Contiene funciones de reconocimiento del hardware del MCU y funciones de depuración para el modo de supervisión.

<sup>12</sup> El registro SR se estudiará con detalle en el Modelo de programación de la máquina.

<sup>13</sup> El registro PC se estudiará con detalle en el Modelo de programación de la máquina.

<sup>14</sup> El *byte* de mayor peso del dato es almacenado en la dirección de menor peso y así sucesivamente con los pesos siguientes.

modo de bajo consumo. Es importante saber que el sistema queda alimentado de manera no regulada y que todos los periféricos no usados son privados de la señal de reloj, vía los registros SCGC1 y SCGC2<sup>15</sup>. También es importante saber que la CPU no puede entrar en modo LPRUN cuando el sistema se encuentra en uso del BDM (*Background Debug Module*).

**Tabla 5.1. Configuración de los modos de la CPU.**

Modos de Operación	SOPT1 SIM		CSR2 BDC	SPMSC1 PMC		SPMSC2 PMC		Reloj de la CPU y los periféricos	Efectos sobre otros sistemas		
	STOP	WAIT		ENBDM	LVD	LVDSE	LPR		Reloj BDC	AI energizar	
Modo RUN: El procesador y los periféricos trabajando a velocidad normal	x	x	x	x	x	0	x	El módulo ICS opera en cualquier modo	Encendido	Encendido	
			x	1	1	x	x		NOTA: Cuando no es necesario, El reloj del BDC puede ser desenganchado a criterio de la		
			1	x	x	x	x				
Modo LPRUN con LVD deshabilitado: El procesador y los periféricos trabajando a baja frecuencia. 250 kHz para la CPU y 125 kHz para periféricos	x	x	0	0	x	1	0	Se requiere baja frecuencia. Módulo ICS está en modo FBELP	El reloj está disponible con algunos pocos ciclos demandados por la CPU, normalmente cuando se detecta flanco abajo en el pin de BKGD.	Loose Reg	
			0	1	0						
Modo WAIT: El reloj de la CPU está inactivo, pero los periféricos tienen reloj.	x	1	x	x	x	0	x	Reloj periféricos encendido y reloj CPU encendido si ENBDM = 1	Encendido		
			x	1	1	x	x		Reloj periféricos encendido y reloj en el pin de BKGD.		
			1	x	x	x	x				
Modo LPWAIT: El reloj del procesador está inactivo, los periféricos tienen reloj en baja potencia y el regulador interno no está en regulación. El LVD permanece inactivo.	x	1	0	0	x	1	0	Reloj CPU apagado y reloj periféricos en baja velocidad. Módulo ICS en modo FBLEP	El reloj del BDC está habilitado si ENBDM = 1, previo haber entrado en STOP.	Loose Reg	
				1	0						
Modos de STOP deshabilitados: Se ejecuta un código ilegal, si la instrucción STOP está habilitada y el bit CPUCR[RD] es aclarado, de otra forma se genera una excepción por código ilegal.	0	0	BKGD / MS en RESET	=1	=1	=0	=0	Encendido	BKGD / MS en RESET	Encendido	
STOP4: Bien sea que los modos de bajo consumo no han sido requeridos, o el LVD ha sido habilitado o el bit ENBDM = 1.	1	0	x	x	x	0	0	Reloj de los periféricos apagado. Reloj de la CPU encendido si ENBDM = 1	Encendido		
			x	1	1	1	0				
			x	1	1	0	1				
			1	x	x	x	x	Reloj de la CPU encendido. Reloj Periféricos apagado.			
STOP3: El LVD no ha sido habilitado en modo STOP. Los relojes deberán ser puestos en baja frecuencia y el regulador interno opera en modo de pérdida de regulación.	1	0	0	1	0	1	0	Se requiere baja frecuencia. El ICS en modo FBLEP. Reloj de CPU y periféricos apagado.	Apagado	Pérdida de regulación	
				0	x						
STOP2: El módulo LVD no está activo. El módulo BDC está habilitado y el modo STOP4 es invocado en vez del modo STOP2.	1	0		1	0	0	1	N/A	N/A	Apagado	
				0	x						

Antes de ingresar al modo LPRUN, las siguientes acciones deberán ser ejecutadas:

- El FLL (*Frequency Loop Locked*) del módulo ICS (*Internal Clock Source*) es llevado a un estado de *bypass*, para la adopción de un modo de operación de bajo consumo llamado FBELP (*FLL Bypassed External Low Power*)<sup>16</sup>
- El bit HGO del registro ICSC2 es aclarado, para configurar un oscilador externo de bajo consumo.
- La frecuencia del bus es menor que 125 kHz.
- El módulo conversor analógico a digital (ADC) deberá trabajar en baja potencia o ser deshabilitado.

<sup>15</sup> Los registros SCGC1 y SCGC2 serán vistos con detalle en el Capítulo 13.

<sup>16</sup> En el Capítulo 13 se ampliará el concepto de FBELP.

- El módulo de detección de bajo voltaje deberá ser deshabilitado, debido a la condición de *standby* del regulador interno.
- No se tiene disponibilidad sobre la programación o borrado de la FLASH.

Finalmente, se puede ingresar al modo, llevando un “1” al bit LPR del registro SPMSC2 (*System Power Management Status and Control 2 Register*).

Para regresar al modo normal de RUN, es necesario aclarar el bit LPR. El bit LPRS indicará si el regulador está en modo normal de funcionamiento y la máquina podrá correr a la máxima velocidad configurada. Si una interrupción se presenta, la máquina podrá salir del estado de LPRUN, esto se puede lograr poniendo en “1” el bit LPWUI del registro SPMSC2 y dentro de la rutina de atención a la interrupción se podrá habilitar la operación del ICS (*Internal Clock Source*) a máxima velocidad.

- **Modos de WAIT:** Para entrar en este modo de bajo consumo es necesario ejecutar la instrucción STOP, después de configurar la máquina como se ilustra en la Tabla 5.1.
  - **Modo normal de WAIT:** La CPU queda en modo STOP y el consumo se reduce significativamente, dado a que el reloj es interrumpido. La arquitectura ColdFire® V1 no hace diferencia entre el modo STOP y el modo WAIT, ambos son catalogados como modos de STOP, desde la perspectiva del núcleo. La diferencia entre ambos modos sólo se aprecia desde el suministro del reloj a los periféricos del sistema. En modo STOP, la mayoría de los periféricos son desalimentados de reloj, mientras que en modo WAIT el reloj alimenta la mayoría de los módulos.

Si es necesario que el sistema responda a comandos en el modo BDM, será prioritario poner a “1” el bit ENBDM.

Al presentarse un evento de interrupción, estando la máquina en modo WAIT, la CPU ejecuta un proceso de excepción, comenzando con un servicio de apilamiento de información valiosa y luego conduciendo la máquina a un servicio de atención a la interrupción.

- **Modo LPWAIT:** La diferencia respecto al modo normal de WAIT es que el regulador de la CPU sale de regulación y queda en estado de *standby*. Lo anterior reduce enormemente el consumo de la máquina, consumo que puede ser reducido aún más deshabilitando los módulos que no se utilicen. Esta última operación se puede lograr poniendo a cero los bits de los módulos a inactivar en el registro SCGC.

Las restricciones vistas en el modo LPRUN se aplican al modo LPWAIT.

Si el bit LPWUI es puesto a “1”, cuando la máquina ha ejecutado la instrucción STOP, el regulador regresa a su estado de regulación y el ICS puede ser llevado a

su máxima velocidad en la entrada a la rutina de atención a la interrupción, que determinó la salida del estado de WAIT.

Si el bit LPWUI es puesto a “0”, cuando la máquina ha ejecutado la instrucción STOP, la CPU regresa al modo LPRUN.

- **Modos de STOP:** Existen tres modos de operación en STOP, siempre y cuando el bit STOPE del registro SOPT1 se encuentre en “1”. El bit WAITE del registro SOPT1 deberá ser aclarado, excepto cuando se desee trabajar en modo WAIT. En el modo STOP3 las fuentes de reloj de la CPU son interrumpidas.

**NOTA:** Si la CPU no está habilitada para el modo WAIT ni para el modo STOP y se presenta una ejecución de instrucción STOP, la CPU inicia un evento de RESET por ILLEGAL OPCODE si el bit IRD del registro CPUCR se encuentra en estado “0” o un proceso de excepción por ILLEGAL OPCODE si el bit IRD del registro CPUCR se encuentra en estado “1”.

Los diferentes modos de STOP son seleccionados mediante el bit PPDC del registro SPMSC2.

La mayoría de los comandos del modo background (BDM) no son reconocidos en los modos de STOP, pero el comando BACKGROUND puede sacar la CPU del modo STOP4 y entrar en modo HALT. Quedando la CPU en el modo HALT y estando el bit ENBDM en “1”, todos los comandos del BDM se podrán utilizar.

- **Modo STOP2:** La tabla 5.1 detalla la forma de ingresar al modo STOP2, en donde la mayoría de los módulos de la CPU son apagados, con excepción de la memoria RAM y el módulo RTC (*Real Time Clock*) de manera opcional. Al entrar a este modo, la CPU almacena el estado de los pines I/O en la RAM, con el propósito de recuperar su estado una vez se decida salir del modo.

Para salir de este modo, es necesario introducir un flanco de bajada<sup>17</sup> en el pin de RESET del sistema o generar un evento de excepción por interrupción en el módulo RTC, siempre y cuando esté habilitado. También, al salir del modo STOP2, la máquina ejecuta un estado de POR (Power On Reset) conformado por los siguientes eventos:

- Todos los módulos de control y los registros de estado son inicializados por el controlador de manejo de la potencia, por el RTC y por el *buffer* de trazo del depurador (*Debug Trace Buffer*). Más adelante, en el texto, serán tratados otros registros afectados por este evento.

---

<sup>17</sup> Tener la precaución de conectar una resistencia de *pullup* en este pin y una forma sería habilitar un *pullup* en el bit PTAPE5 del registro PTAPE.

- La función de RESET por LVD será habilitada y la CPU se quedará en estado de RESET si el voltaje de la fuente VDD queda por debajo del voltaje de comparación (*LVD trip point*).
- La CPU comienza un proceso de excepción por RESET, realizando la captura de los vectores en las direcciones 0x(00)00\_0000 y 0x(00)00\_0004.

Adicionalmente la bandera PPDF del registro SPMSC2 es puesta a “1”. Esta bandera es servida para que el usuario pueda ejecutar una rutina de recuperación por salida del modo STOP2.

Para mantener el estado de los pines de I/O antes de entrar al modo STOP2, es necesario recuperar su estado desde la memoria RAM hacia los registros de los puertos. Esta acción exige que se escriba un “1”, antes de la recuperación, sobre el bit PPDACK del registro SPMSC2. En caso de no escribirse un “1” sobre el bit PPDACK, el estado de los pines I/O será asumido como el indicado para un RESET normal o por defecto.

Para aquellos pines que están trabajando como servicio a los módulos, es necesario reconfigurar el periférico antes de escribir en el bit PPDACK.

Si en el modo STOP2 se tiene la opción de oscilador para bajo rango (bit RANGE = 0 del registro ICSC2), como reloj para el RTC, es necesario reconfigurar el registro ICSC2 antes de escribir en el bit PPDACK. Para deshabilitar el reloj en el modo STOP2, es necesario conmutarse al modo FBI o FEI<sup>18</sup> del módulo SCI, antes de ejecutar la instrucción STOP.

- **Modo STOP3:** La tabla 5.1 detalla la forma de ingresar al modo STOP3, en donde el estado de todos los registros internos, el contenido de la memoria RAM y el estado de los pines I/O, se mantienen. El regulador interno entra a operar en modo de *standby*.

Para salir de este modo, es necesario introducir un flanco de bajada<sup>19</sup> en el pin de RESET del sistema o generarse un evento de excepción por interrupción de los siguientes módulos: RTC, ADC, ACMP, IRQ, SCI o KBI. Si se sale del modo STOP3 vía evento de RESET, el MCU es reinicializado y las operaciones son resumidas después de cargarse el vector de RESET. Si se sale del modo STOP3 vía evento de excepción por fuente de interrupción, el MCU cargará el vector adecuado, dependiendo del módulo que generó la interrupción.

- **Modo STOP4:** A diferencia de los modos STOP2 y STOP3, en este modo el regulador trabaja a plena regulación. Este modo es también llamado modo HALT y está relacionado directamente con la entrada a modo BDM desde STOP o por un evento de LVD desde STOP.

---

<sup>18</sup> Como se verá en le Capítulo 11.

<sup>19</sup> Es necesario asegurar una resistencia de *pullup* en este pin y una forma es habilitar un *pullup* en el bit PTAP5 del registro PTAP5.

Si el bit ENBDM está en “1” cuando la CPU ejecuta la instrucción STOP, el sistema suministra reloj a la lógica de control del modo *background* (BDM) de modo que ésta permanece activa durante el modo STOP. Si el usuario intenta entrar a STOP2 o STOP3 cuando el bit ENBDM está en “1”, el sistema queda en STOP4 (ver Tabla 5.1 para detalles).

El ingreso a este modo también se obtiene si los bits LVDE o LVDSE del registro SPMSC1 están en “1” y se presenta un evento por de bajo voltaje con el módulo LVD, previamente habilitado. Es importante anotar que el LVD trabajará correctamente si el regulador interno se encuentra operando a plena regulación, lo cual descarta los modos STOP2 y STOP3 para esta condición. El LVD puede generar un evento de excepción de RESET o de interrupción.

Para salir del modo STOP4 es necesario que se produzca un evento de RESET o alguna de las siguientes excepciones de interrupción: RTC, LVD, LVW, ADC, ACMP, IRQ, SCI o KBI.

### Modelo de programación

Heredando de la máquina M68000, la arquitectura ColdFire® V1 no especifica registros acumuladores ni registros punteros (índices o bases), para la manipulación de datos y decodificación de las instrucciones en memoria. Esta arquitectura contiene un juego de registros generales, que cumplen con funciones de manipulación de datos y direccionamiento de memoria llamados los Dn y An, respectivamente.

El modelo de programación depende del concepto de **nivel de privilegio**, que el programador elige de la máquina. Existen dos niveles de privilegio llamados **nivel de usuario** y **nivel de supervisor**, que a continuación se describen.

- **Registros para nivel de supervisor únicamente:** Están restringidos para el *software* de control de programa, en donde son implementadas funciones restringidas sobre la operación del sistema, funciones de control sobre los pines I/O y manipulación de la memoria. Los registros relacionados con el nivel de supervisor son:
  - **Registro de Estado (SR: State Register):** En la Figura 5.3 se ilustran los bits que conforman el registro SR, utilizado para almacenar el estado del procesador y que a su vez incluye:
    - El registro CCR (*Code Condition Register*), que no es más que el registro de las banderas principales de la CPU.
    - Los bits de máscara de prioridad para los eventos de excepción por interrupción.
    - Otros bits de control en el nivel de supervisión.

### Registro de Estado (SR)

BDM: Carga : 0xEE  
Almacenamiento : 0xCE

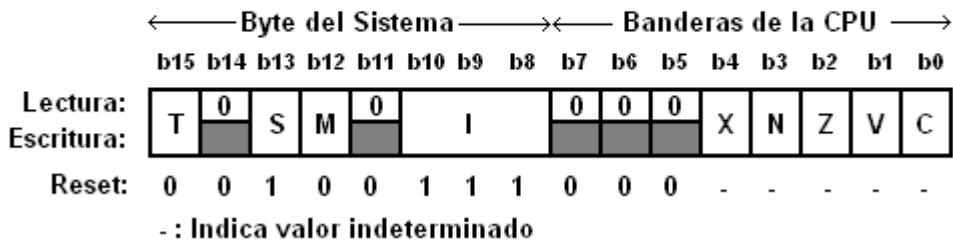


Figura 5.3. Registro de Estado

En donde:

**T (Habilitación de un evento de Trazo (Trace)):** Cuando este bit es puesto en “1”, el modo BDM realiza un evento de excepción del tipo *trace*, después de cada instrucción ejecutada.

**S (Selección del nivel de Supervisor o Usuario):** 0 = Modo Usuario / 1 = Modo Supervisor.

**M (Maestro/Estado de interrupción):** Este bit es borrado por una excepción de interrupción y por programación puede ser puesto a “1” durante la ejecución de una instrucción RTE (*Return of Exception*) o instrucciones de movimiento sobre el registro SR.

**I (Máscara para el Nivel de Interrupción):** Define el nivel actual de las interrupciones. Los requerimientos de interrupción son inhibidos para todos los niveles cuya prioridad sea menor o igual al valor de I. Por defecto el valor de I es 7, fijando el nivel más alto para las interrupciones y que no puede ser inhibido.

- **Registro de Configuración de la CPU (CPUCR: CPU Configuration Register):** Este registro suministra al nivel de supervisor la opción de configuración de funciones del núcleo. Ciertas características del hardware pueden ser habilitadas o inhibidas de forma individual, según sea el valor de los bits del CPUCR. En la Figura 5.4 se ilustran los bits que conforman el registro CPUCR.

En donde:

- **ARD (Address Related reset Disable):** Usado para deshabilitar la generación de un evento de RESET, como respuesta un proceso de excepción causado por un error de direccionamiento, un error de acceso a bus, un error de formato en la instrucción RTE o una condición de *fault on fault half*.

- 0:** Se genera RESET por ese tipo de excepciones.
- 1:** No se genera RESET en respuesta a esas excepciones.

#### Registro de Configuración de la CPU (CPUCR)

BDM: Carga : 0xE2  
Almacenamiento : 0xC2



Figura 5.4. Registro de Configuración de la CPU

- **IRD (Instruction Related reset Disable):** Usado para deshabilitar la generación de un evento de RESET, como respuesta a un proceso de excepción, causado por un error al intentar ejecutar una instrucción no válida o ilegal (*illegal opcode*) o como una violación de privilegio<sup>20</sup>.
  - 0:** Se genera RESET por ese tipo de excepciones.
  - 1:** No se genera RESET en respuesta a esas excepciones.
- **IAE (Interrupt Acknolegde Enable):** Fuerza al procesador a generar una señal de IACK (*Interrupt Acknolegde*) desde el controlador de interrupciones, como respuesta a un proceso de excepción en el intento de recuperar el número del vector a un requerimiento de interrupción. El tiempo de ejecución del procesador para una excepción por interrupción es poco mejorado cuando este bit es aclarado.
  - 0:** El procesador usa el número del vector, proveído por el controlador de interrupciones, en el tiempo en el que el requerimiento fue señalado.
  - 1:** Se produce un ciclo de IACK desde el controlador de interrupciones.
- **IME (Interrupt Mask Enable):** Fuerza al procesador a fijar en valor de la máscara de interrupciones (I) a 7. Adicionalmente, permite la portabilidad de interrupciones de las máquinas HCS08 a la arquitectura ColdFire®.
  - 0:** Como parte de una excepción por interrupción, el procesador asigna a I el valor del nivel de la interrupción que está siendo atendida.
  - 1:** Como parte de una excepción por interrupción, el procesador fuerza a 7 la máscara I. La máquina deshabilita los niveles desde el 1 al 6 y permite reconocimiento de interrupciones por flanco de nivel 7.
- **BWD (Buffered peripheral bus Write Disable):**

---

<sup>20</sup> Intentar ejecutar una instrucción asignada sólo al nivel de supervisión desde el nivel de usuario, genera un proceso de excepción de violación de privilegio (*privilege violation*).

**0:** Las escrituras sobre el bus de periféricos son almacenadas y el ciclo del bus es terminado inmediatamente y no hay espera para el estado de finalización del bus de periféricos.

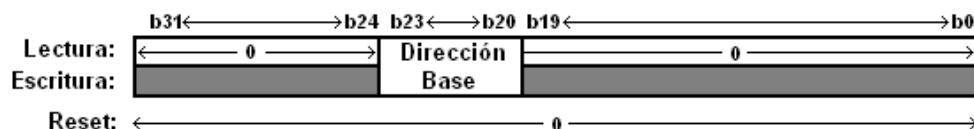
**1:** Deshabilita el almacenamiento de escrituras sobre el bus de periféricos y no finaliza el ciclo del bus hasta que una versión registrada del final del acceso al bus de periféricos haya sido recibida.

- **FSD (Flash Speculation Disable):** Deshabilita cierta capacidad de especulación<sup>21</sup> sobre el direccionamiento en el controlador de memoria FLASH.
  - 0:** El controlador de la FLASH trata de especular en los accesos de lectura, para mejorar el desempeño del procesador. Se trata de minimizar los tiempos de acceso de la FLASH.
  - 1:** Deshabilita la especulación en la FLASH.
- **Registro Base a Vector (VBR: Vector Base Register):** Este registro contiene la dirección base a los vectores de excepción localizados en la memoria. Para acceder a la tabla de vectores, el desplazamiento de un vector es adicionado al VBR. Para los procesadores ColdFire®, los 20 bits de menor peso del VBR no son utilizados. Esto fuerza a que la tabla de vectores esté alineada con 16 bloques de 1MB<sup>22</sup> (ver Figura 5.5).

Los 8 bits superiores del registro VBR son forzados a cero debido a que el núcleo del ColdFire® direcciona en 24 bits, es decir hasta 16 MB. El VBR puede ser utilizado para relocalizar la tabla de vectores de excepción, desde su dirección inicial dentro de la FLASH (0x(00)00\_0000) hacia la base de la RAM (0x(00)80\_0000), si fuera necesario.

#### Registro Base a Vectores (VBR)

BDM: Carga : 0xE1  
Almacenamiento : 0xC1



**Figura 5.5. Registro Base a Vectores**

- **Registros para nivel de supervisor y usuario:** Se acceden desde el nivel de supervisor o usuario, sin ninguna restricción. Estos registros son:
  - **Registros de Dato (D0 – D7):** Estos registros son para hacer operaciones sobre datos en formatos de 8, 16 o 32 bits. También, estos registros pueden ser usados como registro índice.

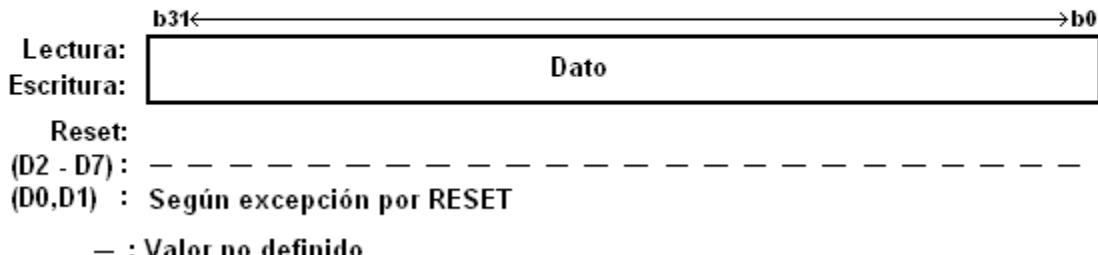
<sup>21</sup> Relacionado con técnicas de ejecución especulativa, que agilizan los tiempos de acceso del procesador.

<sup>22</sup> Este mecanismo permite movilidad entre familias ColdFire® y con la familia HCS08.

Ante un evento de excepción por RESET, los registros D0 y D1 se precargan con un valor de configuración del hardware implementado en el núcleo ColdFire® V1, como se verá más adelante. La Figura 5.6 ilustra los registros Dn.

#### Registros de Dato (D0-D7)

**BDM:** Carga :  $0x60 + n; n = 0\text{-}7$  (Dn)  
 Almacenamiento :  $0x40 + n; n = 0\text{-}7$  (Dn)

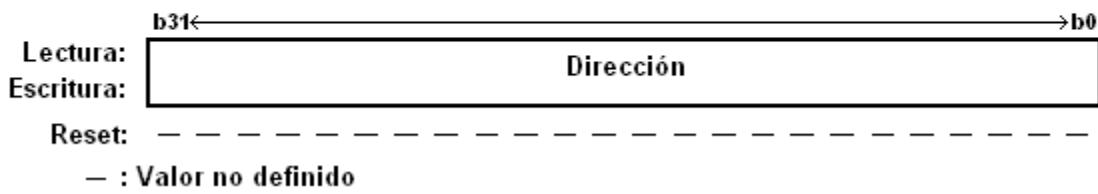


**Figura 5.6. Registros de Dato**

- **Registros de Dirección (A0 – A6):** Estos registros son usados como punteros a pila por *software*, registro índice o registros base a direcciones. También, estos registros pueden ser usados como registros tipo dato para operaciones de 16 o 32 bits. La Figura 5.7 ilustra los registros An.

#### Registros de Dirección (A0 - A6)

**BDM:** Carga :  $0x68 + n; n = 0\text{-}6$  (An)  
 Almacenamiento :  $0x48 + n; n = 0\text{-}6$  (An)



**Figura 5.7. Registros de Dirección**

- **Puntero a Pila del Usuario y el Supervisor (A7 y OTHER A7):** El MCU ColdFire® V1 soporta dos punteros a pila de manera independiente. El puntero a pila para el nivel de supervisor es conocido como SSP (*Supervisor Stack Pointer*). Para el nivel de usuario el puntero a pila es USP (*User Stack Pointer*) (ver Figura 5.8).

En realidad toman el mismo nombre, pero una función ejecutada en la inicialización del núcleo determina cual se usa en cada momento. La función es tan simple como:

```

if SR(S) = 1

    then A7 = SSP
        OTHER A7 = USP

    else   A7 = USP
        OTHER A7 = SSP

```

El modelo de programación del BDM soporta lecturas y escrituras directas sobre el SSP y el USP, siendo responsabilidad del sistema de desarrollo externo, basado en el estado del bit S del registro SR, hacer la elección apropiada.

Existen dos instrucciones en código assembler, que soportan el movimiento de datos sobre el registro A7:

MOVE.L	Ay , USP	;Mover dato hacia el USP
MOVE.L	USP , Ax	;Mover dato desde el USP

**NOTAS:** - Estas instrucciones serán detalladas en el Capítulo 6, aparte 6.2.  
 - El USP deberá ser inicializado antes de entrar al modo usuario.  
 - El SSP es cargado durante el proceso de excepción de RESET con el contenido de la localización 0x(00)00\_0000.

#### Registro Puntero a Pila (A7 y OTHER A7)

BDM: Carga : 0x6F (A7)  
 Almacenamiento : 0x4F (A7)  
 Carga : 0xE0 (OTHER A7)  
 Almacenamiento : 0xC0 (OTHER A7)



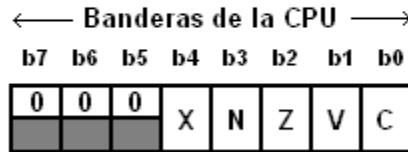
Figura 5.8. Registro Puntero a Pila

- **Registro de Código de Condiciones (CCR: Condition Code Register):** Configura el byte de menor peso del registro SR e informa sobre el resultado de operaciones aritméticas y lógicas de la CPU (ver Figura 5.9).

### Registro de Código de Condiciones (CCR)

LSB del registro SR

BDM: Carga : 0xEE (SR)  
Almacenamiento : 0xCE (SR)



Reset: 0 0 0 - - - -

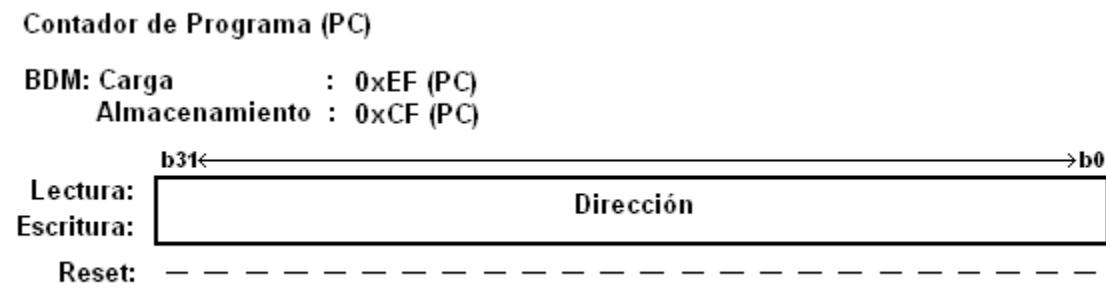
- : Indica valor indeterminado

**Figura 5.9. Registro de Códigos de Condición**

En donde:

- **X (eXtend condition code bit):** Esta bandera se usa en operaciones aritméticas de precisión múltiple y actúa como un bit para indicar desbordamiento (*carry/borrow*) en ese tipo de operaciones  
**0:** No se presentó un desbordamiento en la operación de múltiple precisión.  
**1:** Se presentó un desbordamiento en la operación de múltiple precisión.
  - **N (Negative condition code bit):** Esta bandera es usada para indicar resultados negativos en operaciones aritméticas.  
**0:** El bit de mayor peso del resultado es “0” (resultado positivo).  
**1:** El bit de mayor peso del resultado es “1” (resultado negativo).
  - **Z (Zero condition code bit):** Esta bandera es usada para indicar resultado cero en operaciones aritméticas.  
**0:** El resultado no dio cero.  
**1:** El resultado dio cero.
  - **V (oVerflow condition code bit):** Esta bandera es usada para indicar un sobreflujo en el resultado de una operación aritmética. Indica que el resultado no puede ser contenido en el tamaño del operando destino.  
**0:** No ha ocurrido un sobreflujo en el resultado.  
**1:** Ha ocurrido un sobreflujo en el resultado.
- **Contador de Programa (PC: Program Counter):** Este registro contiene la dirección de la siguiente instrucción a ser ejecutada (ver Figura 5.10). Durante la ejecución de una instrucción, el procesador incrementa automáticamente la dirección del PC (el incremento no necesariamente es 1, depende de la posición de la siguiente instrucción) o localiza un nuevo valor de dirección en el PC, si se trata de un salto, llamado a subrutina o atención a un proceso de excepción.

El PC será cargado durante el proceso de excepción por RESET con el contenido de la dirección 0x(00)00\_0004 (recordar que la carga es en modo *big endian*).



**Figura 5.10. Contador de Programa**

## 5.2. ORGANIZACIÓN DE LA MEMORIA

La memoria en el procesador ColdFire® V1 está respaldada por un controlador diseñado para hacer accesos rápidos y eficientes de los elementos esclavos (módulos de la plataforma. Pej: RGPIO) y de la memoria en general.

Cada módulo esclavo contiene un bus local, para establecer el intercambio de información con la plataforma del bus local (ver Figura 5.2) del controlador de memoria de una manera rápida y eficiente.

La plataforma del núcleo contiene un número de controladores locales, que actúan como puentes hacia el controlador de bus local (maestro). La interfase con los módulos (periféricos externos a la plataforma. Pej: ADC), localizados por fuera de la plataforma, se usa típicamente para los registros de programación de éstos.

En el bus de la plataforma se establece un protocolo de *pipeline* (segmentación) de dos etapas, que se detallan así:

- **Etapa 1:** El bus maestro establece la dirección y los atributos de la decodificación de la operación en curso. Esta etapa es conocida con el nombre de fase de direccionamiento (*address phase*).
- **Etapa 2:** El bus del esclavo responde con el dato y señales de finalización. A esta etapa se le conoce con el nombre de fase del dato (*data phase*).

La memoria en las máquinas ColdFire® V1 está organizada dentro de un marco de direccionamiento de 24 bits, esto provee de una capacidad de direccionamiento de máximo 16 MB. La Tabla 5.2 muestra la organización de la memoria.

**Tabla 5.2. Organización de la memoria**

Rango de Dirección	Esclavo Destino	Tamaño
0x00_0000 – 0x7F_FFFF	Plataforma FLASH	8 Mbytes
0x80_0000 – 0xBF_FFFF	Plataforma RAM	4 Mbytes
0xC0_0000 – 0xDF_FFFF	Módulos de la Plataforma	2 Mbytes
0xE0_0000 – 0xFF_7FFF	Reservado	2 Mbytes – 32 Kbytes
0xFF_8000 – 0xFF_FFFF	Módulos por fuera de Plataforma	32 Kbytes

La ubicación de los diferentes bloques se hace coincidir con múltiplos de 1 MB en un espacio total de 16 MB, para la familia ColdFire® V1.

Dependiendo del tipo de región de memoria, el tipo de dato que se soporta puede variar. La Tabla 5.3 relaciona el tipo de dato respecto a la región de memoria a acceder.

Cualquier acceso a regiones de memoria no comprendidas en los rangos descritos producen un error del tipo Direccionamiento Ilegal (*Illegal Address*), que podría generar en el sistema un evento de RESET por excepción.

**Tabla 5.3. Tipo de dato según región en la memoria**

Dirección Base	Región	Lectura			Escritura		
		Byte	Word	Long	Byte	Word	Long
0x00_0000	Flash	X	X	X	—	—	X
0x80_0000	RAM	X	X	X	X	X	X
0xC0_0000	GPIO	X	X	X	X	X	X
0xFF_8000	Periféricos	X	X	—	X	X	—

X: Dato soportado en la región

- : Dato no soportado en la región

El uso de ciertos modos de direccionamientos, sobre operandos, deben ser tenidos en cuenta dependiendo del tipo de memoria que se acceda. Sobre este tema se hablará en el Capítulo 6 aparte 6.1, destacándose la ventaja de implementar un código eficiente, de baja densidad y alto desempeño.

### 5.3. PROCESAMIENTO DE EXCEPCIONES

Una excepción es un evento generalmente de carácter asíncrono<sup>23</sup>, es decir, la máquina no prevé que éste va a suceder. Dentro de los procesos de excepción de la arquitectura ColdFire® V1, están los eventos que generan RESET (eventos de fallo), la inicialización de algunos registros importantes, eventos tipo *trap*<sup>24</sup> y las interrupciones.

Los procesos de excepción de la familia ColdFire® fueron concebidos para un óptimo desempeño de la máquina. Los componentes interesantes para la atención de los procesos de excepción, insertados en las máquinas ColdFire® son:

- Una tabla simple, en donde se almacenan los vectores para atender los eventos de excepción.
  - Capacidad de relocalización de la tabla de vectores, mediante el uso de un registro base (VBR).
  - Un formato simple de trama de pila (*stack frame*), para los procesos de excepción.
  - Uso independiente de punteros a pila para los niveles de supervisor y usuario.
- 
- **Definición de una pila para el proceso de excepciones:** Todos los procesadores ColdFire® utilizan un modelo de excepción para la reinicialización del sistema, como acciones de detección de fallo del sistema o traída y ejecución de la primera instrucción del sistema. Cuatro pasos generales definen un proceso de excepción, a saber:
    - **Paso 1:** El procesador hace una copia interna del registro SR y luego entra a nivel de supervisión colocando un “1” en el bit S del registro mencionado. El bit T (*trace*) es forzado a “0”. Estando en este estado, la ocurrencia de una excepción de interrupción fuerza el bit M (Maestro/Interrupción) a “0” y la máscara de interrupciones (I), es forzada al nivel de la interrupción presente.
    - **Paso 2:** El procesador determina el número del vector de la excepción. Para todos los eventos de fallo, el procesador realiza la localización del vector basado en el tipo de excepción. Si se presenta una interrupción, el procesador ejecuta un ciclo de reconocimiento a interrupción (IACK) y obtiene el número del vector desde el controlador de interrupciones. La acción anterior requiere que el bit IAE del registro CPUCR esté en “1”. El ciclo IACK es mapeado a una localización especial junto con el espacio de direcciones del controlador de interrupciones. Si el bit IAE vale “0”, el procesador usará el número de vector suministrado por el

---

<sup>23</sup> Como los eventos de RESET o las interrupciones de cualquier sistema, pero pueden existir eventos de excepción buscados o programados por el usuario.

<sup>24</sup> Un evento de *trap* se refiere a la instrucción del tipo TRAP #n, utilizada para generar eventos de excepción, como el cambio de nivel de supervisor a nivel de usuario. En esta máquina es posible tener 16 eventos diferentes de *trap*, que actúan similar a la instrucción SWI (*Software interruption*) de otras máquinas.

controlador de interrupciones en el mismo instante en el que la interrupción fue reconocida, para optimizar desempeño.

- **Paso 3:** El procesador salva información importante, creando una trama de pila (*stack frame*) de excepción. Esta trama es contenida desde la dirección 0x(00)00\_0000 a la dirección 0x(00)00\_0003 y es atendida por el SSP (*Supervisor Stack Pointer*). Para todos los eventos de excepción, el procesador usa una trama de pila de excepción (*Exception Stack Frame*) de longitud fija (ver Figura 5.11)

#### Trama de Pila de Excepción (Exception Stack Frame)

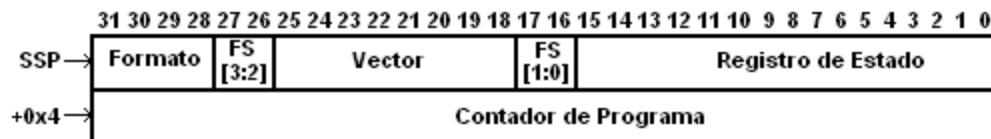


Figura 5.11. Trama de Pila

Los primeros 32 bits de esta pila contienen el formato del vector de excepción y el contenido del registro de estado (SR) y los segundos 32 bits contienen la dirección del contador de programa (PC).

El formato del vector de excepción está conformado por tres campos, que se detallan a continuación:

- **Campo de Formato:** Siempre será escrito, por el procesador, con un valor entre 4, 5, 6 o 7.

Tabla 5.4. Campo Formato del SSP

SSP Original en tiempo de excepción, Bits 1:0	SSP con primera instrucción del handler	Campo del Formato
00	SSP Original - 8	0100
01	SSP Original - 9	0101
10	SSP Original - 10	0110
11	SSP Original - 11	0111

- **Campo de indicación de Fallo (fault):** Esta definido únicamente para acceso y direccionamiento de errores, para otro tipo de excepciones éste campo aparecerá en ceros.

**Tabla 5.5. Campo indicación fallo**

FS[3:0]	Definición
00xx	Reservado
0100	Error en traída de instrucción
0101	Reservado
011-	Reservado
1000	Error en escritura de operando
1001	Intento de escritura en espacio protegido
101-	Reservado
1100	Error en escritura de operando
1101	Reservado
111x	Reservado

- **Campo del número de vector:** Es calculado por el procesador y define el tipo de excepción. Este campo indica el número del vector para todos los modos de fallo del sistema y/o las interrupciones, suministrado por el procesador o por el controlador de interrupciones, respectivamente.

En la Tabla 5.6 aparece la asignación de vectores, según el tipo de excepción. Para la familia ColdFire® V1, las únicas posiciones recomendadas para ubicar la tabla de vectores son la 0x(00)00\_0000 (origen de la FLASH) y la 0x(00)80\_0000 (origen de la RAM). La tabla contiene 256 vectores de excepción distribuidos de la siguiente manera:

- Los primeros 64 son definidos por Freescale.
- Los restantes 192 son definidos como vectores de interrupción de usuario.

Todos los procesadores ColdFire®, inhiben el muestreo de las interrupciones durante la ejecución de la primera instrucción. A diferencia de las demás versiones, la versión 1 de los procesadores ColdFire® incluyen la instrucción STLDSR, utilizada para almacenar el estado de la máscara de interrupciones actual [I], que luego ser recuperada. Esta instrucción deberá ser utilizada como primera instrucción de la rutina de atención a alguna interrupción.

**Tabla 5.6. Asignación de Vectores de Excepción**

Número del Vector	Offset (Hex) del Vector	Apilamiento Contador de Programa	Asignación
0	0x000	—	Puntero a Pila Supervisor inicial
1	0x004	—	Contador de Programa inicial
2	0x008	Fallo	Error de Acceso
3	0x00C	Fallo	Error de Dirección
4	0x010	Fallo	Instrucción ilegal
5	0x014	Fallo	División por cero
6–7	0x018–0x01C	—	Reservada
8	0x020	Fallo	Violación de Privilegio
9	0x024	Próximo	Trazo
10	0x028	Fallo	Línea-a de opcode no implementada
11	0x02C	Fallo	Línea-f de opcode no implementada
12	0x030	Próximo	Interrupción por Depuración
13	0x034	—	Reservada
14	0x038	Fallo	Error de Formato
15–23	0x03C–0x05C	—	Reservada
24	0x060	Próximo	Interrupción Espúrea
25–31	0x064–0x07C	—	Reservada
32–47	0x080–0x0BC	Próximo	Instrucción Trap #(0-15)
48–63	0x0C0–0x0FC	—	Reservada
64–95	0x100–0x17C	Próximo	Reservada para IRQ periféricos
96	0x180	Próximo	SWI de nivel 7
97	0x184	Próximo	SWI de nivel 6
98	0x188	Próximo	SWI de nivel 5
99	0x18C	Próximo	SWI de nivel 4
100	0x190	Próximo	SWI de nivel 3
101	0x194	Próximo	SWI de nivel 2
102	0x198	Próximo	SWI de nivel 1
103–255	0x19C–0x3FC	—	Reservada

- **Comparación entre las máquinas HCS08 y las máquinas ColdFire®, en los procesos de excepción:** Para aquellos usuarios que migren de 8 a 32 bits, utilizando la ruta FLEXIS, la Tabla 5.7 muestra una comparación sobre los procesos de excepción, que puede ser de gran utilidad para comprender los cambios a introducir en la migración de desarrollos.

**Tabla 5.7. Comparación procesos de excepción HCS08 vs ColdFire® V1**

Atributo	HCS08	ColdFire V1
Tabla de Vectores de Excepción	32, 2-b 32, de 2 bytes, localización fija al final de la memoria	103, de un ancho de 4 bytes, localizados en el inicio de la memoria ante RESET y relocalizables vía VBR
Otros Vectores	2 para la CPU + 30 para IRQ de periféricos y RESET al final de la memoria	64 para la CPU + 39 para IRQs con atención a RESET en dirección más baja
Trama de Pila de Excepciones	Trama de 5 bytes: CCR, A, X, PC	Trama de 8 bytes: F/V, SR, PC; registros de propósito general (An,Dn) deben ser salvados/reataurados por la ISR.*
Niveles de Interrupción	Un nivel en el CCR (I)	niveles en el SR(I), con soporte automático de hardware para atender la siguiente
Soporta IRQ no enmascarable	No	Si, con 7 niveles de interrupción
Sensibilidad a forzado de IRQ del núcleo	No	El nivel 7 es sensible al flanco y al nivel
Vectorización del Controlador de interrupciones INTC	Prioridad fija y asignación por vector	Prioridad y asignación de vector fijas, adicionalmente dos IRQs pueden ser remapeadas como la más alta prioridad, para el nivel 6
Software IACK	No	Si
Instrucción para salir de interrupción	RTI	RTE

\* ISR: Rutina de Servicio a la Interrupción

La característica de IACK por software proporciona la habilidad que tiene el controlador de interrupciones, de interrogar por la cercanía al final de una rutina de servicio a un proceso de interrupción. De esta manera la máquina se puede enterar si hay algún requerimiento de interrupción pendiente. La información se encuentra almacenada en un operando tipo byte llamado SWIACK, que pertenece al controlador de interrupciones. Si al leerse este byte se comprueba que no vale cero, la rutina de servicio a interrupciones tomará el valor del registro SWIACK como el número del vector del requerimiento de interrupción con más alta prioridad y pasa el control a un nuevo manipulador (*new handler*) apropiado. En sistemas con alta tasa de actividad de interrupciones, este mecanismo puede proporcionar un mejor desempeño.

La emulación del nivel 1 del proceso de IRQ (*Interrupt Request*) de las máquinas HCS08, puede fácilmente ser manipulado por convención en el software dentro de las rutinas de servicio a las interrupciones de las máquinas ColdFire®. Para este tipo de operación, únicamente son usados dos niveles:

- SR(I) = 0 : Indica que todas las interrupciones son habilitadas.

- SR(I) = 7 : Indica que todas las interrupciones son inhibidas.

Típicamente, sólo el pin de IRQ y la detección de bajo voltaje en la fuente (LVD) son asignadas a los requerimientos del nivel 7. El resto de los requerimientos de interrupción (niveles 1-6) son enmascarados cuando el bit SR[I] = 7. En cualquiera de los casos, todas las máquinas ColdFire® garantizan, que la primera instrucción en un proceso de excepción es ejecutada antes de realizarse el muestreo de interrupciones en el sistema. Lo anterior se puede hacer ejecutando la instrucción:

MOVE.W #2700 , SR ;Hacer SR[I] = 7
------------------------------------

- **Algunos procesos de excepción:** A continuación se describen de una manera resumida, algunos procesos de excepción de las máquinas ColdFire®.
  - **Excepción por Error de Acceso:** Por defecto, las máquinas V1 de ColdFire® generan RESET ante un evento de *illegal address* si un evento de error de acceso (conocido como error de bus) es detectado. Si el bit CPUCR[ARD] = 1, el evento de RESET es deshabilitado y un evento de excepción será generado.
  - **Excepción por Error de Direcccionamiento:** Por defecto, las máquinas V1 de ColdFire® generan RESET si un evento de *illegal address* es detectado. Si el bit CPUCR[ARD] = 1, el evento de RESET es deshabilitado y un proceso de excepción será generado. Cualquier intento de transferencia de control hacia una dirección impar, da como resultado un evento de excepción por error de direcccionamiento. Cualquier intento de uso de un registro índice con formato de *word* (Xn.w) o el uso de un factor de escalamiento de magnitud 8, en un direcccionamiento del tipo indexado efectivo, generaría un error de direcccionamiento. Si un error de direcccionamiento ocurre, cuando se está ejecutando una instrucción RTS (*Return of Subroutine*), la arquitectura V1 sobrescribe en el registro PC (en modo de fallo) con la dirección almacenada en la trama de la pila ante fallos.
  - **Excepción por Error de Instrucción Ilegal:** En modo normal de operación, la máquina V1 genera un RESET por la ejecución de código ilegal, debido a la detección de una instrucción ilegal. Si el bit CPUCR[ARD] = 1, el evento de RESET por ésta vía es deshabilitado y un evento de excepción comenzaría a generarse. Las máquinas ColdFire® soportan instrucciones en tres tamaños: 16, 32 y 48 bits. La primera palabra de instrucción es conocida como la palabra de operación (*opword*), de acuerdo al formato (Formato ISA) mostrado en al Figura 5.12.

### Palabra de operación (opword)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Línea				Modo Operación				Dirección Efectiva Modo Registro							

Figura 5.12. Formato de la palabra de operación

La palabra de operación se encuentra dividida en: Campo de Línea, el cual define 16 líneas de instrucción; Modo de Operación y la Dirección Efectiva. Adicionalmente, la palabra de operación tiene dos extensiones que se detallarán en el aparte 6.1.

- **Excepción por División por cero:** La división por cero, sobre números enteros, causa un evento de excepción.
- **Excepción por Violación de Privilegio:** En modo normal de operación la máquina V1 genera un RESET por la ejecución de código ilegal, debido a la detección de violación de privilegio. Si el bit CPUCR[ARD] = 1, el evento de RESET por ésta vía es deshabilitado y un evento de excepción comenzaría a generarse. La violación de privilegio ocurre cuando una instrucción dedicada al modo supervisor, es ejecutada en modo de usuario.
- **Excepción por Trazo (*Trace*):** Para la ayuda en el desarrollo de programas, todos los procesadores ColdFire® vienen dotados de función de trazabilidad paso a paso para la ejecución del programa (función de trazo o *trace*). Mientras la máquina se encuentra en modo de trazo (mediante el bit SR[T] = 1), toda finalización de una instrucción hace que se genere un evento de excepción por trazo. Esta funcionalidad provee al microcontrolador de monitorear la ejecución del programa en modo de depuración (BDM).
- **Excepción por Código no Implementado tipo Línea-A:** La operación por defecto de un procesador V1 es la de generar un evento de RESET por código ilegal si un código no implementado tipo Línea-A es detectado. Si el bit CPUCR[ARD] = 1, el evento de RESET por ésta vía es deshabilitado y un evento de excepción comenzaría a generarse. Un código no implementado tipo Línea-A se define cuando los bits del 12 al 15 del campo línea del formato de la palabra de operación vale 0b1010, generándose un evento de excepción al presentarse la ejecución de un código no implementado tipo Línea-A.
- **Excepción por Código no Implementado tipo Línea-F:** La operación por defecto, de un procesador V1, es la de generar un evento de RESET por código ilegal si un código no implementado tipo Línea-F es detectado. Si el bit CPUCR[ARD] = 1, el evento de RESET por ésta vía es deshabilitado y un evento de excepción comenzaría a generarse.

Un código no implementado tipo Línea-F se define cuando los bits del 12 al 15 del campo línea del formato de la palabra de operación vale 0b1111, generándose un evento de excepción al presentarse la ejecución de un código no implementado tipo Línea-F.

- **Excepción por Interrupción del Debug:** Generado como respuesta a un disparo por evento de punto de ruptura (*breakpoint register trigger*). El procesador no genera un ciclo de IACK, pero en cambio calcula el número de vector internamente (Vector 12). Adicionalmente los bits SR[M,I] no son afectados por este evento.
- **Excepción por RTE y Error de Formato:** La operación por defecto de un procesador V1 es la de generar un evento de RESET por direccionamiento ilegal si se detecta un error por formato de la instrucción RTE. Si el bit CPUCR[ARD] = 1, el evento de RESET por ésta vía es deshabilitado y un evento de excepción comenzaría a generarse. Cuando una instrucción de RTE es ejecutada, el procesador examina los primeros 4 bits del campo de Formato, para validar el tipo de trama. Para el núcleo ColdFire® cualquier intento de ejecución de una instrucción RTE, donde el formato no coincide con los valores 4, 5, 6 y 7 , genera un error de formato.
- **Excepción por ejecución de la Instrucción TRAP:** La instrucción TRAP #n siempre fuerza la máquina a un evento de excepción como parte de su ejecución y son muy populares para implementar llamadas al sistema. Las instrucciones de TRAP pueden ser usadas para cambiar del modo usuario al modo supervisor. Son en esencia 16 instrucciones que proveen una funcionalidad similar, aunque con expansión adicional, a las SWI (*Software Interrupt*) de las máquinas HCS08.
- **Excepción por Interrupción:** El proceso de excepción por interrupción incluye el reconocimiento de la interrupción y el cálculo del vector de atención a la interrupción. Este vector es recuperado desde el controlador de interrupciones, como quiera que se use un ciclo de IACK o usando el número del vector previamente suministrado. Es importante tener control sobre el bit CPUCR[IAE]. En un evento de excepción por interrupción el bit SR[M] es aclarado y el bit SR[I] es puesto al valor del nivel de la interrupción, que está comenzando a ejecutarse. En el capítulo 12, se estudia con mayor detalle el controlador de interrupciones para la máquina ColdFire® V1.
- **Excepción por paro debido a Falla sobre Falla (*Fault on Fault Halt*):** La operación por defecto de un procesador V1 es la de generar un evento de RESET por direccionamiento ilegal si se detecta una condición de *fault on fault halt*. Si el bit CPUCR[ARD] = 1, el evento de RESET por ésta vía es deshabilitado y un evento de excepción por *fault on fault halt* comenzaría a generarse.

Si el procesador encuentra algún tipo de fallo (*fault*) durante un proceso de excepción por otro fallo (*fault*), la máquina inmediatamente pone en estado de *halt* la ejecución con la condición catastrófica de *fault on fault*. Para salir de este estado es necesario que se genere un evento de RESET.

- **Excepción por RESET:** Son múltiples los eventos que pueden generar una señal de RESET al procesador, como por ejemplo: introduciendo una señal de RESET vía el pin para este propósito, configurando el módulo BDM para forzar a un RESET, detectando algún direccionamiento ilegal o condición de código ilegal. La excepción por RESET tiene la más alta prioridad de los eventos de excepción, esto provee al sistema de la inicialización y recuperación desde una falla catastrófica. El estado de RESET obliga a la CPU a abortar cualquier proceso en progreso y éste no podrá ser recuperado. La excepción por RESET ubica al procesador en el nivel de supervisor poniendo a “1” el bit SR[S] y la máscara SR[I] al valor de 7 (Inhibición de interrupciones). Después de esto, el registro VBR es inicializado al valor 0x0000\_0000 junto con los registros de especificación de configuración de la CPU. La máquina ejecuta el proceso de excepción de RESET, mediante la realización de dos ciclos de lectura de bus, sobre dos registros tipo long (32 bits); el primero localizado en la dirección 0x0000\_0000 y es cargado dentro del SSP (*Supervisor Stack Pointer*); el segundo localizado en la dirección 0x0000\_0004 y es cargado dentro del registro PC (*Program Counter*). Después de que la primera instrucción ha sido traída desde la memoria, la ejecución del programa comienza en la dirección apuntada por el PC. Si un error de acceso ocurre, antes de que la primera instrucción haya sido ejecutada, el procesador entra en el estado de excepción *fault on fault halt*. Los procesadores ColdFire® guardan información de configuración dentro de los registros D0 y D1 de 32 bits, una vez el sistema se ha recuperado del estado de RESET. La configuración del *hardware* instalado es almacenada en el registro D0, como se muestra en la Figura 5.13. Con esto se permite que el sistema emulador lea el contenido del *hardware* instalado, vía BDM, antes de la ejecución de instrucciones.

### Configuración del hardware (D0)

BDM: Lectura : 0x60 (D0)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Lectura:																
Escrivura:																
Reset:	1	1	0	0	1	1	1	1	0	0	0	1	0	0	0	0
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Lectura:	MAC	DIV	EMAC	FPU	MMU	CAU	0	0			ISA				DEBUG	
Escrivura:																
Reset:	*	*	*	0	0	*	0	0	0	0	1	0	1	0	0	1

\* : Función Opcional

Figura 5.13. Configuración del *hardware* en D0.

La Tabla 5.8 detalla cada bit o grupo de bits de configuración del hardware, que viene implementado en los procesadores de la familia ColdFire®.

**Tabla 5.8. Descripción del hardware implementado según**

BIT	DESCRIPCIÓN
31–24 PF	Especifica la familia y es fijo en 0xCF, indicando núcleo ColdFire
23–20 VER	<b>Versión del núcleo</b> 0001 V1 ColdFire 0010 V2 ColdFire 0011 V3 ColdFire 0100 V4 ColdFire 0101 V5 ColdFire <b>Los demás valores se reservan para uso futuro</b>
19–16 REV	<b>Número de revisión del procesador</b>
15 MAC	<b>Si hay presencia de la opción MAC (Unidad Multiplicadora Acumuladora)</b> 0 La MAC no está presente 1 La MAC está presente
14 DIV	<b>Unidad divisora de enteros presente</b> 0 Unidad divisora no presente 1 Unidad divisora presente
13 EMAC	<b>Presencia opcional de Unidad Multiplicadora y Acumuladora Aumentada</b> 0 Unidad EMAC no presente 1 Unidad EMAC presente
12 FPU	<b>Unidad opcional de Punto Flotante presente</b> 0 Unidad FPU no presente 1 Unidad FPU presente
11 MMU	<b>La Unidad Virtual para Manejo de Memoria está presente</b> 0 La MMU no está presente 1 La MMU está presente

El tamaño de la memoria se puede consultar en el registro D1, una vez se ha salido del estado de RESET, la Figura 5.14 muestra los tamaños en FLASH y RAM que vienen implementados en las máquinas ColdFire®. La Tabla 5.9 detalla cada bit o grupo de bits de configuración del *hardware* de la memoria, que viene implementado en los procesadores de la familia ColdFire®.

## Hardware de la Plataforma Memoria (D1)

BDM: Lectura : 0x61 (D1)

	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Lectura:	0	0	0	0	0	0	0	0		FLASHSZ		0	0	0	0	0
Escritura:																
Reset:	0	0	0	0	0	0	0	0	*	*	*	*	0	0	0	0
	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Lectura:	0	0	0	0	0	0	0	0		RAMSZ		0	0	0	0	0
Escritura:																
Reset:	0	0	0	0	0	0	0	0	*	*	*	*	0	0	0	0

\* : Dependiendo del modelo

Figura 5.14. Configuración del *hardware* de la memoria en D1.

Tabla 5.9. Descripción del hardware de la memoria según D0.

BIT	DESCRIPCIÓN
31–24	Reservado
23–20 FLASHSZ	<b>Tamaño de la FLASH</b> 0000-0101 Sin FLASH 0110 16 Kbytes 0111 32 Kbytes 1000 64 Kbytes 1001 128 Kbytes 1010 256 Kbytes 1011 512 Kbytes  <b>Los demás reservados para modelos futuros</b>
19–8	Reservado
7–4 RAMSZ	<b>Tamaño de la RAM</b> 0000 Sin RAM 0001 512 bytes 0010 1 Kbytes 0011 2 Kbytes 0100 4 Kbytes 0101 8 Kbytes 0110 16 Kbytes 0111 32 Kbytes 1000 64 Kbytes 1001 128 Kbytes  <b>Los demás reservados para modelos futuros</b>
3–0	Reservado

## 5.4. LA PILA Y EL PUNTERO A PILA

Definido en el aparte 5.1, el puntero a pila de las máquinas ColdFire® depende del nivel en el cual la máquina se encuentre trabajando. El SSP se define como el puntero a pila en el nivel de supervisor y USP como el puntero a pila a nivel de usuario, utilizando los registros A7 y OTHER A7.

Ambos punteros trabajan en pilas tipo LIFO sobre la RAM y deben ser inicializados una vez el sistema se haya recuperado del estado de RESET. Los punteros deben ser llevados a una dirección alta de la RAM con el propósito de tener buena movilidad ante el llamado de subrutinas o atención a las excepciones de la máquina. Una instrucción típica en lenguaje *assembler* para la inicialización de los registros SSP y/o USP se muestra a continuación:

```
LEA #0x00800910 , A7 ;Inicializa puntero a pila (base de la pila)
```

En este ejemplo, la dirección 0x00800928 se convierte en la base de la pila y a partir de allí el puntero a pila se mueve en forma LIFO, de tal manera que al presentarse un llamado a una subrutina y al retornar de la misma, el movimiento del puntero sería como el ilustrado en la Figura 5.15.

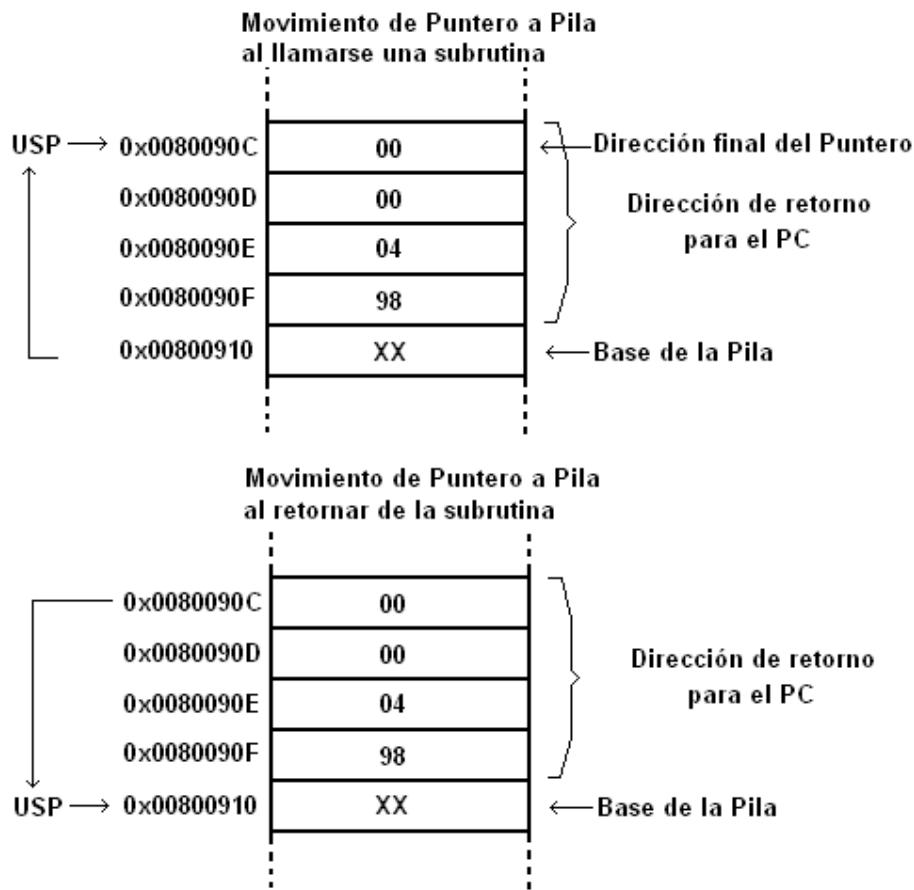


Figura 5.15. Movimiento del puntero a pila.

Para el caso de una atención a un evento de excepción, el mecanismo de la pila y el puntero de pila almacenan la dirección de retorno del sistema (valor del registro PC antes de atender el evento), el contenido del registro SR, el contenido del registro A0 y el contenido del registro D0.

## 5.5. REFERENCIAS

- MCF51JM128 Integrated Microcontroller Reference Manual. Rev 1. 01/2008. Freescale Semiconductor.
- ColdFire® Family, Programmer's Reference Manual. Rev 3. 03/2005. Freescale.
- Versión 1 ColdFire® White paper. Rev 0. 07/2006. Freescale Semiconductor.

## 5.6. PREGUNTAS

- ¿A qué definición corresponde las siglas ISA-C?
- ¿Cuántos y cuáles son los tipos de datos que soporta la arquitectura ColdFire® V1?
- Defina los términos Dhrystone y MIPS.
- Calcule el número del vector de interrupción 32 en una máquina HCS08, desde el punto de vista de una máquina ColdFire® V1.
- ¿Cuál es la característica más importante del núcleo del procesador ColdFire® V1?
- ¿Qué tamaño tiene el bus de direcciones de la máquina ColdFire® V1?
- De acuerdo a la pregunta anterior, ¿hasta cuanta memoria podría direccional el MCU?
- Defina brevemente modo RUN de operación para la ColdFire® V1.
- ¿Cuáles son los modos de bajo consumo de la máquina ColdFire® V1?
- ¿Cómo se llaman los registros internos de la arquitectura ColdFire® V1 y qué tamaño tienen?
- ¿Cuál es el nombre para el puntero a pila de usuario y qué registro lo interpreta?
- Defina las banderas contenidas en el registro CCR (*Code Condition Register*).
- ¿Qué es una excepción?
- ¿Una interrupción es una excepción? Justifique su respuesta.
- Describa los pasos generales que ejecuta el núcleo del MCU, al presentarse un evento de excepción.
- ¿Cuántos niveles de interrupción se pueden lograr en la máquina de 8 bits y cuántos en la máquina de 32 bits?
- ¿En qué consiste un evento de excepción por error de acceso?

# CAPÍTULO 6

## Modos de Direcciónamiento y Juego de Instrucciones

**6.1. Modos de direccionamiento**

**6.2. Resumen del juego de instrucciones**

**6.3. Ejercicios con direccionamientos e instrucciones**

**6.4. Referencias**

**6.5. Preguntas**

## 6.1. MODOS DE DIRECCIONAMIENTO

En la mayoría de los cálculos de las operaciones que contienen operando fuente y operando destino, se almacena el resultado en el operando destino. Para los cálculos hechos en las operaciones con un sólo operando, el resultado es almacenado en la localización destino. Los ejemplos aquí ilustrados se trabajan sobre lenguaje ensamblador (*assembler*), para una ilustración más detallada sobre cada modo de direccionamiento.

- **Formato de una Instrucción:** La longitud de las instrucciones en la familia ColdFire® son de 1, 2 y hasta 3 palabras. La Figura 6.1 ilustra el formato general de una instrucción.

Palabra de Operación
Extensión 1
Extensión 2

**Figura 6.1. Formato de una instrucción.**

La primera palabra, llamada la Palabra de Operación, especifica la longitud de la instrucción, el tipo de direccionamiento efectivo y la operación a ser realizada. Las dos palabras, llamadas Extensiones, especifican otra parte de la operación o los operandos involucrados en ésta.

Las extensiones pueden ser predicados de condicionales, operandos constantes o inmediatos, extensiones del modo de direccionamientos efectivo especificado en la palabra de operación, saltos o desplazamientos, números de bits, especificaciones especiales de registro, operandos de la instrucción TRAP, argumentos de contadores o palabras en punto flotante.

Para el lenguaje *assembler*, las instrucciones se editan mediante la siguiente convención:

ETIQUETA: OPCODE      FUENTE , DESTINO      ;COMENTARIOS
--

En donde:

**ETIQUETA:** Referencia como dirección de la línea de código.

**OPCODE:** Código de Operación o instrucción.

**FUENTE:** Operando fuente de la información.

**DESTINO:** Operando destino de la información.

**COMENTARIOS:** Explican el propósito de la línea de código.

Una instrucción especifica el tipo de operación a realizar y define la localización de cada operando. La Figura 6.2 ilustra sobre el formato de una instrucción con direccionamiento del tipo indexado o indirecto.

### Formato de una Palabra de Instrucción

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-	-	-	-	-	-	-	-	-	-	Dirección Modo	Efectiva Registro				

### Extensión 1 de una Palabra de Instrucción

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D/A	Registro	W/L	Estado	0	Desplazamiento										

- : No definida

**Figura 6.2.** Formato de una instrucción con extensión.

La Tabla 6.1 muestra la definición de cada campo en el formato de palabra de instrucción con extensión. La extensión 2, de haberla, correspondería al valor de un operando.

**Tabla 6.1. Definición de campos de instrucción.**

CAMPO	DEFINICIÓN
<b>Instrucción</b>	
Modo	Modo de Direccionamiento
Registro	Número general del Registro
<b>Extensions</b>	
D/A	Tipo de Registro Índice 0 = $D_n$ 1 = $A_n$
W/L	Word/Longword 0 = Excepción por Error de Direccionamiento 1 = Longword
Escala	Factor de Escalado 00 = 1 01 = 2 10 = 4 11 = 8 ( Solo para la opción FPU )

- **Modos de Direccionamiento Efectivos:** A continuación del *opcode* (Código de Operación) de una instrucción se debe definir la localización de cada operando, de acuerdo a las siguientes tres premisas:

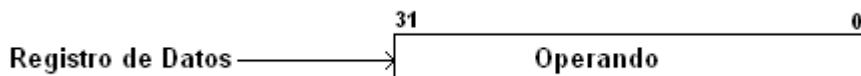
- Un campo de registro a continuación de una instrucción, deberá especificar el registro a ser usado.
- Un campo de direccionamiento efectivo, deberá contener información del modo de direccionamiento.
- La definición de una instrucción deberá implicar el uso de un registro específico. Otros campos a continuación de la instrucción, especifican si el registro seleccionado es una dirección o un dato y cómo el registro será usado.

Todo modo de direccionamiento tiene una sintaxis en lenguaje *assembler*. A continuación se detalla la estructura y sintaxis de los diferentes modos de direccionamiento de la familia ColdFire®.

- **Modo de Direccionamiento Directo a Registro de Datos:** El campo de dirección efectiva especifica el registro de datos que contiene el operando (ver Figura 6.3).

**NOTA:** A manera de convención, se seguirá llamando a la dirección efectiva de un direccionamiento con las iniciales en inglés EA (*Effective Address*).

Generación de la dirección:	EA = Dn
Sintaxis en ensamblador:	Dn
EA en el campo de modo:	000
EA en el campo de registro:	Número del registro (depende de n)
Número de palabras de extensión:	0



**Figura 6.3. Diagrama de direccionamiento directo a registro de datos.**

**Ejemplo:**

Sea el registro de datos D1, contenido la cantidad 0x11111111. Al aplicar la operación NEG (Negar como complemento a dos) sobre éste registro:

NEG D1	;Niegue el contenido del registro de
--------	--------------------------------------

;datos D1

La instrucción del ejemplo ejecuta la operación:  $0 - (D1) \rightarrow D1$ . En este caso el registro de datos D1 es fuente y destino de la operación, de tal manera que al final de la operación el contenido de D1 será cambiado por 0xEEEEEEE, que no es más que el complemento a dos de la cantidad original.

- **Modo de Direcciónamiento Directo a Registro de Dirección:** El campo de dirección efectiva especifica el registro de dirección que contiene el operando (ver Figura 6.4).

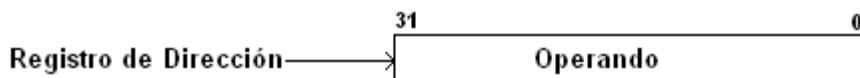
Generación de la dirección:  $EA = An$

Sintaxis en ensamblador:  $An$

EA en el campo de modo:  $001$

EA en el campo de registro: Número del registro (depende de n)

Número de palabras de extensión:  $0$



**Figura 6.4. Diagrama de direcciónamiento directo a registro de dirección.**

**Ejemplo:**

Sea el registro de dirección A2, conteniendo la cantidad 0x00800900 y el puntero de pila de usuario USP (*Stack Pointer* = A7) apuntando a la dirección 0x00800A00 de la pila.

Al aplicar la operación UNLK (Desligar) sobre éste registro:

UNLK A2 ;Almacene en A2 el contenido de las  
;celdas apuntadas por el USP

La instrucción del ejemplo ejecuta la operación:  $A2 \rightarrow SP; (SP) \rightarrow A2; SP + 4 \rightarrow SP$ . La Figura 6.5 muestra el estado de la pila para este ejemplo, tanto antes como al final de la instrucción UNLK.

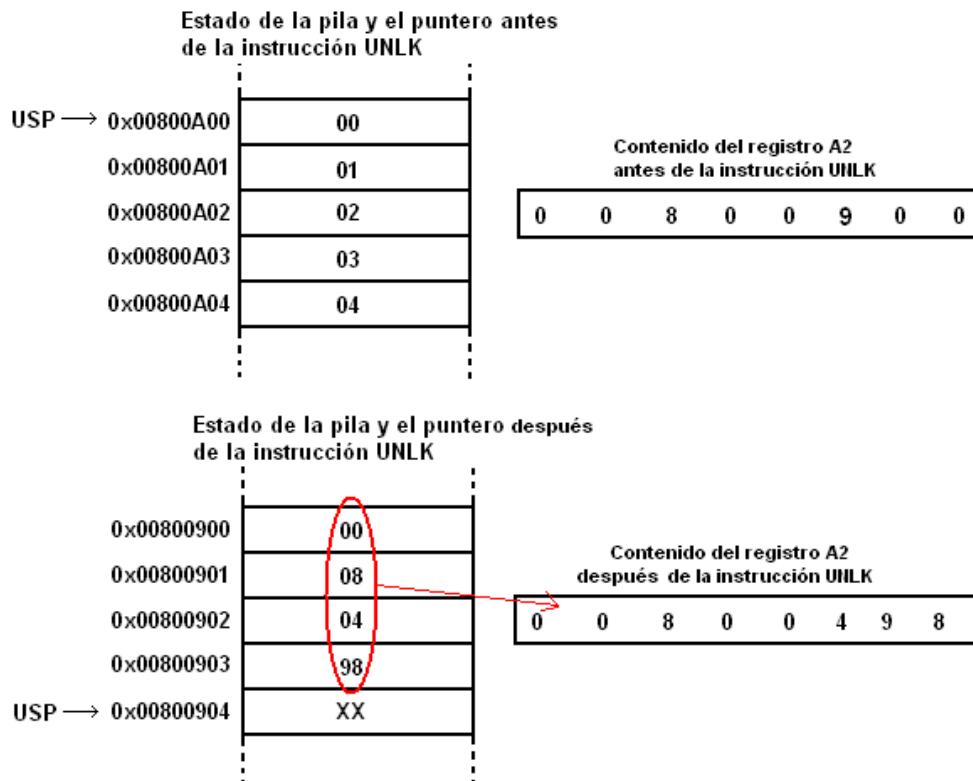


Figura 6.5. Ejemplo para direccionamiento directo a registro de dirección.

- **Modo de Direccionamiento Indirecto a Registro:** El operando de la instrucción se encuentra en la memoria y el campo de dirección efectiva se forma con el contenido del registro de dirección (puntero), en otras palabras, el operando está siendo direccionado por un puntero An (ver Figura 6.6).

Generación de la dirección:	$EA = (An)^{25}$
Sintaxis en ensamblador:	(An)
EA en el campo de modo:	010
EA en el campo de registro:	Número del registro (depende de n)
Número de palabras de extensión:	0

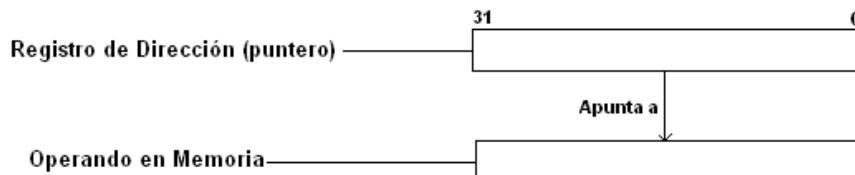


Figura 6.6. Diagrama para direccionamiento indirecto.

<sup>25</sup> La notación (An) indica direccionamiento indirecto, en donde An se comporta como el puntero a la dirección contenida en este.

**Ejemplo:**

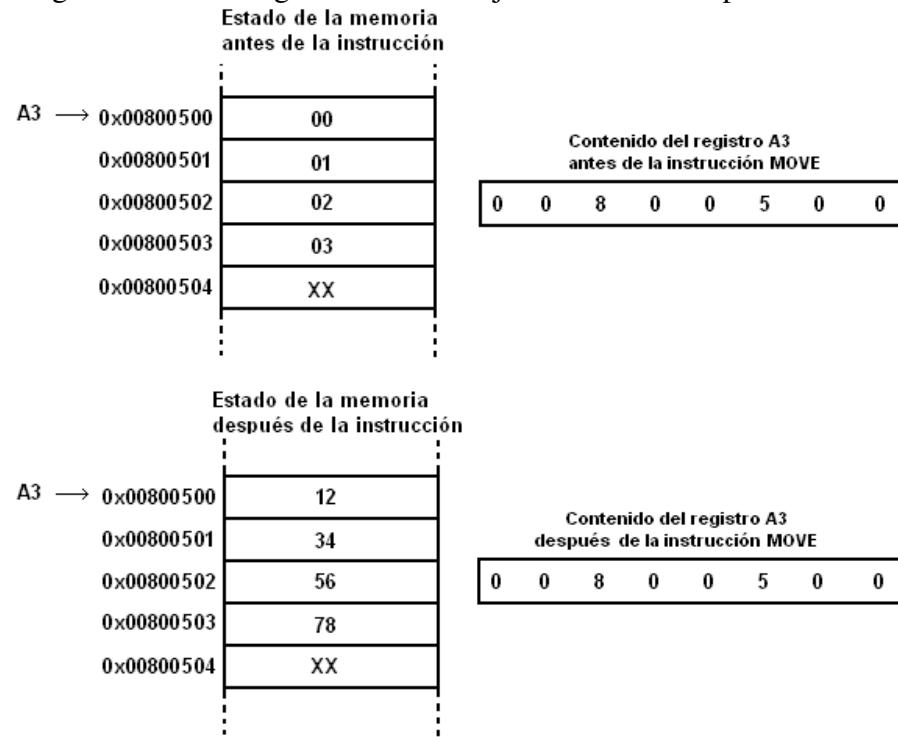
Sea el registro de dirección A3, conteniendo la cantidad 0x00800500.

Al aplicar la operación MOVE (Mover), como se indica a continuación:

```
MOVE.L      #0x12345678 , (A3) ;Almacene en la celda con  
;dirección 0x00800500 la  
;cantidad 0x12345678
```

La instrucción del ejemplo ejecuta la operación: #0x12345678 → (A3), es decir, mueve una *longword* hacia 4 bytes ubicados a partir de la dirección de memoria 0x00800500, utilizando la convención *Big endian*.

La Figura 6.7 muestra gráficamente la ejecución de esta operación.

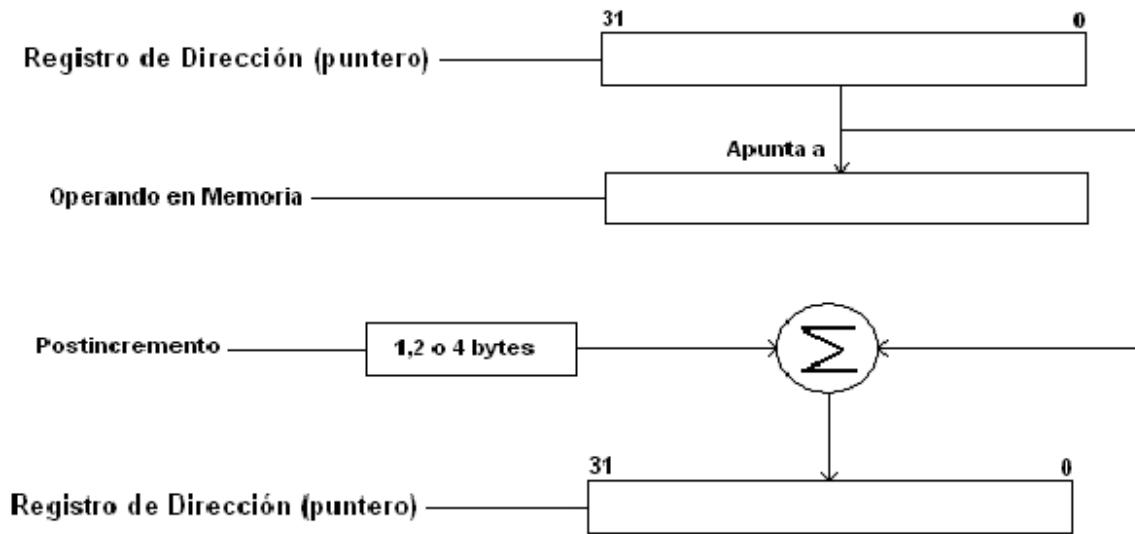


**Figura 6.7. Ejemplo para direccionamiento indirecto a registro.**

- **Modo de Direccionamiento Indirecto a Registro con Postincremento:** El operando de la instrucción se encuentra en la memoria y el campo de dirección efectiva se forma con el contenido del registro de dirección (puntero).

Al finalizar la operación el contenido del puntero se incrementa en 1, 2 o 4, según el tamaño del operando (ver Figura 6.8). El registro A7 (puntero a pila) puede ser utilizado con este tipo de direccionamiento.

Generación de la dirección:	$EA = (An) ; An = An + (\text{tamaño en bytes del operando})$
Sintaxis en ensamblador:	$(An)+$
EA en el campo de modo:	011
EA en el campo de registro:	Número del registro (depende de n)
Número de palabras de extensión:	0



**Figura 6.8. Diagrama para direccionamiento indirecto a registro con postincremento.**

**Ejemplo:**

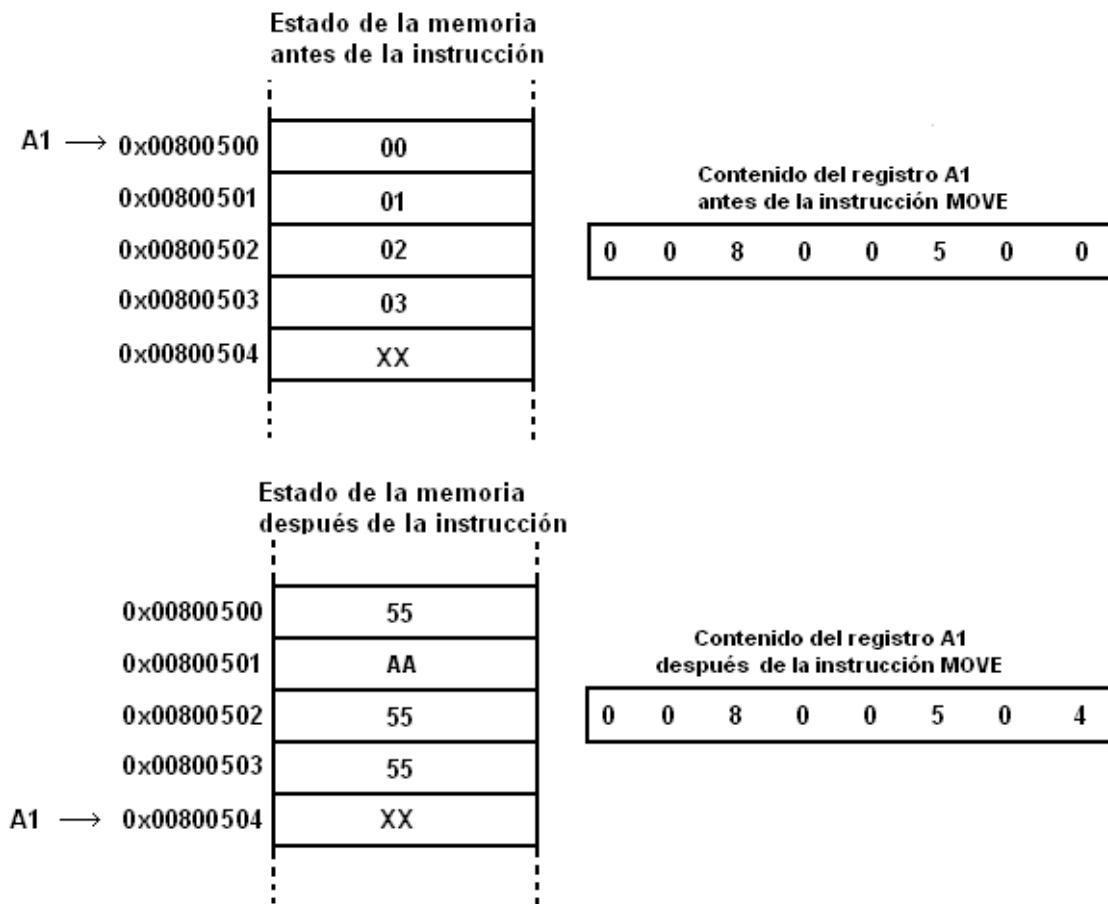
Sea el registro de dirección A1, conteniendo la cantidad 0x00800500.

Al aplicar la operación MOVE (Mover), como se indica a continuación:

```
MOVE.L      #0x55AA5555 , (A1)+;Almacene a partir de la
;celda con dirección
; 0x00800500
```

La instrucción del ejemplo ejecuta la operación: #0x55AA5555 → (A1), es decir, mueve una *longword* hacia 4 bytes ubicados a partir de la dirección de memoria 0x00800500, utilizando la convención *Big endian*. La Figura 6.9 muestra gráficamente la ejecución de esta operación.

Al finalizar la operación el contenido del puntero se incrementa en 4, debido a que el operando ejecutado es del tipo *longword*.

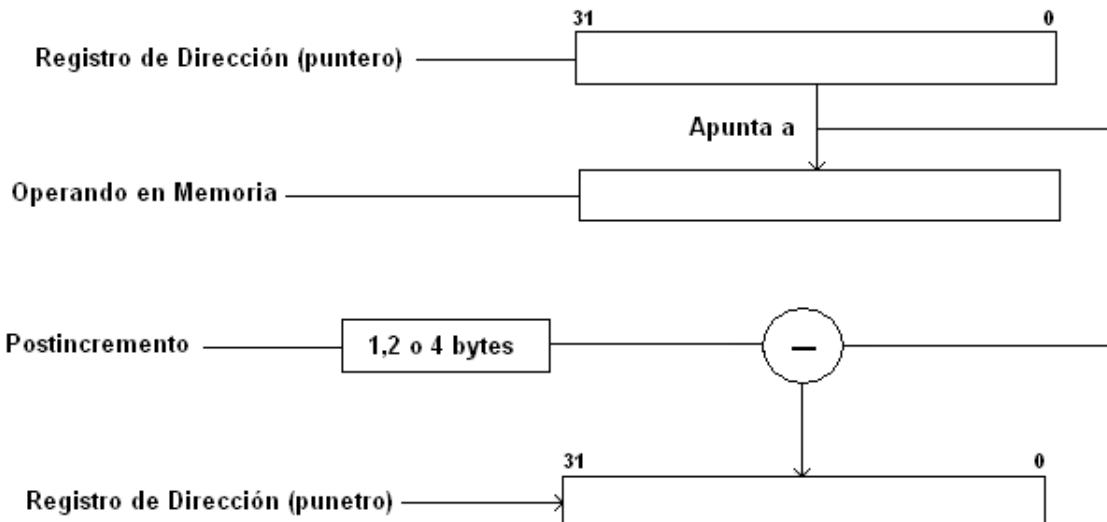


**Figura 6.9. Ejemplo para direccionamiento indirecto a registro con postincremento**

- **Modo de Direccionamiento Indirecto a Registro con Predecremento:** El operando de la instrucción se encuentra en la memoria y el campo de dirección efectiva se forma con el contenido del registro de dirección (puntero).

Al finalizar la operación el contenido del puntero se decrementa en 1, 2 o 4, según el tamaño del operando (ver Figura 6.10). El registro A7 (puntero a pila) puede ser utilizado con este tipo de direccionamiento.

Generación de la dirección:	$EA = (An) ; An = An - (\text{tamaño en bytes del operando})$
Sintaxis en ensamblador:	$-(An)$
EA en el campo de modo:	100
EA en el campo de registro:	Número del registro (depende de n)
Número de palabras de extensión:	0



**Figura 6.10. Diagrama para direccionamiento indirecto a registro con predecremento.**

**Ejemplo:**

Sea el registro de dirección A1, conteniendo la cantidad 0x00800500.

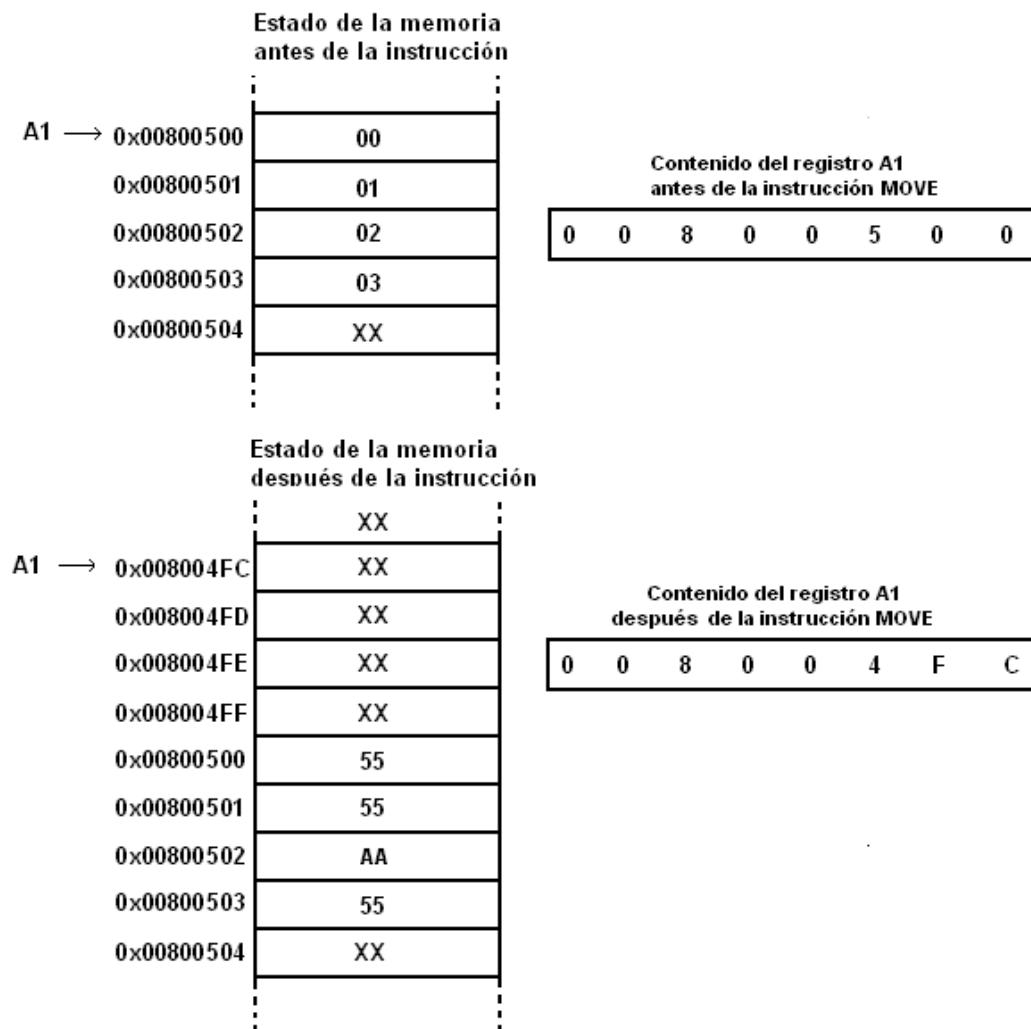
Al aplicar la operación MOVE (Mover), como se indica a continuación:

```
MOVE.L      #0x5555AA55 , -(A1);Almacene a partir de la
            ;celda con dirección
            ;0x00800500
```

La instrucción del ejemplo ejecuta la operación:  $\#0x5555AA55 \rightarrow (A1)$ , es decir, mueve una *longword* hacia 4 bytes ubicados a partir de la dirección de memoria 0x00800500, utilizando la convención *Big endian*. La Figura 6.10 muestra gráficamente la ejecución de esta operación.

Al finalizar la operación el contenido del puntero se decrementa en 4, debido a que el operando ejecutado es del tipo *longword*.

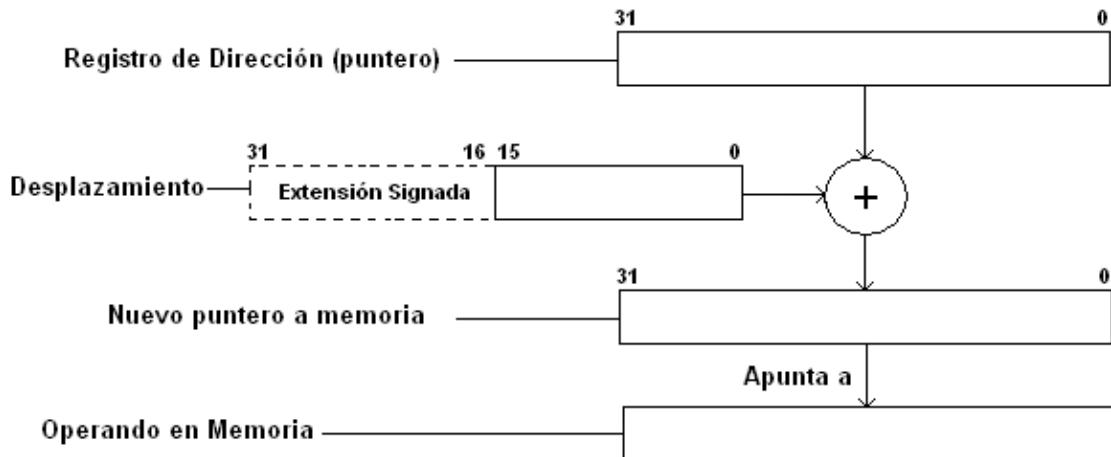
- **Modo de Direcciónamiento Indirecto con Desplazamiento:** El operando de la instrucción se encuentra en la memoria y el campo de dirección efectiva se forma con el contenido del registro de dirección (puntero) más un desplazamiento. Este desplazamiento está formado por un número entero en aritmética signada de 16 bits, que puede contener una extensión hasta 32 bits.



**Figura 6.11. Ejemplo para direccionamiento indirecto a registro con predecremento**

Generación de la dirección:	$EA = (An) + d_{16}$
Sintaxis en ensamblador:	$d_{16}(An)$
EA en el campo de modo:	101
EA en el campo de registro:	Número del registro (depende de n)
Número de palabras de extensión:	1 (el desplazamiento)

La Figura 6.12 representa el diagrama de un direccionamiento indirecto a registro con desplazamiento.



**Figura 6.12. Diagrama para direccionamiento indirecto a registro con desplazamiento.**

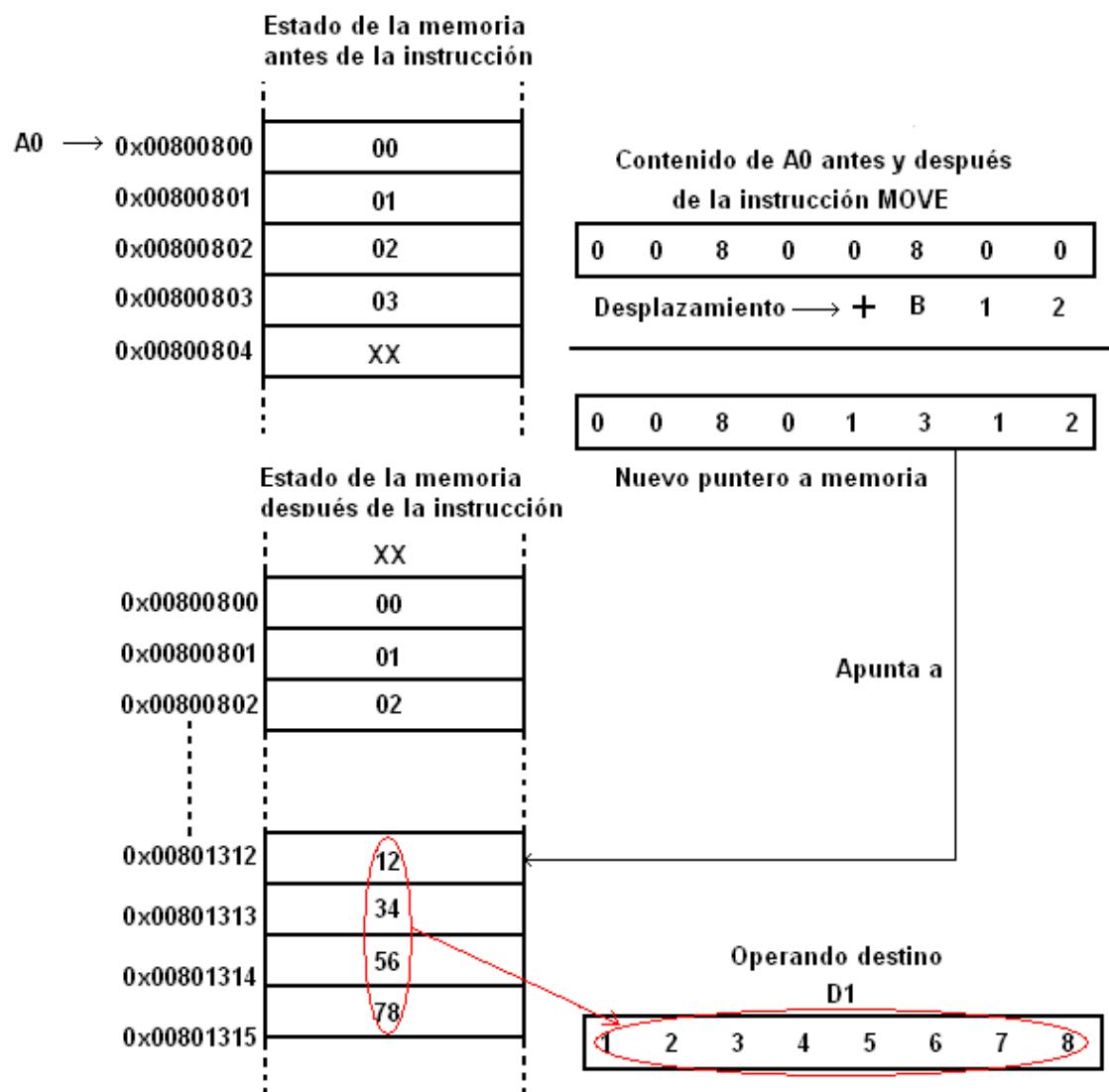
**Ejemplo:**

Sea el registro de dirección A0, conteniendo la cantidad 0x00800800.

Al aplicar la operación MOVE (Mover), como se indica a continuación:

```
MOVE.L    0xB12(A0) , D1      ;Almacene en D1 una longword,  
                                ;que se encuentra a partir de la celda  
                                ;con dirección 0x00800800 más un  
                                ;desplazamiento de 0xB12
```

La Figura 6.13 muestra el movimiento de la información para el ejemplo planteado.



**Figura 6.13. Ejemplo para direccionamiento indirecto a registro con desplazamiento**

- **Modo de Direccionamiento Indirecto con Índice Escalado y Desplazamiento en 8 bits:** Este modo de direccionamiento utiliza un registro como puntero índice, el cual se escala por un factor y un desplazamiento que se suma a la dirección resultante. El operando se encuentra en la memoria. El usuario deberá especificar el registro de dirección, el desplazamiento, el factor de escala y el registro índice.

Generación de la dirección:  $EA = (An) + ((Xi) * SF) + d_8$  con extensión de signo

Sintaxis en ensamblador:  $d_8 (An, Xi * SF)$

EA en el campo de modo:

110

EA en el campo de registro:

Número del registro (depende de n)

Número de palabras de extensión: 1 (el desplazamiento)

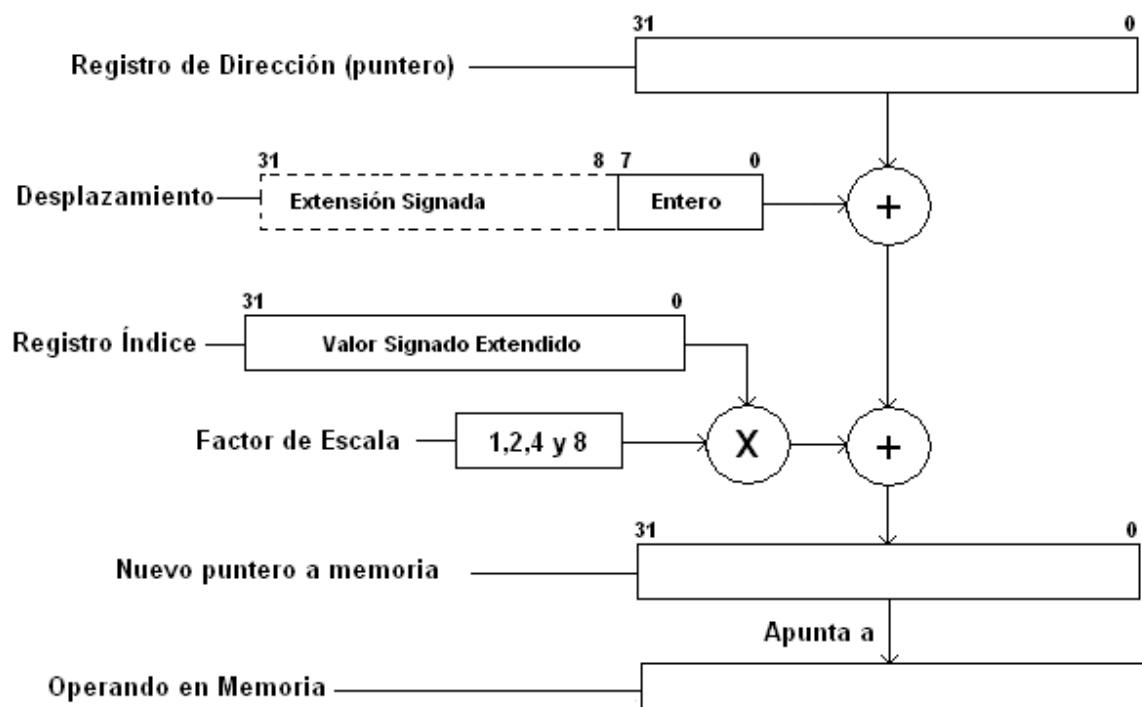
En donde:

**SF (Scale Factor):** Factor de Escala

**Xi:** Registro índice

**$d_8$ :** Desplazamiento en 8bits (signado), con posibilidad de valor extendido (signado).

La Figura 6.14 representa el diagrama de un direccionamiento indirecto a registro con escalamiento y desplazamiento.



**Figura 6.14. Diagrama para direccionamiento indirecto a registro con índice escalado y desplazamiento en 8 bits.**

**Ejemplo:**

Para las siguientes premisas, determinar en donde se encuentra el dato que será llevado al registro D5, en formato de una palabra.

A4 = 0x00800700: Registro de dirección.

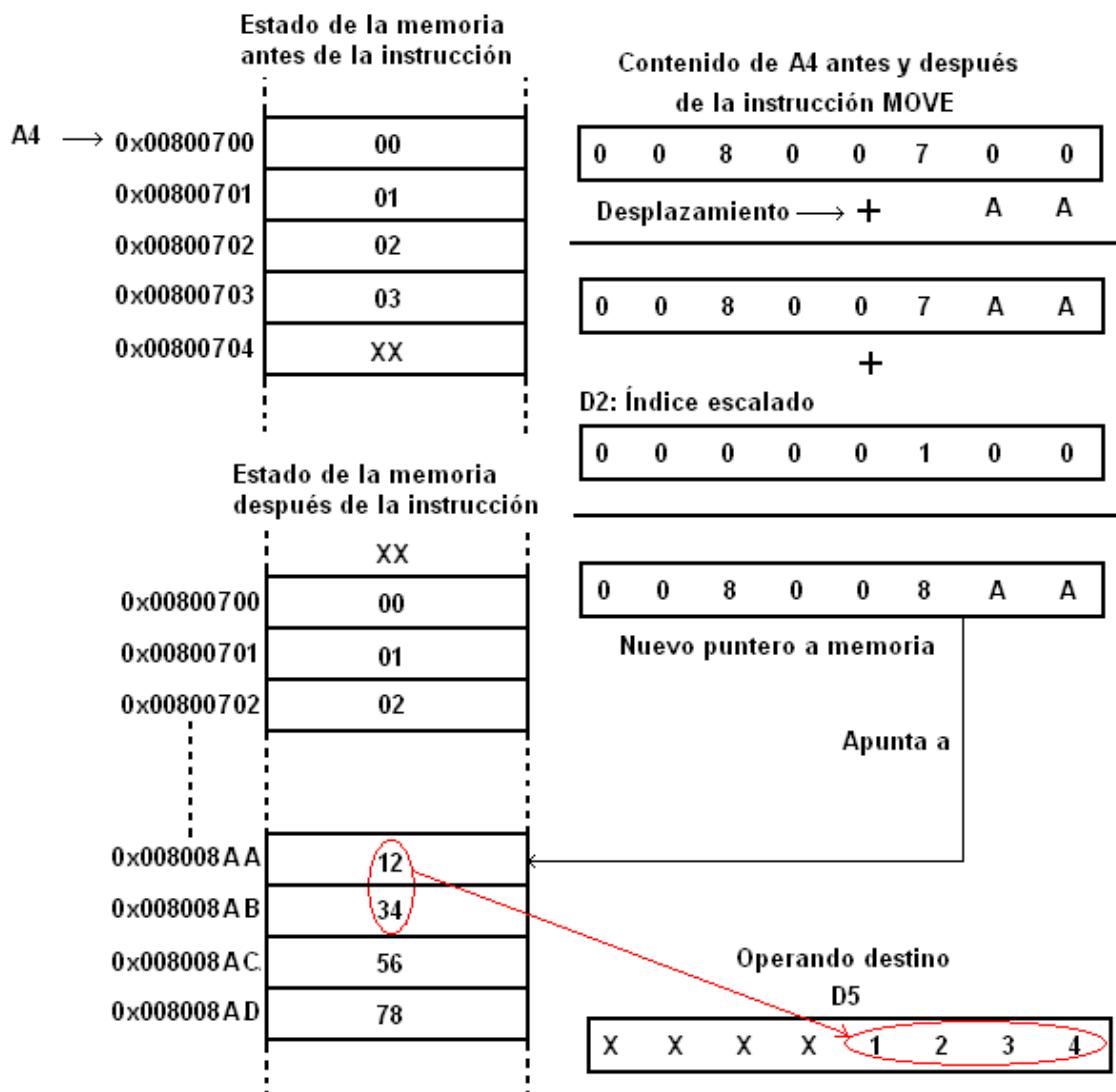
D2 = 0x00000080: Registro índice.

SF = 2: Factor de Escala.

Al aplicar la operación MOVE (Mover), como se indica a continuación:

```
MOVE.W    0xAA(A4,D2*2) , D5 ;Almacene en D5 una palabra, que se
;encuentra a partir de la celda ;con
;dirección 0x00800700, más un
;desplazamiento de 0xAA y más
;el contenido del índice D2 escalado
;en 2
```

La Figura 6.15 muestra el movimiento de la información para el ejemplo planteado.

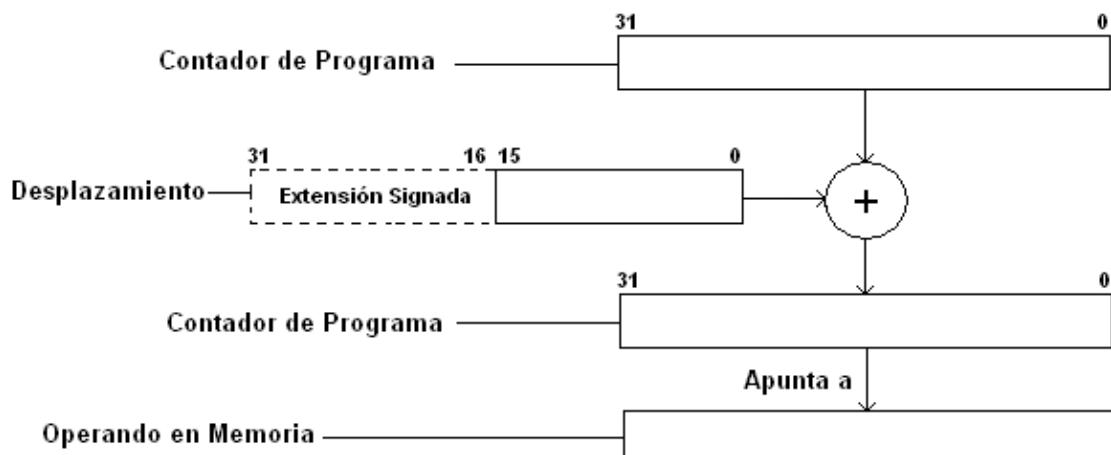


**Figura 6.15. Ejemplo para direccionamiento indirecto a registro con índice escalado y desplazamiento en 8 bits.**

- **Modo de Direcciónamiento Indirecto con Contador de Programa (PC) y desplazamiento:** En este modo el operando se encuentra en la memoria. La dirección de la ubicación del operando es obtenida al sumar el contenido del PC y un desplazamiento en aritmética signada de 16 bits. El valor del PC al momento de efectuarse la instrucción en curso es de PC + 2.

Este tipo de direcciónamiento tiene que ver con el desplazamiento natural del contador de programa, tanto en los llamados a subrutinas, instrucciones de salto y ramificación, como las atenciones a interrupciones. La Figura 6.16 ilustra sobre el uso del PC como puntero a una nueva instrucción (operando)

Generación de la dirección:	$EA = (PC) + d_{16}$
Sintaxis en ensamblador:	$d_{16}(PC)$
EA en el campo de modo:	111
EA en el campo de registro:	010
Número de palabras de extensión:	1 (el desplazamiento)



**Figura 6.16. Diagrama de direcciónamiento indirecto con el PC y desplazamiento en 16 bits.**

**Ejemplo:**

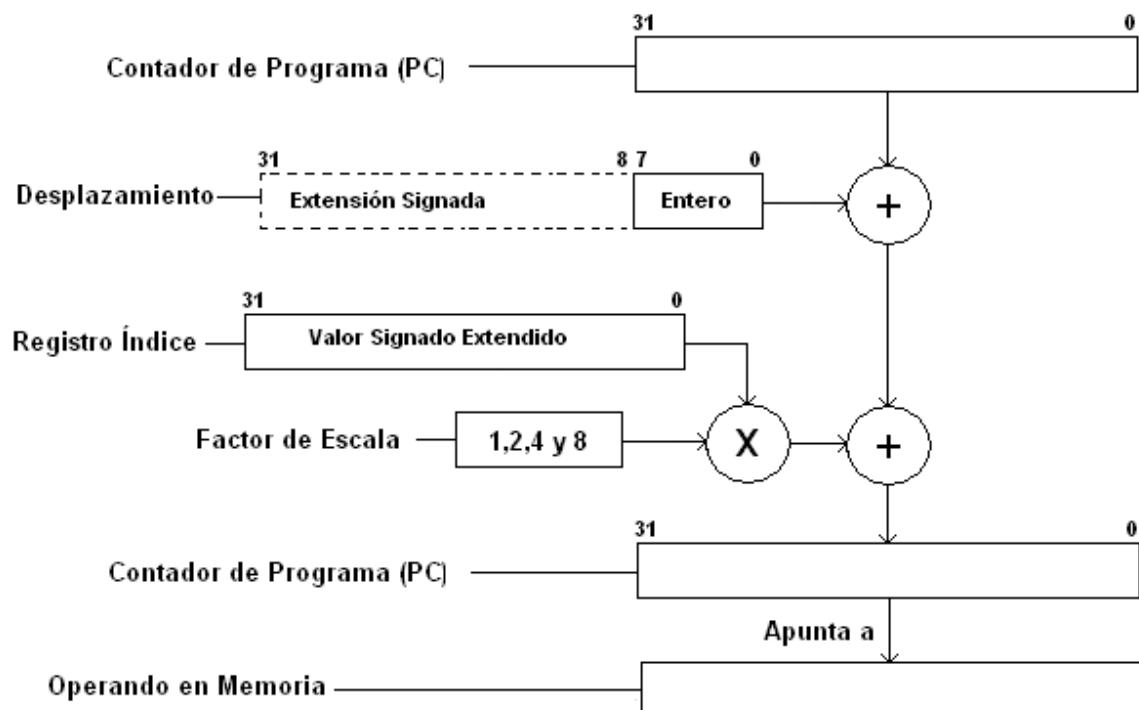
BRA	LOOP	;Salte a LOOP
-----	------	---------------

Esta instrucción ejecuta la operación  $PC + d_{16} \rightarrow PC$ . En este caso el salto como aritmética signada de 16 bits es calculado por el compilador y define la dirección LOOP.

- **Modo de Direccionamiento Indirecto con Contador de Programa (PC) más índice escalado y desplazamiento en 8 bits:** En este modo el operando se encuentra en la memoria. La dirección de la ubicación del operando es obtenida al sumar el contenido del PC más el contenido de un puntero índice escalado y un desplazamiento en aritmética signada de 8 bits. El valor del PC al momento de efectuarse la instrucción en curso es de PC + 2.

Este tipo de direccionamiento tiene que ver con el desplazamiento natural del contador de programa, tanto en los llamados a subrutinas, instrucciones de salto y ramificación, como las atenciones a interrupciones. La Figura 6.17 ilustra sobre el uso del PC como puntero a una nueva instrucción (operando)

Generación de la dirección:	$EA = (PC) + (Xi * SF) + d_{16}$
Sintaxis en ensamblador:	$d_8(PC, Xi * SF)$
EA en el campo de modo:	111
EA en el campo de registro:	011
Número de palabras de extensión:	1 (el desplazamiento)

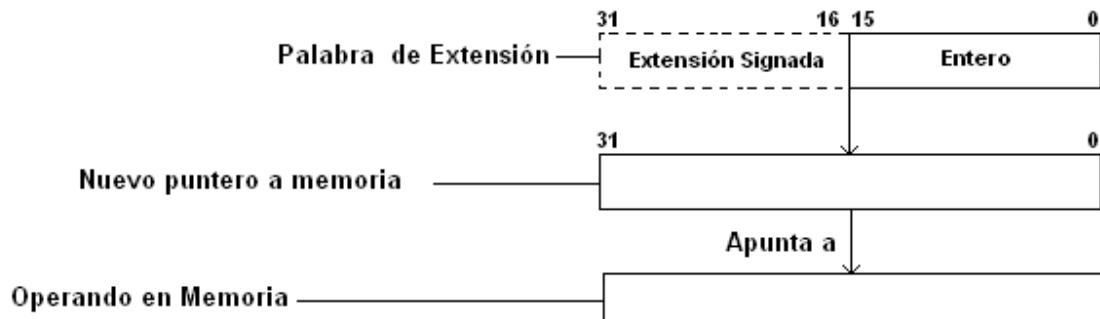


**Figura 6.17. Diagrama de direccionamiento indirecto con el PC más índice escalado y desplazamiento en 8 bits.**

- **Modo de Direccionamiento Absoluto Corto:** En este modo el operando se encuentra en la memoria y su dirección se encuentra en la palabra de extensión. El

direcccionamiento está comprendido en 16 bits, pero puede llegar a 32 bits de manera extendida y signada (ver Figura 6.18).

Generación de la dirección:	EA (Dada en la palabra de extensión)
Sintaxis en ensamblador:	(xxx).W
EA en el campo de modo:	111
EA en el campo de registro:	000
Número de palabras de extensión:	1 (La dirección)

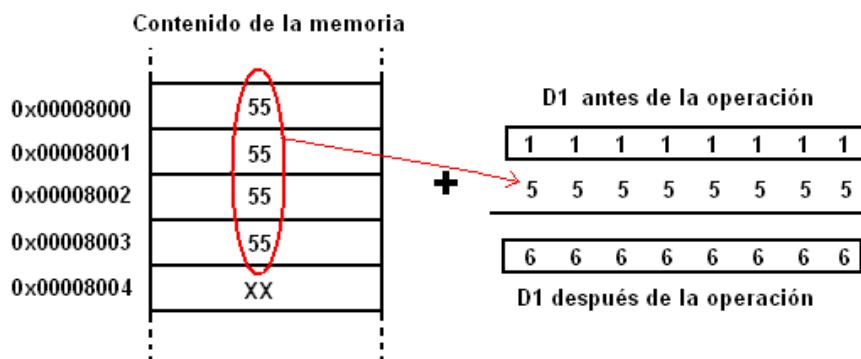


**Figura 6.18. Diagrama de Direccionamiento Absoluto Corto.**

**Ejemplo:**

La instrucción accede de manera directa a la posición de memoria absoluta 0x00008000 y suma el contenido de las celdas 0x00008000, 0x00008001, 0x00008002 y 0x00008003 al registro D1 (Ver Figura 6.19). Observe, que para este tipo de direcccionamiento no se necesita de un puntero para direccionar la memoria.

```
ADD.L      0x8000 , D1 ;Sume al registro D1 la longword
                           ; contenida en la dirección 0x8000
```



**Figura 6.19. Ejemplo para el direccionamiento absoluto corto.**

- **Modo de Direcccionamiento Absoluto Largo:** En este modo el operando se encuentra en la memoria y su dirección se forma con las palabras de extensión. La primera palabra de extensión contiene la parte alta de la dirección del operando en memoria y la segunda palabra de extensión contiene la parte baja (ver Figura 6.20).

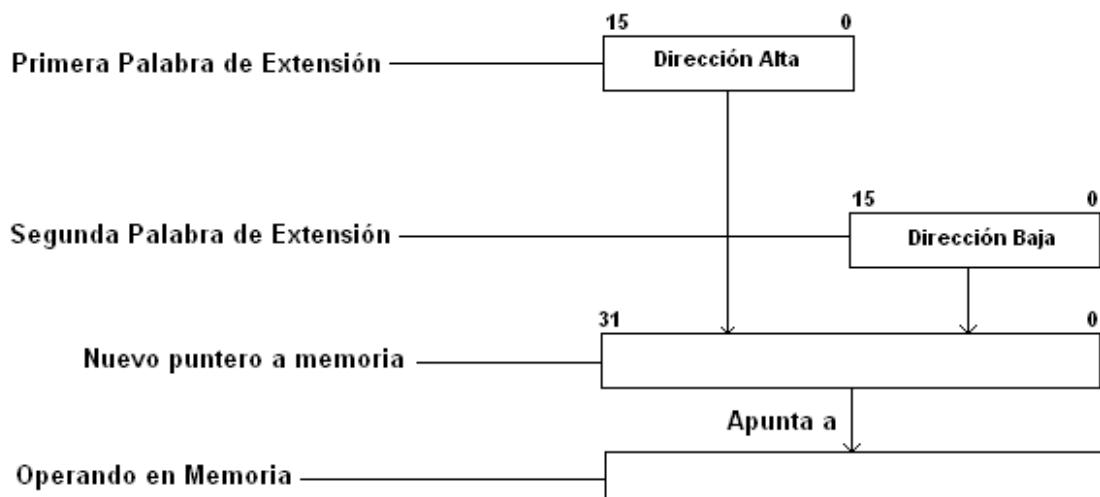
Generación de la dirección: EA (Dada en las palabras de extensión)

Sintaxis en ensamblador: (xxx).L

EA en el campo de modo: 111

EA en el campo de registro: 001

Número de palabras de extensión: 2 (La dirección)



**Figura 6.20. Diagrama de Direcccionamiento Absoluto Largo.**

**Ejemplo:**

La instrucción accede de manera directa a la posición de memoria absoluta 0xFFFF8043 y suma el contenido de las celdas 0xFFFF8043, 0x FFFF8043, 0x FFFF8043 y 0x FFFF8043 al registro D1 (Ver Figura 6.21). Observe, que para este tipo de direcccionamiento no se necesita de un puntero para direccionar la memoria.

```
ADD.L      0xFFFF8043 , D1      ;Sume al registro D1 la longword
                                ;contenido en la dirección
                                ;0xFFFF8043
```

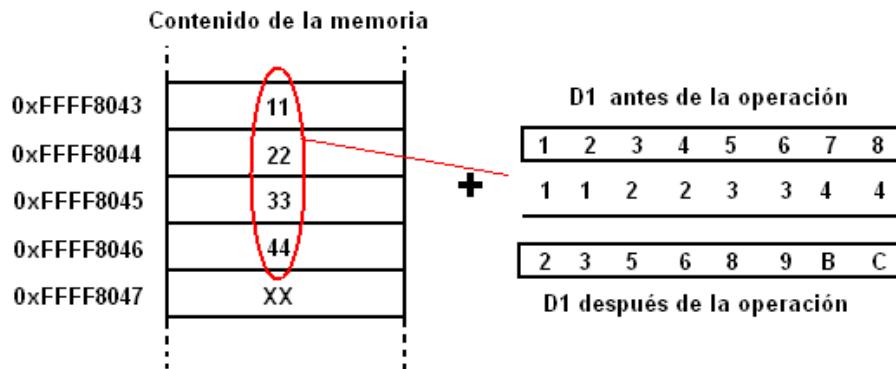


Figura 6.21. Ejemplo con el direccionamiento absoluto largo.

- **Modo de Direcciónamiento Inmediato a Dato:** En este modo de direcciónamiento el operando se forma con una o dos de las palabras de extensión.

Generación de la dirección:	Operando dado
Sintaxis en ensamblador:	# <xxx>
EA en el campo de modo:	111
EA en el campo de registro:	100
Número de palabras de extensión:	1 o 2 (el operando)

**Ejemplo:**

En los registros D0, D1 y A3, serán cargados diferentes formatos de dato de manera inmediata (ver Figura 6.22).

MOVE.B #0x69 , D0	;D0 asume el valor de 0x69
MOVE.W #0x6969 , D1	;D1 asume el valor de 0x6969
MOVE.L #0x69696969 , A3	;A3 asume el valor de 0x69696969

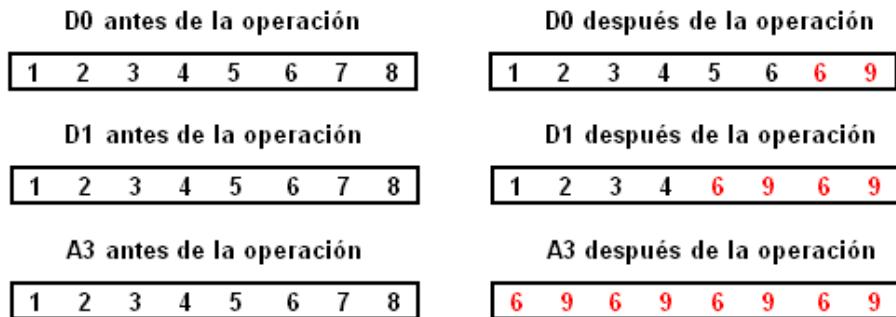


Figura 6.22. Ejemplo con el direccionamiento inmediato.

## 6.2. RESUMEN DEL JUEGO DE INSTRUCCIONES

En este aparte son descritas, de manera resumida, las instrucciones de la familia ColdFire® en sus versiones de código: ISA-A, ISA-B e ISA-C. Algunas convenciones de notación, útiles para la interpretación de la escritura de las instrucciones, son listados en la Tabla 6.2.

**Tabla 6.2. Convenciones para el juego de instrucciones.**

Operaciones con operandos simples y dobles	
+	Suma aritmética o indicador de postincremento
-	Resta aritmética o indicador de predecremento
*	Multiplicación aritmética
/	División aritmética
~	Complemento lógico
&	AND lógica
	OR lógica
$\perp$	OR exclusiva lógica
$\rightarrow$	Mover un operando fuente a un operando destino
$\leftrightarrow$	Intercambiar dos operandos
$<\text{op}>$	Alguna operación con operando doble
$<\text{operand}>\text{tested}$	Un operando es comparado con cero y se afecta el CCR
sign-extended	Los bits de la porción superior se igualan a los bits altos de la porción inferior
Otras Operaciones	
If $<\text{condition}>$ then $<\text{operations}>$ else $<\text{operations}>$	Si la condición se cumple, se ejecuta lo que sigue después del then, pero si no se cumple, se ejecuta lo que hay a continuación del else.
Especificaciones de Registros	
An	Cualquier registro de dirección
Ax, Ay	El destino y la fuente son registros de direcciones
Dn	Cualquier registro de datos
Dx, Dy	El destino y la fuente son registros de datos
Dw	El registro de datos contiene el residuo
Rc	Registro de control
Rn	Cualquier registro de datos o de dirección
Rx, Ry	El destino y la fuente pueden ser registro de datos o registro de direcciones
Xi	Registro índice, que puede ser cualquier registro de datos o direcciones (en 32 bits)
Subcampos y Calificadores	
# $<\text{data}>$	Una constante seguida del código de operación
( )	Identifica un direccionamiento indirecto
d <sub>n</sub>	Desplazamiento en n bits
sz	Tamaño del operando : Byte (B), Word (W), Longword (L)

lsb, msb	<b>Bit menos significativo, bit más significativo</b>
LSW, MSW	<b>Palabra menos significativa, palabra más significativa</b>
SF	<b>Factor de Escala</b>
<b>Nombre de Registros</b>	
CCR	<b>Registro de Código de Condiciones (parte baja del SR)</b>
PC	<b>Contador de Programa</b>
SR	<b>Registro de Estado</b>
USP	<b>Puntero a Pila del Usuario</b>
ic, dc, bc	<b>Instrucción, dato o ambos cache</b>
<b>Códigos de Condición</b>	
*	<b>Caso General</b>
C	<b>Bit de Carry en el CCR</b>
cc	<b>Algún bit de condición del CCR</b>
N	<b>Bit negativo del CCR</b>
V	<b>Bit de sobreflujo del CCR</b>
X	<b>Bit de acarreo extendido en el CCR</b>
Z	<b>Bit de cero en el CCR</b>
—	<b>No afecta o no aplica</b>
<b>Misceláneas</b>	
<ea>x, <ea>y	<b>Dirección efectiva del destino y la fuente , respectivamente</b>
<label>	<b>Etiqueta en lenguaje ensamblador</b>
#list	<b>Lista de registros</b>
<b>Operaciones con la MAC</b>	
ACC, ACCx	<b>Un registro específico de la MAC</b>
ACCx, ACCy	<b>Acumulador destino y fuente, respectivamente</b>
ACCext01	<b>4 bytes de extensión asociados con los acumuladores 1 y 2 de la EMAC</b>
ACCext23	<b>4 bytes de extensión asociados con los acumuladores 3 y 4 de la EMAC</b>
EV	<b>Bandera de sobreflujo de extensión en el registro MACSR</b>
MACSR	<b>Registro de estado de la MAC</b>
MASK	<b>Registro de máscara de la MAC</b>
PAVx	<b>Bandera de sobreflujo en Producto/acumulación del registro MACSR</b>
RxF	<b>Registro que contiene un operando de la MAC a ser escalado</b>
Rw	<b>Registro destino para la MAC con operación de carga</b>
<b>Operaciones en Punto Flotante</b>	
fmt	<b>Formato Operación : Byte (B), Word (W), Longword (L), Single-precision (S), Double-precision(D)</b>
+inf	<b>Infinito Positivo</b>
-inf	<b>Infinito Negativo</b>

FPx, FPy	Destino y fuente, son registros en punto flotante, respectivamente
FPCR	Registro de control en punto flotante
FPIAR	Registro en punto flotante para dirección a instrucción
FPSR	Registro de estado en punto flotante
NAN	No es un número

- Instrucciones de Movimiento de Datos:** Las instrucciones MOVE y FMOVE (mover información en punto flotante) son empleadas para transferir información tipo *byte*, *word* o *longword*, de memoria a memoria, de memoria a registro, de registro a memoria y de registro a registro. La instrucción MOVEA es utilizada para transferir información tipo *word* y *longword* a direcciones comprobadas como válidas. La Tabla 6.3 resume las instrucciones de movimiento de datos.

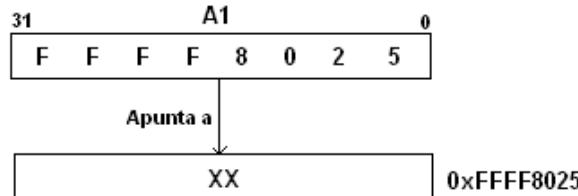
**Tabla 6.3. Instrucciones de movimiento de datos.**

Instrucción	Sintaxis de los Operandos	Tamaño del Operando	Operación	Clasificación
FDMOVE	FPy,FPx	D	Fuente → Destino; redondeo destino a doble	FPU
FMOVE	<ea>y,FPx FPy,<ea>x FPy,FPx FPcr,<ea>x <ea>y,FPcr	B,W,L,S,D B,W,L,S,D D L L	Fuente → Destino FPcr Puede ser un registro de control en punto flotante FPCR, FPIAR, FPSR	FPU
FMOVEM	#list,<ea>x <ea>y,#list	D	Lista de Registros → Destino Fuente → Lista de Registros	FPU
FSMOVE	<ea>y,FPx	B,W,L,S,D	Fuente → Destino; redondeo a simple	FPU
LEA	<ea>y,Ax	L	<ea>y → Ax	ISA_A
LINK	Ay,#<desplazamiento>	W	SP – 4 → SP; Ay → (SP); SP → Ay, SP + d <sub>n</sub> → SP	ISA_A
MOV3Q	#<dato>,<ea>x	L	Dato Inmediato → Destino	ISA_B
MOVCLR	ACCy,Rx	L	Acumulador → Destino ; 0 → Acumulador	EMAC
MOVE	<ea>y,<ea>x MACcr,Dx <ea>y,MACcr	B,W,L L L	Fuente → Destino Donde MACcr puede ser algún registro de control de la MAC como: ACCx, ACCext01, ACCext23, MACSR, MASK	ISA_A MAC MAC
MOVE from CCR MOVE to CCR	CCR,Dx <ea>y,CCR	W W		ISA_A ISA_A
MOVEA	<ea>y,Ax	W,L → L	Fuente → Destino	ISA_A
MOVEM	#lista ,<ea>x <ea>y,#lista	L	Lista de Registros → Destino Fuente → Lista de Registros	ISA_A
MOVEQ	#<dato>,Dx	B → L	Dato Inmediato → Destino	ISA_A
MVS	<ea>y,Dx	B,W	Fuente con extensión de signo → Destino	ISA_B
MVZ	<ea>y,Dx	B,W	Fuente de ceros → Destino	ISA_B
PEA	<ea>y	L	SP – 4 → SP; <ea>y → (SP)	ISA_A
UNLK	Ax	Ninguno	Ax → SP; (SP) → Ax; SP + 4 → SP	ISA_A

Ejemplos de instrucciones de movimiento de datos:

- La instrucción LEA (*Load Effective Address*) carga de manera eficiente una *longword* conteniendo una dirección, hacia un registro Ax (registro de dirección).

LEA 0xFFFF8025 , A1	;El registro de dirección A1 apunta ;a la dirección 0xFFFF8025 (ver ;Figura 6.23)
---------------------	---



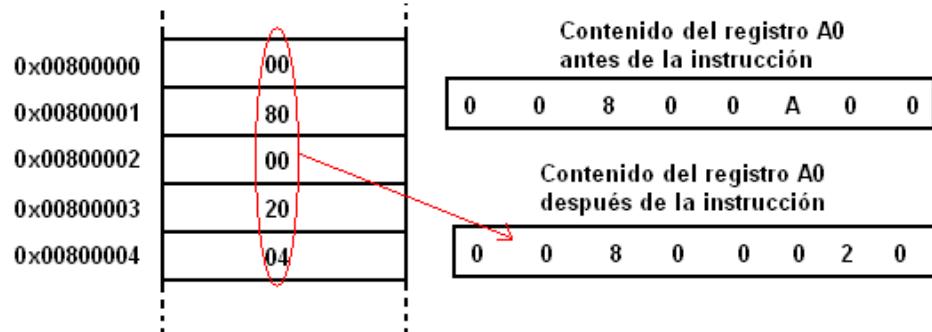
**Figura 6.23. Ejemplo instrucción LEA.**

- La instrucción MOV3Q (*Move 3 bit Quick*) almacena de manera inmediata 3 bits (-1, 1-7) en el destino <ea>x. El valor inmediato se lleva con formato de 32 bit signado.

MOV3Q # -1 , A5	;El registro de dirección A5 es ;cargado con el valor -1 ;(0xFFFFFFFF)
MOV3Q # 2 , D0	;El registro de datos D0 es cargado ;con el valor 2 (0x00000002)

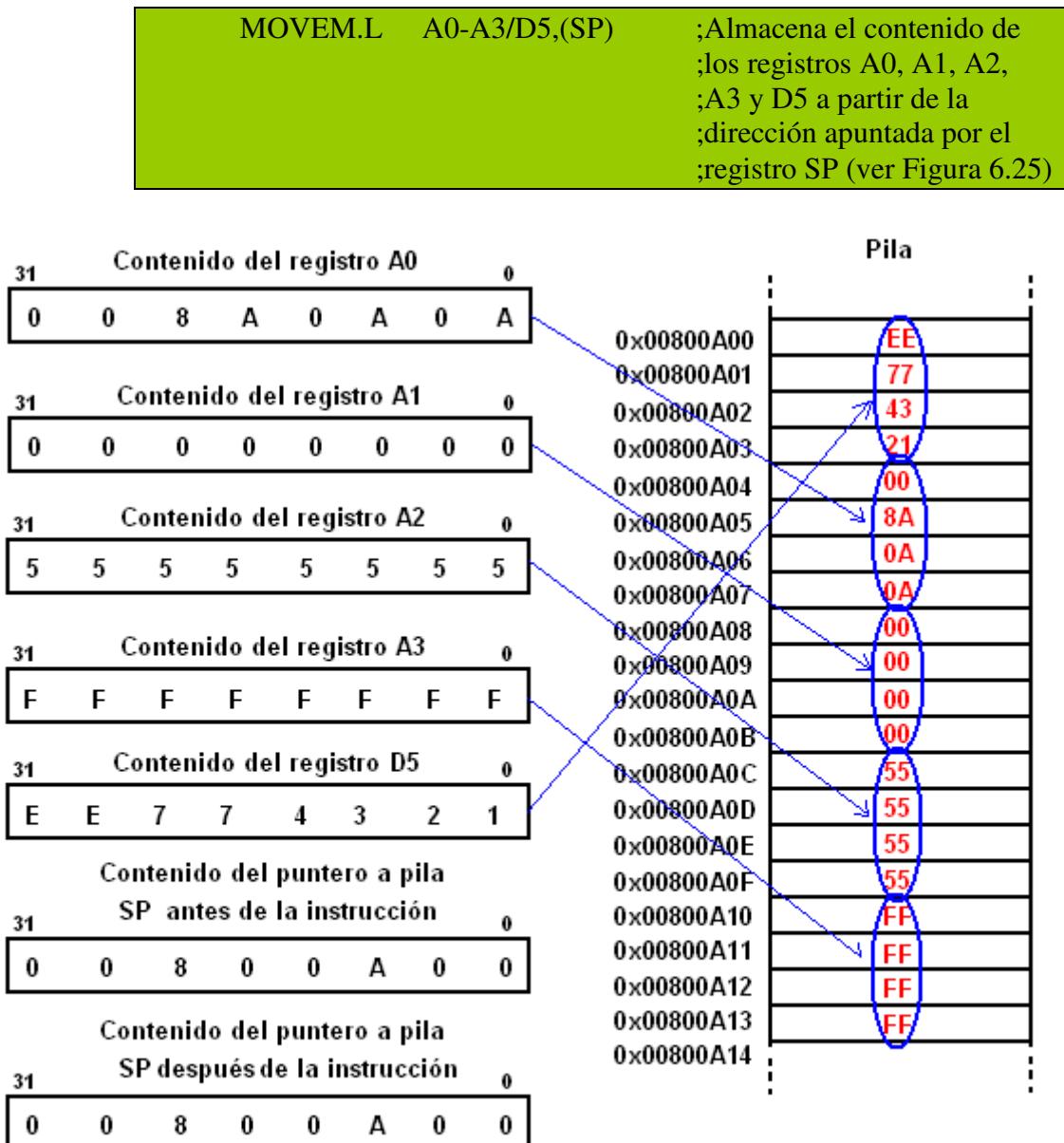
- La instrucción MOVEA (*Move Address from source to destination*) mueve una dirección desde una fuente hacia un registro de dirección como destino.

MOVEA.L 0x00800000 , A0	;El registro de dirección A0 ;es cargado con la dirección ;almacenada en la celda ;00800000 (ver Figura 6.24)
-------------------------	--



**Figura 6.24. Ejemplo instrucción MOVEA.**

- La instrucción MOVEM (*Move Multiple Registers*) mueve el contenido de una lista de registros hacia un destino.



**Figura 6.25. Ejemplo instrucción MOVEM.**

- Instrucciones de Control de Programa:** Los llamados y retornos de las subrutinas y las instrucciones de salto y condicionales, conforman el grupo de las instrucciones de control. La Tabla 6.4 define, rápidamente, las instrucciones de control de la familia ColdFire®.

Las condiciones que se pueden establecer están representadas por las letras **cc** (código de condición) en la tabla y a continuación serán definidas:

<b>CC:</b>	Acarreo aclarado	<b>LS:</b>	Menor o igual
<b>CS:</b>	Acarreo en uno	<b>LT:</b>	Menor que
<b>EQ:</b>	Igual	<b>MI:</b>	Signo negativo
<b>F:</b>	Nunca verdadero <sup>26</sup>	<b>NE:</b>	No igual
<b>GE:</b>	Mayor o igual	<b>PL:</b>	Signo positivo
<b>GT:</b>	Mayor que	<b>T:</b>	Siempre verdadero <sup>27</sup>
<b>HI:</b>	El más alto	<b>VC:</b>	Sobreflujo aclarado
<b>LE:</b>	Menor o igual	<b>VS:</b>	Sobreflujo en uno

**Tabla 6.4. Instrucciones de control de programa.**

Instrucción	Sintaxis del Operando	Tamaño Operando	Operación	Clasificación
<b>Condicionales</b>				
Bcc	<etiqueta>	B, W, L	Si condición verdadera, entonces PC + d <sub>n</sub> → PC	ISA_A
FBcc	<etiqueta>	W, L	Si condición verdadera, entonces PC + d <sub>n</sub> → PC	FPU
Scc	Dx	B	Si condición verdadera, entonces 1s → Destino De lo contrario 0s → Destino	ISA_A
<b>No condicionales</b>				
BRA	<etiqueta>	B, W, L	PC + d <sub>n</sub> → PC	ISA_A
BSR	<etiqueta>	B, W, L	SP - 4 → SP; luego PC → (SP); PC + d <sub>n</sub> → PC	ISA_A
FNOP	ninguno	ninguno	PC + 2 → PC (FPU sincronizado a pipeline)	FPU
JMP	<ea>y	ninguno	Dirección fuente → PC	ISA_A
JSR	<ea>y	ninguno	SP - 4 → SP; nextPC → (SP); Source → PC	ISA_A
NOP	ninguno	ninguno	PC + 2 → PC (Entero sincronizado a pipeline)	ISA_A
TPF	ninguno #<dato> #<dato>	ninguno W L	IPC + 2 → PC PC + 4 → PC PC + 6 → PC	ISA_A
<b>Retornos</b>				ISA_A
RTS	ninguno	ninguno	(SP) → PC; SP + 4 → SP	
<b>Chequeo de bits</b>				
TAS	<ea>x	B	Destino chequeado → CCR; 1 → bit 7 del destino	ISA_B
FTST	<ea>y	B, W, L, S, D	Operando fuente chequeado → FPCC	FPU
TST	<ea>y	B, W, L	Operando fuente chequeado → CCR	ISA_A

Ejemplos de instrucciones de control de programa:

- La instrucción Bcc (*Branch if cc*) evalúa la condición cc y en caso de cumplirse adiciona a la dirección del PC el desplazamiento d<sub>n</sub>, pero si la condición no se cumple, el programa continúa ejecutándose en la dirección del PC.

<sup>26</sup> No aplica para instrucciones del tipo Bcc.

<sup>27</sup> No aplica para instrucciones del tipo Bcc.

BNE	LAZO	;Salte, si no es igual, a la dirección ;con nombre LAZO
-----	------	--

**NOTA:** La dirección LAZO es resuelta por el compilador, como resultado de un desplazamiento en aritmética signada de 16 bits, con posibilidad de extensión a 32.

- La instrucción JMP (*Jump*) ejecuta un salto sin condición a la dirección especificada por la palabra de extensión. A la dirección del PC se le suma el desplazamiento especificado por la palabra de extensión.

JMP	LAZO	;Salte a la dirección LAZO
-----	------	----------------------------

**NOTA:** La dirección LAZO es resuelta por el compilador, como resultado de un desplazamiento en aritmética signada de 16 bits, con posibilidad de extensión a 32.

- La instrucción RTS (*Return of Subroutine*) dispara el mecanismo de retorno de la máquina, estando en un proceso de subrutina. La dirección de la siguiente instrucción a ser ejecutada, después de que la máquina haya retornado de la subrutina, es recuperada desde la pila y llevada al PC. El puntero a pila se incrementa en 4 posiciones (trabajo de una pila LIFO).
- La instrucción TST (*Test*) compara al operando con cero y modifica los bits respectivos en el CCR.

TST.B	(A0)	;Chequee el estado del byte apuntado ;por A0 y lleve el resultado al CCR
-------	------	---

Si la celda apuntada por A0 contiene la cantidad 0xFF, el bit N del CCR se pondrá a “1”.

- **Instrucciones en Aritmética Entera:** Incluyen cinco operaciones básicas: ADD (suma sin acarreo), SUB (substracción sin acarreo), MUL (multiplicación), DIV (división) y REM (residuo). También se incluyen las operaciones de CMP (compare), CLR (aclare) y NEG (niegue como complemento a dos).

La Tabla 6.5 ilustra de manera breve las diferentes operaciones aritméticas en formato entero, que tiene la familia ColdFire®.

**Tabla 6.5. Instrucciones en aritmética entera.**

Instrucción	Sintaxis del operando	Tamaño del operando	Operación	Clasificación
ADD ADDA	Dy,<ea>x <ea>y,Dx <ea>y,Ax	L L L	Fuente + Destino → Destino <dato>	ISA_A
ADDI ADDQ	#<dato>,Dx #<dato>,<ea>x	L L	Dato inmediato + Destino → Destino	ISA_A
ADDX	Dy,Dx	L	Fuente + Destino + CCR[X] → Destino	ISA_A
CLR	<ea>x	B, W, L	0 → Destino	ISA_A
CMP CMPA	<ea>y,Dx <ea>y,Ax	B, W, L W, L	Destino - Fuente → CCR	ISA_A <sup>1</sup>
CMPI	#<dato>,Dx	B, W, L	Destino - Dato inmediato → CCR	ISA_A <sup>1</sup>
DIVS/DIVU	<ea>y,Dx	W, L	Destino/Fuente → Destino (Signado o no signado)	ISA_A
EXT EXTB	Dx Dx Dx	B → W W → L B → L	Destino signado extendido → Destino	ISA_A
MAAAC	Ry, RxSF, ACCx, ACCw	W, L	ACCx + (Ry * Rx){<<1>>} SF → ACCx ACCw + (Ry * Rx){<<1>>} SF → ACCw	ISA_C EMAC_B
MAC	Ry,RxSF,ACCx Ry,RxSF,<ea>y,Rw,ACCx	W, L W, L	ACCx + (Ry * Rx){<<1>>} SF → ACCx ACCx + (Ry * Rx){<<1>>} SF → ACCx; (<ea>y&MASK)) → Rw	ISA_A
MASAC	Ry, RxSF, ACCx, ACCw	W, L	ACCx + (Ry * Rx){<<1>>} SF → ACCx ACCw - (Ry * Rx){<<1>>} SF → ACCw	ISA_C EMAC_B
MSAAC	Ry, RxSF, ACCx, ACCw	W, L	ACCx - (Ry * Rx){<<1>>} SF → ACCx ACCw + (Ry * Rx){<<1>>} SF → ACCw	ISA_C EMAC_B
MSAC	Ry,RxSF,ACCx Ry,RxSF,<ea>y,Rw,ACCx	W, L W, L	ACCx - (Ry * Rx){<<1>>} SF → ACCx ACCx - (Ry * Rx){<<1>>} SF → ACCx; (<ea>y&MASK)) → Rw	ISA_A
MSSAC	Ry, RxSF, ACCx, ACCw	W, L	ACCx - (Ry * Rx){<<1>>} SF → ACCx ACCw - (Ry * Rx){<<1>>} SF → ACCw	ISA_C EMAC_B
MULS/MULU	<ea>y,Dx	W * W → L L * L → L	Fuente x Destino → Destino (Signado o no signado)	ISA_A
NEG	Dx	L	0 - Destino → Destino	ISA_A
NEGX	Dx	L	0 - Destino - CCR[X] → Destino	ISA_A
REMS/REMU	<ea>y,Dw:Dx	L	Destino/Fuente → Residuo (Signado o no signado)	ISA_A
SATS	Dx	L	If CCR[V] == 1; then if Dx[31] == 0; then Dx[31:0] = 0x80000000; else Dx[31:0] = 0xFFFFFFF; else Dx[31:0] no se modifican	ISA_B
SUB SUBA	<ea>y,Dx Dy,<ea>x <ea>y,Ax	L L L	Destino - Fuente → Destino	ISA_A
SUBI SUBQ	#<dato>,Dx #<dato>,<ea>x	L L	Destino - Fuente → Destino	ISA_A
SUBX	Dy,Dx	L	Destino - Fuente - CCR[X] → Destino	ISA_A

Las operaciones de multiplicación y división trabajan sobre los siguientes formatos:

- Multiplicación de palabras, que pueden producir como resultado una *longword*.
- Multiplicación de *longwords*, que producen como resultado una *longword*.

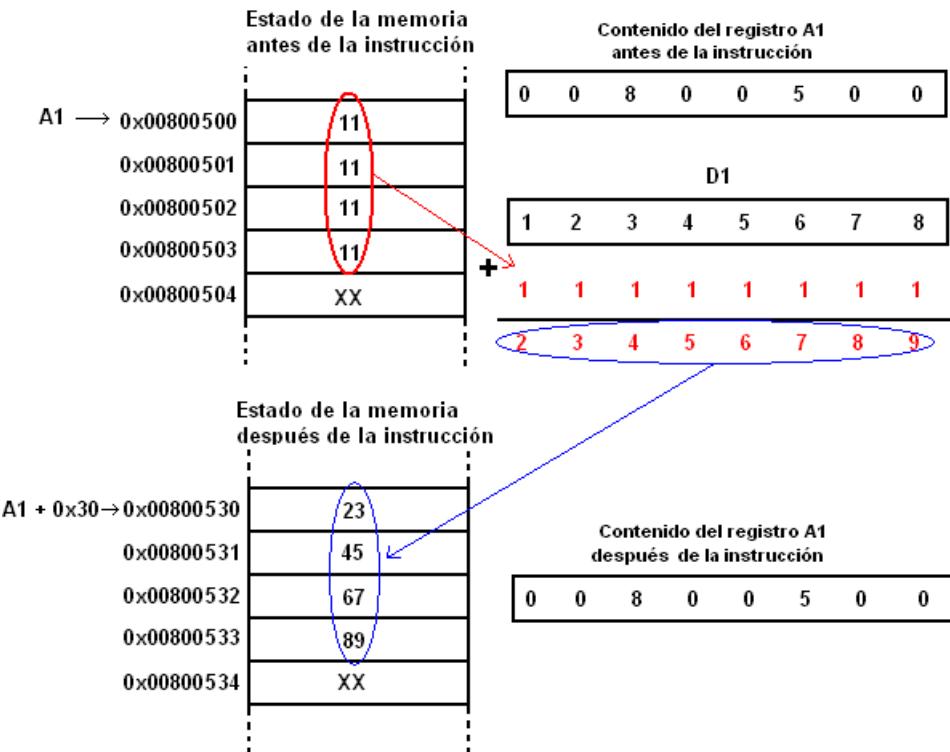
- División de una *longword* por una palabra y que arroja una palabra de cociente y una palabra de residuo.
- División de una *longword* por una *longword* y que arroja una *longword* de cociente.
- División de una *longword* por una *longword* y que arroja una *longword* de residuo.

Para las máquinas dotadas con unidades MAC o EMAC, aparece un juego de instrucciones con manipulación del bit de acarreo extendido CCR[X], útil en operación de precisión múltiple..

Ejemplos de instrucciones aritméticas:

- La instrucción ADD (*Addition*) realiza la suma de dos operandos, donde uno es la fuente y el otro es el destino. El resultado es almacenado en el operando especificado como destino.

ADD.L	D1 , 0x30(A1) ;Sume el contenido del registro D1 ;y el contenido de la longword ;almacenable a partir de la dirección ;especificada en A1 (ver Figura 6.26)
-------	--



**Figura 6.26. Ejemplo instrucción ADD.**

- La instrucción MULU (*Unsigned Multiplication*) realiza el producto entre una fuente y un destino y el resultado es almacenado en el destino.

MULU.W (A1),D1	;Multiplique el contenido de la palabra apuntada por A1, por el contenido de D1 y el resultado se almacena en D1 (ver Figura 6.27)
----------------	--

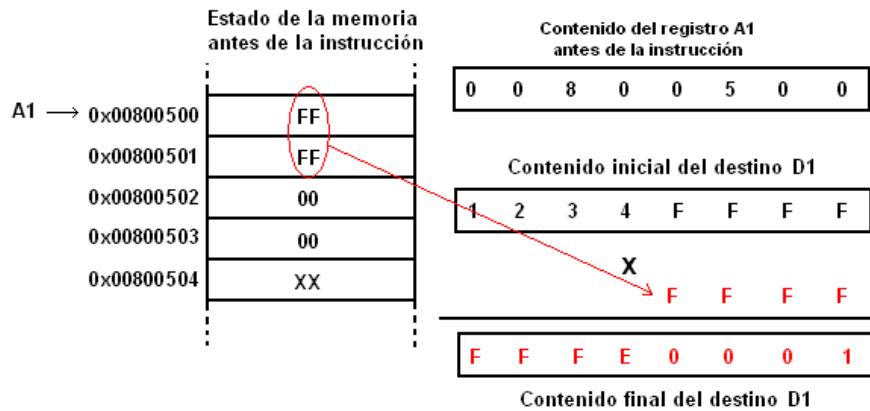


Figura 6.27. Ejemplo instrucción MULU.

- La instrucción SUBX (*Subtract extended*) le substrae al destino la fuente y el bit X (acarreo extendido). Recordar que el bit X tiene sentido en operaciones tipo *longword* usando múltiple precisión.

SUBX.L D1 ,D2	;El contenido de D1 y CCR[X] son restados del contenido de D2 y el y el resultado se almacena en en D2 (ver Figura 6.28)
---------------	--



Figura 6.28. Ejemplo instrucción SUBX.

- **Instrucciones en Punto Flotante:** Están organizadas en dos categorías: *Dyadic* (que requieren dos operandos) y *Monadic* (que requieren un operando).

Las instrucciones *Dyadic* están compuestas por varias funciones como FADD (suma en punto flotante) y FSUB (resta en punto flotante). Para estas instrucciones el primer operando puede estar localizado en memoria como un entero o un número en punto flotante. El segundo operando está localizado en un registro de punto flotante y es a su vez el destino del resultado de la operación.

Todas las operaciones de la FPU (*Floating Point Unit*), con categoría *Dyadic*, soportan todos los formatos de dato y el resultado siempre es redondeado a formato precisión simple o doble. La Tabla 6.6 ilustra el formato general de una instrucción en punto flotante con categoría *Dyadic*.

**Tabla 6.6. Formato instrucciones en punto flotante *Dyadic***

Instrucción	Sintaxis Operando	Tamaño Operando	Operación		Clasificación
F<dop>	<ea>y,FPx FPy,FPx	B, W, L, S, D	FPx <Función>	Fuente → FPx	FPU

El conjunto de instrucciones en punto flotante con categoría *Dyadic*, es ilustrado en la Tabla 6.7.

**Tabla 6.7. Instrucciones en punto flotante *Dyadic***

Instrucción (F<dop>)	Operación
FADD, FSADD, FDADD	Sumar
FCMP	Comparar
FDIV, FSDIV, FDDIV	División
FMUL, FSMUL, FDMUL	Multiplicación
FSUB, FSSUB, FDSUB	Substracción

Las instrucciones *Monadic* están compuestas por varias funciones como FABS (valor absoluto en punto flotante) y sólo requieren de un operando. La operación es realizada sobre el operando fuente y el resultado es almacenado en el registro de punto flotante.

Todas las operaciones de la FPU (*Floating Point Unit*), con categoría *Monadic*, soportan todos los formatos de dato. La Tabla 6.8 ilustra el formato general de una instrucción en punto flotante con categoría *Monadic*.

**Tabla 6.8. Formato instrucciones en punto flotante *Monadic***

Instrucción	Sintaxis del Operando	Tamaño del Operando	Operación	Clasificación
F<mop>	<ea>y,FPx FPy,FPx FPx	B, W, L, S, D	Fuente → <Función> → FPx FPx → <Función> → FPx	FPU

El conjunto de instrucciones en punto flotante con categoría *Monadic*, es ilustrado en la Tabla 6.9.

**Tabla 6.9. Instrucciones en punto flotante *Monadic***

Instrucción (F<mop>)	Operación
FABS, FSABS, FDABS	Valor Absoluto
FINT	Extraer la parte entera
FINTRZ	Extraer parte entera y redondeo a 0
FNEG, FSNEG, FDNEG	Negar
FSQRT, FSSQRT, FDSQRT	Raíz cuadrada

**NOTA:** Ejemplos de instrucciones en punto flotante no aplican para el tipo de máquina ColdFire® V1, debido a la ausencia del módulo FPU.

- **Instrucciones Lógicas:** Se utilizan para ejecutar las instrucciones lógicas básicas (AND, OR, EOR y NOT), sobre datos enteros en formato de *longword*. Sobre datos del tipo inmediato en *longword* aparecen con el nombre de ANDI, ORI, y EORI. La Tabla 6.10 ilustra, de manera resumida, las operaciones lógicas.

**Tabla 6.10. Instrucciones lógicas**

Instrucción	Sintaxis del Operando	Tamaño del Operando	Operación	Clasificación
AND	<ea>y,Dx Dy,<ea>x	L L	Fuente & Destino → Destino	ISA_A
ANDI	#<dato>, Dx	L	Dato inmediato & Destino → Destino	ISA_A
EOR	Dy,<ea>x	L	Fuente ^ Destino → Destino	ISA_A
EORI	#<dato>,Dx	L	Dato inmediato ^ Destino → Destino	ISA_A
NOT	Dx	L	~ Destino → Destino	ISA_A
OR	<ea>y,Dx Dy,<ea>x	L L	Fuente   Destino → Destino	ISA_A
ORI	#<dato>,Dx	L	Dato inmediato   Destino → Destino	ISA_A

Ejemplos de instrucciones lógicas:

- La instrucción ANDI (*And Immediate*) hace la operación AND entre un dato inmediato (constante) y un destino. El resultado de la operación se almacena en el destino.

ANDI.L	#0xFFFF0000 , D2	;Haga la AND entre D2 y ;la constante 0xFFFF0000 ;y almacene el resultado en ;D2 (ver Figura 6.29)
--------	------------------	---



**Figura 6.29. Ejemplo instrucción ANDI.**

- **Instrucciones de Desplazamiento (*Shift*):** Este tipo de instrucciones se pueden ejecutar sólo sobre registros de la máquina y se hacen sobre formato de tipo *longword*.

El contador del número de veces que se hace el desplazamiento aparece en el *opcode* de la instrucción y puede ser un valor entre 1 y 8, pero si el número de veces a desplazar es mayor al rango anterior, será necesario especificarlo en una palabra de extensión y podrá ser un contador cuyo módulo puede ser hasta de 64.

La Tabla 6.11 detalla las instrucciones de desplazamiento.

Ejemplos de instrucciones de desplazamiento:

- La instrucción LSL (*Logical Shift Left*) realiza un desplazamiento lógico a la izquierda de los bits del destino. El número de bits a desplazar se especifica como una constante o como un valor en un registro.

LSL.B	#4 , D0	;Desplace 4 posiciones a la izquierda ;el byte de menor peso de D0 ver ;Figura 6.30
-------	---------	---

**Tabla 6.11. Instrucciones de desplazamiento**

Instrucción	Sintaxis del Operando	Tamaño del Operando	Operación	Clasificación
ASL	Dy,Dx #<dato>,Dx	L L	CCR[X,C] $\leftarrow$ (Dx << Dy) $\leftarrow$ 0 CCR[X,C] $\leftarrow$ (Dx << #<dato>) $\leftarrow$ 0	ISA_A
ASR	Dy,Dx #<dato>,Dx	L L	msb $\rightarrow$ (Dx >> Dy) $\rightarrow$ CCR[X,C] msb $\rightarrow$ (Dx >> #<dato>) $\rightarrow$ CCR[X,C]	ISA_A
LSL	Dy,Dx #<dato>,Dx	L L	CCR[X,C] $\leftarrow$ (Dx << Dy) $\leftarrow$ 0 CCR[X,C] $\leftarrow$ (Dx << #<dato>) $\leftarrow$ 0	ISA_A
LSR	Dy,Dx #<dato>,Dx	L L	0 $\rightarrow$ (Dx >> Dy) $\rightarrow$ CCR[X,C] 0 $\rightarrow$ (Dx >> #<dato>) $\rightarrow$ CCR[X,C]	ISA_A
SWAP	Dx	W	MSW of Dx $\leftrightarrow$ LSW of Dx	ISA_A

**Contenido inicial del destino D1**

X	X	X	X	X	X	0	F
---	---	---	---	---	---	---	---

X	X	X	X	X	X	F	0
---	---	---	---	---	---	---	---

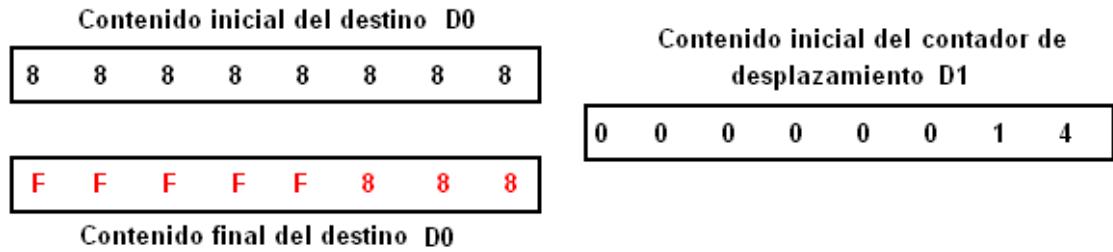
**Contenido final del destino D1**

X: Indica que los valores no aparecen debido al tipo de dato byte en D1

**Figura 6.30. Ejemplo instrucción LSL.**

- La instrucción ASR (*Arithmetic Shift Right*) realiza un desplazamiento aritmético (respeta el bit de signo del destino) a la derecha de los bits del destino. El número de bits a desplazar se especifica como una constante o como un valor en un registro.

ASR.L	D1 , D0	;Desplace el número de posiciones ;especificada por D1, a la derecha, ;el registro D0 (ver Figura 6.31)
-------	---------	---



**Figura 6.31. Ejemplo instrucción ASR.**

- **Instrucciones para Manipulación de Bits:** Pueden ser aplicadas a los registros (en formato de 32 bits) o a la memoria (en formato de 8 bits). El número del bit a manipular se especifica como un dato inmediato o contenido dentro de un registro de datos. La Tabla 6.12 resume las instrucciones de bit.

**Tabla 6.12. Instrucciones de bit**

Instrucción	Sintaxis del Operando	Tamaño del Operando	Operación	Clasificación
BCHG	Dy,<ea>x #<dato>,<ea>x	B, L B, L	$\sim(<\text{número bit}> \text{ del Destino}) \rightarrow \text{CCR}[Z] \rightarrow <\text{número bit}> \text{ del Destino}$	ISA_A
BCLR	Dy,<ea>x #<dato>,<ea>x	B, L B, L	$\sim(<\text{número bit}> \text{ del Destino}) \rightarrow \text{CCR}[Z]; 0 \rightarrow <\text{número bit}> \text{ del Destino}$	ISA_A
BITREV	Dx	L	Reversión de bits Dx → Dx	ISA_A+, ISA_C
BSET	Dy,<ea>x #<dato>,<ea>x	B, L B, L	$\sim(<\text{número bit}> \text{ del Destino}) \rightarrow \text{CCR}[Z]; 1 \rightarrow <\text{número bit}> \text{ del Destino}$	ISA_A
BYTEREV	Dx	L	Reversión de bytes Dx → Dx	ISA_A+, ISA_C
BTST	Dy,<ea>x #<dato>,<ea>x	B, L B, L	$\sim(<\text{número bit}> \text{ del Destino}) \rightarrow \text{CCR}[Z]$	ISA_A
FF1	Dx	L	Encontrar el primer bit en 1 de Dx → Dx	ISA_A+, ISA_C

Ejemplos de instrucciones de desplazamiento:

- La instrucción BITREV (*Bit Reverse*) invierte el orden de los bits en un registro.

BITREV	D0	;Invierte la posición de los bits de D0 ;ver Figura 6.32
--------	----	---



Figura 6.32. Ejemplo instrucción BITREV.

- La instrucción BSET (*Bit Set*) pone en “1” el bit especificado del destino.

BSET.B	#3 , (A0)	;Ponga a “1” el bit 3 del registro tipo ;byte apuntado por A0 (ver Figura ;6.33)
--------	-----------	--

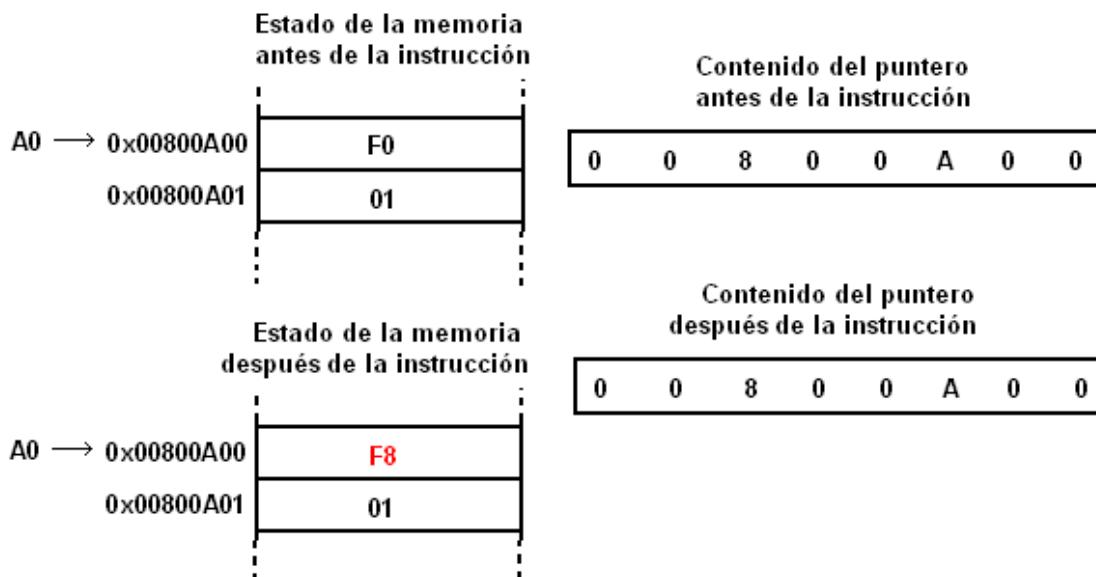


Figura 6.33. Ejemplo instrucción BSET.

- La instrucción FF1 (*Find First 1*) encuentra el primer uno en un registro, a partir del bit de mayor peso. El número del bit se escribe en el mismo registro de búsqueda y va desde el bit 0 al bit 31, comenzando desde el LSb.

FF1.L	D7	;Encuentre el primer 1 de D7 (ver ;Figura 6.34)
-------	----	--

- **Instrucciones de Control del Sistema:** Incluyen instrucciones privilegiadas y de *Trap*. La Tabla 6.13 muestra un resumen de las instrucciones de control del sistema.



**Figura 6.34. Ejemplo instrucción FF1.**

**Tabla 6.13. Instrucciones de control del sistema**

Instrucción	Sintaxis del Operando	Tamaño del Operando	Operación	Clasificación
<b>Privilegiadas</b>				
FRESTORE	<ea>y	ninguno	Trama de estado FPU → Estado interno FPU	FPU
FSAVE	<ea>x	ninguno	Estado interno FPU → Trama de estado FPU	FPU
HALT	ninguno	ninguno	HALT al núcleo de la CPU	ISA_A
MOVE from SR	SR,Dx	W	SR → Destino	ISA_A
MOVE from USP	USP,Dx	L	USP → Destino	ISA_B
MOVE to SR	<ea>y,SR	W	Fuente → SR; únicamente Dy o #<dato> pueden ser la fuente	ISA_A
MOVE to USP	Ay,USP	L	Fuente → USP	ISA_B
MOVEC	Ry,Rc	L	Ry → Rc	ISA_A
RTE	ninguno	ninguno	2 (SP) → SR; 4 (SP) → PC; SP + 8 → SP Ajusta la pila de acuerdo al formato	ISA_A
STOP	#<dato>	ninguno	Dato inmediato → SR; STOP	ISA_A
STLDSR	#<dato>	W	SP - 4 → SP; Llena de ceros a SR → (SP); Dato inmediato → SR	ISA_A+, ISA_C
WDEBUG	<ea>y	L	Direccionamiento del depurador, se ha ejecutado una instrucción WDMREG	ISA_A
<b>Funciones del Depurador</b>				
PULSE	ninguno	ninguno	Hace el PST = 0x4	ISA_A
WDDATA	<ea>y	B, W, L	Fuente → Puerto DDATA	ISA_A
<b>Generación de Traps</b>				
ILLEGAL	ninguno	ninguno	SP - 4 → SP; PC → (SP) → PC; SP - 2 → SP; SR → (SP); SP - 2 → SP; Offset del vector → (SP); (VBR + 0x10) → PC	ISA_A
TRAP	#<vector>	ninguno	1 → S Bit de SR; SP - 4 → SP; próximo PC → (SP); SP - 2 → SP; SR → (SP) SP - 2 → SP; Formato/Offset → (SP) (VBR + 0x80 + 4*n) → PC, donde n es el # del Trap	ISA_A

### 6.3. EJERCICIOS CON DIRECCIONAMIENTOS E INSTRUCCIONES

En este aparte serán ejecutados algunos ejercicios en donde aparecen diferentes tipos de direccionamiento e instrucciones. Los ejercicios se desarrollan en el lenguaje *assembler* para las máquinas ColdFire® V1, con el objetivo de ilustrar el comportamiento a nivel nuclear de éstas máquinas.

#### Ejercicio 1

**Direccionamientos involucrados:** Directo a registro, indirecto, manipulación de bits, control de programa, absoluto corto y de dato inmediato.

**Planteamiento del problema:** Encontrar en una tabla de 10 datos, almacenados en RAM, la ubicación y el número de datos (tipo byte) que tienen valor cero. Los datos se encuentran almacenados a partir de la dirección 0x00800500 y la ubicación de aquellos que sean cero será almacenada a partir de la dirección 0x0080080C. Al finalizar, el sistema debe enterarse del cumplimiento de la tarea y la rutina debe quedar preparada para otra búsqueda.

**Diseño de la solución:** El algoritmo en diagramas de flujo de la Figura 6.35 desarrolla la rutina (Hallar\_ceros) para resolver el ejercicio planteado.

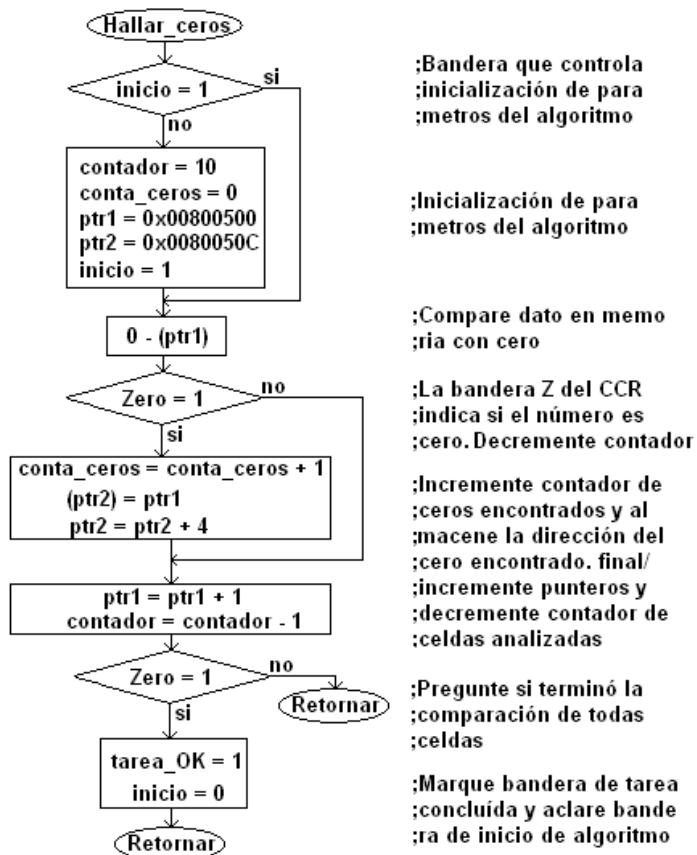


Figura 6.35. Diagrama de flujo búsqueda de ceros en tabla.

**Codificación en lenguaje assembler:** Las siguientes instrucciones en lenguaje *assembler* resuelven el algoritmo de la Figura 6.35. El programa ha sido probado sobre la plataforma CodeWarrior® 6.2, que será explicada en el Capítulo 8, y fue embedido sobre un cuerpo de lenguaje C.

Algunos registros de dato y dirección de la MCU son utilizados y se les hace la siguiente asignación:

**A0:** Servirá como un puntero de propósito general.

**A1:** Asignado como puntero a tabla de 10 datos (ptr1).

**A2:** Asignado como puntero a memoria de almacenamiento de la ubicación del 0 encontrado (ptr2).

**D0:** Registro de datos de propósito general.

**BANDERAS:** Registro de banderas de usuario, para informar de eventos acontecidos en el desarrollo del algoritmo.

**inicio:** Bit que indica que los parámetros de la rutina ya fueron inicializados.

**tarea OK:** Bit que indica el cumplimiento de la tarea de búsqueda.

Desde el punto de vista del tipo de direccionamiento y el tipo de instrucción empleada, los comentarios a la lista de programa son:

```

MOVE.L #BANDERAS,A0 //Direccionamiento inmediato e instruccion de
                     //movimiento de datos
BSET #inicio,(A0) //Direccionamiento indirecto e instruccion de
                     //manipulacion de bits

SIGA:
CLR.L D0           //Direccionamiento directo a registro de datos e
                     //instruccion aritmetica
CMP.B (A1),D0      //Direccionamiento indirecto e instruccion
                     //aritmetica
BNE OTRO          //Direccionamiento absoluto corto e instruccion
                     //de control de programa

CERO:
MOVE.L #CONTA_CEROS,A0 //Direccionamiento inmediato e instruccion de
                     //movimiento de datos
ADD.L #1,(A0)        //Direccionamiento indirecto e instruccion aritmetica
MOVE.L A1,(A2)        //Direccionamiento indirecto e instruccion de
                     //movimiento de datos
ADD.L #4,A2          //Direccionamiento inmediato e instruccion
                     //aritmetica

OTRO:
ADD.L #1,A1          //Direccionamiento inmediato e instruccion
                     //aritmetica
MOVE.L #CONTADOR,A0 //Direccionamiento inmediato e instruccion de
                     //movimiento de datos
SUB.L #1,(A0)        //Direccionamiento indirecto e instruccion aritmetica
BEQ FIN             //Direccionamiento absoluto corto e instruccion
                     //de control de programa
RTS                 //Direccionamiento absoluto corto e instruccion de
                     //control de programa

FIN:
MOVE.L #BANDERAS,A0 //Direccionamiento inmediato e instruccion de
                     //movimiento de datos
BSET #tarea_OK,(A0) //Direccionamiento indirecto e instruccion de
                     //manipulacion de bits
BCLR #inicio,(A0)    //Direccionamiento indirecto e instruccion de
                     //manipulacion de bits
RTS                 //Direccionamiento absoluto corto e instruccion de
                     //control de programa
)

```

Desde el punto de vista de la lógica de programa, los comentarios son los siguientes:

```

//****************************************************************************
/*                      RUTINA PARA HALLAR CEROS                  */
//****************************************************************************
asm(
  HALLAR_CEROS:
    MOVE.L #BANDERAS,A0 //A0 apunta al registro de banderas de usuario
    BTST #inicio,(A0) //La rutina ya inicio?
    BNE SIGA           //Si. Salte a SIGA
    MOVE.L #CONTADOR,A0 //A0 apunta a registro de conteo de celdas a barrer
    MOVE.L #10,(A0)    //Almacene 10 eventos en el contador
)

```

```

MOVE.L #CONTA_CEROS,A0 //A0 apunta a registro de conteo de ceros encontrados
CLR.L (A0) //Aclare contador de numero de ceros encontrados
MOVE.L #0x00800500,A1 //Inicialice ptr1 en la direccion 0x00800500
MOVE.L #0x0080050C,A2 //Inicialice ptr2 en la direccion 0x0080050C
MOVE.L #BANDERAS,A0 //A0 apunta al registro de banderas de usuario
BSET #inicio,(A0) //Ponga en “1” la bandera de inicio

SIGA:
CLR.L D0 //Aclare registro de datos D0
CMP.B (A1),D0 //Compare con dato almacenado en memoria
BNE OTRO //Si no es cero salte a OTRO

CERO:
MOVE.L #CONTA_CEROS,A0 //A0 apunta a contador de ceros
ADD.L #1,(A0) //Incremente en 1 el contador
MOVE.L A1,(A2) //Almacene la direccion del dato encontrado en cero
ADD.L #4,A2 //Incremente direccion del ptr2

OTRO:
ADD.L #1,A1 //Incremente direccion del ptr1
MOVE.L #CONTADOR,A0 //A0 apunta a contador de iteraciones
SUB.L #1,(A0) //Decrementa contador de iteraciones
BEQ FIN //Si es cero, salte a FIN
RTS //Retorne de subrutina

FIN:
MOVE.L #BANDERAS,A0 // A0 apunta al registro de banderas de usuario
BSET #tarea_OK,(A0) //Ponga en “1” bandera de tarea finalizada
BCLR #inicio,(A0) //Aclare bandera de inicio
RTS //Retorne de subrutina
)

```

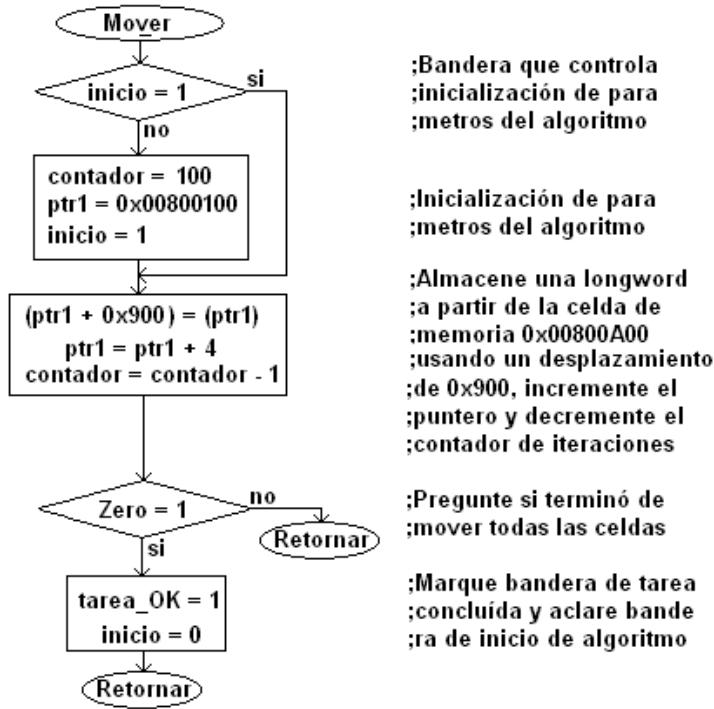
## Ejercicio 2

**Direccionamientos involucrados:** Algunos del ejercicio 1 más indirecto con desplazamiento e indirecto con postincremento.

**Planteamiento del problema:** Trasladar la información contenida en una tabla de 400 datos (tipo byte), con dirección base = 0x00800100, a una tabla con dirección 0x00800A00.

**Diseño de la solución:** El algoritmo en diagramas de flujo de la Figura 6.36 desarrolla la rutina (Mover) para resolver el ejercicio planteado.

**Codificación en lenguaje assembler:** Las siguientes instrucciones en lenguaje *assembler* resuelven el algoritmo de la Figura 6.36. El programa ha sido probado sobre la plataforma CodeWarrior® 6.1, que será explicada en el Capítulo 8, y fue embebido sobre un cuerpo de lenguaje C.



**Figura 6.36. Diagrama de flujo movimiento de datos.**

Algunos registros de dirección de la MCU son utilizados y se les hace la siguiente asignación:

**A1:** Servirá como un puntero de propósito general.

**A0:** Asignado como puntero a tablas de 400 datos a mover (ptr1).

**BANDERAS:** Registro de banderas de usuario, para informar de eventos acontecidos en el desarrollo del algoritmo.

**inicio:** Bit que indica que los parámetros de la rutina ya fueron inicializados.

**tarea\_OK:** Bit que indica el cumplimiento de la tarea de búsqueda.

Desde el punto de vista del tipo de direccionamiento y el tipo de instrucción empleada, los comentarios a la lista de programa son:

```

/*
/* En la definicion e inicializacion de variables se deben incluir: */
*/
byte BANDERAS=0; //Bandera de propósito general de usuario
unsigned long CONTADOR; //Contador para iterar sobre 400 celdas
#define inicio 0 //Bandera de inicio de la rutina de búsqueda
#define tarea_OK 1 //Bandera de tarea cumplida

/*
/* RUTINA PARA MOVER 400 BYTES */

```

```

/*****************/
asm(
    MOVER:
        MOVE.L #BANDERAS,A1          //Direccionamiento inmediato e instruccion de
                                    //movimiento de datos
        BTST   #inicio,(A1)          //Direccionamiento indirecto e instruccion de
                                    //manipulacion de bits
        BNE    SIGA                 //Direccionamiento absoluto corto e instruccion
                                    //de control de programa
        MOVE.L #CONTADOR,A1          //Direccionamiento inmediato e instruccion de
                                    //movimiento de datos
        MOVE.L #100,(A1)             //Direccionamiento indirecto e instruccion de
                                    //aritmetica
        MOVE.L #0x00800100,A0          //Direccionamiento inmediato e instruccion de
                                    //movimiento de datos
        MOVE.L #BANDERAS,A0          //Direccionamiento inmediato e instruccion de
                                    //movimiento de datos
        BSET   #inicio,(A0)          //Direccionamiento indirecto e instruccion de
                                    //manipulacion de bits

    SIGA:
        MOVE.L (A0)+,0x8FC(A0)       //Direccionamiento indirecto con postincremento y
                                    //desplazamiento signado en 16 bits
        MOVE.L #CONTADOR,A1          //Direccionamiento inmediato e instruccion de
                                    //movimiento de datos
        SUB.L #1,(A1)                //Direccionamiento indirecto e instruccion aritmetica
        BEQ    FIN                  //Direccionamiento absoluto corto e instruccion
                                    //de control de programa
        RTS                           //Direccionamiento absoluto corto e instruccion de
                                    //control de programa

    FIN:
        MOVE.L #BANDERAS,A1          //Direccionamiento inmediato e instruccion de
                                    //movimiento de datos
        BSET   #tarea_OK,(A1)         //Direccionamiento indirecto e instruccion de
                                    //manipulacion de bits
        BCLR   #inicio,(A1)          //Direccionamiento indirecto e instruccion de
                                    //manipulacion de bits
        RTS                           //Direccionamiento absoluto corto e instruccion de
                                    //control de programa
)

```

Desde el punto de vista de la lógica de programa, los comentarios son los siguientes:

```

/*****************/
/*                      RUTINA PARA MOVER 400 BYTES                      */
/*****************/
asm(
    MOVER:
        MOVE.L #BANDERAS,A1          //Apunte a registro de banderas de usuario
        BTST   #inicio,(A1)          //Se iniciaron los parametros de la rutina?
        BNE    SIGA                 //Si, salte a iterar
        MOVE.L #CONTADOR,A1          //Apunte a contador de iteraciones
        MOVE.L #100,(A1)             //Inicialice contador en 100, porque se moveran de
                                    //a 4 bytes (longword)
        MOVE.L #0x00800100,A0          //A0 inicia apuntando a la primera longword a mover
)

```

SIGA:	MOVE.L #BANDERAS,A0 BSET # <a href="#">inicio</a> ,(A0)	//Apunte a registro de banderas de usuario //Marque evento de inicio de parametros
	MOVE.L (A0)+,0x8FC(A0)	//Mueva una longword hacia el destino desplazado //0x8FC posiciones, debido al postincremento que //se hace sobre el puntero A0
	MOVE.L #CONTADOR,A1 SUB.L #1,(A1) BEQ FIN RTS	//Apunte a la variable contador //Decremente el contador de iteraciones //Si es cero, salte a FIN //Retorne de subrutina
FIN:	MOVE.L #BANDERAS,A1 BSET # <a href="#">tarea_OK</a> ,(A1) BCLR # <a href="#">inicio</a> ,(A1) RTS	//Apunte a registro de banderas de usuario //Ponga en “1” la bandera de tarea concluida //Aclare bandera local de inicio //Retorne de subrutina
)		

### Ejercicio 3

**Direccionamientos involucrados:** Algunos del ejercicio 2 más indirecto con índice escalado y desplazamiento en 8 bits.

**Planteamiento del problema:** Las hojas de datos de las 151 máquinas, de una planta de inyección de plásticos, contienen los siguientes ítems:

- Número de identificación (ID1): Representado en 10 bytes.
- Nombre de la máquina (NOMBRE): Representado en 32 bytes.
- Lugar de instalación (DIR): Representada en 32 bytes.
- Meses de operación (VIDA): Representado en 4 bytes.
- Nombre del operario (OPE): Representado en 32 bytes.
- Frecuencia de mantenimiento (FREC): Representado en 4 bytes.
- Número de unidades que produce en promedio al mes (UPROM): Representado en 8 bytes.

Para un total de 122 bytes, almacenados en memoria no volátil (FLASH) de un sistema MCU.

Se requiere diseñar una rutina de consulta del número de unidades promedio de producción (UPROM) de alguna de las 151 máquinas. Los datos extraídos de la tabla se deberán almacenar en la RAM, para su posterior análisis.

**Diseño de la solución:** La tabla de almacenamiento de la información es como se muestra en la Figura 6.37 y el algoritmo en diagramas de flujo de la Figura 6.38 desarrolla la rutina (Consulta), para resolver el ejercicio planteado.

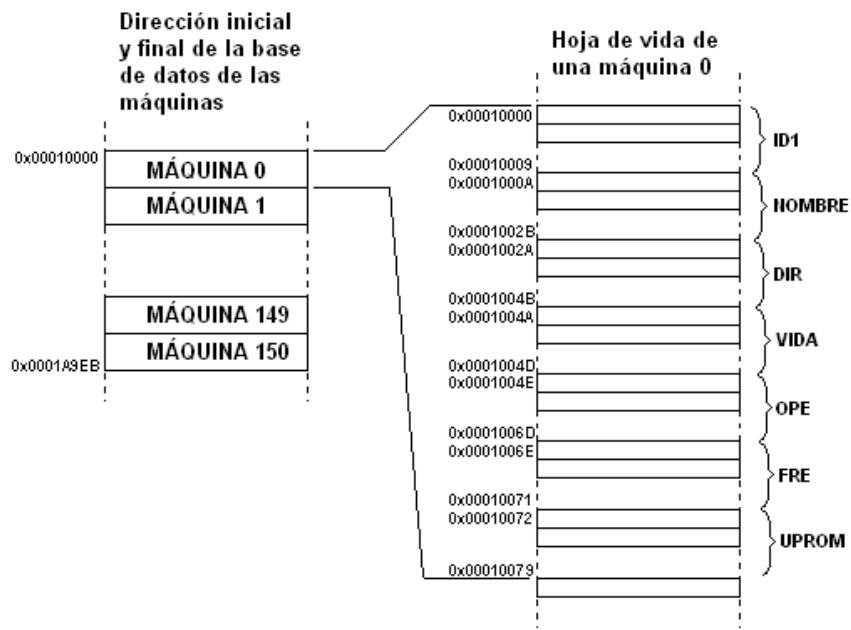


Figura 6.37. Distribución datos en memoria.

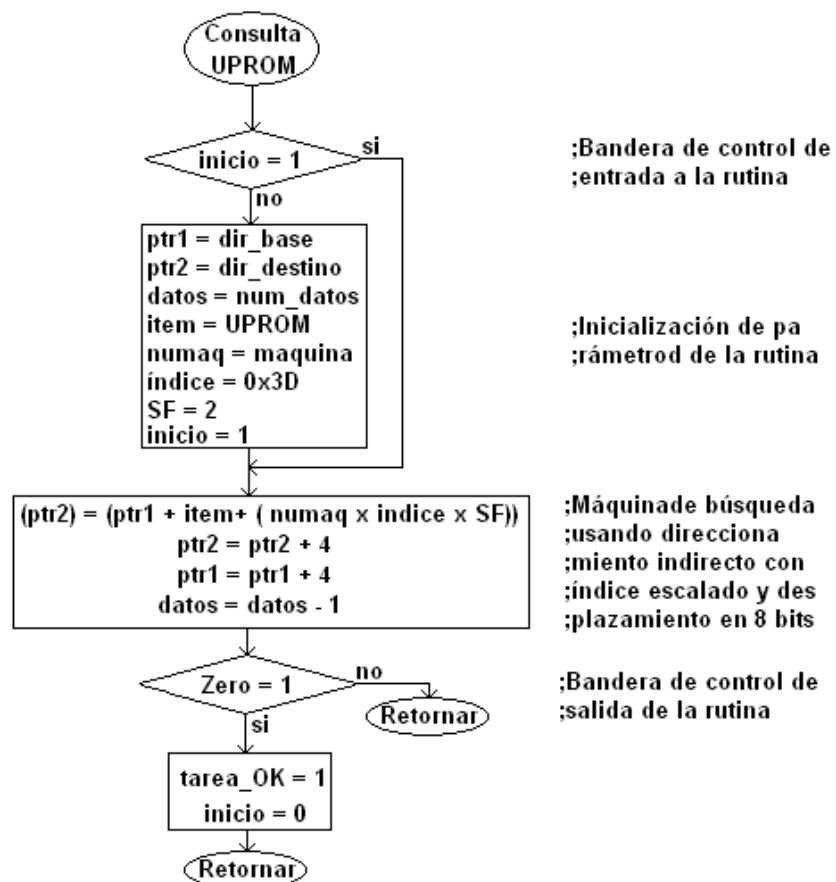


Figura 6.38. Diagrama de flujo consulta.

**Codificación en lenguaje assembler:** La sencillez del algoritmo de búsqueda de datos, en la tabla de información de las máquinas, radica en la potencia del direccionamiento indirecto con índice escalado y desplazamiento en 8 bits.

Las siguientes instrucciones en lenguaje *assembler* resuelven el algoritmo de la Figura 6.38. El programa ha sido probado sobre la plataforma CodeWarrior® 6.1, que será explicada en el Capítulo 8, y fue embebido sobre un cuerpo de lenguaje C.

Algunos registros de dirección de la MCU son utilizados y se les hace la siguiente asignación:

**A3:** Puntero de propósito general.

**A0:** Servirá como un puntero a base de la información (**ptr1**).

**A1:** Asignado como puntero a destino de la información de consulta (**ptr2**).

**D1:** Puntero índice, utilizado para el cálculo del ítem a ser consultado (**índice**)

**numaq:** Constante para ajustar el ítem a consultar.

**SF:** Factor de escala en el ajuste del índice.

El producto de **D1** y el escalar **SF = 2** deberá arrojar como resultado la cantidad 122, que corresponde al desplazamiento por lista completa de datos de cada máquina. Entonces **D1** deberá ser igual a la cantidad 0x3D, llamada **índice** en el ejemplo.

```
/****************************************/
/* En la definicion e inicializacion de variables se deben incluir:          */
/****************************************/
byte BANDERAS=0;                                //Banderas de propósito general de usuario
unsigned int datos=0;                            //Contador para iterar el numero de celdas a mover
unsigned int numaq=0;                            //Variable almacen del numero de maquina
#define inicio 0                         //Bandera de inicio de la rutina de busqueda
#define tarea_OK 1                        //Bandera de tarea cumplida
#define basemaq 0x00010000                //Direccion base de las hojas de vida de las maquinas
#define almacen 0x00800A00                //Lugar a donde se traslada la consulta
#define UPROM 0x72                         //Offset de la informacion a consultar
#define indice 0x3D                        //Direccion base del indice

/****************************************/
/*          RUTINA PARA CONSULTAR EL DATO UPROM           */
/****************************************/
asm(
    MOVER:
        MOVE.L #BANDERAS,A3           //Apunte a registro de bandera de usuario
        BTST   #inicio,(A3)           //Pregunte si ya inicializo parametros
        BNE    SIGA                  //Si, salte
        MOVE.L #datos,A3             //Apunte a registro del numero de datos a barrer
        MOVE.L #2,(A3)                //Almacene 2, que son los datos (en formato long) del
                                       //ítem UPROM
        MOVE.L #numaq,A3              //Apunte a registro contenido el numero de la
                                       //maquina a consultar
        MOVE.L #3,(A3)
        MOVE.L #0x0000003D,D1         //Cargar el ajuste al indice
        MULU.W (A3),D1               //Prepare el indice, segun la maquina a consultar
        MOVE.L #0x00010000,A0         //Inicialice puntero a base de datos
)
```

```

MOVE.L #0x00800A00,A1      //Inicialice puntero destino de la consulta
MOVE.L #BANDERAS,A3        //Apunte a registro de banderas de usuario
BSET    #inicio,(A3)        //Ponga en “1” bandera de inicio

SIGA:
MOVE.L UPROM(A0,D1*2),(A1)          //Direccionamiento indirecto a indice escalado
                                    //((SF=2) con desplazamiento (UPROM), aqui
                                    //se hace la consulta del item UPROM y se al
                                    //maceran 8 bytes a partir de la direccion
                                    //apuntada por A1
ADD     #4,A0                //Incrementa puntero base (en un long)
ADD     #4,A1                //Incrementa puntero a destino consulta (en un long)
MOVE.L #datos,A3            //Apunte a registro de datos a mover por consulta
SUB.L   #1,(A3)              //Decrementa numero de datos a mover
BEQ    FIN                  //Si es cero, salte a FIN
RTS
FIN:
MOVE.L #BANDERAS,A3        //Apunte a registro banderas de usuario
BSET    #tarea_OK,(A3)       //Ponga en “1” bandera de consulta terminada
BCLR    #inicio,(A3)        //Aclare bandera de inicio de parametros
RTS
)

```

## 6.4. REFERENCIAS

- ColdFire® Family, Programmer’s Reference Manual. Rev 3. 03/2005. Freescale.
- MCF51JM128 Integrated Microcontroller Reference Manual. Rev 1. 01/2008. Freescale Semiconductor.

## 6.5. PREGUNTAS

- ¿De hasta cuántos bytes puede estar conformada una instrucción para las máquinas ColdFire® V1?
- Describa los campos que componen una línea de instrucción en *assembler*.
- ¿Qué tipo de direccionamiento utilizan las siguientes instrucciones y qué hace cada instrucción?, para la máquina ColdFire® V1:
  - MOVE.L #0x3DF8042C , A3
  - ADD D1 , D0
  - TST.B 0x800100
  - MOVE.L 0xFE(A0,D1\*4)
  - JMP LISTA
  - BNE LISTA
  - MOVE.L 0x8FC(A0) , (A0)+
  - NOP

Para la realización de este ejercicio se recomienda la consulta del documento, que se encuentra en el sitio WEB del texto: ColdFire® Family, Programmer’s Reference Manual. Rev 3. 03/2005. Freescale.

- Hacer un programa en lenguaje *assembler*, que organice una tabla de 100 números (en formato *long*). de menor a mayor. Los números se encuentran almacenados a partir de la dirección 0x800400. Recuerde no iterar dentro del algoritmo.

# CAPÍTULO 7

## Migración de 8 a 32 bits

**7.1. Una breve descripción de la familia de 8 bits HCS08**

**7.2. Concepto FLEXIS**

**7.3. Cuidados en la migración**

**7.4. Referencias**

**7.5. Preguntas**

## 7.1. UNA BREVE DESCRIPCIÓN DE LA FAMILIA DE 8BITS HCS08

Esta familia de microcontroladores de alto desempeño y bajo costo utiliza la arquitectura ampliada con núcleo HCS08, velocidad de reloj de bus de hasta 24 MHz y una amplia variedad de periféricos, tan útiles como el USB.

- **Modelo de Programación**

Compatible con las familias anteriores en 8 bits, el modelo de programación conserva la estructura de las primeras familias de microcontroladores de 8 bits. La Figura 7.1 ilustra las componentes del modelo de programación de las máquinas HCS08.

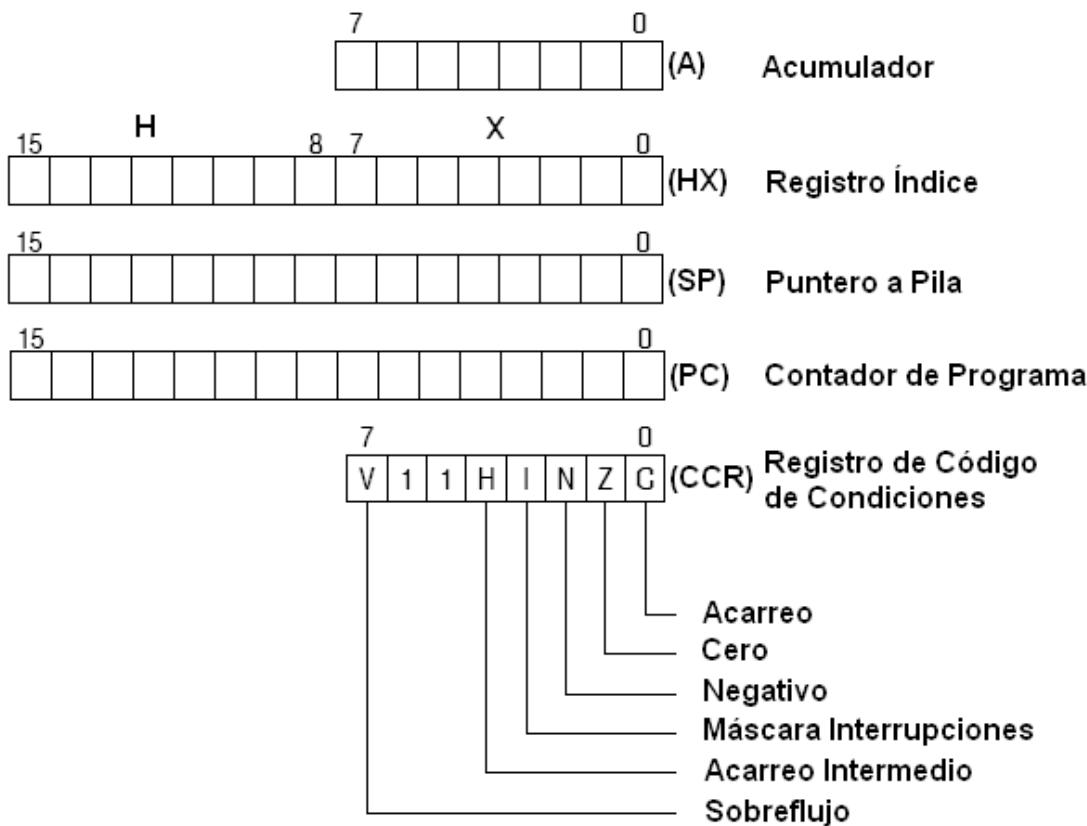


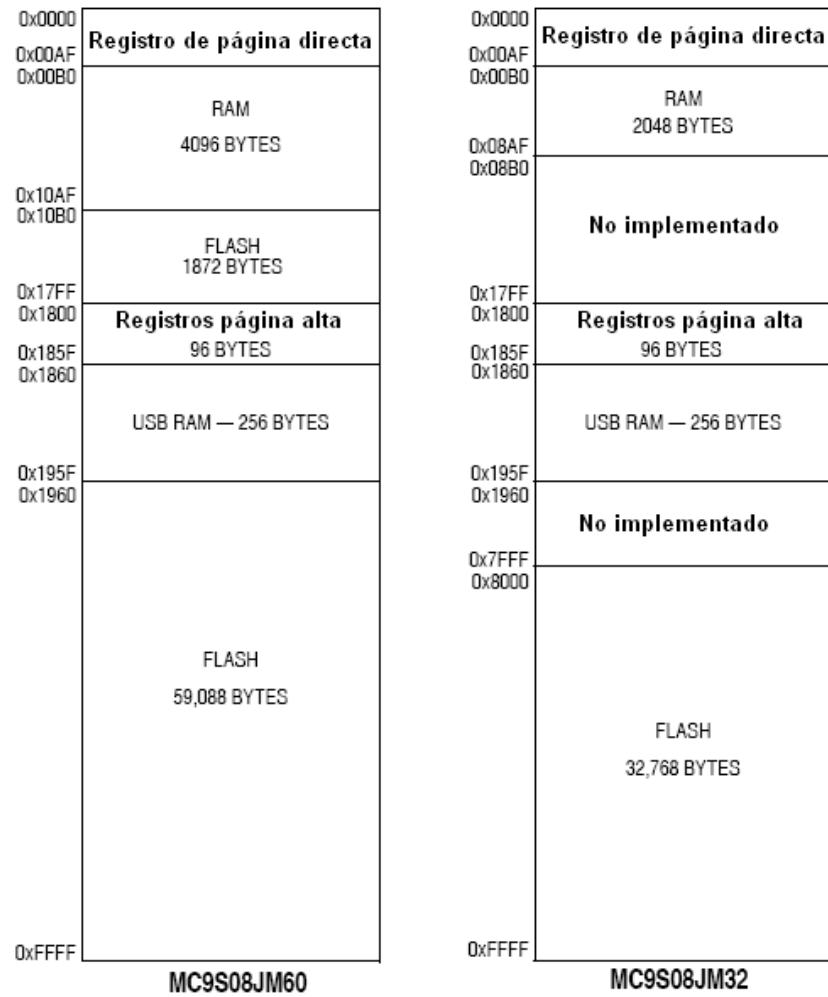
Figura 7.1. Modelo de programación familia HCS08.

- **Modos de Direccionamiento**

Fueron diseñados para acceder un espacio lineal de 64 KB de memoria y a continuación se definen brevemente.

- **Modo Inherente:** No existen operandos en la instrucción, implícito a la instrucción se encuentra el operando.
- **Modo Relativo:** Trata sobre las instrucciones de salto, condicionales y no condicionales, en donde se utiliza un desplazamiento en aritmética signada de 8 bits.

- **Modo Inmediato:** Un operando en 16 bits es seguido a continuación del código de operación, la parte alta del operando se localiza en la parte baja de la memoria y la parte baja se localiza en la siguiente superior (método *big endian*).
- **Modo Directo:** Se refiere al direccionamiento que se realiza en la página cero de la memoria (0x0000 – 0x00FF), es el más rápido y eficiente.
- **Modo Extendido:** La dirección plena en 16 bits del operando es localizada en los siguientes dos *bytes* de la memoria de programa, a continuación del código de operación.
- **Modo de direccionamiento indexado:** Utiliza el registro HX como puntero indirecto a memoria. Este tipo de direccionamiento se subdivide en 7 modos, a saber:
  - **Indexado sin desplazamiento:** El registro HX apunta al operando necesario para completar la instrucción.
  - **Indexado sin desplazamiento y postincremento:** El registro HX apunta al operando necesario para completar la instrucción. Al terminarse la ejecución, la dirección del puntero (HX) se incrementa en 1.
  - **Indexado con desplazamiento en 8 bits:** La dirección del operando se obtiene de sumarle un número en 8 bits (signado) a la dirección del puntero HX.
  - **Indexado con desplazamiento en 8 bits y postincremento:** La dirección del operando se obtiene de sumarle un número en 8 bits (signado) a la dirección del puntero HX. Al terminarse la ejecución, la dirección del puntero (HX) se incrementa en 1.
  - **Indexado con desplazamiento en 16 bits:** La dirección del operando se obtiene de sumarle un número en 16 bits (signado) a la dirección del puntero HX.
  - **Relativo con SP y desplazamiento en 8 bits:** La dirección del operando se obtiene de sumarle un número en 8 bits (signado) a la dirección del puntero de pila (SP).
  - **Relativo con SP y desplazamiento en 16 bits:** La dirección del operando se obtiene de sumarle un número en 16 bits (signado) a la dirección del puntero de pila (SP).
- **Distribución de la memoria**  
La Figura 7.2 muestra la distribución de la memoria, en los 64 KB configurables, de las máquinas HCS08, para la familia JM.



**Figura 7.2. Distribución de la memoria en la familia HCS08.**

- **Distribución de los módulos**

La Figura 7.3 muestra la distribución de los módulos que conforman las máquinas HCS08, para la familia JM.



**Figura 7.3. Distribución de los módulos en la familia HCS08.**

## 7.2. EL CONCEPTO FLEXIS

Freescale se ha convertido en la primera compañía que ha desarrollado máquinas de 8 bits que puedan migrar a máquinas de 32 bits, con la filosofía de hacer que los diseños consuman la menor cantidad de energía y que el tiempo de desarrollo sea el más corto. Todo esto enmarcado en la estrategia llamada *Controller Continuum* (Figura 5.1, en el aparte 5.1).

La estrategia *Controller Continuum* apunta a la migración gradual desde los 8 bits hasta las arquitecturas más poderosas en ColdFire®, como son las versiones de la V2 a la V4. La versión V1 puede ser llamada “el eslabón perdido”, que no es tan poderosa como las versiones superiores pero permite la migración paulatina hacia máquinas más complejas. Esta estrategia también hace atractivas las máquinas ColdFire®, para los millones de consumidores de las arquitecturas ColdFire/68K.

Las aplicaciones realizadas en esta familia pueden migrar, con unos pocos movimientos del ratón, hacia la familia ColdFire® V1 de 32 bits y vice versa. Lo interesante de la migración, radica en el poder de ampliación que pueden tener los proyectos realizados en arquitecturas de 8 bits hacia máquinas de 32 bits, por una diferencia de costo muy reducida. En la migración son conservados aspectos como: el esquema de utilización de módulos, el mecanismo de las interrupciones, la distribución de puertos, el mecanismo de reloj, los modos de operación, entre otros.

De la Figura 7.4 se puede concluir como Freescale hace común los periféricos, un sólo pin para la depuración y hasta donde sea posible una misma distribución de pines, para las máquinas de 8 y 32 bits. La diferencia se encuentra en el acople de diferentes núcleos.

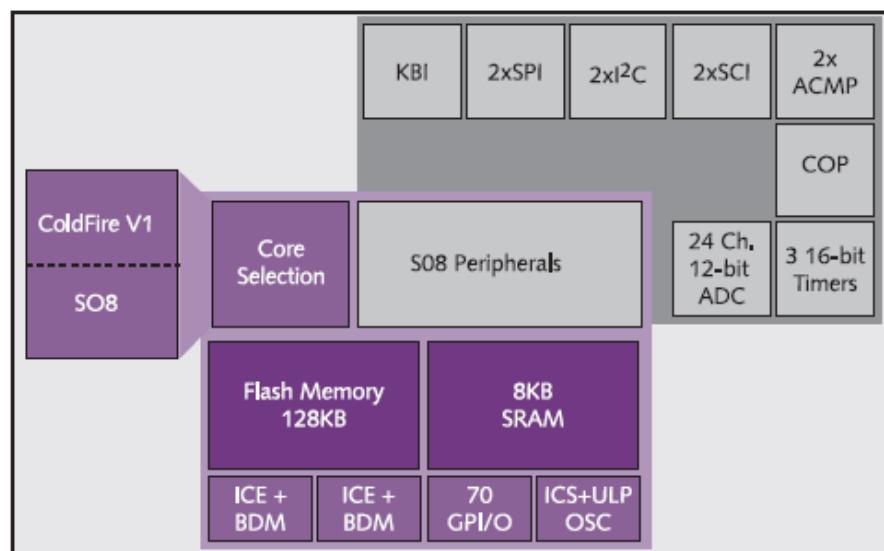


Figura 7.4. Diagrama común arquitectura FLEXIS<sup>28</sup>.

<sup>28</sup> Tomado del documento FREESCALE'S FIRST FLEXIS MCUs por Tom R. Halfhill (6/26/07-01), editado en el magazín MICROPROCESSOR REPORT.

La Tabla 7.4 hace un comparativo para la segunda serie de la familia ColdFire® V1, llamada JM. El comparativo expone las principales componentes de los MCU's, notándose la gran compatibilidad de ambas máquinas.

Para esta segunda serie de la familia FLEXIS, en el mapa de evolución de las familias ColdFire® V1, se nota un gran cambio en el sentido de la riqueza y actualidad de sus periféricos de comunicaciones. Aparecen buses tan importantes como el CAN y el USB, que no tenía la serie QE anterior.

**Tabla 7.1. Comparativo FLEXIS JM**

Característica	Flexis S08JM60	Flexis MCF51JM128
Arquitectura	HCS08	ColdFire V1
Ancho Arquitectura	8 bits	32 bits
SRAM	2K o 4K	8K o 16K
Memoria FLASH	32K o 60K	64K o 128K
Serial de Comunicaciones	2 x SCI	2 x SCI
Interfase IIC	IIC	2 x IIC
Interfase Serial Periféricos	2 x SPI	2 x SPI
USB	USB 2.0 alta velocidad	USB 2.0 On The Go
CAN	No	MSCAN
Interrupciones por Teclado	KBI - 7 a 8 canales	KBI de 8 canales
Pines I/O	33 hasta 51	33 hasta 66
PWM / Temporizadores (16 bits)	1 x 6 canales / 1 x 2 canales	1 x 6 canales / 1 x 2 canales
Watchdog Timer	COP con reloj independiente	COP
Conversor A/D	8 o 12 canales / 12 bits	8 o 12 canales / 12 bits
Comparador Análogo	ACMP	ACMP
Reloj Interno	RTC	RTC
Interfase Depurador	BDM por un pin	BDM por un pin
Frecuencia del Núcleo	48 MHz	50 MHz
Frecuencia del Bus	24 MHz	25 MHz
Proceso de Fabricación	0.25 micrones	0.25 micrones
Voltaje	(2.7 - 5.5) Vdc	(2.7 - 5.5) Vdc
Rango de Temperatura	-40 hasta 105 °C	-40 hasta 105°C
Número de Pines	44 - 64 pines	44 - 80 pines
Tipos de Empaque	44 - 64LQFP, 48QFN, 64QFP	44,64,80 LQFP / 64 QFP
Tamaño Empaque	10 mm2 o 14 mm2	10 mm2 o 14 mm2
Consumo	52 mW @ 3V @ 24MHz	100mW @ 3V @ 25MHz
Desempeño Dhystone	n/a	0.94Dmips/MHz RAM 0.76Dmips/MHz FLASH
Precio (10K)	Desde US\$2.74	Desde US\$3.70
Disponibilidad	Activo	Activo

El consumo es otro de los aspectos cruciales a la hora de migrar de tecnologías de 8 a 32 bits. En este aspecto, Freescale ha sido cuidadosa y ha sorprendido con las mínimas diferencias de consumo entre las máquinas FLEXIS. Mientras que las arquitecturas de 8 bits consumen en promedio 22 mA, las arquitecturas de 32 bits consumen en promedio

60.9 mA. Estos datos son tomados en máximo desempeño y un reloj de operación de 48MHz.

**NOTA:** En el Capítulo 8 aparte 8.4, aparece el primer ejemplo de migración de proyectos elaborados en máquinas de 8 bits, hacia máquinas de 32 bits.

### 7.3. CUIDADOS EN LA MIGRACIÓN

El usuario deberá tener ciertas precauciones al momento de migrar proyectos con la tecnología FLEXIS. Como ya se vió, las arquitecturas de los núcleos de las máquinas son muy distintas, aunque se hayan conservado muchas de las características de los periféricos de las MCU's.

Algunas de estas precauciones son listadas a continuación.

- **Incompatibilidad de instrucciones a bajo nivel:** Debido a la gran diferencia entre sus núcleos, no es recomendable insertar como código de un programa instrucciones en línea de *assembler*.

Si en un programa es necesario la inclusión de instrucciones a bajo nivel, el usuario deberá hacer la traducción al nuevo *assembler*. Pero en lo posible, es recomendable eliminar el código a bajo nivel y reemplazarlo por instrucciones en C/C++. Por ejemplo:

En vez de usar la instrucción (del HCS08):

asm BSET 0 , PTADD

Mejor usar:

PTADD0 = 1;

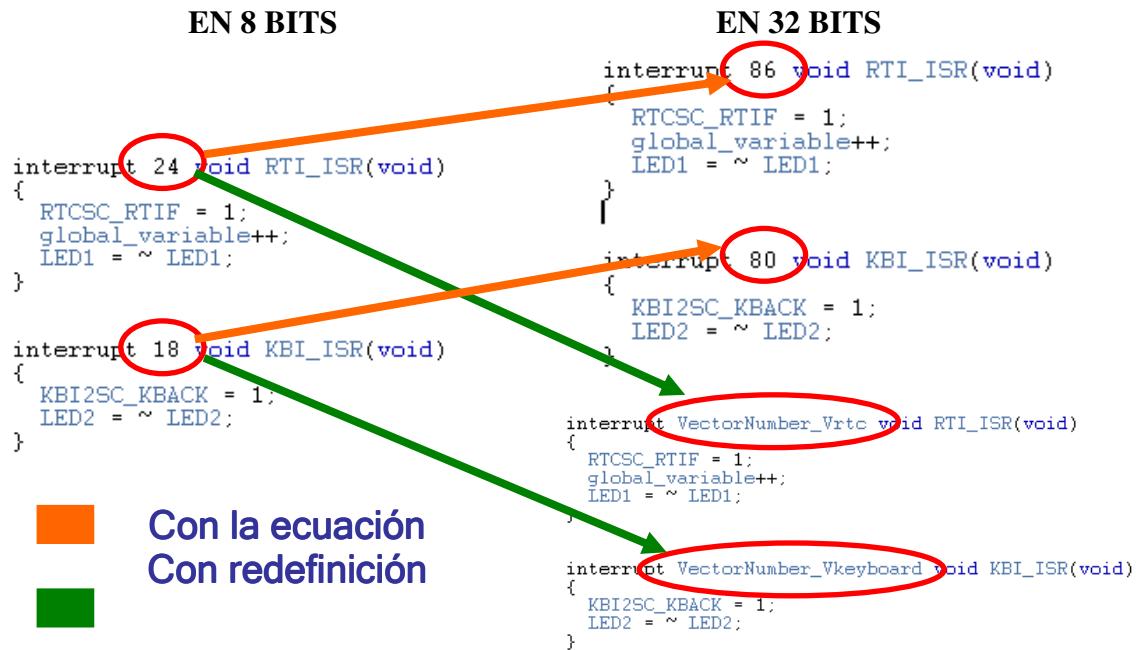
- **Proceso de excepciones y vectorización de las interrupciones de periféricos:** Los vectores de atención a las interrupciones de ambas máquinas no son directamente compatibles, porque los números de los vectores de atención a interrupciones no son los mismos. El usuario debe recordar la ecuación que los relaciona y que a continuación se escribe.

Número del vector en ColdFire®\_V1 = Número del vector en HCS08 + 62

Otra alternativa es consultar en el archivo que tiene la redefinición nombre el número de la MCU que se va a utilizar (Pej: MCF51JM128.h), el cual se localiza en el directorio **Includes** del proyecto. Por ejemplo, el vector de atención a la interrupción del módulo RTC en la máquina MC9S08JM60 es el 29, cuyo equivalente en la máquina MCF51JM128 es el 91. Lo anterior se comprueba en la ecuación:

$$\text{Número del vector en ColdFire®_V1} = 29 + 62$$

La Figura 7.5 explica dos maneras de hallar el vector correcto para migrar de 8 a 32 bits:



**Figura 7.5. Cálculo del vector de atención a interrupción en la migración.**

- **Mapas de memoria no compatibles:** Tener en cuenta que al no compartir las direcciones de la ubicación de la memoria del sistema, las declaraciones absolutas no funcionan.

Las ubicaciones y tamaños de los diferentes tipos de memoria son controladas por el LINKER del sistema.

Un ejemplo de una declaración no apropiada de una variable en memoria sería:

**int var @ 0x400 = 1;**

En este caso las dos arquitecturas lo interpretarían distinto, por ser del tipo absoluto.

Una solución sería utilizar una declaración como:

```
unsigned int var = 1;
```

Pero hay cierta relación en la ubicación de la memoria de configuración de registros de periféricos y está controlada por la siguiente relación:

Dirección registro periférico 8 bits = Dirección registro periférico 32 bits + 0xFFFF8000
---

Por ejemplo:

Para el S08JM60: TPM1SC = 0x0020

Para el 51JM128: TPM1SC = 0xFFFF8000 + 0x0020 = 0xFFFF8020

El compilador CodeWarrior incluye instrucciones tipo `#pragma`, que hacen que el código sea más portable. Por ejemplo:

```
#pragma warn_absolute on //genera un reporte de todas las
//declaraciones absolutas encontradas

#pragma check_asm report //genera un reporte de todas las instrucciones
//que encontró en formato assembler
```

- **Tiempos de ejecución diferentes:** Por razones muy obvias, la máquina de 32 bits supera en velocidad a la máquina de 8 bits. Es por lo anterior que se hace necesario tener en cuenta los ajustes de velocidad, debidos a los retardos y temporizaciones al interior de los programas.

#### **7.4. REFERENCIAS**

- MC908JM60 Series Data Sheet. Rev 1. 11/2007. Freescale Semiconductor.
- MCF51JM128 Integrated Microcontroller Reference Manual. Rev 1. 01/2008. Freescale Semiconductor.
- Halfhill, Tom R. Freescale First Flexis MCUs. 06/2007. Microprocessor Report.

#### **7.5. PREGUNTAS**

- Desde el punto de vista de los modos de direccionamiento, ¿sería posible encontrar la compatibilidad entre las máquinas HCS08 y ColdFire® V1?
- Explique en que consiste el concepto FLEXIS.
- Enuncie dos cuidados que se deben tener al migrar de las máquinas HCS08 hacia las máquinas ColdFire® V1

# **PARTE III**

## **AMBIENTE DE PROGRAMACIÓN, HERRAMIENTAS DE DESARROLLO Y LENGUAJES DE PROGRAMACIÓN**

### **OBJETIVO**

Este aparte tiene como objetivo la introducción al lector sobre el ambiente de programación CodeWarrior®, en su versión 6.2. Otro objetivo importante es la presentación de la herramienta de desarrollo sobre la cual se basan las prácticas del libro, llamada DEMOJM.

### **CONTENIDO**

#### **CAPÍTULO 8. Ambiente de programación y herramienta de desarrollo**

El Capítulo 8 está cargado de un alto contenido de herramientas aplicadas al desarrollo con microcontroladores de 8 y 32 bits. Se hace una descripción mínima para el uso del software CodeWarrior® 6.2, una descripción de iguales características para el DEMOJM y un primer programa en donde se demuestra el uso del editor, compilador y depurador de la herramienta. Al final del capítulo es desarrollado un procedimiento de migración.

# CAPÍTULO 8

## Ambiente de Programación y Herramienta de Desarrollo

- 8.1. Introducción al compilador CodeWarrior®**
- 8.2. Breve descripción de la herramienta DEMOJM**
- 8.3. Creación de proyectos en C**
- 8.4. El primer programa en C**
- 8.5. Referencias**
- 8.6. Preguntas**

## 8.1. INTRODUCCIÓN AL COMPILADOR CODEWARRIOR®

Definiremos brevemente los menús que componen el software CodeWarrior® 6.2 y que aparecen en el menú principal según la Figura 8.1.



Figura 8.1. Menú principal CodeWarrior® 6.2.

- **File:** Relacionado con el manejo de los proyectos y archivos del sistema. La Figura 8.2 muestra un despliegue de este menú.

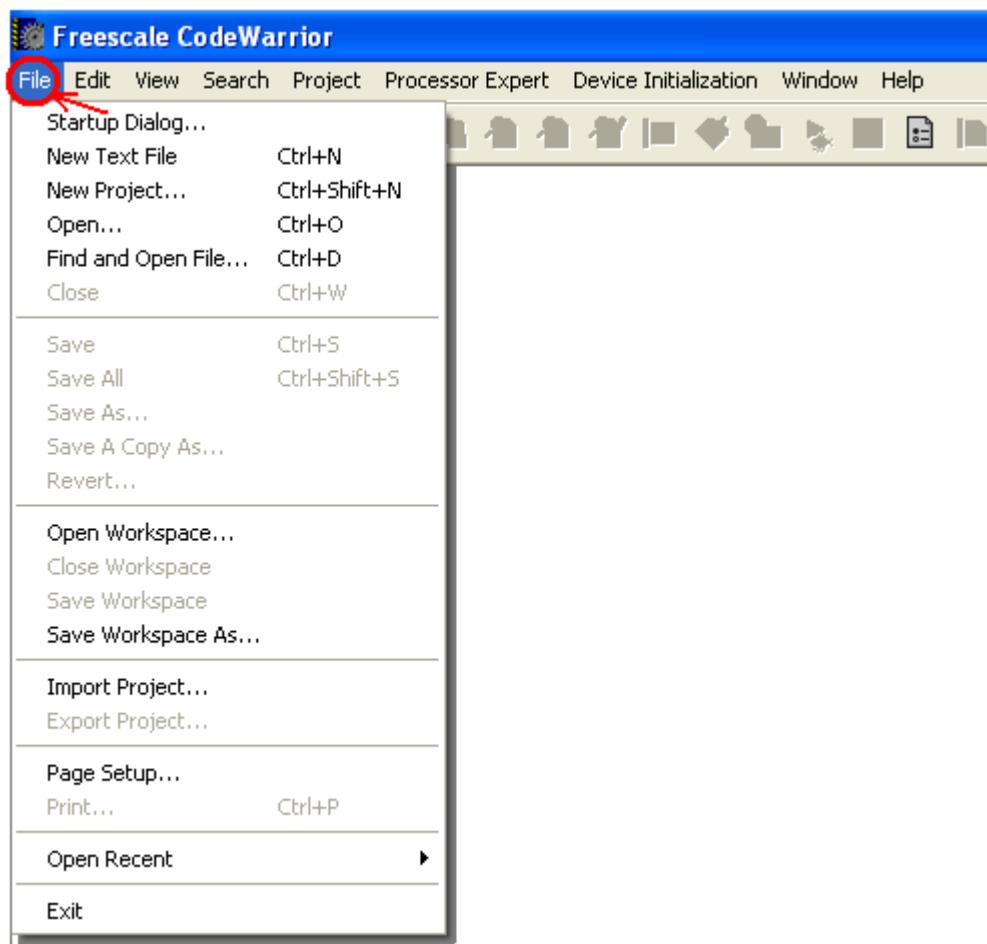
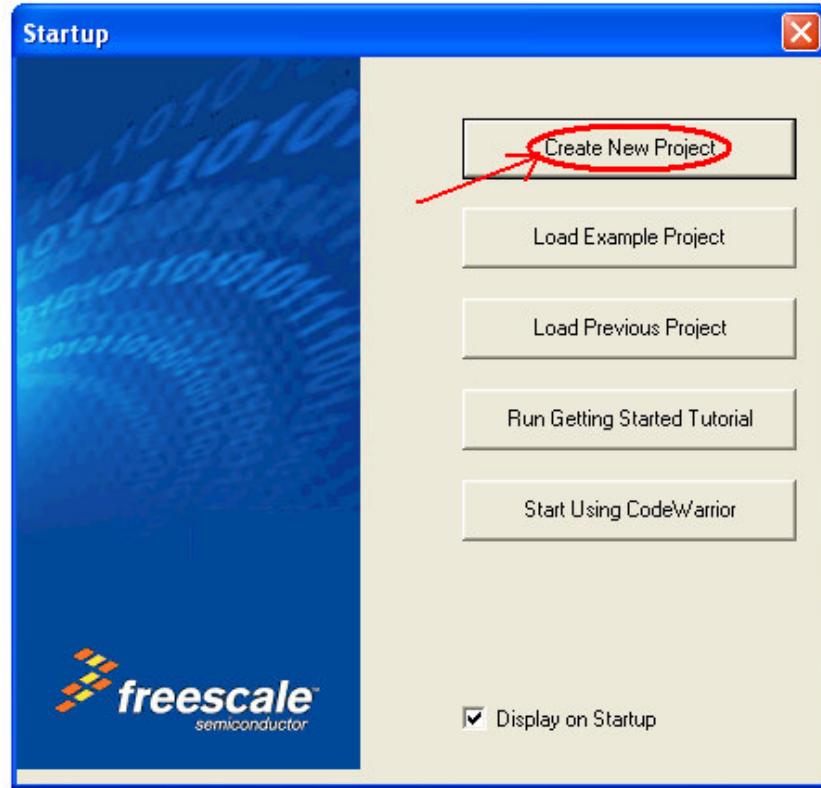


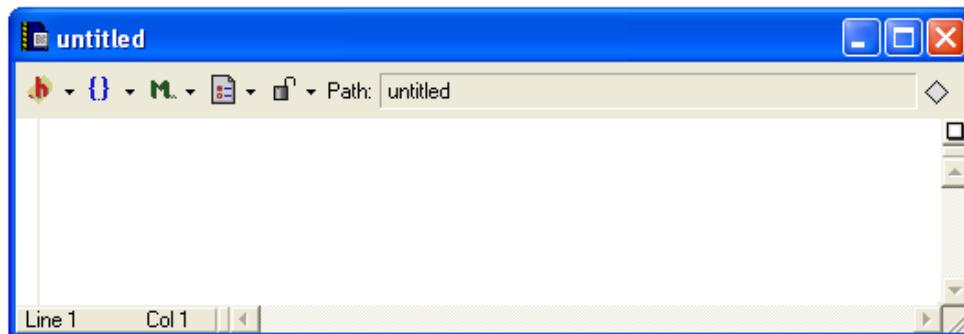
Figura 8.2. Submenú file.

- **Startup Dialog...:** Presenta la ventana inicial, que aparece al lanzar el CodeWarrior® 6.2 (ver Figura 8.3.). Se emplea para configurar un proyecto inicial y puede ser desde crear uno nuevo, cargar un ejemplo ya existente, cargar el inmediatamente anterior, ejecutar un tutorial o comenzar desde cero.



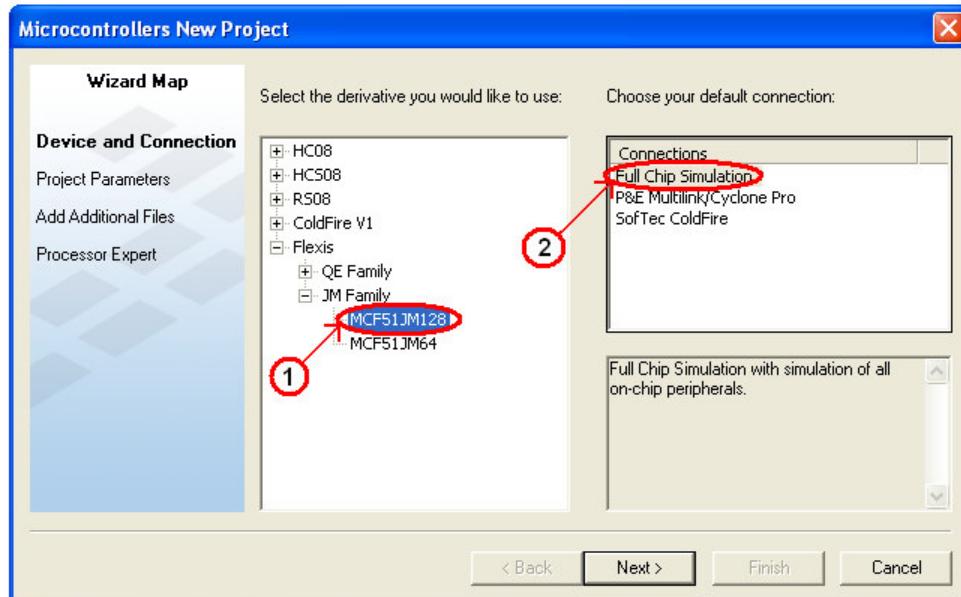
**Figura 8.3. Submenú Startup Dialog.**

- **Next Text File:** Es el editor del software, utilizado para crear archivos de texto de cualquier índole (ver Figura 8.4.).



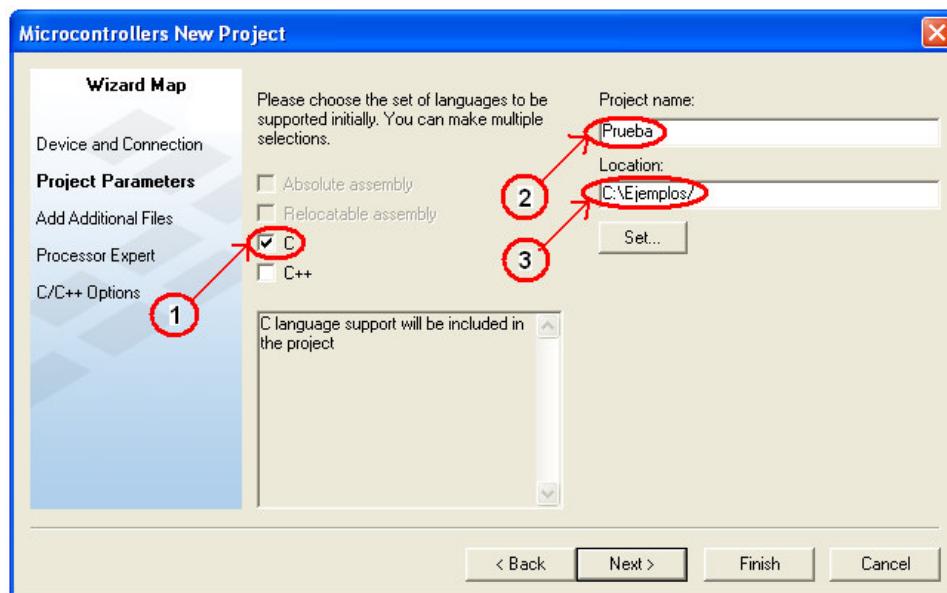
**Figura 8.4. Submenú Next Text File.**

- **New Project:** Usado para generar proyectos nuevos CodeWarrior. Se compone de las ventanas:
  - **Device and Connection:** Selección del MCU y el tipo de conexión para la depuración y programación (ver Figura 8.5).



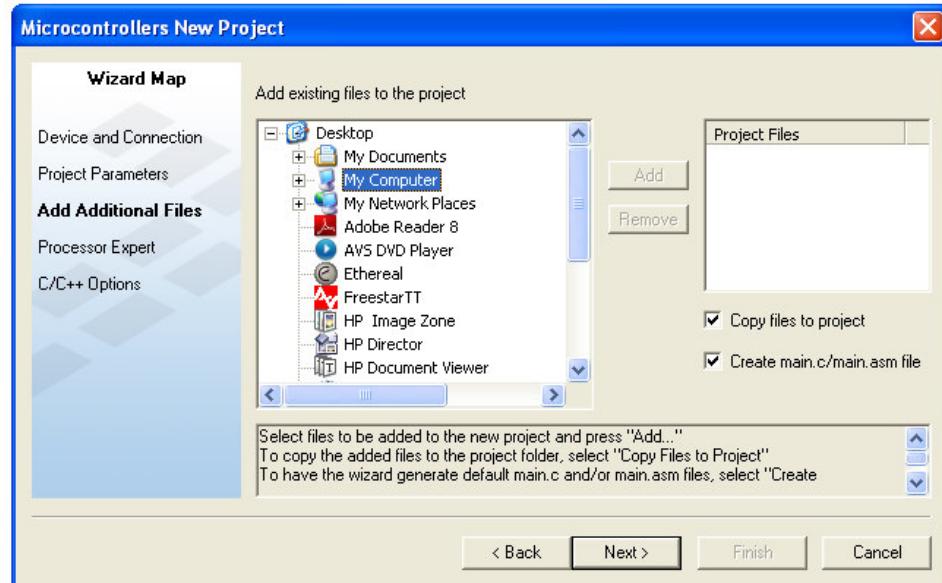
**Figura 8.5. Submenú Device and Connection.**

- **Project Parameters:** Selección del lenguaje de programación, el nombre del proyecto y la ruta de ubicación (ver Figura 8.6).



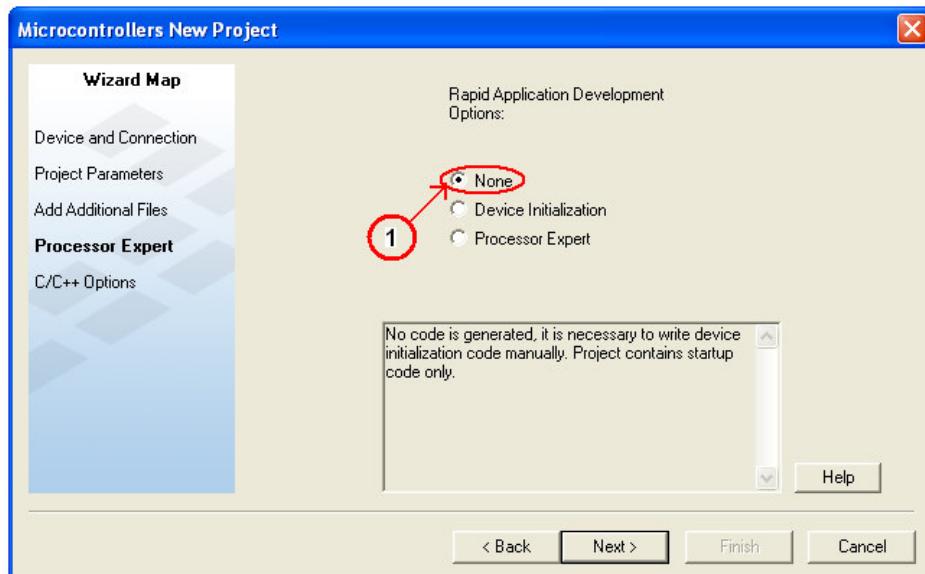
**Figura 8.6. Submenú Project Parameters.**

- **Add Additional Files:** Inserción de archivos adicionales que conformarán el cuerpo del proyecto (C, C++, asm, h..etc) (ver Figura 8.7).



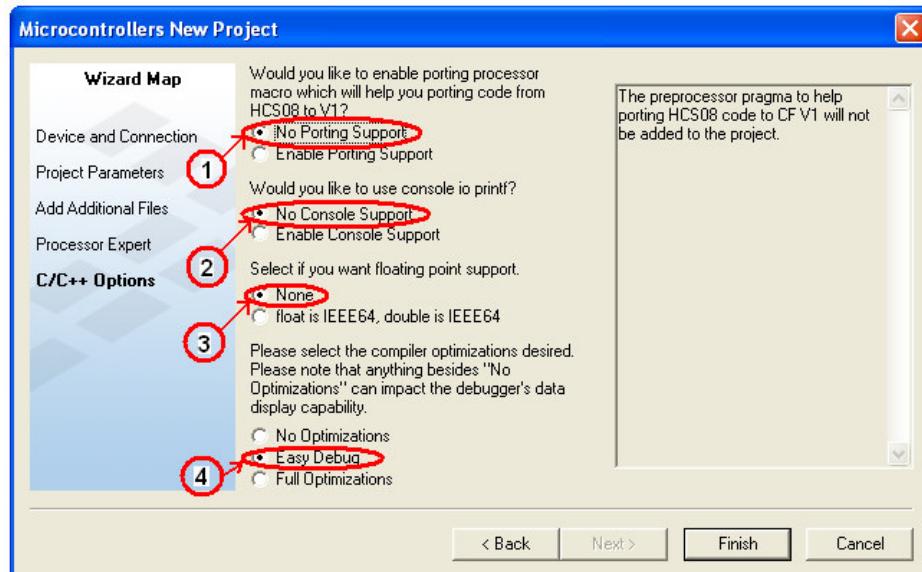
**Figura 8.7. Submenú Add Additional Files.**

- **Processor Expert:** Seleccionar la opción del procesador experto, que facilita la inicialización de muchas de las funciones y periféricos de las máquinas (como se explicará en el Capítulo 10) (ver Figura 8.8).



**Figura 8.8. Submenú Procesor Expert.**

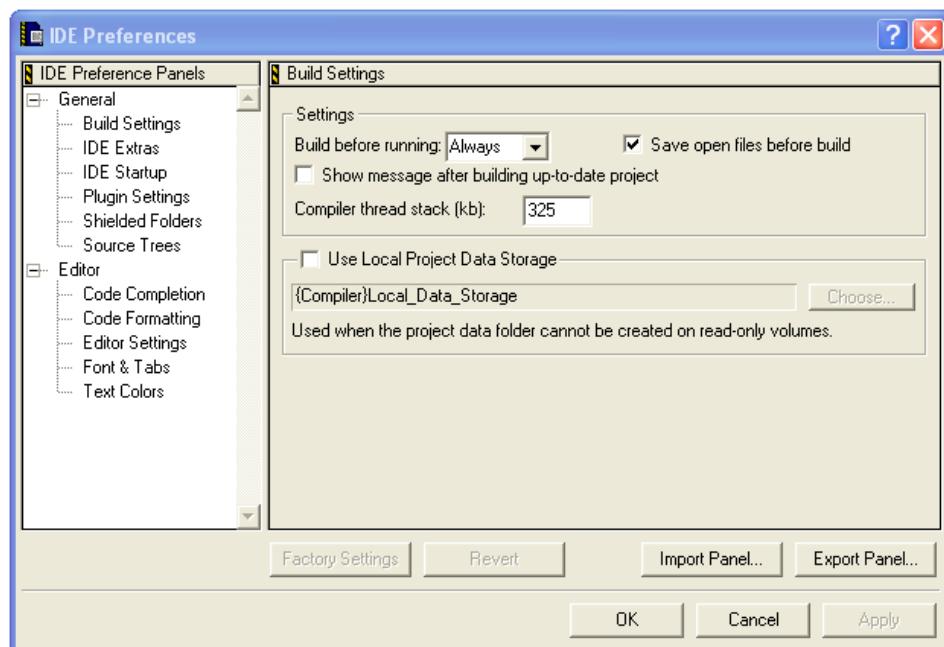
- **C/C++ Options:** Selección del nivel del código a usar, de acuerdo al estándar de C, el modelo de memoria y el formato de punto flotante. Todo lo anterior debe considerar la mejor densidad de código posible (ver Figura 8.9).



**Figura 8.9. Submenú C/C++ Options.**

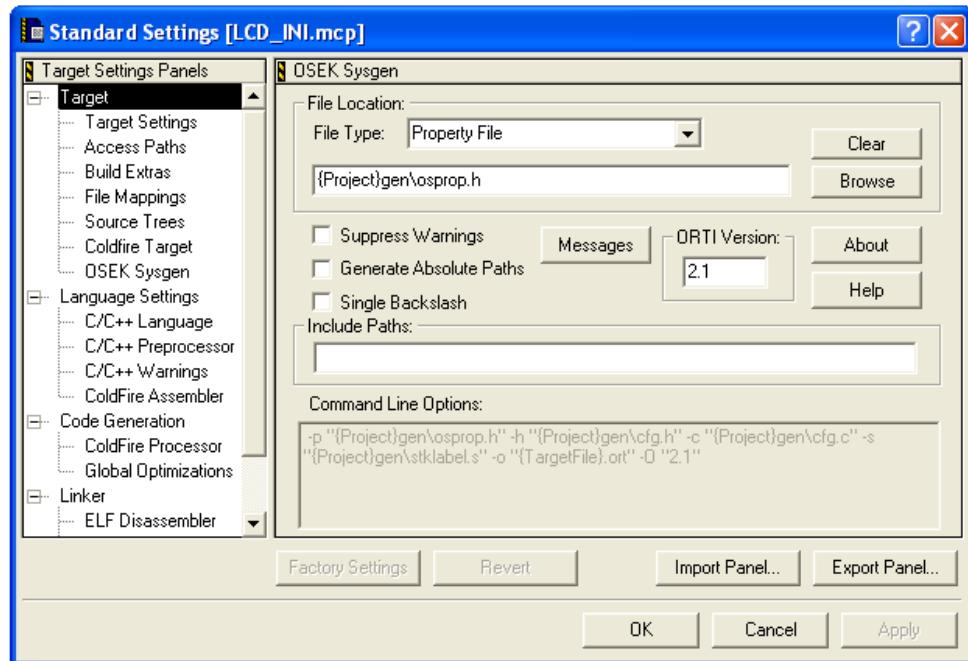
- **Open:** Clásico submenú para abrir proyectos o archivos almacenados en memoria de disco.
- **Find and Open File:** Clásico submenú para abrir hallar y abrir archivos o proyectos.
- **Save, Save All, Save As, Save a Copy As, Revert:** Serie de submenús para guardar archivos y/o proyectos. La opción *Revert*, recupera la última versión guardada y la sobrescribe en la presente.
- **(Open, Close, Save, Save As) Workspace:** Serie de submenús para aplicar al ambiente de trabajo configurado.
- **(Import, Export) Project:** Funciones para importar y exportar proyectos en el ambiente CodeWarrior®.
- **Page Setup:** Opciones para el formato de edición.
- **Print:** Imprimir documento abierto.
- **Open Recent:** Abrir los últimos proyectos trabajados.
- **Exit:** Salir del CodeWarrior® 6.2.

- **Edit:** Submenú de edición del CodeWarrior® 6.2. Contiene las funciones normales de un editor de textos, pero sólo serán descritas las especiales del submenú.
  - **Preferences....:** Permite configurar preferencias generales y del editor de textos (ver Figura 8.10).



**Figura 8.10. Submenú Preferences.**

- **Build Settings:** Preferencias en la construcción del proyecto.
- **IDE Extras:** Algunos aspectos adicionales de la apariencia del editor, como el tipo de menú, disponibilidad de múltiples documentos, entre otros.
- **IDE Startup:** Acciones a ejecutar cuando se abre el editor.
- **Plugins Settings:** Parámetros de visualización para los mensajes del editor.
- **Shielded Folders:** Bloqueo de directorios bajo ciertos símbolos.
- **Source Trees:** Definir los caminos para el almacenamiento de los proyectos y archivos.
- **Code Completion:** Definir las reglas para auto completar el código escrito.
- **Code Formating:** Definir las reglas para el formato del código escrito.
- **Editor Settings:** Configuración general del editor de textos.
- **Font and Tabs:** Definir el tipo de la fuente de caracteres y el espaciamiento del tabulador.
- **Text Colors:** Definir el color de la fuente de caracteres.
- **Standar Settings:** Se utiliza para configurar los parámetros del destino sobre la acción de encadenamiento (linker) (ver Figura 8.11).



**Figura 8.11. Submenú Standar Settings.**

- **Target:** Tipo de encadenador, caminos de acceso a los proyectos y archivos, entre otros.
- **Language Settings:** Configuración sobre la norma del lenguaje (C, C++) y algunas opciones de errores y mensajes de precaución.
- **Code Generation:** Elección del tipo de procesador, modelo de programación, modelo del dato y tipo de optimización en la compilación del código.
- **Linker:** Configuración sobre el des-ensamblador y el encadenador (linker).
  
- o **VCS Settings:** Se utiliza para configurar los parámetros de versión del código y seguridad del mismo (ver Figura 8.12).
- o **Customize IDE Commands:** Relacionado con la configuración de la apariencia del editor IDE (ver Figura 8.13).

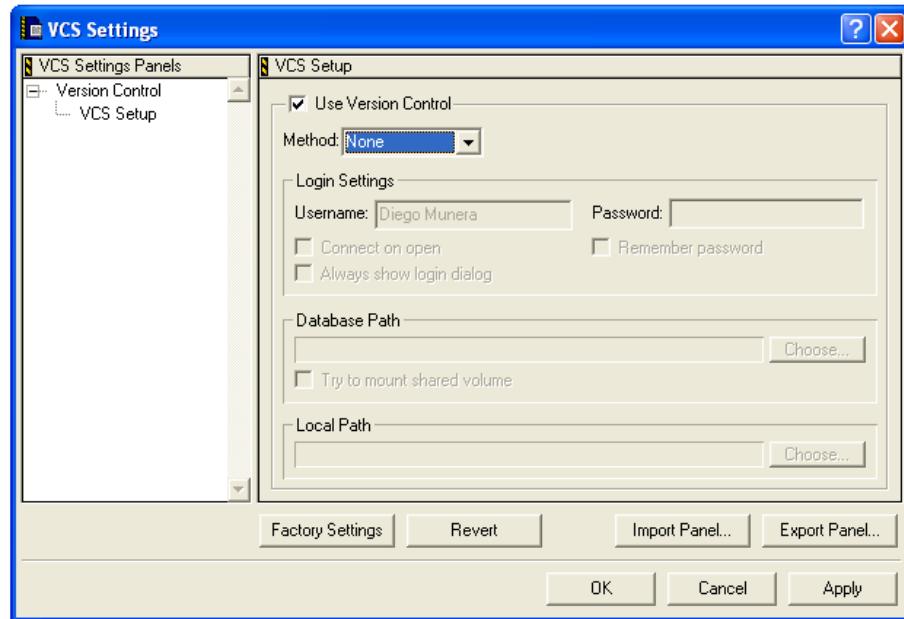


Figura 8.12. Submenú VCS Settings.

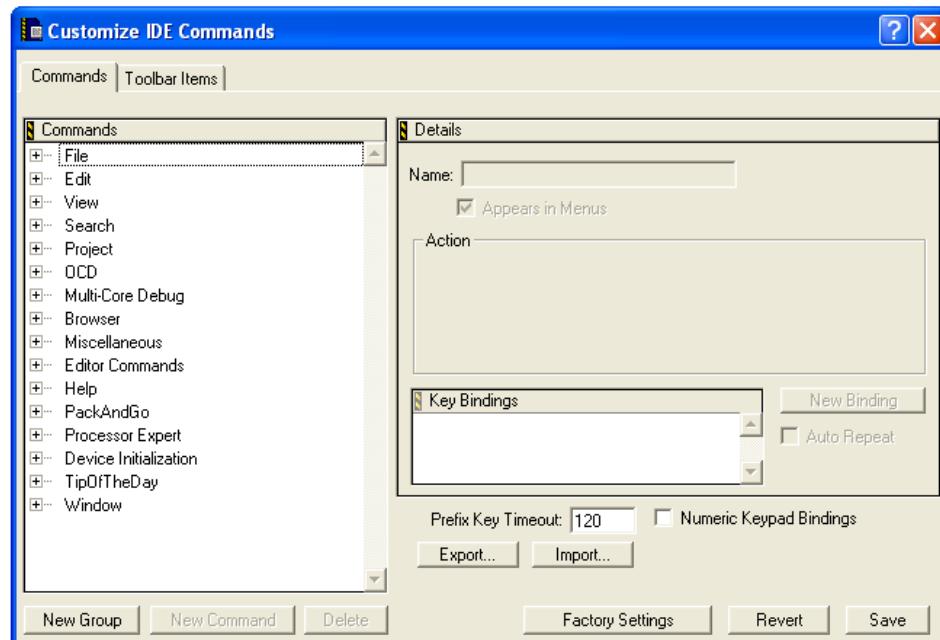
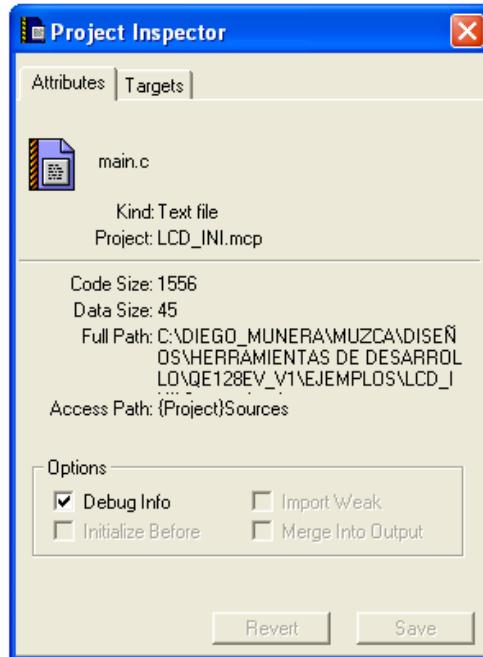


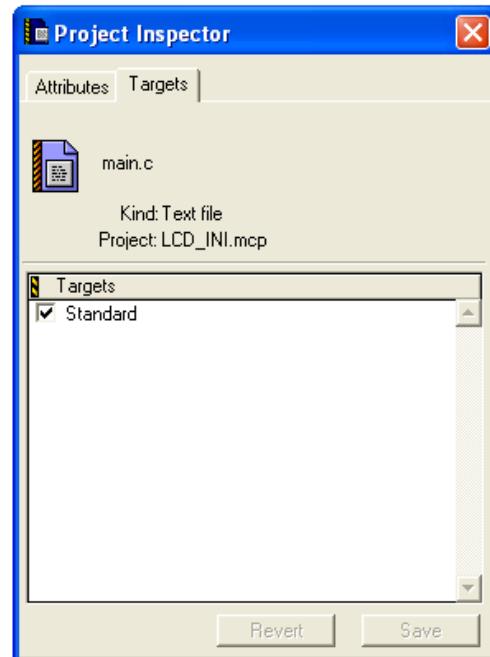
Figura 8.13. Submenú Customize IDE Commands.

- **View:** Submenú de visualización del CodeWarrior® 6.2. Esta compuesto por:
  - **Toolbars:** Barras de visualización de los menú de herramientas del IDE.

- **Project Inspector:** Se pueden visualizar, de manera rápida, algunos atributos del proyecto (ver Figura 8.14) y el destino del main.c del sistema (ver Figura 8.15).



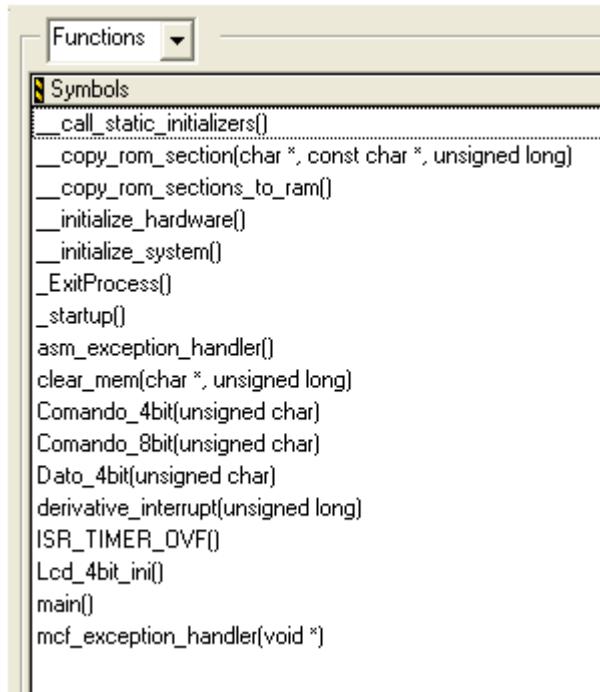
**Figura 8.14. Submenú Attributes**



**Figura 8.15. Submenú Targets**

- **Browser Contents:** Su función principal es la de poder visualizar información del tipo de una variable (ver Figura 8.16):

- Class
- Constants
- Enums
- Functions
- Globals
- Macros
- Templates
- Typedefs



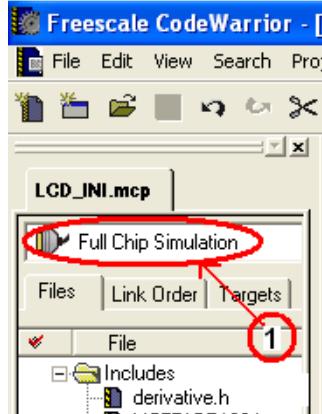
**Figura 8.16. Submenú Browser Contents**

- **Class Browser:** Visualización por clases en el Browser.
- **Class Hierarchy:** Visualización por jerarquías en el Browser.
- **Build Progress:** Visualización del progreso en la construcción del proyecto.
- **Errors and Warnings:** Visualización de errores y precauciones en la construcción del proyecto.
- **View:** Submenú para la búsqueda, reemplazo y comparación de información en un texto que pertenezca a un texto. Para destacar de este menú están:
  - **Compare Files:** Compares contenidos de dos archivos y arroja las diferencias en un reporte tipo texto.

- **Apply Differences:** Actualiza el archivo destino con las diferencias encontradas en la comparación con el archivo fuente.
  - **Unapply Differences:** Revierte el proceso anterior.
- 
- **Project:** Submenú para administración del proyecto.
    - **Add main.c to Project:** Adiciona un archivo main.c al proyecto.
    - **Add Files:** Adiciona archivos al proyecto.
    - **Create a Group:** Crea un nuevo directorio dentro del navegador del proyecto.
    - **Create Target:** Crear un nuevo destino para los archivos del proyecto.
    - **Check Syntax:** Evalúa la sintaxis del archivo abierto.
    - **Preprocess:** Pre-procesa el archivo abierto.
    - **Precompile:** Pre-compila el archivo abierto.
    - **Compile:** Ejecuta la compilación de todo el proyecto.
    - **Dissasemble:** Desensambla el código C/C++ a instrucciones en ensamblador y códigos de máquina.
    - **Make:** Compilación completa del proyecto.
    - **Stop Build:** Interrumpe cualquier acción de compilación.
    - **Remove Object Code:** Se usa para remover código objeto binario del proyecto activo.
    - **Re-search for Files:** Agiliza la construcción y otras operaciones del proyecto, El software IDE captura la localización de los archivos del proyecto después de encontrarlos en las rutas de acceso. Esta opción fuerza al IDE para olvidar las localizaciones de los archivos del proyecto y reinicia su búsqueda en las rutas señaladas.
    - **Reset Project Entry Paths:** Reinicia las rutas de archivos del proyecto.
- 
- **Sincronize Modification Dates:** Actualiza las fechas de modificación de los datos almacenados en el archivo del proyecto. El IDE verifica la fecha de

modificación de cada archivo en el proyecto, y marca los archivos modificados, desde el último proceso de compilación exitosa.

- **Debug:** Lanza la opción de depuración, que dependerá del valor establecido en la ventana del navegador del proyecto, según indica el puntero del ratón en la Figura 8.17.



**Figura 8.17. Ventana para selección del tipo de simulación (Debug)**

La ventana de simulación se muestra en la Figura 8.18 y son sus partes más importantes:

- **Source:** Programa fuente en depuración. Muestra el código tal cual se digitó y que se está ejecutando en el proceso de depuración. Una barra azul indica la instrucción que se va a ejecutar y su dirección debe coincidir con la dirección del registro PC. Dentro de esta ventana, haciendo presión sobre el botón derecho del ratón, es posible generar las siguientes acciones:
  - ❖ **Set Breakpoint:** Ubicar puntos de chequeo.
  - ❖ **Run to Cursor:** Ejecutar el programa hasta la posición del cursor.
  - ❖ **Show Breakpoints:** Mostrar los puntos de cheque activos.
  - ❖ **Show Location:** Mostrar la localización de la instrucción a ejecutar en la memoria y en la ventana del ensamblador.
  - ❖ **Set/Delete Mark Points:** Ubicar/Retirar marcas de referencia visual.
  - ❖ **Show Markpoints:** Mostrar los puntos de referencia visual.
  - ❖ **Set Program Counter:** Cambiar el valor del PC.
  - ❖ **Open Source File:** Abrir un archivo fuente del proyecto.
  - ❖ **Copy:** Copiar.
  - ❖ **Goto Line:** Ir a un número de línea del listado de programa.
  - ❖ **Find:** Buscar una secuencia de caracteres.
  - ❖ **Find Procedure:** Buscar un procedimiento.
  - ❖ **Folding:** Muestra o esconde una sección del programa.
  - ❖ **Marks:** Ubica triángulos donde se pueden ubicar breakpoints.

**True-Time Simulator & Real-Time Debugger C:\DIEGO\_MUNERA\MIUZCA\DISEÑOS\HERRAMIENTAS DE DESARROLLO\QE128EV\_V1\EXAMPLES\LCD\_INIC...**

**S Source**

```
C:\DIEGO_MUNERA\MIUZCA\DISEÑOS\HERRAMIENTAS DE DESARROLLO\QE128EV_V1\EXAMPLES\LCD_INIC... Line: 113
/*
 * MAIN */
void main(void) { //Inicio del lazo principal de pr
    //Habilita interrupciones globales:
    EnableInterrupts();
    //Inicializa reloj interno:
    ICSC1 = 0x06;
    //Baja frec. de bus interno, divisor :
    //Referencia de 32.768KHz, divisor :
    //Este macro ejecuta CLI
    _startup()
```

**P Procedure**

**A Assembly**

	main
D0	MOVE.L A6,-(A7)
0410	D7,-(A7)
D4	MOVE SR,D0
0414	0416 ANDI.L #0xFFFF,D0
D0	MOVE D0,SR
041E	MOV3Q #6,D2
0420	MOVE.B D2,0xFFFF8038
0424	CLR.B 0xFFFF8039

**D Data:1**

	main.c
mensaje	<17> array[17] of unsigned char
time_us	0 unsigned char
_BANDERA	<1> BANDERA
i	1 unsigned int
pm	MULL * unsigned char

**R Register**

	main
D0	0
D4	0
D8	0
A4	0
A5	800004
pr	41n
SR	2714

**M Memory**

	Auto
000080	00 00 00 80 00 00 00 80
000088	00 00 00 80 00 00 00 80
000090	00 00 00 80 00 00 00 80
000098	00 00 00 80 00 00 00 80
0000A0	00 00 00 80 00 00 00 80

**C Command**

STARTED  
RUNNING  
Breakpoint  
ir>

For Help, press F1

4.194304 MHz | 598 | MCF51QE128 | Breakpoint

Figura 8.18. Ventana del depurador (Debug)

- **Assembly:** Muestra el código ensamblado que corresponde a la instrucción en la ventana del programa fuente (Source). Haciendo presión sobre el botón derecho del ratón, es posible generar las siguientes acciones (se destacarán aquellas que son nuevas al lector):
  - ❖ **Display:** Permite visualizar el código en lenguaje ensamblador, la dirección de la ubicación de la instrucción en memoria de programa, los símbolos asignados a direcciones y el código de operación de la instrucción.
  - ❖ **Format:** Selección del formato de presentación de las cantidades numéricas.
- **Procedure:** Muestra los procedimientos ejecutados y en ejecución. Haciendo doble pulsación con el ratón, es posible cambiarse de procedimiento.
- **Register:** Muestra el estado de los registros internos al MCU. También muestra el número de ciclos de reloj consumidos durante la simulación del programa. Haciendo doble pulsación sobre algún registro, es posible cambiar su valor. Haciendo presión sobre el botón derecho del ratón, es posible generar la siguiente acción:
  - ❖ **Format:** Selección del formato de presentación de las cantidades numéricas.
- **Data1, 2:** Visualización de datos en general. Cualquier variable del sistema puede ser visualizada y actualizada de valor haciendo doble pulsación sobre la variable. Haciendo presión sobre el botón derecho del ratón, es posible generar las siguientes acciones:
  - ❖ **Open Module...:** Llama las variables involucradas en un determinado módulo o proceso.
  - ❖ **Add Expression:** Adiciona variables a la ventana.
  - ❖ **Set/Delete>Show Watchpoints:** Permite establecer zonas de visualización de variables, eliminarlas y mostrar cuáles están activas.
  - ❖ **Show Location:** Muestra la ubicación en memoria de la variable seleccionada.
  - ❖ **Zoom In/Out:** Amplifica/reduce al interior de los componentes de una determinada variable.
  - ❖ **Scope:** Permite visualizar globalmente o localmente los datos del proyecto.
  - ❖ **Mode:** Modo de actualización del valor de los datos.
  - ❖ **Format:** selección del formato de presentación de los datos.
  - ❖ **Options:** Permite presentar los punteros como arreglos y establecer el ancho en caracteres de los datos.
  - ❖ **Sort:** Establece el orden de presentación de los datos.

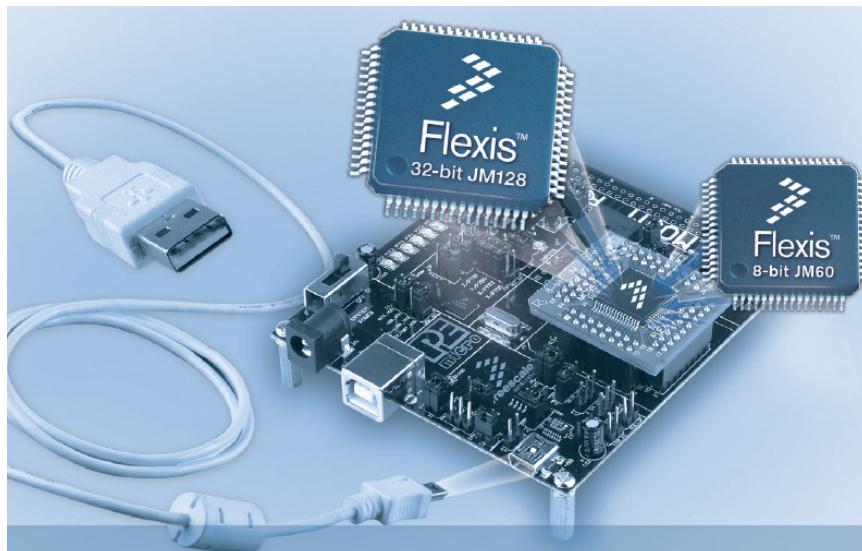
- ❖ **Refresh:** Actualización inmediata del valor de los datos.
- **Memory:** Visualización de la memoria absoluta del sistema. Haciendo doble pulsación sobre el dato en memoria, es posible cambiar su valor (siempre y cuando se pueda cambiar). Haciendo presión sobre el botón derecho del ratón, es posible generar las siguientes acciones (se destacarán aquellas que son nuevas al lector):
  - ❖ **Word Size:** Permite configurar la información de la memoria como byte, word o longword.
  - ❖ **Display:** Visualización de la dirección de las celdas y el contenido en formato ASCII.
  - ❖ **Fill....:** Se usa para llenar áreas de memoria con un valor.
  - ❖ **Address....:** Se usa para hallar una dirección específica de memoria.
  - ❖ **Copymem....:** Se usa para copiar un bloque de la memoria hacia otro destino.
  - ❖ **Search Pattern....:** Se usa para hallar un patrón de valores de memoria.
- **Command:** Visualización de las acciones ejecutadas en la sesión de simulación (Debug). Haciendo presión sobre el botón derecho del ratón, es posible generar las siguientes acciones:
  - ❖ **Clear:** Borra la historia de depuración.
  - ❖ **Execute File:** Permite la ejecución de un archivo como parte del proceso de depuración.
  - ❖ **Cache Size:** Permite definir el tamaño del *cache* de depuración.
- **Set Default Project:** Cuando se tienen varios proyectos abiertos, esta opción permite darle la propiedad de proyecto por defecto a uno de ellos.
- **Change MCU Connection:** Permite cambiar de modelo de MCU. Esta opción es importante cuando se va a migrar de una máquina a otra.
- **Processor Expert:** Submenú de la herramienta de pre-configuración de proyectos. Esta opción puede generar código de inicialización y configuración de los periféricos de la MCU y externos; también puede generar algunos algoritmos de programación.
- **Device Initialization:** Esta herramienta permite generar código de inicialización de los periféricos internos a la MCU, configuración de la tabla de vectores de interrupción y establecer plantillas para las rutinas de atención de interrupciones del sistema.
- **Window:** Submenú clásico para el manejo de ventanas.

- **Help:** Submenú de las ayudas. Se recomienda al lector hacer un ejercicio de navegación por este menú.

## 8.2. BREVE DESCRIPCIÓN DE LA HERRAMIENTA DEMOJM

**NOTA:** Los Autores recomiendan el sistema DEMOJM, para implementar los ejercicios trabajados en el libro, debido a su bajo costo y a la facilidad de aprender sobre la tecnología FLEXIS. También se hace especial recomendación en la lectura del manual de usuario, incluido en el DVD-ROM del DEMOJM.

Este sistema de desarrollo soporta los microcontroladores de Freescale: MC9S08JM60 y MCF51JM128 en empaques 64LQFP, que pueden ser intercambiables bajo el concepto FLEXIS (ver Figura 8.19).



**Figura 8.19. Sistema de desarrollo DEMOJM<sup>29</sup>**

Los contenidos más importantes del sistema DEMOJM son:

- Un analizador lógico de dos canales, que puede ser utilizado para la visualización de datos en tiempo real sobre un PC. Se recomienda al usuario del DEMOJM de informarse sobre esta aplicación en el DVD-ROM incluido en el DEMOJM.
- Un puerto virtual USB conectado a un puerto SCI del MCU JM. Un programa que emula un terminal serial es suministrado en el DVD-ROM y que el usuario utilizará como un puerto serial virtual.

<sup>29</sup> Fuente: DEMOJM Quick Start Guide. Freescale Semiconductor.

- Un conector asimétrico para la inserción de los MCU's (MC9S08JM60 o MCF51JM128), llamado **JM DAUGHTER CARD**.
- Una interfase embebida P&E MULTILINK, para la programación y depuración de los programas.
- Puerto SCI conectado, vía puentes, a la interfase embebida P&E MULTILINK.
- Interruptor ON/OFF con indicador a LED.
- Conector de fuente externa entre 6Vcd y 8Vcd.

**NOTA:** No conectar un voltaje mayor a 8Vdc como fuente externa y verificar bien la polaridad al conectar.

- Selección, vía puentes, del voltaje de alimentación entre las siguientes fuentes:
  - Desde el MULTILINK embebido.
  - Desde fuente externa (ver punto anterior).
  - Desde el conector Mini AB.
  - Desde el conector a puertos I/O.
- Pulsador de RESET e indicador a LED.
- USB a conector Mini AB.
- Módulo CAN.
- Acelerómetro de tres ejes.
- Ocho LED's de usuario.
- Cuatro pulsadores de usuario.
- Un parlante piezo-eléctrico.
- Puerto IIC con *pullups*.
- Un potenciómetro de 10K.

Algunas especificaciones del circuito son:

- Dimensiones: 8.9cms x 10cms.
- Alimentación:
  - Cable USB: 5VCD @ 500mA máximo.
  - Fuente externa: 6VCD a 8VCD con positivo al centro.

La Tabla 8.1 muestra la ubicación, por defecto, de los puentes:

**Tabla 8.1. Establecimiento de puentes por defecto DEMOJM.**

<b>PUENTES</b>	<b>UBICACIÓN</b>
J3	3&4, 7&8
J4	1&2, 3&4
J6	2&3
J7	1&2
J8	1&2, 3&4
J11	1&2
J12	1&2
J13	2&3
J14	2&3
J17	TODOS PUESTOS
J18	2&3
J19	2&3
J20	2&3
J21	1&2, 3&4, 5&6
J24	1&2
J27	1&2, 3&4, 5&6, 7&8
J28	1&2, 3&4
J29	1&2, 3&4
J30	1&2
J31	1&2 3&4
J32	1&2 3&4

La Figura 8.20 muestra la distribución de pines (PINOUT) del conector MCU PORT en el circuito impreso del DEMOJM. La mayoría de los ejercicios harán referencia de conexión sobre este conector.

VDD	1	2	IRQ/TPMCLK
VSS	3	4	RESET
PTE0/TxD1	5	6	BKGD/MS
PTE1/RxD1	7	8	VUSB33
PTG0/KBIP0	9	10	PTB0/MISO2/ADP0
PTG1/KBIP1	11	12	PTB1/MOSI2/ADP1
PTE2/TPM1CH0	13	14	PTB2/SPSCK2/ADP3
PTE3/TPM1CH1	15	16	PTB3/SS2/ADP3
PTE5/MOSI1	17	18	PTB4/KBIP4/ADP4
PTE4/MISO1	19	20	PTB5/KBIP5/ADP5
PTE6/SPSCK1	21	22	PTB6/ADP6
PTE7/SS1	23	24	PTB7/ADP7
PTF0/TPM1CH2	25	26	PTC0/SCL
PTF1/TPM1CH3	27	28	PTC1/SDA
PTF2/TPM1CH4	29	30	PTG2/KBIP6
PTF3/TPM1CH5	31	32	PTG3/KBIP7
VREFH	33	34	PTF4/TPM2CH0
VREFL	35	36	PTF5/TPM2CH1
PTD0/ADP8/ACMP+	37	38	PTC5/RxD2
PTD1/ADP9/ACMP-	39	40	PTC3/TxD2
PTD2/KBIP2ACMPO	41	42	PTG4/XTAL
PTD3/KBIP3/ADP10	43	44	PTG5/EXTAL
PTD4ADP11	45	46	PTA0
PTD5	47	48	PTA1
PTD6	49	50	PTA2
PTD7	51	52	PTA3
PTC2	53	54	PTA4
PTC4	55	56	PTA5
PTC6	57	58	PTF6
NC	59	60	PTF7

**Figura 8.20. Conector MCU PORT**

Para la instalación del DEMOJM es necesario seguir al pie de la letra el aparte 4.5 del documento: DEMOJM\_Board\_User\_Manual.pdf, que se encuentra en el DVD-ROM.

### 8.3. CREACIÓN DE PROYECTOS EN C

A continuación, son listados los pasos mínimos para crear un proyecto en el software CodeWarrior® 6.2. Se sugiere al lector el uso del tutorial, incluido en el menú de arranque del programa, para ampliar el conocimiento sobre el uso del CodeWarrior® 6.2.

#### Paso 1. Verificación del sistema

Es necesario verificar los siguientes requerimientos mínimos para la ejecución del software CodeWarrior® 6.2:

- **Hardware:** PC con 1GHz de velocidad y procesador Pentium® o compatible. Se recomienda 1GB de RAM, CD-ROM y dependiendo del sistema de desarrollo, deberá tener puerto serial DB9 o puerto paralelo o USB.
- **Sistema Operativo:** Microsoft® Windows® 2000, Windows® XP o Windows® Vista™.
- **Espacio en Disco:** 2GB con 400MB en el disco del sistema Windows®.

#### Paso 2. Instalación del CodeWarrior® 6.2

Instalar el CD del CodeWarrior® 6.2 *Development Studio* en el compartimiento del CD-ROM del PC.

El software se ejecuta automáticamente y se deben seguir las instrucciones de instalación. El CodeWarrior® 6.2 *Development Studio* libera hasta 32K de licencia para proyectos con la familia HC(S)08/RS08 y hasta 64 para la familia CodFire® V1.

#### Paso 3. Ejecución del CodeWarrior® 6.2

Para lanzar el programa es necesario seguir la ruta del Windows®: **Start > Programs > Freescale CodeWarrior > CodeWarrior IDE**. El programa comenzará a ejecutarse y la ventana de la Figura 8.21 aparecerá.

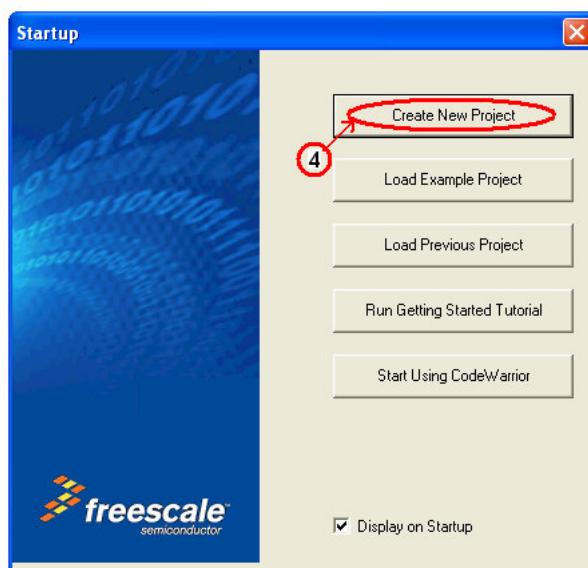
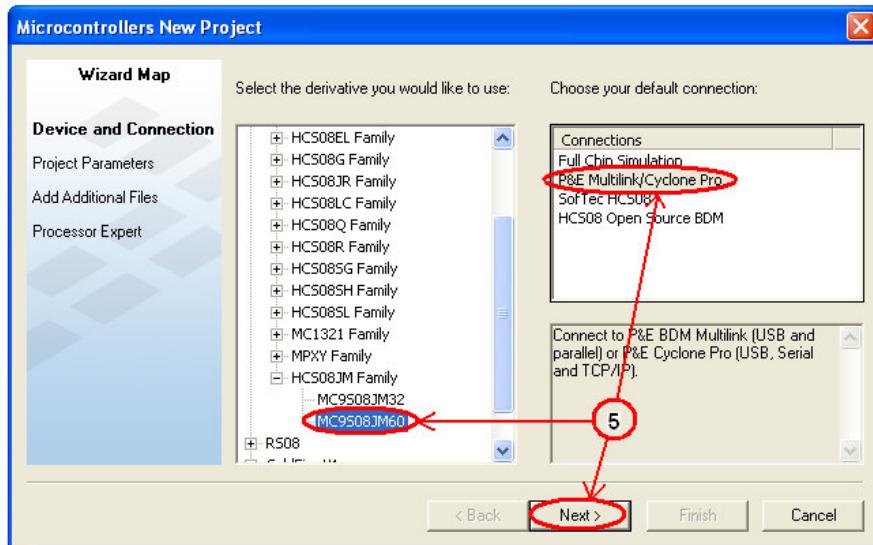


Figura 8.21. Ventana de inicio.

**Paso 4. Creación de un nuevo proyecto**  
Pulsar sobre el botón **Create New Project**.

### **Paso 5. Selección del MCU y la conexión**

La nueva ventana que aparece es la mostrada en la Figura 8.22. En esta ventana seleccionar el microcontrolador a trabajar y la opción de conexión disponible.



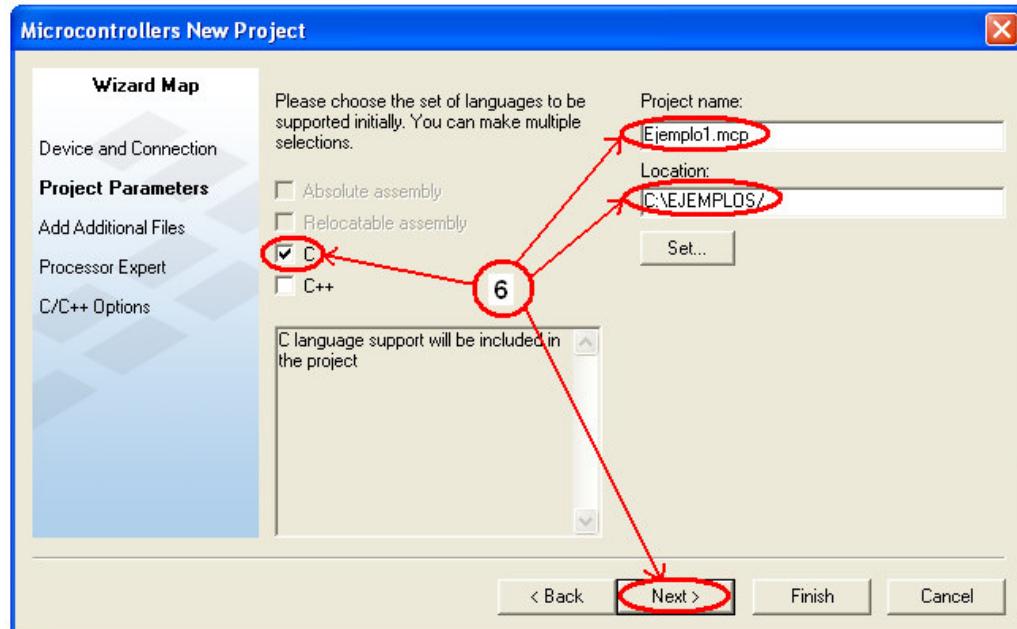
**Figura 8.22. Selección del MCU y la conexión.**

El microcontrolador a trabajar será el **MC908JM60** con la opción de conexión **P&E Multilink/Cyclone Pro**. Presionar el botón de **Next**, para pasar a la siguiente ventana.

### **Paso 6. Selección de parámetros del proyecto**

En la ventana de la Figura 8.23 se detallan: la selección de los tipos de lenguaje a desarrollar, el nombre del proyecto y la ruta de almacenamiento del mismo.

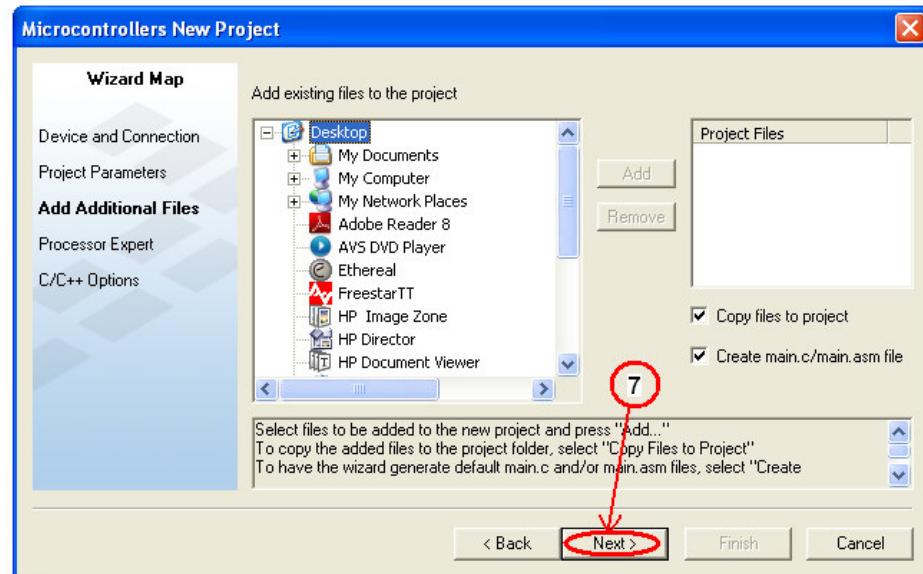
Para este ejercicio en particular, seleccionar como lenguaje el C. Finalmente presionar el botón de **Next**.



**Figura 8.23. Selección del lenguaje y otros.**

#### Paso 7. Opción de adición de archivos

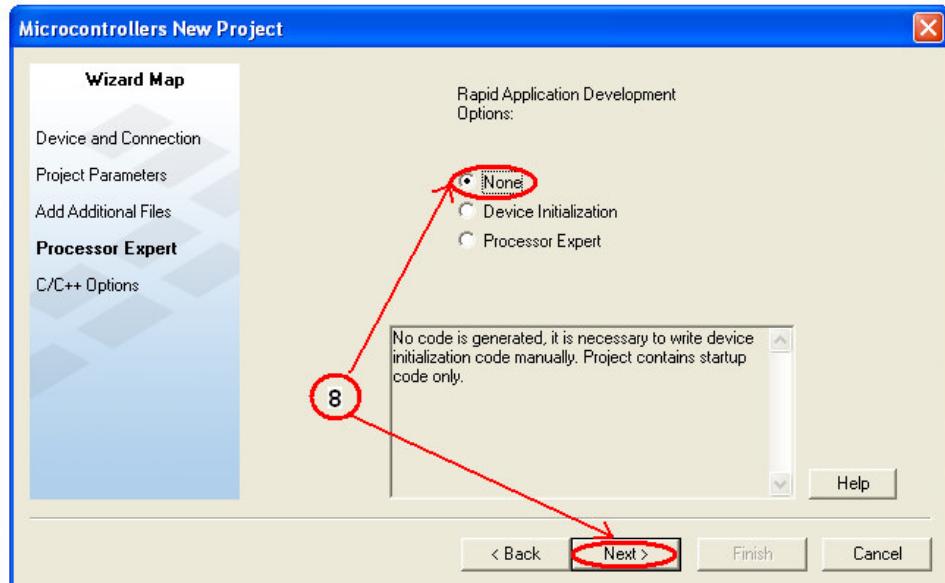
Esta opción de ventana (ver Figura 8.24) no es de interés para el ejercicio, por este motivo se sobre pasa. Presionar el botón de **Next**.



**Figura 8.24. Ventana poco importante para el ejercicio.**

#### Paso 8. Opción de Processor Expert

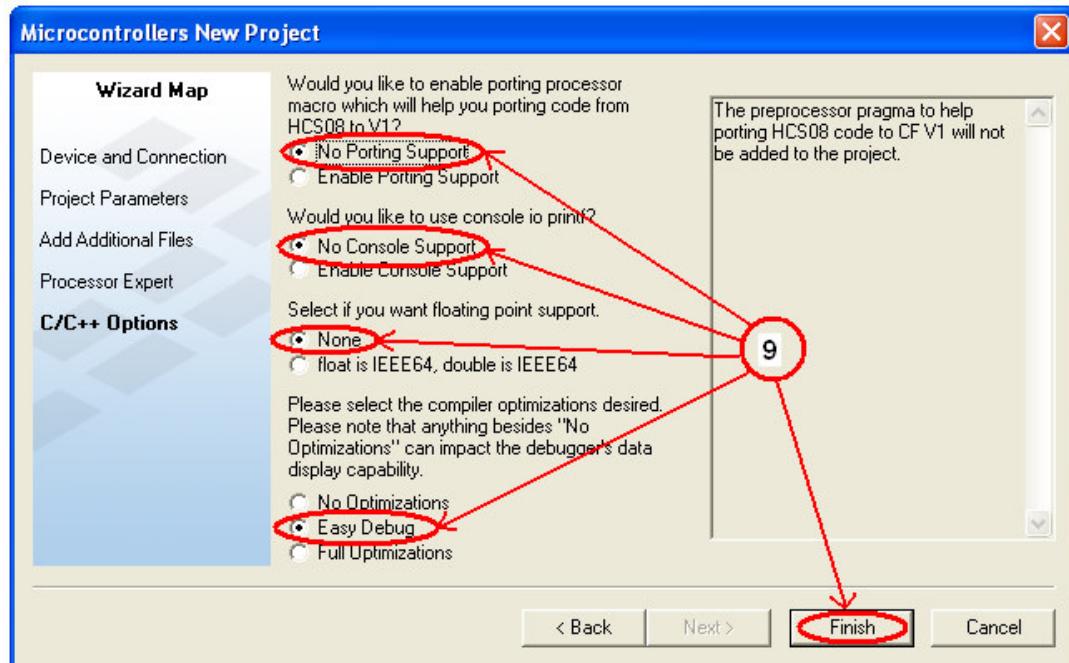
Esta opción de ventana (ver Figura 8.25) no es de interés para el ejercicio, por este motivo se sobre pasa. Seleccionar la opción **none** y presionar el botón de **Next**.



**Figura 8.25.** Ventana poco importante para el ejercicio.

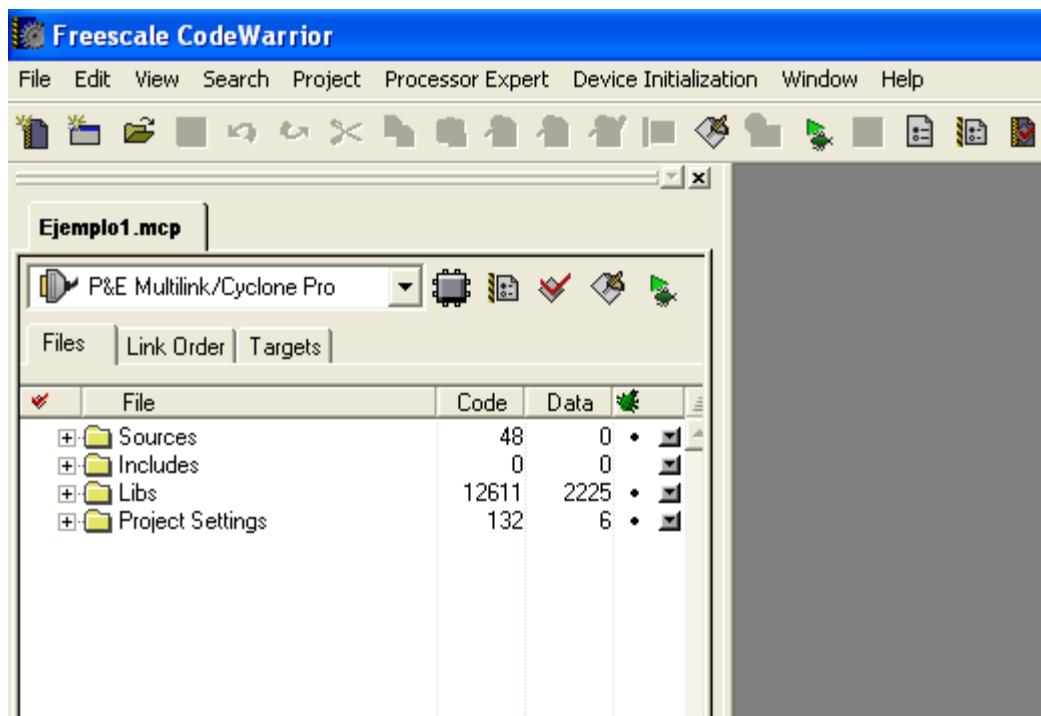
### Paso 9. Opciones de configuración para el C/C++

En esta opción verificar la selección de la Figura 8.26 y presionar el botón de **Finish**.



**Figura 8.26.** Ventana poco importante para el ejercicio.

Finalmente, aparece la ventana del menú principal del CodeWarrior® 6.2 *Development Studio* y la ventana de navegación del proyecto (ver Figura 8.27).



**Figura 8.27. Ventana principal del CodeWarrior® 6.2 Development Studio**

Haciendo doble pulsación sobre el programa **main.c** aparece el cuerpo inicial de cualquier proyecto, que se inicie desde cero y cuyo código es mostrado en la Figura 8.28.

Este cuerpo sólo incluye:

- **hdef.h:** Macro para el proceso de interrupciones.
- **derivative.h:** Declaración de los periféricos de MCU.
- **Función main:** Programa principal del cuerpo en C. Aquí es ejecutado el macro **enable interrupts**, que consiste en poner la máscara I en ceros, con el propósito de habilitar de manera global las interrupciones del sistema. Dentro de la función **main**, se espera que el programador incluya su código. También es ejecutada una instrucción de **for** infinito, que puede ser utilizada como iteración principal del proceso y que es necesaria desde el punto de vista de la acción cíclica que ejecuta un proceso desarrollado con un MCU. Dentro del **for** se ejecuta un macro llamado **\_RESET\_WATCHDOG**, con el objetivo de eliminar la acción del COP y aislar la MCU de un evento de RESET por sobreflujo del temporizador del COP.

En el siguiente aparte realizaremos un primer programa en C, donde se incluye:

- Trabajo sobre el editor IDE.
- Proceso de compilación.
- Simulación del proyecto.
- Bajar el programa al DEMOJM.
- Migrar el proyecto.

```
#include <hidef.h> /* for EnableInterrupts macro */
#include "derivative.h" /* include peripheral declarations */

void main(void) {
    EnableInterrupts; /* enable interrupts */
    /* include your code here */

    for(;;) {
        __RESET_WATCHDOG(); /* feeds the dog */
    } /* loop forever */
    /* please make sure that you never leave main */
}
```

**Figura 8.28. Programa inicial por defecto**

#### 8.4. EL PRIMER PROGRAMA EN C

El primer programa en lenguaje C ha sido llamado Ejemplo1 y coincide con la creación del proyecto del aparte anterior. El programa es desarrollado sobre el microcontrolador de 8 bits MC9S08JM60, para luego ser migrado al microcontrolador de 32 bits MCF51JM128. Este ejemplo también se encuentra en el sitio WEB referido en el libro.

Para el ejercicio es importante verificar que la JM DAUGHTER CARD corresponda con la máquina de 8 bits MC9S08JM60.

El sistema enciende y apaga un LED conectado al pin PTE7 del DEMOJM. Este ejemplo es realizado con la menor complejidad posible, con el objetivo de que el lector de poca experiencia se vaya adaptando a la filosofía de programación de este texto.

Se pretende ilustrar al lector sobre la creación de un primer proyecto y no se profundiza sobre las técnicas de desarrollo de programas en C para sistemas embebidos con microcontroladores.

- **Codificación:** En el menú principal del CodeWarrior® y como se explico en el aparte 8.1, realizar la secuencia de pasos: **file > Open > Ejemplo1.mcp** (creado en el aparte anterior).

Una vez se haya cargado el proyecto, abrir en la ventana de navegación del proyecto el archivo **main.c**, que se encuentra dentro de la carpeta **Sources** y reemplazar el código que hay por el siguiente código:

**NOTA:** Se sugiere que el lector digite el siguiente código con el fin de familiarizarse con el IDE del CodeWarrior®. Si el lector ya tiene un nivel avanzado puede bajar el proyecto desde el sitio WEB del texto.

```
/*********************************************************/
/* Ejemplo 1: Encendido de LED */
/* main.c */
/* Fecha: Mayo 2, 2008 */
/* V1.0, Rev 0 por Diego Munera */
/* Asunto: */
/* Implementar un codigo en C */
/* que realice encendido y apagado */
/* periodico del led en PTD2 de la */
/* tarjeta de evaluacion DEMOJM */
/* */

/* NOTA: Las tildes en los comentarios */
/* son omitidas, para efectos de */
/* compatibilidad con otros editores */
/*********************************************************/

/* Archivos de cabecera */
#include <hidef.h> // macro para habilitar interrupciones
#include "derivative.h" // declaracion de perifericos

/* Macros y definiciones */
#define Disable_COP() SOPT1 &= 0x3F //deshabilita el COP
#define Led1_On() PTDD_PTDD2 = 0; PTDDD_PTDDD2 = 1 //macro para encendido del LED PTD2
#define Led1_Off() PTDD_PTDD2 = 1; PTDDD_PTDDD2 = 0 //macro para apagado del LED PTD2

/* Funcion Delay(): Retarda basado en una variable tipo entera*/
void Delay(void){
unsigned int i;
i= 30000; //inicializa contador i en 30000
while(i > 0){ //llego a cero?
i--; //no --> decrementa
}

/* main(): Funcion principal*/
void main(void) {
MCGC1=0x04; //Reloj en modo FEI y divisor por 1
MCGC2=0x00;
Disable_COP(); //deshabilita el COP
EnableInterrupts(); //habilita interrupciones
for(;;){ //iteracion infinita
Led1_On(); //enciende el Led1
Delay(); //llama funcion de retardo: Delay
Led1_Off(); //apaga el Led1
Delay(); //llama funcion de retardo: Delay
}
/* es necesario asegurarse de nunca abandonar el main */
}
```



- **Compilación:** Para compilar el proyecto se debe presionar el botón . El proyecto no debe generar errores, pero en caso de haberlos es necesario eliminarlos verificando la sintaxis del programa.
- **La Depuración:** Una vez que el proyecto se ha compilado satisfactoriamente, se procede a realizar su depuración. El código fuente es enviado al microcontrolador y comienza su ejecución. Para supervisar el comportamiento del programa es posible ejecutarlo paso a paso o de manera normal, estableciendo puntos de ruptura (*Break Points*).

Para iniciar la depuración es necesario ir al menú **Project→Debug**, o bien de forma rápida mediante el botón . El usuario puede seleccionar entre:

- **Full Chip Simulation:** En la cual no se requiere de una conexión al microcontrolador (simulación en frío) y toda la depuración se realiza en el mismo PC, sin embargo solo es recomendable para secciones de código en las cuales solo interfieren datos y no hardware externo.
- **P&E Multilink/Cyclone Pro:** El código maquina es programado en el microcontrolador y su ejecución se realizará directamente en la maquina final. Esta es la opción recomendada, porque permite el 100% de interacción con el hardware. La simulación es completamente real y la lectura y escritura sobre los pines del microcontrolador se realizan de la misma forma como se ejecutará una vez el sistema de evaluación sea desconectado del PC.

El menú de botones para la depuración es como sigue:



**Run:** Realiza la ejecución desde la función **main()**, en el caso de una depuración con conexión (*in circuit*) la ejecución en el microcontrolador se hará en tiempo real y de forma continua. El procesamiento puede ser parado en cualquier momento mediante el botón (**halt**) .



**Step-In:** Realiza la ejecución de una línea simple de C. En el caso del llamado a una función, lo realiza y se posiciona en la primera línea de la función. Este botón es útil para evaluar el comportamiento de algún procedimiento de forma detallada.



**Step-Over:** Realiza la ejecución de una línea de C, sin embargo si la línea corresponde a un llamado de función, esta será invocada y su ejecución no será intervenida. La función se ejecutará en tiempo real y el PC detiene su ejecución al retornar de la misma.



**Step-Out:** Realiza ejecución hasta que el procesador abandone el procedimiento o función actual. Es útil cuando se ingresa a ejecución detallada (*Step-In*) de una función y pretendiéndose realizar la ejecución de las demás líneas de la función, hay un abandono de esta y el depurador regresa a la línea siguiente en proceso de simulación.



**Single-Step:** Ejecuta la instrucción en *assembler* que está siendo apuntada por el contador de programa (PC). Este botón es útil cuando se requiere conocer el comportamiento del software a nivel de *assembler* y sus efectos sobre las variables, la memoria y los puertos de entrada y salida.

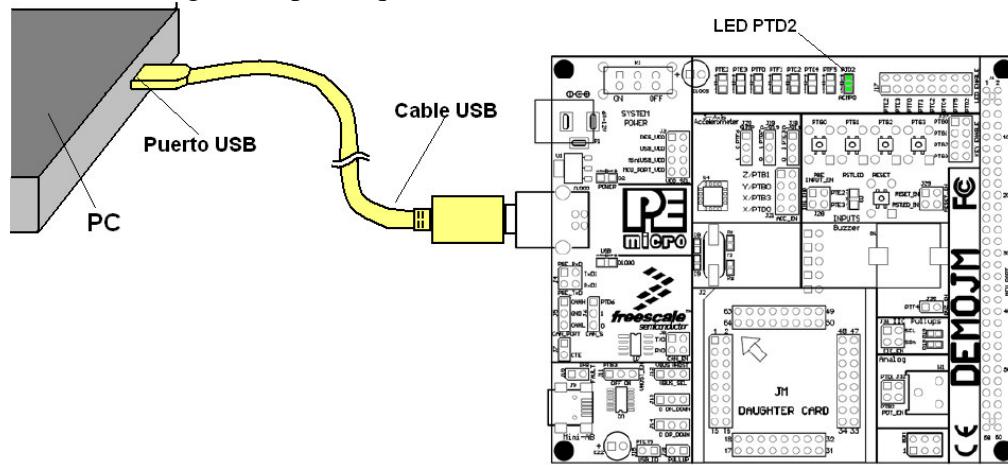


**Halt:** Obliga al microcontrolador a detenerse en la posición en la que se encuentra en contador de programa (PC) y actualizar las sub-ventanas del depurador.



**Reset:** Genera un reset al microcontrolador, obligando al contador de programa (PC) a posicionarse en la función de inicio **Startup ()** o bien a la apuntada por su vector de RESET (0x0000\_0004).

Para proceder a la depuración es necesario establecer la conexión de la Figura 8.29. El PC deberá instalar automáticamente el hardware del DEMOJM y una vez reconocido seguir los pasos que se detallan a continuación:



**Figura 8.29. Conexión DEMOJM - PC**

**Paso 1:** En el menú principal del CodeWarrior® 6.2 hacer la selección de la opción **P&E Multilink/Cyclone Pro**, que se encuentra en la ventana de navegación del proyecto **Ejemplo1.mcp**.

**Paso 2:** Presionar el botón  de esta manera el programa entrará al depurador y la ventana de la Figura 9.30 mostrará la configuración de la conexión establecida.

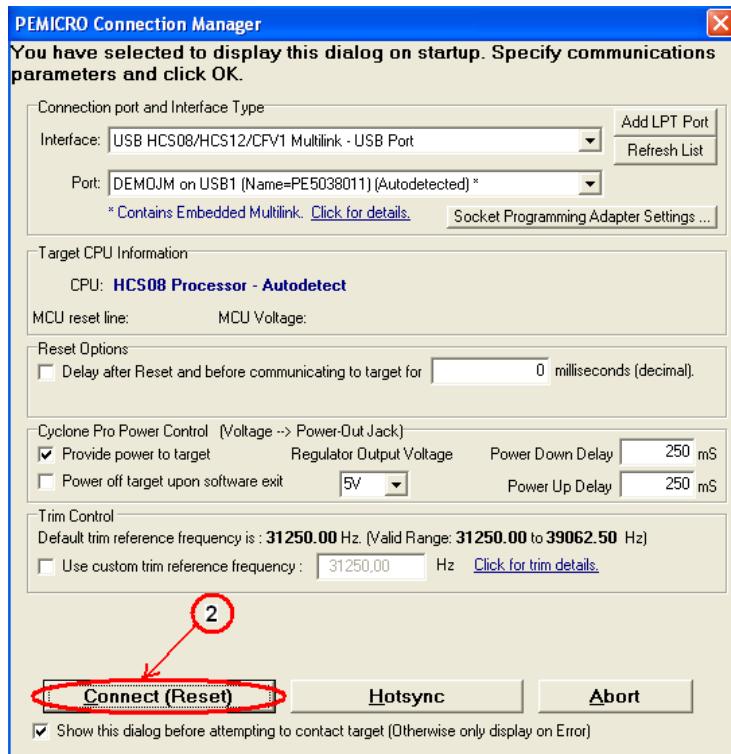


Figura 8.30. Ventana conexión con el DEMOJM

**Paso 3:** Presionar el botón **Connect (Reset)** para establecer la conexión con el depurador. La Figura 8.31 muestra una segunda ventana como ingreso al depurador. Esta ventana indica que se va a borrar la FLASH del microcontrolador y ante esto se debe elegir la opción **Yes**.

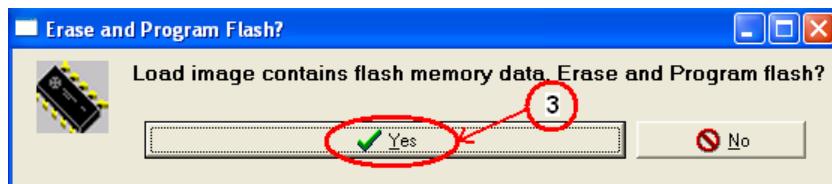


Figura 8.31. Ventana para borrado de la FLASH

**Paso 4:** A esta altura, la ventana del depurador deberá estar presente. Con el botón derecho del ratón, sobre la ventana **Assembly** del depurador, elegir la opción: **Display > Simbolic**.

Observe que la primera instrucción en C a ser ejecutada es **Disable\_COP()**; que equivale a las instrucciones en *assembler*:

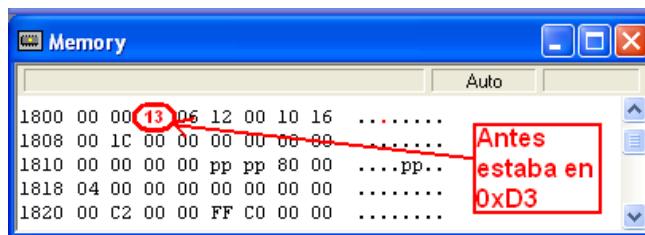
<b>LDHX</b>	<b>#SOPT1.Byte</b>
<b>LDA</b>	<b>,X</b>
<b>AND</b>	<b>#0x3F</b>
<b>STA</b>	<b>,X</b>

Sea que se ejecute en pasos de *assembler* o saltándose la instrucción en C, sería interesante observar la acción de la función **Disable\_COP()** sobre la dirección de memoria 0x1802 (SOPT1). Para lo anterior se debe hacer presión sobre el botón derecho del ratón en la ventana **Memory**, elegir la opción **Address** y digitar el valor que se indica en la Figura 8.32.



**Figura 8.32. Ventana visualizar dirección en memoria**

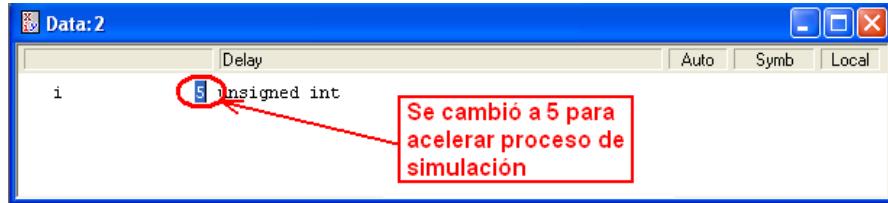
En la Figura 8.33 se puede observar que la celda de memoria 1802 cambiará su valor de 0xD3 a 0x13, con lo que se verifica la inhibición de la acción del COP.



**Figura 8.33. Detalle inhibición del COP**

**Paso 5:** De igual manera se puede verificar la acción del macro **EnableInterrupts()**. Avanzar hasta la función **Led1\_On()** y verificar el encendido del LED PTD2 en la *board* DEMOJM.

**Paso 6:** Para la función **Delay()**, se recomienda ejecutarla paso a paso y verificar el estado de la variable de control del retardo **i**. El estado de esta variable se puede visualizar en la ventana **Data: 2** y comprobar su decremento a medida que se ejecutan pasos de simulación. El usuario puede cambiar el estado de **i**, para acelerar el proceso de la simulación, haciendo doble pulsación con el ratón sobre esta (ver Figura 8.34).



**Figura 8.34. Cambio del valor de una variable**

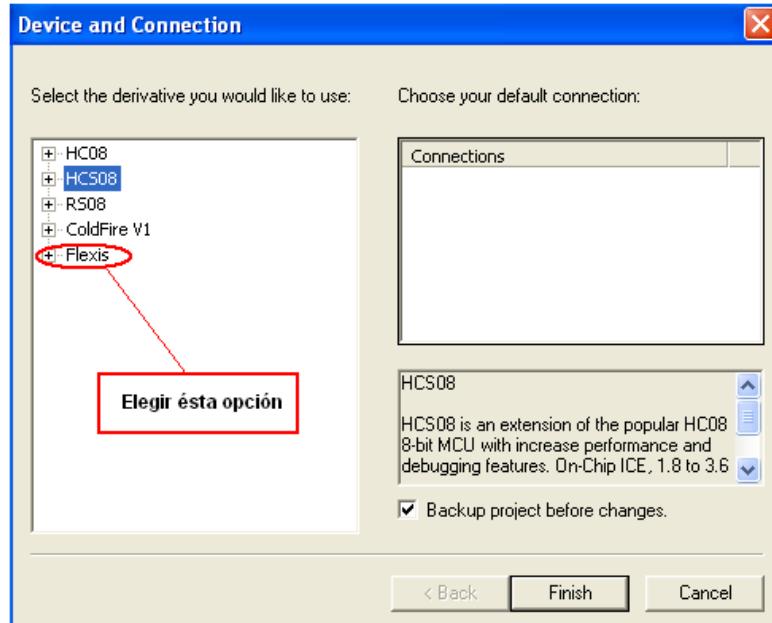
Comprobar la activación y desactivación retardada del LED ejecutando las funciones **Led1\_On()**, **Delay()** y **Led1\_Off()**.

- **Bajar el programa al DEMOJM:** En este momento, el programa ya se encuentra almacenado en la FLASH del sistema y está listo para ser ejecutado.

**Paso 7:** Poner en RUN el sistema, pulsando el ícono y verificar la activación y desactivación intermitente del LED, a una frecuencia aproximada de 12Hz, reportando un reloj de bus interno de aproximadamente 16 MHz. Si el usuario desea, podría salir de la ventana del depurador y verificar que el sistema quedó programado y listo para funcionar desde RESET.

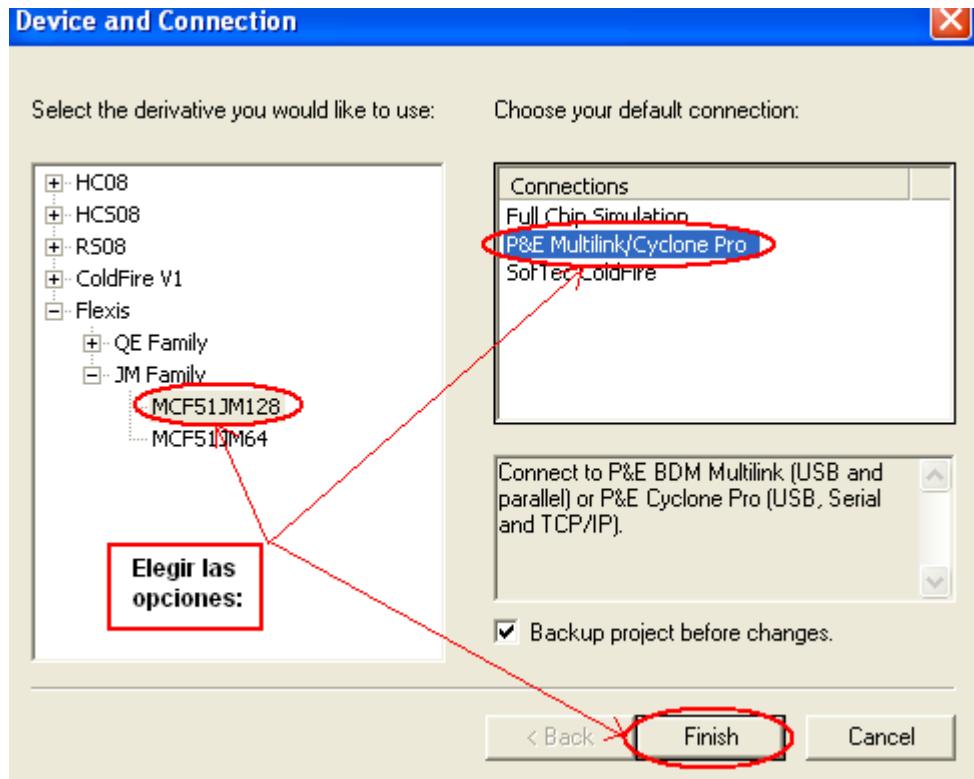
- **Migrar el proyecto a 32 bits:** Los siguientes pasos deben seguirse para migrar el proyecto de un microcontrolador a otro, pero el usuario deberá tener en cuenta las consideraciones expuestas en el Capítulo 7 aparte 7.3.

**Paso 1:** Seleccionar el menú: **Project/Change MCU/Connection** y la siguiente ventana aparecerá (ver Figura 8.35).



**Figura 8.35. Elegir la opción FLEXIS.**

**Paso 2:** Seleccionar el microcontrolador a migrar MCF51JM128 como se indica en la Figura 8.36.



**Figura 8.36. Paso final de la migración.**

**Paso 3:** Compilar nuevamente el proyecto con la opción: .

**Paso 4:** Cambiar la JM DAUGHTER CARD por la máquina MCF51JM128, teniendo la precaución de retirar la alimentación de la DEMOJM.

**Paso 5:** Energizar la DEMOJM y proceder a bajar el programa como se explicó en los pasos anteriores

Se puede observar una frecuencia de apagado/encendido del LED mayor a la obtenida con la máquina de 8 bits MC9S08JM60.

**NOTA:** Con la elaboración de este ejercicio el usuario se puede dar cuenta de la potencia de máquina de 32 bits (MCF51JM128) sobre una máquina de 8 bits (MC9S08JM60), aún trabajando a la mitad de la velocidad del bus interno.

## **8.5. REFERENCIAS**

- CodeWarrior® 6.2, help menú.
- DEMOJM\_Board\_User\_Manual.pdf

## **8.6. PREGUNTAS**

- ¿En qué consiste la característica de *Full Simulation* del CodeWarrior?
- ¿Cuántas y cuáles son las formas en las que se puede ejecutar un código, desde el punto de vista del depurador?
- ¿Cuál es la especificación para la fuente externa de alimentación del DEMOJM?
- Diseñar un programa, sobre la plataforma de 8 bits MC9S08JM60, que rote un “1” a través de los LED’s: PTE2, PTE3, PTF0, PTF1, PTC2, PTC4, PTF5 y PTD2, de la DEMOJM. El intervalo de rote deberá ser de 500 ms.
- Para el programa anterior, realizar la migración a la máquina MCF51JM128 con los ajustes necesarios, para lograr la misma frecuencia de rote de 500 ms.

# PARTE IV

## EL MICROCONTROLADOR DE 32 BITS MCF51JM128

### OBJETIVO

Hacer una presentación introductoria a la máquina MCF51JM128 desde sus generalidades hasta la descripción uno a uno de sus módulos más importantes.

**NOTA:** Se recomienda al lector ampliar los conceptos aquí expuestos con la lectura juiciosa del manual de referencia **MCF51JM128\_Reference\_Manual.pdf**.

### CONTENIDOS

#### CAPÍTULO 9. Generalidades de la máquina MCF51JM128

En este Capítulo es presentada la arquitectura de la máquina MCF51JM128, así como la distribución de pines, la composición del mapa de memoria y los modos de operación en bajo consumo.

#### CAPÍTULO 10. El RESET, la máquina de excepciones y el controlador de interrupciones (CF1\_INTC: *ColdFire VI Interrupt Controller*)

Se hace especial énfasis en la máquina de estados que controla los procesos de excepción y a manera de ejercicio es elaborado un sistema para atender un proceso de excepción por interrupción del pin de IRQ.

#### CAPÍTULO 11. Reloj del sistema (MCG: *Multipurpose Clock Generator*)

Las opciones de configuración del reloj de trabajo del MCU son múltiples y es por esto que en este Capítulo se tratan en detalle y se presentan varios ejemplos de configuración.

#### CAPÍTULO 12. Los puertos de entrada y salida

El Capítulo 14 trata sobre el manejo estándar de los puertos I/O de la máquina. Este Capítulo es importante porque en este aparece un ejemplo de migración de la máquina MC9S08JM60 a la máquina MCF51JM128.

#### CAPÍTULO 13. Funciones de temporización (TPM: *Timer/PWM Module* y RTC: *Real Time Counter*)

Capítulo con una detallada explicación de los procesos de temporización y un ejercicio por cada variación en la aplicación del módulo TPM.

#### CAPÍTULO 14. Conversión A/D (ADC: *Analog to Digital Converter*)

Con una completa explicación de los aspectos relacionados con el proceso de conversión análoga a digital, este Capítulo presenta la aplicación del módulo ADC el cual posee una moderada resolución y una velocidad de conversión muy atractiva para procesos industriales, procesos de automática y de la industria automotriz.

### **CAPÍTULO 15. Comparación análoga (ACMP: *Analog Comparator*)**

Explicación del funcionamiento y puesta en marcha de un novedoso módulo como el ACMP.

### **CAPÍTULOS 16 y 17. Comunicaciones seriales asíncronas (SCI: *Serial Communication Interface*) y Comunicaciones seriales sincrónicas (SPI: *Serial Peripheral Interface* e IIC: *Inter Integrated Circuit*)**

En estos capítulos se hace un tratamiento básico de los módulos de comunicaciones seriales más típicos de los microcontroladores de actualidad, también son presentados ejercicios que complementan la base teórica de cada puerto de comunicaciones

### **CAPÍTULO 18. Comunicación serial universal (USB 2.0: *Universal Serial Bus*)**

Se hace una introducción al estándar universal de comunicaciones seriales y no se profundiza debido a la complejidad de este sistema.

### **CAPÍTULO 19. Otros módulos del MCF51JM128**

Módulos adicionales que no serán tratados con profundidad, pero que se explica su función primaria.

**NOTA:** En un futuro texto, con características de avanzado, se incluirán módulos faltantes, la herramienta *processor expert*, sistema operativo en tiempo real (RTOS) y aplicaciones con módulos más complejos como el USB y el CAN.

# CAPÍTULO 9

## Generalidades de la máquina MCF51JM128

- 9.1. Diagrama en bloques**
- 9.2. Distribución de pines**
- 9.3. Distribución de la memoria**
- 9.4. Modos de bajo consumo**
- 9.5. Referencias**
- 9.6. Preguntas**

## 9.1. DIAGRAMA EN BLOQUES

El MCF51JM128 es un microcontrolador miembro de la familia ColdFire® V1, que ha sido potenciado con las siguientes componentes y/o características internas:

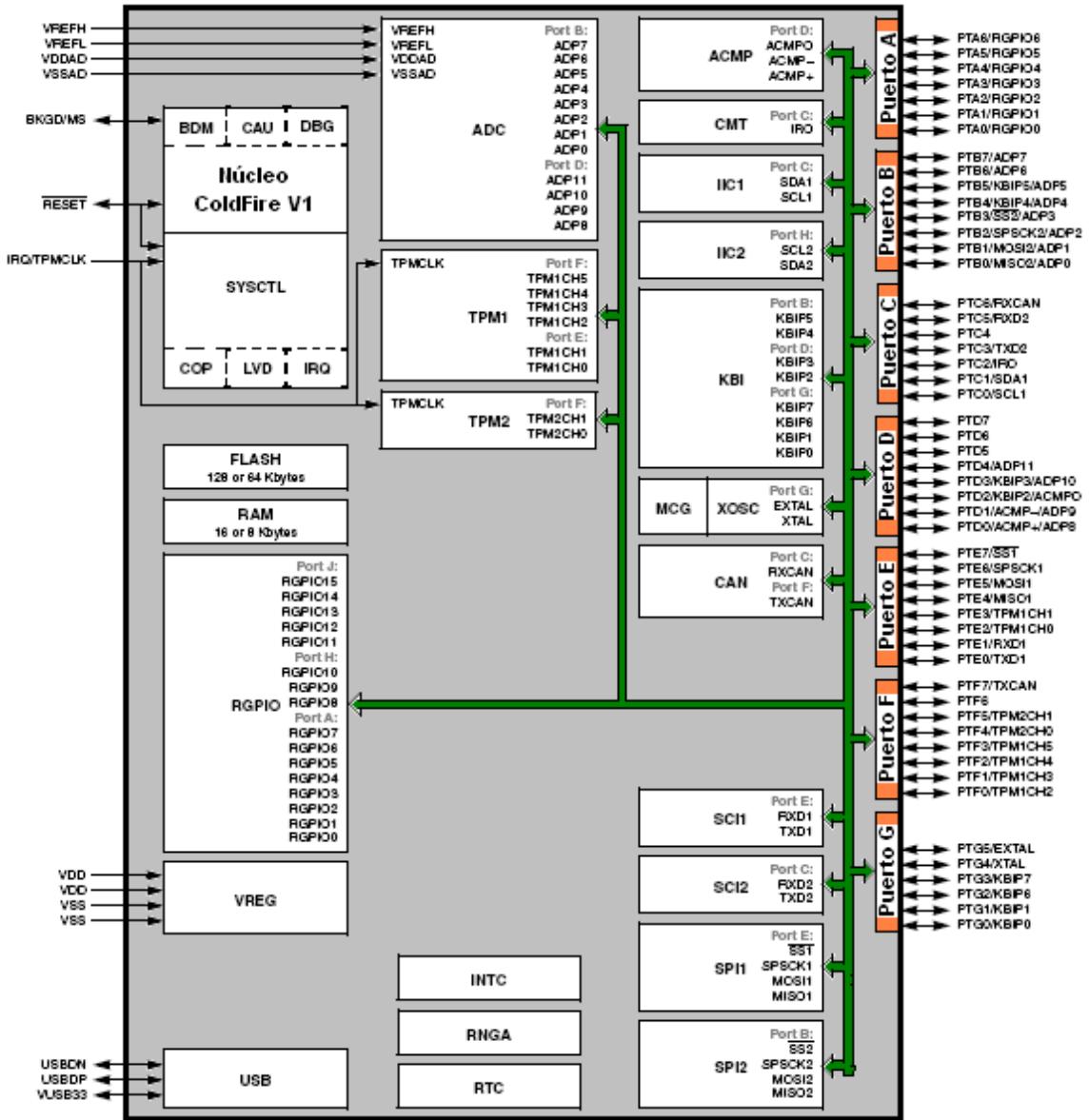
- Núcleo ColdFire® V1 en arquitectura RISC, con puerto para depuración y programación BDM (*Background Debug Module*).
- Migra desde o hacia la máquina de 8 bits MC9S08JM60.
- Velocidad de reloj de hasta 55.33 MHz.
- Hasta 128KB de memoria FLASH.
- Hasta 16KB de memoria SRAM.
- Generación de reloj interno.
- Puerto USB-OTG (USB *On The Go*).
- Puerto CAN (*Controller Area Network*).
- Unidad aceleración criptográfica.
- Módulo generador de números aleatorios.
- Comparadores análogos.
- Hasta 12 conversores A/D de 12 bits.
- Dos puertos IIC (*Inter Integrated Circuit*).
- Dos puertos SPI (*Serial Peripheral Interface*).
- Dos puertos SCI (*Serial Communication Interface*).
- Sistema temporizado para modulación de señal portadora CMT (*Carrier Modulation Timer*).
- Hasta 8 canales temporizadores/moduladores de ancho de pulso TPM (*Timer PWM Module*).
- Contador de tiempo real RTC (*Real Time Counter*).
- Hasta 51/66 pines I/O de propósito general en versiones de 64/80 pines respectivamente.
- Interface para teclado KBI (*KeyBoard Interrupts*).

La Figura 9.1 ilustra un diagrama en bloques de las componentes internas del procesador El MCF51JM128, para empaque LQFP/QFP.

## 11.2. DISTRIBUCIÓN DE PINES Y CONEXIÓN MÍNIMA

Freescale ha diseñado máquinas en empaques muy apropiados y a costos muy razonables, pero todavía falta hacer una asignación más coherente sobre la ubicación de los pines en el circuito integrado (C.I.). Se entiende que estas asignaciones, de pines, actuales obedecen a criterios de inmunidad del C.I. a la presencia de EMI (*ElectroMagnetic Interferences*), más sin embargo sus consumidores siguen esperando una distribución más cómoda con los beneficios actuales de EMC (*ElectroMagnetic Compatibility*).

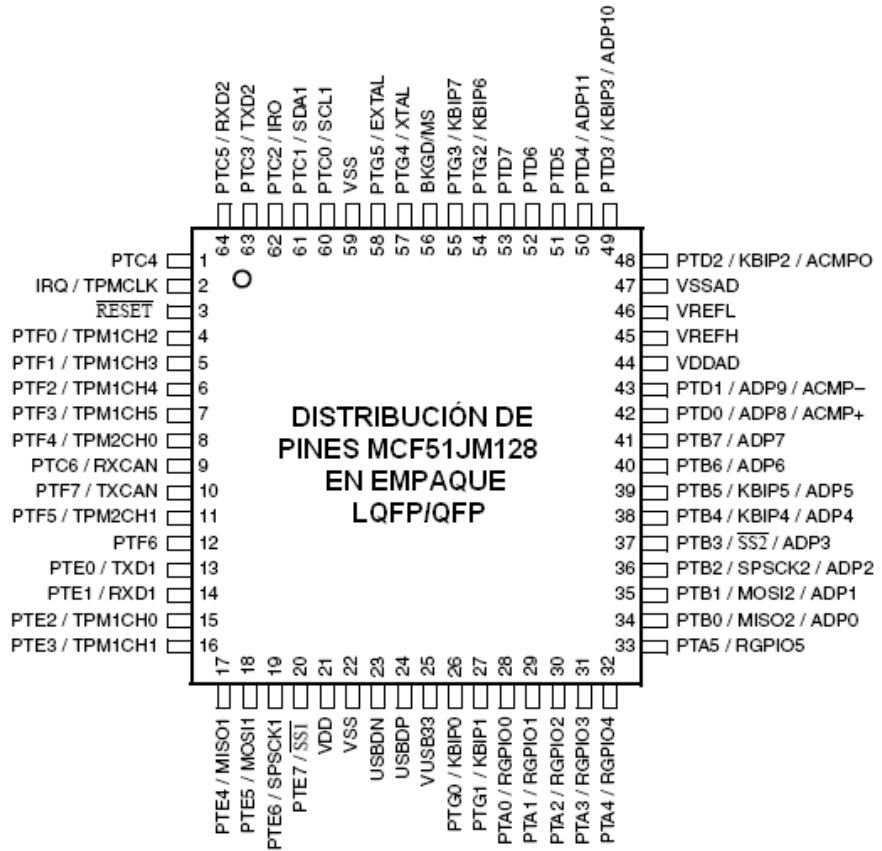
La Figura 9.2 muestra la distribución de pines para la máquina MCF51JM128 en empaque LQFP, que es la máquina sugerida para trabajar en este texto.



**Figura 9.1. Diagrama en bloques del MCF51JM128 LQFP/QFP**

La conexión mínima para hacer desarrollo, utilizando esta máquina es presentada en la Figura 9.3. Esta conexión permite establecer comunicación BDM, con el propósito de hacer depuración y programación del C.I.

En esta conexión son opcionales un reloj externo a cristal, el RESET manual y la conexión a puerto USB.



**Figura 9.2. Distribución de pines MCF51JM128 LQFP/QFP**

### 9.3. DISTRIBUCIÓN DE LA MEMORIA

Las secciones de memoria son del tipo SRAM, FLASH y memoria de registros para los puertos I/O y configuración y estado.

Existen áreas de memoria reservadas para microcontroladores futuros en la familia ColdFire® V1, que se representan en la Figura 9.4 como rectángulos sombreados. La direcciones en el mapa de memoria son representadas por números hexadecimales, como 0x(FF)FF\_8000. Esta representación indica que la máquina MCF51JM128 tiene 24 líneas efectivas para los direccionamientos de la memoria, por lo tanto los 8 primeros bits de la dirección se representan entre ( ). No obstante, se conserva la representación en 32 bits para hacer referencia a máquinas futuras o a máquinas de versiones superiores.

Dependiendo del tipo de dato a manipular (*byte*, *word*, *longword*), es necesario atender la especificación dada por la Tabla 9.1. Allí es importante destacar la restricción que existe para la memoria FLASH sobre la escritura de datos tipo *byte* y *word*.

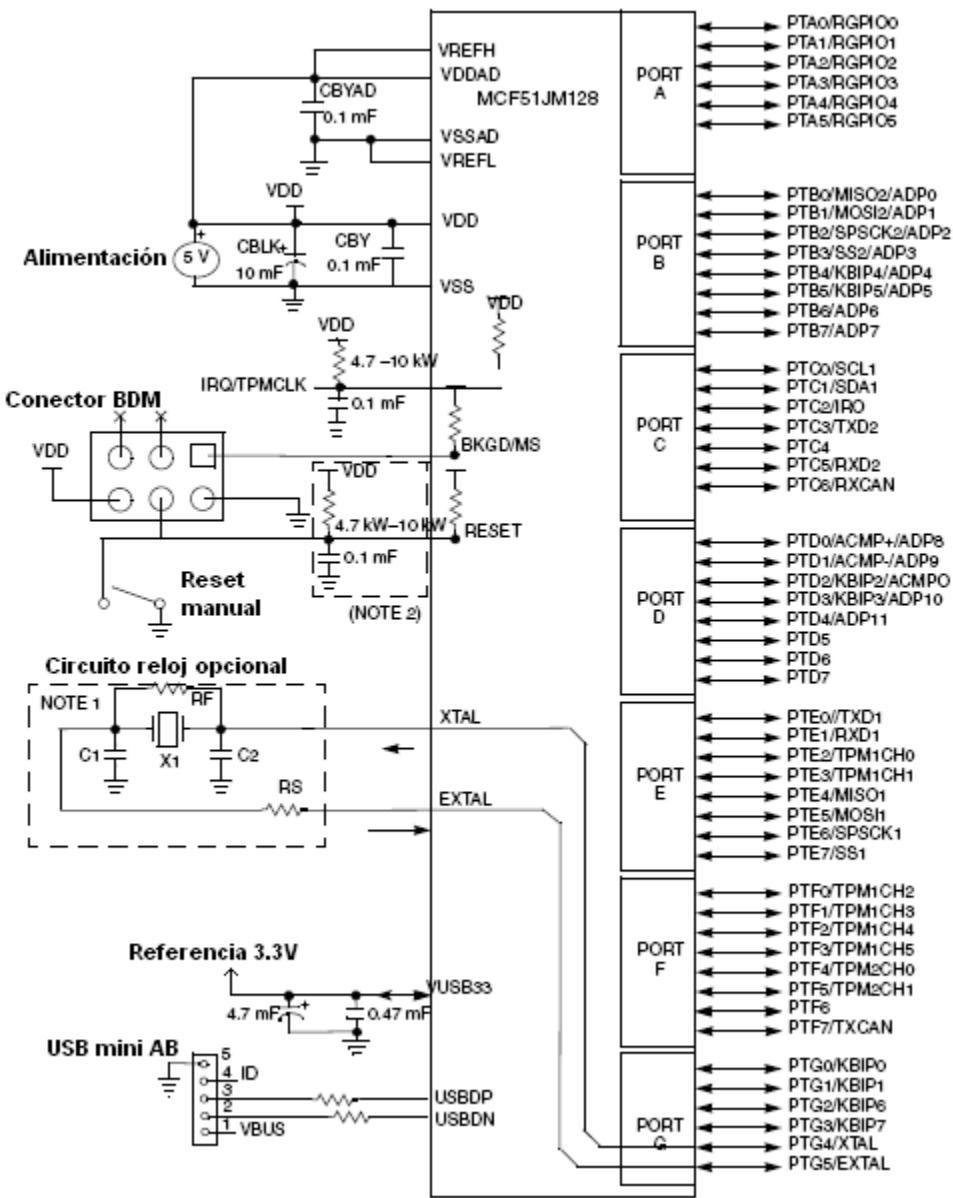


Figura 9.3. Conexión mínima para el MCF51JM128

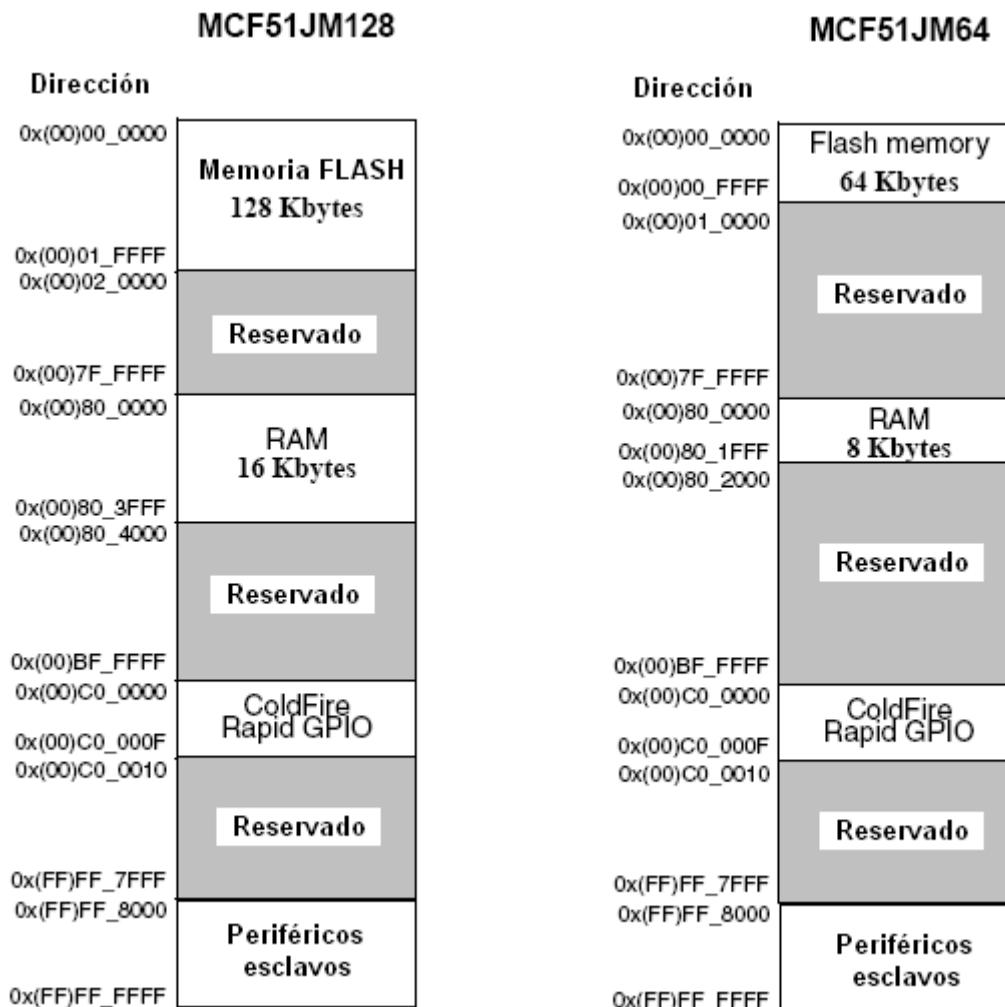
Los registros de configuración de periféricos se encuentran diseminados en tres grupos así:

0x(FF)FF\_8000 – 0x(FF)FF\_807F Registro de periféricos en página directa  
 0x(FF)FF\_9800 – 0x(FF)FF\_98FF Registro de periféricos en la página alta  
 0x(FF)FF\_FFC0 – 0x(FF)FF\_FFFF Controlador de interrupciones

Para los dos primeros grupos, grupos de los periféricos, la máquina suministra un acceso rápido con el modo de direccionamiento absoluto corto.

**Tabla 9.1 Acceso a tipos de dato en memoria.**

Dirección Base	Región	Lectura			Escritura		
		Byte	Word	Long	Byte	Word	Long
0x(00)00_0000	Flash	X	X	X	—	—	X
0x(00)80_0000	RAM	X	X	X	X	X	X
0x(00)C0_0000	Rapid GPIO	X	X	X	X	X	X
0x(FF)FF_8000	Periféricos	X	X	X	X	X	X



**Figura 9.4. Mapa de memoria para la familia JM.**

La Figura 9.5 ilustra la forma como están distribuidos los diferentes registros dentro los grupos asignados para estos en memoria. Debido a lo extenso de la información, sólo se representan algunos registros y las recomendaciones de interpretación de estos. Se recomienda al usuario la lectura del documento: *MCF51JM128\_Reference\_Manual.pdf* (pag: 51-65), para consultarlos de manera más completa.

Los bits (o grupo de bits) que ocupan la celda entera pueden ser de lectura y escritura, por ejemplo PTAD4. Aquellas celdas que aparecen sombreadas, significa que no han sido implementadas o que son espacios reservados para la CPU. Es fijo que estas celdas no se pueden escribir, pero si estas celdas se leen es probable que retornen un “0” o un “1”.

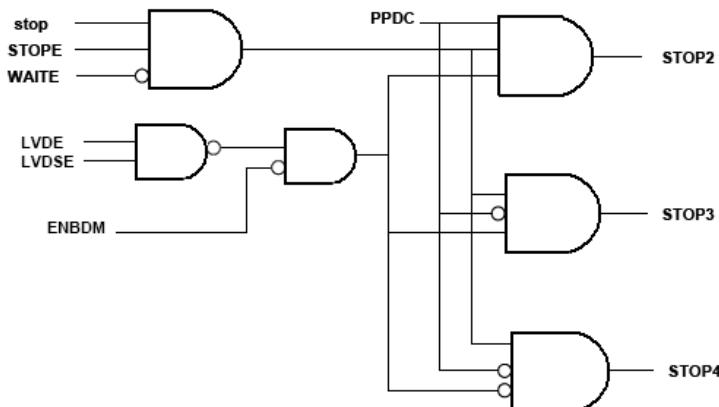
Dirección	Nombre	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8000	PTAD	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0
0x(FF)FF_8001	PTADD	PTADD7	PTADD6	PTADD5	PTADD4	PTADD3	PTADD2	PTADD1	PTADD0
		⋮		⋮		⋮		⋮	
0x(FF)FF_800E	ACMPSC	ACME	ACBGS	ACF	ACIE	ACO	ACOPE	ACMOD1	ACMOD0
0x(FF)FF_800F	Reserved	0	0	0	0	0	0	0	0
0x(FF)FF_8010	ADCSC1	COCO	AIEN	ADCO			ADCH		
0x(FF)FF_8011	ADCSC2	ADACT	ADTRG	ACFE	ACFGT	—	—	—	—
		⋮		⋮		⋮		⋮	

**Figura 9.5. Distribución de registros en memoria.**

En el capítulo 22, aparece con más detalle el módulo para manejo de la FLASH.

#### 9.4. MODOS DE BAJO CONSUMO

En el capítulo 5 (aparte 5.1), se discutieron los diferentes modos de operación de las máquinas ColdFire®. La Figura 9.6 ilustra sobre la máquina de estados que utiliza el procesador para entrar en los diferentes modos de bajo consumo. La arquitectura de la máquina MCF51JM128 cumple con dichos modos, para toda parte que incluya la versión V1. En la Tabla 9.2 se hace un resumen, por módulo, de los modos de bajo consumo para la máquina JM de 32 bits.



**Figura 9.6. Máquina de estados para modos de bajo consumo.**

**Tabla 9.2 Modos de bajo consumo por periféricos.**  
**Fuente: MCF51JM128\_Reference\_Manual.pdf (pag:46,47).**

Peripheral	Mode				
	Stop2	Stop3	Stop4	Wait	RUN
CF1CORE	Off	Standby <sup>1</sup>	Standby	Standby	On
RAM	Standby	Standby	Standby	On	On
Flash	Off	Standby	Standby	On	On
Port I/O Registers	Off	Standby	Standby	On	On
ADC <sup>2</sup>	Off	On	On	On	On
ACMP	Off	On	On	On	On
BDC/BDM	Off	Standby	On	On	On
COP	Off	Standby	Standby	On	On
Crystal Oscillator	Off	On RANGE=0 HGO=0	On All Modes	On All Modes	On
MCG	Off	On <sup>3</sup>	On	On	On
SIM	Off	On	On	On	On
IICx	Off	Standby	Standby	On	On
IRQ	Off (Wake Up via POR) <sup>4</sup>	NoClk (Wake Up)	NoClk (Wake Up)	On	On
KBI	Off	NoClk (Wake Up)	NoClk (Wake Up)	On	On
LVD/LVW	Off	Standby	On (Wake Up)	On	On
RTC	Optionally On if using LPO enabled	On (Wake Up)	On (Wake Up)	On	On
SCIx	Off	Standby	Standby	On	On
SPIx	Off	Standby	Standby	On	On
USB (SIE and Transceiver)	Off	Optionally On <sup>5</sup>	Optionally On <sup>5</sup>	On	On
USB 3.3V Regulator	Off	Optionally On <sup>6</sup>	Optionally On <sup>6</sup>	On	On
MSCAN	Off	Standby	Standby	On	On
TPMx	Off	Standby	Standby	On	On
Voltage Regulator / PMC	Partial Shutdown. 1kHz osc if enabled	On	On	On	On

## **9.5. REFERENCIAS**

- MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual. Freescale Semiconductor. Rev 1, 2008.
- Microcontroller MCF51JM128 ColdFire Data Sheet. Freescale Semiconductor. Rev 0, 2008.

## **9.6. PREGUNTAS**

- ¿Hasta qué velocidad de reloj puede operar la máquina MCF51JM128?
- ¿Cuántos y cuales son los puertos de comunicación de la máquina MCF51JM128?
- ¿Cuántas líneas de dirección maneja el MCF51JM128 y cómo se representa una dirección?
- ¿Qué significa un espacio sombreado en la representación de un registro de la máquina?
- ¿Cuántos y cuáles son los modos de bajo consumo de la máquina MCF51JM128?

# CAPÍTULO 10

## El RESET, la Máquina de Excepciones y el Controlador de Interrupciones

**10.1. Las fuentes de RESET**

**10.2. El controlador de interrupciones**

**10.3. El Pin de Interrupción (IRQ: *Interrupt Request*)**

**10.4. Ejercicio con la IRQ: Atención de una interrupción por flanco en el pin de IRQ**

**10.5. Referencias**

**10.6. Preguntas**

## 10.1. LAS FUENTES DE RESET

Son múltiples las fuentes de RESET que posee la máquina MCF51JM128. Todas estas dejan huella en el registro SRS, que permite al programador poder consultar quién (o quienes) ha sido el causante de un RESET. Las fuentes de RESET de la máquina JM de 32 bits son:

- Reset en el encendido: *Power-on reset* (POR)
- Reset por pin externo: *External pin reset* (PIN)
- Temporizador del computador operando apropiadamente: *Computer operating properly* (COP) *timer*
- Detección de una instrucción ilegal: *Illegal opcode detect* (ILOP)
- Detección de un acceso a dirección ilegal: *Illegal address detect* (ILAD)
- Detección de bajo voltaje: *Low-voltage detect* (LVD)
- Pérdida del enganche del generador de reloj externo: *Clock generator (MCG) loss of clock reset* (LOC)
- Reset por forzado a entrada en modo de depuración: *Background debug forced reset*

A continuación se hace una breve definición de cada una de estas fuentes.

- **Reset en el encendido (POR):** Cuando la máquina está comenzando a ser energizada o cuando el voltaje de alimentación (Vdd) cae por debajo de los niveles mínimos permitidos, el circuito del POR genera un estado de RESET a la máquina. El bit POR del registro SRS se pone a “1” indicando el evento de RESET (ver Figura 10.1).

**Registro SRS**

	7	6	-	4	3	2	1	0
Lectura	POR = 1	PIN	COP	ILOP	ILAD	LOC	LVD	0
Escritura	Escribir algún valor en el SRS, aclara el contador del COP (Watchdog)							

**Figura 10.1. Bit POR como evento causante de RESET.**

- **Reset por pin externo (PIN):** Obedece al flanco de bajada que se presenta como señal externa, aplicada al pin de RESET. El bit PIN del registro SRS se pone a “1” indicando el evento de RESET (ver Figura 10.2).

**Registro SRS**

	7	6	-	4	3	2	1	0
Lectura	POR	PIN = 1	COP	ILOP	ILAD	LOC	LVD	0
Escritura	Escribir algún valor en el SRS, aclara el contador del COP (Watchdog)							

**Figura 10.2. Bit PIN como evento causante de RESET.**

- **Reset por COP (Computer Operating Properly):** Mecanismo equivalente al *Watchdog* de otras máquinas, que detecta la pérdida de flujo del programa y genera un RESET al procesador.

Una vez habilitado el COP, es necesario aclarar periódicamente el contador asociado a este (escribir en el registro SRS algún valor) y así evitar que se genere RESET al sistema. El COP puede ser configurado desde los registros SOPT1 y SOPT2 (como se muestra en la figura 10.3).

**REGISTRO SOPT1**

Lectura	7	6	5	4	3	2	1	0
Escritura	COPT <sup>1</sup>		STOPE <sup>1</sup>	WAITE	0	0	0	0
Reset:	1	1	0	1	0	0	0	0

1: Estos bits sólo puede ser escrito una vez durante la programación

**REGISTRO SOPT2**

Lectura	7	6	5	4	3	2	1	0
Escritura	COPCLKS <sup>1</sup>	COPW <sup>1</sup>	USB_BIGEND	CLKOUT_EN	CMT_CLKSEL	SPI1FE	SPI2FE	ACIC
Reset:	0	0	0	0	0	1	1	0

1: Estos bits sólo puede ser escrito una vez durante la programación

**Figura 10.3. Registros SOPT1, SOPT2 en la configuración del COP.**

**COPT:** Selección del período del COP.

**COPCLKS:** Selección del tipo de reloj que alimenta el contador del COP.

**COPW:** Ventana de activación del COP. Sólo para operación con el BUSCLK y de estar activo, es necesario escribir sobre el SRS durante el 25% final del período del COP seleccionado.

En donde los bits asociados a la operación del COP (COPT, COPCLKS y COPW), trabajan como se indica en la Tabla 10.1.

**Tabla 10.1. Configuración del COP.**

Bits de Control		Fuente de reloj	Ventana del COP (SOPT2[COPW] = 1)	Conteos para el sobreflujo del COP
SOPT2[COPCLKS]	SOPT1[COPT]			
N/A	00	N/A	N/A	COP deshabilitado
0	01	1 kHz LPOCLK	N/A	$2^5$ Ciclos (32 ms )
0	10	1 kHz LPOCLK	N/A	$2^8$ Ciclos (256 ms )
0	11	1 kHz LPOCLK	N/A	$2^{10}$ Ciclos (1,024 ms )
1	01	BUSCLK	6,144 Ciclos	$2^{13}$ Ciclos <sup>1</sup>
1	10	BUSCLK	49,152 Ciclos	$2^{16}$ Ciclos <sup>1</sup>
1	11	BUSCLK	196,608 Ciclos	$2^{18}$ Ciclos <sup>1</sup>

1: Si COPW = 1, el usuario deberá aclarar el contador del COP durante el último 25% del período seleccionado

La señal 1KHz LPOCLK es un reloj de bajo consumo e independiente del reloj del sistema, esto hace que el reloj del *Watchdog* no dependa del reloj de la máquina. El usuario puede optar por el uso del BUSCLK (reloj de bus interno de la MCU), pero debe ser conciente de la posible pérdida del COP debido a la pérdida del reloj del sistema.

El contador del COP inicia una vez se halla escrito sobre los registros SOPT1 y SOPT2. El aclarado del COP (escribiendo sobre el registro SRS) no deberá ser hecho en una atención a interrupción, porque se puede caer en la atención reiterada de la ISR y descuidar el refresco del COP.

Cuando se trabaja con el reloj BUSCLK y se está en modo BDM (*Background Mode Debug*) o mientras el sistema se encuentra en modo STOP, el contador del COP no se incrementa y retiene su valor. Esto se hace una vez se supere el modo BDM o STOP y el COP continúa su conteo.

Cuando se trabaja con el reloj LPOCLK, el contador del COP es inicializado a cero para los modos BDM y STOP, tanto para entrar como para salir de estos modos. El bit COP del registro SRS se pone a “1” indicando el evento de RESET (ver Figura 10.4).

Registro SRS								
	7	6	5	4	3	2	1	0
Lectura	POR = 1	PIN	COP	ILOP	ILAD	LOC	LVD	0
Escritura								
Escribir algún valor en el SRS, aclara el contador del COP ( <i>Watchdog</i> )								

**Figura 10.4. Bit COP como evento causante de RESET.**

El CodeWarrior® 6.2 contiene macros para controlar la activación e inhibición del COP. Por ejemplo, el siguiente macro aclara el contador que causa sobreflujo del sistema del COP:

```
#define __RESET_WATCHDOG() (void)(SRS = 0x00)
```

Para desactivar la acción del COP se puede definir una macro con el siguiente contenido:

```
#define Disable_COP() SOPT1=0x3F;
```

- **Reset por ILOP (*Illegal Opcode*):** Por defecto, la máquina ColdFire® V1 habilita la posibilidad de detectar la ejecución de una operación ilegal. También, este tipo de RESET es generado cuando se intenta ejecutar una instrucción con privilegio para el modo supervisor, desde el modo de usuario.

Intentar ejecutar las instrucciones STOP y HALT cuando no han sido habilitadas, genera un estado de RESET a la máquina. El bit ILOP del registro SRS se pone a “1” indicando el evento de RESET (ver Figura 10.5).

### Registro SRS

	7	6	5	4	3	2	1	0
Lectura	POR = 1	PIN	COP	ILOP	ILAD	LOC	LVD	0
Escritura	Escribir algún valor en el SRS, aclara el contador del COP (Watchdog)							

Figura 10.5. Bit ILOP como evento causante de RESET.

- **Reset por ILAP (Illegal Address):** Por defecto, la máquina ColdFire® V1 habilita la posibilidad de detectar el acceso a una dirección ilegal, error por terminación de bus, error por formato de RTE (*Return of Exception*) o una condición *fault on fault*. El bit ILAD del registro SRS se pone a “1” indicando el evento de RESET (ver Figura 10.6).

### Registro SRS

	7	6	5	4	3	2	1	0
Lectura	POR = 1	PIN	COP	ILOP	ILAD	LOC	LVD	0
Escritura	Escribir algún valor en el SRS, aclara el contador del COP (Watchdog)							

Figura 10.6. Bit ILAD como evento causante de RESET.

- **Reset por LVD (Low Voltage Detect):** La máquina ColdFire V1 contiene un mecanismo para la detección de un evento, por bajo voltaje en la fuente de alimentación. Este mecanismo protege el contenido de la memoria y los estados de control del sistema, ante variaciones en el voltaje de alimentación.

Para habilitar el sistema LVD es necesario poner a “1” el bit SPMSC1[LVDE] y elegir un voltaje de comparación adecuado con el bit SPMSC3[LVDV]. La operación por LVD no será habilitada si la CPU se encuentra en los modos STOP2 y STOP3, a menos que el bit SPMSC1[LVSE] se encuentre en “1”. Si los bits SPMSC1[LVDE] y SPMSC1[LVSE] se encuentran ambos en “1”, el sistema entra en el modo STOP4.

Si se quiere que el LVD genere RESET a la máquina, será necesario poner a “1” el bit SPMSC1[LVRE].

Existe un sistema de precaución (LVW: *Low Voltage warning*), para la detección anticipada de variación en el voltaje de alimentación. El objetivo es indicarle al usuario que la fuente se encuentra cercana al umbral de detección de bajo voltaje (Vtrip) y la manera como lo indica es con la bandera LVWF (*Low Voltage Warning Flag*).

Es posible generar un evento de interrupción por LVW, poniendo a “1” el bit SPMSC3[LVWIE]. Para aclarar la bandera LVWF es necesario escribir un “1” en el bit SPMSC3[LVWACK]. Existen dos niveles de comparación para el Vtrip y se

eligen con el bit SPMSC3[LVWV]. El bit LVD del registro SRS se pone a “1” indicando el evento de RESET (ver Figura 10.7).

**Registro SRS**

	7	6	5	4	3	2	1	0
Lectura	POR = 1	PIN	COP	ILOP	ILAD	LOC	LVD	0
Escritura	Escribir algún valor en el SRS, aclara el contador del COP (Watchdog)							

**Figura 10.7. Bit LVD como evento causante de RESET.**

- **Reset por pérdida del enganche del reloj externo (LOC):** En caso de perderse el reloj que se genera externamente, el sistema de excepciones genera un RESET por evento LOC(*Loss Of Clock*). El bit LOC del registro SRS se pone a “1” indicando el evento de RESET (ver Figura 10.8).

**Registro SRS**

	7	6	5	4	3	2	1	0
R	POR = 1	PIN	COP	ILOP	ILAD	LOC	LVD	0
W	Writing any value to SRS address clears COP watchdog timer.							

**Figura 10.8. Bit LOC como evento causante de RESET.**

## 10.2. EL CONTROLADOR DE INTERRUPCIONES (CF1\_INTC)

Esta diseñado para la línea de microcontroladores ColdFire® V1 de Freescale, que generalmente soportan menos programabilidad que las gamas superiores.

El diseño del controlador se enfoca en la compatibilidad entre la familia de 8 bits HCS08 y la familia de 32 bits ColdFire® V1, conservando características tan importantes como la compatibilidad en la prioridad de atención a las interrupciones y similitud en el modo de procesamiento de las interrupciones (con un solo nivel de *nesting*).

La Tabla 10.2 hace un resumen de la similitud de estas dos arquitecturas, desde el punto de vista del control de las interrupciones.

**Tabla 10.2. Comparación de procesos de excepción.**

Atributo	HCS08	ColdFire V1
Tabla de vectores de excepción	32 entradas de 2 bytes, de localización fija / parte alta del final de la memoria	103 entradas de 4 bytes, relocalizables según el VBR* / parte baja del final de la memoria
Número de vectores	2 para la CPU + 30 para las IRQs, el RESET va en la dirección más alta	64 para la CPU + 39 para IRQs, RESET en la dirección mas baja.
Trama de excepción de Pila	Trama de 5 bytes: CCR, A, X y PC	Trama de 8 bytes: F/V, SR, PC, los registros (An,Dn) pueden ser salvados en la pila
Niveles de interrupción	$1 = f(CCR[I])$	$7=f(SR[I])$ con soporte automático de hardware para el nesting
Soporte de IRQ No enmascarable	No	Si, con 7 niveles de interrupción
Potenciamiento de sensibilidad a la IRQ	No	El nivel 7 es sensible al flanco, y de otra forma es sensible al nivel
Prioridad en la vectorización del INTC	Prioridad fija y asignación por vector	Prioridades y asignaciones de vector fijas. Sólo 2 IRQs pueden ser remapeadas con el nivel 6 (más alto)
IACK por software	No	Yes
Instrucción para salir del ISR	RTI	RTE

Un proceso de excepción por interrupción consta de los siguientes pasos, generales:

- Reconocimiento de la interrupción presente.
- Proceso de suspensión de la instrucción que está siendo ejecutada por la CPU.
- Almacenamiento de 8 bytes en la pila. La información que se almacena en la pila durante la atención a una excepción de interrupción es:
  - 4 bytes con la dirección de retorno. Contenido del PC antes de atender la interrupción en proceso.
  - 2 bytes para el registro SR (*Status Register*).
  - 2 bytes para el F/V (*Format Vector Word*).
- Cálculo del vector apropiado para atender la interrupción.
- Paso del control a la subrutina que atiende la interrupción.

Las máquinas ColdFire® revisan el estado de las interrupciones con cada instrucción que ejecutan, esto coincide con cada ciclo del OEP (*Operand Execution Pipeline*).

Para el manejo de los niveles de interrupción, la máquina está dotada de la máscara de interrupciones I. La máscara I está formada por tres bits, que definen siete niveles de interrupciones con capacidad automática de *nesting*<sup>30</sup>.

El nivel más alto es el 7 y así en forma descendiente hasta el nivel 1, el nivel 0 (I=0) significa la habilitación de todas las interrupciones del sistema.

El nivel 7 aplica para las interrupciones no enmascarables sensibles a flanco y las interrupciones del nivel 6 al 1 son enmascarables y sensibles al nivel.

Toda interrupción que se presente y tenga un nivel de interrupción menor o igual al nivel de la interrupción presente, será inhibida. Lo anterior es norma, excepto para la interrupción no enmascarable de nivel 7 y sensible a flanco.

Los procesadores ColdFire® contienen una tabla de 1024 bytes, localizada mediante el vector VBR y alineada con un vecindario de 1MB. Lo anterior se hace por compatibilidad entre las máquinas desde la versión V1 hasta la V4, para la versión V1 esta localización tiene sentido en las direcciones 0x(00)00\_0000 (FLASH) y la 0x(00)80\_0000 (RAM).

La Tabla 10.3 contiene la ubicación de cada vector de excepción para la familia ColdFire®.

Cada nivel de interrupción soporta nueve asignaciones de prioridad, esto hace que sea posible establecer un total de 63 jerarquías. La relación entre el cálculo del vector de las interrupciones de las máquinas HCS08 y ColdFire, está establecida por la ecuación:

$$\boxed{\text{Número del vector en ColdFire®_V1} = \text{Número del vector en HCS08} + 62}$$

Para emular las capacidades de nivel 1 de las máquinas HCS08 en las máquinas ColdFire® V1, se implementa el manejo de sólo dos valores para la máscara SR[I].

- SR[I] = 7, Se usa para inhibir las interrupciones.
- SR[I] = 0, Se usa para habilitar las interrupciones.

---

<sup>30</sup> Nesting: Se refiere a la capacidad que tienen las máquinas de atender varias interrupciones pendientes, una a continuación de la otra utilizando un procedimiento de priorización.

**Tabla 10.3. Vectores de excepción.**

Número Vector	Offset del Vector	PC en la Pila	Asignación
0	0x000	—	Puntero a pila inicial supervisor
1	0x004	—	Contador de programa inicial
2–63	0x008–0x0FC	—	Reservado para excepciones internas a la CPU
64	0x100	Próximo	Pin IRQ
65	0x104	Próximo	Bajo voltaje
66	0x108	Próximo	TPM1_ch0
67	0x10C	Próximo	TPM1_ch1
68	0x110	Próximo	TPM1_ch2
69	0x114	Próximo	TPM1_ovfl
70	0x118	Próximo	TPM2_ch0
71	0x11C	Próximo	TPM2_ch1
72	0x120	Próximo	TPM2_ch2
73	0x124	Próximo	TPM2_ovfl
74	0x128	Próximo	SPI2
75	0x12C	Próximo	SPI1
76	0x130	Próximo	SCI1_err
77	0x134	Próximo	SCI1_rx
78	0x138	Próximo	SCI1_tx
79	0x13C	Próximo	IICx
80	0x140	Próximo	KBIx
81	0x144	Próximo	ADC
82	0x148	Próximo	ACMPx
83	0x14C	Próximo	SCI2_err
84	0x150	Próximo	SCI2_rx
85	0x154	Próximo	SCI2_tx
86	0x158	Próximo	RTC
87	0x15C	Próximo	TPM3_ch0
88	0x160	Próximo	TPM3_ch1
89	0x164	Próximo	TPM3_ch2
90	0x168	Próximo	TPM3_ch3
91	0x16C	Próximo	TPM3_ch4
92	0x170	Próximo	TPM3_ch5
93	0x174	Próximo	TPM3_ovfl
94–95	0x178–0x17C	—	No se usan para V1
96	0x180	Próximo	SWI 7
97	0x184	Próximo	SWI 6
98	0x188	Próximo	SWI 5
99	0x18C	Próximo	SWI 4
100	0x190	Próximo	SWI 3
101	0x194	Próximo	SWI 2
102	0x198	Próximo	SWI 1
103–255	0x19C–0x3FC	—	No se usan para V1

### 10.3. EL PIN DE INTERRUPCIÓN (IRQ)

El registro IRQSC es el encargado de manipular la señal externa de IRQ, que ocupa lugar privilegiado dentro de los niveles y prioridades en la máquina de las interrupciones. El procesador monitorea permanentemente la lógica del pin de IRQ y puede validar tanto el flanco, como el flanco y nivel de la señal presente en el pin.

La interrupción generada por el pin IRQ puede sacar a la máquina de STOP, aún con el reloj suspendido. La Figura 10.9 ilustra sobre el registro IRQSC.

**Registro IRQSC : (0xFFFF801B)**

Lectura	7	6	5	4	3	2	1	0
Escritura	0	IRQPDD	IRQEDG	IRQPE	IRQF	0	IRQIE	IRQMOD
Reset	0	0	0	0	0	0	0	0

**Figura 10.9. Registro de estado y control del pin IRQ.**

- **Bit IRQPDD (IRQ Pull Device Disable):** Deshabilita el la resistencia de *pullup/pulldown* asociada al pin IRQ.
  - IRQPDD = 0: La resistencia está habilitada.
  - IRQPDD = 1: La resistencia esta deshabilitada.
- **Bit IRQEDG (IRQ Edge Select):** Configura el flanco que activará la entrada de IRQ.
  - IRQEDG = 0: El pin IRQ será sensible al flanco de bajada o flanco de bajada y nivel bajo (de existir una resistencia asociada al pin, deberá ser de *pullup*).
  - IRQEDG = 1: El pin IRQ será sensible al flanco de subida o flanco de subida y nivel alto (de existir una resistencia asociada al pin, deberá ser de *pulldown*)
- **Bit IRQPE (IRQ Pin Enable):** Habilita el funcionamiento del pin IRQ.
  - IRQPE = 0: El pin IRQ ha sido deshabilitado.
  - IRQEDG = 1: El pin IRQ ha sido habilitado.
- **Bit IRQF (IRQ Flag):** Bandera que indica cuando un evento de IRQ ha ocurrido.
  - IRQPE = 0: No hay evento de IRQ.
  - IRQEDG = 1: Ha ocurrido un evento de IRQ.
- **Bit IRQACK (IRQ Acknowledge):** Bit para el reconocimiento de un evento de IRQ. Escribiendo un “1” en este bit, se aclara la bandera IRQF. El usuario deberá aclarar este bit en la atención al evento, siempre y cuando el modo de operación no se encuentre en flanco y nivel (IRQMOD=1) y esté activo.
  - IRQACK = 0: No tiene efecto.
  - IRQACK = 1: Aclara la bandera de IRQF con IRQMOD=0.

- **Bit IRQIE (IRQ Interrupt Enable):** Habilita un evento de interrupción, ante la aparición de un evento de IRQ.
  - IRQIE = 0: No se habilita evento de interrupción por IRQ. El usuario puede usar la lectura iterativa (*polling*) sobre la bandera IRQF, siempre y cuando el pin esté habilitado.
  - IRQIE = 1: Habilita el evento de interrupción por IRQ.
- **Bit IRQMOD (IRQ Mode):** Elige el modo de operación, eléctrico, con el pin de IRQ
  - IRQMOD = 0: El pin de IRQ sólo actúa con el flanco de la señal eléctrica presente.
  - IRQIE = 1: El pin de IRQ actúa tanto en el flanco, como en el nivel de la señal eléctrica.

#### 10.4. EJERCICIO CON EL PIN IRQ

El ejercicio detecta una interrupción asignada al pin IRQ (pin 2 del conector MCU-Port del DEMOJM) del MCU y hace un *toggle* sobre el LED PTD2 del DEMOJM. El circuito de la Figura 10.10 ilustra la conexión sugerida para la prueba del ejercicio.

El programa, desarrollado en C, es mostrado a continuación y es posible tener acceso a este vía sitio WEB del texto con el nombre de PIN\_IRQ.

La función de inicialización del pin IRQ utiliza la definición de tipo (*Typedef*) para el registro de estado y control de la IRQ (IRQSC), que se encuentra en el archivo *header mcf51jm128.h*. De tal manera que todos aquellos bits que se tengan que llevar al estado “1”, bastaría con hacer la OR con el bit máscara que lo representa.

Tomada del archivo mencionado, esta definición es:

```
typedef union {
    byte Byte;
    struct {
        byte IRQMOD      :1;          /* IRQ Detection Mode */
        byte IRQIE       :1;          /* IRQ Interrupt Enable */
        byte IRQACK      :1;          /* IRQ Acknowledge */
        byte IRQF        :1;          /* IRQ Flag */
        byte IRQPE       :1;          /* IRQ Pin Enable */
        byte IRQEDG      :1;          /* IRQ Edge Select */
        byte IRQPDD      :1;          /* IRQ Pull Device Disable */
        byte             :1;
    } Bits;
} IRQSCSTR;
extern volatile IRQSCSTR _IRQSC @ 0xFFFF801B;
#define IRQSC           _IRQSC.Byte
#define IRQSC_IRQMOD   _IRQSC.Bits.IRQMOD
#define IRQSC_IRQIE    _IRQSC.Bits.IRQIE
#define IRQSC_IRQACK   _IRQSC.Bits.IRQACK
#define IRQSC_IRQF     _IRQSC.Bits.IRQF
#define IRQSC_IRQPE    _IRQSC.Bits.IRQPE
```

```

#define IRQSC_IRQEDG _IRQSC.Bits.IRQEDG
#define IRQSC_IQPDD _IRQSC.Bits.IQPDD

#define IRQSC_IRQMOD_MASK 1
#define IRQSC_IRQIE_MASK 2
#define IRQSC_IRQACK_MASK 4
#define IRQSC_IRQF_MASK 8
#define IRQSC_IRQPE_MASK 16
#define IRQSC_IRQEDG_MASK 32
#define IRQSC_IQPDD_MASK 64

```

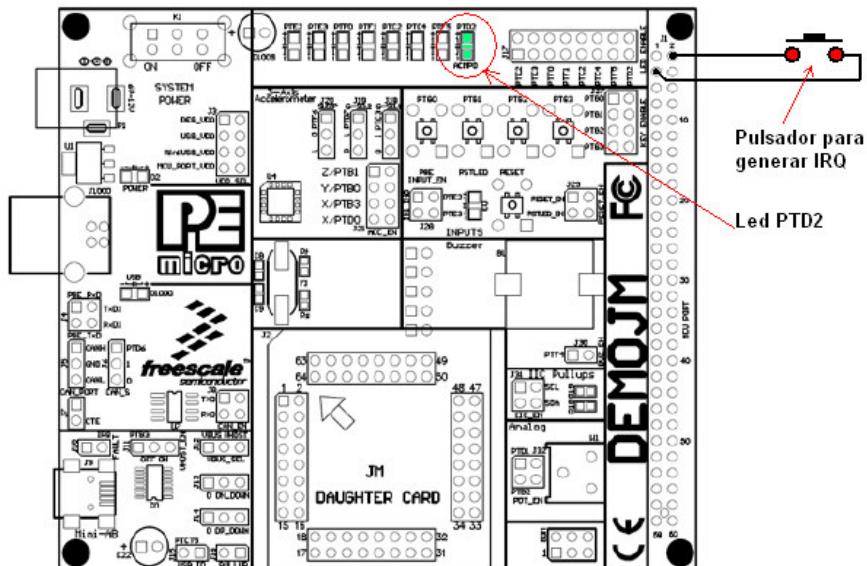


Figura 10.10. Conexión para el ejercicio de la IRQ.

**NOTA:** Se recomienda al usuario revisar cuidadosamente el estado de los puentes del DEMOJM, con el objetivo de hacer compatible el ejercicio actual y de esta manera liberar los pines comprometidos en el circuito.

En la declaración de tipo se aprovecha la definición de las máscaras para los diferentes bits de configuración del registro IRQSC. Finalmente, el programa sería:

```

*****
/* PIN_IRQ: Ejemplo sobre la utilización del pin de */
/* IRQ. */
/* main.c */
/*
/* Fecha: Mayo 2, 2008 */
/*
/* V1.0, Rev 0 por Diego Múnera */
/*
/* Asunto: Implementar un código en C que realice */
/* encendido y apagado del led en PTD2 de la tarje- */
*/

```

```

/*
 * ta de evaluacion DEMOJM, cada que se pulse sobre el pin de IRQ.
 */
/*
 * Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros editores.
 */
//********************************************************************

/* Archivos de cabecera */
#include <hidef.h> //macro para interrupciones
#include "derivative.h" //macro de declaraciones del MCU

/* Declaracion de variables y constantes */
byte bandera=0;
char byteConfig=0;

/* Macros y definiciones */
#define Disable_COP() SOPT1 &= 0x3F //deshabilita el COP
#define Led1_On() PTDD_PTDD2 = 0; PTDDD_PTDDD2 = 1 //macro para encendido de Led1
#define Led1_Off() PTDD_PTDD2 = 1; PTDDD_PTDDD2 = 0 //macro para apagado de Led1

/* Declaracion de funciones */
void IRQ_Init(char byteConfig); //declaracion de funcion que inicializa la IRQ

/* main(): Funcion principal */
void main(void) {
    Led1_On(); //el LED inicia encendido
    Disable_COP(); //deshabilita el COP
    EnableInterrupts(); //habilita interrupciones
    IRQ_Init(IRQSC_IRQPE_MASK | IRQSC_IRQACK_MASK | IRQSC_IRQIE_MASK); //Flanco de bajada, habilita interrupcion y se da ACK
    for(;;) { //iteracion infinita

    }
    /* es necesario asegurarse de nunca abandonar el main */
}

/* Función para inicializar la IRQ */
void IRQ_Init(char byteConfig){ //parametros de entrada a la funcion
    IRQSC = byteConfig; //modifique parametros de inicializacion
}

/* Funcion de atencion a la IRQ */
interrupt VectorNumber_Virq void ISR_IRQ (void){
    IRQSC_IRQACK=1; //aclara bandera de interrupcion por IRQ
    if(bandera==1){ //si bandera es 1
        Led1_On(); //active el LED
        bandera=0; //aclare bandera
    }else{
        Led1_Off(); //desactive el LED
        bandera=1; //ponga bandera a 1
    }
}

```

## **10.5. REFERENCIAS**

- MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual. Freescale Semiconductor. Rev 1, 2008.
- Microcontroller MCF51JM128 ColdFire Data Sheet. Freescale Semiconductor. Rev 0, 2008.

## **10.6. PREGUNTAS**

- Enuncie cuatro potenciales fuentes de RESET de la máquina MCF51JM128.
- ¿Qué es un RESET por instrucción ilegal (ILOP)?
- ¿Cuántos y cuales registros guarda en la pila la máquina MC9S08JM60, al atender un evento de interrupción?
- Será posible saber ¿quién ha sido el causante de un evento de excepción? y de ¿qué manera se podría verificar?
- ¿En qué estado deberá estar la bandera I para recibir cualquier evento de interrupción?
- Diseñe un sistema (*hardware y software*) que reciba una señal periódica de 1 Hz por el pin de IRQ y presente en un LCD el número de IRQ's recibidas. El sistema deberá trabajar con la atención al evento de interrupción.

# CAPÍTULO 11

## Reloj del Sistema (MCG)

- 11.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**
- 11.2. Ejercicio con el MCG: Inicialización del reloj interno de la MCU y comprobación de la velocidad de trabajo**
- 11.3. Referencias**
- 11.4. Preguntas**

## 11.1. DIAGRAMA EN BLOQUES, EXPLICACIÓN DEL FUNCIONAMIENTO Y REGISTROS ASOCIADOS

El módulo MCG (*Multipurpose Clock Generator*) suministra una variedad de fuentes de reloj al MCU y su corazón es un dispositivo que contiene un FLL (*Frequency Locked Loop*: Bucla Enganchada por Frecuencia) y un PLL (*Phase Locked Loop*: Bucla Enganchada por Fase), controlados por referencias externas o internas al procesador. El usuario deberá seleccionar una de las dos opciones de trabajo, con FLL o con PLL.

El reloj que se deriva de los módulos anteriores, es conducido hacia divisores de frecuencia y así poder derivar relojes internos de baja frecuencia.

También es posible conectar un reloj de resonador o cristal externo e inclusive una fuente de reloj externa. La Figura 11.1 muestra un diagrama en bloques del MCG. La máxima frecuencia de cristal externo deberá ser de 16MHz.

**NOTA:** Para trabajar con el módulo USB del MCF51JM128, es necesario configurar el módulo MCG con la opción PLL en modo PEE y frecuencia de salida MCGOUT de 48MHz.

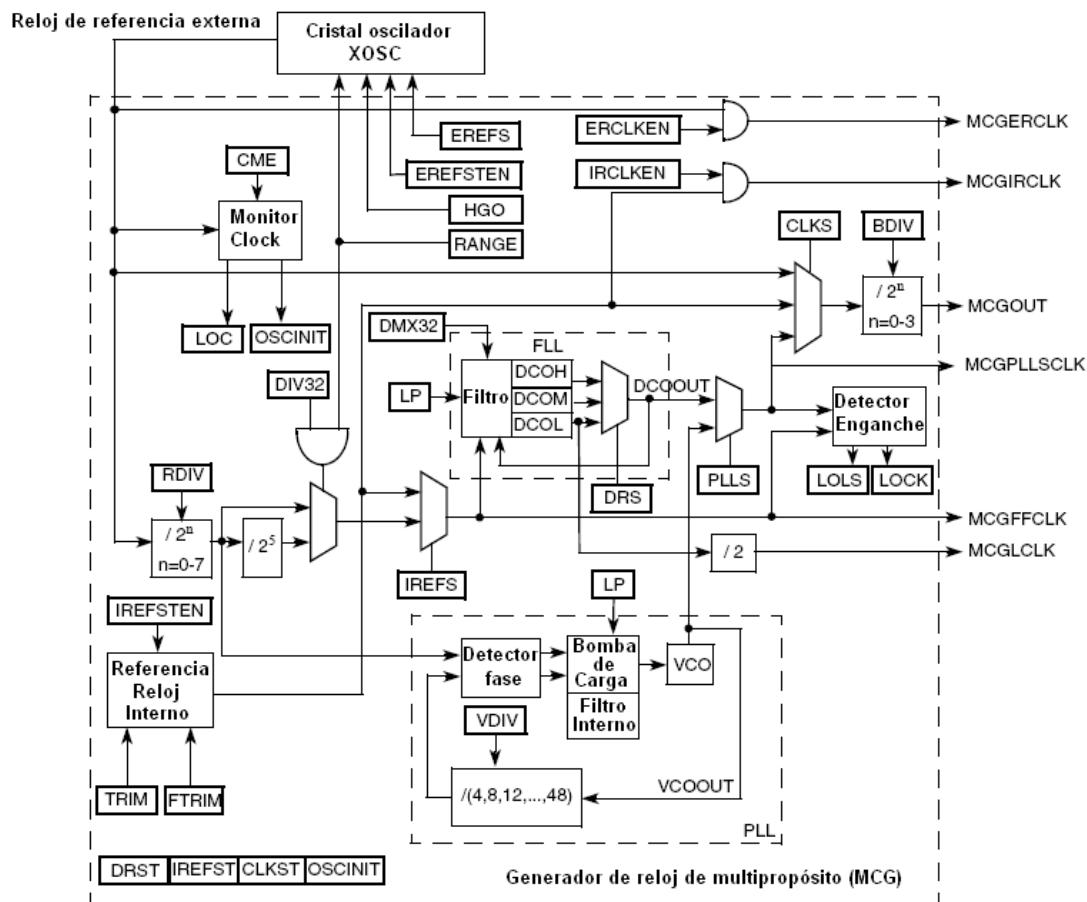


Figura 11.1. Diagrama funcional del módulo generador de reloj (MCG).

Para explicar el funcionamiento del módulo MCG es necesario hacer referencia a los modos de funcionamiento, que a continuación se presentan.

- **Modo FLL referenciado internamente (FEI: FLL Engaged Internal):** El reloj de la MCU es generado internamente y se deriva del bloque FLL con una referencia interna. Este es el modo de arranque por defecto del MCU. Las partes comprometidas aparecen rodeadas de una línea roja y sombreado, como se ilustra en la Figura 11.2.

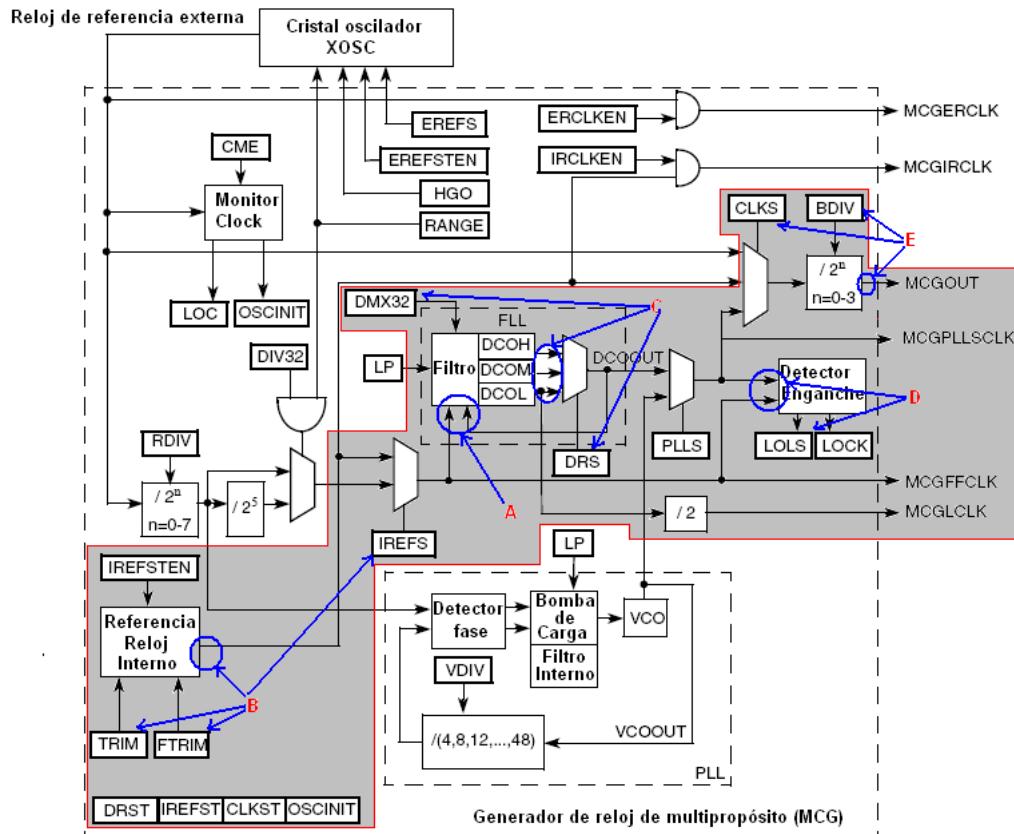


Figura 11.2. Modo FEI del MCG.

La salida DCOOUT es comparada con la referencia interna en el punto A, esto permite hacer la realimentación a la bucle de frecuencia y permitir el ajuste interno del FLL, para generar la frecuencia requerida.

**NOTA:** En el proceso de fabricación del MCU, es programado en un registro de 8 bits (de la memoria no volátil) un valor de ajuste llamado NVIMCGTRM. Este valor corresponde a la cifra que el usuario deberá cargar en el registro MCGTRM, para obtener un reloj interno de igual valor y poder hacer cálculos temporales confiables independientemente del MCU .

La referencia interna es suministrada por el circuito **B**, previamente seleccionada por el bit IREFS (*Internal Referente Select*). La referencia puede ser ajustada mediante los bits TRIM (*Trimmer*) y FTRIM (*Fine TRIM*), pero recuerde que esto no garantiza un reloj igual para una serie de MCU's programados con el mismo programa (ver NOTA anterior).

El circuito FLL genera una señal en frecuencia (circuito **C**), dependiendo del rango asignado para el DCO (*Digital Controller Oscilator*). El rango de frecuencias de salidas del DCO es controlado por DRS (DCO Range Select) y el bit DMX32 ajusta el máximo rango de frecuencias con una referencia de 32,768KHz.

Para la detección del enganche, tanto de la bucle con PLL como la del FLL, se utilizan las señales de LOCK (*Locked*) y LOLS (*Loss Of LOCK Status*). Esto es útil para saber en cualquier momento de la confiabilidad del reloj del sistema (ver circuito **D**).

El circuito **E** es la etapa de salida del reloj generado por el FLL en modo FEI y consta de un divisor de frecuencias, programado por BDIV (*Bus Frequency Divider*). El bit CLKS (*Clock Select*) selecciona la fuente de reloj de la máquina.

- **Modo FLL sobrepasado internamente (FBI: FLL Bypassed Internal):** El reloj de la MCU es generado internamente desde la referencia interna y el circuito FLL, aunque está funcionando, no es considerado. Las partes comprometidas aparecen rodeadas de una línea roja y sombreado, como se ilustra en la Figura 11.3.

Este modo es ampliamente usado cuando se necesita que el FLL adquiera su frecuencia objetivo, mientras que la salida de reloj MCGOUT suministre al sistema la frecuencia producida por la referencia interna.

Pasados 1024 ciclos de la referencia interna, se asume el enganche del FLL y el usuario podrá comutarse a modo FEI. Las partes involucradas en este modo ya fueron explicadas en el modo FEI y más adelante se detallarán algunos de sus parámetros de configuración.

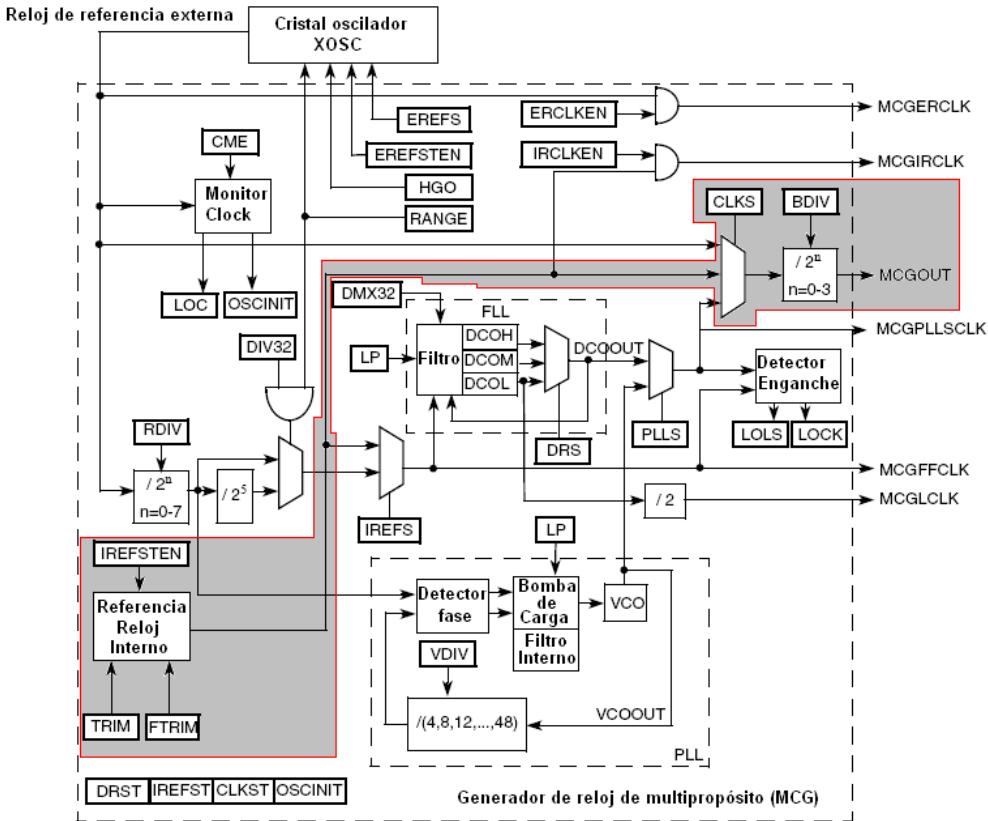


Figura 11.3. Modo FBI del MCG.

- **Modo FLL sobrepasado internamente y baja potencia (BLPI: Bypassed Low Power Internal):** Tanto el PLL como el FLL están deshabilitados, para minimizar consumo de energía. El BDM y los módulos de comunicación no pueden operar en este modo. El bit LP (Low Power) deberá estar al estado “1”. El sistema opera con el reloj de referencia interna.
- **Modo FLL referenciado externamente (FEE: FLL Engaged External):** El reloj de la MCU es generado externamente y se deriva del bloque FLL con una referencia externa. Las partes comprometidas aparecen rodeadas de una línea roja y sombreado, como se ilustra en la Figura 11.4.

Como circuitos nuevos aparecen A, B y C, de la Figura 13.4. Las demás partes aparecen explicadas en el modo FEI.

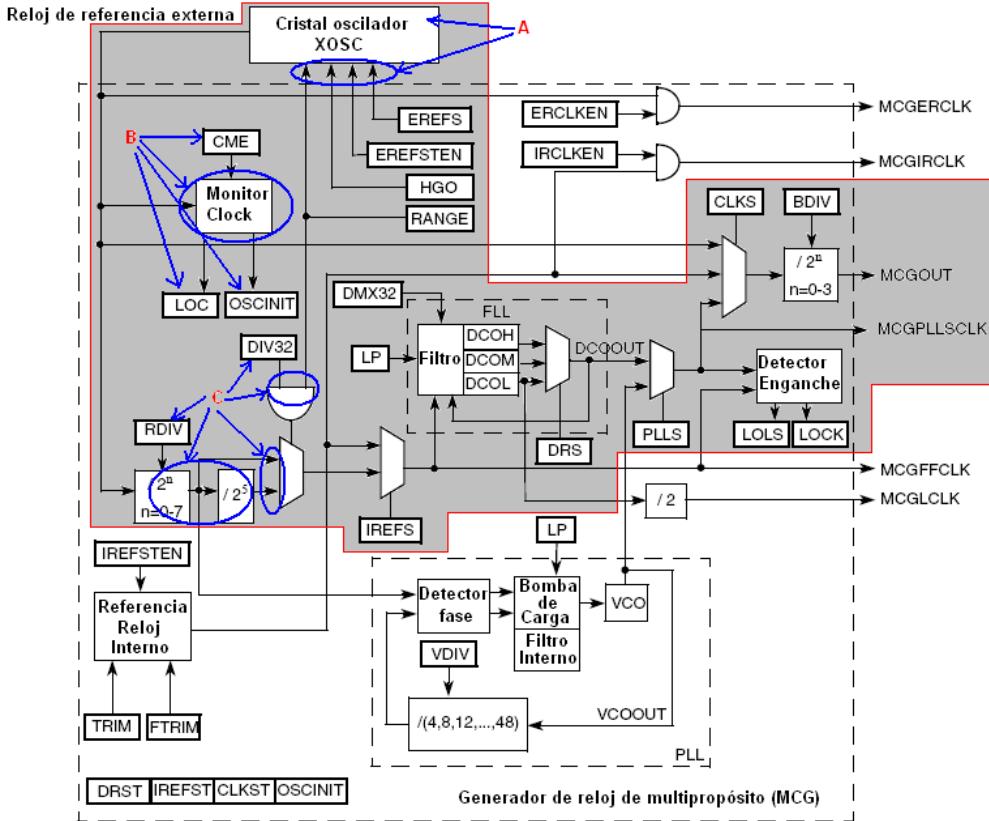


Figura 11.4. Modo FEE del MCG.

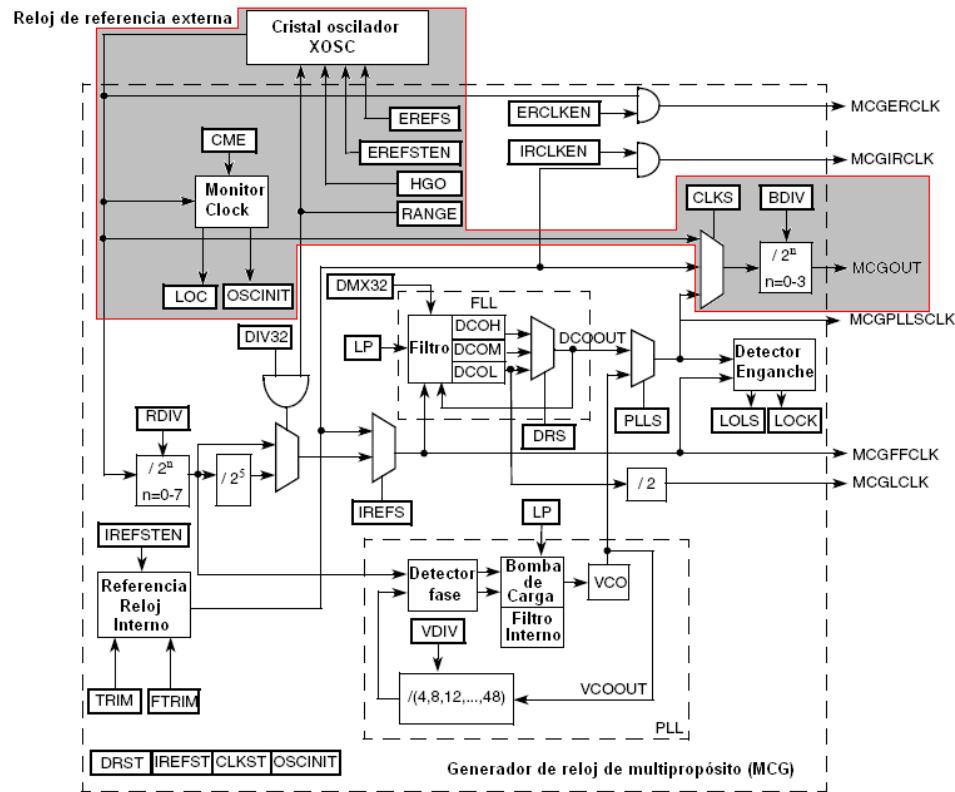
El circuito **A** detalla el uso de un oscilador externo, en donde el bit EREFS (*External Reference Select*) selecciona la referencia externa, el bit EREFSTEN habilita el funcionamiento de la referencia externa como reloj del sistema para efectos del modo STOP, el bit HGO (*High Gain Operation*) controla la ganancia para la operación del cristal y el bit RANGE selecciona el rango de frecuencias para el cristal externo.

En el circuito **B**, el bit CME (*Clock Monitor Enable*) habilita el monitoreo del reloj externo con el fin de generar un RESET ante la pérdida de éste y el bit OSCINIT (*OSC Initialization*) indica al sistema el momento en que se completaron los ciclos de espera ante el arranque del oscilador externo.

El circuito **C** configura la etapa de divisores para la referencia externa, en donde los bits RDIV conforman el divisor de frecuencia (modos FLL y PLL) y el bit DIV32 adiciona un divisor por 32 cuando el bit RANGE = “1”.

- **Modo FLL sobrepasado externamente (FBE: FLL Bypassed External):** El reloj de la MCU es generado externamente desde la referencia externa y el circuito FLL, aunque está funcionando, no es considerado. Este modo es ampliamente usado cuando se necesita que el FLL adquiera su frecuencia objetivo, mientras que la salida de reloj MCGOUT suministre al sistema la

frecuencia producida por la referencia externa. Pasados 1024 ciclos de la referencia interna, se asume el enganche del FLL y el usuario podrá conmutarse a modo FEE. Las partes involucradas en este modo ya fueron explicadas en el modo FEI y FEE, y más adelante se detallarán algunos de sus parámetros de configuración. Las partes comprometidas aparecen rodeadas de una línea roja y sombreado, como se ilustra en la Figura 11.5.



**Figura 11.5. Modo FBE del MCG.**

- **Modo FLL sobrepasado externamente y baja potencia (BLPE: Bypassed Low Power External):** Tanto el PLL como el FLL están deshabilitados, para minimizar consumo de energía. El BDM y los módulos de comunicación no pueden operar en este modo. El bit LP (Low Power) deberá estar al estado “1”. El sistema opera con el reloj de referencia externa.
- **Modo PLL referenciado externamente (PEE: PLL Engaged External):** El reloj del sistema se deriva del circuito PLL, que se encuentra referenciado externamente. Las partes comprometidas aparecen rodeadas de una línea roja y sombreado, como se ilustra en la Figura 11.6.

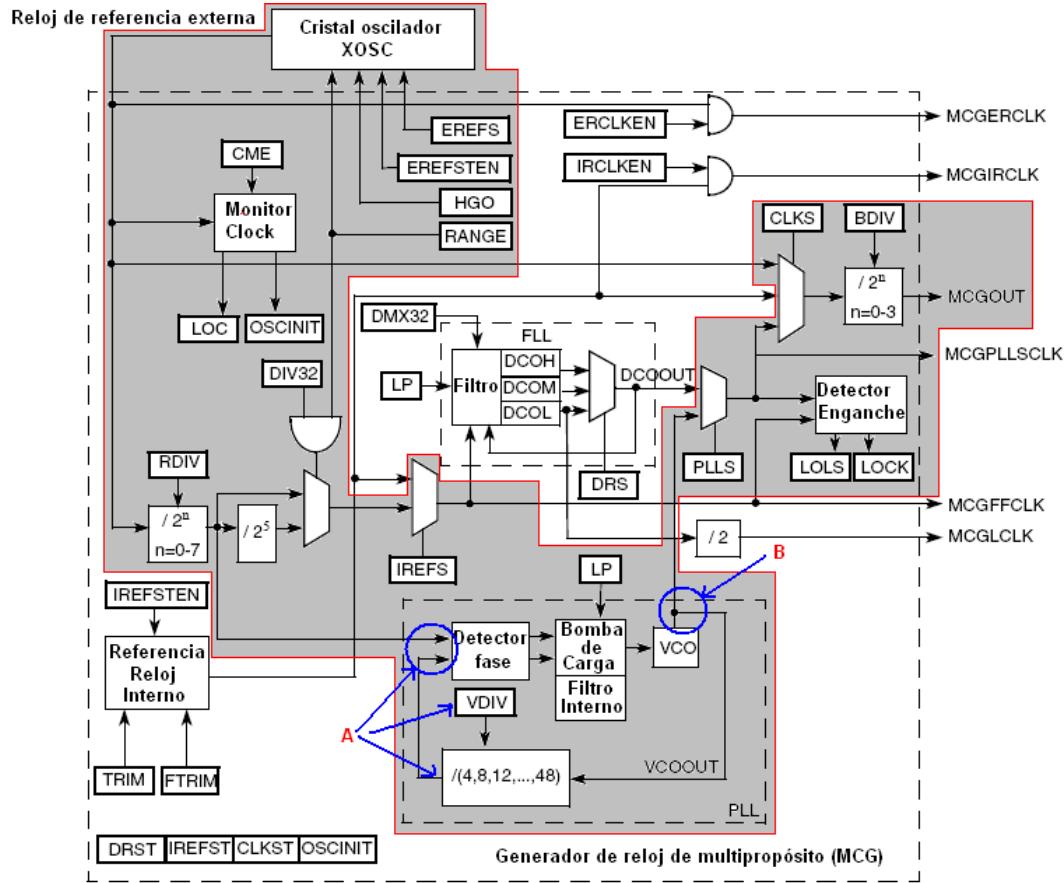


Figura 11.6. Modo PEE del MCG.

El circuito A establece la comparación entre la señal generada por el PLL (salida del VCO: *Voltaje Controller Oscilador*), dividida un factor programado por los bits VDIV, contra la referencia externa, dividida un factor RDIV. Este mecanismo de realimentación permite el enganche del sistema PLL, ante el establecimiento de la frecuencia de trabajo interno.

El circuito B corresponde a la señal de salida del VCO, que viene siendo el reloj de trabajo del sistema.

- **Modo PLL sobrepasado externamente (PBE: PLL bypassed External):** El reloj de la MCU es generado externamente desde la referencia externa y el circuito PLL, aunque está funcionando, no es considerado. Este modo es ampliamente usado cuando se necesita que el PLL adquiera su frecuencia objetivo, mientras que la salida de reloj MCGOUT suministre al sistema la frecuencia producida por la referencia externa.

Las diferencias respecto del modo FBE son: el módulo BDM podría trabajar en esta condición y su reloj de operación se deriva del DCO (*Digital Controller*

*Oscilador*) trabajando en lazo abierto y otra diferencia es que el rango de operación de la referencia externa es diferente.

- **Registros asociados al MCG:** Los registros asociados al módulo MCG se detallan a continuación.
  - **Registro de control 1 del MCG (MCGC1):** La Figura 11.7 detalla los bits asociados al registro MCGC1.
  -

**Registro MCGC1: (0xFFFF8048)**

Lectura Escritura	7	6	5	4	3	2	1	0
	CLKS		RDIV		IREFS	IRCLKEN	IREFSTEN	
Reset:	0	0	0	0	0	1	0	0

**Figura 11.7. Registro MCGC1 del MCG.**

- Los bits CLKS seleccionan la fuente generadora del reloj del sistema, como se detalla a continuación:
  - 00:** Selecciona la salida del PLL o FLL
  - 01:** Selecciona el reloj de referencia interna
  - 10:** Selecciona el reloj de referencia externa
  - 11:** Reservado
- Los bits RDIV conforman el divisor de la referencia externa, como se detalla en las Tablas 11.1 y 11.2:
  -

**Tabla 11.1. RDIV en modo FLL.**

Para opción FLL:

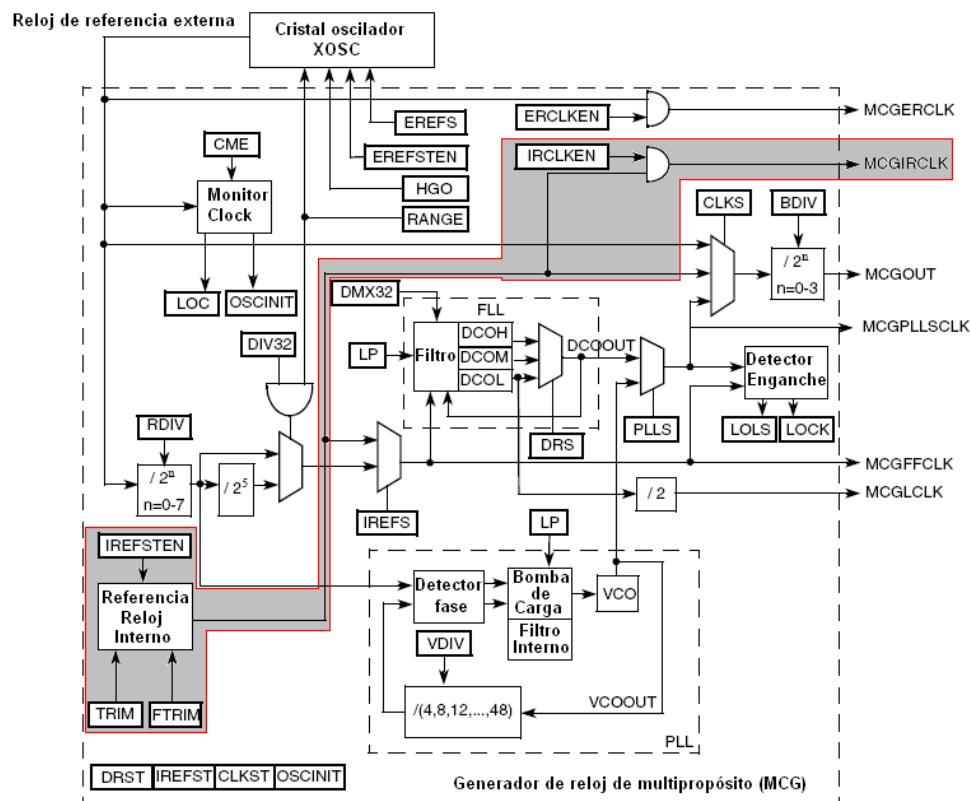
RDIV	Factor Divisor		
	RANGE:DIV32 0:X	RANGE:DIV32 1:0	RANGE:DIV32 1:1
0	1	1	32
1	2	2	64
2	4	4	128
3	8	8	256
4	16	16	512
5	32	32	1024
6	64	64	Reservado
7	128	128	Reservado

**Tabla 11.2. RDIV en modo PLL.**

Para modo PLL:

RDIV	Factor Divisor
0	1
1	2
2	4
3	8
4	16
5	32
6	64
7	128

- El bit IREFS selecciona la referencia de trabajo del MCG , como se detalla a continuación:  
**0:** Selección de la referencia externa  
**1:** Selección de la referencia interna
- El bit IRCLKEN selecciona el reloj de referencia interna como reloj MCGIRCLK. La Figura 11.8, detalla en línea roja y sombreado, este modo.



**Figura 11.8. Habilita reloj MCGIRCLK.**

- El bit IREFSTEN controla cuando el reloj de referencia interna permanece activo en el modo STOP , como se detalla a continuación:
  - 0:** El reloj de referencia interna permanece inactivo en modo STOP
  - 1:** El reloj de referencia interna permanece activo en modo STOP, para modos FEI, FBI o BLPI
- **Registro de control 2 del MCG (MCGC2):** La Figura 11.9 detalla los bits asociados al registro MCGC2.

**Registro MCGC2: (0xFFFF8049)**

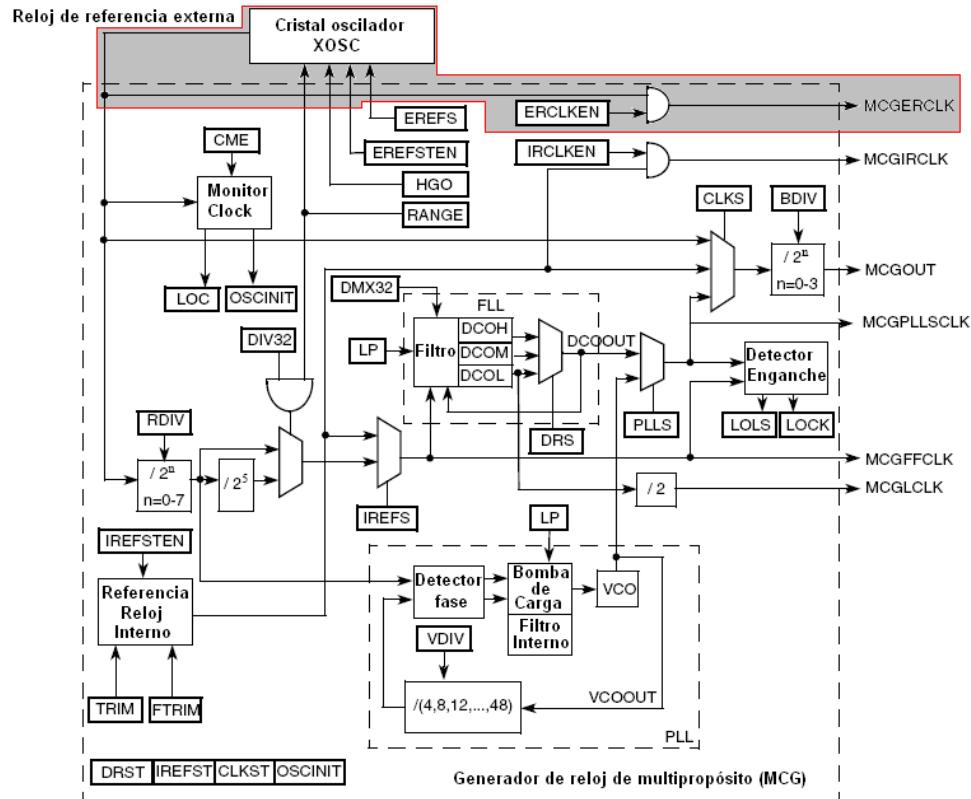
	7	6	5	4	3	2	1	0
Lectura	BDIV	RANGE	HGO	LP	EREFs	ERCLKEN	EREFSTEN	
Escritura	0	1	0	0	0	0	0	0

Reset: 0 1 0 0 0 0 0 0 0

**Figura 11.9. Registro MCGC2 del MCG.**

- Los bits BDIV dividen el reloj de bus interno de la MCU, como se detalla a continuación:
  - 00:** Divisor por 1
  - 01:** Divisor por 2 (valor por defecto)
  - 10:** Divisor por 4
  - 11:** Divisor por 8
- El bit RANGE selecciona el rango para el cristal o el cristal oscilador de referencia externa, como se detalla a continuación:
  - 0:** Cristal/Cristal oscilador entre 32KHz y 100KHz o 32KHz y 1MHz para fuente de reloj externa.
  - 1:** Cristal/Cristal oscilador entre 1MHz y 16MHz o 1MHz y 40MHz para fuente de reloj externa.
- El bit HGO controla el modo de operación del cristal externo, como se detalla a continuación:
  - 0:** Configura operación del cristal externo en bajo consumo.
  - 1:** Configura operación del cristal externo para alta ganancia.
- El bit LP elige la opción de bajo consumo, deshabilitando los módulos FLL y PLL (modo de sobrepaso), como se detalla a continuación:
  - 0:** Modo de sobrepaso deshabilitado.
  - 1:** Modo de sobrepaso habilitado.
- El bit EREFS selecciona la referencia externa, como se detalla a continuación:
  - 0:** Fuente de reloj externa
  - 1:** Oscilador externo (cristal o resonador)

- El bit ERCLKEN selecciona el reloj de referencia externa como reloj MCGERCLK. La Figura 11.10, detalla en línea roja y sombreado, este modo.



**Figura 11.10. Habilita reloj MCGIRCLK.**

- El bit EREFSTEN controla cuando el reloj de referencia externa permanece activo en el modo STOP, como se detalla a continuación:
  - 0:** El reloj de referencia externa permanece inactivo en modo STOP
  - 1:** El reloj de referencia externa permanece activo en modo STOP, para modos FEE, FBE, PEE, PBE o BLPE

- **Registro de ajuste del MCGTRIM:** La Figura 11.11 detalla el formato del registro MCGTRIM.

#### Registro MCGTRIM: (0xFFFF804A)

	7	6	5	4	3	2	1	0
Lectura								
Escritura					TRIM			

**Figura 11.11. Registro TRIM del MCG.**

El valor por defecto de este registro es 0x80, pero se sugiere al usuario que para obtener un valor de reloj homogéneo en una serie de MCU's con el mismo programa, es necesario almacenar en el MCGTRM el valor NVMCGTRM.

**MCGTRM = NVMCGTRM;**

Pero si el usuario sólo necesita programar un MCU, el valor almacenado en el registro TRIM controla la frecuencia de salida del reloj de referencia interna, ajustando su período interno. El valor del período aumenta de manera proporcional al valor del TRIM.

- **Registro de control y estado del MCG (MCGSC):** La Figura 11.12 detalla los bits asociados al registro MCGSC.

#### Registro MCGSC: (0xFFFF804B)

Lectura	7	6	5	4	3	2	1	0
Escritura	LOLS	LOCK	PLLST	IREFST	CLKST	OSCINIT		FTRIM <sup>1</sup>
Reset:	0	0	0	1	0	0	0	

**Figura 11.12. Registro MCGSC del MCG.**

- El bit LOLS indica cuando se ha perdido el enganche del circuito FLL o PLL. El bit puede ser aclarado ante un evento de RESET o escribiendo un “1” en LOLS. A continuación se detalla el comportamiento de este bit:
  - 0:** No se ha perdido el enganche del PLL o del FLL
  - 1:** Se ha perdido el enganche del PLL o del FLL
- El bit LOCK indica cuando se ha logrado el enganche del circuito FLL o PLL. El bit puede ser aclarado cambiando el valor de: DMX32, DRS[1:0] y IREFS bits en los modos FBE, FBI, FEE y FEI; DIV32 en los modos FBE y FEE; TRIM[7:0] en los modos FBI y FEI; RDIV[2:0] en los modos FBE, FEE, PBE y PEE; VDIV[3:0] en los modos PBE y PEE; y PLLS.
  - 0:** El circuito PLL o el FLL están desenganchados
  - 1:** El circuito PLL o el FLL están enganchados
- El bit PLLST indica el estado de selección del reloj del PLLS. El PLLS indica cuando se ha seleccionado el circuito PLL o FLL.
  - 0:** La fuente de reloj del PLLS es el reloj del FLL
  - 1:** La fuente de reloj del PLLS es el reloj del PLL
- El bit IREFST indica cual es la fuente de reloj de la referencia interna.
  - 0:** El reloj de referencia interna es la fuente de referencia externa
  - 1:** El reloj de referencia interna es la fuente de referencia interna

- El bit CLKST indica el modo actual del reloj derivado del MCG.
  - 00:** Ha sido seleccionada la salida del FLL
  - 01:** Ha sido seleccionado el reloj de referencia interna
  - 10:** Ha sido seleccionado el reloj de referencia externa
  - 11:** Ha sido seleccionada la salida del PLL
- El bit OSCINIT indica cuando se han completado los ciclos de inicialización del reloj externo, para los modos de FEE, FBE, PEE, PBE o BLPE. Este bit sólo se puede aclarar cuando se escribe un “0” en el bit EREFS o cuando se pasa al modo FEI, FBI y BLPI y ERCLKEN es cero.
- El bit FTRIM se utiliza para hacer un ajuste fino sobre el período del reloj de referencia interna.
  - 0:** Decrementa el período en una cantidad mínima
  - 1:** Incrementa el período en una cantidad mínima
- **Registro de control 3 del MCG (MCGC3):** La Figura 11.13 detalla los bits asociados al registro MCGC3.

**Registro MCGC3: (0xFFFF804C)**

Lectura	7	6	5	4	3	2	1	0
Escritura	LOLIE	PLLS	CME	DIV32		VDIV		
Reset:	0	0	0	0	0	0	0	1

**Figura 11.13. Registro MCGC3 del MCG.**

- El bit LOLIE habilita un posible evento de interrupción debido a la pérdida de enganche del reloj (LOLS).
  - 0:** No habilita interrupción debida a un LOLS
  - 1:** Habilita interrupción debida a un LOLS
- El bit PLLS es utilizado para la selección del circuito PLL o FLL.
  - 0:** Selecciona el sistema FLL
  - 1:** Selecciona el sistema PLL
- El bit CME determina si un evento de RESET debe ser generado por una pérdida del reloj de referencia externa. Este bit sólo trabaja cuando el sistema MCG está en los modos FEE, FBE, PEE, PBE y BLPE. No se debe cambiar el bit RANGE, cuando el bit CME = “1”.
  - 0:** El monitor del reloj externo se encuentra deshabilitado
  - 1:** Genera un evento de RESET ante la pérdida del reloj externo

- El bit DIV32 es utilizado para adicionar un divisor extra (por 32) al reloj de referencia externa, y que se aplicará al FLL. Lo anterior sólo es válido si el bit de RANGE = "1".
  - 0:** Deshabilita el divisor por 32
  - 1:** Habilita el divisor por 32
- Los bits VDIV se utilizan para dividir la frecuencia generada por el VCO (*Voltage Controller Oscilador*), como salida del PLL y que luego será comparada con la referencia interna.
  - 0000:** Reservado
  - 0001:** Divide por 4
  - 0010:** Divide por 8
  - 0011:** Divide por 12
  - 0100:** Divide por 16
  - 0101:** Divide por 20
  - 0110:** Divide por 24
  - 0111:** Divide por 28
  - 1000:** Divide por 32
  - 1001:** Divide por 36
  - 1010:** Divide por 40
  - 1011:** Divide por 44
  - 1100:** Divide por 48
  - 1101:** Reservado
  - 111X:** Reservado

- o **Registro de control 4 del MCG (MCGC4):** La Figura 11.14 detalla los bits asociados al registro MCGC4.

**Registro MCGC4: (0xFFFF804D)**

Lectura	7	6	5	4	3	2	1	0
Escritura	0	0	DMX32	0	0	0	DRST	
Reset:	0	0	0	0	0	0	0	0

**Figura 11.14. Registro MCGC4 del MCG.**

- El bit DMX32 es utilizado para fijar el valor máximo del rango del DCO con referencia de 32,768KHz.
  - 0:** El DCO opera en su rango normal elegido en el DRS
  - 1:** El DCO opera a su máxima frecuencia de acuerdo al valor de DRS
- Los bits DRST/DRS, en su modo de lectura, dicen el rango de DRS al cual se encuentra trabajando el DCO y en su modo de escritura, se usan para seleccionar el rango de operación del DCO.
  - 00:** El DCO opera rango bajo

- 01:** El DCO opera rango medio
- 10:** El DCO opera rango alto
- 11:** Reservado

**La Tabla 11.3.Rangos de operación del DCO (salida del FLL).**

DRS	DMX32	Rango de la Referencia	Factor FLL	Rango DCO <sup>1</sup>
<b>00</b>	<b>0</b>	31.25 - 39.0625 kHz	512	16 - 20 MHz
	<b>1</b>	32.768 kHz	608	19.92 MHz
<b>01</b>	<b>0</b>	31.25 - 39.0625 kHz	1024	32 - 40 MHz
	<b>1</b>	32.768 kHz	1216	39.85 MHz
<b>10</b>	<b>0</b>	<b>31.25 - 39.0625 kHz</b>	<b>1536</b>	<b>48-60 MHz</b>
	<b>1</b>	<b>32.768 kHz</b>	<b>1824</b>	<b>59.77 MHz</b>
<b>11</b>		Reservado		

<sup>1</sup> La Frecuencia del bus no debe exceder el rango máximo especificado para el DCO

- **Recetas para la inicialización del MCG:** Los pasos recomendados para la conmutación de un modo a otro del MCG son los siguientes.
  - Para pasar del modo por defecto FEI a los modos FEE o FBE seguir el siguiente procedimiento:
    - Habilitar la fuente de reloj externa con el registro MCGC2.
    - Si el bit MCGC2[RANGE] = 1, entonces poner a “1” el bit MCGC3[DIV32] para seleccionar los valores adecuados de los bits MCGC1[RDIV].
    - Escribir en el registro MCGC1 el modo deseado:
      - **Para FEE:** MCGC1[IREFS] = “0” y MCGC1[CLKS] = “00”.
      - **Para FBE:** MCGC1[IREFS] = “0” y MCGC1[CLKS] = “10”.
      - **Para IRCLKEN:** MCGC1[IRCLKEN] = “1”.
    - Verificar en el registro MCGSC el modo programado, de la siguiente manera:
      - **Para ERCLKEN:** Esperar a que el bit OSCINIT = “1”, indicando que el oscilador ha inicializado correctamente y es estable.
      - **Para FEE:** Asegurarse de que el bit MCGSC[IREFST] esté en “0” y el bit MCGSC[LOCK] sea “1”.
      - **Para FBE:** Asegurarse de que el bit MCGSC[IREFST] esté en “0”, el bit MCGSC[LOCK] sea “1” y los bits MCGSC[CLKST] estén en “10”.

- Configurar el rango de frecuencia de salida del VCO en el registro MCGC4, asegurándose de no exceder el rango máximo.

**NOTA:** Cuando se usa la referencia externa en un rango alto (RANGE = “1”) de FLL, se recomienda poner el bit DIV32 = “1”.

- Para pasar del modo por defecto FEI al modo FBI, seguir el siguiente procedimiento:
  - Poner los bits MCGC1[CLKS = “01”], con reloj de referencia interna.
  - Esperar a que los bits MCGSC[CLKST] esten a “01, indicando que el reloj de referencia interna ha sido correctamente seleccionado.
- **La forma de establecer la frecuencia programada en el MCG:** La Tabla 11.4 indica la frecuencia de salida del MCG, dependiendo de la configuración establecida.

**Tabla 11.4. Modos de reloj para el procesador.**

Modo del Reloj	$f_{MCGOUT}^1$	Nota
FEI (FLL referenciado internamente)	$(f_{int} * F) / B$	Typical $f_{MCGOUT} = 16$ MHz
FEE (FLL referenciado externamente)	$(f_{ext} / R * F) / B$	$f_{ext} / R$ debe estar en el rango 31.25 kHz a 39.0625 kHz
FBE (FLL sobrepasado externamente)	$f_{ext} / B$	$f_{ext} / R$ debe estar en el rango 31.25 kHz a 39.0625 kHz
FBI (FLL sobrepasado internamente)	$f_{int} / B$	Typical $f_{int} = 32$ kHz
PEE (PLL referenciado externamente)	$[(f_{ext} / R) * M] / B$	$f_{ext} / R$ debe estar en el rango 1 MHz a 2 MHz
PBE (PLL sobrepasado externamente)	$f_{ext} / B$	$f_{ext} / R$ debe estar en el rango 1 MHz a 2 MHz
BLPI (sobrepasado internamente y baja potencia)	$f_{int} / B$	
BLPE (sobrepasado externamente y baja potencia)	$f_{ext} / B$	

1: En las ecuaciones, **R** es el valor del divisor de la referencia seleccionada (RDIV), **B** es el divisor de la frecuencia del bus (BDIV), **F** es el factor seleccionado para el FLL (DRS y DMX32) y **M** es el divisor VDIV.

**Ejemplo con el MCG:** Se dispone de un cristal externo de 2MHz y es necesario tener 4MHz de frecuencia de bus interna, con referencia externa. El ejercicio

plantea un recorrido desde el modo por defecto FEI hasta el modo PEE, debido a la necesidad de obtener 4MHz internos. Los pasos a seguir son:

- **Transición del modo FEI al modo FBE:** Esta transición garantiza que se aplique un reloj externo estable a la máquina, mientras se hace la transición del circuito FLL al circuito PLL en modo PEE. Las condiciones para esta transición son:
  - **BDIV = “00”:** Divisor por 0 para el bus, es decir que se aplican 2MHz (cristal externo) a la máquina. Este reloj sustentará al MCU mientras se hace la transición FBE.
  - **RANGE = “1”:** Los 2MHz de cristal externo ubican el sistema en el rango alto.
  - **HGO = “1”:** Aplica ganancia alta para el cristal externo.
  - **EREFSS = “1”:** Selección del cristal externo como reloj temporal del bus interno.
  - **ERCLKEN:** Asegura la activación del reloj externo.
- **Iteración esperando inicialización del cristal externo:** No avanzar hasta tanto la señal OSCINIT =”1”.
- **Para RANGE = “1” es necesario poner DIV32 =”1”:** De esta manera se tendrá acceso a la combinación apropiada de los bits RDIV, mientras se está con el funcionamiento del FLL.
- **MCGC1 = “10001000”:**
  - **CLKS = “10”:** Selección del reloj externo, como fuente de reloj de bus interno.
  - **RDIV = “01”:**  $2\text{MHz} / 64 = 31.25\text{KHz}$ , quedando en el rango de  $31.25\text{KHz} - 39.0625\text{ kHz}$ , como condición del FLL.
  - **IREFS = “0”:** Para seleccionar referencia externa.
- **Iteración esperando que la referencia externa sea la fuente de referencia interna:** No avanzar hasta tanto la señal IREFST = “0”.
- **Iteración esperando que la fuente de referencia externa sea la salida MCGOUT:** No avanzar hasta tanto la señal CLKST = “10”.
- **Transición del modo FBE al modo BLPE y luego al modo PBE:** Esta transición garantiza que se pase primero a trabajar con un reloj de baja potencia en el circuito PLL (modo BLPE) y luego se haga la commutación al modo de PLL sobrepasado (PBE).
  - **LP = “1”:** Ejecuta la transición a BLPE.
  - **Transición de BLPE → PBE / MCGC3 = “01010010”:**

- **PLLS = “1”:** El divisor por 64 aplicado al FLL en modo FBE (RDIV = “01”) es aplicado al PLL en modo PBE, con un equivalente de 2 (ver tabla 13.2). Entonces,  $2\text{MHz} / 2 = 1\text{MHz}$ , que será el reloj BLPE y que alimentará al sistema mientras se realiza la conmutación a modo PBE.
  - **VDIV = “0010”:** Aunque no es importante definir el valor de VDIV, esto hace que la referencia de reloj BLPE = 1MHz tenga un factor de multiplicación por 8. Por lo anterior la salida del PLL en modo PBE, de ser habilitada, será de  $1\text{MHz} * 8 = 8\text{MHz}$ .
- **LP = “0”:** Ejecuta la transición de BLPE a BPE.
  - **Iterar hasta que se produzca el cambio de BLPE a PBE:** bit PLLST = “1”. La salida del PLL es de 8MHz, aunque se encuentra sobrepasado del sistema.
  - **Iterar hasta que el reloj del PLL sea estable:** bit LOCK = “1”.
- **Transición del modo PBE al modo PEE / MCGC1 = “00011000”:** Esta es la transición final.
    - **CLKS = “00”:** La salida del PLL como fuente de reloj de bus interno.
    - **Esperar a que CLKST = “11”:** Indicando que la salida del PLL es la fuente del MCGOUT. De esta manera el  $\text{MCGOUT} = 8\text{MHz} / 2 = 4\text{MHz}$ , que era el propósito final del ejercicio.

## 11.2. EJERCICIO CON EL MCG

Con la fuente de reloj por defecto, referencia interna y FEE (modo FEI), detectar la frecuencia a la cual se encuentra trabajando la máquina. El programa, desarrollado en C, es mostrado a continuación y es posible tener acceso a éste vía sitio WEB del texto con el nombre de MCG\_INI.

**NOTA:** Se recomienda al usuario revisar cuidadosamente el estado de los puentes del DEMOJM, con el objetivo de hacer compatible el ejercicio actual y de esta manera liberar los pines comprometidos en el circuito.

Para este ejercicio es indispensable contar con un osciloscopio, como comprobación de la frecuencia de trabajo interno (MCGOUT).

```
/*
 * MCG_INI: Ejemplo sobre la inicializacion del MCG
 */
/* main.c */
/*
 * Fecha: Mayo 2, 2008
 */
/* V1.0, Rev 0 por Diego Munera
 */
/* Asunto: Implementar un codigo en C que realice la inicializacion del modulo */

```

```

/*
 * MCG en modo FEI y verificar la frecuencia de trabajo generando una señal      */
/* periodica con el modulo TPM1 (TPM1CH0)                                         */
*/
/*
 * Nota: Las tildes en los comentarios son omitidas para efectos de compabilidad   */
/* con otros editores.                                                               */
//****************************************************************************
#include "derivative.h"          //Incluye declaracion de perifericos
#include <hidef.h>                //Macro para habilitar las interrupciones

/* Declaracion de funciones */
void Onda_Cuadrada(void);        //Declaracion de funcion generadora de una
                                  //señal cuadrada

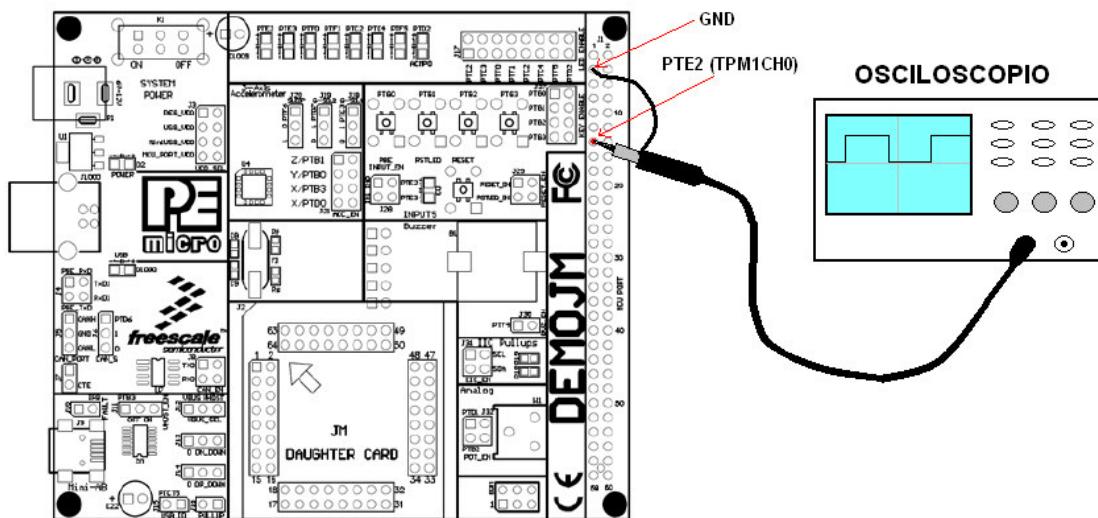
/* Programa principal */
void main (void){                //Funcion principal
    MCGC1=0x04;                  //Reloj en modo FEI y divisor por 1
    MCGC2=0x00;
    Onda_Cuadrada(void);         //Inicializa señal cuadrada

    for(;;) {                    //Iteración infinita
        __RESET_WATCHDOG(); //Aclare contador del COP
    }
}

/* Funcion de generacion de señal cuadrada */
void Onda_Cuadrada(void) {
    TPM1SC = 0x08;               //Temporizador TPM1, fuente reloj interna
    TPM1C0SC = 0x28;              //PWM alineado a flanco e inicia en alto, canal 0 (PTE2)
    TPM1MOD = 1000;                //Periodo del PWM = MCGOUT / 1000
    TPM1C0V = 500;                 //Ciclo de trabajo del PWM = 50 %
}

```

Ubicando la punta del osciloscopio en el pin PTE2 (TPM1CH0) se puede visualizar una señal cuadrada, cuya frecuencia multiplicada por 1000 corresponde a la frecuencia de trabajo de bus interno de la MCU (ver circuito de la Figura 11.15).



**Figura 11.15. Comprobación de la frecuencia del MCG.**

Otra forma de conocer el reloj interno (BUSCLK) de trabajo del MCU es verificar en el depurador (*Debug*) la frecuencia final reportada, después de introducir en el programa las siguientes líneas:

```
//Inicializacion del reloj en modo FEI:  
MCGTRM = NVMCGTRM + 52;           //Tome valor programado de fabrica del TRIM y  
                                     //sume una constante para obtener un reloj  
                                     //cercano a 25 MHz  
MCGC1 = 0x04;                      //Modo FEI divisor por 1  
MCGC2 = 0x00;                      //Desactiva modo de reloj externo  
MCGC4 = 0x22;                      //Rango alto del DCO
```

Para el momento de la depuración, el sistema reportará un reloj de 24999945 Hz, que deberá ser igual al reportado por otras máquinas programadas con la misma inicialización del reloj. La desviación respecto de 25 MHz es de 0,00022%, que será bastante aceptable para cálculos con módulos de temporización y de comunicaciones seriales.

El valor sumado al registro **NVMCGTRM** es al tanteo, debido a que no existe una formula para el cálculo de un valor exacto.

### **11.3 REFERENCIAS**

- MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual.  
Freescale Semiconductor. Rev 1, 2008.
- Microcontroller MCF51JM128 ColdFire Data Sheet. Freescale Semiconductor.  
Rev 0, 2008.

### **11.4. PREGUNTAS**

- Describa los diferentes modos de operación del módulo MCG.
- ¿Cómo se puede garantizar un reloj homogéneo, para varias máquinas que deberán ser programadas con el mismo programa?
- ¿Qué es un DCO?
- ¿Cuál es la máxima frecuencia de cristal externo que se puede instalar al JM128?
- Elabore una rutina que partiendo de un cristal externo de 5 MHz, se pueda obtener un reloj de salida (MCGOUT) de 25Mhz.
- ¿Cómo obtener un reloj de 25.5MHz, trabajando en modo FEI y de qué valor sería la cantidad que habría que adicionar a la constante **NVMCGTRM** ?

# CAPÍTULO 12

## Los Puertos de Entrada y Salida

- 12.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**
- 12.2. La función KBI (*KeyBoard Interrupt*)**
- 12.3. Ejercicio con los puertos: Inicialización de una pantalla de cristal líquido (LCD) y migración de 8 a 32 bits.**
- 12.4. Referencias**
- 12.5. Preguntas**

## 12.1. DIAGRAMA EN BLOQUES, EXPLICACIÓN DEL FUNCIONAMIENTO Y REGISTROS ASOCIADOS

La máquina MCF51JM128 contiene hasta 66 pines I/O en la versión de 80 pines, distribuidos en 9 puertos (A, B, C, D, E, F, G, H, J) o hasta 56 pines I/O en la versión de 64 pines. Adicionalmente, los puertos C y E tienen control pin a pin, de tal manera que se pueden forzar a “1” (SET), a “0” (RESET) o complementar su valor (TOGGLE).

La mayoría de los pines cumplen una doble función y algunos una triple función, por ejemplo: el pin PTD3 trabaja como pin de propósito general (I/O) o como una entrada de la función KBI (KBIP3) o como una entrada del conversor análogo a digital ADC (ADP10).

Por defecto, cuando la máquina ha salido del estado de RESET, los pines son configurados como entradas (alta impedancia). Lo anterior se debe a la seguridad que brinda el establecimiento de un pin como entrada, en vez de salida. Si un pin es inicializado como salida, se corre con el riesgo de aplicarle al pin una diferencia de potencial e incurrir en daños del MCU.

**NOTA:** Todos aquellos pines que no se utilicen en un proyecto deberán ser definidos a algún estado, siempre y cuando conserven su dirección como entrada. Lo anterior es debido a la exposición del sistema ante señales espúreas o ruido EMI y/o a consumos extras por pines flotantes.

- **Diagrama en bloques de un pin:** La Figura 12.1 muestra un diagrama simplificado de la composición de un pin de un puerto.

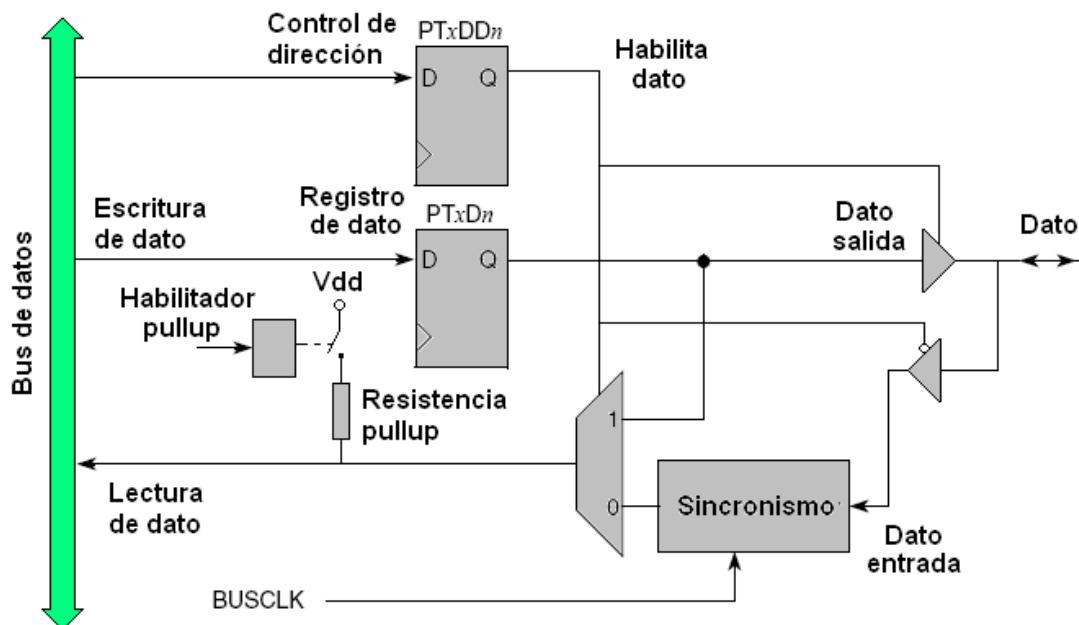


Figura 12.1. Diagrama en bloque de pin I/O.

Todo puerto está compuesto por un registro que configura la dirección de cada pin asociado a este. Los registros PTxDD contienen los bits PTxDDn, que habilitan la dirección de un pin.

Al registro que establece el valor lógico del pin, sea como entrada o salida, se le conoce como el PTxD y los bits PTxDn contienen dicho valor.

**NOTA:** Es recomendable establecer primero el valor lógico del bit (o de los bits), antes de resolver la dirección de este (o estos). Ejemplo:

PTAD = 0x00;	//salidas del puerto A en "0"
PTADD = 0xF0;	//define el nibble de mayor peso del puerto A
	//como salida

- **Registros asociados a los puertos I/O:** Las funciones básicas, que se pueden definir alrededor de un puerto I/O, son soportadas por los siguientes registros:
  - **Registro de datos (PTxD):** La Figura 12.2 muestra la configuración general de un registro de datos de un puerto x.

**Registro PTxD**

	7	6	5	4	3	2	1	0
Lectura	PTxD7	PTxD6	PTxD5	PTxD4	PTxD3	PTxD2	PTxD1	PTxD0
Escritura	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0

**Figura 12.2. Registro de datos de un puerto I/O x.**

**PTxDn:** Establece el estado de cada pin I/O del correspondiente puerto. Al salir del estado de RESET, los pines son configurados como entradas en alta impedancia (*pullups* deshabilitados). Los pines de salida se actualizan al valor de este registro, una vez se haya definido su dirección.

La Figura 12.3 muestra la disposición de este registro para todos los puertos de la máquina.

0x(FF)FF_8000	PTAD	PTAD7	PTAD6	PTAD5	PTAD4	PTAD3	PTAD2	PTAD1	PTAD0
0x(FF)FF_8002	PTBD	PTBD7	PTBD6	PTBD5	PTBD4	PTBD3	PTBD2	PTBD1	PTBD0
0x(FF)FF_8004	PTCD	PTCD7	PTCD6	PTCD5	PTCD4	PTCD3	PTCD2	PTCD1	PTCD0
0x(FF)FF_8006	PTDD	PTDD7	PTDD6	PTDD5	PTDD4	PTDD3	PTDD2	PTDD1	PTDD0
0x(FF)FF_8008	PTED	PTED7	PTED6	PTED5	PTED4	PTED3	PTED2	PTED1	PTED0
0x(FF)FF_800A	PTFD	PTFD7	PTFD6	PTFD5	PTFD4	PTFD3	PTFD2	PTFD1	PTFD0
0x(FF)FF_800C	PTGD	PTGD7	PTGD6	PTGD5	PTGD4	PTGD3	PTGD2	PTGD1	PTGD0
0x(FF)FF_9874	PTH	—	—	—	PTHD4	PTHD3	PTHD2	PTHD1	PTHD0
0x(FF)FF_9876	PTJD	—	—	—	PTJD4	PTJD3	PTJD2	PTJD1	PTJD0

**Figura 12.3. Detalle de los registros de datos de la MCU.**

- **Registro de dirección de pin (PTxDD):** La Figura 12.4 muestra la configuración general de un registro de dirección de pines de un puerto x.

#### Registro PTxDD

	7	6	5	4	3	2	1	0
Lectura Escritura	PTxDD7	PTxDD6	PTxDD5	PTxDD4	PTxDD3	PTxDD2	PTxDD1	PTxDD0
Reset:	0	0	0	0	0	0	0	0

Figura 12.4. Registro de dirección de pines de un puerto I/O x.

**PTxDDn:** Establece la dirección de cada pin I/O del correspondiente puerto.

**0:** El pin es definido como entrada (valor por defecto)

**1:** El pin es definido como salida

La Figura 12.5 muestra la disposición de este registro para todos los puertos de la máquina.

0x(FF)FF_8001	PTADD	PTADD7	PTADD6	PTADD5	PTADD4	PTADD3	PTADD2	PTADD1	PTADD0
0x(FF)FF_8003	PTBDD	PTBDD7	PTBDD6	PTBDD5	PTBDD4	PTBDD3	PTBDD2	PTBDD1	PTBDD0
0x(FF)FF_8005	PTCDD	PTCDD7	PTCDD6	PTCDD5	PTCDD4	PTCDD3	PTCDD2	PTCDD1	PTCDD0
0x(FF)FF_8007	PTDDD	PTDDD7	PTDDD6	PTDDD5	PTDDD4	PTDDD3	PTDDD2	PTDDD1	PTDDD0
0x(FF)FF_8009	PTEDD	PTEDD7	PTEDD6	PTEDD5	PTEDD4	PTEDD3	PTEDD2	PTEDD1	PTEDD0
0x(FF)FF_800B	PTFDD	PTFDD7	PTFDD6	PTFDD5	PTFDD4	PTFDD3	PTFDD2	PTFDD1	PTFDD0
0x(FF)FF_800D	PTGDD	PTGDD7	PTGDD6	PTGDD5	PTGDD4	PTGDD3	PTGDD2	PTGDD1	PTGDD0
0x(FF)FF_9875	PTHDD	—	—	—	PTHDD4	PTHDD3	PTHDD2	PTHDD1	PTHDD0
0x(FF)FF_9877	PTJDD	—	—	—	PTJDD4	PTJDD3	PTJDD2	PTJDD1	PTJDD0

Figura 12.5. Detalle de los registros de dirección de pines.

- **Registro de habilitación de *pullups*<sup>31</sup> (PTxPE):** La Figura 12.6 muestra la configuración general de un registro de habilitación de *pullups* para los pines de entrada de un puerto x.

#### Registro PTxPE

	7	6	5	4	3	2	1	0
Lectura Escritura	PTxPE7	PTxPE6	PTxPE5	PTxPE4	PTxPE3	PTxPE2	PTxPE1	PTxPE0
Reset:	0	0	0	0	0	0	0	0

Figura 12.6. Registro de habilitación de *pullups*.

**PTxPEn:** Establece una resistencia de *pullup* en el pin de entrada indicado.

**0:** El *pullup* interno para este pin es deshabilitado (valor por defecto)

**1:** El *pullup* interno para este pin es habilitado

<sup>31</sup> *Pullup*: Resistencia ubicada entre la señal y la fuente de la lógica del sistema (Vdd).

La Figura 12.7 muestra la disposición de este registro para todos los puertos de la máquina.

**NOTA:** Los *pullups* son típicamente de  $45\text{K}\Omega$

0x(FF)FF_9840	<b>PTAPE</b>	PTAPE7	PTAPE6	PTAPE5	PTAPE4	PTAPE3	PTAPE2	PTAPE1	PTAPE0
0x(FF)FF_9844	<b>PTBPE</b>	PTBPE7	PTBPE6	PTBPE5	PTBPE4	PTBPE3	PTBPE2	PTBPE1	PTBPE0
0x(FF)FF_9848	<b>PTCPE</b>	PTCPE7	PTCPE6	PTCPE5	PTCPE4	PTCPE3	PTCPE2	PTCPE1	PTCPE0
0x(FF)FF_984C	<b>PTDPE</b>	PTDPE7	PTDPE6	PTDPE5	PTDPE4	PTDPE3	PTDPE2	PTDPE1	PTDPE0
0x(FF)FF_9850	<b>PTEPE</b>	PTEPE7	PTEPE6	PTEPE5	PTEPE4	PTEPE3	PTEPE2	PTEPE1	PTEPE0
0x(FF)FF_9854	<b>PTFPE</b>	PTFPE7	PTFPE6	PTFPE5	PTFPE4	PTFPE3	PTFPE2	PTFPE1	PTFPE0
0x(FF)FF_9858	<b>PTGPE</b>	PTGPE7	PTGPE6	PTGPE5	PTGPE4	PTGPE3	PTGPE2	PTGPE1	PTGPE0
0x(FF)FF_985C	<b>PTHPE</b>	—	—	—	PTHPE4	PTHPE3	PTHPE2	PTHPE1	PTHPE0
0x(FF)FF_9860	<b>PTJPE</b>	—	—	—	PTJPE4	PTJPE3	PTJPE2	PTJPE1	PTJPE0

**Figura 12.7. Detalle de los registros de *pullups* de pines de entrada.**

- **Registro de habilitación del *slew rate*<sup>32</sup> (PTxSE):** La Figura 12.8 muestra la configuración general de un registro de habilitación del *slew rate* para los pines de salida de un puerto x.

#### Registro PTxSE

Lectura Escritura	7	6	5	4	3	2	1	0
Reset:	1	1	1	1	1	1	1	1

**Figura 12.8. Registro de habilitación del *slew rate*.**

**PTxSEN:** Limita la tasa de ascenso de la señal eléctrica de un pin de salida. El efecto es blindar el pin contra fenómenos del tipo EMI (*ElectroMagnetic Interferences*).

**0:** El *slew rate* interno para este pin es deshabilitado

**1:** El *slew rate* interno para este pin es habilitado (valor por defecto)

La Figura 12.9 muestra la disposición de este registro para todos los puertos de la máquina.

<sup>32</sup> *Slew rate*: Tasa de ascenso de la señal de salida de un pin de un puerto [V/us].

0x(FF)FF_9841	PTASE	PTASE7	PTASE6	PTASE5	PTASE4	PTASE3	PTASE2	PTASE1	PTASE0
0x(FF)FF_9845	PTBSE	PTBSE7	PTBSE6	PTBSE5	PTBSE4	PTBSE3	PTBSE2	PTBSE1	PTBSE0
0x(FF)FF_9849	PTCSE	PTCSE7	PTCSE6	PTCSE5	PTCSE4	PTCSE3	PTCSE2	PTCSE1	PTCSE0
0x(FF)FF_984D	PTDSE	PTDSE7	PTDSE6	PTDSE5	PTDSE4	PTDSE3	PTDSE2	PTDSE1	PTDSE0
0x(FF)FF_9851	PTESE	PTESE7	PTESE6	PTESE5	PTESE4	PTESE3	PTESE2	PTESE1	PTESE0
0x(FF)FF_9855	PTFSE	PTFSE7	PTFSE6	PTFSE5	PTFSE4	PTFSE3	PTFSE2	PTFSE1	PTFSE0
0x(FF)FF_9859	PTGSE	PTGSE7	PTGSE6	PTGSE5	PTGSE4	PTGSE3	PTGSE2	PTGSE1	PTGSE0
0x(FF)FF_985D	PTHSE	—	—	—	PTHSE4	PTHSE3	PTHSE2	PTHSE1	PTHSE0
0x(FF)FF_9861	PTJSE	—	—	—	PTJSE4	PTJSE3	PTJSE2	PTJSE1	PTJSE0

**Figura 12.9. Detalle de los registros de *slew rate* sobre pines de salida.**

- **Registro de habilitación del *drive strength*<sup>33</sup> (PTxDS):** La Figura 12.10 muestra la configuración general de un registro de habilitación del *drive strength* para los pines de salida de un puerto x.

**Registro PTxDS**

	7	6	5	4	3	2	1	0
Escritura	PTxDS7	PTxDS6	PTxDS5	PTxDS4	PTxDS3	PTxDS2	PTxDS1	PTxDS0
Lectura	0	0	0	0	0	0	0	0

Reset: 0 0 0 0 0 0 0 0 0

**Figura 12.10. Registro de habilitación del *drive strength*.**

**PTxDSn:** Habilita el drenaje o suministro de altas corrientes para un pin de salida. El efecto sobre la EMC (*ElectroMagnetic Compliance*, contrario a la EMI) es negativo y se debe ser cuidadoso en su manejo.

**0:** Selecciona una baja capacidad de manejo de corriente a la salida (valor por defecto).

**1:** Selecciona una alta capacidad de manejo de corriente a la salida

La Figura 12.11 muestra la disposición de este registro para todos los puertos de la máquina.

0x(FF)FF_9842	PTADS	PTADS7	PTADS6	PTADS5	PTADS4	PTADS3	PTADS2	PTADS1	PTADS0
0x(FF)FF_9846	PTBDS	PTBDS7	PTBDS6	PTBDS5	PTBDS4	PTBDS3	PTBDS2	PTBDS1	PTBDS0
0x(FF)FF_984A	PTCDS	PTCDS7	PTCDS6	PTCDS5	PTCDS4	PTCDS3	PTCDS2	PTCDS1	PTCDS0
0x(FF)FF_984E	PTDDS	PTDDS7	PTDDS6	PTDDS5	PTDDS4	PTDDS3	PTDDS2	PTDDS1	PTDDS0
0x(FF)FF_9852	PTEDS	PTEDS7	PTEDS6	PTEDS5	PTEDS4	PTEDS3	PTEDS2	PTEDS1	PTEDS0
0x(FF)FF_9856	PTFDS	PTFDS7	PTFDS6	PTFDS5	PTFDS4	PTFDS3	PTFDS2	PTFDS1	PTFDS0
0x(FF)FF_985A	PTGDS	PTGDS7	PTGDS6	PTGDS5	PTGDS4	PTGDS3	PTGDS2	PTGDS1	PTGDS0
0x(FF)FF_985E	PTHDS	—	—	—	PTHDS4	PTHDS3	PTHDS2	PTHDS1	PTHDS0
0x(FF)FF_9862	PTJDS	—	—	—	PTJDS4	PTJDS3	PTJDS2	PTJDS1	PTJDS0

**Figura 12.11. Detalle de los registros de *drive strength* sobre pines de salida.**

<sup>33</sup> *Drive Strength:* Fortalecimiento de la capacidad de salida de un pin eléctrico, referido a la capacidad de corriente de drenaje o de suministro que puede manipular el pin.

**NOTA:** La máxima corriente que entrega la MCU desde los puertos, sumadas todas las corrientes parciales por cada pin de salida, es de 100mA @ 5V (60mA @ 3V). El usuario deberá tener este factor en cuenta, cuando esté utilizando la propiedad del *drive strength*.

- **Registro de habilitación de un filtro a la entrada (PTxIFE):** La Figura 12.12 muestra la configuración general de un registro de habilitación de un filtro pasa bajos a un pin de entrada al MCU.

**Registro PTxIFE**

R	7	6	5	4	3	2	1	0
W	PTxIFE7	PTxIFE6	PTxIFE5	PTxIFE4	PTxIFE3	PTxIFE2	PTxIFE1	PTxIFE0
Reset:	1	1	1	1	1	1	1	1

**Figura 12.12. Registro de habilitación del filtro de entrada.**

**PTxIFEn:** Habilita se aplique un filtro pasa bajos a un pin configurado como entrada.

**0:** No hay un filtro pasabajos conectado a la entrada

**1:** Hay un filtro pasabajos conectado a la entrada (valor por defecto)

La Figura 12.13 muestra la disposición de este registro para todos los puertos de la máquina.

0x(FF)FF_9843	PTAIFE	PTAIFE7	PTAIFE6	PTAIFE5	PTAIFE4	PTAIFE3	PTAIFE2	PTAIFE1	PTAIFE0
0x(FF)FF_9847	PTBIFE	PTBIFE7	PTBIFE6	PTBIFE5	PTBIFE4	PTBIFE3	PTBIFE2	PTBIFE1	PTBIFE0
0x(FF)FF_984B	PTCIFE	PTCIFE7	PTCIFE6	PTCIFE5	PTCIFE4	PTCIFE3	PTCIFE2	PTCIFE1	PTCIFE0
0x(FF)FF_984F	PTDIFE	PTDIFE7	PTDIFE6	PTDIFE5	PTDIFE4	PTDIFE3	PTDIFE2	PTDIFE1	PTDIFE0
0x(FF)FF_9853	PTEIFE	PTEIFE7	PTEIFE6	PTEIFE5	PTEIFE4	PTEIFE3	PTEIFE2	PTEIFE1	PTEIFE0
0x(FF)FF_9857	PTFIFE	PTFIFE7	PTFIFE6	PTFIFE5	PTFIFE4	PTFIFE3	PTFIFE2	PTFIFE1	PTFIFE0
0x(FF)FF_985B	PTGIFE	PTGIFE7	PTGIFE6	PTGIFE5	PTGIFE4	PTGIFE3	PTGIFE2	PTGIFE1	PTGIFE0
0x(FF)FF_985F	PTHIFE	—	—	—	PTHIFE4	PTHIFE3	PTHIFE2	PTHIFE1	PTHIFE0
0x(FF)FF_9863	PTJIFE	—	—	—	PTJIFE4	PTJIFE3	PTJIFE2	PTJIFE1	PTJIFE0

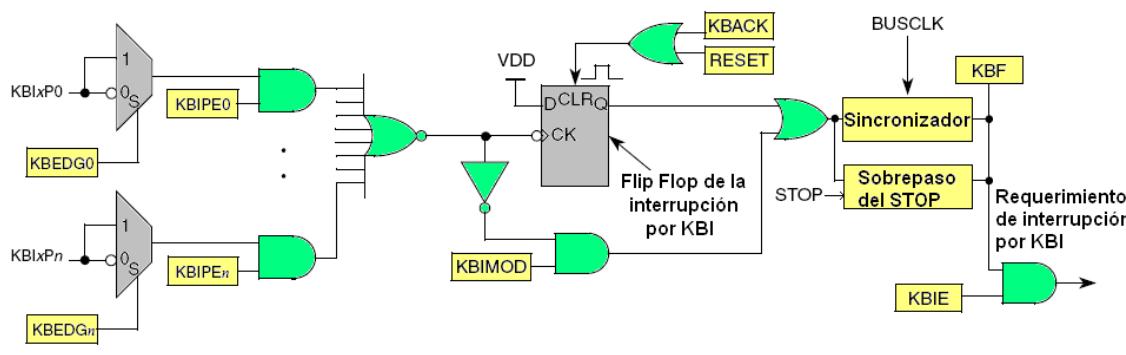
**Figura 12.13. Detalle de los registros de inserción de un filtro pasa bajos sobre pines de entrada.**

## 12.2. LA FUNCIÓN KBI

Esta función está intimamente ligada a los pines I/O, permite generar interrupción a la MCU por varias líneas de entrada y por los puertos B, D y G.

El aprovechamiento de esta función radica en la posibilidad de implementar la lectura de un teclado por el mecanismo de las interrupciones y generar a la máquina un *wakeup* (despertar) desde los modos de bajo consumo.

El diagrama de la Figura 12.14 muestra la forma de como es atendido un evento de interrupción desde un teclado conectado a la MCU.



**Figura 12.14. Diagrama de bloque función KBI**

Cada pin con posibilidad de generar interrupción es habilitado independientemente vía los bits KBIxPE[KBIPEn].

La interrupción puede ser generada por flanco o flanco y nivel, y mediante los bits KBIxES[KBEDGn] se selecciona si la señal obedece a flanco de bajada o subida y/o a nivel bajo o alto. Lo anterior se combina con el bit de modo KBIxSC[KBIMOD], mediante el cual se selecciona si se va a trabajar con flanco únicamente o con flanco y nivel.

Existe la posibilidad de hacer un encuestamiento (*polling*) sobre la bandera KBIxSC[KBF], para detectar cuando ha ocurrido un evento de KBI. Luego de confirmar el evento es necesario aclarar manualmente el estado del *Flip Flop* del KBI, haciendo “1” el bit KBIxSC[KBACK]. La MCU una vez ha salido de RESET, aclara el estado del *Flip Flop*.

Si por el contrario es necesario trabajar por interrupción, el usuario deberá poner a “1” el bit KBIxSC[KBIE]. En este punto, el estado del *Flip Flop* será aclarado una vez sea atendida la interrupción.

La bandera KBIxSC[KBF] es aclarada siempre que se está en el modo de flanco, ya sea que se trabaje por *polling* o por interrupción; sin embargo si se está en el modo de flanco y nivel, la bandera KBIxSC[KBF] no será aclarada hasta tanto no se abandone el modo y el usuario deberá tener especial cuidado en su manejo.

La función KBI puede sobrepasar el sincronismo de la señal eléctrica a la entrada ante un STOP. Esta característica hace que el KBI pueda sacar a la máquina estando en modo STOP. A continuación se describen los registros asociados al KBI.

- **Registro del KBI1:** La Figura 12.15 muestra el registro de los pines asociados al KBI y su estado eléctrico. Este registro deberá ser leído para saber quién ha generado un evento de KBI.

#### Registro KBI1

Pin del Puerto	PTG3	PTG2	PTB5	PTB4	PTD3	PTD2	PTG1	PTG0
Pin del KBI1	KBI1P7	KBI1P6	KBI1P5	KBI1P4	KBI1P3	KBI1P2	KBI1P1	KBI1P0

Figura 12.15. Registro KBI1

- **Registro de estado y control KBI1SC:** La Figura 12.16 muestra el registro de estado y control del KBI1.

#### Registro KBI1SC: (0xFFFF801C)

	7	6	5	4	3	2	1	0
Lectura	0	0	0	0	KBF	0	KBIE	KBIMOD
Escritura						0		
Reset:	0	0	0	0	0	0	0	0

Figura 12.16. Registro KBI1SC

En donde:

- **KBF:** Bandera que indica cuando un evento de KBI ha sido detectado.
  - 0:** No hay presencia de evento por KBI
  - 1:** Hay un evento de KBI presente
- **KBACK:** Bit utilizado para aclarar el estado del *Flip Flop* del KBI, como reconocimiento a un evento de KBI
  - 0:** No tiene efecto
  - 1:** Aclara el estado del *Flip Flop* del KBI
- **KBIE:** Bit utilizado habilitar la opción de generar un evento de interrupción debido a un evento de KBI.
  - 0:** Inhabilita un requerimiento de interrupción por KBI
  - 1:** Habilita un requerimiento de interrupción por KBI
- **KBIMOD:** Controla el modo de detección de la función KBI (ver Figura 14.17).
  - 0:** Detección por flanco únicamente
  - 1:** Detección por flanco y nivel

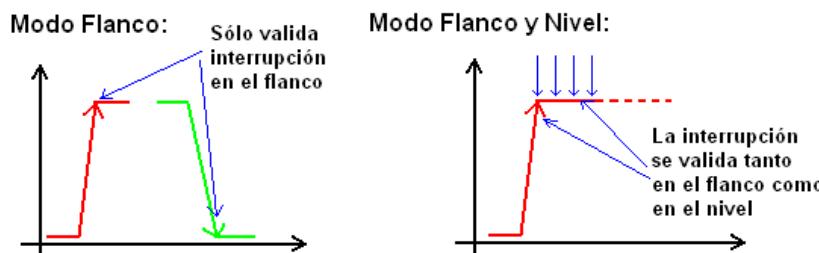


Figura 12.17. Modos de validación de la interrupción por KBI.

- **Registro de selección de pin del KBI (KBI1PE):** La Figura 12.18 muestra el registro de selección del pin para la función KBI.

**Registro KBI1PE: (0xFFFF801D)**

Lectura Escritura	7 KBIPE7	6 KBIPE6	5 KBIPE5	4 KBIPE4	3 KBIPE3	2 KBIPE2	1 KBIPE1	0 KBIPE0
Reset:	0	0	0	0	0	0	0	0

**Figura 12.18. Registro KBI1PE**

En donde:

**KBIPEn:** Habilita el pin asociado a la función KBI.

**0:** El pin no está habilitado

**1:** El pin está habilitado como función KBI

- **Registro de selección del flanco y nivel del KBI (KBI1ES):** La Figura 12.19 muestra el registro de selección del flanco y nivel del pin, para la función KBI.

**Registro KBI1ES: (0xFFFF801E)**

Lectura Escritura	7 KBEDG7	6 KBEDG6	5 KBEDG5	4 KBEDG4	3 KBEDG3	2 KBEDG2	1 KBEDG1	0 KBEDG0
Reset:	0	0	0	0	0	0	0	0

**Figura 12.19. Registro KBI1ES**

En donde:

**KBEDGn:** Selecciona la polaridad de la señal eléctrica aplicada al pin del KBI y a su vez selecciona el *pullup/pulldown* asociado al pin.

**0:** Habilita flanco de bajada y nivel bajo, para la detección de evento de KBI y a su vez habilita *pullup* al pin asociado.

**1:** Habilita flanco de subida y nivel alto, para la detección de evento de KBI y a su vez habilita *pulldown* al pin asociado.

### 12.3. EJERCICIO CON LOS PUERTOS

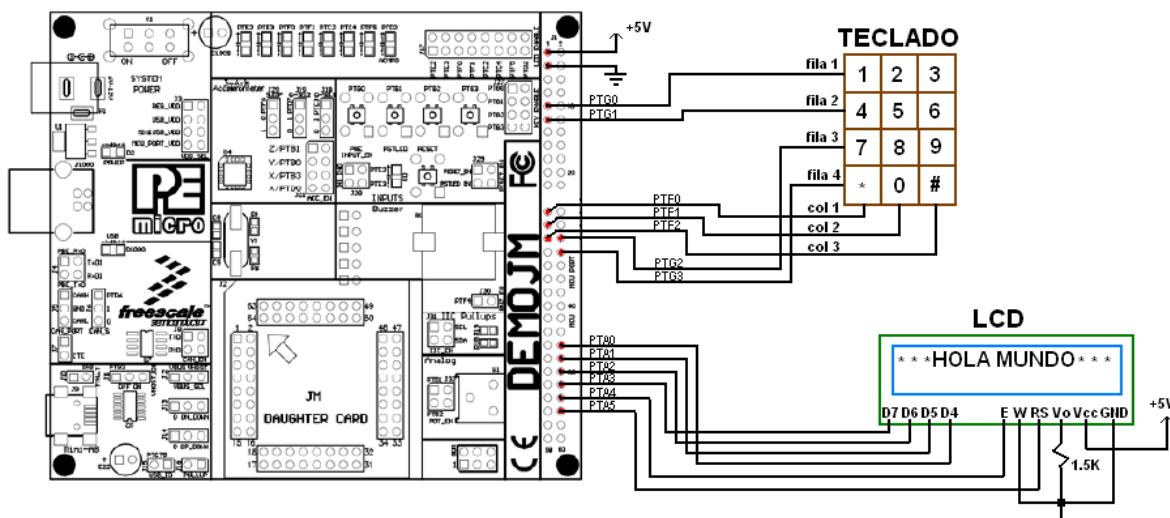
El circuito de la Figura 12.20 detalla la aplicación a implementar, como ejercicio del manejo de los pines de puertos y la función KBI.

**NOTA:** Se recomienda al usuario revisar cuidadosamente el estado de los puentes del DEMOJM, con el objetivo de hacer compatible el ejercicio actual y de esta manera liberar los pines comprometidos en el circuito.

El objetivo del ejercicio es inicializar una pantalla de cristal líquido (LCD) e implementar la lectura de un teclado matricial, utilizando la función KBI. El proyecto LCD\_KBI\_8BIT.mcp fue implementado en CodeWarrior 6.1 con el microcontrolador de 8 bits MC9S08JM60 y luego fue migrado al microcontrolador MCF51JM128 con el nombre de LCD\_KBI.mcp.

En la migración sólo hay que considerar que el registro PTxIFE no existe para las máquinas de 8 bits y los registros KBISC, KBIPe y KBIES, son nombrados en las máquinas de 32 como KBI1SC, KBI1PE y KBI1ES, respectivamente.

Un aspecto adicional sería el aumento de los retardos en los diferentes procesos, tanto en las rutinas de inicialización del LCD como las de manipulación del teclado.



**Figura 12.20.** Circuito del ejercicio con puertos y función KBI.

Un listado en C para la máquina MCF51JM128 se presenta a continuación:

```
*****  
/*LCD_KBI: Ejemplo sobre la utilización del modulo de puertos y la funcion KBI.  
/* main.c  
/*  
/*Fecha: Mayo 10, 2008  
/*  
/*V1.0, Rev 0 por Diego Múnera  
/*  
/*Asunto: Implementar un codigo en C que inicialice una pantalla de cristal líquido (LCD) y haga la lectura de un teclado  
/*matricial utilizando la funcion KBI. Al presionarse una tecla esta debe aparecer en la linea baja del LCD (se excluyen)  
/*las teclas * y #).  
/*  
/*Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros editores.  
/*  
*****  
  
*****  
/* DISTRIBUCION DE LOS PINES DE PUERTOS  
/*  
*****
```

```

/*
 * PUERTO * PIN * I/O * TIPO * INICIA      *COMENTARIO
 */
/*PTA * - * - * - * - *LCD
/*  *PTA0 * O * D * 1 *D4 del LCD
/*  *PTA1 * O * D * 1 *D5 del LCD
/*  *PTA2 * O * D * 1 *D6 del LCD
/*  *PTA3 * O * D * 1 *D7 del LCD
/*  *PTA4 * O * D * 1 *Señal ENABLE del LCD
/*  *PTA5 * O * D * 1 *Señal RS del LCD
*/
/*PTB * - * - * - *No usado
*/
/*PTC * - * - * - *No usado
*/
/*PTD * - * - * - *No usado
*/
/*PTE * - * - * - *No usado
*/
/*PTF * - * - * - *No usado
/*  *PTF0 * O * D * 1 *Columna 1 teclado
/*  *PTF1 * O * D * 1 *Columna 2 teclado
/*  *PTF2 * O * D * 1 *Columna 3 teclado
*/
/*PTG * - * - * - *Teclado
/*  *PTG0 * I * D * 1 *Fila 1 del teclado
/*  *PTG1 * I * D * 1 *Fila 2 del teclado
/*  *PTG2 * I * D * 1 *Fila 3 del teclado
/*  *PTG3 * I * D * 1 *Fila 4 del teclado
*/
/*PTH * - * - * - *No usado
*/
/*PTJ * - * - * - *No usado
*/
/* Archivos de cabecera */
#include <hidef.h>           //macro para interrupciones
#include "derivative.h"         //macro de declaraciones del MCU

/* Declaracion de variables y constantes */
byte bandera=0;                //bandera de proposito general
unsigned int retardo=0;          //parametro para hacer retardo
unsigned int i=0;                //variable de iteracion
char columna=0;                //columna detectada del teclado
char fila=0;                   //fila detectada del teclado
char comando8=0;                //parametro inicializacion LCD modo 8 bits
char dato4;                     //dato a enviar al LCD
char rs=0;                      //señal de dato o comando al LCD
char a=0;                       //variable temporal
char KBI1SC_Config=0;           //byte de inicializacion del registro KBI1SC
char KBI1PE_Config=0;           //byte de inicializacion del registro KBI1PE
char KBI1ES_Config=0;           //byte de inicializacion del registro KBI1ES
char PTAD_Config=0;              //byte de inicializacion del registro PTAD
char PTADD_Config=0;             //byte de inicializacion del registro PTADD
char PTAPE_Config=0;             //byte de inicializacion del registro PTAPE
char PTASE_Config=0;             //byte de inicializacion del registro PTASE
char PTADS_Config=0;             //byte de inicializacion del registro PTADS
char PTAIFE_Config=0;            //byte de inicializacion del registro PTAIFE
char PTFD_Config=0;              //byte de inicializacion del registro PTFD
char PTFDD_Config=0;             //byte de inicializacion del registro PTFDD
char PTFPE_Config=0;              //byte de inicializacion del registro PTFPE
char PTFSE_Config=0;              //byte de inicializacion del registro PTFSE
char PTFDS_Config=0;              //byte de inicializacion del registro PTFDS
char PTFIFE_Config=0;             //byte de inicializacion del registro PTFIFE_
const unsigned char mensaje[] = "****HOLA MUNDO***"; //Arreglo del mensaje para llevar al LCD
char *pM;                        //Puntero a mensaje para el LCD

/* Macros y definiciones */

```

```

#define Disable_COP() SOPT1 &= 0x3F //deshabilita el COP

/* Declaracion de funciones */
void KBI_Init(char KBI1SC_Config,char, KBI1PE_Config,char KBI1ES_Config);
                                         //declare funcion que inicializa KBI1
void PTA_Init(char PTAD_Config,char, PTADD_Config,char, PTAPE_Config,char, PTASE_Config,char,
PTADS_Config,char, PTAIFE_Config);          //declare funcion que inicializa PTA
void PTF_Init(char PTFD_Config, char PTFDD_Config,char, PTFPE_Config,char, PTFSE_Config,char,
PTFDS_Config,char, PTFIFE_Config);           //declare funcion que inicializa PTF
void Delay(unsigned int retardo);            //funcion para hacer reatrdos varios
void LCD_Init(void);                       //funcion para inicializar LCD
void Lcd_4bit_Wr(char dato4,char rs);       //funcion para escribir dato al LCD
void Comando_8bit(char comando8);           //funcion para inicio del LCD a 8 bits
void Rote_Col (void);                      //funcion para rotar un cero por columnas teclado
void Analisis_Tecla(void);                 //funcion para analisis de tecla presionada

/* main(): Funcion principal */
void main(void) {

    // Inicializacion del reloj:
    MCGTRM = NVMCGTRM + 52;                  //Tome valor programado de fabrica del TRIM
    MCGC1 = 0x04;                            //Modo FEI divisor por 1
    MCGC2 = 0x00;                            //Desactiva modo de reloj externo
    MCGC4 = 0x22;                            //Rango alto del DCO
                                              //El reloj MCGOUT queda en 24999945 Hz
    Disable_COP();                          //deshabilita el COP
    EnableInterrupts;                      //habilita interrupciones
    PTA_Init(0x00,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "0",pines como salida, no pullups, si slew, no strength y si filtro
    PTF_Init(0xFF,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "1",pines como salida, no pullups, si slew, no strength y si filtro
    PTGPE=0xFF;                             //habilite pullups puerto G
    KBI_Init(0x06,0xC3,0x00);               //da ACK, habilita interr, habilita pullups y flanco de bajada
    LCD_Init();                            //inicialice LCD
    pM = (char *)mensaje;                  //Puntero toma direccion del mensaje con definicion
    for (i=0;i<=15;i++){                  //Ciclo para imprimir 10 caracteres del mensaje
        Lcd_4bit_Wr(*pM,1);                //Envia caracter al LCD
        pM++;                                //Incrementa puntero al arreglo del mensaje
    }

    for(;;){                               //iteracion para lectura del teclado (for infinito)
        Rote_Col();
        if(bandera==1){
            Analisis_Tecla();
            bandera=0;
        }
    }
    /* es necesario asegurarse de nunca abandonar el main */
}

/* Funcion para inicializar el puerto A */
void PTA_Init(char PTAD_Config,char, PTADD_Config,char, PTAPE_Config,char, PTASE_Config,char,
PTADS_Config,char, PTAIFE_Config){
    PTAD=PTAD_Config;                     //inicialice estado pines PTA
    PTADD=PTADD_Config;                   //inicialice direccion de pines PTA
    PTAPE=PTAPE_Config;                  //inicialice estado de pullups PTA
    PTASE=PTASE_Config;                  //inicialice estado del slew rate PTA
    PTADS=PTADS_Config;                  //inicialice estado de drive strength PTA
    PTAIFE=PTAIFE_Config;                //inicialice estado del filtro PTA
}

/* Funcion para inicializar el puerto F */
void PTF_Init(char PTFD_Config,char, PTFDD_Config,char, PTFPE_Config,char, PTFSE_Config,char,
PTFDS_Config,char, PTFIFE_Config){
    PTFD=PTFD_Config;                    //inicialice estado pines PTF
    PTFDD=PTFDD_Config;                  //inicialice direccion de pines PTF
    PTFPE=PTFPE_Config;                  //inicialice estado de pullups PTF
    PTFSE=PTFSE_Config;                  //inicialice estado del slew rate PTF
}

```

```

PTFDS=PTFDS_Config;           //inicialice estado de drive strength PTF
PTFIFE=PTFIFE_Config;         //inicialice estado del filtro PTF
}

/* Funcion para inicializar el KBI */
void KBI_Init(char KBI1SC_Config,char, KBI1PE_Config,char, KBI1ES_Config){
    KBI1PE=KBI1PE_Config;      //inicialice registro de habilitacion de pines KBI
    KBI1SC_KBACK=1;            //aclara interrupciones espureas
    KBI1ES=KBI1ES_Config;      //inicialice registro de flanco y nivel del KBI
    KBI1SC=KBI1SC_Config;      //inicialice registro de estado y control del KBI
}

/* Funcion para inicializacion del LCD a 4 bits */
void LCD_Init(void){
    Delay(100000);             //hace retardo
    Comando_8bit(0x30);        //llama función enviar comando en 8 bits con comando
    Delay(100000);             //hace retardo
    Comando_8bit(0x30);        //llama función enviar comando en 8 bits con comando
    Delay(100000);             //hace retardo
    Comando_8bit(0x20);        //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x28,0);       //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x0C,0);       //ON LCD, OFF cursor
    Lcd_4bit_Wr(0x01,0);       //borrar pantalla LCD
    Lcd_4bit_Wr(0x06,0);       //desplaza cursor a la derecha con cada caracter
    Lcd_4bit_Wr(0x80,0);       //primer caracter fila 1 columna 1
}

/* Funcion para escribir a 4 bits en el LCD */
void Lcd_4bit_Wr(char dato4,char rs){
    if (rs==0){                //pregunte si se va a enviar un comando o un dato
        PTAD_PTAD5=0;          //prepara envío de comando RS = 0
    }
    else{
        PTAD_PTAD5=1;          //prepara envio de dato RS = 1
    }
    a=dato4;                   //almacena temporalmente el dato
    a=a>>4;                   //desplaza parte alta dato para la baja
    PTAD&=0xF0;                //enmascara parte alta del puerto A
    a&=0x0F;                   //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;              //sube pin de enable
    Delay(10000);              //hace retardo
    PTAD|=a;                   //envía nibble alto del dato
    Delay(10000);              //hace retardo
    PTAD_PTAD4=0;              //baja pin de enable
    Delay(10000);              //hace retardo
    PTAD&=0xF0;                //enmascara parte alta del puerto A
    dato4&=0x0F;                //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;              //sube pin de enable
    Delay(10000);              //hace retardo
    PTAD|=dato4;               //envía nibble bajo del dato
    Delay(10000);              //hace retardo
    PTAD_PTAD4=0;              //baja pin de enable
    Delay(10000);              //hace retardo
}

/* Función para envio de comando en 8 bits al LCD (solo para inicio) */
void Comando_8bit(char comando8){
    PTAD_PTAD5=0;              //prepara envio de comando
    PTAD&=0xF0;                //enmascara parte baja del puerto A
    comando8&=0xF0;             //enmascara parte alta de comando a enviar al LCD
    comando8=comando8>>4;      //desplaza para tomar nibble alto
    PTAD_PTAD4=1;              //sube pin de enable
    Delay(10000);              //hace retardo
    PTAD|=comando8;             //envía comando
    Delay(5000);                //hace retardo
    PTAD_PTAD4=0;              //sube pin de enable
    Delay(10000);              //hace retardo
}

```

```

/* Funcion Delay(): Retarda basado en una variable tipo entera */
void Delay(unsigned int retardo){
    while(retardo>0){           //llego a cero?
        retardo--;             //no --> decrementa
    }
}

/* Funcion para rotar cero por columnas teclado */
void Rote_Col (void){
    PTFD = 0x0E;                //Cero a la columna 1
    Delay(5000);               //hace retardo
    PTFD = 0x0D;                //Cero a la columna 2
    Delay(5000);               //hace retardo
    PTFD = 0x0B;                //Cero a la columna 3
    Delay(5000);               //hace retardo
}

/* Funcion para analizar tecla presionada y llevarla al LCD */
void Analisis_Tecla(void){
    Lcd_4bit_Wr(0xC0,0);        //primer caracter fila 2 columna 1
    if(columna==0x0E & fila==1){ //si se detecta columna 1 y fila 1
        Lcd_4bit_Wr(0x31,1);    //envie un 1 al LCD
    }
    if(columna==0x0D & fila==1){ //si se detecta columna 2 y fila 1
        Lcd_4bit_Wr(0x32,1);    //envie un 2 al LCD
    }
    if(columna==0x0B & fila==1){ //si se detecta columna 3 y fila 1
        Lcd_4bit_Wr(0x33,1);    //envie un 3 al LCD
    }
    if(columna==0x0E & fila==2){ //si se detecta columna 1 y fila 2
        Lcd_4bit_Wr(0x34,1);    //envie un 4 al LCD
    }
    if(columna==0x0D & fila==2){ //si se detecta columna 2 y fila 2
        Lcd_4bit_Wr(0x35,1);    //envie un 5 al LCD
    }
    if(columna==0x0B & fila==2){ //si se detecta columna 3 y fila 2
        Lcd_4bit_Wr(0x36,1);    //envie un 6 al LCD
    }
    if(columna==0x0E & fila==3){ //si se detecta columna 1 y fila 3
        Lcd_4bit_Wr(0x37,1);    //envie un 7 al LCD
    }
    if(columna==0x0D & fila==3){ //si se detecta columna 2 y fila 3
        Lcd_4bit_Wr(0x38,1);    //envie un 8 al LCD
    }
    if(columna==0x0B & fila==3){ //si se detecta columna 3 y fila 3
        Lcd_4bit_Wr(0x39,1);    //envie un 9 al LCD
    }
    if(columna==0x0D & fila==4){ //si se detecta columna 1 y fila 4
        Lcd_4bit_Wr(0x30,1);    //envie un 0 al LCD
    }
}

/* Funcion de atencion a la interrupcion del KBI1 */
interrupt VectorNumber_Vkeyboard void ISR_KBI (void){
    columna=PTFD & 0x0F;          //capture el valor de la columna
    if(PTGD_PTGD0==0){            //si detecto la fila 1
        fila=1;                  //marque fila 1
    }
    if(PTGD_PTGD1==0){            //si detecto la fila 2
        fila=2;                  //marque fila 2
    }
    if(PTGD_PTGD2==0){            //si detecto la fila 3
        fila=3;                  //marque fila 3
    }
    if(PTGD_PTGD3==0){            //si detecto la fila 4
        fila=4;                  //marque fila 4
    }
    KBI1SC_KBACK = 1;             //de reconocimiento de tecla presionada
}

```

```
bandera=1; //pone bandera de tecla presionada  
}
```

En el programa el LCD es controlado en modo de 4 bits, de tal manera que al escribir un dato es necesario enviar primero el nibble alto y luego el bajo.

La lectura de teclado se hace rotando un “0” por las tres columnas y mediante la detección de la interrupción de la función KBI1, se lee la fila. De esta forma se establece la tecla presionada, que luego será llevada al LCD.

En este ejercicio, es de especial interés analizar cómo se atiende una interrupción. Por ejemplo, la interrupción del módulo KBI fue atendida utilizando la definición planteada en el archivo *include*: MCF51JM128, que es incluido en todo proyecto nuevo y que el usuario luego nombrará como: **void nombre\_funcion (void)**.

```
/* Funcion de atencion a la interrupcion del KBI1 */  
interrupt VectorNumber_VKeyboard void ISR_KBI (void){  
    columna=PTFD & 0x0F; //capture el valor de la columna  
    if(PTGD_PTGD0==0){ //si detecto la fila 1  
        fila=1; //marque fila 1  
    }  
    if(PTGD_PTGD1==0){ //si detecto la fila 2  
        fila=2; //marque fila 2  
    }  
    if(PTGD_PTGD2==0){ //si detecto la fila 3  
        fila=3; //marque fila 3  
    }  
    if(PTGD_PTGD3==0){ //si detecto la fila 4  
        fila=4; //marque fila 4  
    }  
    KBI1SC_KBACK = 1; //de reconocimiento de tecla presionada  
    bandera=1; //pone bandera de tecla presionada  
}
```

## **12.4. REFERENCIAS**

- MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual. Freescale Semiconductor. Rev 1, 2008.
- Microcontroller MCF51JM128 ColdFire Data Sheet. Freescale Semiconductor. Rev 0, 2008.

## **12.5. PREGUNTAS**

- ¿Cuál es la convención para definir la dirección de un pin de un puerto?
- ¿Cuál es la capacidad maxima en corriente que se puede conducir por un pin de un puerto?
- ¿Qué diferencia existe entre la atención a un evento de interrupción por flanco o por flanco y nivel en el módulo KBI?
- ¿En qué configuración de dirección tiene sentido el parámetro de *slew rate*, para un pin I/O de la máquina?
- ¿Una vez se ha establecido la dirección de un pin de la máquina, será posible cambiarla más adelante en la ejecución del programa de usuario?
- Implementar en el DEMOJM un sistema para lectura de teclado matricial diferente al implementado en el aparte 12.3.

# CAPÍTULO 13

## Funciones de Temporización (TPM: *Timer/Pulse Modulator Module*)

**13.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**

**13.2. Ejercicios con el TPM:**

- Funcionamiento del TPM como temporizador de propósito general
- Funcionamiento del TPM como captura de evento de entrada (INPUT CAPTURE)
- Funcionamiento del TPM como PWM (*Pulse Width Modulation*)
- Funcionamiento del TPM como OUTPUT COMPARE

**13.3. El RTC (*Real Time Counter*)**

**13.4. Ejercicio con el RTC: Reloj digital inicializado por teclado**

**13.5. Referencias**

**13.6. Ejercicios y preguntas**

### 13.1. DIAGRAMA EN BLOQUES, EXPLICACIÓN DEL FUNCIONAMIENTO Y REGISTROS ASOCIADOS

El módulo de temporización (TPM) puede ser utilizado en cuatro modos de funcionamiento, que a continuación se definen.

- **Modo temporizador de propósito general:** Similar al comportamiento de un temporizador al trabajo, como se representa en la Figura 13.1.

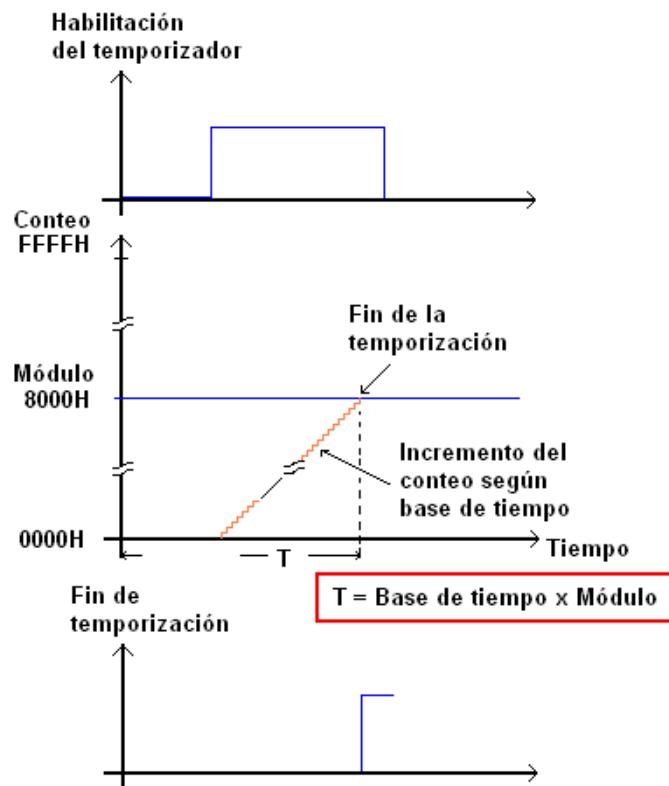


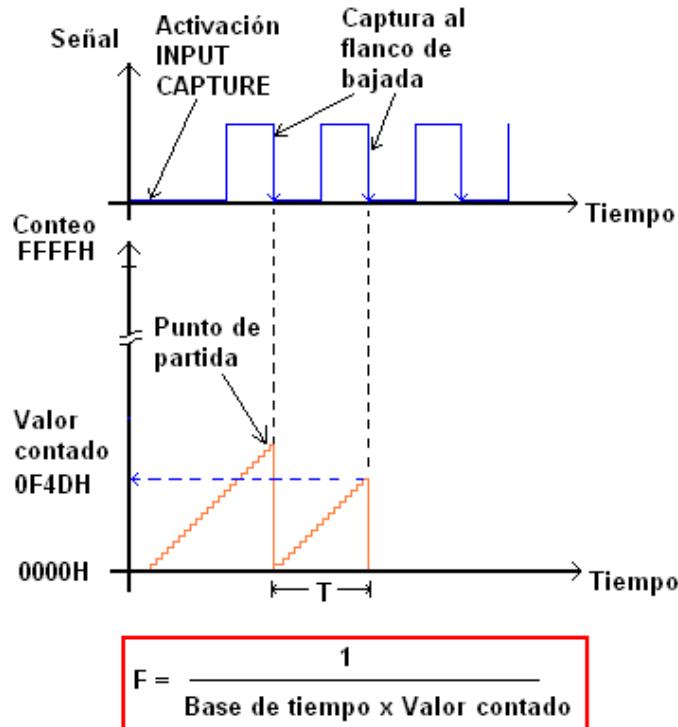
Figura 13.1. Temporización de propósito general.

La cantidad de conteos es programada en el módulo y que multiplicada por la base de tiempo, que alimenta al contador principal del sistema, da como resultado el tiempo programado. El fin de la temporización es anunciado por un evento tipo bit (como una bandera).

Este modo es utilizado cuando se requiere de temporizaciones precisas y los recursos de temporización interna, como el RTC, se encuentran ocupados.

- **Modo de captura de eventos externos (INPUT CAPTURE):** En este modo la máquina medirá eventos temporales externos, aplicados a pines de puertos. Estos eventos pueden ser: la medida del ancho de un pulso o la frecuencia de una señal.

La Figura 13.2 ilustra sobre la medida de una señal cuadrada, aplicada a un pin de la máquina.

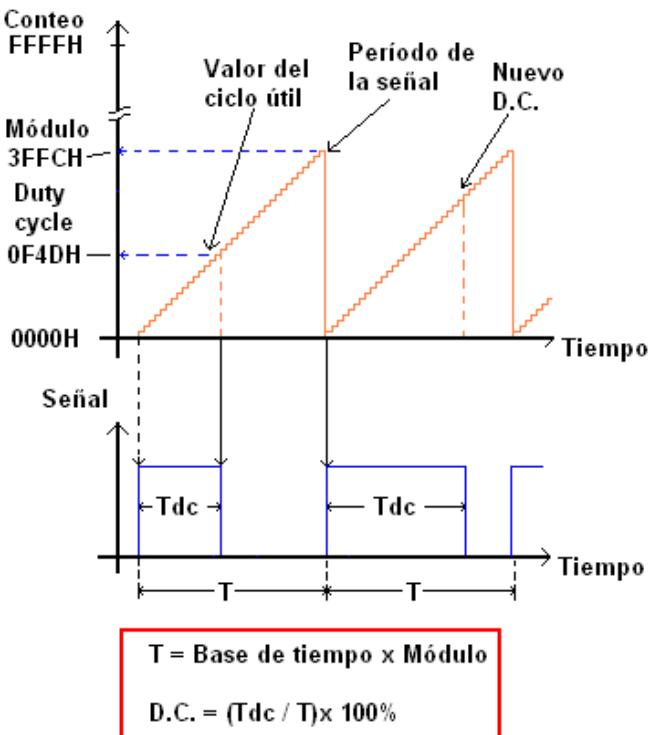


**Figura 13.2. Modo INPUT CAPTURE.**

Una vez se habilite el modo de captura de la **señal**, por flanco de bajada (**Activación INPUT CAPTURE**), el sistema queda a la espera del evento externo y el contador del módulo inicia el conteo. Al presentarse el primer **flanco de bajada** de la **señal** externa, se toma este evento como el **punto de partida** (contador en ceros) para la medida del período de la señal externa. El valor de frecuencia de la señal será el inverso del **valor contado** multiplicado por la **base de tiempo** del sistema, al presentarse el segundo evento de flanco.

- **Modo de generación de una señal periódica, como modulación del ancho del pulso (PWM):** En este modo la máquina puede generar una señal cuadrada, aplicada a un pin del MCU. A la señal generada se le puede controlar el ciclo útil (*Duty Cycle*), convirtiéndose en una señal PWM (*Pulse Width Modulation*).

La Figura 13.3 ilustra sobre la generación de una señal PWM, aplicada a un pin de la máquina.



**Figura 13.3. Modo OUTPUT COMPARE/PWM.**

El ciclo útil de la señal (*Duty Cycle*) es programado como un porcentaje entre el 0 y 100% del período de la señal. Generalmente el ciclo útil se controla desde un canal del módulo temporizador y deberá ser un valor menor que el período de la señal. El período de la señal es programado dentro del módulo del contador del temporizador.

- **Modo de generación de una señal periódica (OUTPUT COMPARE):** En este modo la máquina puede generar una señal cuadrada con un ciclo de trabajo del 50% (es un modo particular del PWM).

El módulo TPM, para la máquina ColdFire® MCF51JM128, contiene dos temporizadores (TPM1 y TPM2) con seis canales y dos canales respectivamente. Lo anterior indica que sería posible controlar dos eventos temporales con independencia periódica.

La Figura 13.4 muestra el diagrama en bloques del TPM1 de la máquina MCF51JM128, en donde se puede apreciar un contador principal (*16 bit counter*) y hasta 6 canales representados desde TPMC0 hasta TPMC5.

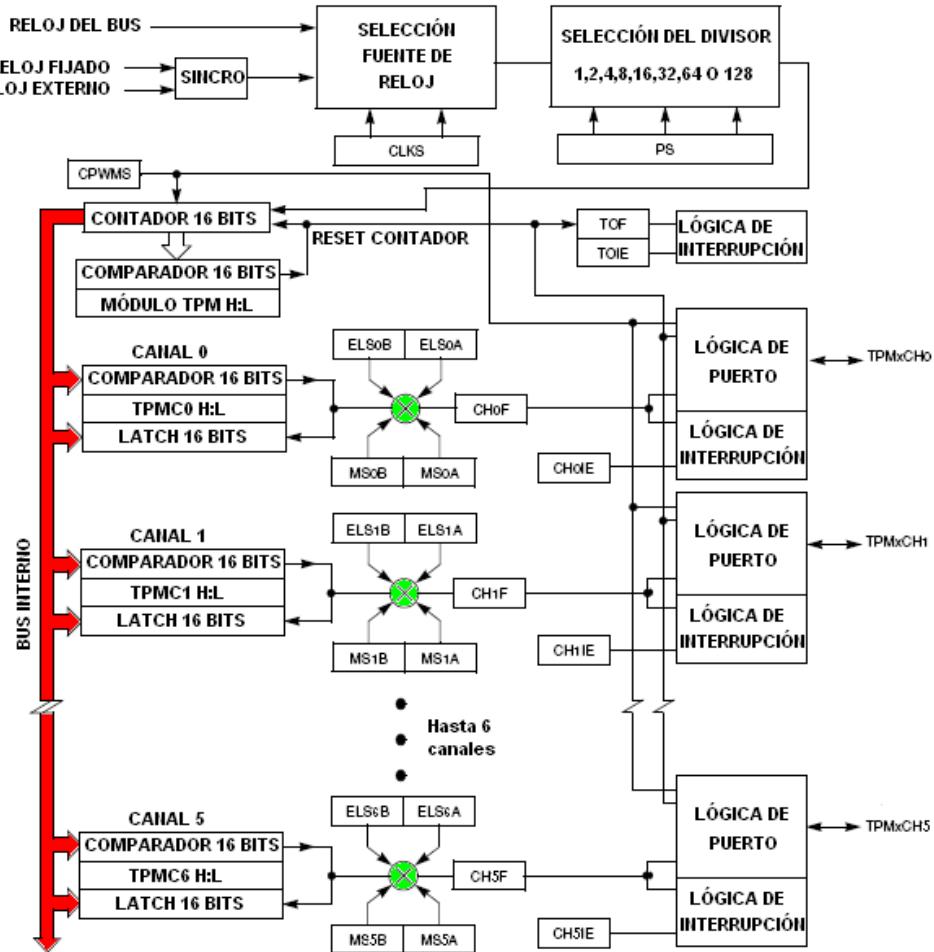


Figura 13.4. Módulo TPM.

- **Registros asociados al TPM:**
  - **Registro de estado y control (TPMxSC):** La Figura 13.5 muestra la configuración del registro de estado y control del TPM.

Registro TPMxSC: TPM1 = 0xFFFF8020 y TPM2 = 0xFFFF8060

	7	6	5	4	3	2	1	0
Lectura	TOF	TOIE	CPWMS	CLKS				
Escritura	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0

Figura 13.5. Registro estado y control del TPM.

- **TOF:** Bandera de sobreflujo del contador de 16 bits. Esta bandera se pone a “1” cuando se ha alcanzado el valor de 0x0000 una vez ha sido superado el valor programado en el registro de módulo del contador. Para aclarar el bit TOF es necesario leer el registro TPMSC y luego escribir un “0” en el bit TOF.
- 0:** El contador del TPM no ha alcanzado el sobreflujo
- 1:** El contador ha alcanzado un sobreflujo

- **TOIE:** Bit para habilitar un evento de interrupción. Cuando la bandera TOF es “1” y el bit TOIE=“1”, el sistema genera un evento de interrupción por sobreflujo del contador del TPM.  
**0:** Para detectar un evento de sobreflujo es necesario hacer *polling* sobre TOF  
**1:** Habilita un evento de interrupción cuando TOF = “1”
- **CPWMS:** Habilita que todos los canales del TPM actúen como PWM alineado en el centro del período (*center align*). El objetivo es disminuir ruido en las conmutaciones del pin de salida y el contador trabaja en modo *up/down*.  
**0:** No está activa la opción de alineado al centrado  
**1:** Activa opción de PWM alineado al centro
- **CLKS:** Selecciona la fuente de reloj del contador del TPM.  
**00:** Módulo inactivo  
**01:** Reloj del bus interno  
**10:** Reloj fijo del sistema (sólo para opción con circuito PLL)  
**11:** Reloj externo
- **PS:** Selección del divisor de la fuente de reloj.  
**000:** Divisor por 1  
**001:** Divisor por 2  
**010:** Divisor por 4  
**011:** Divisor por 8  
**100:** Divisor por 16  
**101:** Divisor por 32  
**110:** Divisor por 64  
**111:** Divisor por 128
- **Registro contador del TPM (TPMxCNTH:TPMxCNTL):** Está configurado por dos registros de 8 bits. La Figura 13.6 muestra los registros que conforman el contador de 16 bits del TPM. La acción de escribir en cualquiera de los dos registros, hace que se aclare el contador de 16 bits.

**Registros del contador TPM (TPMxCNTH:TPMxCNTL): TPM1 = 0xFFFF8021 : 0xFFFF8022  
TPM2 = 0xFFFF8061:0xFFFF8062**

Lectura	7	6	5	4	3	<i>&lt;</i>	<i>+</i>	<i>^</i>	Bit 8
Escritura	Bit 15	14	13	12	11	10	9		
Escribir algún valor en este registro, aclara el contador de 16 bits									
Reset	0	0	0	0	0	0	0	0	0
Lectura	7	6	5	4	3	2	1	0	Bit 0
Escritura	Bit 7	6	5	4	3	2	1	0	
Escribir algún valor en este registro, aclara el contador de 16 bits									
Reset	0	0	0	0	0	0	0	0	0

**Figura 13.6. Registro contador del TPM.**

- **Registro módulo del TPM (TPMxMODH:TPMxMODL):** Está configurado por dos registros de 8 bits. La Figura 13.7 muestra los registros que conforman el módulo de 16 bits del TPM. El módulo es el valor hasta donde debe contar el contador del TPM. Una vez que el contador haya alcanzado el valor programado en el módulo, este se fija en ceros (0x0000) y la bandera TOF se pone a “1”.

La escritura de un nuevo valor en el registro módulo obedece al valor de los bits CLKS del registro TPMSC y su comportamiento es:

CLKS = “00”: El módulo es actualizado cuando el segundo byte es escrito.

CLKS ≠ “00”: El módulo cambia cuando ambos registros hayan sido escritos y el contador pasa del valor TPMxMODH:TPMxMODL – 1, al valor TPMxMODH:TPMxMODL. Si el contador está en modo *free-running*, el módulo cambia cuando el contador pasa de 0xFFFF a 0xFFFF.

Registros de módulo (TPMxMODH:TPMxMODL)									TPM1 = 0xFFFF8023 : 0xFFFF8024	TPM2 = 0xFFFF8063 : 0xFFFF8064
Lectura Escritura	7	6	5	4	3	2	1	0		
Reset	Bit 15	14	13	12	11	10	9	Bit 8		
	0	0	0	0	0	0	0	0		
Lectura Escritura	7	6	5	4	3	2	1	0		
Reset	Bit 7	6	5	4	3	2	1	Bit 0		
	0	0	0	0	0	0	0	0		

Figura 13.7. Registro de módulo del TPM.

- **Registro de estado y control de canal (TPMxCnSC):** La Figura 13.8 muestra la configuración del registro de estado y control de los canales del TPM y la Figuras 13.9 ilustra la dirección asignada a cada uno.

#### Registros de estado y control de los canales del TPM (TPMxCnSC)

Lectura Escritura	7	6	5	4	3	2	1	0		
Reset	CHnF	CHnIE	MSnB	MSnA	ELSnB	ELSnA	0	0		
	0	0	0	0	0	0	0	0		
= Unimplemented or Reserved										

Figura 13.8. Registro de estado y control de canal.

0x(FF)FF_8025	TPM1C0SC	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	0	0
0x(FF)FF_8028	TPM1C1SC	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	0
0x(FF)FF_802B	TPM1C2SC	CH2F	CH2IE	MS2B	MS2A	ELS2B	ELS2A	0	0
0x(FF)FF_802E	TPM1C3SC	CH3F	CH3IE	MS3B	MS3A	ELS3B	ELS3A	0	0
0x(FF)FF_8031	TPM1C4SC	CH4F	CH4IE	MS4B	MS4A	ELS4B	ELS4A	0	0
0x(FF)FF_8034	TPM1C5SC	CH5F	CH5IE	MS5B	MS5A	ELS5B	ELS5A	0	0
0x(FF)FF_8065	TPM2C0SC	CH0F	CH0IE	MS0B	MS0A	ELS0B	ELS0A	0	0
0x(FF)FF_8068	TPM2C1SC	CH1F	CH1IE	MS1B	MS1A	ELS1B	ELS1A	0	0

Figura 13.9. Dirección de los registros de estado y control.

- **CHnF:** En el modo de INPUT CAPTURE, esta bandera se pone a “1” cuando el evento de flanco se presenta. En los modos OUTPUT COMPARE o PWM, esta bandera se pone a “1” cuando el valor del contador iguala al valor programado en el registro del canal. Para aclarar esta bandera durante un evento de interrupción, es necesario leer primero el estado del registro TPMxCnSC y luego escribir un cero en la bandera CHnF.

**0:** No ha ocurrido de INPUT CAPTURE, PWM u OUTPUT COMPARE.

**1:** Ha ocurrido un evento de INPUT CAPTURE, PWM u OUTPUT COMPARE en el canal

- **CHnIE:** Bit para habilitar un evento de interrupción. Cuando la bandera CHnF es “1” y el bit CHnIE=“1”, el sistema genera un evento de interrupción por canal.

**0:** Para detectar un evento de canal es necesario hacer *polling* sobre CHnF

**1:** Habilita un evento de interrupción cuando CHnF = “1”

**MSnB, MSnA, ELSnB y ELSnA:** Se utilizan para configuración del modo y el flanco de operación del TPM, según Tabla 13.1.

**1:** Habilita un evento de interrupción cuando CHnF = “1”

**Tabla 13.1. Modos de configuración del TPM**

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Modo	Configuración	
X	XX	00	Módulo TPM deshabilitado		
0	00	01	INPUT CAPTURE	Capture en el flanco de subida	
		10		Capture en el flanco de bajada	
		11		Capture en el flanco de subida o bajada	
	01	01	OUTPUT COMPARE	Cambie estado pin en OUTPUT COMPARE	
		10		Ponga pin en cero en OUTPUT COMPARE	
		11		Ponga pin en uno en OUTPUT COMPARE	
	1X	10	PWM ALINEADO AL FLANCO	Comienza en alto y cae en OUTPUT COMPARE	
		X1		Comienza en bajo y sube en OUTPUT COMPARE	
1	XX	10	PWM ALINEADO AL CENTRO	Comienza en alto y cae en OUTPUT COMPARE	
		X1		Comienza en bajo y sube en OUTPUT COMPARE	

- **Registro de valor del canal (TPMxCnVH:TPMxCnVL):** Está configurado por dos registros de 8 bits. La Figura 13.10 muestra los

registros que conforman el valor del canal y la Figura 13.11 ilustra las direcciones para dichos registros, según el TPM.

**Registro de valor de canal (TPMxCnVH:TPMxCnVL)**

Lectura Escritura	7	6	5	4	3	2	1	0
	Bit 15	14	13	12	11	10	9	Bit 8
Reset	0	0	0	0	0	0	0	0
Lectura Escritura	7	6	5	4	3	2	1	0
	Bit 7	6	5	4	3	2	1	Bit 0
Reset	0	0	0	0	0	0	0	0

**Figura 13.10. Registros de valor del canal.**

0x(FF)FF_8026	TPM1C0VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8027	TPM1C0VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8029	TPM1C1VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_802A	TPM1C1VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_802C	TPM1C2VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_802D	TPM1C2VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_802F	TPM1C3VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8030	TPM1C3VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8032	TPM1C4VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8033	TPM1C4VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8035	TPM1C5VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8036	TPM1C5VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8066	TPM2C0VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8067	TPM2C0VL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8069	TPM2C1VH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_806A	TPM2C1VL	Bit 7	6	5	4	3	2	1	Bit 0

**Figura 13.11. Dirección de los registros de valor del canal.**

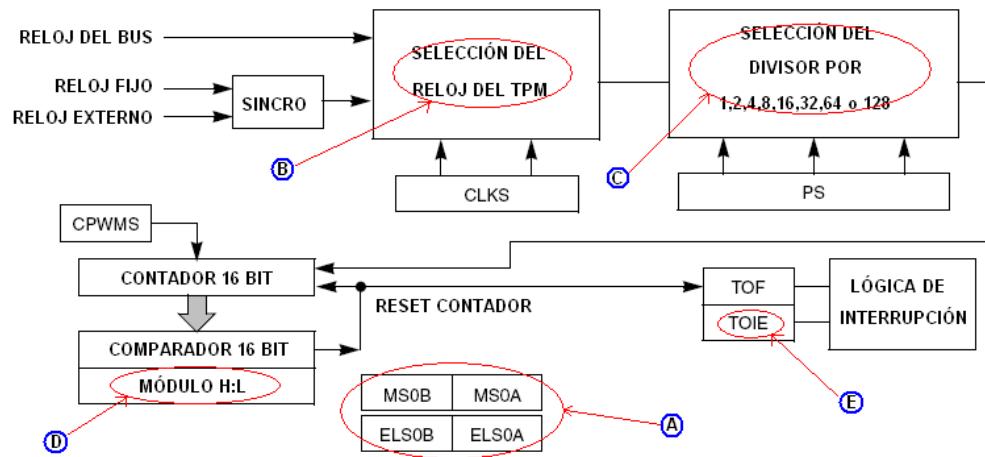
### 13.2. EJEMPLOS CON EL TPM

- **Funcionamiento del TPM como temporizador de propósito general:** Como se había explicado en aparte anterior, la temporización general es un modo de funcionamiento del TPM y no involucra a los canales ni a los pines del sistema.

Basta con habilitar el TPM en alguno de sus cuatro modos y utilizar la bandera de sobreflujo (TOF), para dar cuenta de una temporización lograda. La Figura 13.12 ilustra sobre las partes involucradas en una temporización de propósito general.

En el primer paso (A), es necesario elegir alguno de los modos de operación del TPM, ubicando un valor diferente a “00” en los bits ELS0A y ELS0B. A

continuación se realiza la selección de la fuente de reloj del TPM (**B**), teniendo en cuenta que la opción de **RELOJ FIJO** depende del circuito PLL y sus modos. Seguido (**C**), es necesario seleccionar un divisor del reloj del TPM, de acuerdo al fenómeno de temporización que se requiera. Lo anterior establece la base de tiempo del contador de 16 bits. Luego (**D**), se establece el módulo de conteo, que se calcula como: **M = Tiempo / Base de tiempo**. A continuación, se habilita el evento de interrupción (**E**) o si se prefiere se hace un *polling* sobre la bandera TOF. Finalmente, aclarar el contador escribiendo cualquier valor en el registro del contador (**F**).



**Figura 13.12. Temporización de propósito general.**

Un ejemplo para trabajar el TPM como un temporizador de propósito general es el siguiente:

```
/****************************************************************************
 * TPM_PROP_GRAL: Ejemplo sobre la utilización del modulo TPM como temporizador de propósito *
 * general. *
 * main.c *
 * *
 * Fecha: Mayo 15, 2008 *
 * *
 * V1.0, Rev 0 por Diego Munera *
 * *
 * Asunto: Implementar un código en C que encienda y apague un LED a una razón de 200ms. El re-
 * loj del sistema es FEI y el reloj de bus interno es la salida del FLL dividida por
 * dos. Se recomienda revisar el reloj que reporta el CodeWarrior en la simulación, pa-
 * ra el establecimiento de los divisores en el TPM.
 * *
 * Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros
 * editores.
 */

/*
 * DISTRIBUCION DE LOS PINES DE PUERTOS
 */

/*
 * PUERTO * PIN * I/O * TIPO * INICIA * COMENTARIO
 */
/*PTA * - * - * - * No usado */
*/
```

```

/*
*****PTB***** * - * - * - * - * - * No usado *****/
/*
*****PTC***** * - * - * - * - * - * No usado *****/
/*
*****PTD***** * - * - * - * - * - * Puerto del LED *****/
/* PTD0 * O * D * 1 * LED PTD2 DEMOJM */
/*
*****PTE***** * - * - * - * - * No usado *****/
/*
*****PTF***** * - * - * - * - * No usado *****/
/*
*****PTG***** * - * - * - * - * No usado *****/
/*
*****PTH***** * - * - * - * - * No usado *****/
/*
*****PTJ***** * - * - * - * - * No usado *****/
/*
*****Archivos de cabecera *****/
#include <hedef.h> //macro para interrupciones
#include "derivative.h" //macro de declaraciones del MCU

/* Declaracion de variables y constantes */
byte bandera=0; //bandera de propósito general
char PTDD_Config=0; //byte de inicialización del registro PTDD
char PTDDD_Config=0; //byte de inicialización del registro PTDDD
char PTDPE_Config=0; //byte de inicialización del registro PTDPE
char PTDSE_Config=0; //byte de inicialización del registro PTDSE
char PTDDS_Config=0; //byte de inicialización del registro PTDDS
char PTDIIFE_Config=0; //byte de inicialización del registro PTDIIFE
char TPM1SC_Config=0; //byte de inicialización del registro TPM1SC

/* Macros y definiciones */
#define Disable_COP() SOPT1 &= 0x3F //deshabilita el COP
#define Led1_On() PTDD_PTDD2 = 0; PTDDD_PTDDD2 = 1 //macro para encendido de LED
#define Led1_Off() PTDD_PTDD2 = 1; PTDDD_PTDDD2 = 0 //macro para apagado de LED

/* Declaracion de funciones */
void PTD_Init(char PTDD_Config, char PTDDD_Config, char PTDPE_Config, char PTDSE_Config, char PTDDS_Config, char PTDIIFE_Config); //declare función que inicializa PTD
void TPM1_Init(char TPM1SC_Config); //declaración de función que inicializa el registro TPM1SC

/* main(): Función principal */
void main(void) {
    Disable_COP(); //deshabilita el COP
    EnableInterrupts; //habilita interrupciones

    // Inicialización del reloj:
    MCGTRM = NVCMCGTRM + 52; //Tome valor programado de fábrica del TRIM
    MCGC1 = 0x04; //Modo FEI divisor por 1
    MCGC2 = 0x00; //Desactiva modo de reloj externo
    MCGC4 = 0x22; //Rango alto del DCO
    //El reloj MCGOUT queda en 24999945 Hz
    PTD_Init(0xFF,0xFF,0x00,0xFF,0x00,0xFF); //pines comienzan en "0", pines como salida, no pullups, si slew, no strength y si filtro
    TPM1_Init(0x4F); //habilita interrupción y divisor por 128
    //el CodeWarrior reporta un reloj FEI de bus = 24999945Hz
    //para una base de tiempo del TPM = 5.12 us
    TPM1MODH=0x98; //para el cálculo del módulo de conteo se procede así:
    TPM1MODL=0x96; //M = 200ms / 5.12us = 39062 = 0x9896

    for(;;) { //iteración para lectura del teclado (for infinito)
        do{
        }while (bandera==0);
        bandera=0; //aclara bandera
        Led1_On(); //enciende el LED
        do{
        }while (bandera==0);
    }
}

```

```

}while (bandera==0);
bandera=0;                                //aclare bandera
Led1_Off();                                 //apague el LED

}

/* es necesario asegurarse de nunca abandonar el main */
}

/* Funcion para inicializar el puerto D */
void PTDD_Init(char PTDD_Config, char PTDDD_Config, char PTDPE_Config, char PTDSE_Config, char PTDDS_Config, char PTDIFE_Config){
    PTDD=PTDD_Config;                      //inicialice estado pinos PTD
    PTDDD=PTDDD_Config;                   //inicialice direccion de pinos PTD
    PTDPE=PTDPE_Config;                  //inicialice estado de pullups PTD
    PTDSE=PTDSE_Config;                  //inicialice estado del slew rate PTD
    PTDDS=PTDDS_Config;                  //inicialice estado de drive strength PTD
    PTDIFE=PTDIFE_Config;                //inicialice estado del filtro PTD
}

/* Funcion para inicializar el TPM1 como temporizador de propósito general */
void TPM1_Init(char TPM1SC_Config){
    TPM1SC = TPM1SC_Config;           //carga valor de configuracion para registro TPM1SC
    TPM1CNTH=0;                      //aclare contador de 16 bits
}

/* Funcion de atencion a la interrupcion del TPM por sobreflujo */
interrupt VectorNumber_Vtpm1ovf void ISR TPM_OVF (void){
    TPM1SC;                           //lee registro de estado y control
    TPM1SC_TOF=0;                    //aclare bandera de sobreflujo
    bandera=1;                       //pone bandera
}

```

- Funcionamiento del TPM como captura de evento de entrada (INPUT CAPTURE):** La Figura 13.13 muestra las partes involucradas en un evento de captura por TPM.

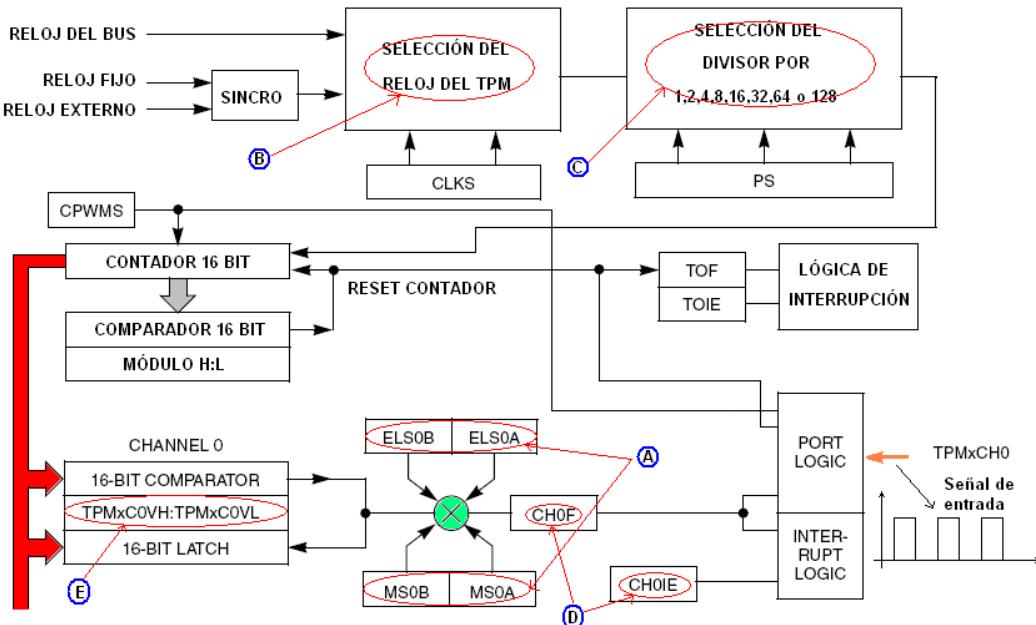


Figura 13.13. Captura de un evento externo.

Para la medida de una señal periódica aplicada al pin TPM1CH0, el primer paso es definir el modo de funcionamiento como captura de evento en el flanco de subida (**A**) (ver Tabla 13.1). A continuación se elige el reloj apropiado según el rango de frecuencias de la señal a medir (**B**). Seguido (**C**), es necesario especificar un divisor para el reloj elegido. En este momento se podría hacer un *polling* sobre la bandera CH0F (**D**) y de esta manera detectar un evento de flanco de subida en el pin TPM1CH0, pero si el propósito es generar un evento de interrupción se deberá poner a “1” el bit CH0IE (**D**). Finalmente, es necesario leer el valor capturado desde el contador de 16 bits y almacenado en el registro del canal TPM1C0VH:TPM1C0L (**E**). Este registro contiene el valor que alcanzó el contador de 16 bits, desde la habilitación del TPM hasta el evento de flanco de subida de la señal de entrada.

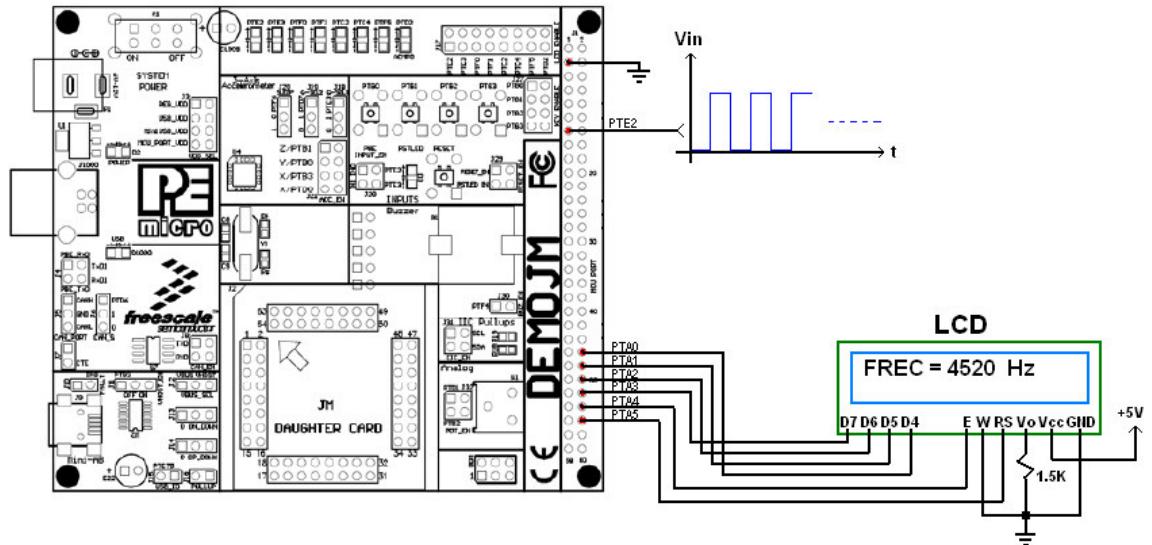
En términos de la medida de la frecuencia de la señal de entrada, el valor anterior carece de importancia y podría asignarse como el punto de partida, para medir el siguiente valor del contador debido al siguiente flanco. El usuario deberá aclarar el contador antes de iniciar la nueva medida, para que de esta manera se obtenga el valor del período (TPM1C0VH:TPM1C0L) de la señal de entrada.

La manera para calcular la frecuencia de la señal de entrada es:

$$\begin{aligned}\text{Frecuencia Base} &= \text{Frecuencia Reloj Elegido} / \text{Divisor Elegido} \\ \text{Frecuencia Señal} &= \text{Frecuencia Base} / \text{TPM1C0VH:TPM1C0L}\end{aligned}$$

El circuito de la Figura 13.14 muestra la implementación de la captura de una señal externa (Vin) con rango de frecuencia entre (1KHz y 10KHz), aplicada al pin PTE2 (TPM1CH0).

**NOTA:** Se recomienda al usuario revisar cuidadosamente el estado de los puentes del DEMOJM, con el objetivo de hacer compatible el ejercicio actual y de esta manera liberar los pines comprometidos en el circuito.



**Figura 13.14.** Circuito para ejemplo del INPUT CAPTURE.

**NOTA:** Para una correcta medida del frecuencímetro, El usuario deberá adecuar los conteos capturados en el canal, de acuerdo a la base de tiempo del contador de 16 bits y al reloj interno del sistema.

El siguiente listado ilustra una aplicación para el modo INPUT CAPTURE, como medidor de frecuencia de una señal de entrada en el rango de 1KHz a 10KHz:

```
/*
 * TPM_INPUT_CAPTURE: Ejemplo sobre la utilización del modulo TPM en el modo INPUT CAPTURE
 */
/*
 * main.c
 */
/*
 * Fecha: Mayo 15, 2008
 */
/*
 * V1.0, Rev 0 por Diego Múnera
 */
/*
 * Asunto: Implementar un código en C que mida la frecuencia de una señal cuadrada aplicada
 * al pin TPM1CH0. La frecuencia oscila en un rango de 1KHz a 10KHz y deberá ser presentada en un display LCD.
 */
/*
 * Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros
 * editores.
 */
*****
```

```
/*
 * DISTRIBUCION DE LOS PINES DE PUERTOS
 */
*****
```

```
/*
 * PUERTO * PIN * I/O * TIPO * INICIA * COMENTARIO
 */
*****
```

```
/*PTA * - * - * - * - *LCD
/* *PTA0 * O * D * 1 *D4 del LCD
/* *PTA1 * O * D * 1 *D5 del LCD
/* *PTA2 * O * D * 1 *D6 del LCD
/* *PTA3 * O * D * 1 *D7 del LCD
/* *PTA4 * O * D * 1 *Señal ENABLE del LCD
/* *PTA5 * O * D * 1 *Señal RS del LCD
*/
*****
```

```

/*PTB * - * - * - * *No usado */  

/*PTC * - * - * - * *No usado */  

/*PTD * - * - * - * *No usado */  

/*PTE * - * - * - * *No usado */  

/* PTE2 * I * D * - *Entrada de la señal de 1KHz-10KHz */  

/*PTF * - * - * - * *No usado */  

/*PTG * - * - * - * *Teclado */  

/*PTH * - * - * - * *No usado */  

/*PTJ * - * - * - * *No usado */  

/* Archivos de cabecera */  

#include <hidef.h> //macro para interrupciones  

#include "derivative.h" //macro de declaraciones del MCU  

/* Declaracion de variables y constantes */  

byte bandera=0; //bandera de propósito general  

unsigned long valor_canal_0=0;  

unsigned long retardo=0; //parametro para hacer retardo  

char i=0; //variable de iteracion  

byte miles_frec=0,centenas_frec=0,decenas_frec=0,unidades_frec=0;  

char PTAD_Config=0; //byte de inicializacion del registro PTAD  

char PTADD_Config=0; //byte de inicializacion del registro PTADD  

char PTAPE_Config=0; //byte de inicializacion del registro PTAPE  

char PTASE_Config=0; //byte de inicializacion del registro PTASE  

char PTADS_Config=0; //byte de inicializacion del registro PTADS  

char PTAIFE_Config=0; //byte de inicializacion del registro PTAIFE  

char comando8=0; //parametro inicializacion LCD modo 8 bits  

char dato4; //dato a enviar al LCD  

char rs=0; //señal de dato o comando al LCD  

char a=0; //variable temporal  

const unsigned char mensaje[] = "FREC = Hz "; //Arreglo del mensaje para llevar al LCD  

char *pM; //Puntero a mensaje para el LCD  

char TPM1SC_Config=0; //byte de inicializacion del registro TPM1SC  

char TPM1C0SC_Config=0; //byte de inicializacion del registro TPM1C0SC  

/* Macros y definiciones */  

#define Disable_COP() SOPT1 &= 0x3F //deshabilita el COP  

/* Declaracion de funciones */  

void Delay(unsigned long retardo); //funcion para hacer reatrdos varios  

void LCD_Init(void); //funcion para inicializar LCD  

void Lcd_4bit_Wr(char dato4,char rs); //funcion para escribir dato al LCD  

void Comando_8bit(char comando8); //funcion para inicio del LCD a 8 bits  

void TPM1_Init(char TPMISC_Config, char TPM1C0SC_Config); //funcion para inicializar el TPM  

void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config); //declare funcion que inicializa PTA  

void Convertir_Vvalor(unsigned long valor_canal_0);  

/* main(): Funcion principal */  

void main(void) {  

    Disable_COP(); //deshabilita el COP  

    // Inicializacion del reloj:  

    MCGTRM = NVMMCGRM + 52; //Tome valor programado de fabrica del TRIM  

    MCGC1 = 0x04; //Modo FEI divisor por 1  

    MCGC2 = 0x00; //Desactiva modo de reloj externo  

    MCGC4 = 0x22; //Rango alto del DCO  

    //El reloj MCGOUT queda en 24999945 Hz  

    PTA_Init(0x00,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "0",pines como salida, no pullups, si slew, no strength y si filtro

```

```

LCD_Init();
TPM1_Init(0x08,0x44);
pM = (char *)mensaje;
for (i=0;i<=14;i++){
    Lcd_4bit_Wr(*pM,1);
    pM++;
}

EnableInterrupts; //habilita interrupciones

for(;;) { //iteracion para lectura del teclado (for infinito)

    do{ //espere primera interrupcion del TPM1 como INPUT CAPTURE
        //mientras no se haya dado la interrupcion
    }while(bandera!=1);
    bandera=0;
    TPM1CNTH=0;

    do{ //espere segunda interrupcion del TPM1 como INPUT CAPTURE
        //mientras no se haya dado la interrupcion
    }while(bandera!=1);
    bandera=0;
    Convertir_Valor(valor_canal_0);
    Lcd_4bit_Wr(0x87,0); //llame funcion para convertir valor de frecuencia del canal 0
    Lcd_4bit_Wr(miles_frec,1); //actualice valor de miles en el LCD
    Lcd_4bit_Wr(0x88,0); //miles a la fila fila 1 columna 8 del LCD
    Lcd_4bit_Wr(cientos_frec,1); //actualice valor de centenas en el LCD
    Lcd_4bit_Wr(0x89,0); //miles a la fila fila 1 columna 9 del LCD
    Lcd_4bit_Wr(decenas_frec,1); //actualice valor de decenas en el LCD
    Lcd_4bit_Wr(0x8A,0); //miles a la fila fila 1 columna 10 del LCD
    Lcd_4bit_Wr(unidades_frec,1); //actualice valor de unidades en el LCD
    bandera=0; //aclare bandera
    TPM1CNTH=0; //aclare contador de 16 bits
}
/* es necesario asegurarse de nunca abandonar el main */

/* Funcion para inicializar el puerto A */
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config){
    PTAD=PTAD_Config; //inicialice estado pinos PTA
    PTADD=PTADD_Config; //inicialice direccion de pinos PTA
    PTAPE=PTAPE_Config; //inicialice estado de pullups PTA
    PTASE=PTASE_Config; //inicialice estado del slew rate PTA
    PTADS=PTADS_Config; //inicialice estado de drive strength PTA
    PTAIFE=PTAIFE_Config; //inicialice estado del filtro PTA
}

/* Funcion para inicializacion del LCD a 4 bits */
void LCD_Init(void){
    Delay(200000); //hace retardo
    Comando_8bit(0x30); //llama funcion enviar comando en 8 bits con comando
    Delay(200000); //hace retardo
    Comando_8bit(0x30); //llama funcion enviar comando en 8 bits con comando
    Delay(200000); //hace retardo
    Comando_8bit(0x20); //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x28,0); //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x0C,0); //ON LCD, OFF cursor
    Lcd_4bit_Wr(0x01,0); //borrar pantalla LCD
    Lcd_4bit_Wr(0x06,0); //desplaza cursor a la derecha con cada caracter
    Lcd_4bit_Wr(0x80,0); //primer caracter fila 1 columna 1
}

/* Funcion para escribir a 4 bits en el LCD */
void Lcd_4bit_Wr(char dato4,char rs){
    if (rs==0){ //pregunte si se va a enviar un comando o un dato
        PTAD_PTAD5=0; //prepara envio de comando RS = 0
    }
    else{
        PTAD_PTAD5=1; //prepara envio de dato RS = 1
    }
}

```

```

    }
    a=dato4;                                //almacena temporalmente el dato
    a=a>>4;                                //desplaza parte alta dato para la baja
    PTAD&=0xF0;                             //enmascara parte alta del puerto A
    a&=0x0F;                                //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;                            //sube pin de enable
    Delay(20000);                           //hace retardo
    PTAD|=a;                                 //envia nibble alto del dato
    Delay(20000);                           //hace retardo
    PTAD_PTAD4=0;                            //baja pin de enable
    Delay(20000);                           //hace retardo
    PTAD&=0xF0;                             //enmascara parte alta del puerto A
    dato4&=0x0F;                            //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;                            //sube pin de enable
    Delay(20000);                           //hace retardo
    PTAD|=dato4;                            //envia nibble bajo del dato
    Delay(20000);                           //hace retardo
    PTAD_PTAD4=0;                            //baja pin de enable
    Delay(20000);                           //hace retardo
}

/* Función para envio de comando en 8 bits al LCD (solo para inicio) */
void Comando_8bit(char comando8){
    PTAD_PTAD5=0;                          //prepara envio de comando
    PTAD&=0xF0;                            //enmascara parte baja del puerto A
    comando8&=0xF0;                          //enmascara parte alta de comando a enviar al LCD
    comando8=comando8>>4;                //desplaza para tomar nibble alto
    PTAD_PTAD4=1;                            //sube pin de enable
    Delay(20000);                           //hace retardo
    PTAD|=comando8;                         //envia comando
    Delay(20000);                           //hace retardo
    PTAD_PTAD4=0;                            //sube pin de enable
    Delay(20000);                           //hace retardo
}

/* Funcion para inicializar el TPM1 en modo INPUT CAPTURE */
void TPM1_Init(char TPM1SC_Config, char TPM1C0SC_Config){
    TPM1SC=TPM1SC_Config;                  //carga valor de configuracion para registro TPM1SC
    TPM1C0SC=TPM1C0SC_Config;              //carga valor de configuracion para registro TPM1C0SC
    TPM1CNTH=0;                            //aclare contador de 16 bits
}

/* Funcion Delay(): Retarda basado en una variable tipo entero */
void Delay(unsigned long retardo){
    while(retardo>0){                     //llego a cero?
        retardo--;                         //no --> decrementa
    }
}

/* Funcion para convertir valor del canal 0 del TPM a miles, centenas, decenas y unidades */
//el valor del canal esta escalado por un factor de 119, para evitar dividir y generar un
//número fraccionario, lo que implicaría hacer un retardo enorme para esta maquina
void Convertir_Valor(unsigned long valor_canal_0){
    i=0;                                  //extrae los miles contenidos en valor del canal
    do{
        if(valor_canal_0>=0){
            valor_canal_0=valor_canal_0-100000;
            ++i;
        }
    }

    }while (valor_canal_0>=100000);
    miles_frec= i+0x30;                   //extrae los cientos contenidos en valor del canal
    do{
        if(valor_canal_0>=0){
            valor_canal_0=valor_canal_0-10000;
            ++i;
        }
    }
}

```

```

}while (valor_canal_0>=10000);
centenas_frec= i+0x30;
i=0;
do{                                //extrae las decenas contenidos en valor del canal
    if(valor_canal_0>=0){
        valor_canal_0=valor_canal_0-1000;
        ++i;
    }
}while (valor_canal_0>=1000);
decenas_frec= i+0x30;
i=0;
do{                                //extrae las unidades contenidos en valor del canal
    if(valor_canal_0>=0){
        valor_canal_0=valor_canal_0-100;
        ++i;
    }
}while (valor_canal_0>=100);
unidades_frec= i+0x30;
}

/* Funcion de atencion a la interrupcion del TPM1 canal 0 */
interrupt VectorNumber_Vtpm1ch0 void ISR_TPM1_CH0 (void){
    TPM1C0SC;                      //preparese para borrar bandera del canal
    TPM1C0SC_CH0F=0;                //aclare bandera del canal
    valor_canal_0=TPM1C0V;          //tome valor del canal
    valor_canal_0=valor_canal_0 * 119; //se escala para ajustar valor a reloj interno del
                                       //micro y a base de tiempo del TPM1 como INPUT CAPTURE
    bandera=1;                      //ponga bandera en "1"
}

```

- **Funcionamiento del TPM como PWM (Pulse Width Modulation):** La Figura 13.15 muestra las partes involucradas en un evento de salida de PWM.

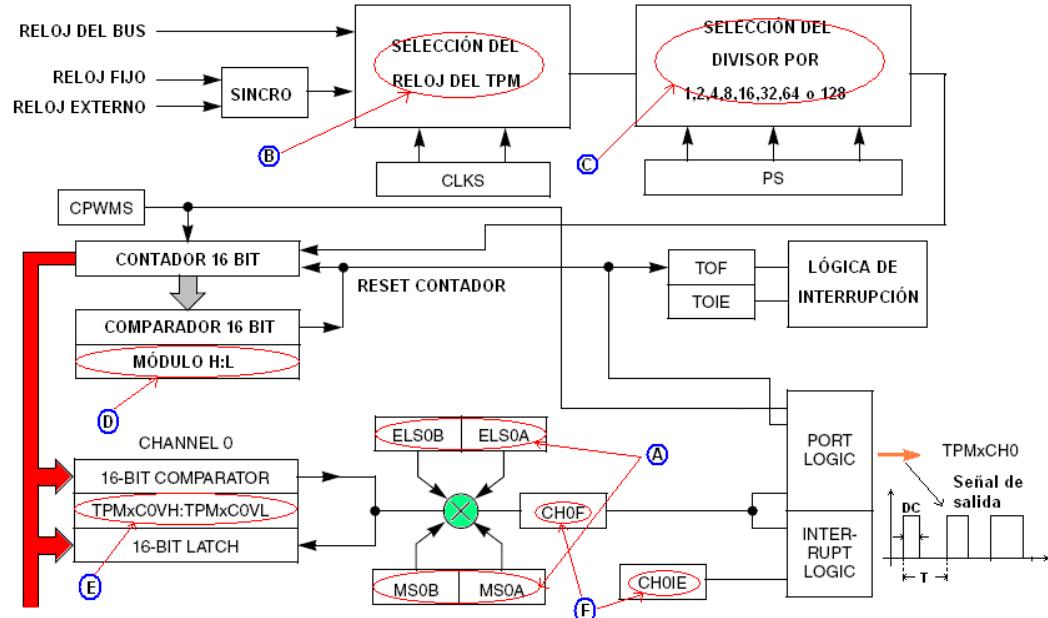


Figura 13.15. Generación de señal PWM.

Para la generación de una señal periódica tipo PWM presentada en algún pin TPMxCHn, el primer paso es definir el modo de funcionamiento como PWM en los bits marcados como (A) (ver Tabla 13.1). El modo PWM tiene la opción de trabajar iniciando con un flanco de subida o terminando con un flanco de bajada, desde el punto de vista del ciclo de trabajo (DC: *Duty Cycle*); o como alineado al centro del período T. Esta última característica es muy popular para el control de servo motores de 3 fases en CA y sin escobillas (*brushless*) en CD, en donde son necesarios varios canales de PWM. A continuación se elige el reloj apropiado según el rango de frecuencias de la señal a generar (B). Segundo (C), es necesario especificar un divisor para el reloj elegido. Luego se programa el período (T) de la señal en el módulo del contador de 16 bits (D) y el ciclo de trabajo (DC) de la señal en el registro del canal (E).

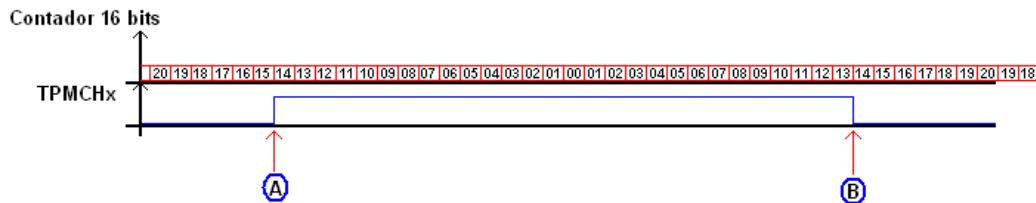
Cada que sea necesario actualizar el valor del ciclo de trabajo, el usuario deberá hacer un *polling* o generar un evento de interrupción, utilizando la bandera CHxF y/o habilitando el mecanismo de interrupción con el bit CHxIE (F).

Si la opción elegida es la de atender un evento de interrupción por canal (OUTPUT COMPARE), se recomienda actualizar los valores del nuevo DC en la atención a dicha interrupción.

Para la opción de PWM alineado al centro, el contador de 16 bits trabaja en modo up/down. En la Figura 15.16 se ilustra de manera temporal la generación de una señal PWM alineada al centro y pulso a alto, con una programación de 20 conteos en el módulo (T) y un ciclo de trabajo (DC) de 14 conteos. El contador inicia en un conteo descendente y el pin de salida se mantiene en bajo, cuando el contador llega al valor programado en el canal el pin sube (A). El contador se decrementa hasta adquirir un conteo ascendente y cuando llega al valor de conteo programado en el canal, el pin de salida vuelve a caer (B). Este esquema se repite hasta tanto no se detenga el PWM o se cambien los valores de su período y ciclo de trabajo.

**T = Módulo = 20**

**DC = Canal = 14**



**Figura 13.16. PWM alineado al centro y pulso a alto.**

El período (T) de la señal de salida del PWM se calcula así:

$$\text{Base de tiempo} = 1 / (\text{Frecuencia Reloj Elegido} / \text{Divisor})$$

$$T = \text{Base de tiempo} \times \text{TPMxMODH:TPMxMODL}$$

$$DC = \text{Base de Tiempo} \times \text{TPMxCnVH:TPMxCnVL}$$

**NOTA:** Se recomienda al usuario revisar cuidadosamente el estado de los puentes del DEMOJM, con el objetivo de hacer compatible el ejercicio actual y de esta manera liberar los pines comprometidos en el circuito.

El circuito de la Figura 13.17 muestra un sistema para controlar la energía aplicada a la resistencia calentadora de un horno, en donde la energía será programada desde un 10% hasta un 98%. Utilizando un TRIAC y una rectificación de onda completa, como elementos actuadores sobre la carga (*Dimmer*).

El sistema regula la cantidad de ciclos completos de la señal de CA, entregados a la carga, que luego serán rectificados y filtrados.

Para incrementar o decrementar la energía entregada a la carga, se modifica el valor del CD mediante la pulsación de PB1 y PB2. En un LCD se presentará el valor porcentual del DC (Duty Cycle) aplicado a la carga.

El siguiente listado ilustra un programa en C, que resuelve el problema propuesto.

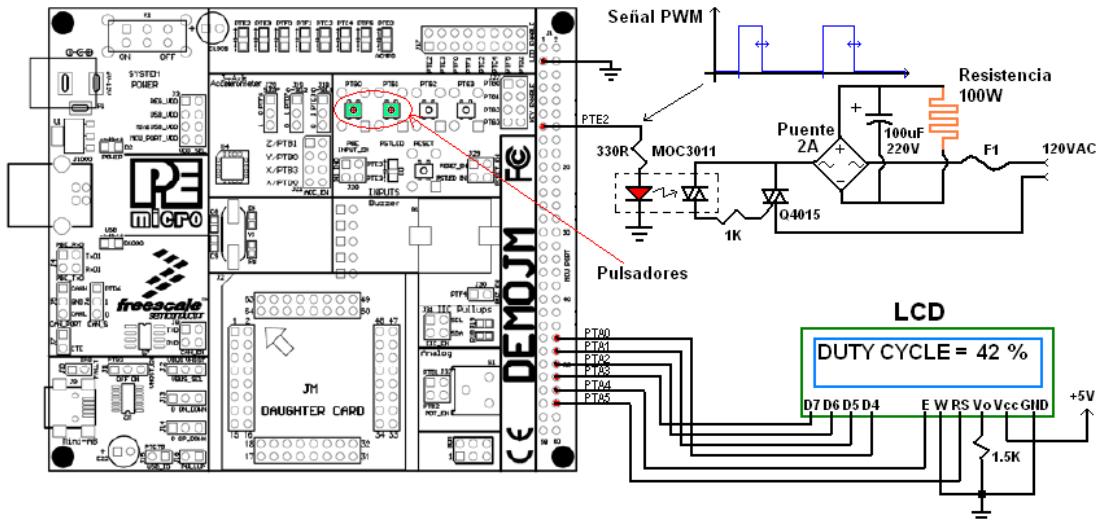


Figura 13.17. Circuito PWM para control de intensidad en bombilla.

```

/*
 * TPM_PWM: Ejemplo sobre la utilizacion del modulo TPM en el modo PWM EDGE ALIGN
 */
/* main.c
*/
/*
 * Fecha: Mayo 18, 2008
*/
/*
 * V1.0, Rev 0 por Diego Munera
*/
/*
 * Asunto: Implementar un codigo en C que controle la energia aplicada a una resistencia calentadora
 * para horno, mediante el control de los ciclos aplicados a la carga (Dimmer). El circuito se controla
 * utilizando un TPM en modo PWM. El valor minimo sera del 10% y el valor maximo sera del 98%.
*/
/*
 * Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros
 * editores.
*/
/*
 * DISTRIBUCION DE LOS PINES DE PUERTOS
*/
/*
 * PUERTO   * PIN   * I/O   * TIPO   * INICIA   *      COMENTARIO
*/
/*
 *PTA      * -     * -     * -     * -     *LCD
/*      *PTA0    * O     * D     * 1     *D4 del LCD
/*      *PTA1    * O     * D     * 1     *D5 del LCD
/*      *PTA2    * O     * D     * 1     *D6 del LCD
/*      *PTA3    * O     * D     * 1     *D7 del LCD
/*      *PTA4    * O     * D     * 1     *Señal ENABLE del LCD
/*      *PTA5    * O     * D     * 1     *Señal RS del LCD
/*
 *PTB      * -     * -     * -     * -     *No usado
/*
 *PTC      * -     * -     * -     * -     *No usado
/*
 *PTD      * -     * -     * -     * -     *No usado
/*
 *PTE      * -     * -     * -     * -     *TPM1 como PWM
/*      *PTE2    * O     * D     * -     *Salida PWM
/*
 *PTF      * -     * -     * -     * -     *No usado
/*
 *PTG      * -     * -     * -     * -     *Pulsadores
/*      *PTG0    * I     * D     * 1     *Pulsador incremento energia bombilla
/*      *PTG1    * I     * D     * 1     *Pulsador decremento energia bombilla
/*
 *PTH      * -     * -     * -     * -     *No usado
/*
 *PTJ      * -     * -     * -     * -     *No usado
/*
 * Archivos de cabecera */
#include <hidef.h>           //macro para interrupciones
#include "derivative.h"         //macro de declaraciones del MCU

/* Declaracion de variables y constantes */
byte bandera=0;                //bandera de proposito general
unsigned long retardo=0;        //parametro para hacer retardo
char i=0;                      //variable de iteracion
byte decenas_DC=0,unidades_DC=0; //variables para presentacion del DC en el LCD
char PTAD_Config=0;             //byte de inicializacion del registro PTAD
char PTADD_Config=0;            //byte de inicializacion del registro PTADD
char PTAPE_Config=0;            //byte de inicializacion del registro PTAPE
char PTASE_Config=0;            //byte de inicializacion del registro PTASE
char PTADS_Config=0;            //byte de inicializacion del registro PTADS
char PTAIFE_Config=0;           //byte de inicializacion del registro PTAIFE
char PTGD_Config=0;              //byte de inicializacion del registro PTGD
char PTGDD_Config=0;             //byte de inicializacion del registro PTGDD

```

```

char PTGPE_Config=0;           //byte de inicializacion del registro PTGPE
char PTGSE_Config=0;           //byte de inicializacion del registro PTGSE
char PTGDS_Config=0;           //byte de inicializacion del registro PTGDS
char PTGIFE_Config=0;          //byte de inicializacion del registro PTGIFE
char comando8=0;               //parametro inicializacion LCD modo 8 bits
char dato4;                   //dato a enviar al LCD
char rs=0;                     //señal de dato o comando al LCD
char a=0;                      //variable temporal
const unsigned char mensaje[] = "DUTY CYCLE= %"; //Arreglo del mensaje para llevar al LCD
char *pM;                      //puntero a mensaje para el LCD
char TPM1SC_Config=0;          //byte de inicializacion del registro TPM1SC
char TPM1C0SC_Config=0;         //byte de inicializacion del registro TPM1C0SC
unsigned int new_DC=3250;       //variable para control del Duty Cycle
unsigned long tempo=0;          //variable de proposito general

/* Macros y definiciones */
#define Disable_COP() SOPT1 &= 0x3F //deshabilita el COP
#define INPUT_1() !(PTGD_PTGD0) //macro para lectura de tecla de incrementar DC presionada
#define INPUT_2() !(PTGD_PTGD1) //macro para lectura de tecla de decrementar DC presionada

/* Declaracion de funciones */
void Delay(unsigned long retardo); //funcion para hacer reatrdos varios
void LCD_Init(void);              //funcion para inicializar LCD
void Lcd_4bit_Wr(char dato4,char rs); //funcion para escribir dato al LCD
void Comando_8bit(char comando8); //funcion para inicio del LCD a 8 bits
void TPM1_Init(char TPM1SC_Config, char TPM1C0SC_Config); //funcion para inicializar el TPM
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config); //declare funcion que inicializa PTA
void PTG_Init(char PTGD_Config,char PTGDD_Config,char PTGPE_Config,char PTGSE_Config,char PTGDS_Config,char PTGIFE_Config); //declare funcion que inicializa PTG
void Convertir_Value(unsigned long tempo); //funcion para convertir valor del DC en unidades y decenas, para ser presentadas en el LCD

/* main(): Funcion principal */
void main(void) {
    Disable_COP(); //deshabilita el COP

    // Inicializacion del reloj:
    MCGTRM = NVMCGTRM + 52; //Tome valor programado de fabrica del TRIM
    MCGC1 = 0x04; //Modo FEI divisor por 1
    MCGC2 = 0x00; //Desactiva modo de reloj externo
    MCGC4 = 0x22; //Rango alto del DCO
    //El reloj MCGOUT queda en 24999945 Hz
    PTA_Init(0x00,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "0",pines como salida, no pullups, si slew, no strength y si filtro
    PTG_Init(0x00,0x00,0xFF,0x00,0x00,0xFF); //pines inician en "0",pines como entrada, pullups, no slew, no strength y si filtro
    LCD_Init(); //inicialice LCD
    TPM1_Init(0x0C,0x78); //PWM EDGE ALIGN, reloj bus interno, divisor por 16, interr por canal
    //para una base de tiempo del contador del TPM de 0.24useg
    pM = (char *)mensaje; //Puntero toma direccion del mensaje con definición
    for (i=0;i<=15;i++){ //Ciclo para imprimir 16 caracteres del mensaje
        Lcd_4bit_Wr(*pM,1); //Envía caracter al LCD
        pM++; //Incrementa puntero al arreglo del mensaje
    }
    EnableInterrupts; //habilita interrupciones
    for(;;) { //iteracion para lectura del teclado (for infinito)

        //Procedimiento para leer los pulsadores que incrementan o decrementan el DC del PWM. Para la base de tiempo que alimenta el contador de 16 bits del TPM1, el período de la señal PWM
        //se establece en 65000 x 0,24useg = 15.6 mseg
        while(INPUT_1()){ //mientras se encuentre presionado el pulsador de incrementar
            new_DC++; //el DC
            if (new_DC > 63700){ //incremente el DC
                new_DC = 63700; //si el DC llego al 98%
            }
            Delay(500); //pequeño retardo para pulsado de INPUT_1
        };
    }
}

```

```

while(INPUT_20){
    new_DC--;
    if (new_DC < 6500){
        new_DC = 6500;
    }
    Delay(500);
};

tempo=(new_DC*100/65000); //valor del DC para visualizacion en el LCD, este valor
//es redondeado debido a que el MCU no tiene unidad de
//punto flotante. El valor se calcula tomando como base
//65000 como valor del 100% del modulo del TPM1, que no
//mas que el periodo de la señal
Convertir_Valor(tempo); //convertir el valor a unidades y decenas para llevar
    /al LCD
Lcd_4bit_Wr(0x8C,0); //decenas a la fila fila 1 columna 8 del LCD
Lcd_4bit_Wr(decenas_DC,1); //actualice valor de decenas en el LCD
Lcd_4bit_Wr(0x8D,0); //unidades a la fila fila 1 columna 9 del LCD
Lcd_4bit_Wr(unidades_DC,1); //actualice valor de unidades en el LCD

}

/*
/* es necesario asegurarse de nunca abandonar el main */

/* Funcion para inicializar el puerto A */
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config){
    PTAD=PTAD_Config; //inicialice estado pinos PTA
    PTADD=PTADD_Config; //inicialice dirección de pinos PTA
    PTAPE=PTAPE_Config; //inicialice estado de pullups PTA
    PTASE=PTASE_Config; //inicialice estado del slew rate PTA
    PTADS=PTADS_Config; //inicialice estado de drive strength PTA
    PTAIFE=PTAIFE_Config; //inicialice estado del filtro PTA
}

/* Funcion para inicializar el puerto G */
void PTG_Init(char PTGD_Config,char PTGDD_Config,char PTGPE_Config,char PTGSE_Config,char PTGDS_Config,char PTGIFE_Config){
    PTGD=PTGD_Config; //inicialice estado pinos PTG
    PTGDD=PTGDD_Config; //inicialice dirección de pinos PTG
    PTGPE=PTGPE_Config; //inicialice estado de pullups PTG
    PTGSE=PTGSE_Config; //inicialice estado del slew rate PTG
    PTGDS=PTGDS_Config; //inicialice estado de drive strength PTG
    PTGIFE=PTGIFE_Config; //inicialice estado del filtro PTG
}

/* Funcion para inicializacion del LCD a 4 bits */
void LCD_Init(void){
    Delay(200000); //hace retardo
    Comando_8bit(0x30); //llama función enviar comando en 8 bits con comando
    Delay(200000); //hace retardo
    Comando_8bit(0x30); //llama función enviar comando en 8 bits con comando
    Delay(200000); //hace retardo
    Comando_8bit(0x20); //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x28,0); //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x0C,0); //ON LCD, OFF cursor
    Lcd_4bit_Wr(0x01,0); //borrar pantalla LCD
    Lcd_4bit_Wr(0x06,0); //desplaza cursor a la derecha con cada caracter
    Lcd_4bit_Wr(0x80,0); //primer caracter fila 1 columna 1
}

/* Funcion para escribir a 4 bits en el LCD */
void Lcd_4bit_Wr(char dato4,char rs){
    if (rs==0){ //pregunte si se va a enviar un comando o un dato
        PTAD_PTAD5=0; //prepara envío de comando RS = 0
    }
    else{
        PTAD_PTAD5=1; //prepara envio de dato RS = 1
    }
}

```

```

a=dato4;                                //almacena temporalmente el dato
a=>>4;                                  //desplaza parte alta dato para la baja
PTAD&=0xF0;                             //enmascara parte alta del puerto A
a&=0x0F;                                //enmascara parte baja del dato a enviar al LCD
PTAD_PTAD4=1;                           //sube pin de enable
Delay(20000);                           //hace retardo
PTAD|=a;                                 //envía nibble alto del dato
Delay(20000);                           //hace retardo
PTAD_PTAD4=0;                           //baja pin de enable
Delay(20000);                           //hace retardo
PTAD&=0xF0;                             //enmascara parte alta del puerto A
dato4&=0x0F;                            //enmascara parte baja del dato a enviar al LCD
PTAD_PTAD4=1;                           //sube pin de enable
Delay(20000);                           //hace retardo
PTAD|=dato4;                           //envía nibble bajo del dato
Delay(20000);                           //hace retardo
PTAD_PTAD4=0;                           //baja pin de enable
Delay(20000);                           //hace retardo
}

/* Función para envio de comando en 8 bits al LCD (solo para inicio) */
void Comando_8bit(char comando8){
    PTAD_PTAD5=0;                         //prepara envio de comando
    PTAD&=0xF0;                           //enmascara parte baja del puerto A
    comando8&=0xF0;                        //enmascara parte alta de comando a enviar al LCD
    comando8=comando8>>4;                //desplaza para tomar nibble alto
    PTAD_PTAD4=1;                         //sube pin de enable
    Delay(20000);                          //hace retardo
    PTAD|=comando8;                      //envía comando
    Delay(20000);                          //hace retardo
    PTAD_PTAD4=0;                         //sube pin de enable
    Delay(20000);                          //hace retardo
}

/* Funcion para inicializar el TPM1 en modo PWM EDGE ALIGN */
void TPM1_Init(char TPM1SC_Config, char TPM1C0SC_Config){
    TPM1SC=TPM1SC_Config;                //carga valor de configuracion para registro TPM1SC
    TPM1C0SC=TPM1C0SC_Config;           //carga valor de configuracion para registro TPM1C0SC
    TPM1MOD=65000;                      //valor del periodo del PWM
    TPM1C0V=6500;                       //valor equivalente al 10%, como valor por defecto
}

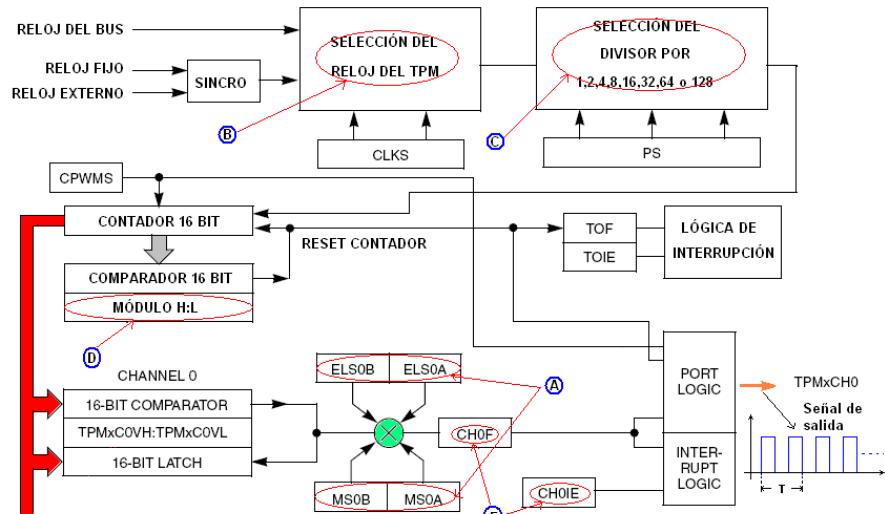
/* Funcion Delay(): Retarda basado en una variable tipo entero */
void Delay(unsigned long retardo){
    while(retardo>0){                  //llego a cero?
        retardo--;                     //no --> decrementa
    }
}

/* Funcion para convertir valor del DC del TPM1 a decenas y unidades */
void Convertir_Vvalor(unsigned long tempo){
    i=0;                                //inicializa variable de iteracion
    do{                                  //extrae las decenas contenidas en el DC
        tempo=tempo-10;
        ++i;
    }while (tempo>=10);
    decenas_DC= i+0x30;                  //actualice valor de las decenas
    unidades_DC= tempo+0x30;            //el resto son las unidades
}

/* Funcion de atencion a la interrupcion del TPM1 canal 0 */
interrupt VectorNumber_Vtpm1ch0 void ISR_TPM1_CH0 (void){
    TPM1C0SC;                           //preparese para borrar bandera del canal
    TPM1C0SC_CH0F=0;                   //aclare bandera del canal
    TPM1C0V = new_DC;                  //Actualice D.C. (Duty Cycle)
}

```

- **Funcionamiento del TPM como OUTPUT COMPARE:** En el modo OUTPUT COMPARE es posible generar pulsos de un determinado ancho o señales periódicas de una determinada frecuencia. La Figura 13.18 muestra las partes involucradas en un evento de salida de OUTPUT COMPARE.



**Figura 13.18. Generación de señal periódica con OUTPUT COMPARE.**

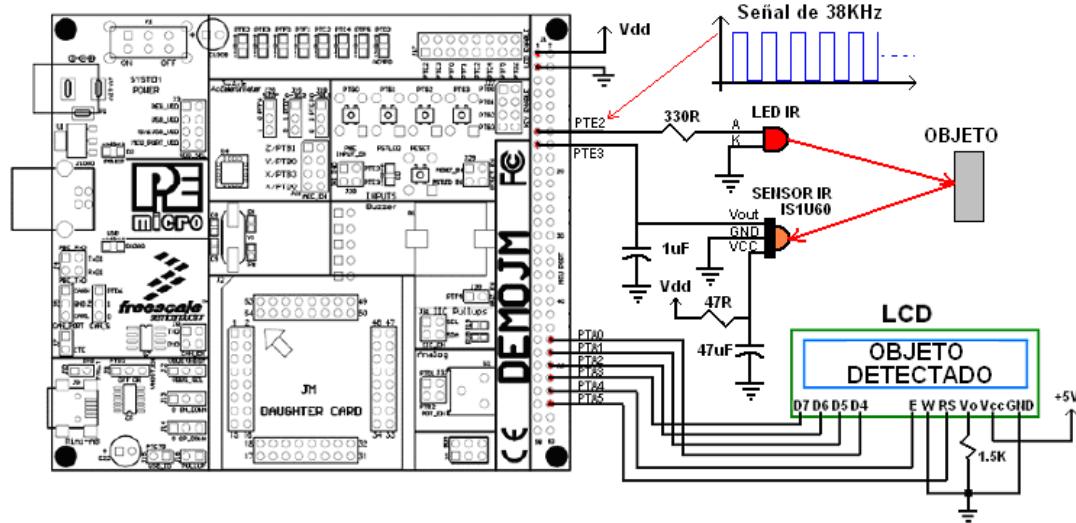
Para la generación de una señal periódica tipo OUTPUT COMPARE o de pulsos con un determinado ancho, sobre algún pin TPMxCHn, el primer paso es definir el modo de funcionamiento como OUTPUT COMPARE en los bits marcados como (A) (ver Tabla 13.1). El modo OUTPUT COMPARE tiene la opción de trabajar iniciando con un flanco de subida, flanco de bajada o cambiar de estado ante el evento de OUTPUT COMPARE. A continuación se elige el reloj apropiado según el rango de frecuencias de la señal a generar (B). Seguido (C), es necesario especificar un divisor para el reloj elegido. Luego se programa el período (T) de la señal en el módulo del contador de 16 bits (D), que para esta aplicación se convierte en el punto de cambio ante un evento de OUTPUT COMPARE. El evento de OUTPUT COMPARE puede ser atendido mediante *polling* sobre la bandera del canal CHxF o con la habilitación del bit CHxIE y la atención a la respectiva interrupción (E).

El circuito de la Figura 13.19 detalla una aplicación, para el modo OUTPUT COMPARE del TPM1. El sistema genera una señal de 38KHz y la aplica sobre un diodo infla-rojo (LED IR).

**NOTA:** Se recomienda al usuario revisar cuidadosamente el estado de los puentes del DEMOJM, con el objetivo de hacer compatible el ejercicio actual y de esta manera liberar los pines comprometidos en el circuito.

El LED IR emite la señal de luz y si esta choca contra un objeto que la refleje, de tal manera que el haz de luz alcance a llegar al sensor IR, se produce la señal Vout

aplicada al pin PTE3 del DEMOJM. La señal Vout se pone a “0” cuando el haz incide sobre el sensor IR.



**Figura 13.19. Circuito OUTPUT COMPARE para detección de objeto.**

El listado del programa que resuelve el problema se detalla a continuación.

```
/*
 * TPM_OUTPUT_COMPARE: Ejemplo sobre la utilizacion del modulo TPM en el modo OUTPUT COMPARE
 * main.c
 *
 * Fecha: Mayo 20, 2008
 *
 * V1.0, Rev 0 por Diego Munera
 *
 * Asunto: Implementar un codigo en C que genere una señal cuadrada periodica de 38KHz. El
 * objetivo es alimentar un sensor optico de objetos, por reflexion. El sensor debe
 * distinguir objetos que se encuentren proximos a un foco de deteccion.
 *
 * Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros
 * editores.
 */
***** DISTRIBUCION DE LOS PINES DE PUERTOS *****
***** PUERTO      * PIN      * I/O      * TIPO      * INICIA      * COMENTARIO *****
/*PTA      * -      * -      * -      * -      *LCD
/*      *PTA0      * O      * D      * 1      *D4 del LCD
/*      *PTA1      * O      * D      * 1      *D5 del LCD
/*      *PTA2      * O      * D      * 1      *D6 del LCD
/*      *PTA3      * O      * D      * 1      *D7 del LCD
/*      *PTA4      * O      * D      * 1      *Señal ENABLE del LCD
/*      *PTA5      * O      * D      * 1      *Señal RS del LCD
/*PTB      * -      * -      * -      *      *No usado
/*PTC      * -      * -      * -      *      *No usado
*****
```

```

/*PTD      * -      * -      * -      * -      *No usado      */
/********************************************* */
/*PTE      * -      * -      * -      * -      *TPM1 como OUTPUT COMPARE      */
/*      *PTE2      * O      * D      * -      *Salida del OUTPUT COMPARE (señal de 38KHz) */
/*      *PTE3      * I      * D      * 1      *Entrada de tección de objeto      */
/********************************************* */
/*PTF      * -      * -      * -      * -      *No usado      */
/********************************************* */
/*PTG      * -      * -      * -      * -      *Pulsadores      */
/********************************************* */
/*PTH      * -      * -      * -      * -      *No usado      */
/********************************************* */
/*PTJ      * -      * -      * -      * -      *No usado      */
/********************************************* */

/* Archivos de cabecera */
#include <hidef.h>           //macro para interrupciones
#include "derivative.h"        //macro de declaraciones del MCU

/* Declaracion de variables y constantes */
byte bandera=0;                //bandera de proposito general
unsigned long retardo=0;         //parametro para hacer retardo
char i=0;                       //variable de iteracion
char PTAD_Config=0;             //byte de inicializacion del registro PTAD
char PTADD_Config=0;             //byte de inicializacion del registro PTADD
char PTAPE_Config=0;             //byte de inicializacion del registro PTAPE
char PTASE_Config=0;             //byte de inicializacion del registro PTASE
char PTADS_Config=0;             //byte de inicializacion del registro PTADS
char PTAIFE_Config=0;            //byte de inicializacion del registro PTAIFE
char PTED_Config=0;              //byte de inicializacion del registro PTED
char PTEDD_Config=0;              //byte de inicializacion del registro PTEDD
char PTEPE_Config=0;              //byte de inicializacion del registro PTEPE
char PTESE_Config=0;              //byte de inicializacion del registro PTESE
char PTEDS_Config=0;              //byte de inicializacion del registro PTEDS
char PTEIFE_Config=0;              //byte de inicializacion del registro PTEIFE
char comando8=0;                 //parametro inicializacion LCD modo 8 bits
char dato4;                      //dato a enviar al LCD
char rs=0;                       //señal de dato o comando al LCD
char a=0;                         //variable temporal
const unsigned char mensaje1[] = "  OBJETO  ";           //Arreglo del mensaje 1 para llevar al LCD
const unsigned char mensaje2[] = "  DETECTADO  ";          //Arreglo del mensaje 2 para llevar al LCD
const unsigned char mensaje3[] = "  NO DETECTADO  ";       //Arreglo del mensaje 3 para llevar al LCD
char *pM;                          //puntero a mensaje para el LCD
char TPM1SC_Config=0;              //byte de inicializacion del registro TPM1SC
char TPM1C0SC_Config=0;             //byte de inicializacion del registro TPM1C0SC

/* Macros y definiciones */
#define Disable_COP() SOPT1 &= 0x3F //deshabilita el COP
#define DETECTO() !(PTED_PTED3) //macro para lectura de señal deteccion de objeto

/* Declaracion de funciones */
void Delay(unsigned long retardo);    //funcion para hacer reatrdos varios
void LCD_Init(void);                  //funcion para inicilaizar LCD
void Lcd_4bit_Wr(char dato4,char rs); //funcion para escribir dato al LCD
void Comando_8bit(char comando8);     //funcion para inicio del LCD a 8 bits
void TPM1_Init(char TPM1SC_Config, char TPM1C0SC_Config); //funcion para inicializar el TPM
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config); //declare funcion que inicializa PTA
void PTE_Init(char PTED_Config,char PTEDD_Config,char PTEPE_Config,char PTESE_Config,char PTEDS_Config,char PTEIFE_Config); //declare funcion que inicializa PTG

/* main(): Funcion principal */
void main(void) {
    Disable_COP();                  //deshabilita el COP

    // Inicializacion del reloj:
    MCGTRM = NVMCGTRM + 52;         //Tome valor programado de fabrica del TRIM
}

```

```

MCGC1 = 0x04;           //Modo FEI divisor por 1
MCGC2 = 0x00;           //Desactiva modo de reloj externo
MCGC4 = 0x22;           //Rango alto del DCO
                        //El reloj MCGOUT queda en 24999945 Hz

PTA_Init(0x00,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "0",pines como salida, no pullups, si slew, no strength y si filtro
PTE_Init(0x00,0x00,0x08,0x00,0x00,0xFF); //pines inician en "0",pines como entrada, pullup PTE3, no slew, no strength y si filtro

LCD_Init();              //inicialice LCD
TPM1_Init(0x08,0x54);   //PWM OUTPUT COMPARE, Toggle, reloj bus interno, divisor por 1
                        //habilita interrupcion por canal
                        //para una base de tiempo del contador del TPM de 0.4useg
pM = (char *)mensaje1;
for (i=0;i<=15;i++){
    Lcd_4bit_Wr(*pM,1);
    pM++;
}

for(;;){
    if(DETECTO()){
        pM = (char *)mensaje2;
        Lcd_4bit_Wr(0xC0,0);
        for (i=0;i<=15;i++){
            Lcd_4bit_Wr(*pM,1);
            pM++;
        }
    }
    if(!DETECTO()){
        pM = (char *)mensaje3;
        Lcd_4bit_Wr(0xC0,0);
        for (i=0;i<=15;i++){
            Lcd_4bit_Wr(*pM,1);
            pM++;
        }
    }
}
/* es necesario asegurarse de nunca abandonar el main */

/* Funcion para inicializar el puerto A */
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config){
    PTAD=PTAD_Config;          //inicialice estado pinos PTA
    PTADD=PTADD_Config;        //inicialice dirección de pinos PTA
    PTAPE=PTAPE_Config;        //inicialice estado de pullups PTA
    PTASE=PTASE_Config;        //inicialice estado del slew rate PTA
    PTADS=PTADS_Config;        //inicialice estado de drive strength PTA
    PTAIFE=PTAIFE_Config;      //inicialice estado del filtro PTA
}

/* Funcion para inicializar el puerto E */
void PTE_Init(char PTED_Config,char PTEDD_Config,char PTEPE_Config,char PTESE_Config,char PTEDS_Config,char PTEIFE_Config){
    PTED=PTED_Config;          //inicialice estado pinos PTE
    PTEDD=PTEDD_Config;        //inicialice dirección de pinos PTE
    PTEPE=PTEPE_Config;        //inicialice estado de pullups PTE
    PTESE=PTESE_Config;        //inicialice estado del slew rate PTE
    PTEDS=PTEDS_Config;        //inicialice estado de drive strength PTE
    PTEIFE=PTEIFE_Config;      //inicialice estado del filtro PTE
}

/* Funcion para inicializacion del LCD a 4 bits */
void LCD_Init(void){
    Delay(200000);             //hace retardo
    Comando_8bit(0x30);        //llama función enviar comando en 8 bits con comando
    Delay(200000);             //hace retardo
    Comando_8bit(0x30);        //llama función enviar comando en 8 bits con comando
    Delay(200000);             //hace retardo
    Comando_8bit(0x20);        //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x28,0);       //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x0C,0);       //ON LCD, OFF cursor
}

```

```

Lcd_4bit_Wr(0x01,0);           //borrar pantalla LCD
Lcd_4bit_Wr(0x06,0);           //desplaza cursor a la derecha con cada caracter
Lcd_4bit_Wr(0x80,0);           //primer caracter fila 1 columna 1
}

/* Funcion para escribir a 4 bits en el LCD */
void Lcd_4bit_Wr(char dato4,char rs){
if (rs==0){                      //pregunte si se va a enviar un comando o un dato
    PTAD_PTAD5=0;                //prepara envío de comando RS = 0
}
else{
    PTAD_PTAD5=1;                //prepara envio de dato RS = 1
}
a=dato4;                         //almacena temporalmente el dato
a=a>>4;                          //desplaza parte alta dato para la baja
PTAD&=0xF0;                       //enmascara parte alta del puerto A
a&=0x0F;                          //enmascara parte baja del dato a enviar al LCD
PTAD_PTAD4=1;                     //sube pin de enable
Delay(20000);                     //hace retardo
PTAD|=a;                           //envía nibble alto del dato
Delay(20000);                     //hace retardo
PTAD_PTAD4=0;                     //baja pin de enable
Delay(20000);                     //hace retardo
PTAD&=0xF0;                       //enmascara parte alta del puerto A
dato4&=0x0F;                      //enmascara parte baja del dato a enviar al LCD
PTAD_PTAD4=1;                     //sube pin de enable
Delay(20000);                     //hace retardo
PTAD|=dato4;                      //envía nibble bajo del dato
Delay(20000);                     //hace retardo
PTAD_PTAD4=0;                     //baja pin de enable
Delay(20000);                     //hace retardo
}

/* Función para envio de comando en 8 bits al LCD (solo para inicio) */
void Comando_8bit(char comando8){
PTAD_PTAD5=0;                     //prepara envio de comando
PTAD&=0xF0;                        //enmascara parte baja del puerto A
comando8&=0xF0;                    //enmascara parte alta de comando a enviar al LCD
comando8=comando8>>4;            //desplaza para tomar nibble alto
PTAD_PTAD4=1;                     //sube pin de enable
Delay(20000);                     //hace retardo
PTAD|=comando8;                   //envía comando
Delay(20000);                     //hace retardo
PTAD_PTAD4=0;                     //sube pin de enable
Delay(20000);                     //hace retardo
}

/* Funcion para inicializar el TPM1 en modo OUTPUT COMPARE TOGGLE */
void TPM1_Init(char TPM1SC_Config, char TPM1C0SC_Config){
TPM1SC=TPM1SC_Config;             //carga valor de configuracion para registro TPM1SC
TPM1C0SC=TPM1C0SC_Config;         //carga valor de configuracion para registro TPM1C0SC
TPM1MOD=329;                      //valor del periodo del PWM para 38KHz con una base de
                                  //tiempo de 0.4 us. El toggle se produce a la mitad del periodo
}

/* Funcion Delay(): Retarda basado en una variable tipo entero */
void Delay(unsigned long retardo){
while(retardo>0){                  //llego a cero?
    retardo--;                      //no --> decrementa
}
}

```

### 13.3. EL RTC (REAL TIME COUNTER)

El contador de tiempo real está formado por un contador de 8 bits, que se incrementa contres posibles fuentes de reloj y se preescala con una gran variedad de divisores. Este contador contiene un comparador de 8 bits, que permanentemente esta verificando la igualdad del contador contra el valor programado en este.

El RTC opera en los modos de bajo consumo (WAIT, STOP2 y STOP3), con el propósito de permitir que el usuario libere la máquina de dichos modos, ante un evento de conteo del RTC.

El diagrama de la Figura 13.20 ilustra las principales componentes del RTC.

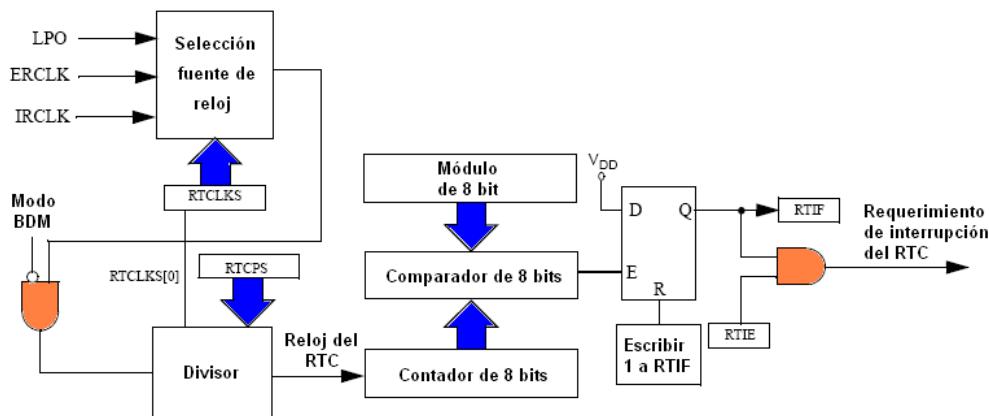


Figura 13.20. Diagrama en bloques módulo RTC.

El reloj del sistema puede provenir de tres diferentes fuentes, como el reloj de referencia externa (ERCLK), el reloj de referencia interna (IRCLK) y un reloj de 1KHz, interno, de baja potencia (LPO). Este reloj puede ser dividido por múltiplos de  $2^n$  o múltiplos de  $10^n$ .

El valor a contar se programa en el registro de módulo de 8 bits, el cual es comparado contra el valor actual del contador de 8 bits. En caso de producirse una coincidencia el usuario podrá enterarse vía la bandera RTIF, mediante *polling* o habilitando un evento de interrupción por RTC, mediante el bit RTIE.

Para aclarar el estado de la bandera RTIF y permitir que el RTC quede listo para un nuevo conteo, es necesario aclarar el bit RTIF.

- **Registros asociados al RTC**

Los registros para examinar el estado y permitir el control sobre el módulo RTC son los siguientes:

- **Registro de estado y control (RTCSC):** La Figura 13.21 ilustra el detalle de cada bit de estado y control del RTC.

### Registro RTCSC: (FFFF806C)

	7	6	5	4	3	2	1	0
Lectura	RTIF	RTCLKS		RTIE	RTCPS			
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 13.21. Registro RTCSC.

- **RTIF:** Bandera de interrupción del RTC, que indica que el contador ha alcanzado el valor programado en el módulo. La señal de RESET aclara el valor de la bandera y la acción de escribir un “0” en esta, también.
  - 0:** No se ha presentado evento de interrupción del RTC
  - 1:** Hay un evento de interrupción del RTC
- **RTCLKS:** Bits para la selección de la fuente de reloj que alimenta al contador del RTC. La acción de cambiar la fuente de reloj, aclara el divisor y el contador del RTC.
  - 00:** Selección del reloj de bajo consumo LPO de 1KHz
  - 01:** Selección del reloj de referencia externa ERCLK
  - 1X:** Selección del reloj de referencia interna IRCLK
- **RTIE:** Bit para habilitar un evento de interrupción por RTC.
  - 0:** Interrupción por RTC inhibida
  - 1:** Interrupción por RTC habilitada
- **RTCPS:** Bits para la selección del divisor aplicado al reloj del RTC. La Tabla 35.2 ilustra los preescaler posibles para la fuente de reloj del RTC.

Tabla 13.2. Valores del preescaler para el reloj del RTC.

RTCLKS[0]	0	RTCPS														
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	Apagado	$2^3$	$2^5$	$2^6$	$2^7$	$2^8$	$2^9$	$2^{10}$	1	2	$2^2$	10	$2^4$	$10^2$	$5 \times 10^2$	$10^3$
1	Apagado	$2^{10}$	$2^{11}$	$2^{12}$	$2^{13}$	$2^{14}$	$2^{15}$	$2^{16}$	$10^3$	$2 \times 10^3$	$5 \times 10^3$	$10^4$	$2 \times 10^4$	$5 \times 10^4$	$10^5$	$2 \times 10^5$

- **Registro contador de 8 bits (RTCCNT):** La Figura 13.22 ilustra el detalle del registro de sólo lectura, como contador de 8 bits del RTC.

### Registro contador de 8 bits del RTC (RTCCNT): (FFFF806D)

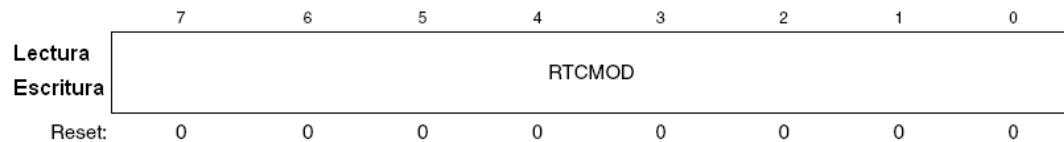
Lectura	7	6	5	4	3	2	1	0
Escritura	RTCCNT							
Reset:	0	0	0	0	0	0	0	0

Figura 13.22. Registro RTCCNT.

**RTCCNT:** Estos 7 bits indican el estado de conteo del contador del RTC. Escribir en los bits RTCLKS y RTCPS, aclara el registro RTCCNT.

- **Registro módulo de 8 bits (RTCMOD):** La Figura 13.23 ilustra el detalle del registro módulo de 8 bits del RTC.

**Registro módulo de 8 bits del RTC (RTCMOD): (FFFF806E)**



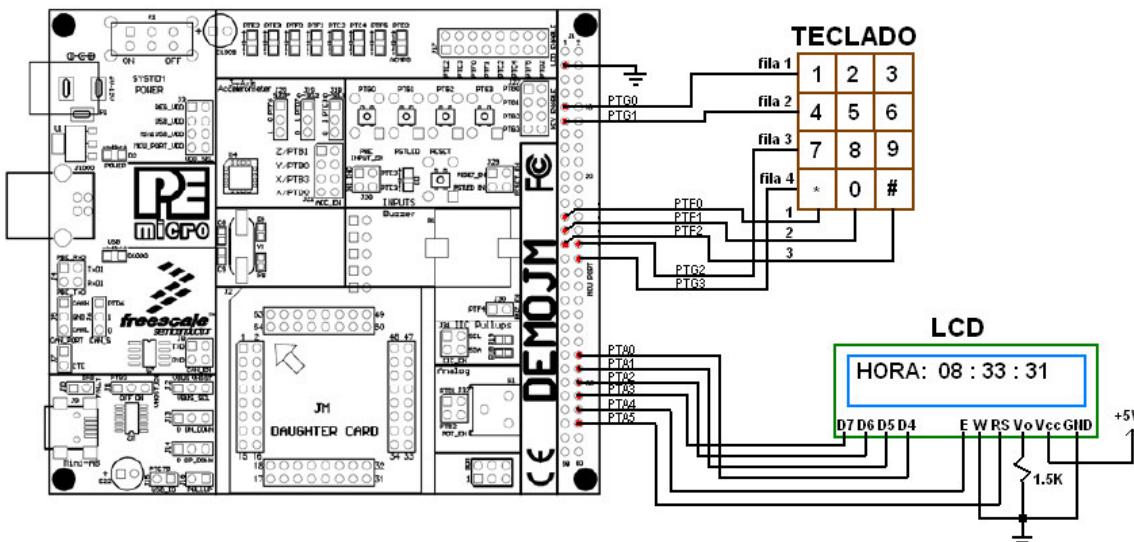
**Figura 13.23. Registro RTCMOD.**

**RTCMOD:** En estos 7 bits debe programarse el módulo de conteo del contador del RTC. Cuando el contador del RTC alcanza dicho módulo, la bandera RTIF se pone a “1” y el contador del RTC se pone a “0x00”.

#### 13.4. EJERCICIO CON EL RTC: RELOJ DIGITAL INICIALIZADO POR TECLADO

El ejercicio propuesto a continuación ilustra el uso del RTC como reloj de tiempo real, preestablecido mediante un teclado tipo telefónico. La Figura 13.24 detalla el circuito sugerido para el ejercicio.

**NOTA:** Se recomienda al usuario revisar cuidadosamente el estado de los puentes del DEMOJM, con el objetivo de hacer compatible el ejercicio actual y de esta manera liberar los pines comprometidos en el circuito.



**Figura 13.24. Circuito reloj con RTC**

El usuario introduce, inicialmente, el estado de arranque del reloj (*preset*), vía teclado. Una vez finalice la entrada de la hora de arranque, el reloj comenzará a contar.

El reloj trabaja en el formato HH:MM:SS y cuenta hasta las 12:59:50, para volver a 00:00:00. El listado del programa que resuelve el problema se detalla a continuación.

```
/****************************************************************************
 * RTC_RELOJ: Ejemplo sobre la utilización del modulo RTC.          */
 /* main.c                                                               */
 /*                                                                     */
 /* Fecha: Mayo 25, 2008                                              */
 /*                                                                     */
 /* V1.0, Rev 0 por Diego Múnera                                       */
 /*                                                                     */
 /* Asunto: Implementar un codigo en C que presente en un LCD un reloj de tiempo real con for- */
 /* mato HH:MM:SS. El reloj debe poderse inicializar al salir de RESET el sistema.           */
 /*                                                                     */
 /* Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros   */
 /* editores.                                                       */
 */

/*
 * DISTRIBUCION DE LOS PINES DE PUERTOS
 */

/* PUERTO * PIN * I/O * TIPO * INICIA *      COMENTARIO */
/*-----*/
/*PTA * - * - * - *LCD */
/* *PTA0 * O * D * 1 *D4 del LCD */
/* *PTA1 * O * D * 1 *D5 del LCD */
/* *PTA2 * O * D * 1 *D6 del LCD */
/* *PTA3 * O * D * 1 *D7 del LCD */
/* *PTA4 * O * D * 1 *Señal ENABLE del LCD */
/* *PTA5 * O * D * 1 *Señal RS del LCD */
/*-----*/
/*PTB * - * - * D *0x00 *No usado */
/*-----*/
/*PTC * - * - * - *No usado */
/*-----*/
/*PTD * - * - * - *No usado */
/*-----*/
/*PTE * - * - * - *No usado */
/*-----*/
/*PTF * - * - * - *No usado */
/* *PTF0 * O * D * 1 *Columna 1 teclado */
/* *PTF1 * O * D * 1 *Columna 2 teclado */
/* *PTF2 * O * D * 1 *Columna 3 teclado */
/*-----*/
/*PTG * - * - * - *Teclado */
/* *PTG0 * I * D * 1 *Fila 1 del teclado */
/* *PTG1 * I * D * 1 *Fila 2 del teclado */
/* *PTG2 * I * D * 1 *Fila 3 del teclado */
/* *PTG3 * I * D * 1 *Fila 4 del teclado */
/*-----*/
/*PTH * - * - * - *No usado */
/*-----*/
/*PTJ * - * - * - *No usado */
/*-----*/
/* Archivos de cabecera */
```

```

#include <hidef.h>           //macro para interrupciones
#include "derivative.h"       //macro de declaraciones del MCU

/* Declaracion de variables y constantes */
const unsigned char mensaje[] = "HORA: : : "; //mensaje fijo en el LCD
unsigned int segundos=0,minutos=0,horas=0;      //variables para llevar el tiempo
byte dseg,useg,dmin,umin,dhor,uhor;            //variables temporales manejo del tiempo
byte decseg,uniseg,decmin,unimin,dechor,unihor; //variables temporales tiempo al LCD

byte bandera=0;                                //bandera de proposito general
unsigned long retardo=0;                        //parametro para hacer retardo
char i=0;                                       //variable de iteracion
char columna=0;                                 //columna detectada del teclado
char fila=0;                                    //fila detectada del teclado
char comando8=0;                               //parametro inicializacion LCD modo 8 bits
char dato4;                                     //dato a enviar al LCD
char rs=0;                                      //señal de dato o comando al LCD
char a=0;                                       //variable temporal
char KBI1SC_Config=0;                          //byte de inicializacion del registro KBI1SC
char KBI1PE_Config=0;                          //byte de inicializacion del registro KBI1PE
char KBI1ES_Config=0;                          //byte de inicializacion del registro KBI1ES
char PTAD_Config=0;                            //byte de inicializacion del registro PTAD
char PTADD_Config=0;                           //byte de inicializacion del registro PTADD
char PTAPE_Config=0;                           //byte de inicializacion del registro PTAPE
char PTASE_Config=0;                           //byte de inicializacion del registro PTASE
char PTADS_Config=0;                           //byte de inicializacion del registro PTADS
char PTAIFE_Config=0;                          //byte de inicializacion del registro PTAIFE
char PTFD_Config=0;                            //byte de inicializacion del registro PTFD
char PTFDD_Config=0;                           //byte de inicializacion del registro PTFDD
char PTFPE_Config=0;                           //byte de inicializacion del registro PTFPE
char PTFSE_Config=0;                           //byte de inicializacion del registro PTFSE
char PTFDS_Config=0;                           //byte de inicializacion del registro PTFDS
char PTFIFE_Config=0;                          //byte de inicializacion del registro PTFIFE
char RTCMOD_Config=0;                          //byte de inicializacion del registro RTCMOD
char RTCSC_Config=0;                           //byte de inicializacion del registro RTCSC
char *pM;                                      //Puntero a mensaje para el LCD
byte tecla,tipo;                             //mas variables temporales

/* Macros y definiciones */
#define Disable_COP() SOPT1 &= 0x3F          //deshabilita el COP

/* Declaracion de funciones */
void KBI_Init(char KBI1SC_Config,char KBI1PE_Config,char KBI1ES_Config); //declare funcion que inicializa KBI1
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config); //declare funcion que inicializa PTA
void PTF_Init(char PTFD_Config,char PTFDD_Config,char PTFPE_Config,char PTFSE_Config,char PTFDS_Config,char PTFIFE_Config); //declare funcion que inicializa PTF
void Delay(unsigned long retardo);           //funcion para hacer reatrdos varios
void LCD_Init(void);                        //funcion para inicializar LCD
void Lcd_4bit_Wr(char dato4,char rs);       //funcion para escribir dato al LCD
void Comando_8bit(char comando8);           //funcion para inicio del LCD a 8 bits
void Rote_Col (void);                      //funcion para rotar un cero por columnas teclado
void Preset(unsigned char tipo);            //funcion para establecer preset del reloj
void RTC_Init(char RTCMOD_Config,char RTCSC_Config); //funcion para inicializacion del modulo RTC

/* main(): Funcion principal */
void main(void) {
    Disable_COP();                         //deshabilita el COP

    // Inicializacion del reloj:
    MCGTRM = NVCMCGTRM + 52;               //Tome valor programado de fabrica del TRIM
    MCGC1 = 0x04;                          //Modo FEI divisor por 1
    MCGC2 = 0x00;                          //Desactiva modo de reloj externo
    MCGC4 = 0x22;                          //Rango alto del DCO
                                         //El reloj MCGOUT queda en 2499945 Hz
}

```

```

EnableInterrupts;
PTA_Init(0x00,0xFF,0x00,0xFF,0x00,0xFF);
PTF_Init(0xFF,0xFF,0x00,0xFF,0x00,0xFF);

PTGPE=0xFF;
KBI_Init(0x06,0xC3,0x00);
RTC_Init(0x00,0x0F);

LCD_Init();
pM = (char *)mensaje;
for (i=0;i<=13;i++){
    Lcd_4bit_Wr(*pM,1);
    pM++;
}

//Programación preset de la hora:
Lcd_4bit_Wr(0x86,0);
do{
    Rote_Col();
}while (bandera==0);
bandera = 0;
Preset(0);
Delay(800000);
Lcd_4bit_Wr(0x87,0);
do{
    Rote_Col();
}while (bandera==0);
bandera = 0;
Preset(1);
horas=dhor*10 + uhor;
Delay(800000);
Lcd_4bit_Wr(0x89,0);
do{
    Rote_Col();
}while (bandera==0);
bandera = 0;
Preset(2);
Delay(800000);
Lcd_4bit_Wr(0x8A,0);
do{
    Rote_Col();
}while (bandera==0);
bandera = 0;
Preset(3);
minutos=dmin*10 + umin;
Delay(800000);
Lcd_4bit_Wr(0x8C,0);
do{
    Rote_Col();
}while (bandera==0);
bandera = 0;
Preset(4);
Delay(800000);
Lcd_4bit_Wr(0x8D,0);
do{
    Rote_Col();
}while (bandera==0);
bandera = 0;
Preset(5);
segundos=dseg*10 + useg;
Lcd_4bit_Wr(0x0C,0);
Delay(800000);
RTCSC_RTIE=1;

//Ciclo iterativo:
for (;;) {
}

//Actualización del LCD:
Lcd_4bit_Wr(0x8D,0);

```

//habilita interrupciones  
 //pines inician en "0", pins como salida, no pullups, si slew, no strength y  
 //si filtro  
 //pines inician en "1", pins como salida, no pullups, si slew, no strength y  
 //si filtro  
 //habilita pullups puerto G  
 //da ACK, habilita interr, habilita pullups y flanco de bajada  
 //modulo al maximo, reloj LPO (1KHz) y divisor por 1000 para  
 //conteo de un segundo, inhibe interrupcion  
 //inicialice LCD  
 //puntero toma dirección del mensaje con definición  
 //ciclo para imprimir 10 caracteres del mensaje  
 //envia carácter al LCD  
 //incrementa puntero al arreglo del mensaje

//cursor del LCD a posición decenas horas  
 //espere a que se ingrese una tecla  
 //mientras, rote el cero por las columnas  
 //aclare bandera  
 //función para establecer preset del reloj  
 //retardo para esperar siguiente tecla  
 //cursor del LCD a posición unidades horas  
 //espere a que se ingrese una tecla  
 //mientras, rote el cero por las columnas  
 //aclare bandera  
 //función para establecer preset del reloj  
 //actualice valor de las horas según preset  
 //retardo para esperar siguiente tecla  
 //cursor del LCD a posición decenas minutos  
 //espere a que se ingrese una tecla  
 //mientras, rote el cero por las columnas  
 //aclare bandera  
 //función para establecer preset del reloj  
 //actualice valor de las horas según preset  
 //retardo para esperar siguiente tecla  
 //cursor del LCD a posición unidades minutos  
 //espere a que se ingrese una tecla  
 //mientras, rote el cero por las columnas  
 //aclare bandera  
 //función para establecer preset del reloj  
 //actualice valor de las horas según preset  
 //retardo para esperar siguiente tecla  
 //cursor del LCD a posición decenas segundos  
 //actualice valor de las horas según preset  
 //mientras, rote el cero por las columnas  
 //aclare bandera  
 //función para establecer preset del reloj  
 //retardo para esperar siguiente tecla  
 //cursor del LCD a posición unidades segundos  
 //espere a que se ingrese una tecla  
 //mientras, rote el cero por las columnas  
 //aclare bandera  
 //función para establecer preset del reloj  
 //actualice valor de las horas según preset  
 //ON LCD, OFF cursor  
 //retardo para esperar siguiente tecla  
 //habilita interrupción del RTC

//proceso iterativo del main  
 //posición para las unidades de segundo



```

}

/* Funcion para escribir a 4 bits en el LCD */
void Lcd_4bit_Wr(char dato4,char rs){
    if (rs==0){                                //pregunte si se va a enviar un comando o un dato
        PTAD_PTAD5=0;                          //prepara envío de comando RS = 0
    }
    else{
        PTAD_PTAD5=1;                          //prepara envio de dato RS = 1
    }
    a=dato4;                                  //almacena temporalmente el dato
    a=a>>4;                                   //desplaza parte alta dato para la baja
    PTAD&=0xF0;                               //enmascara parte alta del puerto A
    a&=0x0F;                                 //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;                            //sube pin de enable
    Delay(10000);                            //hace retardo
    PTADl=a;                                 //envía nibble alto del dato
    Delay(10000);                            //hace retardo
    PTAD_PTAD4=0;                            //baja pin de enable
    Delay(10000);                            //hace retardo
    PTAD&=0xF0;                               //enmascara parte alta del puerto A
    dato4&=0x0F;                             //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;                            //sube pin de enable
    Delay(10000);                            //hace retardo
    PTADl=dato4;                            //envía nibble bajo del dato
    Delay(10000);                            //hace retardo
    PTAD_PTAD4=0;                            //baja pin de enable
    Delay(10000);                            //hace retardo
}

/* Funcion para envio de comando en 8 bits al LCD (solo para inicio) */
void Comando_8bit(char comando8){
    PTAD_PTAD5=0;                            //prepara envio de comando
    PTAD&=0xF0;                             //enmascara parte baja del puerto A
    comando8&=0xF0;                           //enmascara parte alta de comando a enviar al LCD
    comando8=comando8>>4;                  //desplaza para tomar nibble alto
    PTAD_PTAD4=1;                            //sube pin de enable
    Delay(10000);                            //hace retardo
    PTADl=comando8;                         //envía comando
    Delay(10000);                            //hace retardo
    PTAD_PTAD4=0;                            //sube pin de enable
    Delay(10000);                            //hace retardo
}

/* Funcion Delay(): Retarda basado en una variable tipo entera */
void Delay(unsigned long retardo){
    while(retardo>0){                      //llego a cero?
        retardo--;                          //no --> decrementa
    }
}

/* Funcion para rotar cero por columnas teclado */
void Rote_Col (void){
    PTFD = 0x0E;                            //cero a la columna 1
    Delay(500);                            //hace retardo
    PTFD = 0x0D;                            //cero a la columna 2
    Delay(500);                            //hace retardo
    PTFD = 0x0B;                            //cero a la columna 3
    Delay(500);                            //hace retardo
}

/* Función para establecer el PRESET del reloj */
void Preset (unsigned char tipo){

    if(tipo==0){                           //si se están actualizando las decenas de hora
        switch(tecla){
            case 1:                        //y si se presiono un 1
                Lcd_4bit_Wr(0x31,1);      //envie un uno
                dhor=1;                   //presetea las decenas de hora
        }
    }
}

```

```

break;
case 2:
Lcd_4bit_Wr(0x32,1);
dhor=2;
break;
case 3:
Lcd_4bit_Wr(0x33,1);
dhor=3;
break;
case 4:
Lcd_4bit_Wr(0x34,1);
dhor=4;
break;
case 5:
Lcd_4bit_Wr(0x35,1);
dhor=5;
break;
case 6:
Lcd_4bit_Wr(0x36,1);
dhor=6;
break;
case 7:
Lcd_4bit_Wr(0x37,1);
dhor=7;
break;
case 8:
Lcd_4bit_Wr(0x38,1);
dhor=8;
break;
case 9:
Lcd_4bit_Wr(0x39,1);
dhor=9;
break;
case 0:
Lcd_4bit_Wr(0x30,1);
break;
dhor=0;
}
}

if(tipo==1){
switch(tecla){
case 1:
Lcd_4bit_Wr(0x31,1);
uhor=1;
break;
case 2:
Lcd_4bit_Wr(0x32,1);
uhor=2;
break;
case 3:
Lcd_4bit_Wr(0x33,1);
uhor=3;
break;
case 4:
Lcd_4bit_Wr(0x34,1);
uhor=4;
break;
case 5:
Lcd_4bit_Wr(0x35,1);
uhor=5;
break;
case 6:
Lcd_4bit_Wr(0x36,1);
uhor=6;
break;
case 7:
Lcd_4bit_Wr(0x37,1);
uhor=7;
break;
case 8:
}
}

//y si se presiono un 2
//envíe un dos
//presetee las decenas de hora

//Y si se presiono un 3
//envíe un tres
//presetee las decenas de hora

//y si se presiono un 4
//envíe un cuatro
//presetee las decenas de hora

//y si se presiono un 5
//envíe un cinco
//presetee las decenas de hora

//y si se presiono un 6
//envíe un seis
//presetee las decenas de hora

//y si se presiono un 7
//envíe un siete
//presetee las decenas de hora

//y si se presiono un 8
//envíe un ocho
//presetee las decenas de hora

//y si se presiono un 9
//envíe un nueve
//presetee las decenas de hora

//y si se presiono un 0
//envíe un cero
//presetee las decenas de hora

//si se están actualizando las unidades de hora

//y si se presiono un 1
//envíe un uno
//presetee las unidades de hora

//y si se presiono un 2
//envíe un dos
//presetee las unidades de hora

//y si se presiono un 3
//envíe un tres
//presetee las unidades de hora

//y si se presiono un 4
//envíe un cuatro
//presetee las unidades de hora

//y si se presiono un 5
//envíe un cinco
//presetee las unidades de hora

//y si se presiono un 6
//envíe un seis
//presetee las unidades de hora

//y si se presiono un 7
//envíe un siete
//presetee las unidades de hora

//y si se presiono un 8

```

```

Lcd_4bit_Wr(0x38,1);           //envie un ocho
uhor=8;                         //presetee las unidades de hora
break;
case 9:                          //y si se presiono un 9
Lcd_4bit_Wr(0x39,1);           //envie un nueve
uhor=9;                         //presetee las unidades de hora
break;
case 0:                          //y si se presiono un 0
Lcd_4bit_Wr(0x30,1);           //envie un cero
break;
uhor=0;                          //presetee las unidades de hora
}
}

if(tipo==2){                     //si se están actualizando las decenas de minuto
switch(tecla){
    case 1:                      //y si se presiono un 1
    Lcd_4bit_Wr(0x31,1);         //envie un uno
    dmin=1;                       //presetee las decenas de minuto
    break;
    case 2:                      //y si se presiono un 2
    Lcd_4bit_Wr(0x32,1);         //envíe un dos
    dmin=2;                       //presetee las decenas de minuto
    break;
    case 3:                      //y si se presiono un 3
    Lcd_4bit_Wr(0x33,1);         //envie un tres
    dmin=3;                       //presetee las decenas de minuto
    break;
    case 4:                      //y si se presiono un 4
    Lcd_4bit_Wr(0x34,1);         //envie un cuatro
    dmin=4;                       //presetee las decenas de minuto
    break;
    case 5:                      //y si se presiono un 5
    Lcd_4bit_Wr(0x35,1);         //envie un cinco
    dmin=5;                       //presetee las decenas de minuto
    break;
    case 6:                      //y si se presiono un 6
    Lcd_4bit_Wr(0x36,1);         //envie un seis
    dmin=6;                       //presetee las decenas de minuto
    break;
    case 7:                      //y si se presiono un 7
    Lcd_4bit_Wr(0x37,1);         //envie un siete
    dmin=7;                       //presetee las decenas de minuto
    break;
    case 8:                      //y si se presiono un 8
    Lcd_4bit_Wr(0x38,1);         //envie un ocho
    dmin=8;                       //presetee las decenas de minuto
    break;
    case 9:                      //y si se presiono un 9
    Lcd_4bit_Wr(0x39,1);         //envie un nueve
    dmin=9;                       //presetee las decenas de minuto
    break;
    case 0:                      //y si se presiono un 0
    Lcd_4bit_Wr(0x30,1);         //envie un cero
    dmin=0;                       //presetee las decenas de minuto
}
}

if(tipo==3){                     //si se están actualizando las unidades de minuto
switch(tecla){
    case 1:                      //y si se presiono un 1
    Lcd_4bit_Wr(0x31,1);         //envie un uno
    umin=1;                       //presetee las unidades de minuto
    break;
    case 2:                      //y si se presiono un 2
    Lcd_4bit_Wr(0x32,1);         //envíe un dos
    umin=2;                       //presetee las unidades de minuto
    break;
    case 3:                      //y si se presiono un 3
    Lcd_4bit_Wr(0x33,1);         //envie un tres
    umin=3;                       //presetee las unidades de minuto
}
}

```

```

break;
case 4:
Lcd_4bit_Wr(0x34,1);
umin=4;
break;
case 5:
Lcd_4bit_Wr(0x35,1);
umin=5;
break;
case 6:
Lcd_4bit_Wr(0x36,1);
umin=6;
break;
case 7:
Lcd_4bit_Wr(0x37,1);
umin=7;
break;
case 8:
Lcd_4bit_Wr(0x38,1);
umin=8;
break;
case 9:
Lcd_4bit_Wr(0x39,1);
umin=9;
break;
case 0:
Lcd_4bit_Wr(0x30,1);
break;
umin=0;
}
}

if(tipo==4){
switch(tecla){
case 1:
Lcd_4bit_Wr(0x31,1);
dseg=1;
break;
case 2:
Lcd_4bit_Wr(0x32,1);
dseg=2;
break;
case 3:
Lcd_4bit_Wr(0x33,1);
dseg=3;
break;
case 4:
Lcd_4bit_Wr(0x34,1);
dseg=4;
break;
case 5:
Lcd_4bit_Wr(0x35,1);
dseg=5;
break;
case 6:
Lcd_4bit_Wr(0x36,1);
dseg=6;
break;
case 7:
Lcd_4bit_Wr(0x37,1);
dseg=7;
break;
case 8:
Lcd_4bit_Wr(0x38,1);
dseg=8;
break;
case 9:
Lcd_4bit_Wr(0x39,1);
dseg=9;
break;
case 0:
}
}

//y si se presiono un 4
//envie un cuatro
//presetee las unidades de minuto

//y si se presiono un 5
//envie un cinco
//presetee las unidades de minuto

//y si se presiono un 6
//envie un seis
//presetee las unidades de minuto

//y si se presiono un 7
//envie un siete
//presetee las unidades de minuto

//y si se presiono un 8
//envie un ocho
//presetee las unidades de minuto

//y si se presiono un 9
//envie un nueve
//presetee las unidades de minuto

//y si se presiono un 0
//envie un cero
//presetee las unidades de minuto

//si se están actualizando las decenas de segundo

//y si se presiono un 1
//envie un uno
//presetee las decenas de segundo

//y si se presiono un 2
//envíe un dos
//presetee las decenas de segundo

//y si se presiono un 3
//envie un tres
//presetee las decenas de segundo

//y si se presiono un 4
//envie un cuatro
//presetee las decenas de segundo

//y si se presiono un 5
//envie un cinco
//presetee las decenas de segundo

//y si se presiono un 6
//envie un seis
//presetee las decenas de segundo

//y si se presiono un 7
//envie un siete
//presetee las decenas de segundo

//y si se presiono un 8
//envie un ocho
//presetee las decenas de segundo

//y si se presiono un 9
//envie un nueve
//presetee las decenas de segundo

//y si se presiono un 0

```

```

        Lcd_4bit_Wr(0x30,1);           //envie un cero
        break;                         //presetee las decenas de segundo
        dseg=0;
    }
}

if(tipo==5){
    switch(tecla){
        case 1:                   //y si se presiono un 1
        Lcd_4bit_Wr(0x31,1);       //envie un uno
        useg=1;                   //presetee las unidades de segundo
        break;
        case 2:                   //y si se presiono un 2
        Lcd_4bit_Wr(0x32,1);       //envíe un dos
        useg=2;                   //presetee las unidades de segundo
        break;
        case 3:                   //y si se presiono un 3
        Lcd_4bit_Wr(0x33,1);       //envie un tres
        useg=3;                   //presetee las unidades de segundo
        break;
        case 4:                   //y si se presiono un 4
        Lcd_4bit_Wr(0x34,1);       //envie un cuatro
        useg=4;                   //presetee las unidades de segundo
        break;
        case 5:                   //y si se presiono un 5
        Lcd_4bit_Wr(0x35,1);       //envie un cinco
        useg=5;                   //presetee las unidades de segundo
        break;
        case 6:                   //y si se presiono un 6
        Lcd_4bit_Wr(0x36,1);       //envie un seis
        useg=6;                   //presetee las unidades de segundo
        break;
        case 7:                   //y si se presiono un 7
        Lcd_4bit_Wr(0x37,1);       //envie un siete
        useg=7;                   //presetee las unidades de segundo
        break;
        case 8:                   //y si se presiono un 8
        Lcd_4bit_Wr(0x38,1);       //envie un ocho
        useg=8;                   //presetee las unidades de segundo
        break;
        case 9:                   //y si se presiono un 9
        Lcd_4bit_Wr(0x39,1);       //envie un nueve
        useg=9;                   //presetee las unidades de segundo
        break;
        case 0:                   //y si se presiono un 0
        Lcd_4bit_Wr(0x30,1);       //envie un cero
        useg=0;                   //presetee las unidades de segundo
        break;
    }
}
}

/* Funcion de atencion a la interrupcion del KBI1 */
interrupt VectorNumber_Vkeyboard void ISR_KBI (void){
    columna=PTFD & 0x0F;           //capture el valor de la columna
    if(PTGD_PTGD0==0 & columna==0x0E){ //si detecto la fila 1 y la columna 1
        tecla=1;                   //marque tecla 1
    }
    if(PTGD_PTGD0==0 & columna==0x0D){ //si detecto la fila 1 y la columna 2
        tecla=2;                   //marque tecla 2
    }
    if(PTGD_PTGD0==0 & columna==0x0B){ //si detecto la fila 1 y la columna 3
        tecla=3;                   //marque tecla 3
    }
    if(PTGD_PTGD1==0 & columna==0x0E){ //si detecto la fila 2 y la columna 1
        tecla=4;                   //marque tecla 4
    }
    if(PTGD_PTGD1==0 & columna==0x0D){ //si detecto la fila 2 y la columna 2
        tecla=5;                   //marque tecla 5
    }
    if(PTGD_PTGD1==0 & columna==0x0B){ //si detecto la fila 2 y la columna 3

```

```

tecla=6;                                //marque tecla 6
}
if(PTGD_PTGD2==0 & columna==0x0E){
    tecla=7;                            //si detecto la fila 3 y la columna 1
}
if(PTGD_PTGD2==0 & columna==0x0D){
    tecla=8;                            //si detecto la fila 3 y la columna 2
}
if(PTGD_PTGD2==0 & columna==0x0B){
    tecla=9;                            //si detecto la fila 3 y la columna 3
}
if(PTGD_PTGD4==0 & columna==0x0D){
    tecla=0;                            //si detecto la fila 4 y la columna 2
}
}
KBIISC_KBACK = 1;                      //de reconocimiento de tecla presionada
bandera=1;                            //pone bandera de tecla presionada
}

/* Funcion de atencion a la interrupcion por RTC */
interrupt VectorNumber_Vrtc void ISR_RTC_OVF(void){
    RTCSC_RTIF = 1;                    //aclara bandera de sobreflujo del RTC
    useg++;                           //incremente segundos
    if (useg > 9){                  //si pasaron 10 segundos
        useg = 0;                     //aclare segundos
        dseg++;                      //incremente decenas de segundos
    }

    if (dseg > 5){                  //si pasaron 6 decenas de segundos
        dseg = 0;                     //aclare decenas de segundos
        umin++;                      //incremente unidades de minutos
    }

    if (umin > 9){                  //si pasaron 10 unidades de minutos
        umin = 0;                     //aclare unidades de minutos
        dmin++;                      //incremente decenas de minutos
    }

    if (dmin > 5){                  //si pasaron 6 decenas de minutos
        dmin = 0;                     //aclare decenas de minutos
        uhor++;                      //incremente unidades de horas
    }

    if (uhor > 9){                  //si pasaron 10 unidades de horas
        uhor = 0;                     //aclare unidades de horas
        dhor++;                      //incremente decenas de horas
    }

    if (dhor == 1 & uhor>2){       //si pasaron 12 horas
        uhor=0;                      //aclare unidades de horas
        dhor=0;                      //aclare decenas de horas
    }
}

```

El usuario podrá verificar la poca exactitud de este reloj, debido a la base de 1KHz interna que se usa. Para corregir este problema se recomienda tener una base de tiempo de 1ms en vez de un segundo y hacer ajustes por software, para compensar la imprecisión temporal. Otra forma sería usar las otras fuentes de reloj para el RTC, con más precisión temporal como son el reloj interno MCGOUT o un cristal externo de alta Q.

### **13.5. REFERENCIAS**

- MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual. Freescale Semiconductor. Rev 1, 2008.
- Microcontroller MCF51JM128 ColdFire Data Sheet. Freescale Semiconductor. Rev 0, 2008.

### **13.6. PREGUNTAS**

- Describa los modos en los que se puede utilizar el módulo TPM.
- Para una base de tiempo de 100 us, alimentando el contador principal, ¿qué período máximo podrá tener una señal cuadrada generada con el TPM en modo INPUT CAPTURE?
- Para la base de tiempo anterior, ¿qué frecuencia máxima de señal cuadrada de entrada se podrá medir en el TPM trabajando en modo INPUT CAPTURE?
- ¿De cuántas y cuáles maneras se puede generar un pulso PWM?
- ¿Qué ventajas y restricciones tiene el uso del reloj interno de 1KHz sobre el módulo RTC?
- ¿De qué manera se puede compensar el desfase causado por el uso del reloj interno de 1KHz en el módulo RTC?
- ¿Cuál sería la forma más recomendada para obtener un reloj de tiempo real confiable, usando el módulo RTC?

# CAPÍTULO 14

## Conversión A/D (ADC: *Analog to Digital Converter*)

- 14.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**
- 14.2. Programa ejemplo: Medida de temperatura utilizando un sensor LM35 y visualización en LCD**
- 14.3. Referencias**
- 14.4. Preguntas**

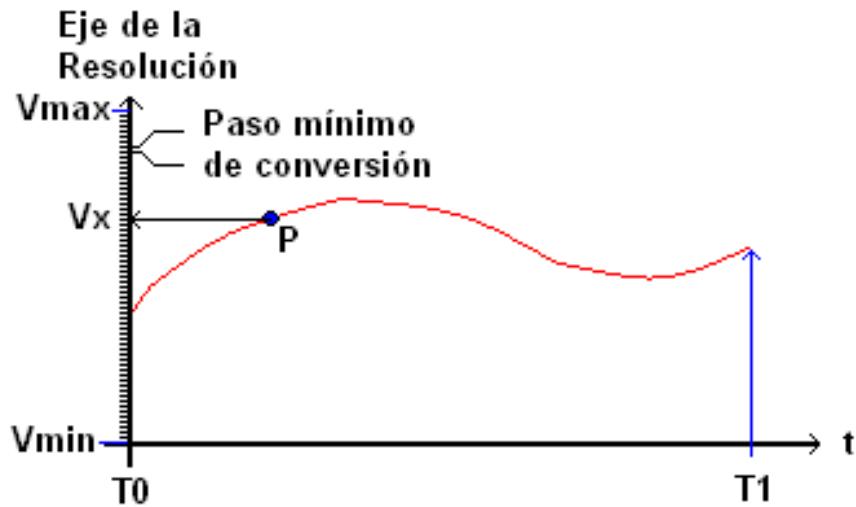
## 14.1. DIAGRAMA EN BLOQUES, EXPLICACIÓN DEL FUNCIONAMIENTO Y REGISTROS ASOCIADOS

- **Breve repaso de la conversión análoga a digital**

Antes de comenzar el estudio del módulo ADC de la máquina MCF51JM128, se hará un pequeño repaso sobre los conceptos relacionados con la conversión análoga a digital de una señal.

Las máquinas, como los microcontroladores, tratan de medir y/o convertir las variables análogas que el hombre manipula y entiende, pero el proceso de convertir introduce una inevitable pérdida de información. Esta pérdida es inherente al proceso de discretizar las señales análogas y continuas, que finalmente serán llevadas a cantidades binarias.

La Figura 16.1 representa una señal análoga continua entre los puntos  $t_0$  y  $t_1$ , que desde el punto de vista de la magnitud será sometida a un número discreto de valores binarios y que la cantidad de valores se conoce con el nombre de **Resolución** del sistema.



**Figura 14.1. Eje de la resolución en la conversión A/D.**

La **Resolución** de un conversor análogo a digital es la cantidad de valores, discretos, en los cuales se interpreta la señal a digitalizar. Por ejemplo, para un procesador con un conversor A/D que tiene una resolución de 12 bits el número de valores discretos en los cuales se puede valorar a una señal, sería de  $2^{12} = 4096$ . El valor ideal para esos valores sería un número infinito, pero tecnológicamente es imposible.

Esos valores deben estar comprendidos dentro de dos límites, que forman la ventana de conversión o valores de referencia ( $V_{min}$ ,  $V_{max}$ ). Para la Figura 14.1, el punto P tiene una interpretación en el mundo de lo discreto y es de  $V_x$ .

Se recomienda que el usuario aproveche al máximo la resolución del sistema, adecuando la señal análoga para que excursione de la manera más completa en la ventana de conversión. Por ejemplo, una señal con un valor máximo de 100mV deberá ser amplificada por un factor de 30, para una ventana de conversión de 3V y de ésta manera aprovechar la resolución del sistema.

Para calcular el paso mínimo de conversión y por otro lado conocer el intervalo de pérdida de información, supóngase que se tiene una señal sometida a un conversor de 12 bits de resolución, un  $V_{min} = 0V$  y un  $V_{max} = 5V$ . El paso mínimo de conversión está dado por:

$$\text{Paso mínimo} = (V_{max} - V_{min}) / \text{Resolución}$$

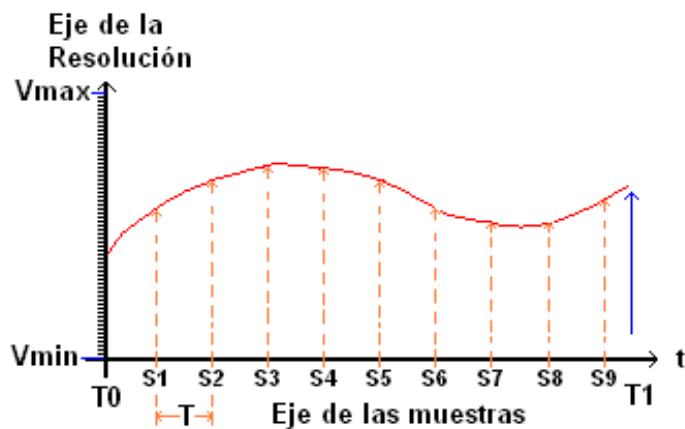
$$\text{Paso mínimo} = (5V - 0V) / 2^{12}$$

$$\text{Paso mínimo} = 1.22\text{mV}$$

El cálculo anterior indica que la diferencia en magnitud entre el resultado de una conversión y la inmediatamente superior (o inferior) es de 1.22mV. Todo valor que no sea múltiplo entero de un paso mínimo, se deberá aproximar al valor más cercano y es allí donde un conversor A/D ignora información del mundo análogo.

El **Muestreo** (*sample*) es otra característica importante de un conversor A/D y se refiere a la cantidad de muestras en la unidad de tiempo que se pueden procesar y convertir a cantidades discretas.

La Figura 14.2 presenta una señal análoga continua entre los puntos  $t_0$  y  $t_1$ , que desde el punto de vista del muestreo es sometida a un número finito de muestras y que cada muestra es tomada a un intervalo constante  $T$ , llamado período de muestreo.



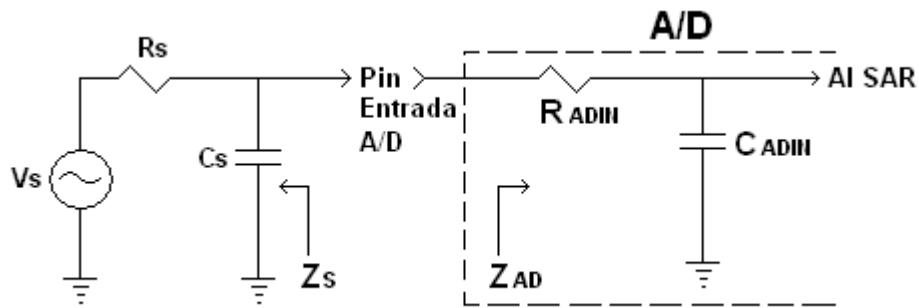
**Figura 14.2. Eje del muestreo en la conversión A/D.**

Al igual que en la resolución el muestreo introduce pérdida de información, debido a los valores que no son muestreados entre dos intervalos de muestreo contiguos.

Idealmente la tasa de muestreo debería ser infinita, pero existen restricciones tecnológicas. Entonces, mientras menor sea la separación entre los *Si* (*T* pequeño), mostrados en la Figura 14.6, más fiel será la señal digitalizada con respecto a la señal analógica original.

A continuación se describen otros errores inherentes a la conversión analógica a digital.

- **Error de muestreo (Sampling Error):** Este tipo de error está asociado al modelo de entrada del conversor A/D y tiene que ver con la impedancia y capacitancia características de entrada versus la impedancia característica de la señal a convertir. La Figura 14.3 muestra un modelo típico de impedancia de entrada de un conversor A/D.



**Figura 14.3. Modelo simplificado de entrada de un A/D.**

Por ejemplo: Para el conversor A/D de la máquina MCF51JM128, cuya impedancia típica es de  $R_{ADIN} = 7K$  y  $C_{ADIN} = 5.5pF$ , en donde muestras cercanas a  $1/4$  LSb<sup>34</sup> (*Less Significant bit*) con una resolución de 12 bits, pueden ser realizadas con la mayor velocidad de conversión de 3.5 ciclos de un reloj de 8MHz.

Lo anterior es posible siempre y cuando la impedancia característica ( $Z_s$ ) de la señal aplicada al pin A/D, permanezca cercana a un valor de  $2K\Omega$ , que sería el equivalente a la impedancia característica del A/D ( $Z_{AD}$ ).

Para impedancias más altas de la fuente a digitalizar, se recomienda disminuir la velocidad de conversión (incremento en el tiempo de muestreo).

**NOTA:** El usuario deberá siempre conocer las limitaciones de velocidad del conversor y aplicar el teorema de Nyquist-Shannon a la hora de digitalizar.

<sup>34</sup> Significa que para una resolución de 12 bits este error tendrá un peso del 0.024%.

- **Error por goteo del pin (Pin Leakage Error):** Es posible que se presente una corriente de goteo debida a una alta resistencia de la fuente análoga a digitalizar. El usuario deberá garantizar que  $R_s$  siempre esté por debajo de:

$$V_{DDAD} / (2^N \times I_{LEAK})$$

En donde:

**V<sub>DDAD</sub>:** Voltaje de alimentación del A/D

**N:** Resolución de la conversión A/D

**I<sub>LEAK</sub>:** Corriente de goteo (dato del manual)

Esta condición garantiza que el error por goteo estará por debajo de  $\frac{1}{4}$  LSB.

- **Error por ruido inducido (Noise Induced Error):** El ruido en una conversión análoga a digital ocurre durante el muestreo (*sample*) o el proceso de conversión. El ruido inducido es inevitable, pero puede ser minimizado y para esto se recomiendan las siguientes acciones:
  - Ubicar un condensador de 0.1uF, con baja ESR<sup>35</sup> entre V<sub>REFH</sub> y V<sub>REFL</sub>.
  - Ubicar un condensador de 0.1uF, con baja ESR entre V<sub>DDAD</sub> y V<sub>SSAD</sub>.
  - Si la fuente V<sub>DD</sub> usa un aislamiento inductivo, es necesario ubicar un condensador adicional de 1uF entre V<sub>DDAD</sub> y V<sub>SSAD</sub>.
  - Si V<sub>REFL</sub> se encuentra al nivel de V<sub>ss</sub>, es necesario garantizar una buena unión eléctrica.
  - No hacer commutaciones de pines I/O durante una conversión.

Existen situaciones en donde el ruido radiado o conducido (EMI) es intenso, para esto se hacen las siguientes recomendaciones:

- Ubicar un condensador de 0.01uF, con baja ESR a la entrada del canal de conversión, pero se debe tener en cuenta que esto altera la velocidad de muestreo.
  - Hacer algoritmos que promedien las muestras, para suavizar los cambios ruidosos.
  - Utilizar un reloj asíncrono como fuente del ADC y promediar las muestras.
- **Error de cuantización (Quantization Error):** Para una resolución de 12 bits, un paso de conversión (*code*) corresponde a una 1/4096 parte en la ventana de conversión. Este paso es el delta entre un código y el vecino inmediato, entonces la relación que define este paso sería:

$$1 \text{ LSb} = (V_{REFH} - V_{REFL}) / 2^N$$

En esta relación se encuentra un error inherente de cuantización, debido al resultado de la digitalización. Por ejemplo para un ADC con un error de  $\pm\frac{1}{2}$  LSb

---

<sup>35</sup> El término corresponde a la resistencia serie equivalente.

en una resolución de 10 bits y con una ventana de conversión de 5V, el error asociado a un paso de conversión sería de:

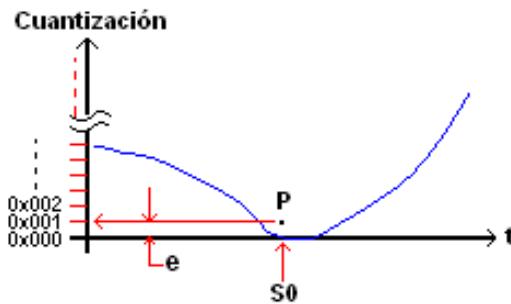
$$1 \text{ LSb} = 5\text{V}/1024$$

$$1 \text{ LSb} = 4.88\text{mV}$$

De donde:

$$\text{Error de cuantización} = \pm 2.44\text{mV}$$

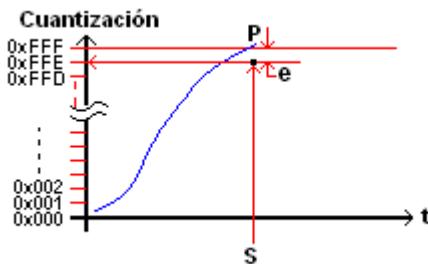
- **Error de escala cero (Zero Scale Error =Offset Error):** Está definido como la diferencia entre el ancho del paso actual de conversión y el paso de conversión teórico, esto para el primer paso de conversión. La Figura 14.4 ilustra el fenómeno de error de escala cero.



**Figura 14.4. Error por escala de cero.**

La muestra tomada en el punto **P** debería arrojar un valor de cuantización de 0x000, para la muestra tomada en **S0** (valor teórico). Como se puede apreciar el resultado real fue de 0x001, es decir un error de cero (**e**) de 1LSb.

- **Error de escala plena (Full Scale Error):** Está definido como la diferencia entre el ancho del paso actual de conversión y el paso de conversión teórico, esto para el último paso de conversión. La Figura 14.5 ilustra el fenómeno de error de escala plena.



**Figura 14.5. Error por escala plena.**

La muestra tomada en el punto **P** debería arrojar un valor de cuantización de 0xFFFF, para la muestra tomada en **S** (valor teórico). Como se puede apreciar el resultado real fue de 0xFFE, es decir un error de escala plena (**e**) de 1LSb.

- **Error diferencial no lineal (Differential non Linearity Error):** Está definido como la peor diferencia entre el valor teórico y el valor actual, de un paso de conversión. Esto aplica para toda conversión en todo el rango de la escala de conversión.
- **Error integral no lineal (Integral non Linearity Error):** Está definido como el más alto valor (valor absoluto) de la suma de los errores diferenciales acumulados.
- **Error por parpadeo en la conversión (Code Jitter Error):** Se presenta cuando en ciertos puntos, un voltaje de entrada determinado es convertido de uno a dos valores, durante un muestreo repetitivo.

Idealmente, cuando un voltaje de entrada es infinitesimalmente pequeño respecto de un voltaje de transición (paso mínimo de conversión), el convertidor arrojará el código más pequeño.

Es usual que pequeñas cantidades de ruido en el sistema, puedan causar que el convertidor se indetermine (valor equivocado entre dos códigos consecutivos). El error típico de *jitter* es de  $\pm\frac{1}{2}$  LSb.

Para minimizar el error por *jitter* se recomienda hacer un promedio de las medidas.

- **Error por no monotonicidad (Non Monotonicity Error):** Se presenta cuando el conversor arroja un código bajo ante un voltaje alto de entrada.

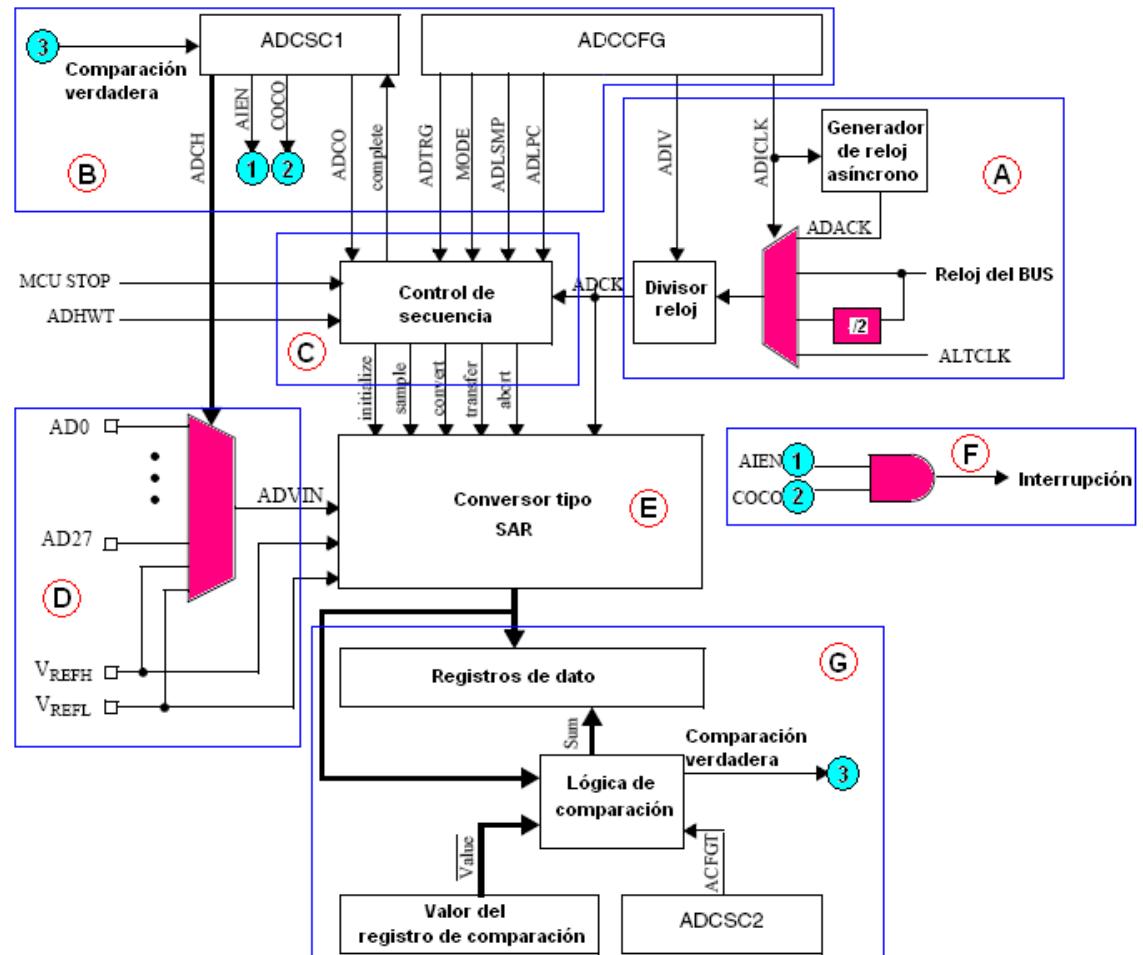
- **Breve descripción del módulo ADC y diagrama en bloques**

Algunas de las características más importantes del módulo ADC son:

- Técnica de conversión por aproximaciones sucesivas hasta 12 bits de resolución.
- Hasta 28 entradas análogas.
- Conversión programable para 8, 10 y 12 bits, justificada a la derecha y no signada.
- Modo de conversión simple y continua.
- Tiempo de conversión programable y modo de reducción de consumo.
- Evento de finalización de una conversión y generación de evento de interrupción al finalizar la conversión.
- Hasta cuatro fuentes de reloj de conversión.
- Operación en modo WAIT y STOP 3, para reducción de ruido.

- Posibilidad de selección de reloj asincrónico, para operación en bajo ruido.
- Posibilidad de elegir una señal de disparo por hardware (*trigger*), para iniciar conversión.
- Comparación del valor convertido contra un valor programado, para mayor que, igual que o menor que.

La Figura 14.6 muestra el diagrama en bloques del módulo ADC, en donde el circuito (A) corresponde al sistema de reloj del ADC. Este circuito tiene la posibilidad de seleccionar cuatro fuentes, como: Reloj asíncrono (ADACK), reloj del BUS, reloj del BUS dividido por dos y un reloj alterno (ALTCLK).



**Figura 14.6. Diagrama en bloques del ADC.**

El circuito (B) corresponde a los registros de configuración y control del ADC. El circuito (C) sincroniza toda la operación de conversión, actuando como una gran máquina de estados. El circuito (D) multiplexa las diferentes entradas analógicas (canales A/D) que puede atender el sistema ADC y establece la ventana de conversión.

El circuito (E) es el corazón del módulo ADC y corresponde al SAR (*Successive Approximation Register*) de la conversión. El circuito (F) configura la lógica de interrupción del módulo ADC y finalmente, el circuito (G) establece las componentes para la comparación del valor convertido contra un valor programado.

- **Registros asociados al módulo ADC**

- **Registro de estado y control 1 (ADCSC1):** La Figura 14.7 muestra la configuración del registro de estado y control 1 de módulo ADC. El evento de escribir en este registro hace que se aborde una conversión en proceso, inicializando una nueva.

**Registro de estado y control 1 del ADC(ADCSC1): (FFFF8010)**

	7	6	5	4	3	2	1	0
Lectura	COCO							
Escritura		AIEN	ADCO			ADCH		
Reset:	0	0	0	1	1	1	1	1

**Figura 14.7. Registro ADCSC1.**

- **COCO:** Bandera de conversión completa. Cuando COCO es “1” y AIEN = “1”, entonces se genera un evento de interrupción.
  - 0:** No se ha completado una conversión
  - 1:** Se ha completado una conversión
- **AIEN:** Bit para habilitar un evento de interrupción por la finalización de una conversión análoga a digital.
  - 0:** Inhibe evento de interrupción por conversión completa
  - 1:** Habilita evento de interrupción por conversión completa
- **ADCO:** Bit para seleccionar el modo de conversión.
  - 0:** Habilita conversión simple. Una sola conversión es iniciada cuando se escribe sobre el registro ADCSC1 o cuando se trabaja con señal de disparo externa (*external trigger*).
  - 1:** Habilita conversión continua. La conversión es iniciada cuando se escribe sobre el registro ADCSC1 o cuando se trabaja con señal de disparo externa (*external trigger*).
- ADCH:** Bits para seleccionar el canal a convertir (multiplexor). En la Tabla 14.1 se detalla la combinación de bits según el canal.

**Tabla 14.1. Selección del canal ADC.**

ADCH	Selección Canal
00000–01111	AD0–15
10000–11011	AD16–27
11100	Reservado
11101	$V_{REFH}$
11110	$V_{REFL}$
11111	Módulo Inhibido

- **Registro de estado y control 2 (ADCSC2):** La Figura 14.8 muestra la configuración del registro de estado y control 2 de módulo ADC.

#### Registro de estado y control 2 del ADC (ADCSC2): (FFFF8011)

Lectura	7	6	5	4	3	2	1	0
Lectura	ADACT	ADTRG	ACFE	ACFGT	0	0	R <sup>1</sup>	R <sup>1</sup>
Escritura							0	0

Reset: 0 0 0 0 0 0 0 0 0

R<sup>1</sup>: Estos bits se deben conservar siempre en "0"

**Figura 14.8. Registro ADCSC2.**

- **ADACT:** Bit que indica cuando una conversión está en progreso. Este bit es puesto a “0” una vez se haya finalizado la conversión o se aborte el proceso.  
**0:** No hay conversión en proceso  
**1:** Hay una conversión en proceso
- **ADTRG:** Bit para seleccionar el tipo de señal que inicia una conversión. Cuando se elige una conversión por *hardware*, el pin ADCHWT inicializa una conversión (esta opción viene implementada en ciertos modelos del MCF51JM128). Cuando se elige la conversión por *software*, la simple escritura sobre el registro ADCSC1 inicializa un conversión.  
**0:** Disparo de conversión por *software*  
**1:** Disparo de conversión por *hardware*
- **ACFE:** Bit para habilitar la función de comparación del módulo ADC.  
**0:** Deshabilita modo de comparación  
**1:** Habilita modo de comparación
- **ACFGT:** Bit para habilitar la comparación si mayor que.  
**0:** Función de comparación si menor que  
**1:** Función de comparación si mayor que

- **Registro resultado alto de la conversión (ADCRH):** La Figura 14.9 muestra el registro del resultado alto de la conversión A/D, que contiene el *nibble* de mayor peso en una conversión de 12 y 10 bits.

**Registro del resultado alto de la conversión (ADCRH): (FFFF8012)**

	7	6	5	4	3	2	1	0
Lectura	0	0	0	0	ADR11	ADR10	ADR9	ADR8
Escritura								
Reset:	0	0	0	0	0	0	0	0

**Figura 14.9. Registro ADCRH.**

- **Registro resultado bajo de la conversión (ADCRL):** La Figura 14.10 muestra el registro del resultado bajo de la conversión A/D, que contiene el *byte* de menor peso en una conversión de 12 y 10 bits; y contiene los ocho bits de una conversión en 8 bits.

**Registro resultado bajo de la conversión (ADCRL): (FFFF8013)**

	7	6	5	4	3	2	1	0
Lectura	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0
Escritura								
Reset:	0	0	0	0	0	0	0	0

**Figura 14.10. Registro ADCRL.**

- **Registro alto de comparación (ADCCVH):** La Figura 14.11 muestra el registro alto de la comparación del A/D, que contiene el *nibble* de mayor peso en una comparación de 12 y 10 bits. El valor de este registro se compara con el registro ADCRH.

**Registro alto de comparación (ADCCVH): (FFFF8014)**

	7	6	5	4	3	2	1	0
Lectura	0	0	0	0	ADCV11	ADCV10	ADCV9	ADCV8
Escritura								
Reset:	0	0	0	0	0	0	0	0

**Figura 14.11. Registro ADCCVH.**

- **Registro bajo de comparación (ADCCVL):** La Figura 14.12 muestra el registro bajo de la comparación del A/D, que contiene el *byte* de menor peso en una conversión de 12 y 10 bits; y contiene los ocho bits de una conversión en 8 bits. El valor de este registro se compara con el registro ADCRL.

### Registro bajo de comparación (ADCCVL): (FFFF8015)

	7	6	5	4	3	2	1	0
Lectura Escritura	ADCV7	ADCV6	ADCV5	ADCV4	ADCV3	ADCV2	ADCV1	ADCV0
Reset:	0	0	0	0	0	0	0	0

Figura 14.12. Registro ADCCVL.

- **Registro de configuración (ADCCFG):** La Figura 14.13 muestra el registro de configuración del módulo ADC.

### Registro de configuración del ADC (ADCCFG): (FFFF8016)

	7	6	5	4	3	2	1	0
Lectura Escritura	ADLPC	ADIV	ADLSMP	MODE				
Reset:	0	0	0	0	0	0	0	0

Figura 14.13. Registro ADCCFG.

- **ADLPC:** Bit para configurar el ADC en bajo consumo. Optimiza el SAR para bajo consumo y reloj lento, esta opción es adecuada para digitalizar señales de baja velocidad.  
**0:** No habilita modo de bajo consumo  
**1:** Habilita modo de bajo consumo
- **ADIV:** Bits para configurar el divisor del reloj elegido para el módulo ADC.  
**00:** Divisor por 1  
**01:** Divisor por 2  
**10:** Divisor por 4  
**11:** Divisor por 8
- **ADLSMP:** Bit para disponer el ADC en conversiones rápidas, asociadas a entradas de baja impedancia y consumos altos o para conversiones lentas de impedancias altas y bajo consumo.  
**0:** Tiempo corto para muestreo  
**1:** Tiempo largo para muestreo
- **MODE:** Bits para seleccionar la resolución de trabajo del ADC.  
**00:** Conversión con resolución en 8 bits  
**01:** Conversión con resolución en 12 bits  
**10:** Conversión con resolución en 10 bits  
**11:** Reservado
- **ADICLK:** Bits para seleccionar el reloj que alimentará el módulo ADC.  
**00:** Reloj del BUS interno

- 01:** Reloj del BUS interno dividido por 2
- 10:** Reloj alterno ALTCLK
- 11:** Reloj asíncrono ADACK

- **Registro de control de pin del ADC o habilitación de canal A/D (APCTLx):**  
La Figura 14.14 muestra los registros de control de pines del ADC.

0x(FF)FF_8017	APCTL1	ADPC7	ADPC6	ADPC5	ADPC4	ADPC3	ADPC2	ADPC1	ADPC0
0x(FF)FF_8018	APCTL2	—	—	—	—	ADPC11	ADPC10	ADPC9	ADPC8

**Figura 14.14. Registros APCTLx.**

Todo canal que se desee habilitar necesita ser confirmado escribiendo un “1” en el respectivo bit ADPCx

- **Cálculo del tiempo de una conversión**

Para efectos del cálculo del tiempo que toma una conversión en ejecutarse, la Tabla 14.2 relaciona el tiempo dependiendo del reloj elegido (ADCLK = ADICLK/ADIV) y el bit ADSMP (tiempocorto o largo).

**Tabla 14.2. Tiempos de conversión.**

Tipo de Conversión	ADICLK	ADLSMP	Máximo tiempo de conversión
Simple o primera en modo continuo (8bits)	0x, 10	0	20 ADCK ciclos + 5 ciclos reloj del BUS
Simple o primera en modo continuo (10bits y 12 bits)	0x, 10	0	23 ADCK ciclos + 5 ciclos reloj del BUS
Simple o primera en modo continuo (8bits)	0x, 10	1	40 ADCK ciclos + 5 ciclos reloj del BUS
Simple o primera en modo continuo (10bits y 12 bits)	0x, 10	1	43 ADCK ciclos + 5 ciclos reloj del BUS
Simple o primera en modo continuo (8bits)	11	0	5 µs + 20 ADCK + 5 ciclos reloj del BUS
Simple o primera en modo continuo (10bits y 12 bits)	11	0	5 µs + 23 ADCK + 5 ciclos reloj del BUS
Simple o primera en modo continuo (8bits)	11	1	5 µs + 40 ADCK + 5 ciclos reloj del BUS
Simple o primera en modo continuo (10bits y 12 bits)	11	1	5 µs + 43 ADCK + 5 ciclos reloj del BUS
Las siguientes en modo continuo (8 bits) $f_{BUS} \geq f_{ADCK}$	xx	0	17 ADCK ciclos
Las siguientes en modo continuo (10bits y 12 bits) $f_{BUS} \geq f_{ADCK}$	xx	0	20 ADCK ciclos
Las siguientes en modo continuo (8 bits) $f_{BUS} \geq f_{ADCK}/11$	xx	1	37 ADCK ciclos
Las siguientes en modo continuo (10bits y 12 bits) $f_{BUS} \geq f_{ADCK}/11$	xx	1	40 ADCK ciclos

A manera de ejemplo, supóngase que se está realizando una conversión A/D con resolución de 12 bits. Para la conversión se ha elegido el reloj de bus interno, que es de 16MHz. La conversión se realizará en modo continuo y tiempos cortos de muestreo.

Como se vió en apartes anteriores el máximo reloj de conversión para el ADC del MCF51JM128 es de 8MHz, entonces es necesario aplicar un valor de ADIV = 01 (divisor por 2). El valor resultante conformaría el ADCK de la conversión, de tal

manera que el tiempo total de conversión, para las muestras subsecuentes se calcula así:

$$\text{Tiempo de conversión} = 20 \text{ ADCK ciclos} / (16\text{MHZ} / 2)$$

$$\text{Tiempo de conversión} = 2.5 \times 10^{-6} \text{ seg}$$

## 14.2. EJEMPLO CON EL MÓDULO ADC

Para la ilustración de la operación del módulo ADC se trabajará sobre el clásico ejemplo de la medida de temperatura con un transductor de bajo costo, como lo es el LM35. El problema consiste en la medida en grados Celsius de la temperatura ambiente y visualización de la misma en un LCD.

El fenómeno propone un sistema de inercia grande, de tal manera que se trabajará el módulo ADC en bajo consumo y tiempos largos de muestreo. La Figura 14.15 ilustra el circuito sugerido para el ejercicio.

**NOTA:** Se recomienda al usuario revisar cuidadosamente el estado de los puentes del DEMOJM, con el objetivo de hacer compatible el ejercicio actual y de esta manera liberar los pines comprometidos en el circuito.

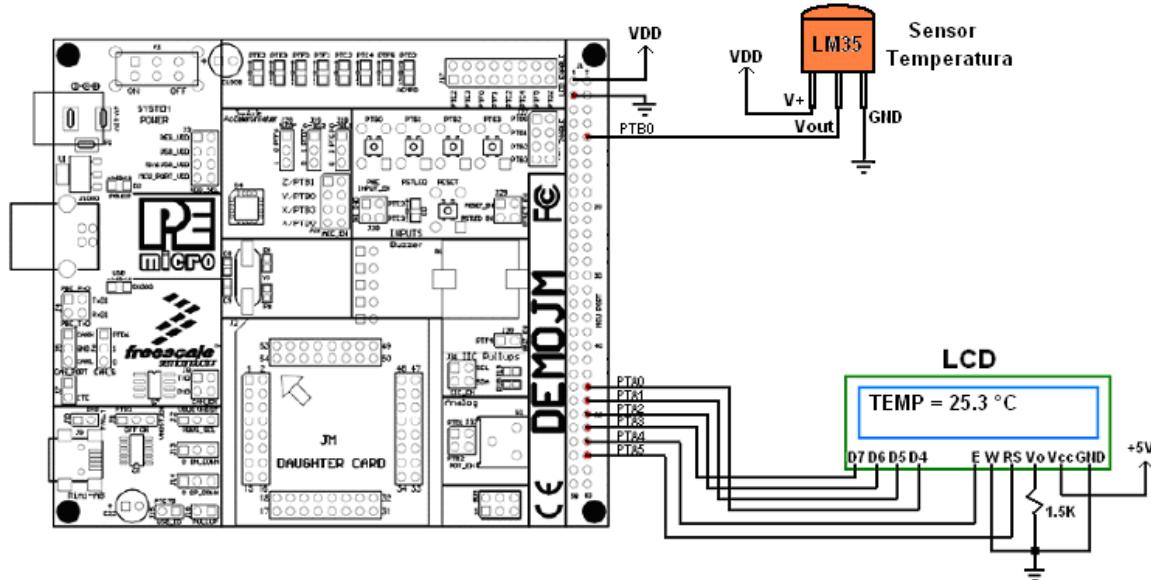


Figura 14.15. Circuito medida de temperatura.

La función de transferencia del LM35 es:

$$f(T) = 10\text{mV} / ^\circ\text{C}$$

De tal manera que si el ambiente se encuentra a una temperatura de 25°C, a la salida del sensor se tendría un voltaje de 250mV.

Las referencias de conversión VREFH y VREFL del sistema DEMOJM, están alimentadas a 4.64 V. Si se trabaja con una resolución de 12 bits, un paso de conversión sería de  $4.64V / 4096 = 1.132$  mV. Este valor sería bastante apropiado para el sistema, porque la variación de un grado centígrado en el sensor es de 10mV.

Un programa en C que resuelve el ejercicio de la temperatura planteado, se detalla a continuación.

```
/*
 * ADC_TEMPERATURA: Ejemplo sobre la utilizacion del modulo ADC
 */
/* main.c
 */
/*
 * Fecha: Mayo 28, 2008
 */
/*
 * V1.0, Rev 0 por Diego Múnera
 */
/*
 * Asunto: Implementar un codigo en C que mida la temperatura en °C, entregada por un sensor
 * LM35. La temperatura debe ser visualizada en un LCD y debe tener por lo menos una
 * cifra decimal.
 */
/*
 * Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros
 * editores.
 */
/*
 * DISTRIBUCION DE LOS PINES DE PUERTOS
 */

/*
 * PUERTO      * PIN      * I/O      * TIPO      * INICIA      * COMENTARIO
 */
/*PTA      * -      * -      * -      * -      *LCD
/*      *PTA0      * O      * D      * 1      *D4 del LCD
/*      *PTA1      * O      * D      * 1      *D5 del LCD
/*      *PTA2      * O      * D      * 1      *D6 del LCD
/*      *PTA3      * O      * D      * 1      *D7 del LCD
/*      *PTA4      * O      * D      * 1      *Señal ENABLE del LCD
/*      *PTA5      * O      * D      * 1      *Señal RS del LCD
/*PTB      * -      * -      * -      *      *Medida de la temperatura
/*      *PTB0      * I      * A      * -      *Entrada señal temperatura del sensor
/*PTC      * -      * -      * -      * -      *No usado
/*PTD      * -      * -      * -      * -      *No usado
/*PTE      * -      * -      * -      * -      *No usado
/*PTF      * -      * -      * -      * -      *No usado
/*PTG      * -      * -      * -      * -      *No usado
/*PTH      * -      * -      * -      * -      *No usado
/*PTJ      * -      * -      * -      * -      *No usado
/*
 * Archivos de cabecera */
#include <hidef.h>          //macro para interrupciones
```

```

#include "derivative.h"           //macro de declaraciones del MCU

/* Declaracion de variables y constantes */
byte bandera=0;                  //bandera de proposito general
word dato_ADC=0;                 //parametro para hacer retardo
unsigned long retardo=0;          //variable de iteracion
char i=0;                         //byte de inicializacion del registro PTAD
char PTAD_Config=0;               //byte de inicializacion del registro PTADD
char PTADD_Config=0;              //byte de inicializacion del registro PTAPE
char PTAPE_Config=0;              //byte de inicializacion del registro PTASE
char PTASE_Config=0;              //byte de inicializacion del registro PTADS
char PTADS_Config=0;              //byte de inicializacion del registro PTAIFE
char ADCS1_Config=0;              //byte de inicializacion del registro ADCCFG
char ADCCFG_Config=0;             //byte de inicializacion del registro APCTL1
char APCTL1_Config=0;              //byte de inicializacion del registro ADCSC2
char ADCSC2_Config=0;              //variables para conversion de la temperatura a ASCII
char dec_ad,uni_ad,dem_ad,cem_ad; //variables para conversion de la temperatura a ASCII

char comando8=0;                 //parametro inicializacion LCD modo 8 bits
char dato4;                       //dato a enviar al LCD
char rs=0;                         //señal de dato o comando al LCD
char a=0;                          //variable temporal
const unsigned char mensaje[] = "TEMP = . C"; //Arreglo del mensaje para llevar al LCD
char *pM;                          //puntero a mensaje para el LCD

/* Macros y definiciones */
#define Disable_COP() SOPT1 &= 0x3F //deshabilita el COP

/* Declaracion de funciones */
void Delay(unsigned long retardo);   //funcion para hacer reatrdos varios
void LCD_Init(void);                //funcion para inicializar LCD
void Lcd_4bit_Wr(char dato4,char rs); //funcion para escribir dato al LCD
void Comando_8bit(char comando8);    //funcion para inicio del LCD a 8 bits
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config); //declare funcion que inicializa PTA
void ADC_Init(char ADCCFG_Config,char ADCSC2_Config,char APCTL1_Config,char ADCSC1_Config); //declare funcion para inicializar ADC

/* main(): Funcion principal */
void main(void) {
    MCGC1=0x04;                      //reloj en modo FEI y divisor por 1
    MCGC2=0x00;                        //rango alto
    MCGC4=0x22;                        //deshabilita el COP
    Disable_COP();                     //pines inician en "0",pines como salida, no pullups,
    PTA_Init(0x00,0xFF,0x00,0xFF,0x00,0xFF); //si slew, no strength y si filtro
    LCD_Init();                         //inicialice LCD
    pM =(char *)mensaje;                //puntero toma direccion del mensaje con definicion
    for (i=0;i<13;i++){                //ciclo para imprimir 15 caracteres del mensaje
        Lcd_4bit_Wr(*pM,1);            //envia caracter al LCD
        pM++;                           //incrementa puntero al arreglo del mensaje
    }
    ADC_Init(0x97,0x00,0x01,0x40);     //habilita interr, ADC ON, conversion simple, bajo consumo
                                         //divisor x 1, muestreo largo, 12 bits resolucion, canal 0
                                         //reloj ADACK
    EnableInterrupts();                //habilita interrupciones

    for(;;){                           //iteracion para presentacion de la temperatura (for infinito)
        if (bandera==1){
            ADCSC1 = 0x00;              //inhibe interrupcion por A/D
            bandera=0;                  //aclara bandera
            dato_ADC = dato_ADC*113;    //ajuste resultado (1LSb = 0.001132V), escalado por 10000
            dec_ad = dato_ADC/10000;     //halle decenas de grados C
            uni_ad = (dato_ADC - dec_ad*10000)/1000; //halle unidades de grados C
            dem_ad = (dato_ADC - dec_ad*10000-uni_ad*1000)/100; //halle decimas de grados C
            cem_ad = (dato_ADC - dec_ad*10000-uni_ad*1000-dem_ad*100)/10; //halle centesimas de grados C
            Lcd_4bit_Wr(0x8B,0);         //prepara para enviar centesimas de temperatura
            Lcd_4bit_Wr(cem_ad+0x30,1);  //envia las centesimas de temperatura
            Lcd_4bit_Wr(0x8A,0);         //prepara para enviar decimas de temperatura
        }
    }
}

```

```

Lcd_4bit_Wr(dem_ad+0x30,1);           //envía las decimas de temperatura
Lcd_4bit_Wr(0x88,0);                 //prepare para enviar unidades de temperatura
Lcd_4bit_Wr(uni_ad+0x30,1);          //envia las unidades de temperatura
Lcd_4bit_Wr(0x87,0);                 //prepare para enviar decenas de temperatura
Lcd_4bit_Wr(dec_ad+0x30,1);          //envia las decenas de temperatura
Delay(5000000);                      //retardo entre muestras
ADCSC1 = 0x40;                       //habilita otra conversion por A/D
}
}
}
/* es necesario asegurarse de nunca abandonar el main */

/* Funcion para inicializar el puerto A */
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config){
    PTAD=PTAD_Config;                  //inicialice estado pinos PTA
    PTADD=PTADD_Config;                //inicialice dirección de pinos PTA
    PTAPE=PTAPE_Config;                //inicialice estado de pullups PTA
    PTASE=PTASE_Config;                //inicialice estado del slew rate PTA
    PTADS=PTADS_Config;                //inicialice estado de drive strength PTA
    PTAIFE=PTAIFE_Config;              //inicialice estado del filtro PTA
}

/*Funcion para inicializacion del ADC */
void ADC_Init(char ADCCFG_Config,char ADCSC2_Config,char APCTL1_Config,char ADCSC1_Config){
    ADCCFG=ADCCFG_Config;             //inicializa registro ADCCFG del ADC
    ADCSC2=ADCSC2_Config;             //inicializa registro ADCSC2 del ADC
    APCTL1=APCTL1_Config;              //inicializa registro APCTL1 del ADC
    ADCSC1=ADCSC1_Config;              //inicializa registro ADCSC1 del ADC
}

/* Funcion para inicializacion del LCD a 4 bits */
void LCD_Init(void){
    Delay(200000);                   //hace retardo
    Comando_8bit(0x30);              //llama función enviar comando en 8 bits con comando
    Delay(200000);                   //hace retardo
    Comando_8bit(0x30);              //llama función enviar comando en 8 bits con comando
    Delay(200000);                   //hace retardo
    Comando_8bit(0x20);              //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x28,0);             //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x0C,0);              //ON LCD, OFF cursor
    Lcd_4bit_Wr(0x01,0);              //borrar pantalla LCD
    Lcd_4bit_Wr(0x06,0);              //desplaza cursor a la derecha con cada caracter
    Lcd_4bit_Wr(0x80,0);              //primer caracter fila 1 columna 1
}

/* Funcion para escribir a 4 bits en el LCD */
void Lcd_4bit_Wr(char dato4,char rs){
    if (rs==0){                      //pregunte si se va a enviar un comando o un dato
        PTAD_PTAD5=0;                //prepara envío de comando RS = 0
    }
    else{
        PTAD_PTAD5=1;                //prepara envio de dato RS = 1
    }
    a=dato4;                         //almacena temporalmente el dato
    a=>>4;                          //desplaza parte alta dato para la baja
    PTAD&=0xF0;                      //enmascara parte alta del puerto A
    a&=0x0F;                          //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;                    //sube pin de enable
    Delay(20000);                   //hace retardo
    PTAD|=a;                          //envía nibble alto del dato
    Delay(20000);                   //hace retardo
    PTAD_PTAD4=0;                    //baja pin de enable
    Delay(20000);                   //hace retardo
    PTAD&=0xF0;                      //enmascara parte alta del puerto A
    dato4&=0x0F;                      //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;                    //sube pin de enable
    Delay(20000);                   //hace retardo
    PTAD|=dato4;                     //envía nibble bajo del dato
    Delay(20000);                   //hace retardo
}

```

```

PTAD_PTAD4=0;           //baja pin de enable
Delay(20000);           //hace retardo
}

/* Función para envio de comando en 8 bits al LCD (solo para inicio) */
void Comando_8bit(char comando8){
    PTAD_PTAD5=0;           //prepara envio de comando
    PTAD&=0xF0;             //enmascara parte baja del puerto A
    comando8&=0xF0;          //enmascara parte alta de comando a enviar al LCD
    comando8=comando8>>4;   //desplaza para tomar nibble alto
    PTAD_PTAD4=1;           //sube pin de enable
    Delay(20000);           //hace retardo
    PTAD|=comando8;          //envía comando
    Delay(20000);           //hace retardo
    PTAD_PTAD4=0;           //sube pin de enable
    Delay(20000);           //hace retardo
}

/* Funcion Delay(): Retarda basado en una variable tipo entero */
void Delay(unsigned long retardo){
    while(retardo>0){        //llego a cero?
        retardo--;            //no --> decrementa
    }
}

/* Funcion de atencion a la interrupcion del ADC canal 0 */
interrupt VectorNumber_Vadc void ISR_ADC (void){
    //ADCSC1 = 0x40;          //reconozca interrupción e inicie otra
    dato_ADC=ADCR;           //capture dato del conversor
    bandera=1;                //ponga bandera en "1"
}

```

### **14.3. REFERENCIAS**

- MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual. Freescale Semiconductor. Rev 1, 2008.
- Microcontroller MCF51JM128 ColdFire Data Sheet. Freescale Semiconductor. Rev 0, 2008.

### **14.4. PREGUNTAS**

- ¿Qué es resolución en un conversor análogo digital?
- La señal que entrega cierto sensor es de  $2\text{mV}^{\circ}\text{C}$ , ¿cuál sería la recomendación para digitalizarla usando al módulo ADC?
- ¿Cuál sería el paso mínimo de conversión para una señal cuyos valores excusionan entre 1V y 4V, trabajando con el módulo ADC con voltajes de referencia entre 0V y 5V a 10 bits?
- ¿Cuál es la recomendación desde el punto de vista de la tasa de muestreo para señales con impedancia mayor a la impedancia de entrada del módulo ADC?
- Realice un listado de las recomendaciones para mitigar el error por ruido inducido en un ADC.
- Para un ADC con un error de  $\pm 1/4 \text{ LSB}$  en una resolución de 12 bits y con una ventana de conversión de 3.3V ¿cuál es el error asociado a un paso de conversión?
- ¿Qué diferencia hay entre el modo de conversión simple y continua?

# CAPÍTULO 15

## Comparación Análoga (ACMP: *Analog Comparator*)

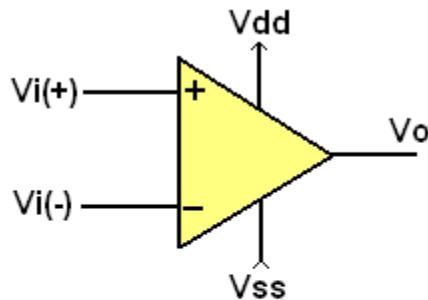
- 15.1. Diagrama en bloques, explicación del funcionamiento y registros asociados**
- 15.2. Programa ejemplo: Monitoreo del voltaje en una celda NiMH y control sobre la carga**
- 15.3. Referencias**
- 15.4 Ejercicios y preguntas**

## 15.1. DIAGRAMA EN BLOQUES, EXPLICACIÓN DEL FUNCIONAMIENTO Y REGISTROS ASOCIADOS

- **Breve repaso de la comparación análoga**

Antes de comenzar el estudio del módulo ACMP de la máquina MCF51JM128, se hará un breve repaso del concepto de comparación análoga.

La Figura 15.1 ilustra un comparador análogo. Para este circuito, si la señal  $V_{i(+)}$  es mayor o igual a la señal  $V_{i(-)}$ , la salida  $V_o$  será aproximadamente igual a  $V_{dd}$ . Pero si la señal  $V_{i(+)}$  es menor a  $V_{i(-)}$ , la salida  $V_o$  será aproximadamente igual a  $V_{ss}$ .



$$\boxed{\begin{aligned}V_{i(+)} &\geq V_{i(-)} \rightarrow V_o = V_{dd} \\V_{i(+)} &< V_{i(-)} \rightarrow V_o = V_{ss}\end{aligned}}$$

Figura 15.1. Comparador análogo.

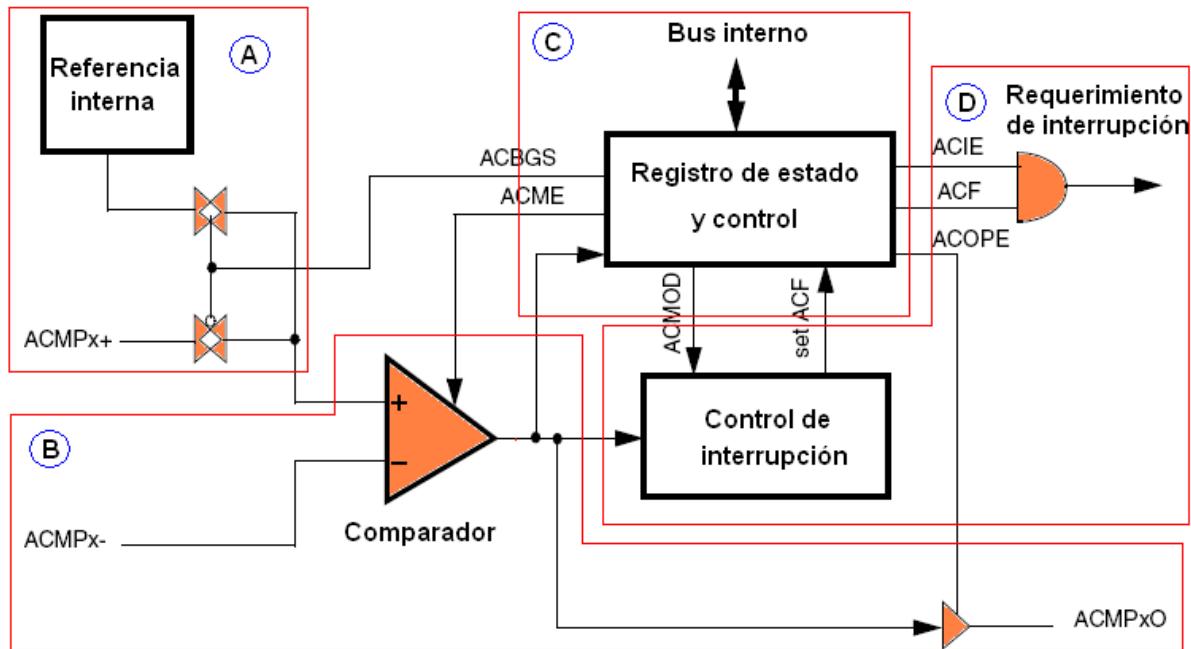
- **Breve descripción del módulo ACMP y diagrama en bloques**

Algunas de las características más importantes del módulo ACMP son:

- Hasta dos comparadores, ACMP1 y ACMP2.
- Plena excursión sobre la fuente de alimentación.
- Generación de evento de interrupción ante un flanco de subida o bajada en la salida del comparador.
- Comparación contra una referencia interna (bandgap = 1.2V típico) o una señal externa.
- Posibilidad de presentar el estado de la salida del comparador en un pin del MCU.

La Figura 15.2 muestra el diagrama en bloques del módulo ACMP, en donde el circuito (A) corresponde al sistema de referencia de comparación, que puede ser interna o una señal externa (ACMPx+). El circuito (B) es el corazón del módulo y corresponde al comparador con su señal de entrada (ACMPx-), la señal de habilitación del módulo (ACME) y la salida ACMPOx. El registro de estado y control

del módulo ACMP, corresponde la circuito (C). Finalmente, el circuito (D) configura la máquina de interrupción del módulo, con la bandera de interrupción (ACF), el bit de habilitación de interrupción (ACIE) y la selección del flanco (ACOPE).



**Figura 15.2. Diagrama en bloques el módulo ACMP.**

- **Registros asociados al módulo ACMP**

**Registro de estado y control (ACMPSC):** La Figura 15.3 muestra el único registro asociado al módulo ACMP y corresponde al registro de estado y control.

**Registro de estado y control del ACMP (ACMPxSC) : (FFFF800E)**

Lectura	7	6	5	4	3	2	1	0
Escritura	ACME	ACBGS	ACF	ACIE	ACO	ACOPE		
Reset:	0	0	0	0	0	0	0	0

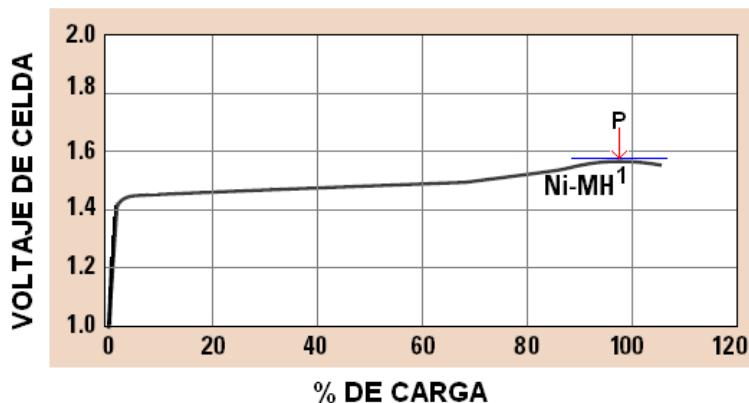
**Figura 15.3. Registro ACMPSC.**

- **ACME:** Bit para habilitar el funcionamiento del módulo ACMP.  
**0:** Módulo ACMP inhibido  
**1:** Módulo ACMP habilitado
- **ACBGS:** Bit para seleccionar la señal de comparación, que puede ser una referencia interna (Bandgap = 1.2V típico) o una señal externa (ACMPx+).  
**0:** Selecciona el pin externo ACMPx+, como señal de comparación

- 1:** Selecciona la referencia interna de 1.2V, como señal de comparación
- **ACF:** Bandera para indicar que ha ocurrido un evento de comparación. El evento es seleccionado dependiendo del bit ACMOD. Esta bandera se aclara, cuando se escribe un “1” sobre esta.  
**0:** No ha ocurrido un evento de comparación  
**1:** Ha ocurrido un evento de comparación
  - **ACIE:** Bit para habilitar un evento de interrupción ante la puesta a “1” de la bandera ACF.  
**0:** Inhibe interrupción en el ACMP  
**1:** Habilita evento de interrupción en le ACMP
  - **ACO:** Bit para indicar el estado de la salida del comparador.
  - **ACOPE:** Bit para habilitar el estado de la salida del comparador, hacia un pin de la MCU.  
**0:** Inhibe la salida del comparador al pin ACMPO<sub>x</sub>  
**1:** Habilita la salida del comparador al pin ACMPO<sub>x</sub>
  - **ACMOD:** Bits para elegir el tipo de señal de salida del comparador, reflejada en la bandera ACF.  
**X0:** Salida a flanco de bajada ante evento de comparación  
**01:** Salida a flanco de subida ante evento de comparación  
**11:** La salida cambia de estado ante evento de comparación

## 15.2. EJEMPLO CON EL MÓDULO ACMP

El ejemplo para ilustrar el funcionamiento del módulo ACMP, trata sobre el control de la carga de una celda de batería tipo NIMH (*Niquel Metal Hydride*). La Figura 15.4 muestra la curva característica de carga de una celda NIMH a 25°C.



1: Curva característica de carga de una celda a 25°C

**Figura 15.4. Curva de carga de celda NIMH.**

**NOTA:** Se recomienda al usuario revisar cuidadosamente el estado de los puentes del DEMOJM, con el objetivo de hacer compatible el ejercicio actual y de esta manera liberar los pines comprometidos en el circuito.

El punto **P** de la gráfica corresponde al codo máximo de carga de la celda, ubicado en un 98% de la curva. En términos del voltaje almacenado en la celda, correspondería aproximadamente a 1.55V.

Asumiendo que la celda a cargar es de 200mAh, que utilizando una fuente de corriente de 200mA (Tasa de carga = 1C) se tomaría una hora en llegar a su carga máxima (100%), cuando el voltaje llega a 1.1V. Este punto se podría alcanzar temporizando la carga de la celda y correspondería a un ligero descenso en el codo máximo de la gráfica (punto del 100% de la carga).

La Figura 15.5 ilustra sobre el circuito sugerido para el control de la carga de la celda.

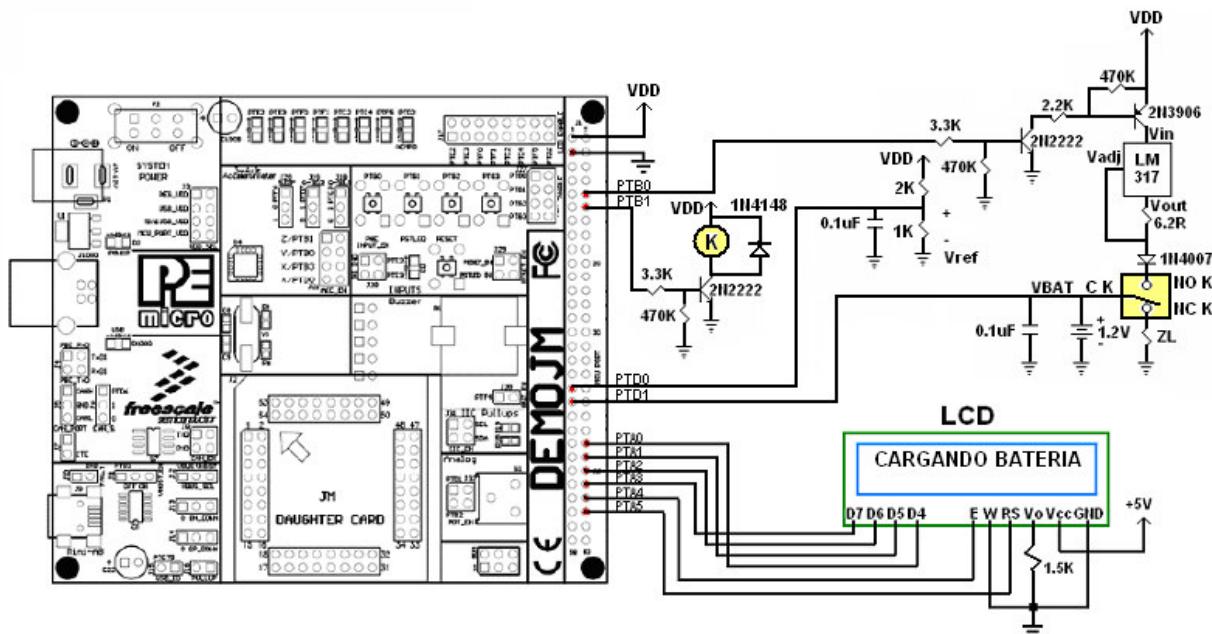


Figura 15.5. Circuito carga de batería.

El circuito de carga está conformado por una fuente de corriente, implementada con un LM317 y ajustada a 200mA (1.25V / 6.2Ω). La fuente de corriente es controlada por el pin PTB0, que prende y apaga el transistor 2N3906.

La batería es conectada al circuito de carga mediante el pin PTB1, que opera el relé K. Cuando el voltaje en la batería esté por debajo de 1.1V, el relé conecta la fuente de corriente y la carga ZL es abandonada de la alimentación que le proporciona la batería. El umbral de comparación para el límite de la descarga ( $V_{bat} < 1.1V$ ) es suministrado por la referencia externa del módulo ACMP.

La batería será cargada hasta que el tiempo sea de una hora y su voltaje será monitoreado en la entrada PTD1 (ACMPx-). Para esta comparación se utiliza el divisor de tensión conectado a PTD0 (ACMPx+).

A continuación se lista un programa en C, que resuelve el ejercicio propuesto.

```
/*
 * ACMP_CARGA: Ejemplo sobre la utilizacion del modulo ACMP
 * main.c
 */
/*
 * Fecha: Mayo 30, 2008
 */
/*
 * V1.0, Rev 0 por Diego Múnera
 */
/*
 * Asunto: Implementar un codigo en C que controle la carga en una celda de NIMH de 200mAh.
 */
/*
 * Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros
 * editores.
 */
/*
 * DISTRIBUCION DE LOS PINES DE PUERTOS
 */

/*
 * PUERTO      * PIN    * I/O    * TIPO    * INICIA    * COMENTARIO
 */
/*PTA      * -    * -    * -    * -    *LCD
/*      *PTA0    * O    * D    * 1    *D4 del LCD
/*      *PTA1    * O    * D    * 1    *D5 del LCD
/*      *PTA2    * O    * D    * 1    *D6 del LCD
/*      *PTA3    * O    * D    * 1    *D7 del LCD
/*      *PTA4    * O    * D    * 1    *Señal ENABLE del LCD
/*      *PTA5    * O    * D    * 1    *Señal RS del LCD
/*PTB      * -    * -    * -    * -    *Control carga
/*      *PTB0    * O    * D    * -    *ON/OFF de la fuente de corriente
/*      *PTB1    * O    * D    * -    *ON/OFF del rele para conexion de bateria
/*PTC      * -    * -    * -    * -    *No usado
/*PTD      * -    * -    * -    * -    *Comparacion analoga
/*      *PTD0    * I    * A    * -    *Referencia alta de la carga (1.55V)
/*      *PTD1    * I    * A    * -    *Voltaje en la bateria
/*PTE      * -    * -    * -    * -    *No usado
/*PTF      * -    * -    * -    * -    *No usado
/*PTG      * -    * -    * -    * -    *No usado
/*PTH      * -    * -    * -    * -    *No usado
/*PTJ      * -    * -    * -    * -    *No usado
/*
 * Archivos de cabecera */
#include <hidef.h>           //macro para interrupciones
#include "derivative.h"         //macro de declaraciones del MCU

/* Declaracion de variables y constantes */
byte bandera=0;                //bandera de proposito general
word seg=0;                     //variable para conteo de segundos
unsigned long retardo=0;        //parametro para hacer retardo
char i=0;                       //variable de iteracion
```

```

char PTAD_Config=0;           //byte de inicializacion del registro PTAD
char PTADD_Config=0;          //byte de inicializacion del registro PTADD
char PTAPE_Config=0;          //byte de inicializacion del registro PTAPE
char PTASE_Config=0;          //byte de inicializacion del registro PTASE
char PTADS_Config=0;          //byte de inicializacion del registro PTADS
char PTAIFE_Config=0;         //byte de inicializacion del registro PTAIFE
char PTBD_Config=0;           //byte de inicializacion del registro PTBD
char PTBDD_Config=0;          //byte de inicializacion del registro PTBDD
char PTBPE_Config=0;          //byte de inicializacion del registro PTBPE
char PTBSE_Config=0;          //byte de inicializacion del registro PTBSE
char PTBDS_Config=0;          //byte de inicializacion del registro PTBDS
char PTBIFE_Config=0;          //byte de inicializacion del registro PTBIFE
char comando8=0;              //parametro inicializacion LCD modo 8 bits
char dato4;                  //dato a enviar al LCD
char rs=0;                   //señal de dato o comando al LCD
char a=0;                     //variable temporal
const unsigned char mensaje1[] = " BATERIA NORMAL "; //Arreglo del mensaje 1 para llevar al LCD
const unsigned char mensaje2[] = "CARGANDO BATERIA"; //Arreglo del mensaje 2 para llevar al LCD
char *pM;                      //puntero a mensaje para el LCD

/* Macros y definiciones */
#define Disable_COP() SOPT1 &= 0x3F //deshabilita el COP
#define Carga_On()  PTBD_PTBD0 = 1; PTBDD_PTBDD0 = 1      //macro para encendido del rele
#define Carga_Off() PTBD_PTBD0 = 0; PTBDD_PTBDD0 = 1      //macro para apagado del rele
#define Rele_On()   PTBD_PTBD1 = 1; PTBDD_PTBDD1 = 1      //macro para encendido del rele
#define Rele_Off()  PTBD_PTBD1 = 0; PTBDD_PTBDD1 = 1      //macro para apagado del rele

/* Declaracion de funciones */
void Delay(unsigned long retardo);           //funcion para hacer reatrdos varios
void LCD_Init(void);                        //funcion para inicilaizar LCD
void Lcd_4bit_Wr(char dato4,char rs);        //funcion para escribir dato al LCD
void Comando_8bit(char comando8);            //funcion para inicio del LCD a 8 bits
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config); //declare funcion que inicializa PTA
void PTB_Init(char PTBD_Config,char PTBDD_Config,char PTBPE_Config,char PTBSE_Config,char PTBDS_Config,char PTBIFE_Config); //declare funcion que inicializa PTB
void ACMP_Init(char ACMPSC_Config); //declare funcion para inicializar modulo ACMP
void RTC_Init(char RTCMOD_Config,char RTCSConfig); //funcion para inicializacion del modulo RTC

/* main(): Funcion principal */
void main(void) {
    MCGC1=0x04;                           //reloj en modo FEI y divisor por 1
    MCGC2=0x00;
    MCGC4=0x22;                           //rango alto del DCO
    Disable_COP();                         //deshabilita el COP
    EnableInterrupts;                      //habilita interrupciones

    PTA_Init(0x00,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "0",pines como salida, no pullups,
                                                //si slew, no strength y si filtro
    PTB_Init(0x00,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "0",pines como salida, no pullups,
                                                //si slew, no strength y si filtro
    LCD_Init();                            //inicialice LCD
    pM = (char *)mensaje1;                //puntero toma direccion del mensaje con definicion de tipo
    for (i=0;i<15;i++){
        Lcd_4bit_Wr(*pM,1);             //envia caracter al LCD
        pM++;                          //incrementa puntero al arreglo del mensaje
    }
    ACMP_Init(0x81);                      //modulo ON, ref externa (1.1V), polling y flanco subida
    ACMPSC_ACF=1;                         //aclare bandera de deteccion espurea

    for(;;) {                             //iteracion para presentacion de la temperatura (for infinito)

//Procedimiento para detectar voltaje de bateria por debajo de 1.1V:
    if (ACMPSC_ACF==1){                 //si se detecta evento de comparacion a flanco de subida
        ACMPSC_ACF=1;                  //aclare bandera de deteccion
        Rele_On();                     //commuta bateria a carga
        Carga_On();                   //enciende fuente de corriente
        Lcd_4bit_Wr(0x80,0);           //primer caracter fila 1 columna 1
    }
}

```

```

pM = (char *)mensaje2;           //puntero toma direccion del mensaje 2
for (i=0;i<15;i++){
    Lcd_4bit_Wr(*pM,1);        //envia caracter al LCD
    pM++;
}
//Inicia temporizacion para carga de bateria por 1 hora
RTC_Init(0x00,0x1F);          //modulo al maximo, reloj LPO (1KHz) y divisor por 1000 para
                               //conteo de un segundo, habilita interrupcion
}

if(bandera==1){
    bandera=0;
    Rele_Off();                //desconecta bateria de la carga
    Carga_Off();               //apaga fuente de corriente
    Lcd_4bit_Wr(0x80,0);       //primer caracter fila 1 columna 1
    pM = (char *)mensaje1;     //puntero toma direccion del mensaje 1
    for (i=0;i<15;i++){
        Lcd_4bit_Wr(*pM,1);   //envia caracter al LCD
        pM++;
    }
    RTCSC_RTCIE=0;             //inhiba interrupcion por RTC
}

/*
/* Funcion para inicializar el puerto A */
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config){
    PTAD=PTAD_Config;          //inicialice estado pinos PTA
    PTADD=PTADD_Config;        //inicialice direcci n de pinos PTA
    PTAPE=PTAPE_Config;        //inicialice estado de pullups PTA
    PTASE=PTASE_Config;        //inicialice estado del slew rate PTA
    PTADS=PTADS_Config;        //inicialice estado de drive strength PTA
    PTAIFE=PTAIFE_Config;      //inicialice estado del filtro PTA
}

/* Funcion para inicializar el puerto B */
void PTB_Init(char PTBD_Config,char PTBDD_Config,char PTBPE_Config,char PTBSE_Config,char PTBDS_Config,char PTBIFE_Config){
    PTBD=PTBD_Config;          //inicialice estado pinos PTB
    PTBDD=PTBDD_Config;        //inicialice direcci n de pinos PTB
    PTBPE=PTBPE_Config;        //inicialice estado de pullups PTB
    PTBSE=PTBSE_Config;        //inicialice estado del slew rate PTB
    PTBDS=PTBDS_Config;        //inicialice estado de drive strength PTB
    PTBIFE=PTBIFE_Config;      //inicialice estado del filtro PTB
}

/* Funcion para inicializar el RTC */
void RTC_Init(char RTCMOD_Config,char RTCSC_Config){
    RTCMOD=RTCMOD_Config;      //inicialice modulo del RTC
    RTCSC=RTCSC_Config;        //inicialice registro de estado y control del RTC
}

/*Funcion para inicializacion del ACMP */
void ACMP_Init(char ACMPSC_Config){
    ACMPSC=ACMPSC_Config;      //inicializa registro ACMPSC del ACMP
}

/* Funcion para inicializacion del LCD a 4 bits */
void LCD_Init(void){
    Delay(200000);             //hace retardo
    Comando_8bit(0x30);        //llama funci n enviar comando en 8 bits con comando
    Delay(200000);             //hace retardo
    Comando_8bit(0x30);        //llama funci n enviar comando en 8 bits con comando
    Delay(200000);             //hace retardo
    Comando_8bit(0x20);        //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x28,0);       //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x0C,0);       //ON LCD, OFF cursor
}

```

```

Lcd_4bit_Wr(0x01,0);           //borrar pantalla LCD
Lcd_4bit_Wr(0x06,0);           //desplaza cursor a la derecha con cada caracter
Lcd_4bit_Wr(0x80,0);           //primer caracter fila 1 columna 1
}

/* Funcion para escribir a 4 bits en el LCD */
void Lcd_4bit_Wr(char dato4,char rs){
if (rs==0){                      //pregunte si se va a enviar un comando o un dato
    PTAD_PTAD5=0;                //prepara envío de comando RS = 0
}
else{
    PTAD_PTAD5=1;                //prepara envio de dato RS = 1
}
a=dato4;                         //almacena temporalmente el dato
a=a>>4;                          //desplaza parte alta dato para la baja
PTAD&=0xF0;                       //enmascara parte alta del puerto A
a&=0x0F;                          //enmascara parte baja del dato a enviar al LCD
PTAD_PTAD4=1;                     //sube pin de enable
Delay(20000);                     //hace retardo
PTAD|=a;                           //envía nibble alto del dato
Delay(20000);                     //hace retardo
PTAD_PTAD4=0;                     //baja pin de enable
Delay(20000);                     //hace retardo
PTAD&=0xF0;                       //enmascara parte alta del puerto A
dato4&=0x0F;                      //enmascara parte baja del dato a enviar al LCD
PTAD_PTAD4=1;                     //sube pin de enable
Delay(20000);                     //hace retardo
PTAD|=dato4;                      //envía nibble bajo del dato
Delay(20000);                     //hace retardo
PTAD_PTAD4=0;                     //baja pin de enable
Delay(20000);                     //hace retardo
}

/* Función para envio de comando en 8 bits al LCD (solo para inicio) */
void Comando_8bit(char comando8){
    PTAD_PTAD5=0;                  //prepara envio de comando
    PTAD&=0xF0;                    //enmascara parte baja del puerto A
    comando8&=0xF0;                //enmascara parte alta de comando a enviar al LCD
    comando8=comando8>>4;         //desplaza para tomar nibble alto
    PTAD_PTAD4=1;                  //sube pin de enable
    Delay(20000);                 //hace retardo
    PTAD|=comando8;                //envía comando
    Delay(20000);                 //hace retardo
    PTAD_PTAD4=0;                  //sube pin de enable
    Delay(20000);                 //hace retardo
}

/* Funcion Delay(): Retarda basado en una variable tipo entero */
void Delay(unsigned long retardo){
while(retardo>0){                  //llego a cero?
    retardo--;                      //no --> decrementa
}
}

/* Funcion de atencion a la interrupcion por RTC */
interrupt VectorNumber_Vrtc void ISR_RTC_OVF(void){
    RTCSC_RTCIF = 1;                //aclara bandera de sobreflujo del RTC
    seg++;                           //incremente segundos
    if (seg == 3600){                //si pasaron 3600 segundos (1 hora)
        seg = 0;                     //aclare segundos
        bandera=1;                   //ponga bandera en "1"
    }
}

```

### **15.3. REFERENCIAS**

- MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual. Freescale Semiconductor. Rev 1, 2008.
- Microcontroller MCF51JM128 ColdFire Data Sheet. Freescale Semiconductor. Rev 0, 2008.

### **15.4. PREGUNTAS**

- ¿Cuántas y cuáles son las maneras como se puede establecer una comparación usando el módulo ACMP, contra una señal de entrada?
- ¿Cómo hacer para obtener eléctricamente el resultado de la comparación de dos señales que entran al módulo ACMP?

# CAPÍTULO 16

## Comunicaciones Seriales Asíncronas (SCI: *Serial Communication Interface*)

17.1. Diagrama en bloques, explicación del funcionamiento y registros asociados

17.2. Programa ejemplo: Comunicación con un PC vía Hyperterminal

17.3. Referencias

17.4. Ejercicios y preguntas

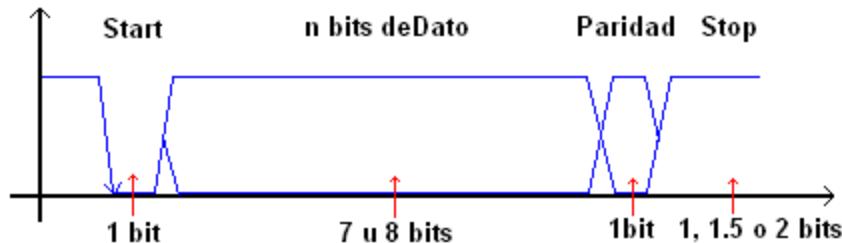
## 16.1. DIAGRAMA EN BLOQUES, EXPLICACIÓN DEL FUNCIONAMIENTO Y REGISTROS ASOCIADOS

- **Breve repaso de las comunicaciones seriales asíncronas**

Una comunicación asíncrona es aquella que no transporta información del reloj de sincronización de los bits. El reloj no es transportado en un conductor independiente ni va embebido en los bits de información.

El reloj se maneja independientemente para cada transceptor (Transmisor/Receptor) y debe existir un acuerdo entre ambos, para el establecimiento de la sincronía. A continuación se presentan algunos conceptos importantes sobre comunicaciones seriales asíncronas.

**Trama:** Los bits viajan envueltos en una trama (frame) o protocolo de nivel 1, que tiene la forma de la Figura 16.1.



**Figura 16.1. Trama serial asíncrona**

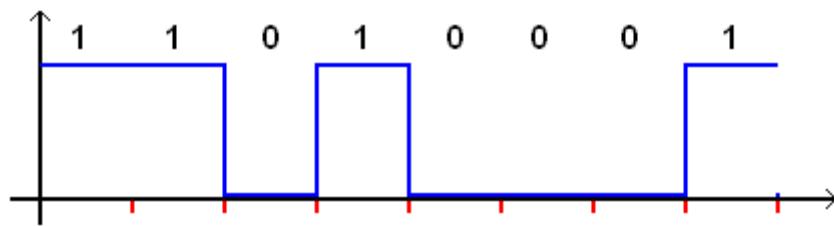
La condición de *Start* toma un tiempo de bit y es validada cuando la señal en el canal pasa de “1” a “0”, esta condición abre la comunicación.

Los bits de información pueden variar entre 7, 8 y 9, dependiendo de si se tiene en cuenta la paridad de la información y formán la carga útil de la información (*payload*).

La paridad es opcional y puede fijarse a par o impar o simplemente no ir. Finalmente, la condición de *Stop* cierra la comunicación y puede ser pactada a 1, 1.5 y hasta 2 bits, siendo la más común de 1 bit.

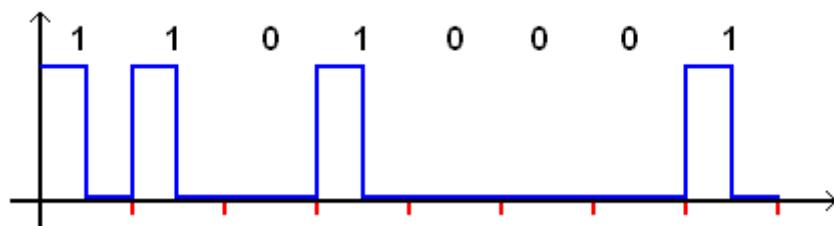
Cada dato a enviarse deberá estar acompañado por el formato de trama descrito y a la razón de bits en el tiempo se le denomina baudio. Un baudio es un bit por segundo (bps) y establece la tasa de transferencia de información.

**El formato eléctrico:** De una comunicación serial asíncrona puede ser con retorno a cero (RZ) o sin éste (NRZ). Cuando el formato es NRZ (*Non Return to Zero*), la señal permanece todo el tiempo de bit sosteniendo el nivel lógico de éste (ver Figura 16.2).



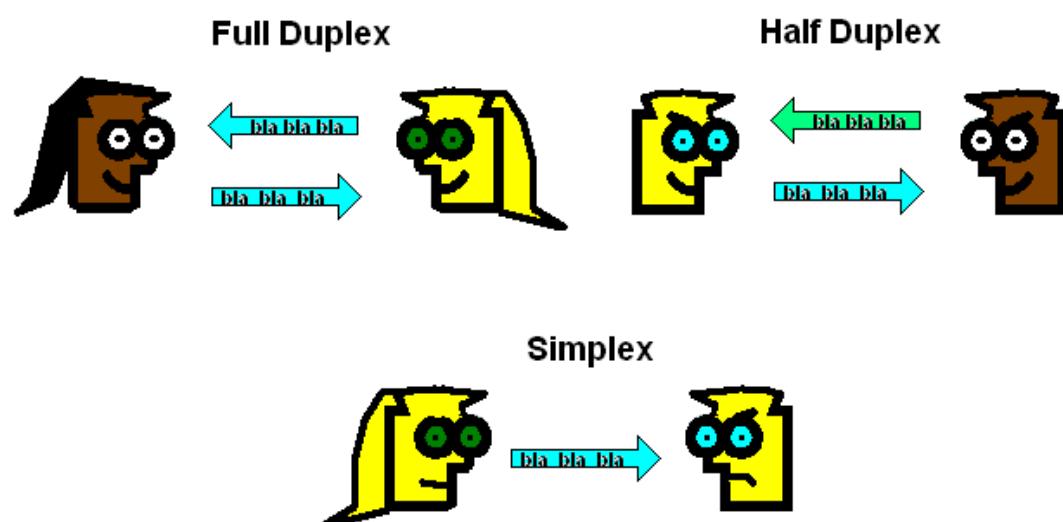
**Figura 16.2. Señal NRZ.**

Cuando el formato es RZ (*Return to Zero*), un “1” lógico permanece medio tiempo en nivel alto y luego cae (ver Figura 16.3). En promedio, esta técnica consumiría la mitad de la energía para enviar información de un sistema a otro y viceversa.



**Figura 16.3. Señal RZ.**

**Flujo de la información:** Referido a como se origina la información y al aprovechamiento temporal de la comunicación. En este sentido, la comunicación puede ser *Full Duplex*, *Half Duplex* y *Simplex*. La Figura 16.4 muestra la clasificación anterior.



**Figura 16.4. Flujo de la información.**

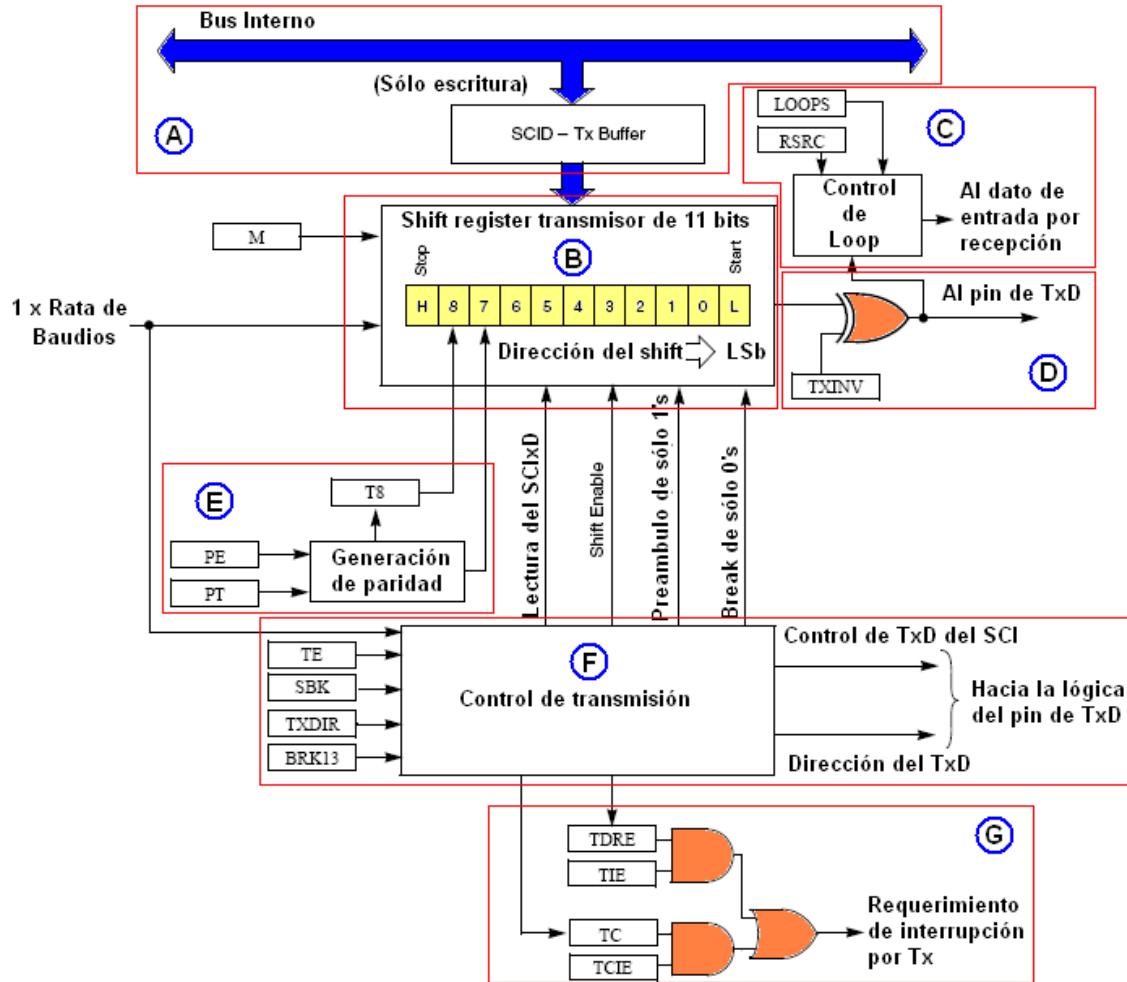
Para el flujo *Full Duplex* la información es transmitida y recibida simultáneamente, de tal manera que el aprovechamiento del canal es el máximo. *Half Duplex* trabaja en ambas direcciones, pero en tiempos distintos y no existe simultaneidad en la comunicación. Finalmente, *Simplex* sólo utiliza una dirección en el flujo de la comunicación.

- **Breve descripción del módulo SCI y diagrama en bloques**

Algunas de las características más importantes del módulo ACMP son:

- Hasta dos canales de comunicación SCI, SCI1 y SCI2.
- Transmisor y receptor habilitados independientemente y con *buffer* doble.
- Rata de baudios programable con un divisor de 13 bits.
- Fuentes de interrupción o evento de *polling*, por:
  - *Buffer* de transmisor vacío
  - Transmisión completa
  - Registro de datos del receptor lleno
  - Sobrecarrera en la recepción (Receive Overrun)
  - Error por paridad (*Parity Error*)
  - Error en la trama (*Framming Error*)
  - Error por ruido (*Noise Error*)
  - Detección de receptor en modo vago (*Idle Receptor*)
  - Activo a flanco en el pin de reopción (*Active Edge Receptor Pin*)
  - Detección de pausa (*Break Detect*)
- Generación y chequeo de paridad.
- Selección de la longitud del dato en 8 o 9 bits.
- Despertar del receptor por modo vago o marca de dirección (*Idle-Line /Address-Mark*).
- Opción de generación de carácter de pausa en 13 bits (*Break Character Generation*).
- Opción de detección de carácter de pausa en 11 bits (*Break Character Detection*).
- Polaridad del transmisor programable.

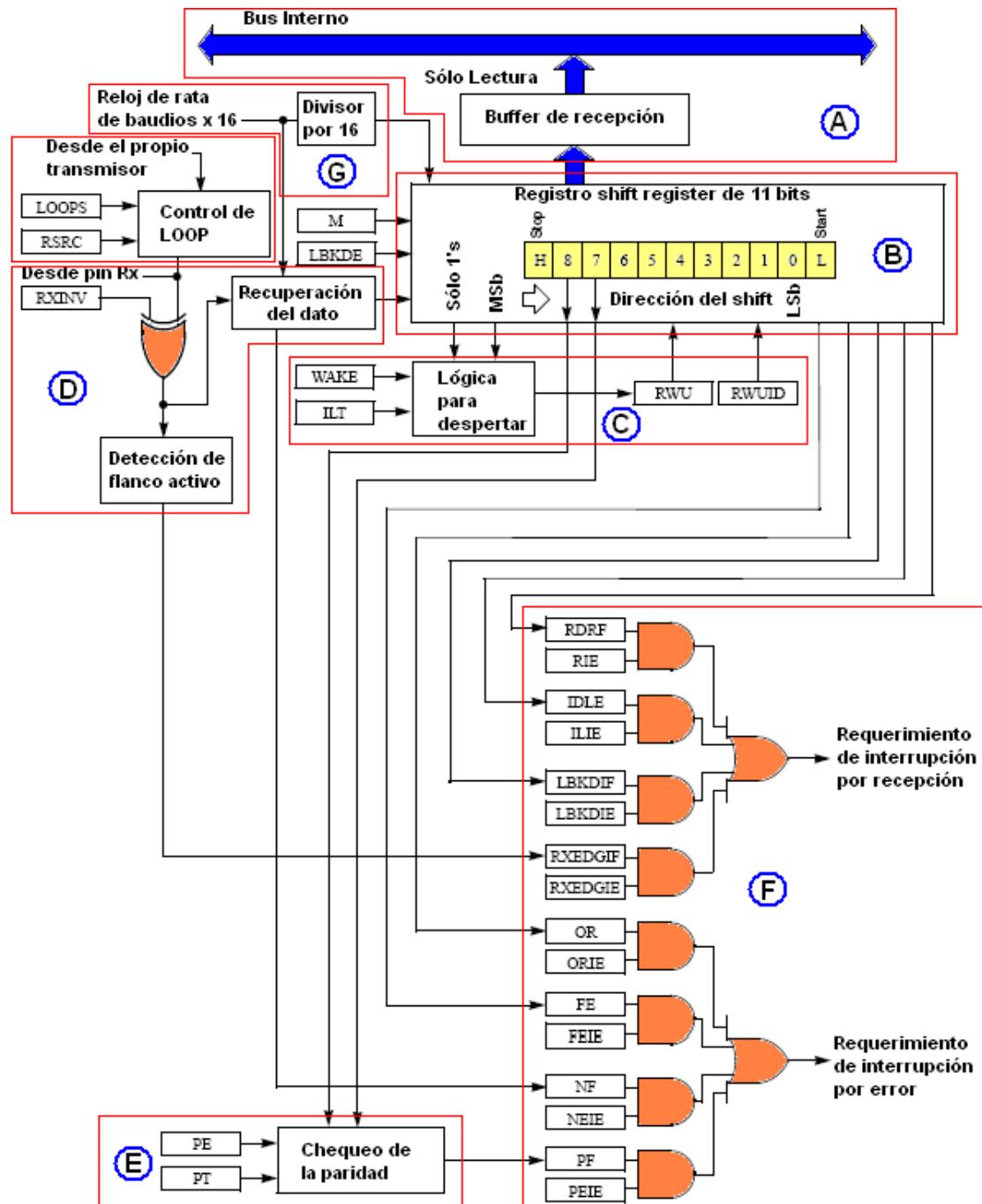
La Figura 16.5. ilustra el diagrama en bloques del transmisor del módulo SCI, en donde el circuito (A) configura el *buffer* de transmisión, el cual recibe el dato a transmitir desde el bus de datos interno.



**Figura 16.5. Diagrama en bloques del transmisor del SCI.**

El circuito (B) corresponde al *shift register* que controla el formato de la trama SCI y su salida hacia el pin de transmisión, en donde el primer bit del dato en salir es el LSb. El circuito (C) conforma el control de realimentación (LOOP) del dato hacia el shift register del receptor, con el fin de hacer comprobaciones del módulo SCI en una comunicación. A esta propiedad se le conoce también con el nombre de eco local y consiste en la realimentación de todo lo que se transmite hacia el propio módulo. La función OREX del circuito (D), permite que los bits de salida del transmisor se complementen en su lógica. Esta opción es interesante desde el punto de vista de confundir a sistemas espías, que se conecten en paralelo al canal. El circuito generador de paridad (E), permite introducir el bit de paridad a la trama del shift register y seleccionar el tipo de la misma. El usuario puede optar por omitir la paridad y optar por transmitir 9 bits de dato. El circuito (F) corresponde al controlador de los eventos de la transmisión, como envío de caracteres especiales (*break*), controlar la dirección del pin de transmisión, monitoreo de cuándo una transmisión ha finalizado, entre otros. Finalmente, el circuito (G) controla los eventos de interrupción del transmisor.

La Figura 16.6. ilustra el diagrama en bloques del receptor del módulo SCI, en donde el circuito (A) configura el *buffer* de recepción, el cual almacena el dato recibido para ser llevado al bus de datos interno.



El circuito (**B**) corresponde al *shift register* que controla el formato de la trama SCI y su entrada desde el pin de recepción en donde el primer bit del dato en llegar es el LSb. El circuito (**C**) conforma el control de realimentación (LOOP) del dato que viene del shift register del transmisor local. La función OREX del circuito (**D**), permite que los bits de entrada recuperen la polaridad pactada con el transmisor. El circuito chequeador de paridad (**E**), permite comprobar la paridad pactada en la comunicación. El circuito (**F**) controla los eventos de interrupción del receptor, tanto por eventos de error como los propios de la recepción. El circuito (**G**) corresponde con la lógica de despertar (*Wakeup*). Finalmente el circuito de reloj (**H**), extrae el reloj del receptor dividiendo por 16 el reloj utilizado para el análisis de los bits.

- **Otras funciones importantes asociadas al módulo SCI**

La comunicación vía SCI permite el envío de caracteres de pausa (*break*) y la posibilidad de despertar el circuito receptor de un modo vago (IDLE).

- **Envío de caracteres de pausa (break):** Esta función permite hacer pausas para comunicaciones con receptores de ratas de baudios lentas. Un carácter de pausa está formado de sólo “0’s”, incluyendo el bit de *start* y el bit de *stop*.

Es importante tener en cuenta que si se está conectado a otro dispositivo microcontrolador de la familia Freescale se va a generar un error de trama, debido a la ausencia del bit de *stop*.

La tabla 18.1 ilustra sobre el tamaño de un carácter break dependiendo de los bits de configuración SCIxC1[M] y SCIxS2[BRK13].

**Tabla 16.1. Tamaño de los caracteres break.**

BRK13	M	Longitud del carácter break
0	0	10 Tiempos de bit
0	1	11 Tiempos de bit
1	0	13 Tiempos de bit
1	1	14 Tiempos de bit

- **Mecanismo de despertar al receptor (*Wakeup*):** Esta función permite que el receptor ignore mensajes que no son propios y de esta manera evitar desgaste innecesario de tiempo y energía.

En un sistema SCI elaborado como una red, todos los receptores evalúan el primer carácter de cada mensaje y tan pronto como ellos determinan que el mensaje no es propio, escriben un “1” en el bit SCIxC2[RWU]. Este bit en “1” hace que las banderas de estado asociadas al receptor sean inhibidas de ser puestas a “1”, con

esto se evita que el *software* asociado al receptor dedique tiempo innecesario en la atención a mensaje que no son para este.

Cuando un mensaje termina o comienza uno nuevo, los receptores automáticamente ponen en “0” el bit SCIxC2[RWU], con el fin de hacer un despertar (*Wake up*) y analizar el primer carácter del nuevo mensaje. Existen dos maneras de generar un evento de *wakeup* a un receptor, a saber:

- **Despertar por línea desocupada (*Idle Line Wakeup*):** Se establece con el bit SCIxC1[WAKE] en “0” y en este modo el bit SCIxC2[RWU] es aclarado automáticamente al detectarse un carácter IDLE (sólo “1’s”, incluyendo el bit de *start* y *stop*).

Cuando el bit SCIxC2[RWU] es “1” y el bit SCIxS2[RWUID] es cero, la condición de línea desocupada que despierta el receptor no pondrá a “1” la bandera SCIxS1[IDLE] del receptor. El receptor despierta y espera el primer carácter del siguiente mensaje el cual pone a “1” la bandera SCIxS1[RDRF], que genera un evento de interrupción, si está habilitada. Lo anterior se hace sin importar el estado del bit SCIxC2[RWU].

El bit de selección de tipo de línea desocupada SCIxC1[ILT] (*Idle Line Type*) elige entre iniciar el contador de bits en modo vago (*Idle Line*). Cuando ILT = “0”, se espera a que haya pasado el bit de *start* y/o cualquiera otro bit en “1” al final del carácter recibido. Cuando ILT = “1”, el contador de bits en modo vago no cuenta hasta tanto no haya pasado el tiempo del bit de *stop*, esto hace que la detección de la línea en modo vago no sea afectada por el dato en el último carácter del mensaje previo.

- **Despertar por marca de dirección (*Address Mark Wakeup*):** Se establece con el bit SCIxC1[WAKE] en “1” y en este modo el bit SCIxC2[RWU] es aclarado automáticamente al detectarse un “1” en el MSb del carácter recibido.

El “1” detectado en el MSb del carácter recibido aclara el bit SCIxC2[RWU] y pone a “1” la bandera SCIxS1[RDRF], antes de que el bit de *stop* sea recibido. Para este caso el carácter con el MSb en “1” es recibido, incluso si el receptor permaneció dormido durante la mayor parte del tiempo del carácter recibido.

- **Registros asociados al módulo SCI**

- **Registros de tasa de baudios (SCIxBDH:SCIxBDL):** La Figura 16.7 muestra el registro de configuración de la tasa de baudios del SCI. Se recomienda al usuario escribir primero sobre el registro SCIxBDH y luego sobre el registro SCIxBDL. Para el caso de programación en C, es posible escribir sobre el registro SCIxBD, el cual agrupa los registros anteriores.

### Registros de rata de baudios (SCIxBDH:SCIxBDL)

Lectura Escritura	7	6	5	4	3	2	1	0
	LBKDIIE	RXEDGIE	0			SBR[12:8]		
Reset	0	0	0	0	0	0	0	0
Lectura Escritura	7	6	5	4	3	2	1	0
						SBR[7:0]		
Reset	0	0	0	0	0	1	0	0

Figura 16.7. Registros SCIxBDH.

- **LBKDIIE:** Bit para habilitar que se genere un evento de interrupción debido a la recepción de un carácter de pausa (*break*) .
  - 0:** No habilita interrupción por evento de pausa
  - 1:** Habilita interrupción por evento de pausa
- **RXEDGIE:** Bit para habilitar que se genere un evento de interrupción debido a un flanco presente en el pin de recepción .
  - 0:** No habilita interrupción por flanco en pin de Rx
  - 1:** Habilita interrupción por flanco en pin de Rx
- **SBR(12:0):** Bits para la programación del divisor de la rata de baudios. Cuando el valor de estos bits es cero, el generador de la rata de baudios se detiene con el fin de ahorrar energía. La ecuación para el cálculo de la rata de baudios es:

$$\text{Rata de Baudios} = \text{Reloj del BUS} / (16 \times \text{SBR})$$

Por ejemplo, si se necesita una rata de baudios de 9600 y se está trabajando con un reloj de BUS interno de 8MHZ, el SBR será de:

$$\text{SBR} = 8\text{MHz} / (16 \times 9600\text{Hz})$$

$$\text{SBR} = 52.08$$

El valor anterior se puede aproximar a 52, sin ningún deterioro del sincronismo de la comunicación. La Figura 16.8 ilustra la dirección de los registros de rata de baudios asociados a los módulos SCIx.

0x(FF)FF_8038	SCI1BDH	LBKDIIE	RXEDGIE	0	SBR12	SBR11	SBR10	SBR9	SBR8
0x(FF)FF_8039	SCI1BDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0
0x(FF)FF_8040	SCI2BDH	LBKDIIE	RXEDGIE	0	SBR12	SBR11	SBR10	SBR9	SBR8
0x(FF)FF_8041	SCI2BDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0

Figura 16.8. Dirección de los registros SCIxBD.

- **Registro de control 1 del SCI (SCIxC1)** La Figura 16.9 detalla el registro de configuración 1 del SCI.

### Registro de configuración 1 (SCIxC1)

Lectura Escritura	7	6	5	4	3	2	1	0
Reset	0	0	0	0	0	0	0	0
0x(F)FF_803A	SCI1C1	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE
0x(F)FF_8042	SCI2C1	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PT

**Figura 16.9. Registro SCIxC1.**

- **LOOPS:** Bit para seleccionar entre operación normal y modo de realimentación (LOOP). Cuando este bit vale “1”, la salida del transmisor es conectada internamente a la entrada del receptor.  
**0:** Operación normal del SCI. Los pines RxD y TxD son usados de manera independiente  
**1:** Habilita modo LOOP o conexión por un único alambre (*Single Wire Mode*). El pin RxD se desconecta del SCI
- **SCISWAI:** Bit para detener el SCI en modo WAIT.  
**0:** El SCI continúa trabajando estando la máquina en modo WAIT, de tal forma que un evento de interrupción al interior del SCI podría despertar la máquina.  
**1:** El módulo SCI está congelado en modo WAIT.
- **RSRC:** La operación con este bit tiene sentido si el bit LOOP está en “1”. En modo LOOP el transmisor está conectado internamente al bloque de entrada del receptor, entonces el bit RSRC decide cuando ésta unión se conecta al pin de salida (TxD) del SCI.  
**0:** Modo normal de LOOP, la entrada RxD está desconectada del sistema y la salida del transmisor es conectada internamente a la entrada del bloque receptor.  
**1:** Pone el SCI en modo de un único alambre (*Single Wire Mode*), entonces el pin de TxD es conectado a la salida del transmisor y a la entrada del receptor, internamente.
- **M:** Bit para seleccionar dato de 8 o 9 bits.  
**0:** Modo estándar: 1 bit de start + 8 bits de dato + 1 bit de stop.  
**1:** Modo a 9 bits: 1 bit de start + 8 bits de dato + 1 bit de stop.
- **WAKE:** Bit para seleccionar el método de despertar (*Wakeup*) del SCI.  
**0:** Modo de despertar por *Idle Line*.

**1:** Modo de despertar por *Address Mark*.

- **ILT:** Bit para seleccionar el tipo de línea en modo IDLE. Cuando este bit es puesto a “1”, se asegura de que el bit de stop y el MSb del dato no cuenten dentro de los 10 (u 11) bits necesarios como nivel IDLE para la lógica de detección de dicho modo.
  - 0:** El contador de bits, del carácter para la condición de IDLE, comienza después del bit de start.
  - 1:** El contador de bits, del carácter para la condición de IDLE, comienza después del bit de stop.
- **PE:** Bit para habilitar la generación y el chequeo de paridad en la trama de comunicación.
  - 0:** No se trabaja con paridad.
  - 1:** Habilita el trabajo con paridad.
- **PT:** Bit para seleccionar el tipo de paridad a generarse o a chequear.
  - 0:** Paridad par (Even).
  - 1:** Paridad impar (Odd).

- **Registro de control 2 del SCI (SCIxC2)** La Figura 16.10 detalla el registro de configuración 1 del SCI.

**Registro de control 2 (SCIxC2)**

Lectura Escribir	7	6	5	4	3	2	1	0
	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0
0x(FF)FF_803B	SCI1C2	TIE	TCIE	RIE	ILIE	TE	RE	RWU
0x(FF)FF_8043	SCI2C2	TIE	TCIE	RIE	ILIE	TE	RE	RWU
		SBK	SBK					

**Figura 16.10. Registro SCIxC2.**

- **TIE:** Bit para habilitar un posible evento de interrupción por transmisión, para cuando el *buffer* de transmisión ha bajado el dato al *shift register*.
  - 0:** Inhibe interrupción por evento de TDRE = 1.
  - 1:** Habilita interrupción por evento de TDRE = 1.
- **TCIE:** Bit para habilitar un posible evento de interrupción por transmisión, para cuando el último bit ha salido del *shift register*.
  - 0:** Inhibe interrupción por evento de TC = 1.
  - 1:** Habilita interrupción por evento de TC = 1.
- **RIE:** Bit para habilitar un posible evento de interrupción por recepción.

- 0:** Inhibe interrupción por evento de RDRF = 1.
- 1:** Habilita interrupción por evento de RDRF = 1.

- **ILIE:** Bit para habilitar un posible evento de interrupción por línea en modo IDLE.
    - 0:** Inhibe interrupción por evento de línea en estado de IDLE.
    - 1:** Habilita interrupción por evento de línea en estado de IDLE.
  - **TE:** Bit para habilitar la operación del transmisor.
    - 0:** Inhibe la operación del transmisor.
    - 1:** Habilita la operación del transmisor.
  - **RE:** Bit para habilitar la operación del receptor.
    - 0:** Inhibe la operación del receptor.
    - 1:** Habilita la operación del receptor.
  - **RWU:** Bit para colocar el receptor del SCI en modo de *standby* (atento), esperando a que se dé un evento de hardware o se presente un modo de despertar (*Wakeup*) sea por *idle line* (WAKE = 0) o por *address mark* (WAKE = 1).
    - 0:** SCI en operación normal.
    - 1:** SCI en modo de *standby*, esperando un estado de WAKE.
  - **SBK:** Bit para permitir enviar un carácter *break*. Un carácter *break* consiste en el apilamiento de “0’s” en toda la trama de comunicación. La condición no desaparece hasta tanto no se haga SBK = 0.
    - 0:** SCI en operación normal.
    - 1:** SCI enviando caracteres *break*.
- **Registro de estado 1 del SCI (SCIxS1)** La Figura 18.11 detalla el registro de estado 1 del SCI.

#### Registro de estado 1 (SCIxS1)

	7	6	5	4	3	2	1	0
Lectura	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
Escritura								
Reset	1	1	0	0	0	0	0	0
0x(FF)FF_803C	SCI1S1	TDRE	TC	RDRF	IDLE	OR	NF	FE
0x(FF)FF_8044	SCI2S1	TDRE	TC	RDRF	IDLE	OR	NF	PF

Figura 16.11. Registro SCIxS1.

- **TDRE:** Bandera que indica cuando un dato del *buffer* de transmisión ha pasado al *shift register*. Este evento indica que se puede depositar otro

dato en el *buffer* de transmisión. Para borrar la bandera TDRE es necesario leer el registro SCIxS1 y luego escribir un nuevo dato en el registro de datos del SCI (SCIxD).

**0:** El *buffer* de transmisión está lleno.

**1:** El *buffer* de transmisión está vacío.

- **TC:** Bandera que indica cuando el último bit de una trama ha salido por el *shift register*. Este evento indica que se puede depositar otro dato en el *buffer* de transmisión. Para borrar la bandera TC es necesario leer el registro SCIxS1 y luego ejecutar una de las siguientes acciones:
  - Escribir un nuevo dato en el registro de datos del SCI (SCIxD)
  - Escribir un preámbulo pasando el bit TE de “1” a “0”.
  - Enviar un carácter de *break* escribiendo un “1” en el bit SBK.

**0:** Transmisor enviando un dato.  
**1:** Transmisor en modo IDLE.
- **RDRF:** Bandera que indica cuando el *buffer* de recepción está lleno. El evento indica cuando un dato recibido en el *shift register* ha pasado al registro de datos de SCI (SCIxD). Para aclarar el estado del bit RDRF es necesario leer el registro SCIxS1 y luego leer el registro de datos SCIxD.
  - 0:** El registro de datos del SCI está vacío.  
**1:** El registro de datos del SCI está lleno.
- **IDLE:** Bandera se pone a “1” cuando el receptor ha recibido un carácter de sólo “1’s”. Para aclarar la bandera de IDLE es necesario leer el registro SCIxS1 y luego leer el registro de datos SCIxD.
  - 0:** No se ha detectado un carácter IDLE.  
**1:** Se ha detectado un carácter IDLE.
- **OR:** Bandera de sobrecarrera del receptor (*Receiver Overrun*). Esta bandera indica cuando se va a escribir un nuevo dato entrante sobre el registro de datos de SCI (SCIxD) y aún no se ha leído el dato anterior, en cuyo caso el dato nuevo se pierde. Para aclarar el bit OR es necesario leer el registro SCIxS1 y luego leer el registro de datos SCIxD.
  - 0:** No se ha detectado una sobrecarrera en el receptor.  
**1:** Se ha detectado una sobrecarrera en el receptor y se ha perdido el último dato.
- **NF:** Todo bit de start es muestreado siete veces y tres muestras para los demás bits. Si alguna de esas muestras es errónea, la bandera NF se pone a “1” al igual que el bit RDRF. Para aclarar el bit NF es necesario leer el registro SCIxS1 y luego leer el registro de datos SCIxD.
  - 0:** No se ha detectado ruido en el canal.  
**1:** Se ha detectado ruido en el canal.

- **FE:** Bandera para detección de error en una trama (aunque no es la única condición para descartar una trama corrupta). Una trama se invalida cuando se detecta un “0” en el tiempo del bit de *stop*. Para aclarar el bit FE es necesario leer el registro SCIxS1 y luego leer el registro de datos SCIxD.
  - 0:** No se ha detectado una trama errónea.
  - 1:** Se ha detectado una trama errónea.
- **PF:** Bandera que indica cuando el bit de paridad no coincide con el valor de la paridad pactada. Para aclarar el bit PF es necesario leer el registro SCIxS1 y luego leer el registro de datos SCIxD.
  - 0:** No se ha detectado paridad inválida.
  - 1:** Se ha detectado paridad inválida.
- o **Registro de estado 2 del SCI (SCIxS2)** La Figura 18.12 detalla el registro de estado 2 del SCI.

#### REgistro de estado 2 (SCIxS2)

Lectura Escritura	7	6	5	4	3	2	1	0
Reset	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK13	LBKDE	RAF
0x(FF)FF_803D	SCI1S2	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK13	LBKDE
0x(FF)FF_8045	SCI2S2	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK13	RAF

Figura 16.12. Registro SCIxS2.

- **LBKDIF:** Bandera que indica cuando se ha detectado un carácter de *break* en la línea. Esta bandera se puede aclarar escribiendo un “1” en ella.
  - 0:** No se ha detectado un carácter de *break* en la línea.
  - 1:** Se ha detectado un carácter de *break* en la línea.
- **RXEDGIF:** Bit para indicar cuando ha ocurrido un evento de interrupción por un flanco recibido en el pin de RxD. El flanco puede ser de bajada (RXINV = 0) o de subida (RXINV = 1). Esta bandera se puede aclarar escribiendo un “1” en ella.
  - 0:** No se ha detectado un flanco en el pin RxD.
  - 1:** Se ha detectado un flanco en el pin RxD.
- **RXINV:** Bit para invertir la polaridad de la señal eléctrica aplicada al pin RxD.
  - 0:** No se ha invertido la polaridad en el pin RxD.
  - 1:** Se ha invertido la polaridad en el pin RxD.

- **RXUID:** Bit para indicar la detección de un estado de despertar por modo vago (*Idle Wakeup*). Indica cuando un carácter IDLE ha puesto la bandera IDLE en “1”.
  - 0:** Durante el estado de atento del receptor ( $RWU = "1"$ ), el bit IDLE no ha sido puesto a “1” en la detección de un carácter IDLE.
  - 1:** Durante el estado de atento del receptor ( $RWU = "1"$ ), el bit IDLE ha sido puesto a “1” en la detección de un carácter IDLE.
- **BRK13:** Bit para seleccionar la longitud de un carácter *break*.
  - 0:** El carácter de *break* será de 10 bits (11 si  $M = 1$ ).
  - 1:** El carácter de *break* será de 13 bits (14 si  $M = 1$ ).
- **LBKDE:** Bit para habilitar un carácter *break* en la línea. Mientras este bit sea “1”, el sistema previene que el bit FE (*Framming Error*) y RDRF sean puestos a “1”.
  - 0:** Un carácter de *break* de 10 bits es detectado (11 si  $M = 1$ ).
  - 1:** Un carácter de *break* de 13 bits es detectado (14 si  $M = 1$ ).
- **RAF:** Bandera para indicar que el receptor ha detectado un bit de *start* válido y es borrada automáticamente cuando el receptor detecta la línea en modo IDLE. Esta bandera puede ser usada para chequear cuando un carácter está comenzando a ser recibido, antes de que la CPU entre en un estado de STOP.
  - 0:** El receptor en modo IDLE está esperando un bit de *start*.
  - 1:** El receptor del SCI está activo y el pin RxD no está en IDLE.
- **Registro de control 3 del SCI (SCIxC3)** La Figura 16.13 detalla el registro de control 3 del SCI.

#### Registro de control 3 (SCIxC3)

	7	6	5	4	3	2	1	0
Lectura	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
Escritura								
Reset	0	0	0	0	0	0	0	0
0x(FF)FF_803E	SCI1C3	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE
0x(FF)FF_8046	SCI2C3	R8	T8	TXDIR	TXINV	ORIE	NEIE	FEIE
								PEIE

Figura 16.13. Registro SCIxC3.

- **R8:** Noveno bit del receptor, para  $M = 1$  y conforma el MSb del dato recibido. Cuando se está trabajando en modo de 9 bits, se recomienda leer R8 antes de leer el SCIxD.

- **T8:** Noveno bit a transmitir, para  $M = 1$  y conforma el MSb del dato a transmitir. Cuando se está trabajando en modo de 9 bits, escribir primero el dato a transmitir en el registro SCIx D y luego definir el bit T8.
  - **TXDIR:** Bit para definir la dirección del pin de TxD, en modo de un sólo hilo (*Simple Wire*) *half duplex* (LOOPS = RSRC = 1).
    - 0:** El pin TxD es una entrada en el modo de un solo hilo.
    - 1:** El pin TxD es una salida en el modo de un solo hilo.
  - **TXINV:** Bit para invertir el estado eléctrico de los bits en el pin TxD.
    - 0:** Los bits transmitidos no son invertidos.
    - 1:** Los bits transmitidos son invertidos.
  - **ORIE:** Bit para habilitar un evento de interrupción por *Receiver Overrun*.
    - 0:** Inhibe interrupción por *Receiver Overrun*.
    - 1:** Habilita interrupción por *Receiver Overrun*.
  - **NEIE:** Bit para habilitar un evento de interrupción por *Noise Error*.
    - 0:** Inhibe interrupción por *Noise Error*.
    - 1:** Habilita interrupción por *Noise Error*.
  - **FEIE:** Bit para habilitar un evento de interrupción por *Framming Error*.
    - 0:** Inhibe interrupción por *Framming Error*.
    - 1:** Habilita interrupción por *Framming Error*.
  - **PEIE:** Bit para habilitar un evento de interrupción por *Parity Error*.
    - 0:** Inhibe interrupción por *Parity Error*.
    - 1:** Habilita interrupción por *Parity Error*.
- **Registro de datos del SCI (SCIx D)** La Figura 16.14 detalla el registro de datos del SCI. Cuando este registro se lee, retorna el valor del dato que llegó por recepción. Lo que se escriba en este registro será transmitido. Ambas operaciones son independientes, es decir la una no afecta la otra.

#### Registro de datos (SCIx D)

	7	6	5	4	3	2	1	0
Lectura	R7	R6	R5	R4	R3	R2	R1	R0
Escritura	T7	T6	T5	T4	T3	T2	T1	T0
Reset	0	0	0	0	0	0	0	0
0x(FF)FF_803F	SCI1D	Bit 7	6	5	4	3	2	1 Bit 0
0x(FF)FF_8047	SCI2D	Bit 7	6	5	4	3	2	1 Bit 0

Figura 16.14. Registro SCIx D.

## 16.2. EJEMPLO CON EL MÓDULO SCI

El ejemplo a realizar trata sobre la comunicación de un PC, en ejecución del programa Hyperterminal, con el MCF51JM128.

El formato de la comunicación será de:

- *Full Duplex*
- Rata de baudios: 115000
- Longitud del carácter: 8
- Paridad: No
- Bits STOP: 1

El circuito de la Figura 16.15 ilustra el montaje necesario para el ejemplo. Nótese que es necesario la instalación de un *driver*, convertidor de los niveles de la lógica del microcontrolador a niveles RS232.

Todo carácter digitado desde el teclado del PC deberá aparecer en el LCD de la aplicación y a la vez, toda tecla digitada en el teclado de la aplicación deberá aparecer en la pantalla configurada por el Hyperterminal.

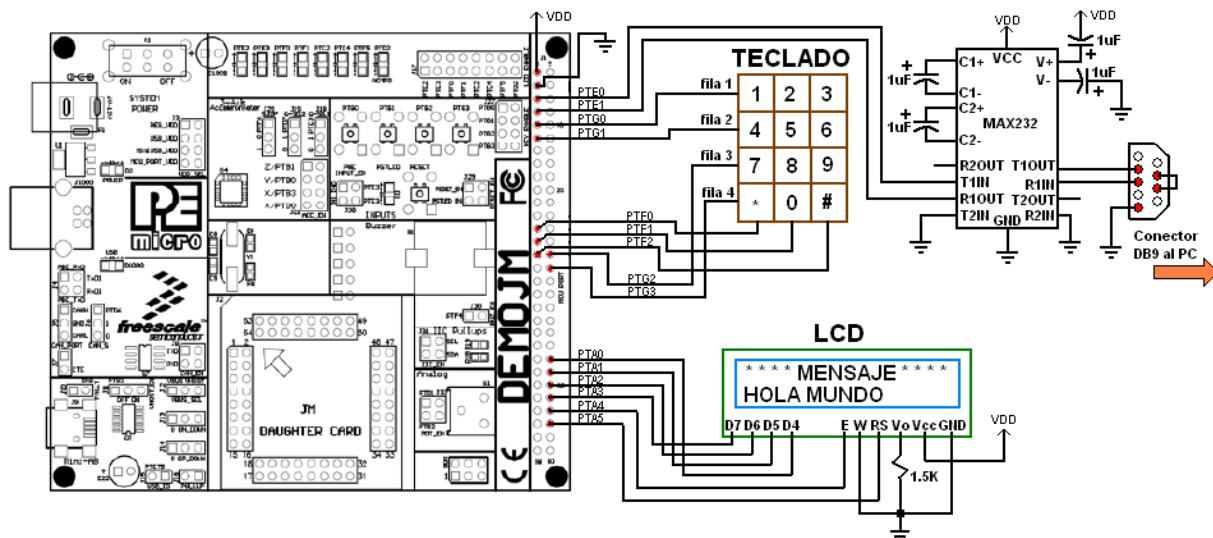


Figura 16.15. Circuito ejemplo SCI.

**NOTA:** Se recomienda al usuario revisar cuidadosamente el estado de los puentes del DEMOJM, con el objetivo de hacer compatible el ejercicio actual y de esta manera liberar los pines comprometidos en el circuito. En particular, en este ejercicio es necesario retirar los puentes en J4.

```
/*
 * SCI_PC: Ejemplo sobre la utilización del modulo SCI.
 */
/* main.c
 */
/* Fecha: Junio 1, 2008 */
```

```

/*
 * V1.0, Rev 0 por Diego Múnera
 */
/*
 * Asunto: Implementar un codigo en C que comunique un PC, vía Hyperterminal, con un MCF51JM */
/* 128 vía SCI1.Los caracteres que provienen desde el Hyperterminal deberan ser presen */
/* tados en un LCD y los caracteres que envía el MCU deberan aparecer en la pantalla */
/* del Hyperterminal.
*/
/*
 * Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros
 * editores.
*/
//********************************************************************

/*
 * DISTRIBUCION DE LOS PINES DE PUERTOS
*/
//********************************************************************

/*
 * PUERTO   * PIN   * I/O   * TIPO   * INICIA   * COMENTARIO
*/
//********************************************************************

/*PTA   * -   * -   * -   * -   *LCD
/*PTA0  * O   * D   * 1   *D4 del LCD
/*PTA1  * O   * D   * 1   *D5 del LCD
/*PTA2  * O   * D   * 1   *D6 del LCD
/*PTA3  * O   * D   * 1   *D7 del LCD
/*PTA4  * O   * D   * 1   *Señal ENABLE del LCD
/*PTA5  * O   * D   * 1   *Señal RS del LCD
//********************************************************************

/*PTB   * -   * -   * -   *No usado
*/
//********************************************************************

/*PTC   * -   * -   * -   *No usado
*/
//********************************************************************

/*PTD   * -   * -   * -   *No usado
*/
//********************************************************************

/*PTE   * -   * -   * -   *Puerto SCI
/*PTE0  * O   * D   * 1   *TxD
/*PTE1  * O   * D   * 1   *RxD
*/
//********************************************************************

/*PTF   * -   * -   * -   *No usado
/*PTF0  * O   * D   * 1   *Columna 1 teclado
/*PTF1  * O   * D   * 1   *Columna 2 teclado
/*PTF2  * O   * D   * 1   *Columna 3 teclado
*/
//********************************************************************

/*PTG   * -   * -   * -   *Teclado
/*PTG0  * I   * D   * 1   *Fila 1 del teclado
/*PTG1  * I   * D   * 1   *Fila 2 del teclado
/*PTG2  * I   * D   * 1   *Fila 3 del teclado
/*PTG3  * I   * D   * 1   *Fila 4 del teclado
*/
//********************************************************************

/*PTH   * -   * -   * -   *No usado
*/
//********************************************************************

/*PTJ   * -   * -   * -   *No usado
*/
//********************************************************************

/* Archivos de cabecera */
#include <hidef.h>           //macro para interrupciones
#include "derivative.h"         //macro de declaraciones del MCU

/* Declaracion de variables y constantes */
byte bandera0=0;                //bandera de propósito general
byte bandera1=0;                //bandera de propósito general
byte bandera2=0;                //bandera de propósito general
byte rx_dato=0;                 //variable temporal para dato por Rx
unsigned long retardo=0;         //parametro para hacer retardo
unsigned int i=0;                //variable de iteracion
char columna=0;                 //columna detectada del teclado
char fila=0;                    //fila detectada del teclado
char comando8=0;                //parametro inicializacion LCD modo 8 bits
char dato4;                     //dato a enviar al LCD

```

```

char rs=0;                                //señal de dato o comando al LCD
char a=0;                                  //variable temporal
char KBI1SC_Config=0;                      //byte de inicializacion del registro KBI1SC
char KBI1PE_Config=0;                      //byte de inicializacion del registro KBI1PE
char KBI1ES_Config=0;                      //byte de inicializacion del registro KBI1ES
char PTAD_Config=0;                        //byte de inicializacion del registro PTAD
char PTADD_Config=0;                        //byte de inicializacion del registro PTADD
char PTAPE_Config=0;                        //byte de inicializacion del registro PTAPE
char PTASE_Config=0;                        //byte de inicializacion del registro PTASE
char PTADS_Config=0;                        //byte de inicializacion del registro PTADS
char PTAIFE_Config=0;                       //byte de inicializacion del registro PTAIFE
char PTFD_Config=0;                         //byte de inicializacion del registro PTFD
char PTFDD_Config=0;                        //byte de inicializacion del registro PTFDD
char PTFPE_Config=0;                        //byte de inicializacion del registro PTFPE
char PTFSE_Config=0;                        //byte de inicializacion del registro PTFSE
char PTFDS_Config=0;                        //byte de inicializacion del registro PTFDS
char PTFIFE_Config=0;                       //byte de inicializacion del registro PTFIFE_
char SCI1C1_Config=0;                      //byte de inicializacion del registro SCI1C1
char SCI1C2_Config=0;                      //byte de inicializacion del registro SCI1C2
char SCI1BD_Config=0;                      //byte de inicializacion del registro SCI1BD
const unsigned char mensaje[] = "*****MENSAJE*****"; //Arreglo del mensaje para llevar al LCD
char *pM;                                  //puntero a mensaje para el LCD

/* Macros y definiciones */
#define Disable_COP() SOPT1 &= 0x3F //deshabilita el COP

/* Declaracion de funciones */
void KBI_Init(char KBI1SC_Config,char KBI1PE_Config,char KBI1ES_Config);           //declare funcion que inicializa KBI1
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config);          //declare funcion que inicializa PTA
void PTF_Init(char PTFD_Config,char PTFDD_Config,char PTFPE_Config,char PTFSE_Config,char PTFDS_Config,char PTFIFE_Config);           //declare funcion que inicializa PTF
void SCI1_Init(char SCI1C1_Config,char SCI1C2_Config,char SCI1BD_Config);           //declare funcion que inicializa SCI1
void Delay(unsigned long retardo);          //funcion para hacer retardos varios
void LCD_Init(void);                      //funcion para inicializar LCD
void Lcd_4bit_Wr(char dato4,char rs);     //funcion para escribir dato al LCD
void Comando_8bit(char comando8);          //funcion para inicio del LCD a 8 bits
void Rote_Col (void);                     //funcion para rotar un cero por columnas teclado
void Analisis_Tecla(void);                //funcion para analisis de tecla presionada

/* main(): Funcion principal */
void main(void) {

    // Inicializacion del reloj:
    MCGTRM = NVMCGTRM + 52;                 //Tome valor programado de fabrica del TRIM
    MCGC1 = 0x04;                           //Modo FEI divisor por 1
    MCGC2 = 0x00;                           //Desactiva modo de reloj externo
    MCGC4 = 0x22;                           //Rango alto del DCO
                                            //El reloj MCGOUT queda en 24999945 Hz
    Disable_COP();                          //deshabilita el COP
    EnableInterrupts();                     //habilita interrupciones
    PTA_Init(0x00,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "0",pines como salida, no pullups, si slew, no strength y si filtro
    PTF_Init(0xFF,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "1",pines como salida, no pullups, si slew, no strength y si filtro
    PTGPE=0xFF;                            //habilite pullups puerto G
    KBI_Init(0x06,0xC3,0x00);              //da ACK, habilita interr, habilita pullups y flanco de bajada
    LCD_Init();                            //inicialice LCD
    SCI1_Init(0x00,0x0C,0x0D);             //enciende transmisor y receptor,8:N:1, inhibe interr
    /SBR = (24999945/(16x115200)) = 14 = 0xD
    pM = (char *)mensaje;                  //Puntero toma direccion del mensaje con definicion
    for (i=0;i<15;i++){
        Lcd_4bit_Wr(*pM,1);              //Ciclo para imprimir 10 caracteres del mensaje
        pM++;
    }
    Lcd_4bit_Wr(0xC0,0);                  //Incrementa puntero al arreglo del mensaje
                                            //primer caracter por Rx a fila 2 columna 1 del LCD
}

```

```

SCI1C2_RIE=1;           //habilite interrupcion por Rx
SCI1D = 0x20;           //transmite un caracter vacio para abrir transmision
SCI1C2_TCIE = 1;        //habilite int por TXD 1
do{                     //espere a que se complete la TX
}while(bandera1 == 0);
bandera1 = 0;
i=0;
for(;;) {
    Rote_Col();
    if(bandera0==1){
        bandera0=0;
        Analisis_Tecla();
    }
    if(bandera2==1){
        bandera2=0;
        Lcd_4bit_Wr(rx_dato,1);
        i++;
        if(i==16){
            Lcd_4bit_Wr(0xC0,0);
            i=0;
        }
    }
/* es necesario asegurarse de nunca abandonar el main */
}

/* Funcion para inicializar el puerto A */
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config){
    PTAD=PTAD_Config;          //inicialice estado pinos PTA
    PTADD=PTADD_Config;        //inicialice direccion de pinos PTA
    PTAPE=PTAPE_Config;        //inicialice estado de pullups PTA
    PTASE=PTASE_Config;        //inicialice estado del slew rate PTA
    PTADS=PTADS_Config;        //inicialice estado de drive strength PTA
    PTAIFE=PTAIFE_Config;      //inicialice estado del filtro PTA
}

/* Funcion para inicializar el puerto F */
void PTF_Init(char PTFD_Config,char PTFDD_Config,char PTFPE_Config,char PTFSE_Config,char PTFDS_Config,char PTFIFE_Config){
    PTFD=PTFD_Config;          //inicialice estado pinos PTF
    PTFDD=PTFDD_Config;        //inicialice direccion de pinos PTF
    PTFPE=PTFPE_Config;        //inicialice estado de pullups PTF
    PTFSE=PTFSE_Config;        //inicialice estado del slew rate PTF
    PTFDS=PTFDS_Config;        //inicialice estado de drive strength PTF
    PTFIFE=PTFIFE_Config;      //inicialice estado del filtro PTF
}

/* Funcion para inicializar el KBI */
void KBI_Init(char KBI1SC_Config,char KBI1PE_Config,char KBI1ES_Config){
    KBI1PE=KBI1PE_Config;      //inicialice registro de habilitacion de pinos KBI
    KBI1SC_KBACK=1;            //aclara interrupciones espureas
    KBI1ES=KBI1ES_Config;      //inicialice registro de flanco y nivel del KBI
    KBI1SC=KBI1SC_Config;      //inicialice registro de estado y control del KBI
}

/* Funcion para inicializar el SCI1 */
void SCI1_Init(char SCI1C1_Config,char SCI1C2_Config,char SCI1BD_Config){
    SCI1C1=SCI1C1_Config;      //inicialice registro SCI1C1
    SCI1C2=SCI1C2_Config;      //inicialice registro SCI1C2
    SCI1BD=SCI1BD_Config;      //inicialice registro SCI1BD
}

/* Funcion para inicializacion del LCD a 4 bits */
void LCD_Init(void){
    Delay(200000);             //hace retardo
    Comando_8bit(0x30);        //llama funcion enviar comando en 8 bits con comando
    Delay(200000);             //hace retardo
    Comando_8bit(0x30);        //llama funcion enviar comando en 8 bits con comando
    Delay(200000);             //hace retardo
}

```

```

Comando_8bit(0x20);           //LCD 2x16 y manejo a 4 bits
Lcd_4bit_Wr(0x28,0);          //LCD 2x16 y manejo a 4 bits
Lcd_4bit_Wr(0x0C,0);          //ON LCD, OFF cursor
Lcd_4bit_Wr(0x01,0);          //borrar pantalla LCD
Lcd_4bit_Wr(0x06,0);          //desplaza cursor a la derecha con cada caracter
Lcd_4bit_Wr(0x80,0);          //primer caracter fila 1 columna 1
}

/* Funcion para escribir a 4 bits en el LCD */
void Lcd_4bit_Wr(char dato4,char rs){
    if (rs==0){                //pregunte si se va a enviar un comando o un dato
        PTAD_PTAD5=0;           //prepara envío de comando RS = 0
    }
    else{
        PTAD_PTAD5=1;           //prepara envio de dato RS = 1
    }
    a=dato4;                   //almacena temporalmente el dato
    a=a>>4;                   //desplaza parte alta dato para la baja
    PTAD&=0xF0;                //enmascara parte alta del puerto A
    a&=0x0F;                   //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;               //sube pin de enable
    Delay(20000);              //hace retardo
    PTAD|=a;                   //envía nibble alto del dato
    Delay(20000);              //hace retardo
    PTAD_PTAD4=0;               //baja pin de enable
    Delay(20000);              //hace retardo
    PTAD&=0xF0;                //enmascara parte alta del puerto A
    dato4&=0x0F;                //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;               //sube pin de enable
    Delay(20000);              //hace retardo
    PTAD|=dato4;               //envía nibble bajo del dato
    Delay(20000);              //hace retardo
    PTAD_PTAD4=0;               //baja pin de enable
    Delay(20000);              //hace retardo
}

/* Función para envio de comando en 8 bits al LCD (solo para inicio) */
void Comando_8bit(char comando8){
    PTAD_PTAD5=0;               //prepara envio de comando
    PTAD&=0xF0;                 //enmascara parte baja del puerto A
    comando8&=0xF0;              //enmascara parte alta de comando a enviar al LCD
    comando8=comando8>>4;       //desplaza para tomar nibble alto
    PTAD_PTAD4=1;               //sube pin de enable
    Delay(20000);               //hace retardo
    PTAD|=comando8;             //envía comando
    Delay(20000);               //hace retardo
    PTAD_PTAD4=0;               //sube pin de enable
    Delay(20000);               //hace retardo
}

/* Funcion Delay(): Retarda basado en una variable tipo entera */
void Delay(unsigned long retardo){
    while(retardo>0){           //llego a cero?
        retardo--;                //no --> decrementa
    }
}

/* Funcion para rotar cero por columnas teclado */
void Rote_Col (void){
    PTFD = 0x0E;                //Cero a la columna 1
    Delay(500000);              //hace retardo
    PTFD = 0x0D;                //Cero a la columna 2
    Delay(500000);              //hace retardo
    PTFD = 0x0B;                //Cero a la columna 3
    Delay(500000);              //hace retardo
}

/* Funcion para analizar tecla presionada y enviarla al PC */
void Analisis_Tecla(void){
}

```

```

if(columna==0x0E & fila==1){           //si se detecta columna 1 y fila 1
    SCI1D = 0x31;                      //transmite un 1
    SCI1C2_TCIE = 1;                   //habilite int por TxD 1
    do{                                //espere a que se complete la TX
        }while(bandera1 == 0);
        bandera1 = 0;
    }
if(columna==0x0D & fila==1){           //si se detecta columna 2 y fila 1
    SCI1D = 0x32;                      //transmite un 2
    SCI1C2_TCIE = 1;                   //habilite int por TxD 1
    do{                                //espere a que se complete la TX
        }while(bandera1 == 0);
        bandera1 = 0;
    }
if(columna==0x0B & fila==1){           //si se detecta columna 3 y fila 1
    SCI1D = 0x33;                      //transmite un 3
    SCI1C2_TCIE = 1;                   //habilite int por TxD 1
    do{                                //espere a que se complete la TX
        }while(bandera1 == 0);
        bandera1 = 0;
    }
if(columna==0x0E & fila==2){           //si se detecta columna 1 y fila 2
    SCI1D = 0x34;                      //transmite un 4
    SCI1C2_TCIE = 1;                   //habilite int por TxD 1
    do{                                //espere a que se complete la TX
        }while(bandera1 == 0);
        bandera1 = 0;
    }
if(columna==0x0D & fila==2){           //si se detecta columna 2 y fila 2
    SCI1D = 0x35;                      //transmite un 5
    SCI1C2_TCIE = 1;                   //habilite int por TxD 1
    do{                                //espere a que se complete la TX
        }while(bandera1 == 0);
        bandera1 = 0;
    }
if(columna==0x0B & fila==2){           //si se detecta columna 3 y fila 2
    SCI1D = 0x36;                      //transmite un 6
    SCI1C2_TCIE = 1;                   //habilite int por TxD 1
    do{                                //espere a que se complete la TX
        }while(bandera1 == 0);
        bandera1 = 0;
    }
if(columna==0x0E & fila==3){           //si se detecta columna 1 y fila 3
    SCI1D = 0x37;                      //transmite un 7
    SCI1C2_TCIE = 1;                   //habilite int por TxD 1
    do{                                //espere a que se complete la TX
        }while(bandera1 == 0);
        bandera1 = 0;
    }
if(columna==0x0D & fila==3){           //si se detecta columna 2 y fila 3
    SCI1D = 0x38;                      //transmite un 8
    SCI1C2_TCIE = 1;                   //habilite int por TxD 1
    do{                                //espere a que se complete la TX
        }while(bandera1 == 0);
        bandera1 = 0;
    }
if(columna==0x0B & fila==3){           //si se detecta columna 3 y fila 3
    SCI1D = 0x39;                      //transmite un 9
    SCI1C2_TCIE = 1;                   //habilite int por TxD 1
    do{                                //espere a que se complete la TX
        }while(bandera1 == 0);
        bandera1 = 0;
    }
if(columna==0x0D & fila==4){           //si se detecta columna 1 y fila 4
    SCI1D = 0x30;                      //transmite un 0
    SCI1C2_TCIE = 1;                   //habilite int por TxD 1
    do{                                //espere a que se complete la TX
        }while(bandera1 == 0);
        bandera1 = 0;
    }
}

```

```

Delay(1000000);                                //hace retardo para lectura de teclado
}

/* Funcion de atencion a la interrupcion del KBI1 */
interrupt VectorNumber_Vkeyboard void ISR_KBI (void){
    columna=PTFD & 0x0F;                      //capture el valor de la columna
    if(PTGD_PTGD0==0){                         //si detecto la fila 1
        fila=1;                                //marque fila 1
    }
    if(PTGD_PTGD1==0){                         //si detecto la fila 2
        fila=2;                                //marque fila 2
    }
    if(PTGD_PTGD2==0){                         //si detecto la fila 3
        fila=3;                                //marque fila 3
    }
    if(PTGD_PTGD3==0){                         //si detecto la fila 4
        fila=4;                                //marque fila 4
    }
    KBI1SC_KBACK = 1;                          //de reconocimiento de tecla presionada
    bandera0=1;                               //pone bandera de tecla presionada
}

/* Funcion de atencion a la interrupcion por Tx */
interrupt VectorNumber_Vsci1tx void ISR_SCI1_TX1 (void){
    SCI1S1;                                  //reconoce la interrupción
    SCI1C2_TCIE=0;                           //inhibe interrupciones por Tx
    bandera1= 1;                             //pone bandera de caracter enviado
}

/* Funcion de atencion a la interrupcion por Rx */
interrupt VectorNumber_Vsci1rx void ISR_SCI1_RX1 (void){
    SCI1S1;                                  //reconoce la interrupción
    rx_dato = SCI1D;                          //lea caracter que entro
    bandera2= 1;                            //pone bandera de caracter recibido
}

```

### **16.3. REFERENCIAS**

- MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual. Freescale Semiconductor. Rev 1, 2008.
- Microcontroller MCF51JM128 ColdFire Data Sheet. Freescale Semiconductor. Rev 0, 2008.

### **16.4. PREGUNTAS**

- ¿Cómo es una señal del tipo NRZ (haga una gráfica)?
- Estabezca las diferencia entre los modos *simplex*, *duplex* y *full-duplex*.
- ¿Qué es modo IDLE?
- ¿De cuántas fuentes de reloj se dispone para obtener la rata de baudios de una comunicación serial con SCI?
- ¿Cuántas maneras de tamaño de dato existen en el modo SCI y cómo se seleccionan?
- ¿Cómo se calcula la rata de baudios en una comunicación serial con SCI?
- ¿Cómo se puede verificar, por software, si una comunicación vía SCI es efectiva hasta los pines de salida del MCU?
- Implementar un circuito y desarrollar un programa para comunicar dos sistemas DEMOJM vía SCI, a una razón de 4800 baudios, 8 bits de dato, paridad par y un bit de stop.

# CAPÍTULO 17

## Comunicaciones Seriales Sincrónicas (SPI: *Serial Peripheral Interface*) (IIC: *Inter. Integrated Circuit*)

- 17.1. Diagrama en bloques, explicación del funcionamiento y registros asociados al módulo SPI
- 17.2. Diagrama en bloques, explicación del funcionamiento y registros asociados al módulo IIC
- 17.3. Ejemplos con los módulos SPI e IIC
  - Comunicación SPI entre dos sistemas MCU
  - Lectura y escritura sobre una memoria serial IIC
- 17.4. Referencias
- 17.5. Ejercicios y preguntas

## 17.1. DIAGRAMA EN BLOQUES, EXPLICACIÓN DEL FUNCIONAMIENTO Y REGISTROS ASOCIADOS AL MÓDULO SPI

- **Breve repaso de las comunicaciones seriales sincrónicas**

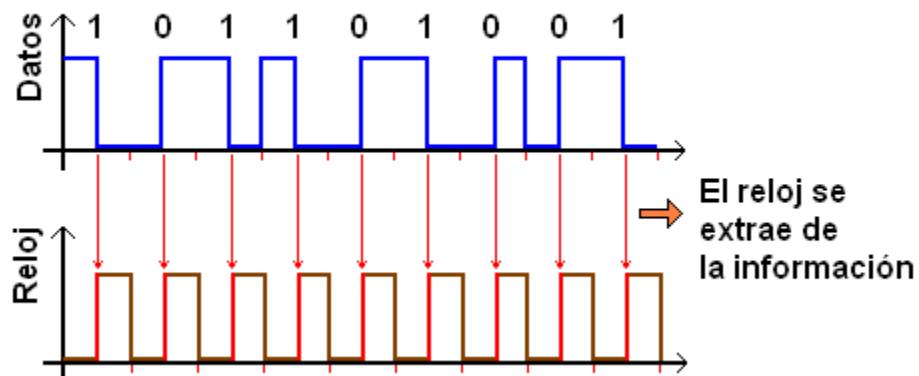
Una comunicación sincrónica es aquella en donde el reloj viaja con la información, ya sea como un hilo independiente o embebido dentro de la misma.

Generalmente existe un dispositivo maestro, que es el generador de la sincronía de la comunicación. De tal manera que el reloj es generado en un hilo independiente del sistema y es el maestro quien lo presenta en el canal de comunicación.

Los demás dispositivos del sistema actúan como esclavos de la comunicación y la muestra de reloj entra por un pin para tal propósito, estableciendo la sincronía de los bits de información que llegan o salen.

Otros sistemas utilizan la misma señal de datos, para generar la sincronía de los bits. La Figura 17.1 ilustra el protocolo Manchester, del cual se puede extraer el reloj del sistema.

Basta con tomar cualquier flanco de la señal para y de allí derivar (utilizando los cambios de flanco) el reloj de los datos.



**Figura 17.1. Protocolo Manchester y reloj sincrónico embebido.**

En los sistemas seriales sincrónicos para la interconexión de periféricos, se manejan distancias cortas y modos de un solo maestro y múltiples esclavos.

Un maestro podría perder su función y convertirse en un esclavo, mediante un mecanismo de arbitramiento de bus. De esta manera cualquier dispositivo podría ser maestro en un momento determinado.

Las tramas de comunicación son de variada presentación y algunas sólo involucran la carga útil, sin adicionar bits que permitan el entendimiento (*handshaking*) entre esclavos y maestro. Para estos protocolos deberá existir una línea tanto para el dato saliente (Tx D)

como para el dato entrante (RxD), así como la posibilidad de hacer comunicación simultánea y bidireccional (*Full Duplex*) (Ejemplo: SPI).

En otros casos podría hacerse la transferencia de datos por un solo hilo, pero deberían existir bits adicionales en la trama de datos, de tal manera que sincronicen la dirección de la información. Estos sistemas sólo podrían configurarse en modo *Half Duplex* (Ejemplo: IIC).

- **Breve descripción del módulo SPI y diagrama en bloques**

Las características más importantes del módulo SPI son:

- Operación como maestro o como esclavo.
- Comunicación *Full Duplex* o por un sólo hilo (*Single Wire*).
- Rata de transmisión programable.
- Registro de datos del transmisor y receptor, separados (*double buffer*).
- Protocolo NRZ con opción de seleccionar la fase y la polaridad del reloj.
- Salida para selección del esclavo.
- Error por modo equivocado de operación (cuando un esclavo se va a configurar como maestro y no está permitido).
- Operación en estado de WAIT.
- Selección de transmitir primero el MSb o el LSb.
- Longitud del dato programable, en 8 o 16 bits.
- Posibilidad de una lista FIFO de 64 bits, para alta velocidad y gran cantidad de datos en la transferencia.

La Figura 17.2 detalla el diagrama en bloques del módulo SPI, en donde el circuito (A) representa el corazón de la manipulación de los datos. El circuito está formado por el shift register para los bits de entrada y salida y los buffer FIFO para el transmisor y el receptor.

El usuario puede establecer si el dato a manipular será de un ancho de 8 o 16 bits y si establece un buffer de hasta 64 bits para la transmisión y recepción de la información.

En el circuito (B) se configura la manipulación de los pines involucrados en la comunicación SPI, que a continuación serán definidos:

- **MOSI/MOMI (Master Output Slave Input/Master Output Master Input):** Pin de transmisión desde el punto de vista del maestro y de recepción desde el punto de vista del esclavo, para cuando se está en modo *Full Duplex*. Pero se convierte en un pin bidireccional, para cuando se está en modo *Single Wire* y el dispositivo es un maestro.
- **MISO/SISO (Master Input Slave Output/Slave Input Slave Output):** Pin de recepción desde el punto de vista del maestro y de transmisión desde el punto de vista del esclavo, para cuando se está en modo *Full Duplex*. Pero se convierte en

un pin bidireccional, para cuando se está en modo *Single Wire* y el dispositivo es un esclavo.

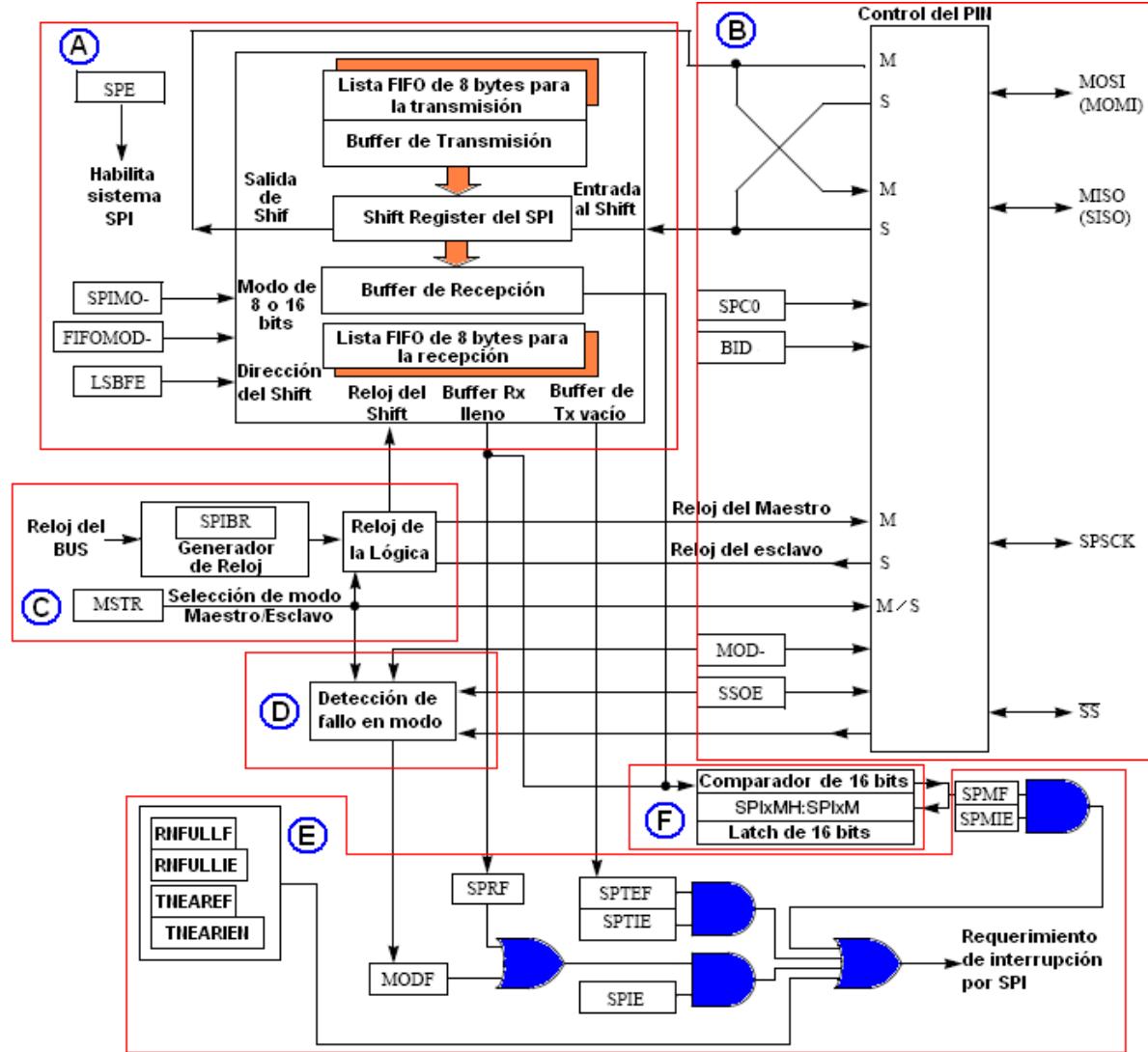
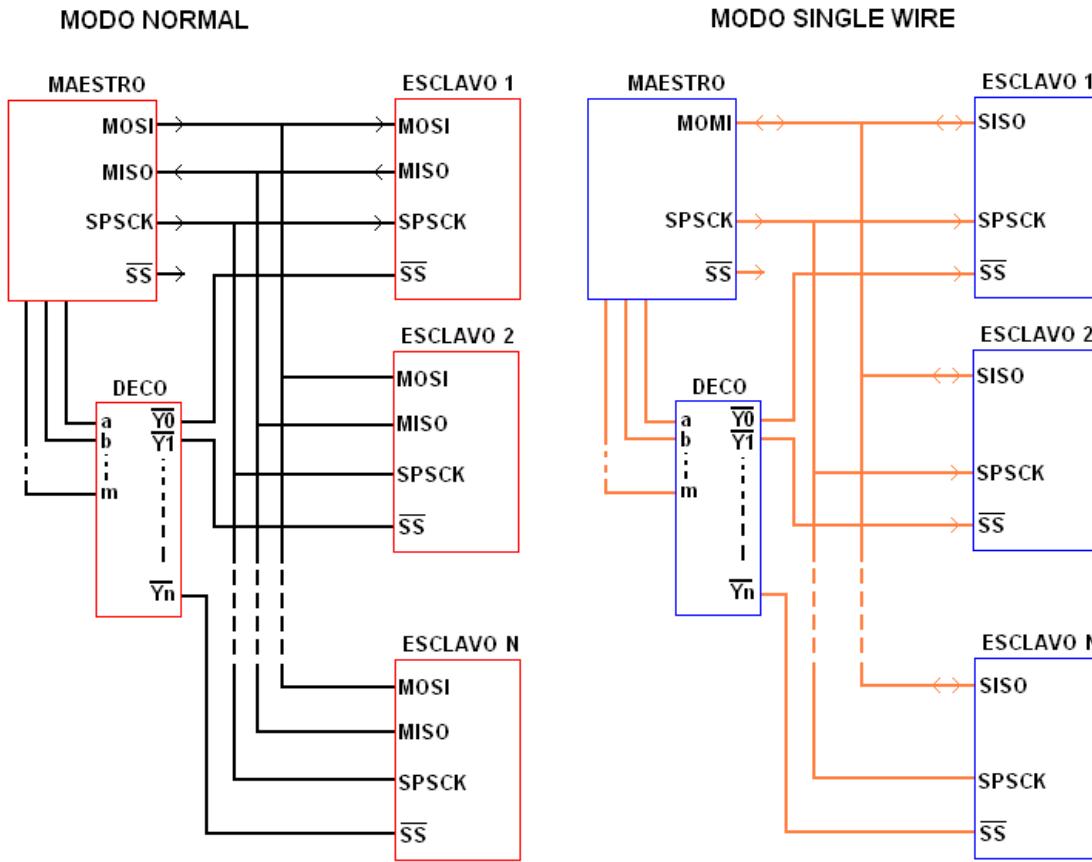


Figura 17.2. Diagrama en bloque del módulo SPI.

- **SPSCK (SPI Serial Clock):** Pin de señal de reloj sincrónico. Este reloj es servido por el maestro y se configura como entrada al esclavo.
- **!SS (Slave Select):** Pin para seleccionar, ubicando un nivel bajo, el esclavo vía hardware.

La Figura 17.3 ilustra sobre una conexión típica de un maestro y varios esclavos en los modos normal y *single wire*. El decodificador es manejado por el dispositivo maestro y esto garantiza que sólamente un esclavo es seleccionado al tiempo.



**Figura 17.3. Conexión de un maestro y múltiples esclavos.**

El circuito (C) corresponde con el reloj del sistema, el cual es derivado del reloj de BUS del MCU. Si el dispositivo fue configurado como maestro, este circuito se activa y es usado como reloj de referencia para la comunicación SPI (SPSCK).

Si un esclavo intenta ser maestro y no está habilitado para esto, el sistema puede dar cuenta de ello y generar un evento de error por falla en el modo (*Mode Fault Error*). Esta labor la ejecuta el circuito (D). El fallo por modo se puede dar por:

- Cambio en la definición de modo.
- Cambio en la definición del pin !SS.
- Cambio en el nivel del pin !SS, por pasar de alto a bajo.

El circuito (E) configura la máquina de estados para el sistema de interrupciones del módulo SPI. Existen cuatro posibles fuentes de interrupción y son:

- Interrupción por detección en el modo (MODF). Se presenta cuando el maestro detecta un problema en el pin !SS.
- Interrupción por detección de dato almacenado en el registro de datos de SPI (SPIxD).
- Interrupción por buffer de transmisión vacío (SPTEF).
- Cuando el dato en el buffer de recepción es igual al dato escrito en el registro de comparación (SPIxM: Match Register), se produce una interrupción por coincidencia (SPMF).

El circuito (F) se configura como un comparador de datos que llegan al SPI y puede ser utilizado por el usuario para validar información, como por ejemplo la validación de una dirección en un sistema de múltiples esclavos.

- **Registros asociados al módulo SPI**

- **Registros de control 1 (SPIxC1):** La Figura 17.4 muestra el registro de configuración 1 del módulo SPI.

**Registro de control 1 (SPIxC1)**

Lectura Escritura	7	6	5	4	3	2	1	0
	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
Reset	0	0	0	0	0	1	0	0
0x(FF)FF_8050	SPI1C1	SPI1E	SP1E	SP1TIE	MSTR	CPOL	CPHA	SSOE
0x(FF)FF_8070	SPI2C1	SPI2E	SP2E	SP2TIE	MSTR	CPOL	CPHA	LSBFE

**Figura 17.4. Registros SPIxC1.**

- **SPIE:** Bit para habilitar un posible evento de interrupción por evento de *buffer* de recepción lleno (SPRF) o por fallo de modo (MODF).
  - Para modo FIFO = 0**
  - 0:** No habilita interrupción por evento SPRF o MODF
  - 1:** Habilita interrupción por evento SPRF o MODF
- **Para modo FIFO = 1**
- 0:** Deshabilita interrupción por *buffer* FIFO de recepción lleno
  - 1:** Habilita interrupción por *buffer* FIFO de recepción lleno
- **SPE:** Bit para habilitar el funcionamiento del módulo SPI.
  - 0:** Desactiva el módulo SPI
  - 1:** Activa el módulo SPI

- **SPTIE:** Bit para habilitar un evento de interrupción por *buffer* de transmisión vacío (SPTEF).
   
**Para modo FIFO = 0**
  - 0:** No habilita interrupción por evento SPTEF
  - 1:** Habilita interrupción por evento SPTEF  
**Para modo FIFO = 1**
  - 0:** Deshabilita interrupción por *buffer* FIFO de transmisión vacío
  - 1:** Habilita interrupción por *buffer* FIFO de transmisión vacío
- **MSTR:** Bit para habilitar el funcionamiento del módulo SPI como maestro.
   
**0:** El módulo SPI se configura como esclavo
   
**1:** El módulo SPI se configura como maestro
- **CPOL:** Bit para seleccionar la polaridad del reloj SPSCK, que se presenta como primer flanco en el ciclo de reloj del dato a transferir.
   
**0:** El reloj del SPI es activo en alto (en modo vago permanece en bajo)
   
**1:** El reloj del SPI es activo en bajo (en modo vago permanece en alto)
- **CPHA:** Bit para seleccionar uno de dos posibles formatos de reloj, dependiendo del tipo de periférico a comunicar.
   
**0:** El primer flanco de la señal SPSCK ocurre en la mitad del primer ciclo del dato a transferir.
   
**1:** El primer flanco de la señal SPSCK ocurre en el inicio del primer ciclo del dato a transferir.
- **SSOE:** Bit para habilitar la señal !SS, como selección del esclavo. Es usado en combinación con los bits SPIxC2[MODFEN] y SPIxC1[MSTR], para establecer las condiciones del pin !SS. La Tabla 19.1 configura las diferentes opciones para el pin.

**Tabla 17.1. Selección del pin !SS.**

MODFEN	SSOE	Modo Maestro	Modo Esclavo
0	0	Pin de propósito general	Entrada de selección del esclavo
0	1	Pin de propósito general	Entrada de selección del esclavo
1	0	Pin SS como entrada falla modo	Entrada de selección del esclavo
1	1	Pin SS como salida automática	Entrada de selección del esclavo

- **LSBFE:** Bit para seleccionar primer bit a transmitir de un dato.
   
**0:** Primer bit a transmitir es el MSb
   
**1:** Primer bit a transmitir es el LSb
- **Registros de control 2 (SPIxC2)** La Figura 17.5 muestra el registro de configuración 2 del módulo SPI.

### Registro de configuración 2 (SPIxC2)

	7	6	5	4	3	2	1	0
Lectura Escritura	SPMIE	SPI MODE	0	MODFEN	BIDIROE	0	SPI SWAI	SPC0
Reset	0	0	0	0	0	0	0	0
0x(FF)FF_8051	SPI1C2	SPMIE	SPI MODE	0	MODFEN	BIDIROE	0	SPI1SWAI SP1C0
0x(FF)FF_8071	SPI2C2	SPMIE	SPI MODE	0	MODFEN	BIDIROE	0	SPI1SWAI SP1C0

Figura 17.5. Registros SPIxC2.

- **SPMIE:** Bit para habilitar un posible evento de interrupción por evento de coincidencia en la comparación del dato reciente por recepción y el contenido del registro SPIxM (*Match Register*).
  - 0:** Inhibe interrupción por SPIxM
  - 1:** Habilita interrupción por SPIxM
- **SPI MODE:** Bit para seleccionar la longitud del dato, entre 8 y 16 bits. Un cambio en este bit durante una transmisión, aborta la misma y forza al SPI a entrar en modo vago (IDLE: estado de alta impedancia en el bus).
  - 0:** Selección de dato, registro de comparación (SPIxM) y tamaño de los registros de *buffer* en 8 bits.
  - 1:** Selección de dato, registro de comparación (SPIxM) y tamaño de los registros de *buffer* en 16 bits.
- **MODFEN:** Bit para habilitar el fallo por modo (sólo para modo maestro).
  - 0:** Deshabilita el fallo por modo y el pin !SS del maestro pasa a ser un pin de propósito general.
  - 1:** Habilita el fallo por modo y el pin !SS del maestro pasa a ser un pin entrada de fallo por modo o como un pin de salida de selección de esclavo.
- **BIDIROE:** Bit para habilitar modo bidireccional de salida. Dependiendo si el bit SPC0 es “0” o “1” (ver Tabla 17.2).
- **SPI SWAI:** Bit para habilitar hacer que el SPI pare en modo WAIT.
  - 0:** El reloj del SPI continúa trabajando en modo WAIT.
  - 1:** El reloj del SPI para mientras se encuentra trabajando en modo WAIT.
- **SPC0:** Bit para habilitar operación bidireccional, según Tabla 17.2.
  - 0:** El SPI usa pines separados para dato de entrada y dato de salida.
  - 1:** El SPI queda configurado para operación bidireccional, *single wire*.

**Tabla 17.2. Modo bidireccional.**

Modo del PIN	SPC0	BIDIROE	MISO	MOSI
<b>Operación bajo modo Maestro</b>				
Normal	0	X	Entrada al Maestro	Salida del Maestro
Bidireccional	1	0	Pin MISO no usado por el SPI	Entrada al Maestro
		1		I/O Maestro
<b>Operación bajo modo Esclavo</b>				
Normal	0	X	Salida del Esclavo	Entrada al Esclavo
Bidireccional	1	0	Entrada al Esclavo	Pin MOSI no usado por el SPI
		1	Esclavo I/O	

- **Registros rata de baudios (SPIxBR)** La Figura 17.6 ilustra sobre el registro de configuración de la rata de baudios del módulo SPI (ver Tablas 17.3 y 17.4).

#### Registro de configuración de la rata de baudios (SPIxBR)

Lectura	7	6	5	4	3	2	1	0	
Escritura	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0	
Reset	0	0	0	0	0	0	0	0	
0x(FF)FF_8052	SPI1BR	0	SP1PR2	SP1PR1	SP1PR0	0	SP1R2	SP1R1	SP1R0
0x(FF)FF_8072	SPI2BR	0	SP2PR2	SP2PR1	SP2PR0	0	SP2R2	SP2R1	SP2R0

**Figura 17.6. Registros SPIxBD.**

- **SPPRn:** Bits para configurar un *prescaler* de la rata de baudios del SPI.
- **SPRn:** Bits para configurar un divisor de la rata de baudios del SPI.

**Tabla 17.3. Preescaler reloj SPI.**

SPPR2:SPPR1:SPPR0	Prescaler Divisor
0:0:0	1
0:0:1	2
0:1:0	3
0:1:1	4
1:0:0	5
1:0:1	6
1:1:0	7
1:1:1	8

**Tabla 17.4. Divisor reloj SPI.**

SPR2:SPR1:SPR0	Divisor de rata
0:0:0	2
0:0:1	4
0:1:0	8
0:1:1	16
1:0:0	32
1:0:1	64
1:1:0	128
1:1:1	256

- **Registros de estado (SPIxS)** La Figura 17.7 ilustra sobre el registro de estado del módulo SPI. El usuario debe saber que sólo el SPI2 soporta modo FIFO.

#### Registro de estado (SPIxS)

	7	6	5	4	3	2	1	0
Lectura	SPRF	SPMF	SPTEF	MODF	RNFULLF	TNEAREF	TXFULLF	RFIFOEOF
Escritura								
Reset	0	0	1	0	0	0 <sup>1</sup>	0	0 <sup>1</sup>
0x(FF)FF_8053	SPI1S	SP1RF	0	SP1TEF	MODF	0	0	0
0x(FF)FF_8073	SPI2S	SP2RF	SP2MF	SP2TEF	MODF	RNFULLF	TNEAREF	TXFULLF
								RFIFOEOF

**Figura 17.7. Registros SPIxS.**

- **SPxRF:** Bandera que indica cuando el *buffer* de Rx está lleno y un dato puede ser tomado del registro SPIxD. Esta bandera se aclara cuando se lee en su estado de “1” y luego se debe leer el registro de datos SPIxD.

#### Para el modo normal:

**0:** No hay dato disponible en el registro SPIxD.

**1:** Hay dato disponible en el registro SPIxD.

**Para el modo FIFO:** Cuando el buffer FIFO (64 bits) está lleno, la bandera SPxRF se pone a “1”. Esta bandera es aclarada cuando se lee el registro SPIxD, es decir desocupando el *buffer* FIFO.

**0:** El *buffer* FIFO de lectura no está lleno.

**1:** El *buffer* FIFO de lectura está lleno.

- **SP2MF:** Bandera que indica cuando se ha alcanzado una comparación (*Match*) entre el registro de comparación SPIxM y el *buffer* de recepción. Para aclarar el estado de esta bandera es necesario leerla estando en “1” y luego escribir un “1” sobre esta.

**0:** El valor en *buffer* de entrada no ha alcanzado el valor en el registro SPIxM.

**1:** El valor en *buffer* de entrada ha alcanzado el valor en el registro SPIxM.

- **SPxTEF:** Bandera que indica cuando el *buffer* de Tx está vacío. Esta bandera es aclarada cuando se lee el registro SPIxS estando la bandera en “1” y luego escribiendo un nuevo dato a transmitir en el registro de datos SPIxD.

**0:** El *buffer* de transmisión no está vacío.

**1:** El *buffer* de transmisión está vacío.

**Para el modo FIFO:** Cuando el buffer FIFO (64 bits) está vacío, la bandera SPxTEF se pone a “1”. Esta bandera es aclarada cuando se escribe el segundo dato de una tanda nueva a transmitir, sobre el *buffer* FIFO.

**0:** El *buffer* FIFO no está vacío.

**1:** El *buffer* FIFO está vacío.

- **MODF:** Bandera que se pone a “1” cuando el SPI es un maestro y el pin !SS va al estado bajo, indicando que hay otro maestro tomándose el bus. Esta bandera es aclarada cuando, estando en “1”, se hace una lectura sobre el registro SPIxS.

**0:** No ha ocurrido un error por fallo en el modo.

**1:** Ha ocurrido un error por fallo en el modo.

- **RNFULLF:** Bandera que se pone a “1” cuando el *buffer* FIFO de recepción está cercano a llenarse (han llegado 6 bytes).

**0:** El *buffer* FIFO de recepción ha recibido menos de 6 bytes.

**1:** El *buffer* FIFO de recepción ha recibido 6 bytes o más.

- **TNEAREF:** Bandera que se pone a “1” cuando el *buffer* FIFO de transmisión está cercano a desocuparse (faltan 2 bytes).

**0:** El *buffer* FIFO de transmisión está a más de dos bytes en desocuparse.

**1:** El *buffer* FIFO de transmisión está a 2 bytes de desocuparse.

- **TXFULLF:** Bandera que se pone a “1” cuando el *buffer* FIFO de transmisión está lleno.

**0:** El *buffer* FIFO de transmisión tiene menos de 8 bytes.

**1:** El *buffer* FIFO de transmisión está lleno.

- **RFIFOEOF:** Bandera que se pone a “1” cuando el *buffer* FIFO de recepción está vacío.

**0:** El *buffer* FIFO de recepción tiene uno o más bytes.

**1:** El *buffer* FIFO de recepción está vacío.

- **Registros de datos (SPIxD)** La Figura 19.8 muestra el registro de datos del módulo SPI. Son a la vez registro de entrada y salida de datos. Si un dato recibido no es leído en su momento y otro dato ha llegado, se genera un evento de *receiver overrun* y el último dato se pierde. En modo de 8 bits, el registro alto del dato (SPIxDH) tiene un valor de 0 y escribir en este no tiene ningún efecto.

#### Registros de datos (SPIxDH: SPIxDL)

Lectura Escritura	7	6	5	4	3	2	1	0	
	Bit 15	14	13	12	11	10	9	Bit 8	
Reset	0	0	0	0	0	0	0	0	
	7	6	5	4	3	2	1	0	
Lectura Escritura	Bit 7	6	5	4	3	2	1	Bit 0	
Reset	0	0	0	0	0	0	0	0	
0x(FF)FF_8054	SPI1DH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8055	SPI1DL	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8074	SPI2DH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8075	SPI2DL	Bit 7	6	5	4	3	2	1	Bit 0

Figura 17.8. Registros SPIxD.

- **Registros de coincidencia (Match) (SPIxM)** La Figura 17.9 muestra el registro de coincidencia, el cual es comparado con el *buffer* de recepción y en caso tal de coincidir, la bandera SPMF es puesta a “1”.

#### Registros de coincidencia (SPIxMH:SPIxML)

Lectura Escritura	7	6	5	4	3	2	1	0	
	Bit 15	14	13	12	11	10	9	Bit 8	
Reset	0	0	0	0	0	0	0	0	
	7	6	5	4	3	2	1	0	
Lectura Escritura	Bit 7	6	5	4	3	2	1	Bit 0	
Reset	0	0	0	0	0	0	0	0	
0x(FF)FF_8056	SPI1MH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8057	SPI1ML	Bit 7	6	5	4	3	2	1	Bit 0
0x(FF)FF_8076	SPI2MH	Bit 15	14	13	12	11	10	9	Bit 8
0x(FF)FF_8077	SPI2ML	Bit 7	6	5	4	3	2	1	Bit 0

Figura 17.9. Registros SPIxM.

- **Registro de control 3 (SPI2C3)** La Figura 17.10 muestra el registro de configuración 3 del módulo SPI, que sólo opera para el SPI2. Registro dedicado a

la configuración del modo FIFO, función que potencializa el módulo SPI para soportar grandes velocidades y rutas de datos

#### Registro de control 3 (SPIxC3) : (FFFF8078)

	7	6	5	4	3	2	1	0
Lectura	0	0	TNEAREF MARK	RNFULL MARK	0	TNEARIEN	RNFULLIEN	FIFOMODE
Escritura	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0

Figura 17.10. Registros SPIxC3.

- **TNEAREFMARK:** Bit para poner la marca en la cual el *buffer* de transmisión se aproxima a estar vacío.  
 0: La bandera TNEAREF es puesta a “1” cuando el *buffer* de transmisión tiene 16 bits o menos.  
 1: La bandera TNEAREF es puesta a “1” cuando el *buffer* de transmisión tiene 32 bits o menos.
- **RNFULLMARK:** Bit para poner la marca en la cual el *buffer* de recepción se considera lleno.  
 0: La bandera RNFULLF es puesta a “1” cuando el *buffer* de recepción tiene 48 bits o más.  
 1: La bandera RNFULLF es puesta a “1” cuando el *buffer* de recepción tiene 32 bits o más.
- **TNEARIEN:** Bit que habilita un evento de interrupción para indicar que el *buffer* de transmisión se aproxima a estar vacío.  
 0: Inhibe interrupción para cuando el buffer de Tx está próximo a desocuparse.  
 1: Habilita interrupción para cuando el buffer de Tx está próximo a desocuparse.
- **RNFULLIEN:** Bit que habilita un evento de interrupción para cuando el *buffer* de recepción se considera próximo a llenarse.  
 0: Inhibe interrupción para cuando el buffer de Rx está próximo a llenarse.  
 1: Habilita interrupción para cuando el buffer de Rx está próximo a llenarse.
- **FIFOMODE:** Bit para seleccionar el modo de FIFO para el SPI.  
 0: Modo normal.  
 1: Modo FIFO
- o **Registro de borrado de interrupción (SPI2CI)** La Figura 17.11 muestra el registro de borrado de evento de interrupción del SPI. Este registro sólo está implementado para el módulo SPI2.

### Registro de borrado de interrupciones (SPI2CI)

	7	6	5	4	3	2	1	0
Lectura	TXFERR	RXFERR	TXFOF	RXFOF	TNEAREFCI	RNFULLFCI	SPTEFCI	SPRFCI
Escritura								
Reset	0	0	0	0	0	0	0	0

Figura 17.11. Registro SPI2CI.

- **TXFERR:** Bandera que indica cuando ha habido un error de transmisión en modo FIFO del SPI.  
**0:** No ha ocurrido error de transmisión en modo FIFO.  
**1:** Ha ocurrido error de transmisión en modo FIFO.
- **RXFERR:** Bandera que indica cuando ha habido un error de recepción en modo FIFO del SPI.  
**0:** No ha ocurrido error de recepción en modo FIFO.  
**1:** Ha ocurrido error de recepción en modo FIFO.
- **TXFOF:** Bandera que indica cuando ha habido un error por sobreflujo en la transmisión en modo FIFO del SPI.  
**0:** No ha ocurrido error por sobreflujo en la transmisión en modo FIFO.  
**1:** Ha ocurrido error por sobreflujo en la transmisión en modo FIFO.
- **RXFOF:** Bandera que indica cuando ha habido un error por sobreflujo en la recepción en modo FIFO del SPI.  
**0:** No ha ocurrido error por sobreflujo en la recepción en modo FIFO.  
**1:** Ha ocurrido error por sobreflujo en recepción en modo FIFO.
- **TNEAREFCI:** Bit para aclarar la bandera TNEAREF. Escribir un “1” en este bit aclara la bandera de evento de *buffer* de transmisión próximo a desocuparse.
- **RNFULLFCI:** Bit para aclarar la bandera RNFULLF. Escribir un “1” en este bit aclara la bandera de evento de *buffer* de recepción próximo a llenarse.
- **SPTEFCI:** Bit para aclarar la bandera SPTEF. Escribir un “1” en este bit aclara la bandera de evento de *buffer* de transmisión vacío.
- **SPRFCI:** Bit para aclarar la bandera SPRF. Escribir un “1” en este bit aclara la bandera de evento de *buffer* de recepción lleno.

- **Otras funciones importantes del módulo SPI**
  - **Modo maestro:** El maestro es quien inicializa un evento de comunicación SPI y este inicia cuando se escribe sobre el registro de datos (SPIxD), siempre y cuando el bit de *buffer* de transmisión vacío SPIxS[SPTEF] = 1.

El maestro es quien inyecta el reloj al sistema (pin SPSCK) y lo obtiene del reloj del bus interno, dividido por un *preescaler* formado por los bits SPR2, SPR1 y SPR0.

El pin de transmisión es el MOSI y el de recepción es el MISO, pero en modo bidireccional cuando el bit SPIxC2[BIDIROE] = 1 el pin de entrada y salida de la información es el MOMI.

Si los bits SPIxC2[MODFEN] y SPIxC1[SSOE] son “1”, el pin !SS actúa como salida para selección del esclavo. Si el bit SPIxC2[MODFEN] = 1 y el bit SPIxC1[SSOE] = 0, el pin !SS actúa como una entrada para detección de falla por modo.

En el caso anterior si esta línea va al estado “0”, esto indica al maestro que hay otro dispositivo queriendo actuar como maestro y no queda otro remedio que conmutarse a esclavo (evento de pérdida de la maestría).

Si lo anterior ocurre durante una transmisión en proceso, el maestro pasa a un estado de IDLE, liberando los pines de salida.

Estos eventos producen un asituación de falla en modo (*Fault Mode*), el cual es marcado por la bandera SPIxS[MODF] = 1 y si se tiene habilitada la interrupción por el hecho (SPIxC1[SPIE]=1), entonces se genera un evento de interrupción.

- **Modo esclavo:** El modo esclavo es entendido cuando el bit SPIxC1[MSTR] = 1. El esclavo recibe del maestro el reloj de comunicación vía el pinn SPSCK.

El pin de transmisión es el MISO y el de recepción es el MOSI, pero en modo bidireccional cuando el bit SPIxC2[BIDIROE] = 1 el pin de entrada y salida de la información es el SISO.

El pin !SS se configura como entrada y permanece en estado bajo mientras se esté adelantando una comunicación. Si este pin va al estado alto, las líneas de salida del esclavo se ponen en estado de alta impedancia (IDLE).

**NOTA:** Nunca pueden haber dos esclavos seleccionados simultáneamente, esto llevaría a una colisión de los datos en el bus de comunicaciones.

- **Selección de 8 o 16 bits:** El bit SPIxC2[SPIMODE] selecciona cuando la transferencia de datos se hace en 8 o 16 bits. Si el usuario (en modo maestro) hace un cambio de modo cuando se está llevando a cabo una transmisión, el maestro aborta la transmisión y fuerza el sistema a entrar en modo IDLE. El esclavo sólo podrá configurar este modo una vez en su programa.

Tanto el maestro como el esclavo deberán tener la misma longitud de transferencia de datos, de no ser así se perderá la información.

- **Selección de la fase y la polaridad del reloj:** El módulo SPI permite trabajar con diferentes fases y polaridades del reloj SPSCK. La idea es poder acoger diferentes tipos de dispositivos en la comunicación.

Existen cuatro combinaciones de fase (PHA) y polaridad (POL), según se ilustran en las Figuras 17.12 y 17.13.

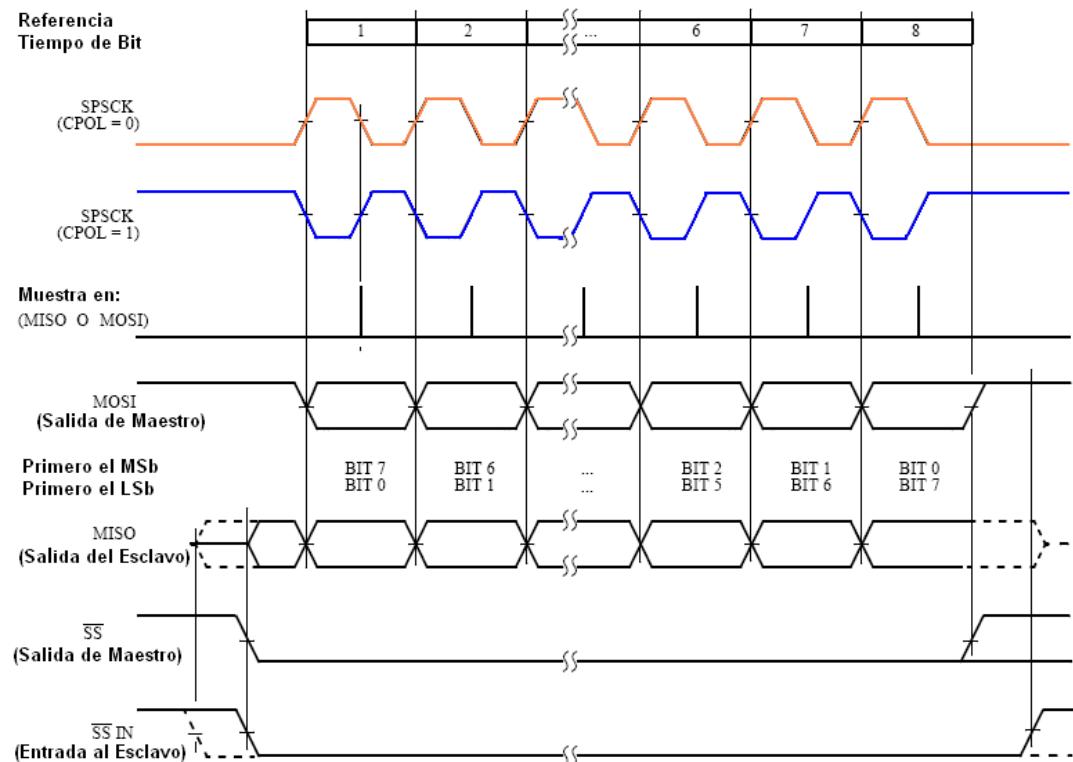
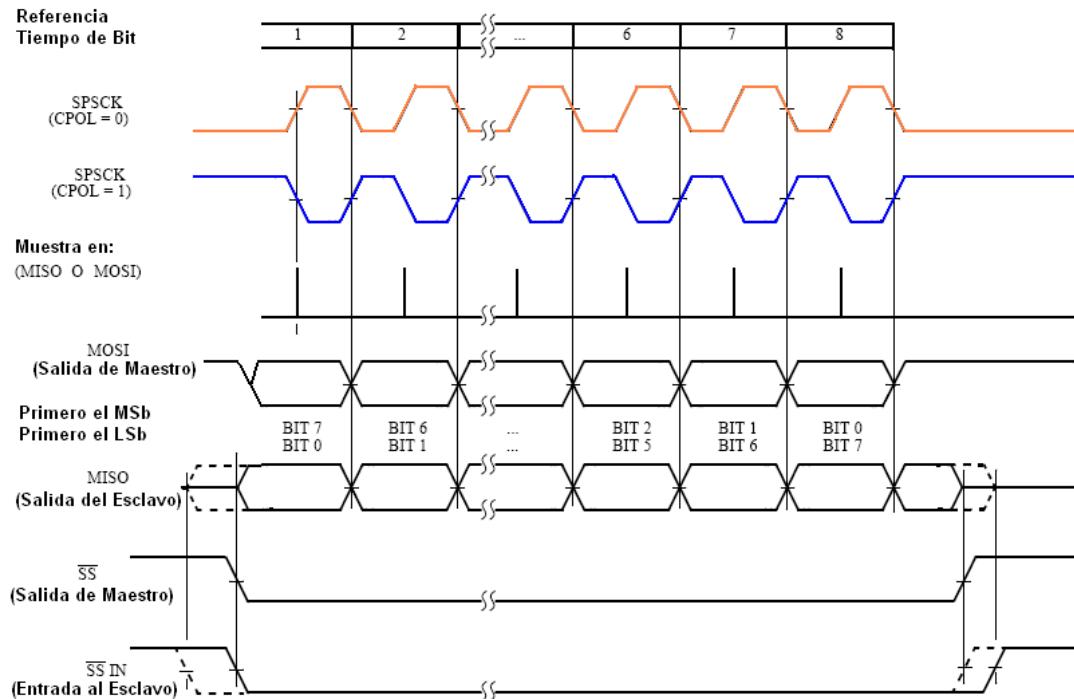


Figura 17.12. Selección de reloj con fase 1 y polaridad 1 y 2.



**Figura 17.13. Selección de reloj con fase 0 y polaridad 1 y 2.**

- **Cálculo de la rata de baudios en las comunicaciones SPI:** Existen 8 combinaciones de preescaler y 8 combinaciones del divisor, para establecer la rata de baudios en el módulo SPI. La siguiente ecuación establece el cálculo para la rata de baudios.

$$\text{Rata de Baudios} = \text{Reloj del BUS} / ((\text{SPPR} + 1) \times 2^{(\text{SPR}+1)})$$

- **Modo bidireccional:** Este modo permite manejar el módulo SPI en modo bidireccional, es decir por un solo pin se envía y se recibe el dato. A este modo se entra con el bit SPC0 = “1” y mediante el bit BIDIROE se convierte la señal de MOSI en MOMI y/o la señal MISO en SISO, dependiendo de quien es maestro o esclavo.
- **Modo FIFO:** Este modo permite al usuario la posibilidad de manejar altas ratas de baudios y gran cantidad de información, sin necesidad de rutinas complejas de manejo de listas de información.

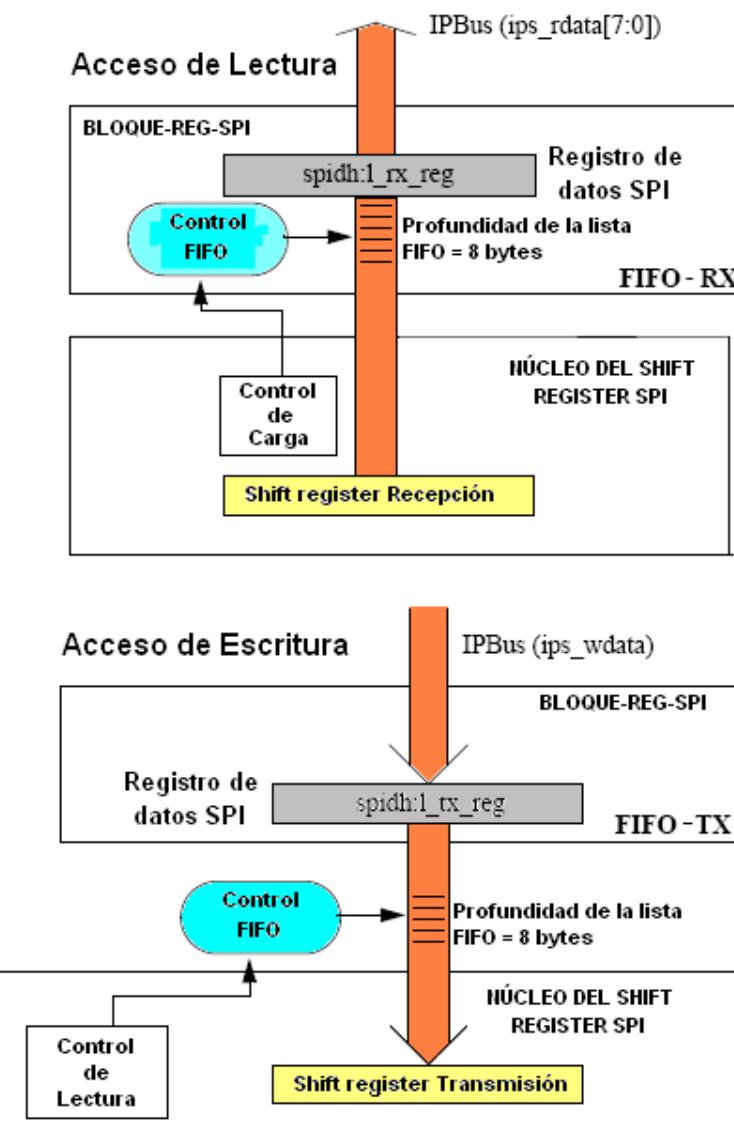
El modo FIFO permite almacenar en una lista de 8 bytes, la información que llega o se envía por el bus SPI. La Figura 17.4 ilustra sobre las máquinas FIFO tanto para la transmisión como para la recepción del módulo SPI.

Los datos recibidos entran por el *shift register* de recepción y son almacenados en una lista de 8 bytes, vía el registro de datos del SPI. El usuario podrá hacer lectura

a la lista, leyendo iterativamente sobre el registro de datos del SPI, con el criterio de lista FIFO (*First Input First Output*).

Los datos a enviar se escriben en el registro de datos del SPI y son almacenados en la lista FIFO para la transmisión, pasando finalmente al *shift register* de transmisión. El usuario podrá hacer escritura iterativa de los 8 bytes. Los datos son enviados con el de lista FIFO (*First Input First Output*).

**NOTA:** Dependiendo del tamaño de los datos, 8 o 16 bits, se podrían enviar 8 o cuatro, respectivamente.



**Figura 17.14. Máquina FIFO del SPI.**

## 17.2. DIAGRAMA EN BLOQUES, EXPLICACIÓN DEL FUNCIONAMIENTO Y REGISTROS ASOCIADOS AL MÓDULO IIC

- **Breve repaso del protocolo IIC**

Este sistema de comunicaciones fue desarrollado por Philips como un sistema multamaestro, para la conexión de periféricos a distancias relativamente cortas y velocidades de unos cientos de Kbps.

El protocolo se implementa sobre dos líneas, una para el reloj (SCL: *Serial Clock*) y otra para los datos (SDA: *Serial Data*). Esta definición lo configura como un sistema bidireccional del tipo maestro/esclavo.

La Figura 17.15 muestra una trama, para intercambio de datos en 8 bits (también existe en 10 bits). En la trama se puede apreciar que el protocolo es encabezado por una condición de *start*, seguido por los bits de dato, después se presenta un bit para la señal de reconocimiento y finalmente la condición de *stop*.

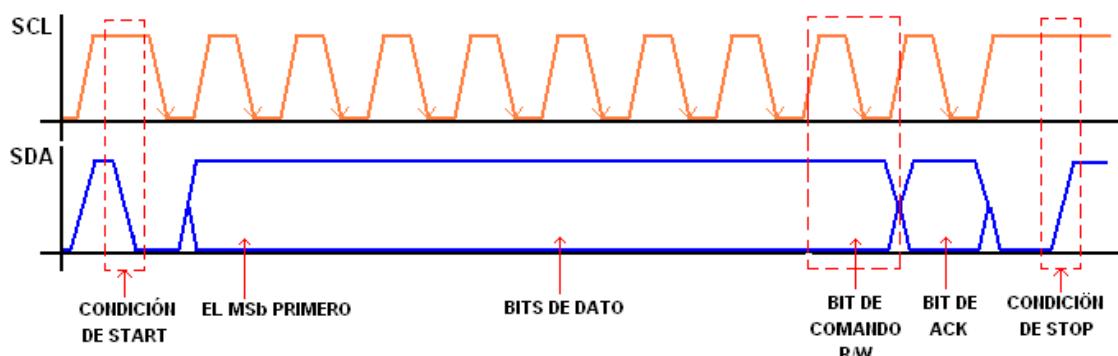


Figura 17.15. Protocolo IIC.

Las partes que conforman este protocolo son:

- **Condición de start:** Es válida cuando estando la señal de SCL en estado alto, se presenta un flanco de bajada de la señal de SDA.
- **Bits de dato:** Luego de haberse dado la condición de *start*, los bits de datos comienzan a aparecer (el MSb primero), sincronizados con el flanco descendente de la señal de SCL.
- **Bit de comando (R/W):** Cuando se está direccionando el dispositivo con el cual se va a establecer una comunicación IIC, este bit no pertenece al dato como tal, sino a la acción de escritura o lectura sobre el dispositivo en cuestión (Lectura (R) = nivel alto y Escritura (W) = nivel bajo).
- **Bit de reconocimiento (ACK = Acknowledge):** Este bit es verdadero en estado bajo y lo envía quien recibe el dato.
- **Condición de stop:** Es válida cuando estando la señal de SCL en estado alto, se presenta un flanco de subida de la señal de SDA.

Dependiendo del tipo de transferencia que se desee realizar, las tramas IIC se pueden configurar de la siguiente manera:

- **Envío de un byte a un destino:** La Figura 17.16 detalla las tramas para el envío de un byte hacia un destino.

En la primera trama la carga útil conforma la dirección del destino de la información. También es especificada la acción sobre el destino, que detalla una escritura sobre el destino (nivel bajo en el bit R/W).

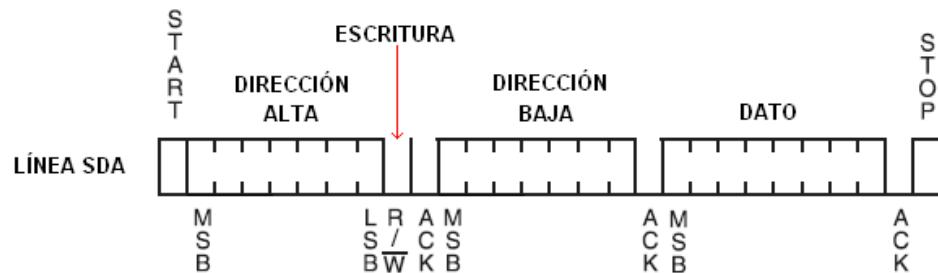
La dirección del dispositivo puede estar representada en 8 o 10 bits, de tal manera que podría existir un byte adicional para completar la dirección (este byte carecería de bit de *start* y de *stop*).

La segunda trama, carece de bit de *start* y representa el byte a transferir.

#### CON DIRECCIÓN EN 8 BITS



#### CON DIRECCIÓN EN 10 BITS

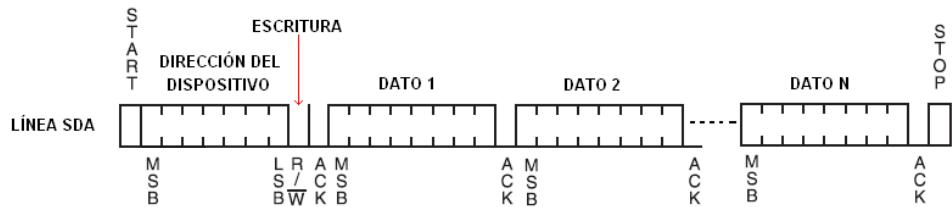


**Figura 17.16. Trama de escritura de un byte IIC.**

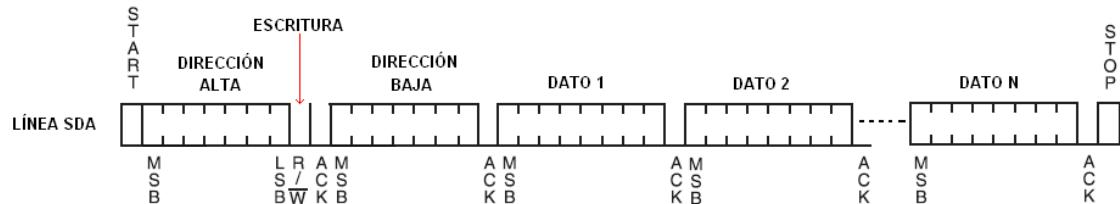
- **Envío de n bytes a un destino:** La Figura 17.17 detalla las tramas para el envío de n bytes hacia un destino.

Es idéntico al caso anterior, sólo que los datos se presentan uno a continuación del otro.

CON DIRECCIÓN EN 8 BITS



CON DIRECCIÓN EN 10 BITS

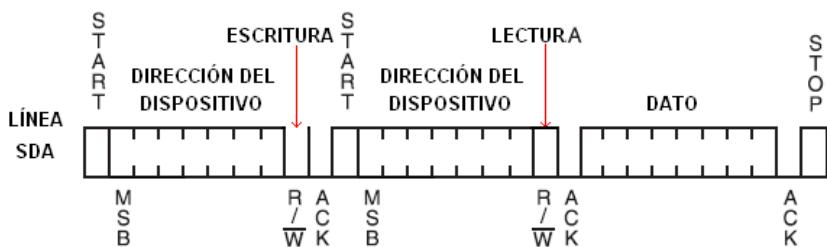


**Figura 17.17. Trama de escritura de n bytes IIC.**

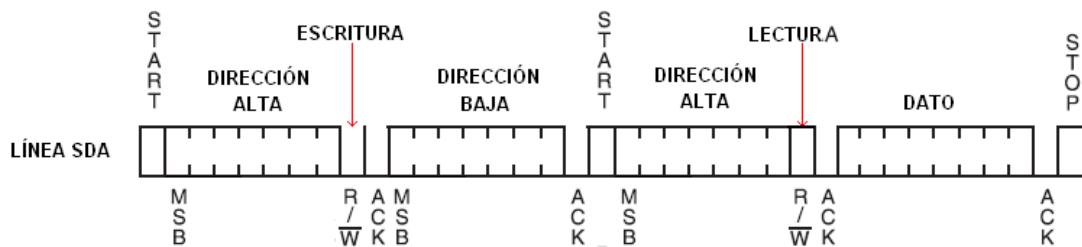
- **Lectura de un byte desde un dispositivo:** La Figura 17.18 detalla las tramas para la lectura de un byte desde un dispositivo.

En este caso el bit de lectura/escritura (R/W) va al estado “1”. Las primeras tramas envían la dirección del dispositivo a leer y las siguientes la acción de lectura sobre el mismo.

CON DIRECCIÓN EN 8 BITS

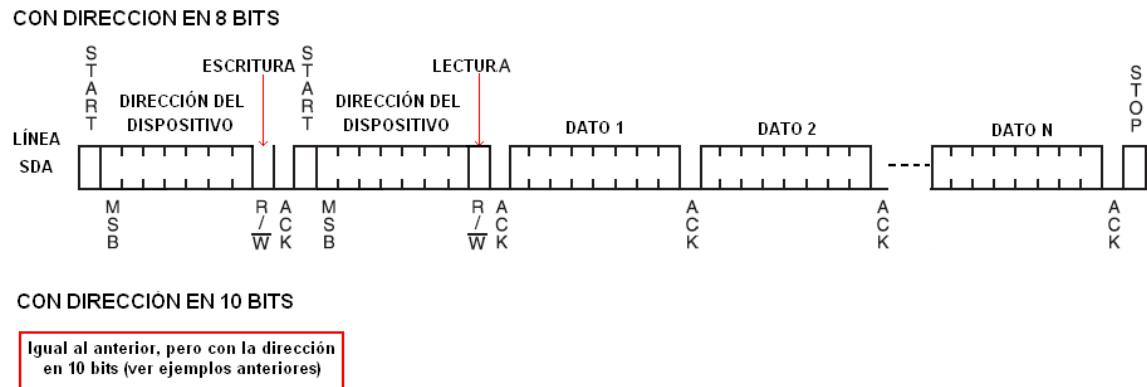


CON DIRECCIÓN EN 10 BITS



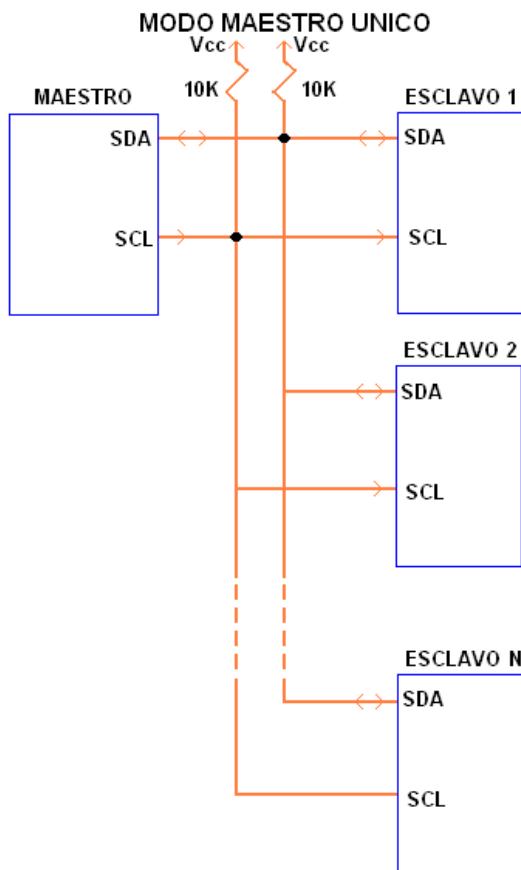
**Figura 17.18. Trama de lectura de un byte IIC.**

- **Lectura de n bytes desde un dispositivo:** La Figura 19.19 detalla las tramas para la lectura de n bytes desde un dispositivo.



**Figura 17.19. Trama de lectura de n bytes IIC.**

La conexión de varios dispositivos en un bus IIC, obedece al modelo maestro/esclavo y puede existir la posibilidad de que varios maestros manipulen el bus. Esto último se logra mediante un mecanismo de arbitramento de bus. La Figura 17.20 ilustra sobre una conexión maestro/esclavo IIC.



**Figura 17.20. BUS maestro único IIC.**

- **Breve descripción del módulo IIC y diagrama en bloques**

Las características más importantes del módulo IIC son:

- Trabaja hasta 100Kbps con máxima carga.
- El número máximo de dispositivos que se pueden conectar al bus IIC está limitado por la capacitancia de carga de 400pF máxima.
- Compatible con el bus IIC estándar.
- Operación en modo multimaestro.
- Hasta 64 frecuencias distintas del reloj del maestro.
- Generación y detección de bit de acknowledgement, seleccionable por software.
- Posibilidad de generar un evento de interrupción por:
  - La transferencia de cada byte.
  - Pérdida del arbitramiento del bus por parte del maestro.
  - Identificación de un llamado a una dirección.
- Generación y detección de señal de *start* y *stop*.
- Repetición de la señal de *start*.
- Detección de bus ocupado.
- Reconocimiento de llamado general.
- Extensión de la dirección a 10 bits.

La Figura 17.21 detalla el diagrama en bloques del módulo IIC, en donde el circuito (A) se refiere a la dirección del dispositivo cuando actúa en modo de esclavo. Esta dirección será la dirección del dispositivo en el bus IIC.

El circuito (B) configura la serie de registros de estado y control sobre el módulo IIC, los cuales se detallarán más adelante.

El circuito (C) establece la máquina de interrupciones del módulo.

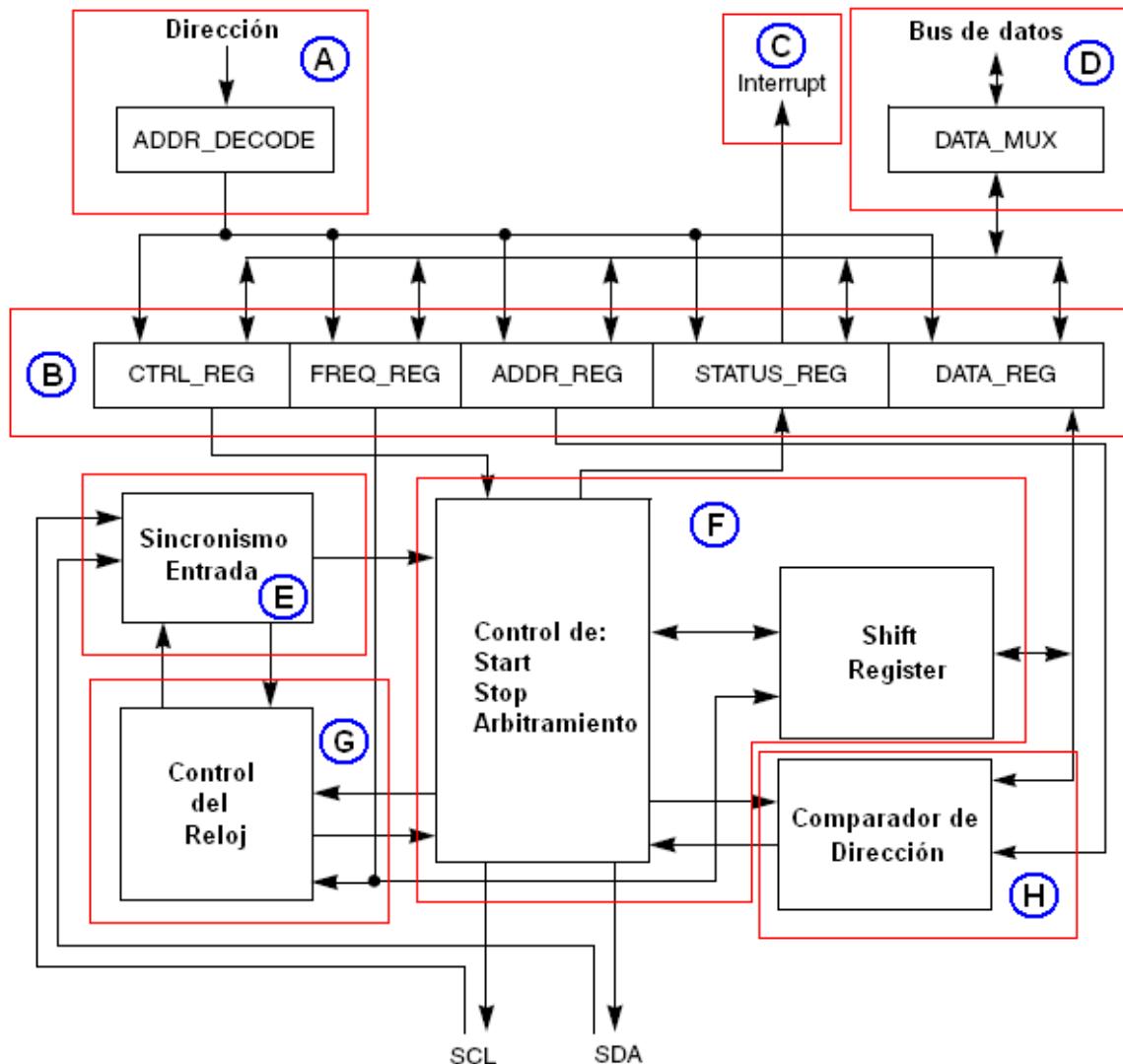
El dato en el módulo IIC se recibe y se envía sobre el mismo registro (configuración *double buffered*), el circuito (D) controla esta función.

Desde el punto de vista del entendimiento (*handshaking*), de los bits de sincronía de la trama de datos, es necesario establecer un mecanismo de sincronización, el cual se establece en el circuito (E).

El circuito (F) desarrolla toda la máquina para generar la trama de comunicación, esto incluye el shift register, la adición de bits de control y el establecimiento del arbitramiento del bus.

El circuito (G) genera y controla el reloj del sistema, si de un maestro se trata o sólo recibe el reloj, si se trata de un esclavo.

Finalmente el circuito (H), compara la dirección recibida con la propia, para establecer una coincidencia en un llamado desde el bus IIC.



**Figura 17.21. Diagrama en bloques del módulo IIC.**

- **Registros asociados al módulo IIC**
  - **Registro de dirección (IICA):** La Figura 17.22 muestra el registro de dirección del módulo IIC.
  -

**Registro de dirección (IICA): (FFFF8058)**

	7	6	5	4	3	2	1	0
Lectura	AD7	AD6	AD5	AD4	AD3	AD2	AD1	0
Escritura	0	0	0	0	0	0	0	0
Reset								

**Figura 17.22. Registro IICA.**

**AD1:AD7:** Bits que conforman la dirección baja del esclavo en un bus IIC.

- **Registro divisor de frecuencia del reloj (IICF):** La Figura 17.23 ilustra sobre el registro para dividir la frecuencia del reloj de bus, que controla la rata de baudios del módulo IIC.

**Registro divisor de frecuencia (IICF): (FFFF8059)**

	7	6	5	4	3	2	1	0
Lectura	MULT				ICR			
Escritura	0	0	0	0	0	0	0	0
Reset								

**Figura 17.23. Registro IICF.**

- **MULT:** Bits para definir un factor multiplicador de la rata de baudios IIC.  
**00:** Multiplica por 1  
**01:** Multiplica por 2  
**10:** Multiplica por 4  
**11:** Reservado
- **ICR:** Bits para definir la rata de baudios IIC. La siguiente ecuación define la rata de baudios de una comunicación IIC y los tiempos de las señales de *start* y *stop*:

$$\text{Rata Baudios} = \text{Frecuencia del reloj de BUS} / (\text{MULT} \times \text{Divisor\_SCL})$$

La Tabla 17.5 define los diferentes valores para el Divisor\_SCL.

**Tabla 17.5. Divisor \_SCL**

ICR (hex)	Divisor SCL	ICR (hex)	Divisor SCL
00	20	20	160
01	22	21	192
02	24	22	224
03	26	23	256
04	28	24	288
05	30	25	320
06	34	26	384
07	40	27	480
08	28	28	320
09	32	29	384
0A	36	2A	448
0B	40	2B	512
0C	44	2C	576
0D	48	2D	640
0E	56	2E	768
0F	68	2F	960
10	48	30	640
11	56	31	768
12	64	32	896
13	72	33	1024
14	80	34	1152
15	88	35	1280
16	104	36	1536
17	128	37	1920
18	80	38	1280
19	96	39	1536
1A	112	3A	1792
1B	128	3B	2048
1C	144	3C	2304
1D	160	3D	2560
1E	192	3E	3072
1F	240	3F	3840

- **Registro de control 1 (IICC1):** La Figura 17.24 ilustra sobre el registro de control 1 del módulo IIC.

### Registro de control 1 (IICC1): (FFFF805A)

	7	6	5	4	3	2	1	0
Lectura	IICEN	IICIE	MST	TX	TXAK	0	0	0
Escrutura	0	0	0	0	0	0	0	0
Reset						RSTA		

Figura 17.24. Registro IICC1.

- **IICEN:** Bit para habilitar la operación del módulo IIC.  
**0:** Inhibe la operación del módulo IIC  
**1:** Habilita la operación del módulo IIC
  - **IICIE:** Bit para habilitar un posible evento de interrupción del módulo IIC.  
**0:** Inhibe la interrupción del módulo IIC  
**1:** Habilita la interrupción del módulo IIC
  - **MST:** Bit para asignar la operación del módulo como maestro.  
**0:** Modo esclavo  
**1:** Modo Maestro
  - **TX:** Bit para seleccionar la dirección de la transferencia en modo maestro y esclavo. En modo maestro este bit deberá estar de acuerdo con el tipo de transferencia requerida. Para ciclos de envío de direcciones, este bit siempre estará en “1”. En caso de un direccionamiento sobre un esclavo, este bit se llevará a “1”, vía software y depende del estado del bit SRW.  
**0:** Receptor  
**1:** Transmisor
  - **TXACK:** Bit para determinar el valor del espacio de bit dedicado al *acknowledge* (ACK), en la recepción de el esclavo o el maestro.  
**0:** Una señal de ACK es enviada fuera del BUS, después de recibirse un byte de dato  
**1:** No se envía ACK como respuesta
  - **RSTA:** Bit para insertar un evento de *start* en el BUS, dada por el maestro. El maestro podría perder el arbitramiento del BUS si inyecta esta señal en un momento no indicado.
- **Registro de estado (IICS):** La Figura 17.25 ilustra sobre el registro de estado del módulo IIC.

### Registro de estado (IICS): (FFFF805B)

	7	6	5	4	3	2	1	0
Lectura	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
Escritura								
Reset	1	0	0	0	0	0	0	0

Figura 17.25. Registro IICS.

- **TCF:** Bandera para indicar cuando se ha completado la transferencia de un byte. Esta bandera se aclara cuando se lee el registro de datos IICD, en modo receptor o cuando se escribe un nuevo dato en el IICD, en modo transmisor.  
**0:** Transferencia en progreso  
**1:** Se ha completado una transferencia
- **IAAS:** Esta bandera indica cuando un esclavo, solicitado por coincidencia en dirección, ha respondido al llamado o cuando el bit GCAEN es “1” y un evento de llamada general ha sido recibido. Escribir en el registro IICC1, aclara esta bandera.  
**0:** No se ha dado un direccionamiento  
**1:** Ha coincidido un direccionamiento
- **BUSY:** Esta bandera indica el estado del BUS sin importar si se es maestro o esclavo. Esta bandera es “1” cuando se ha detectado un evento de *start* y es “0” cuando se detecta un evento de *stop*.  
**0:** El BUS está desocupado (IDLE)  
**1:** El BUS está ocupado
- **ARBL:** Esta bandera indica cuando un maestro ha perdido el arbitramiento del BUS. Esta bandera se aclara escribiendo un “1” en ella.  
**0:** Sistema en operación normal  
**1:** Se ha perdido el arbitramiento
- **SRW:** Cuando se está direccionando un esclavo, este bit indica el valor del bit R/W de la dirección a la que se llama y que fue enviada por el maestro.  
**0:** Esclavo recibiendo, el maestro escribe sobre el esclavo  
**1:** Maestro recibiendo, el esclavo escribe sobre el maestro
- **IICIF:** Este bit indica cuando hay una interrupción pendiente. Esta bandera se aclara escribiendo un “1” en ella, durante la rutina a la atención de interrupción. Los eventos que ponen a “1” esta bandera son:
  - Se ha completado la transferencia de un byte
  - Se ha dado una coincidencia en la solicitud de dirección sobre un esclavo
  - Se ha perdido el arbitramiento**0:** No hay interrupción pendiente  
**1:** Hay interrupción pendiente

- **RXAK:** Bandera que indica que se ha dado una señal de *acknowledge* (ACK), desde el dispositivo que ha recibido el dato enviado.
  - 0:** Se recibió señal de ACK
  - 1:** No se recibió señal de ACK
- o **Registro de datos (IICD):** La Figura 17.26 ilustra sobre el registro de datos del módulo IIC. La lectura sobre este registro devuelve el byte que acaba de llegar y la escritura deposita el byte siguiente a enviarse.

**Registro de datos (IICD): (FFFF805C)**

	7	6	5	4	3	2	1	0
Lectura Escritura	DATA							
Reset	0	0	0	0	0	0	0	0

**Figura 17.26. Registro IICD.**

- o **Registro de control 2 (IICC2):** La Figura 17.27 ilustra sobre el registro de control 2 del módulo IIC.

**Registro de control 2 (IICC2): (FFFF805D)**

	7	6	5	4	3	2	1	0
Lectura Escritura	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8
Reset	0	0	0	0	0	0	0	0

**Figura 17.27. Registro IICC2.**

- **GCAEN:** Bit para habilitar generación de una llamada general en el BUS IIC.
  - 0:** Inhibe una llamada general
  - 1:** Habilita una llamada general
- **ADEXT:** Bit para determinar el formato de la dirección de los dispositivos.
  - 0:** Direcciones en 7 bits
  - 1:** Direcciones en 10 bits
- **AD8:AD10:** Bits que configuran la parte alta de la dirección del dispositivo, en modo de 10 bits

- **Otras funciones importantes del módulo IIC**
  - **Repetición de la condición de *start*:** Esta propiedad del módulo IIC permite que se genere, en cualquier momento, una condición de *start* sin necesidad de haber sucedido una condición de *stop* anterior.
  - **Procedimiento de arbitramiento:** Esta propiedad permite que varios maestros estén conectados al BUS IIC. Cuando dos o más maestros tratan de acceder al BUS al mismo tiempo, un proceso de sincronización (circuito (F) en la Figura 17.21) determinará si el maestro pierde esta propiedad.

Un maestro pierde el arbitramiento si este transmite un “1” mientras otro maestro está transmitiendo un “0”, en ese momento el maestro perdedor se deberá conmutar como esclavo. Para este caso, la trama hacia el esclavo no llevará condición de *stop*. Esta acción es indicada por el bit IICS[ARBL] = 1.

- **Procedimiento de llamada general:** Esta propiedad permite al maestro encuestar sobre la presencia de un esclavo con una dirección determinada. Una llamada general se puede hacer en formato de 7 o 10 bits de dirección.

Cuando un esclavo identifica que el llamado coincide con su propia dirección, el bit IICS[IAAS] es puesto a “1” y el llamado deja de ser general.

Si la dirección llamada es la 0x00, se tratará entonces de una llamada general y el *software* del usuario determinará la acción a tomar.

Si el bit IICC2[GCAEN] = 0, el bit de ACK no es devuelto y el dato es ignorado.

### 17.3. EJEMPLOS CON EL MÓDULO SPI E IIC

- **Ejemplo con el módulo SPI**

El ejemplo trata sobre la comunicación SPI entre dos DEMOJM, de tal manera que todo lo digitado en el teclado se transmita y aparezca en el LCD de cada sistema.

El circuito de la Figura 17.28 detalla la conexión para el establecimiento de la comunicación planteada.

La configuración de la comunicación será del tipo *peer to peer*, en donde uno de los DEMOJM será configurado como maestro y el otro como esclavo.

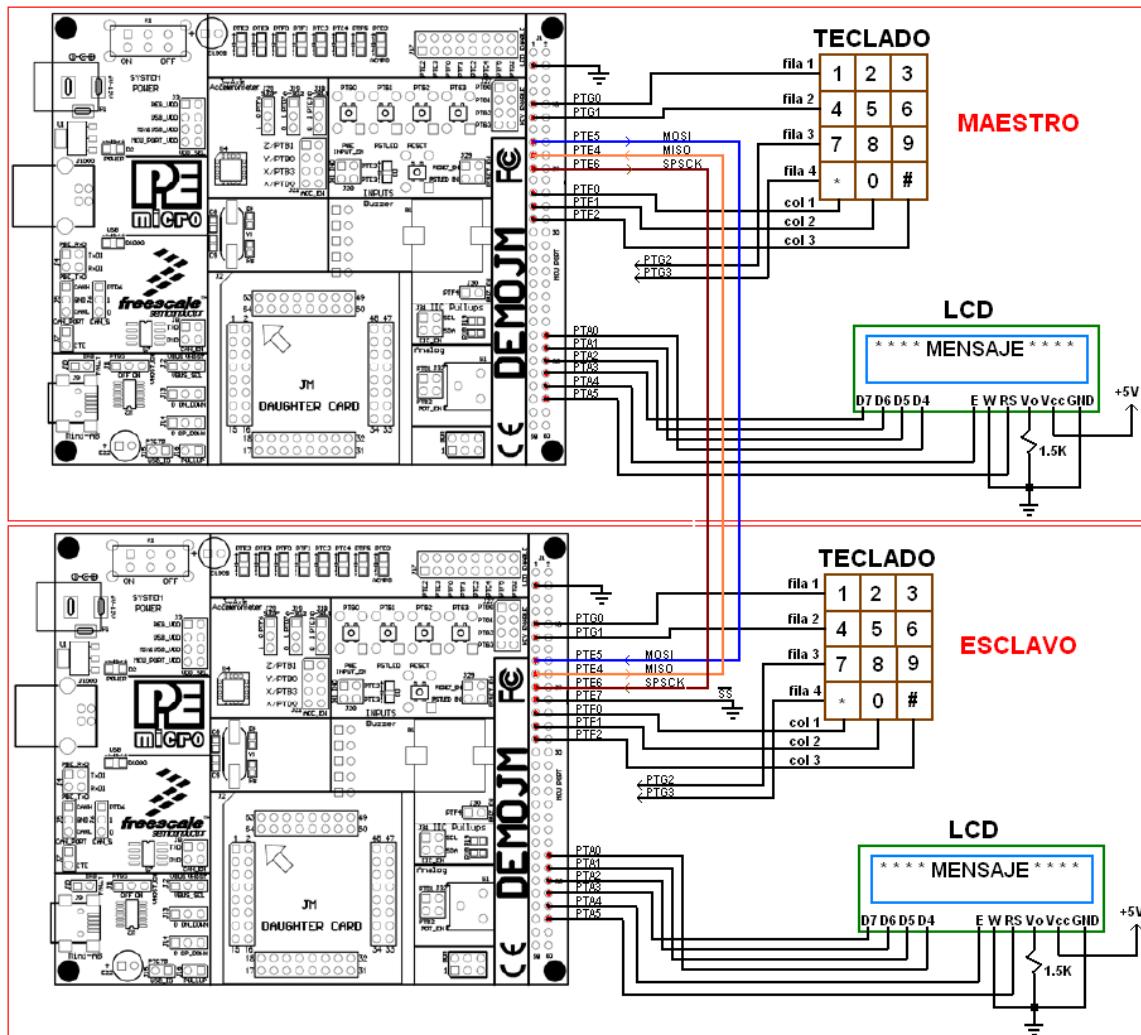


Figura 17.28. Circuito ejemplo SPI.

El siguiente listado en C resuelve el ejemplo planteado para la comunicación vía SPI entre dos sistemas DEMOJM, desde el punto de vista del maestro.

```
/****************************************************************************
 *  COMUNICACIÓN SPI: Ejemplo sobre la utilización del modulo SPI - Código del maestro
 *  main.c
 *
 *  Fecha: Junio 10, 2008
 *
 *  V1.0, Rev 0 por Diego Múnera
 *
 *  Asunto: Implementar un código en C que comunique dos sistemas DEMOJM vía puerto SPI. Toda
 *          tecla digitada deberá enviarse y aparecer en el LCD del dispositivo remoto. El sis-
 *          tema no trabaja en fallo por modo (MODEFAULT).
 *
 *  Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros
 *        editores.
 */
 ****
 *  DISTRIBUCION DE LOS PINES DE PUERTOS
 */
```

```

/*****
/* PUERTO * PIN * I/O * TIPO * INICIA * COMENTARIO */
/*****
/*PTA * - * - * - * - *LCD */
/* PTA0 * O * D * 1 *D4 del LCD */
/* PTA1 * O * D * 1 *D5 del LCD */
/* PTA2 * O * D * 1 *D6 del LCD */
/* PTA3 * O * D * 1 *D7 del LCD */
/* PTA4 * O * D * 1 *Señal ENABLE del LCD */
/* PTA5 * O * D * 1 *Señal RS del LCD */
/*****
/*PTB * - * - * - * - *No usado */
/*****
/*PTC * - * - * - * - *No usado */
/*****
/*PTD * - * - * - * - *No usado */
/*****
/*PTE * - * - * - * - *Puerto SPI */
/* PTE4 * O * D * 1 *MISO1 */
/* PTE5 * O * D * 1 *MOSI1 */
/* PTE6 * O * D * 1 *SPSCK1 */
/* PTE7 * O * D * 1 *SS1 */
/*****
/*PTF * - * - * - * - *No usado */
/* PTF0 * O * D * 1 *Columna 1 teclado */
/* PTF1 * O * D * 1 *Columna 2 teclado */
/* PTF2 * O * D * 1 *Columna 3 teclado */
/*****
/*PTG * - * - * - * - *Teclado */
/* PTG0 * I * D * 1 *Fila 1 del teclado */
/* PTG1 * I * D * 1 *Fila 2 del teclado */
/* PTG2 * I * D * 1 *Fila 3 del teclado */
/* PTG3 * I * D * 1 *Fila 4 del teclado */
/*****
/*PTH * - * - * - * - *No usado */
/*****
/*PTJ * - * - * - * - *No usado */
/*****



/* Archivos de cabecera */
#include <hidef.h> //macro para interrupciones
#include "derivative.h" //macro de declaraciones del MCU

/* Declaracion de variables y constantes */
byte bandera0=0; //bandera de propósito general
byte bandera1=0; //bandera de propósito general
byte bandera2=0; //bandera de propósito general
byte rx_dato=0; //variable temporal para dato por Rx
unsigned long retardo=0; //parametro para hacer retardo
unsigned int i=0; //variable de iteración
char columna=0; //columna detectada del teclado
char fila=0; //fila detectada del teclado
char comando8=0; //parametro inicialización LCD modo 8 bits
char dato4; //dato a enviar al LCD
char rs=0; //señal de dato o comando al LCD
char a=0; //variable temporal
char KBIISC_Config=0; //byte de inicialización del registro KBIISC
char KBIIEPE_Config=0; //byte de inicialización del registro KBIIEPE
char KBIIES_Config=0; //byte de inicialización del registro KBIIES
char PTAD_Config=0; //byte de inicialización del registro PTAD
char PTADD_Config=0; //byte de inicialización del registro PTADD
char PTAPE_Config=0; //byte de inicialización del registro PTAPE
char PTASE_Config=0; //byte de inicialización del registro PTASE
char PTADS_Config=0; //byte de inicialización del registro PTADS
char PTAIFE_Config=0; //byte de inicialización del registro PTAIFE
char PTFD_Config=0; //byte de inicialización del registro PTFD

```

```

char PTFDD_Config=0;                                //byte de inicializacion del registro PTFDD
char PTFPE_Config=0;                                //byte de inicializacion del registro PTFPE
char PTFSE_Config=0;                                //byte de inicializacion del registro PTFSE
char PTFDS_Config=0;                                //byte de inicializacion del registro PTFDS
char PTFIFE_Config=0;                                //byte de inicializacion del registro PTFIFE_
char SPI1C1_Config=0;                                //byte de inicializacion del registro SPI1C1
char SPI1C2_Config=0;                                //byte de inicializacion del registro SPI1C2
char SPI1BR_Config=0;                                //byte de inicializacion del registro SPI1BR
const unsigned char mensaje[] = "MENSAJE ESCLAVO"; //Arreglo del mensaje para llevar al LCD
char *pM;                                         //puntero a mensaje para el LCD

/* Macros y definiciones */
#define Disable_COP() SOPT1 &= 0x3F      //deshabilita el COP

/* Declaracion de funciones */
void KBI_Init(char KBI1SC_Config,char KBI1PE_Config,char KBI1ES_Config);           //declare funcion que inicializa KBI1
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config);          //declare funcion que inicializa PTA
void PTF_Init(char PTFD_Config,char PTFDD_Config,char PTFPE_Config,char PTFSE_Config,char PTFDS_Config,char PTFIFE_Config);          //declare funcion que inicializa PTF
void SPI1_Init(char SPI1C1_Config,char SPI1BR_Config,char SPI1C2_Config);           //declare funcion que inicializa SPI1
void Delay(unsigned long retardo);          //funcion para hacer retardos varios
void LCD_Init(void);                      //funcion para inicializar LCD
void Lcd_4bit_Wr(char dato4,char rs);       //funcion para escribir dato al LCD
void Comando_8bit(char comando8);          //funcion para inicio del LCD a 8 bits
void Rote_Col (void);                     //funcion para rotar un cero por columnas teclado
void Analisis_Tecla(void);                //funcion para analisis de tecla presionada

/* main(): Funcion principal */
void main(void) {

// Inicializacion del reloj:
    MCGTRM = NVMCGTRM + 52;                  //Tome valor programado de fabrica del TRIM
    MCGC1 = 0x04;                            //Modo FEI divisor por 1
    MCGC2 = 0x00;                            //Desactiva modo de reloj externo
    MCGC4 = 0x22;                            //Rango alto del DCO
                                            //El reloj MCGOUT queda en 24999945 Hz
    Disable_COP();                          //deshabilita el COP
    EnableInterrupts;                      //habilita interrupciones
    PTA_Init(0x00,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "0",pines como salida, no pullups, si slew, no strength y
                                                //si filtro
    PTF_Init(0xFF,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "1",pines como salida, no pullups, si slew, no strength y
                                                //si filtro
    PTGPE=0xFF;                            //habilite pullups puerto G
    KBI_Init(0x06,0xC3,0x00);              //da ACK, habilita interr, habilita pullups y flanco de bajada
    LCD_Init();                            //inicialice LCD
    SPI1_Init(0x50,0x07,0x00);              //enciende SPI, inhibe interr por Rx y Tx, modo maestro
                                            //pin SS como I/O, SPPR = 0 y SPR = 7 para una
                                            //rata de baudios = 24999945/(1 x 2^8) = 97656Hz
    pM = (char *)mensaje;                  //Puntero toma direccion del mensaje con definicion
    for (i=0;i<=15;i++){                 //Ciclo para imprimir 10 caracteres del mensaje
        Lcd_4bit_Wr(*pM,1);               //Envia caracter al LCD
        pM++;                             //Incrementa puntero al arreglo del mensaje
    }
    Lcd_4bit_Wr(0xC0,0);                  //primer caracter por Rx a fila 2 columna 1 del LCD
    SPI1C1_SPIE=1;                      //habilite interrupcion por receptor SPI lleno
    for(;;){                            //iteracion para lectura del teclado (for infinito)
        Rote_Col();                      //llame funcion para rotar cero en columnas teclado
        if(bandera0==1){                //si se detecto tecla
            bandera0=0;                  //aclare bandera de tecla detectada
            Analisis_Tecla();           //analice tecla presionada y transmitala
        }
    }
    /* es necesario asegurarse de nunca abandonar el main */
}

```

```

/* Funcion para inicializar el puerto A */
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char
PTADS_Config,char PTAIFE_Config){
    PTAD=PTAD_Config;                                //inicialice estado pines PTA
    PTADD=PTADD_Config;                             //inicialice dirección de pines PTA
    PTAPE=PTAPE_Config;                            //inicialice estado de pullups PTA
    PTASE=PTASE_Config;                            //inicialice estado del slew rate PTA
    PTADS=PTADS_Config;                            //inicialice estado de drive strength PTA
    PTAIFE=PTAIFE_Config;                           //inicialice estado del filtro PTA
}

/* Funcion para inicializar el puerto F */
void PTF_Init(char PTFD_Config,char PTFDD_Config,char PTFPE_Config,char PTFSE_Config,char
PTFDS_Config,char PTFIFE_Config){
    PTFD=PTFD_Config;                                //inicialice estado pines PTF
    PTFDD=PTFDD_Config;                             //inicialice dirección de pines PTF
    PTFPE=PTFPE_Config;                            //inicialice estado de pullups PTF
    PTFSE=PTFSE_Config;                            //inicialice estado del slew rate PTF
    PTFDS=PTFDS_Config;                            //inicialice estado de drive strength PTF
    PTFIFE=PTFIFE_Config;                           //inicialice estado del filtro PTF
}

/* Funcion para inicializar el KBI */
void KBI_Init(char KBI1SC_Config,char KBI1PE_Config,char KBI1ES_Config){
    KBI1PE=KBI1PE_Config;                          //inicialice registro de habilitacion de pines KBI
    KBI1SC_KBACK=1;                               //aclara interrupciones espureas
    KBI1ES=KBI1ES_Config;                         //inicialice registro de flanco y nivel del KBI
    KBI1SC=KBI1SC_Config;                         //inicialice registro de estado y control del KBI
}

/* Funcion para inicializar el SPI1 */
void SPI1_Init(char SPI1C1_Config,char SPI1BR_Config,char SPI1C2_Config){
    SPI1C1=SPI1C1_Config;                          //inicialice registro SPI1C1
    SPI1BR=SPI1BR_Config;                         //inicialice registro SPI1BR
    SPI1C2=SPI1C2_Config;                          //inicialice registro SPI1C2
}

/* Funcion para inicializacion del LCD a 4 bits */
void LCD_Init(void){
    Delay(200000);                                //hace retardo
    Comando_8bit(0x30);                           //llama función enviar comando en 8 bits con comando
    Delay(200000);                                //hace retardo
    Comando_8bit(0x30);                           //llama función enviar comando en 8 bits con comando
    Delay(200000);                                //hace retardo
    Comando_8bit(0x20);                           //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x28,0);                          //LCD 2x16 y manejo a 4 bits
    Lcd_4bit_Wr(0x0C,0);                          //ON LCD, OFF cursor
    Lcd_4bit_Wr(0x01,0);                           //borrar pantalla LCD
    Lcd_4bit_Wr(0x06,0);                           //desplaza cursor a la derecha con cada caracter
    Lcd_4bit_Wr(0x80,0);                           //primer caracter fila 1 columna 1
}

/* Funcion para escribir a 4 bits en el LCD */
void Lcd_4bit_Wr(char dato4,char rs){
    if(rs==0){                                     //pregunte si se va a enviar un comando o un dato
        PTAD_PTAD5=0;                            //prepara envío de comando RS = 0
    }
    else{
        PTAD_PTAD5=1;                            //prepara envio de dato RS = 1
    }
    a=dato4;                                      //almacena temporalmente el dato
    a=a>>4;                                       //desplaza parte alta dato para la baja
    PTAD&=0xF0;                                    //enmascara parte alta del puerto A
    a&=0x0F;                                       //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;                                  //sube pin de enable
    Delay(20000);                                 //hace retardo
    PTAD|=a;                                       //envía nibble alto del dato
    Delay(20000);                                 //hace retardo
    PTAD_PTAD4=0;                                  //baja pin de enable
    Delay(20000);                                 //hace retardo
}

```

```

PTAD&=0xF0;                                //enmascara parte alta del puerto A
dato4&=0x0F;                                //enmascara parte baja del dato a enviar al LCD
PTAD_PTAD4=1;                               //sube pin de enable
Delay(20000);                             //hace retardo
PTAD|=dato4;                                //envía nibble bajo del dato
Delay(20000);                             //hace retardo
PTAD_PTAD4=0;                                //baja pin de enable
Delay(20000);                             //hace retardo

}

/* Función para envio de comando en 8 bits al LCD (solo para inicio) */
void Comando_8bit(char comando8){
    PTAD_PTAD5=0;                            //prepara envio de comando
    PTAD&=0xF0;                                //enmascara parte alta del puerto A
    comando8&=0xF0;                            //enmascara parte alta de comando a enviar al LCD
    comando8=comando8>>4;                    //desplaza para tomar nibble alto
    PTAD_PTAD4=1;                            //sube pin de enable
    Delay(20000);                           //hace retardo
    PTAD|=comando8;                            //envía comando
    Delay(20000);                           //hace retardo
    PTAD_PTAD4=0;                            //baja pin de enable
    Delay(20000);                           //hace retardo
}

/* Funcion Delay(): Retarda basado en una variable tipo entera */
void Delay(unsigned long retardo){
    while(retardo>0){                         //llego a cero?
        retardo--;                            //no --> decrementa
    }
}

/* Funcion para rotar cero por columnas teclado */
void Rote_Col (void){
    PTFD = 0x0E;                                //Cero a la columna 1
    Delay(500000);                            //hace retardo
    PTFD = 0x0D;                                //Cero a la columna 2
    Delay(500000);                            //hace retardo
    PTFD = 0x0B;                                //Cero a la columna 3
    Delay(500000);                            //hace retardo
}

/* Funcion para analizar tecla presionada y enviarla esclavo */
void Analisis_Tecla(void){
    if(columna==0xE & fila==1){                //si se detecta columna 1 y fila 1
        SPIID = 0x31;                            //transmite un 1
        SPI1C1_SPTIE=1;                          //habilite interrupcion por buffer de Tx vacio
        do{                                     //espere a que se complete la TX
            }while(bandera2 == 0);
        bandera2 = 0;                            //aclare bandera
    }
    if(columna==0xD & fila==1){                //si se detecta columna 2 y fila 1
        SPIID = 0x32;                            //transmite un 2
        SPI1C1_SPTIE = 1;                        //habilite interrupcion por buffer de Tx vacio
        do{                                     //espere a que se complete la TX
            }while(bandera2 == 0);
        bandera2 = 0;                            //aclare bandera
    }
    if(columna==0xB & fila==1){                //si se detecta columna 3 y fila 1
        SPIID = 0x33;                            //transmite un 3
        SPI1C1_SPTIE = 1;                        //habilite int por TXD 1
        do{                                     //espere a que se complete la TX
            }while(bandera2 == 0);
        bandera2 = 0;                            //aclare bandera
    }
    if(columna==0xE & fila==2){                //si se detecta columna 1 y fila 2
        SPIID = 0x34;                            //transmite un 4
        SPI1C1_SPTIE = 1;                        //habilite interrupcion por buffer de Tx vacio
        do{                                     //espere a que se complete la TX
            }while(bandera2 == 0);
    }
}

```

```

bandera2 = 0;                                //aclare bandera
}
if(columna==0x0D & fila==2){
    SPIID = 0x35;
    SPI1C1_SPTIE = 1;
    do{
        }while(bandera2 == 0);
    bandera2 = 0;
}
if(columna==0x0B & fila==2){
    SPIID = 0x36;
    SPI1C1_SPTIE = 1;
    do{
        }while(bandera2 == 0);
    bandera2 = 0;
}
if(columna==0x0E & fila==3){
    SPIID = 0x37;
    SPI1C1_SPTIE = 1;
    do{
        }while(bandera2 == 0);
    bandera2 = 0;
}
if(columna==0x0D & fila==3){
    SPIID = 0x38;
    SPI1C1_SPTIE = 1;
    do{
        }while(bandera2 == 0);
    bandera2 = 0;
}
if(columna==0x0B & fila==3){
    SPIID = 0x39;
    SPI1C1_SPTIE = 1;
    do{
        }while(bandera2 == 0);
    bandera2 = 0;
}
if(columna==0x0D & fila==4){
    SPIID = 0x30;
    SPI1C1_SPTIE = 1;
    do{
        }while(bandera2 == 0);
    bandera2 = 0;
}
Delay(1000000);
Lcd_4bit_Wr(0xC0,0);
Lcd_4bit_Wr(rx_dato,1);
}

/* Funcion de atencion a la interrupcion del KBI1 */
interrupt VectorNumber_Vkeyboard void ISR_KBI (void){
    columna=PTFD & 0x0F;                      //capture el valor de la columna
    if(PTGD_PTGD0==0){                         //si detecto la fila 1
        fila=1;                                //marque fila 1
    }
    if(PTGD_PTGD1==0){                         //si detecto la fila 2
        fila=2;                                //marque fila 2
    }
    if(PTGD_PTGD2==0){                         //si detecto la fila 3
        fila=3;                                //marque fila 3
    }
    if(PTGD_PTGD3==0){                         //si detecto la fila 4
        fila=4;                                //marque fila 4
    }
    KBI1SC_KBACK = 1;                          //de reconocimiento de tecla presionada
    bandera0=1;                               //pone bandera de tecla presionada
}

/* Funcion de atencion a la interrupcion por SPI */

```

```

interrupt VectorNumber_Vspi1 void ISR_SPI1(void){
    a=SPI1S;                                //aclara bandera de interrupcion
    rx_data = SPI1ID;                         //lea dato que entro por recepcion
    SPI1C1_SPTIE=0;                           //inhiba interrupciones por Tx
    bandera2= 1;                             //pone bandera de dato recibido
}

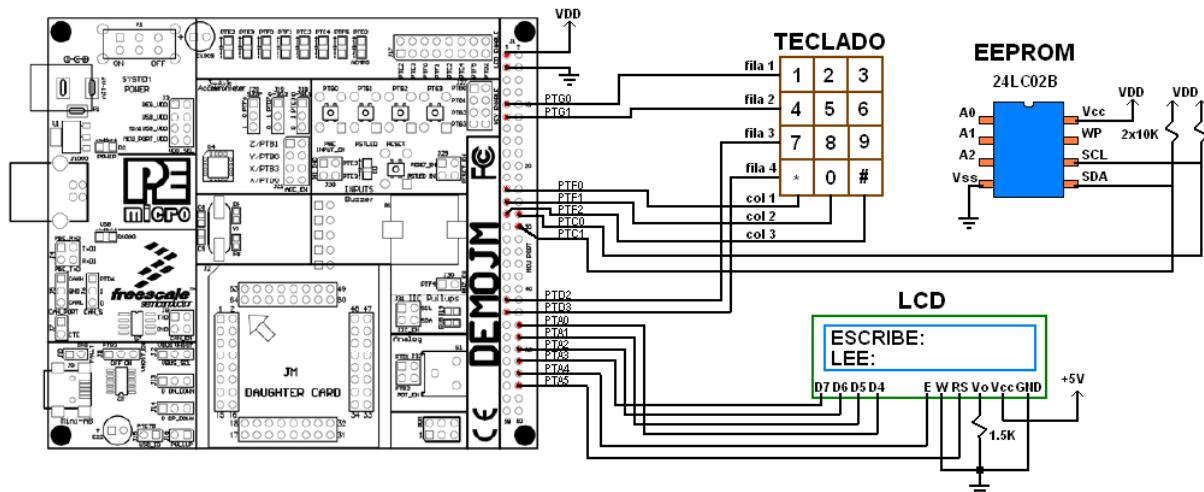
```

Para el código del esclavo se podría utilizar el mismo, pero con la precaución de definir bien el modo con el bit SPI1C1[MSTR] , la fase y polaridad del reloj con los bits SPI1C1[CPOL] y SPI1C1[CPHA] . Para el esclavo (*peer to peer*) no es necesario definir el reloj, puesto que es el maestro quien lo controla y define.

- **Ejemplo con el módulo IIC**

El ejemplo trata sobre la escritura/lectura entre un DEMOJM y una memoria serial EEPROM (24LC02B).

El circuito de la Figura 17.29 detalla la conexión para el establecimiento de la comunicación entre el DEMOJM y la memoria serial 24LC02B.



**Figura 17.29. Circuito ejemplo IIC.**

El siguiente listado en C resuelve el ejemplo planteado para la comunicación vía IIC, entre un sistema DEMOJM y una memoria EEPROM

```

/****************************************************************************
 *  COMUNICACIÓN IIC: Ejemplo sobre la utilización del modulo SPI - Código del maestro
 *  main.c
 *
 *  Fecha: Junio 15, 2008
 *
 *  V1.0, Rev 0 por Diego Múnera
 */

```

```

/*
 * Asunto: Implementar un codigo en C que escriba un byte sobre una memoria EEPROM, desde un
 * sistema DEMOJM. El dato escrito sera introducido desde un teclado numerico y sera
 * leido continuamente y presentado en el LCD, al segundo de haberse escrito.
 */
/*
 * Nota: Las tildes en los comentarios son omitidas para efectos de compatibilidad con otros
 * editores.
 *****/
/*
 * DISTRIBUCION DE LOS PINES DE PUERTOS
 *****/
/*
 * PUERTO * PIN * I/O * TIPO * INICIA * COMENTARIO */
 *****/
/*PTA * - * - * - * - *LCD */
/* PTA0 * O * D * 1 *D4 del LCD */
/* PTA1 * O * D * 1 *D5 del LCD */
/* PTA2 * O * D * 1 *D6 del LCD */
/* PTA3 * O * D * 1 *D7 del LCD */
/* PTA4 * O * D * 1 *Señal ENABLE del LCD */
/* PTA5 * O * D * 1 *Señal RS del LCD */
*****/
/*PTB * - * - * - * - *No usado */
*****/
/*PTC * - * - * - * - *Puerto IIC */
/* PTC0 * O * D * 1 *SCL */
/* PTC1 * I/O * D * 1 *SDA */
*****/
/*PTD * - * - * - * - *No usado */
*****/
/*PTE * - * - * - * - *No usado */
*****/
/*PTF * - * - * - * - *No usado */
/* PTF0 * O * D * 1 *Columna 1 teclado */
/* PTF1 * O * D * 1 *Columna 2 teclado */
/* PTF2 * O * D * 1 *Columna 3 teclado */
*****/
/*PTG * - * - * - * - *Teclado */
/* PTG0 * I * D * 1 *Fila 1 del teclado */
/* PTG1 * I * D * 1 *Fila 2 del teclado */
/* PTG2 * I * D * 1 *Fila 3 del teclado */
/* PTG3 * I * D * 1 *Fila 4 del teclado */
*****/
/*PTH * - * - * - * - *No usado */
*****/
/*PTJ * - * - * - * - *No usado */
*****/
/*
 * Archivos de cabecera */
#include <hidef.h> //macro para interrupciones
#include "derivative.h" //macro de declaraciones del MCU
/*
 * Declaracion de variables y constantes */
byte bandera0=0; //bandera de propósito general
byte bandera1=0; //bandera de propósito general
byte bandera2=0; //bandera de propósito general
byte rx_dato=0; //variable temporal para dato por Rx
byte tmp=0; //variable temporal para captura de dato RX IIC
byte direccion = 0; //direccion de la celda a accesar en la 24LC02B
byte dato=0; //dato a TX a la 24LC02B
unsigned long retardo=0; //parametro para hacer retardo
unsigned int i=0; //variable de iteracion
char columna=0; //columna detectada del teclado
char fila=0; //fila detectada del teclado
char comando8=0; //parametro inicializacion LCD modo 8 bits
char dato4; //dato a enviar al LCD
char rs=0; //señal de dato o comando al LCD

```

```

char a=0;                                //variable temporal
char KBIISC_Config=0;                     //byte de inicializacion del registro KBIISC
char KBIPE_Config=0;                      //byte de inicializacion del registro KBIPE
char KBIIES_Config=0;                     //byte de inicializacion del registro KBIIES
char PTAD_Config=0;                       //byte de inicializacion del registro PTAD
char PTADD_Config=0;                      //byte de inicializacion del registro PTADD
char PTAPE_Config=0;                      //byte de inicializacion del registro PTAPE
char PTASE_Config=0;                      //byte de inicializacion del registro PTASE
char PTADS_Config=0;                      //byte de inicializacion del registro PTADS
char PTAIFE_Config=0;                     //byte de inicializacion del registro PTAIFE
char PTFD_Config=0;                       //byte de inicializacion del registro PTFD
char PTFDD_Config=0;                      //byte de inicializacion del registro PTFDD
char PTFPE_Config=0;                      //byte de inicializacion del registro PTFPE
char PTFSE_Config=0;                      //byte de inicializacion del registro PTFSE
char PTFDS_Config=0;                      //byte de inicializacion del registro PTFDS
char PTFIFE_Config=0;                     //byte de inicializacion del registro PTFIFE
char I2C1F_Config=0;                      //byte de inicializacion del registro I2C1F
char IIC1C1_Config=0;                     //byte de inicializacion del registro IIC1C1
char IIC1S_Config=0;                      //byte de inicializacion del registro IIC1S
const unsigned char mensaje1[] = "ESCRIBA:    "; //Arreglo del mensaje para llevar al LCD
const unsigned char mensaje2[] = "LEE:      "; //Arreglo del mensaje para llevar al LCD
char *pM;                                 //puntero a mensaje para el LCD

/* Macros y definiciones */
#define Disable_COP() SOPT1 &= 0x3F           //deshabilita el COP
#define I2C_E2PROM_WRITE 0xA0                 //direccion de escritura sobre la 24LC02B
#define I2C_E2PROM_READ 0xA1                  //direccion de lectura sobre la 24LC02B

/* Declaracion de funciones */
void KBI_Init(char KBIISC_Config,char KBIPE_Config,char KBIIES_Config); //declare funcion que inicializa KBI
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char PTADS_Config,char PTAIFE_Config); //declare funcion que inicializa PTA
void PTF_Init(char PTFD_Config,char PTFDD_Config,char PTFPE_Config,char PTFSE_Config,char PTFDS_Config,char PTFIFE_Config); //declare funcion que inicializa PTF
void Delay(unsigned long retardo); //funcion para hacer retardos varios
void LCD_Init(void); //funcion para inicializar LCD
void Lcd_4bit_Wr(char dato4,char rs); //funcion para escribir dato al LCD
void Comando_8bit(char comando8); //funcion para inicio del LCD a 8 bits
void Rote_Col (void); //funcion para rotar un cero por columnas teclado
void Analisis_Tecla(void); //funcion para analisis de tecla presionada
void i2cCfgI2C(char I2C1F_Config,char IIC1C1_Config,char IIC1S_Config); //funcion para configuracion inicial del IIC
void i2cfWrite(byte dato); //funcion para escribir un dato en la 24LC02B
void i2cfWriteE2PROM1Byte (byte direccion,byte dato); //funcion para hacer TX al IIC
void i2cfReadE2Prom1Byte(byte direccion); //funcion para hacer RX al IIC

/* main(): Funcion principal */
void main(void) {

// Inicializacion del reloj:
    MCGTRM = NVMCGTRM + 52; //Tome valor programado de fabrica del TRIM
    MCGC1 = 0x04; //Modo FEI divisor por 1
    MCGC2 = 0x00; //Desactiva modo de reloj externo
    MCGC4 = 0x22; //Rango alto del DCO
    Disable_COP(); //deshabilita el COP
    EnableInterrupts(); //habilita interrupciones
    PTA_Init(0x00,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "0",pines como salida, no pullups, si slew, no strength y si filtro
    PTF_Init(0xFF,0xFF,0x00,0xFF,0x00,0xFF); //pines inician en "1",pines como salida, no pullups, si slew, no strength y si filtro
    PTGPE=0xFF; //habilite pullups puerto G
    KBI_Init(0x06,0xC3,0x00); //da ACK, habilita interr, habilita pullups y flanco de bajada
    LCD_Init(); //inicialice LCD
    pM = (char *)mensaje1; //puntero a direccion del mensaje 1 con definicion de tipo
}

```

```

for (i=0;i<=15;i++){
    Lcd_4bit_Wr(*pM,1);
    pM++;
}
Lcd_4bit_Wr(0xC0,0);
pM = (char *)mensaje2;
for (i=0;i<=15;i++){
    Lcd_4bit_Wr(*pM,1);
    pM++;
}
i2cCfgI2C(0x7F,0xE0,0x10); //llame función para inicialización del puerto IIC

for(;;) {
    Rote_Col();
    if(bandera0==1){
        bandera0=0;
        Analisis_Tecla();
        Delay(5000000);
        i2cReadE2Prom1Byte(0x00);
        Lcd_4bit_Wr(0xC5,0);
        Lcd_4bit_Wr(tmp,1);
    }
}
/* es necesario asegurarse de nunca abandonar el main */
}

/* Funcion para inicializar el puerto A */
void PTA_Init(char PTAD_Config,char PTADD_Config,char PTAPE_Config,char PTASE_Config,char
PTADS_Config,char PTAIFE_Config){
    PTAD=PTAD_Config; //inicialice estado pinos PTA
    PTADD=PTADD_Config; //inicialice dirección de pinos PTA
    PTAPE=PTAPE_Config; //inicialice estado de pullups PTA
    PTASE=PTASE_Config; //inicialice estado del slew rate PTA
    PTADS=PTADS_Config; //inicialice estado de drive strength PTA
    PTAIFE=PTAIFE_Config; //inicialice estado del filtro PTA
}

/* Funcion para inicializar el puerto F */
void PTF_Init(char PTFD_Config,char PTFDD_Config,char PTFPE_Config,char PTFSE_Config,char
PTFDS_Config,char PTFIFE_Config){
    PTFD=PTFD_Config; //inicialice estado pinos PTF
    PTFDD=PTFDD_Config; //inicialice dirección de pinos PTF
    PTFPE=PTFPE_Config; //inicialice estado de pullups PTF
    PTFSE=PTFSE_Config; //inicialice estado del slew rate PTF
    PTFDS=PTFDS_Config; //inicialice estado de drive strength PTF
    PTFIFE=PTFIFE_Config; //inicialice estado del filtro PTF
}

/* Funcion para inicializar el KBI */
void KBI_Init(char KBI1SC_Config,char KBI1PE_Config,char KBI1ES_Config){
    KBI1PE=KBI1PE_Config; //inicialice registro de habilitacion de pinos KBI
    KBI1SC_KBACK=1; //aclara interrupciones espureas
    KBI1ES=KBI1ES_Config; //inicialice registro de flanco y nivel del KBI
    KBI1SC=KBI1SC_Config; //inicialice registro de estado y control del KBI
}

/* Función para inicializar el puerto IIC */
void i2cCfgI2C(char I2C1F_Config,char IIC1C1_Config,char IIC1S_Config){
    IIC1F=I2C1F_Config; //rate de baudios = (2499945/(2*3840)) = 3255Hz aprox
    IIC1C1=IIC1C1_Config; //modo maestro y habilita interrupcion del IIC
    IIC1S=IIC1S_Config; //borra bandera de perdida de arbitramiento
}

/* Funcion para inicializacion del LCD a 4 bits */
void LCD_Init(void){
    Delay(200000); //hace retardo
    Comando_8bit(0x30); //llama función enviar comando en 8 bits con comando
    Delay(200000); //hace retardo
    Comando_8bit(0x30); //llama función enviar comando en 8 bits con comando
    Delay(200000); //hace retardo
}

```

```

Comando_8bit(0x20);           //LCD 2x16 y manejo a 4 bits
Lcd_4bit_Wr(0x28,0);          //LCD 2x16 y manejo a 4 bits
Lcd_4bit_Wr(0x0C,0);          //ON LCD, OFF cursor
Lcd_4bit_Wr(0x01,0);          //borrar pantalla LCD
Lcd_4bit_Wr(0x06,0);          //desplaza cursor a la derecha con cada caracter
Lcd_4bit_Wr(0x80,0);          //primer caracter fila 1 columna 1
}

/* Funcion para escribir a 4 bits en el LCD */
void Lcd_4bit_Wr(char dato4,char rs){
    if (rs==0){                //pregunte si se va a enviar un comando o un dato
        PTAD_PTAD5=0;           //prepara envío de comando RS = 0
    }
    else{
        PTAD_PTAD5=1;           //prepara envio de dato RS = 1
    }
    a=dato4;                   //almacena temporalmente el dato
    a=a>>4;                   //desplaza parte alta dato para la baja
    PTAD&=0xF0;                //enmascara parte alta del puerto A
    a&=0x0F;                   //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;               //sube pin de enable
    Delay(20000);              //hace retardo
    PTAD|=a;                   //envía nibble alto del dato
    Delay(20000);              //hace retardo
    PTAD_PTAD4=0;               //baja pin de enable
    Delay(20000);              //hace retardo
    PTAD&=0xF0;                //enmascara parte alta del puerto A
    dato4&=0x0F;                //enmascara parte baja del dato a enviar al LCD
    PTAD_PTAD4=1;               //sube pin de enable
    Delay(20000);              //hace retardo
    PTAD|=dato4;               //envía nibble bajo del dato
    Delay(20000);              //hace retardo
    PTAD_PTAD4=0;               //baja pin de enable
    Delay(20000);              //hace retardo
}

/* Función para envio de comando en 8 bits al LCD (solo para inicio) */
void Comando_8bit(char comando8){
    PTAD_PTAD5=0;               //prepara envio de comando
    PTAD&=0xF0;                 //enmascara parte baja del puerto A
    comando8&=0xF0;              //enmascara parte alta de comando a enviar al LCD
    comando8=comando8>>4;       //desplaza para tomar nibble alto
    PTAD_PTAD4=1;               //sube pin de enable
    Delay(20000);               //hace retardo
    PTAD|=comando8;              //envía comando
    Delay(20000);               //hace retardo
    PTAD_PTAD4=0;               //sube pin de enable
    Delay(20000);               //hace retardo
}

/* Funcion Delay(): Retarda basado en una variable tipo entera */
void Delay(unsigned long retardo){
    while(retardo>0){           //llego a cero?
        retardo--;                //no --> decrementa
    }
}

/* Funcion para rotar cero por columnas teclado */
void Rote_Col (void){
    PTFD = 0x0E;                 //Cero a la columna 1
    Delay(500000);               //hace retardo
    PTFD = 0x0D;                 //Cero a la columna 2
    Delay(500000);               //hace retardo
    PTFD = 0x0B;                 //Cero a la columna 3
    Delay(500000);               //hace retardo
}

/* Funcion para hacer un acceso de lectura sobre una direccion de la 24LC02B */
void i2cfReadE2Prom1Byte(byte direccion){
}

```

```

IIC1C1_TX=1; //habilite modo TX por el maestro
IIC1C1_TXAK=0; //no se esperan los ACK ante un envio de la 24LC02B
IIC1C1_MST=1; //genere condicion de START
i2cfWrite(I2C_E2PROM_WRITE); //envie la direccion de la 24LC02B para escritura
i2cfWrite(direccion); //envie direccion de la celda a accesar
IIC1C1_RSTA=1; //genere condicion de START
i2cfWrite(I2C_E2PROM_READ); //envie la direccion de la 24LC02B para lectura
bandera2=0; //aclare bandera de acceso por interrupcion al IIC
IIC1C1_TXAK=1; //se espera recibir ACK
IIC1C1_TX=0; //comutarse a modo de RX del maestro
tmp=IIC1D; //toma dato dummy (yo no es el que se espera)
while(bandera2 == 0); //espere a que se haya dado un acceso por interr IIC
IIC1C1_MST=0; //se comunica a modo esclavo para recibir de la 24LC02B
while(IIC1S_BUSY); //espere a que el IIC se desocupe
tmp=IIC1D; //lea el dato solicitado
}

/* Funcion para escribir un dato a la EEPROM 24LC02B*/
void i2cfWrite(byte dato){
    bandera2=0; //inicie bandera de acceso a la IIC por interrupcion
    IIC1D=dato; //transmita dato a la 24LC02B
    while(bandera2 == 0); //espere que se transmita
}

/* Funcion para hacer un acceso de escritura a una direccion de la EEPROM 24LC02B */
void i2cfWriteE2PROM1Byte (byte direccion,byte dato){
    IIC1C1_TX=1; //habilite modo TX por el maestro
    IIC1C1_TXAK=1; //se espera recibir ACK
    IIC1C1_MST=1; //genere condicion de START
    i2cfWrite(I2C_E2PROM_WRITE); //envie la direccion de la 24LC02B para escritura
    i2cfWrite(direccion); //envie direccion de la celda a accesar
    i2cfWrite(dato); //envie el dato a la celda
    IIC1C1_MST=0; //genere una condicion de STOP
    while(IIC1S_BUSY); //espere a que el IIC se desocupe
}

/* Funcion para analizar tecla presionada y enviarla esclavo */
void Analisis_Tecla(void){
    if(columna==0x0E & fila==1){ //si se detecta columna 1 y fila 1 del teclado
        Lcd_4bit_Wr(0x89,0); //prepare escribir segunda fila primera columna LCD
        Lcd_4bit_Wr(0x31,1); //envíe lo leido al LCD
        i2cfWriteE2PROM1Byte (0x00,0x31); //escriba un "1" en la celda 0x00 de la memoria
        do{ //espere a que se complete la TX
            }while(bandera2 == 0);
        bandera2 = 0; //aclare bandera
    }
    if(columna==0x0D & fila==1){ //si se detecta columna 2 y fila 1 del teclado
        Lcd_4bit_Wr(0x89,0); //prepare escribir segunda fila primera columna LCD
        Lcd_4bit_Wr(0x32,1); //envíe lo leido al LCD
        i2cfWriteE2PROM1Byte (0x00,0x32); //escriba un "2" en la celda 0x00 de la memoria
        do{ //espere a que se complete la TX
            }while(bandera2 == 0);
        bandera2 = 0; //aclare bandera
    }
    if(columna==0x0B & fila==1){ //si se detecta columna 3 y fila 1 del teclado
        Lcd_4bit_Wr(0x89,0); //prepare escribir segunda fila primera columna LCD
        Lcd_4bit_Wr(0x33,1); //envíe lo leido al LCD
        i2cfWriteE2PROM1Byte (0x00,0x33); //escriba un "3" en la celda 0x00 de la memoria
        do{ //espere a que se complete la TX
            }while(bandera2 == 0);
        bandera2 = 0; //aclare bandera
    }
    if(columna==0x0E & fila==2){ //si se detecta columna 1 y fila 2 del teclado
        Lcd_4bit_Wr(0x89,0); //prepare escribir segunda fila primera columna LCD
        Lcd_4bit_Wr(0x34,1); //envíe lo leido al LCD
        i2cfWriteE2PROM1Byte (0x00,0x34); //escriba un "4" en la celda 0x00 de la memoria
        do{ //espere a que se complete la TX
            }while(bandera2 == 0);
        bandera2 = 0; //aclare bandera
    }
}

```

```

if(columna==0x0D & fila==2){
Lcd_4bit_Wr(0x89,0);
Lcd_4bit_Wr(0x35,1);
i2cWriteE2PROM1Byte (0x00,0x35);
do{
}while(bandera2 == 0);
bandera2 = 0;
}
if(columna==0x0B & fila==2){
Lcd_4bit_Wr(0x89,0);
Lcd_4bit_Wr(0x36,1);
i2cWriteE2PROM1Byte (0x00,0x36);
do{
}while(bandera2 == 0);
bandera2 = 0;
}
if(columna==0x0E & fila==3){
Lcd_4bit_Wr(0x89,0);
Lcd_4bit_Wr(0x37,1);
i2cWriteE2PROM1Byte (0x00,0x37);
do{
}while(bandera2 == 0);
bandera2 = 0;
}
if(columna==0x0D & fila==3){
Lcd_4bit_Wr(0x89,0);
Lcd_4bit_Wr(0x38,1);
i2cWriteE2PROM1Byte (0x00,0x38);
do{
}while(bandera2 == 0);
bandera2 = 0;
}
if(columna==0x0B & fila==3){
Lcd_4bit_Wr(0x89,0);
Lcd_4bit_Wr(0x39,1);
i2cWriteE2PROM1Byte (0x00,0x39);
do{
}while(bandera2 == 0);
bandera2 = 0;
}
if(columna==0x0D & fila==4){
Lcd_4bit_Wr(0x89,0);
Lcd_4bit_Wr(0x30,1);
i2cWriteE2PROM1Byte (0x00,0x30);
do{
}while(bandera2 == 0);
bandera2 = 0;
}
Delay(5000000);
}

/* Funcion de atencion a la interrupcion del KBI1 */
interrupt VectorNumber_Vkeyboard void ISR_KBI(void){
columna=PTFD & 0x0F; //capture el valor de la columna
if(PTGD_PTGD0==0){ //si detecto la fila 1
fila=1; //marque fila 1
}
if(PTGD_PTGD1==0){ //si detecto la fila 2
fila=2; //marque fila 2
}
if(PTGD_PTGD2==0){ //si detecto la fila 3
fila=3; //marque fila 3
}
if(PTGD_PTGD3==0){ //si detecto la fila 4
fila=4; //marque fila 4
}
KBI1SC_KBACK = 1; //de reconocimiento de tecla presionada
bandera0=1; //pone bandera de tecla presionada
}

```

```
/* Función de atencion a la interrupcion por IIC */
interrupt VectorNumber_Viic1x void ISR_IIC (void){
    IIC1S_IICIF=1;           //aclare bandera
    bandera2 = 1;             //ponga bandera acceso alcanzado
}
```

### **17.3. REFERENCIAS**

- MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual. Freescale Semiconductor. Rev 1, 2008.
- Microcontroller MCF51JM128 ColdFire Data Sheet. Freescale Semiconductor. Rev 0, 2008.

### **17.4. PREGUNTAS**

- ¿Cuál es la característica más importante de un módulo de comunicaciones sincrónico?
- ¿Será posible implementar una comunicación en modo full-duplex, usando el módulo IIC?
- ¿De cuántas formas un esclavo SPI se conmuta al modo maestro y qué tipo de error se presenta en el módulo?
- ¿En modo IIC, quién devuelve el bit de *acknoledge* cuando se escribe un dato sobre el esclavo?
- ¿Qué significa fase y polaridad 0 en el módulo SPI (CPHA = 0 / CPOL = 0)?
- ¿Cómo es una condición de START en el módulo IIC?

# CAPÍTULO 18

## Comunicación Serial Universal (USB: *Universal Serial Bus*)

**18.1. Diagrama en bloques, explicación del funcionamiento y registros asociados al módulo USB**

**18.2. Ejemplo con el módulo USB: Conexión de un mouse USB al DEMOJM  
(Comunicación como HOST)**

**18.3. Referencias**

## 18.1 DIAGRAMA EN BLOQUES, EXPLICACIÓN DEL FUNCIONAMIENTO Y REGISTROS ASOCIADOS AL MÓDULO USB

- **Breve explicación de la tecnología USB**

En la última década se ha venido dando un importante giro en la interconectividad de periféricos con los PC, de tal manera que interfaces como el puerto paralelo, serial, entre otros, han desaparecido de los computadores.

En su reemplazo, la interfase USB (*Universal Serial Bus*) se ha convertido en la manera más popular para conectar dispositivos periféricos a un computador personal o a otros dispositivos periféricos.

USB aparece en su versión 1.0 en Enero de 1996, después de un gran tiempo en evolución y con un sin número de problemas, que para el año 1998 habían sido solucionados en la versión 1.1. Para el año 2000, se presenta la versión 2.0 libre de errores y con un nuevo conector llamado mini B (como el implementado en la DEMOJM).

Entre las razones más poderosas para adoptar el cambio a USB, se pueden mencionar:

- **Fácil de usar:** En un solo conector se pueden conectar varios periféricos, en vez de tener un conector para cada tipo de periférico.

La configuración de cada dispositivo en el bus es automática (el sistema operativo lo detecta e instala el *driver* apropiado), sin necesidad de reiniciar el sistema.

El usuario no necesita configurar el dispositivo para que este sea reconocido, desde el punto de vista de configuración de IRQ's o direcciones al interior del PC.

La conectividad es simple respecto de la expansión del sistema, sin necesidad de abrir el PC e instalar tarjetas adicionales, basta con adicionar HUB's (concentradores).

Es posible conectar y desconectar dispositivos sin apagar el sistema para su reconocimiento (*plug and play – hot pluggable*).

- **Velocidad:** Se pueden tener tres rangos de velocidad, que superan con creces las velocidades de los buses seriales asíncronos (RS232) e igualan o hasta superan los paralelos. Claro que mientras más dispositivos están colgados al bus, menor será la velocidad promedio de comunicación. Las tres velocidades mencionadas son:

- **HS (High Speed):** Hasta 480Mbps, para cámaras y otros dispositivos que manejan alto tráfico de información a grandes velocidades.

- **FS (Full Speed):** Hasta 12Mbps, para el reemplazo de puertos serials y paralelos tradicionales, como también para sistemas embebidos (caso del DEMOJM).
- **LS (Low Speed):** Hasta 1.5Mbps, para dispositivos con cables poco apantallados y bastante flexibles, como teclados y ratones
- **Confiabilidad:** Diseño simple de su hardware, con el fin de minimizar los errores por ruido aditivo y protocolos robustos, que pueden detectar errores y pedir retransmisiones.
- **Bajo costo:** Cables y componentes de muy bajo costo, por su diseño simple y limpio.
- **Bajo consumo:** Con su característica de poder entrar en modos de ahorro de energía (modos de WAIT, SLEEP o STANDBY) y sólo responder cuando sea requerido. Característica bastante importante en sistemas soportados por batería, en donde cada milivatio cuenta.

Pero USB también presenta limitaciones y que a continuación se mencionan:

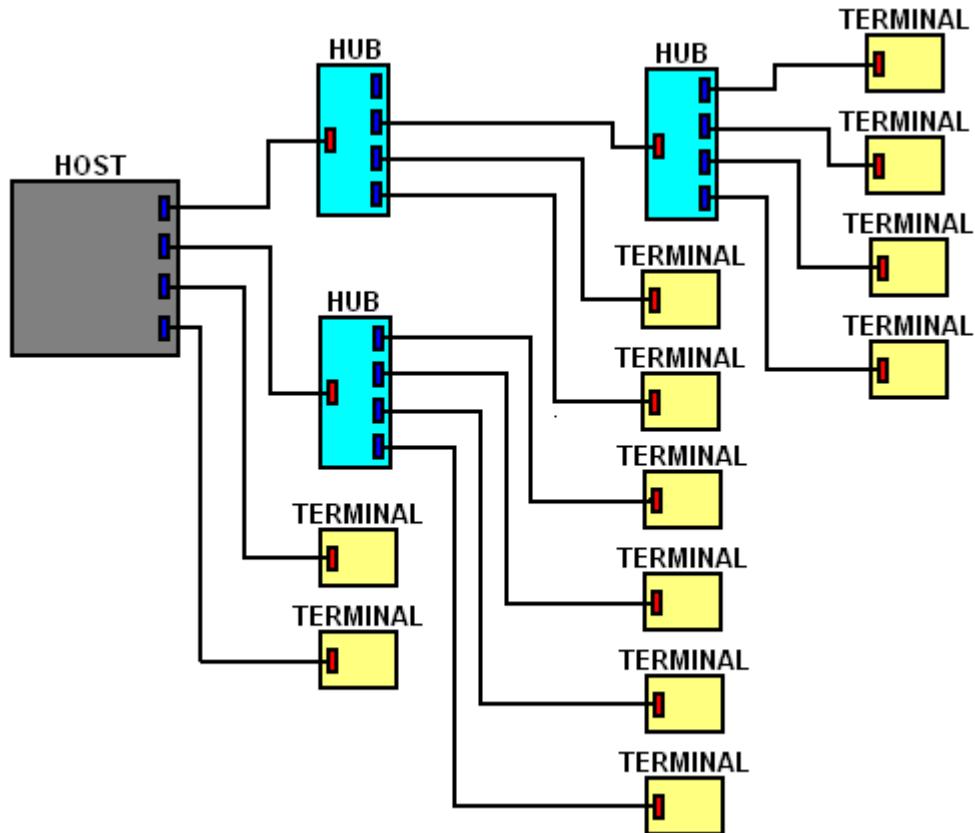
- **Ausencia en el soporte de los sistemas viejos:** Hoy en día se consiguen interfaes entre los viejos sistemas, como RS232, RS485, puerto paralelo, entre otros, y el sistema USB, pero no siempre funcionan bien en las aplicaciones de los usuarios.
- **Límites en la velocidad:** USB no puede competir con buse de velocidades mayores a 400Mbps, como la IEEE-1394b que tiene tasas de transferencia de hasta 3.2Gbps y otras más rápidas de actualidad.
- **Límites de distancia:** USB ha sido diseñado para trabajo en escritorio, en donde distancias de hasta 5mts son muy buenas, pero no podría competir con los viejos sistemas como RS232 y 485, que desarrollan distancias de varios cientos de metros. USB podría desarrollar hasta 30 metros, pero necesita el gasto adicional de los HUB's.

La Figura 18.1 detalla la topología primaria de un bus USB. Esta topología se basa en conexión estrella, con conectividad mediante HUB's y puntos terminales (*endpoint*), que conforman los periféricos unidos al bus.

Nunca un HUB se puede adicionar a un dispositivo, siempre tendrá que estar conectado a otro HUB y este se encarga de pasar las direcciones de los periféricos en la red, servir de repetidor del tráfico que recibe (en ambas direcciones), manejar la alimentación de otros dispositivos inteligentemente y dosificar la velocidad de transferencia de información.

Las tareas del HOST en la topología planteada son:

- **Detección de los dispositivos:** Cuando el sistema reinicia, es tarea del HOST identificar todo dispositivo conectado a la red. Este trabajo es llamado proceso de numeración y mediante este, el HOST direcciona cada dispositivo y pide información adicional. Cada que un dispositivo se conecta o desconecta de la red, el HOST deberá actualizar la numeración.



**Figura 18.1. Topología de una red USB.**

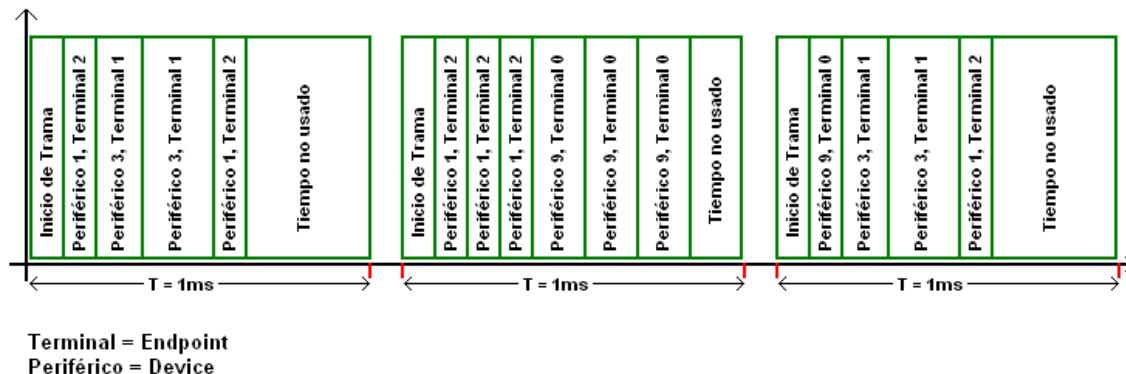
- **Manejo del flujo de datos:** Si en un determinado momento, varios periféricos desean transferir información, el HOST. Esto lo hace dividiendo el tiempo en segmentos llamados tramas (o microtramas). Cada trama tendrá un tiempo fijo de dedicación, para garantizar un ancho de banda equitativo para todos los terminales activos en la red.
- **Chequeeo de errores:** El HOST deberá estar pendiente de todo error que se presente en la red.
- **Suministro de la alimentación:** El cable USB dedica dos líneas para transmitir la energía a la red, siendo estas +5V y *Ground*. La corriente total disponible por cada puerto USB es de 500mA, que a 5V serían 2.5Watts.

- **Intercambio de datos con los periféricos:** Dentro de todos los trabajos asignados a un HOST, está el de soportar el intercambio de datos con los periféricos, asignando *drivers* especializados para esto.

Las tareas de los periféricos en la topología planteada son:

- **Detección de una comunicación:** Cada periférico debe monitorear el bus de comunicaciones y saber cuando su dirección está contenida en una trama. Si la dirección coincide con la propia, el dispositivo almacena el dato en el *buffer* de recepción y se produce un evento de interrupción.
- **Respuesta a requerimientos estándar:** Un evento de encendido (*On Power Up*), cuando el dispositivo se conecta a la red (proceso de numeración), ante una petición de información del estado, requerimientos del fabricante, etc, el periférico deberá responder al HOST.
- **Chequeo de errores:** Al igual que el HOST, el periférico deberá estar pendiente de todo error que se presente en la red.
- **Manejo de la alimentación:** Un periférico puede alimentar o ser alimentado y manejar los modos de bajo consumo ante espacios de inoperancia.
- **Intercambio de datos el HOST:** Dentro de todos los trabajos asignados a un periférico, está el de soportar el intercambio de datos con el HOST. El HOST deberá continuamente hacer *polling* sobre todos los periféricos conectados al bus o cuando la aplicación lo requiera. Para cada transferencia, el periférico deberá responder enviando un código que indique cuando la transferencia fue aceptada o indicando si se encuentra ocupado.

La Figura 18.2 ilustra sobre una trama típica USB con un intervalo de tiempo de 1 milisegundo, para un sistema FS (*Full Speed*).



**Figura 18.2. Trama USB FS.**

Cada transferencia USB puede contener varios paquetes y cada paquete contiene información. En cada transferencia participan los siguientes actores:

- **Dispositivo terminal (*device endpoint*):** Toda transmisión viaja desde o hacia los terminales y allí se encuentran los *buffer* de almacenamiento de información, que podría ser para enviar o que se ha recibido.

Del otro lado se encuentra el HOST (servidor de la red) , también con su *buffer* de almacenamiento de información, que podría ser para enviar o que se ha recibido.

Todo terminal tiene un número de identificación y una dirección. El número de identificación es un valor entre 0 y 16, la dirección se asigna según la aplicación embebida en el HOST.

Un terminal puede tener una dirección como entrada de datos (IN) y otra como salida de datos (OUT), por ejemplo el terminal 1 podría tener una dirección (IN) para recibir información desde el HOST, pero también podría tener otra dirección (OUT) para enviar información hacia el HOST. La Tabla 18.1 resume la dirección de las transacciones desde el punto de vista del HOST.

**Tabla 18.1. Tipo de transacción USB.**

Tipo de Transacción	Fuente del Dato	Tipo de Transferencia de la Transacción	Contenido
IN	Terminal	Todo tipo	Datos genéricos
OUT	HOST	Todo tipo	Datos genéricos
SETUP	HOST	Control	Requerimiento

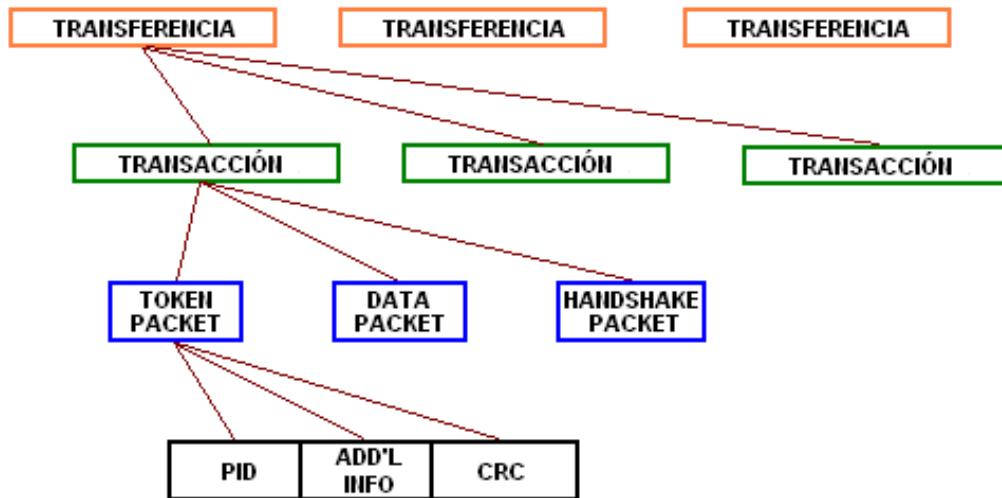
- **Tuberías de conexión de terminales con el HOST (*Pipes*):** Una tubería no es un objeto físico, es más bien una asociación entre los dispositivos y el *software* de control del HOST.

La información de configuración recibida por el HOST, incluye un descriptor por cada terminal. Un descriptor es un bloque de información que le dice al HOST lo necesario a cerca del terminal en cuestión, para poder establecer comunicación con este.

Esto incluye la dirección del terminal, el tipo de transferencia a establecer, el tamaño máximo del paquete de datos y el intervalo entre transferencias.

El HOST determina si el ancho de banda disponible es adecuado para todas las transferencias configuradas, de no ser así, el HOST niega la configuración requerida y volverá a intentarlo más adelante.

- **Tipos de transferencias:** Una transferencia consta de una o más transacciones y una transacción consta de uno, dos o tres paquetes. Las transferencias son controladas por la SIE (*Serial Interface Engine*), que no es más que la máquina de estados del controlador USB. La Figura 18.3 muestra un paquete de transferencias y transacciones.



**Figura 18.3. Formato de una transferencia USB.**

En donde:

**TOKEN PACKET:** Paquete de información que identifica el receptor y el tipo de transacción.

**DATA PACKET:** Paquete con los datos (*payload*) en la comunicación.

**HANDSHAKE PACKET:** Paquete que contiene información sobre el estado y control de la transacción en la comunicación USB. Las señales que se manejan aquí son:

- **ACK (acknowledge):** Señal que indica que quien recibe los datos lo ha logrado sin error.
- **NAK (BUSY) (no acknowledge):** Indica que quien recibe se encuentra ocupado y quien envía deberá intentar más tarde la transferencia.
- **STALL:** Puede significar que el requerimiento o solicitud no puede soportarse, que el requerimiento ha fallado o que el terminal ha fallado.

**CRC:** Código de Redundancia Cíclica para comprobación de errores.

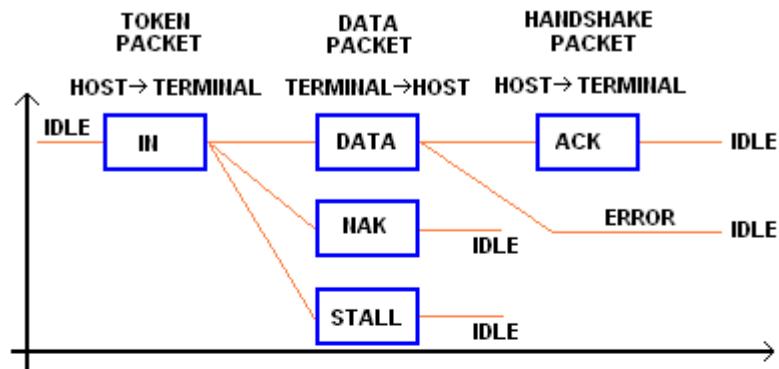
## ADD'L INFO: Información adicional

**PID:** Identificación del paquete contenido en información sobre la transacción en curso.

Dependiendo de la tasa de transferencia, tiempos de respuesta y corrección de errores, las transferencias USB pueden ser de los siguientes tipos:

- **Transferencia en masa (Bulk Transfer):** Es usada cuando el tiempo no es una variable crítica. Con este tipo de transferencia se puede enviar una gran cantidad de datos, por ejemplo en el envío de datos hacia una impresora o la recolección de una imagen desde una cámara. La Figura 18.4 detalla el formato para una transacción tipo *bulk*.
- **Transferencia por interrupción (Interrupt Transfer):** Es usada cuando se tiene un tiempo fijo para la transferencia del dato y que generalmente no se puede ignorar ni alargar. La Figura 18.4, también cumple para este tipo de transacción.

### Bulk tipo IN : (El HOST solicita información)



### Bulk tipo OUT : (El HOST envía información)

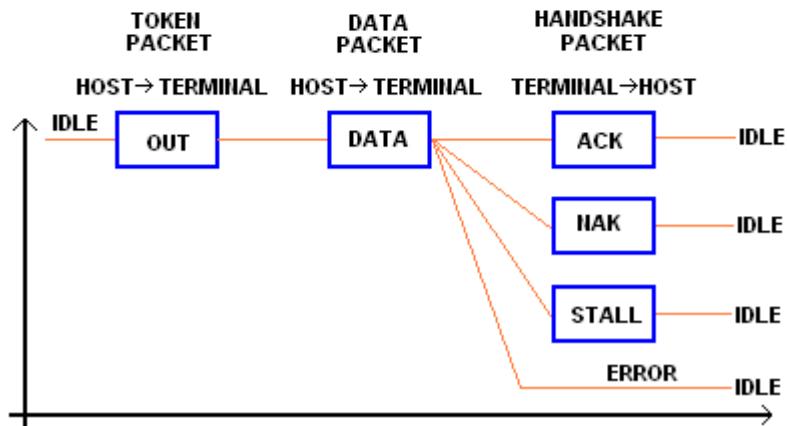


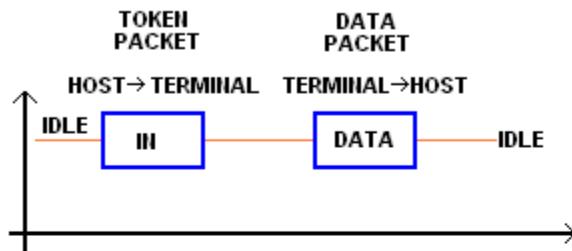
Figura 18.4. Formato de una transferencia *Bulk* o por Interrupción.

- **Transferencia isocrónica (Isochronous Transfer):** Es usada cuando se tienen transferencias en tiempo real, en donde los datos llegan a una tasa constante.

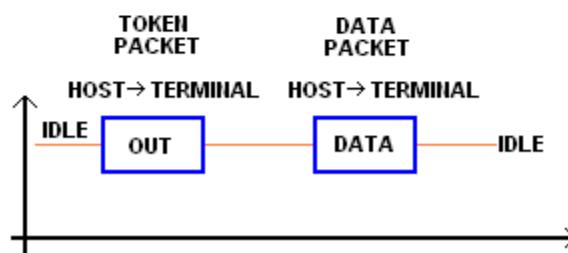
En este tipo de transferencia se soportan algunos errores, sin posibilidad de retransmitir ante la ocurrencia de estos y es por esto que se considera más rápida que la transferencia por interrupción.

Las aplicaciones más comunes de este tipo de transferencia son el video y la música en tiempo real. La Figura 18.5 detalla el formato para una transacción tipo *Isochronous*.

#### Bulk tipo IN : (El HOST solicita información)



#### Bulk tipo OUT : (El HOST envía información)



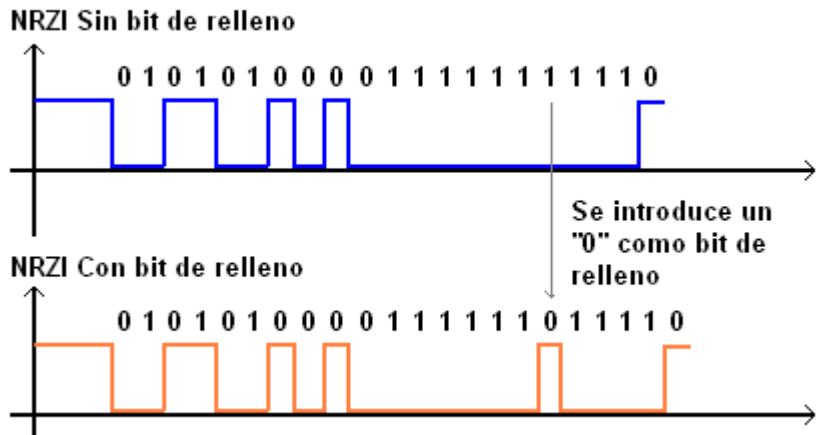
**Figura 18.5. Formato de una transferencia Isocrónica.**

- **Codificación de los datos:** Todo dato que aparece en las líneas del bus es codificado en formato de NRZI (*Non Return to Zero Inverted*), que consiste en cambiar el nivel de la señal con cada bit en cero que se vaya a transmitir. Cuando el bit a transmitir es un “1”, la señal no cambia de nivel (ver Figura 18.6).

Cuando se tiene una serie de “1’s” seguidos se correría con el riesgo de perder el sincronismo, porque el protocolo no cuenta con bits adicionales de sincronía. La técnica para no perder la sincronía consiste en introducir un bit de relleno (*Bit*

*Stuffing*) o introducir campos de sincronización, los cuales introducen algo de carga innútil a la comunicación (aproximadamente un 17%).

El receptor en el bus USB se sincroniza con las transiciones de los bits, de tal manera que una trama larga de “1’s” haría que el sincronismo se pierda. La técnica es introducir cada 6 “1’s” consecutivos un bit de relleno en “0” (ver Figura 18.6), esto asegura que por lo menos cada 6 bits ocurre una transición.



**Figura 18.6. Codificación NRZI.**

- **Señales eléctricas y distribución de pines en conectores USB:** Existen dos tipos de conectores llamados Series A/B, Mini A y Mini B, cuya distribución de pines se presenta en la Tabla 18.2.

La serie A corresponde al usado en los HOST o en los HUB y la serie B se utiliza en los terminales.

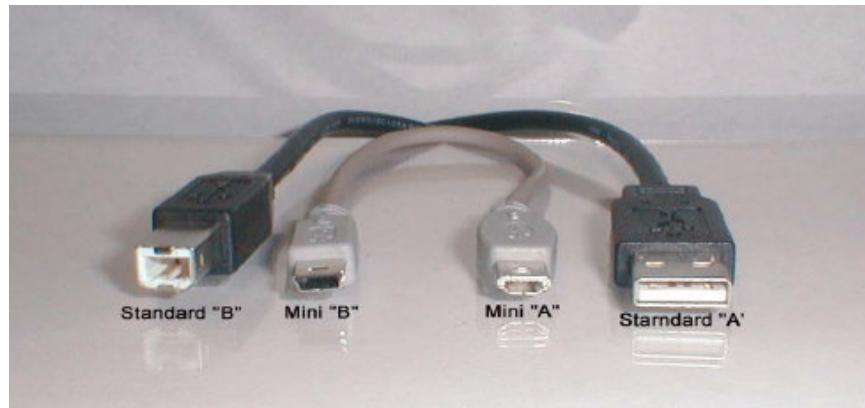
La serie mini AB se origina debido al tamaño de las series A/B, para aplicaciones de poco espacio (Ver Figura 18.7) y para aplicaciones *On The Go* (OTG), para identificar el HOST.

**Tabla 18.2. Distribución de pines en conectores USB.**

Series A/B	Mini AB	Señal	Color Cable
1	1	VBUS (+5V)	Rojo
2	2	D-	Blanco
3	3	D+	Verde
4	5	GND	Negro
—	4	Pin ID*	—
Coraza	—	Blindaje	Malla

\* Pin opcional para identificación del dispositivo

Los cables pueden tener una distancia de hasta 5 mts y la fuente (VBUS) puede proporcionar hasta 100mA aplicados a un periférico.



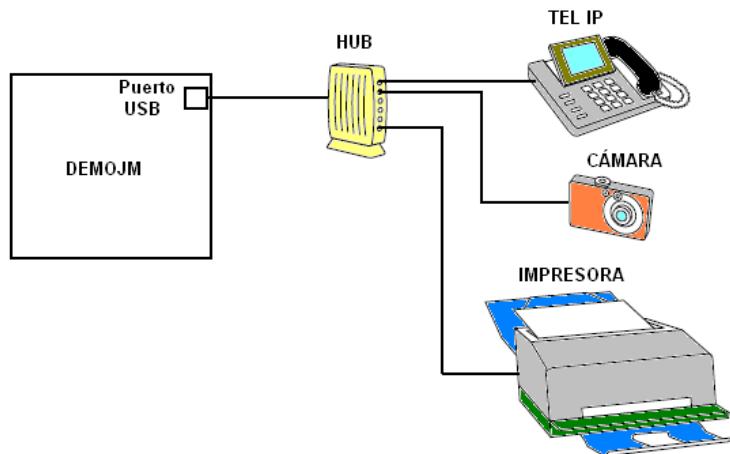
**Figura 18.7. Conectores USB.**

**NOTA:** Es altamente recomendado para el usuario tener a la mano los documentos [usb\\_20.pdf](#) y [USB\\_OTG\\_1-3.pdf](#), que encontrarán en el sitio: <http://www.usb.org/developers/docs>

- **Características más relevantes del módulo USB de la familia JM**

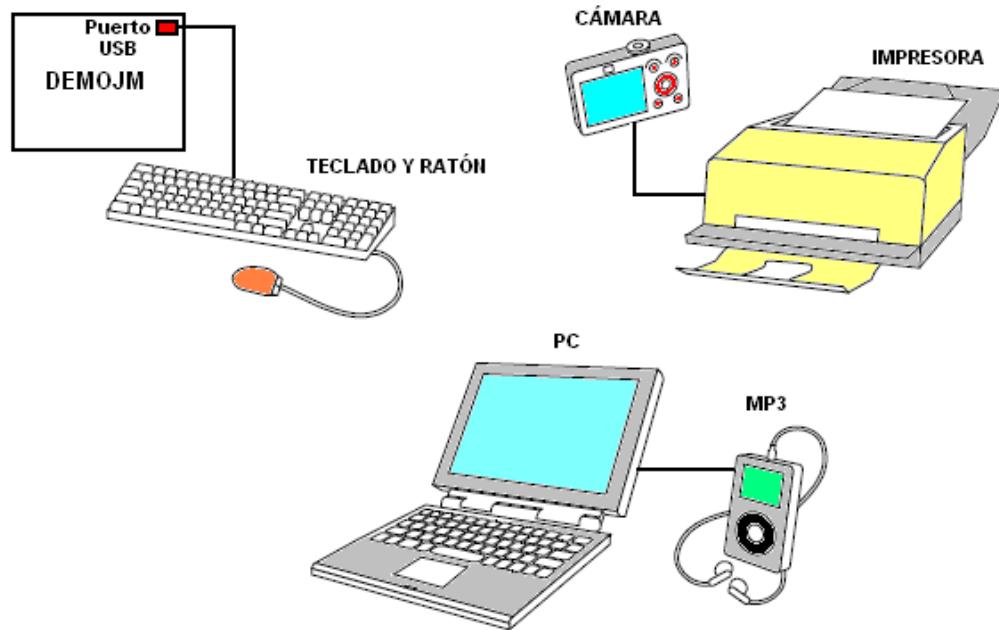
El módulo USB implementado en la máquina ColdFire® MCF51JM128 cumple con dos modos de operación, que son:

- **USB 1.1/2.0 FS:** En este modo es posible la definición de un único HOST y conectados a este una variedad de dispositivos terminales como teclados, ratones, impresoras, entre otros. La Figura 18.8 muestra una configuración típica en este modo.



**Figura 18.8. Modo USB 1.1/2.0**

- USB revision 2.0/OTG (*On The Go*): Orientado a comunicar un par de dispositivos entre sí (*peer to peer*). La Figura 18.9 muestra una configuración típica en este modo. OJO DEFINIR MEJOR OTG.

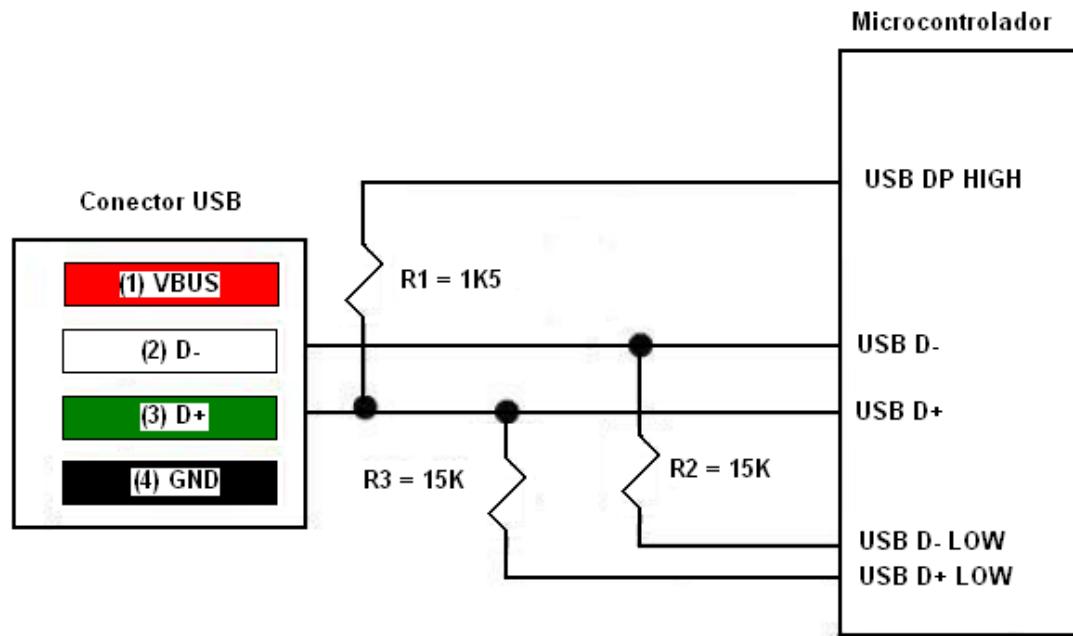


**Figura 18.9. Modo USB OTG**

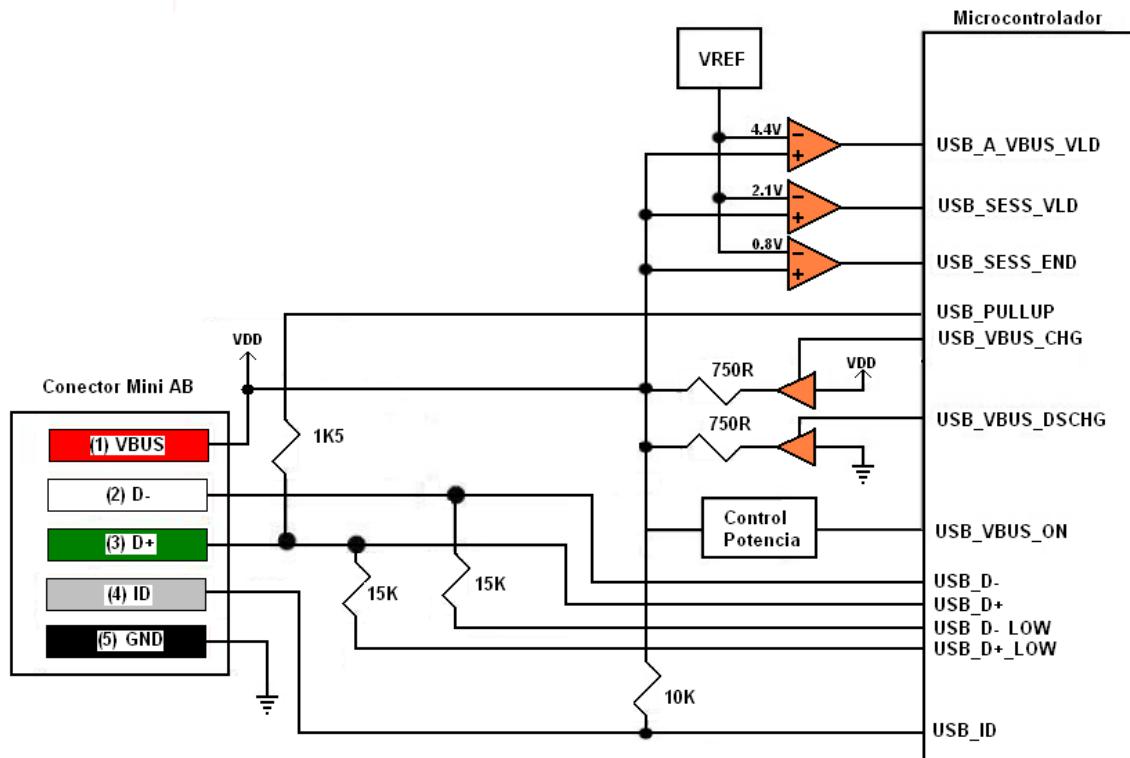
Dentro de las características más relevantes se tienen:

- Compatible con USB 1.1 y 2.0 FS (*Full Speed*).
- Conexión con 16 terminales.
- Interfase de datos a DMA o lista FIFO.
- Bajo consumo de energía.
- Lógica para protocolo OTG.

Para la conexión con dispositivos externos se tienen las siguientes configuraciones (ver Figuras 18.10 y 18.11).



**Figura 18.10. Conexión USB en PULL/UP - PULL/DOWN.**



**Figura 18.11. Conexión USB OTG a conector Mini AB.**

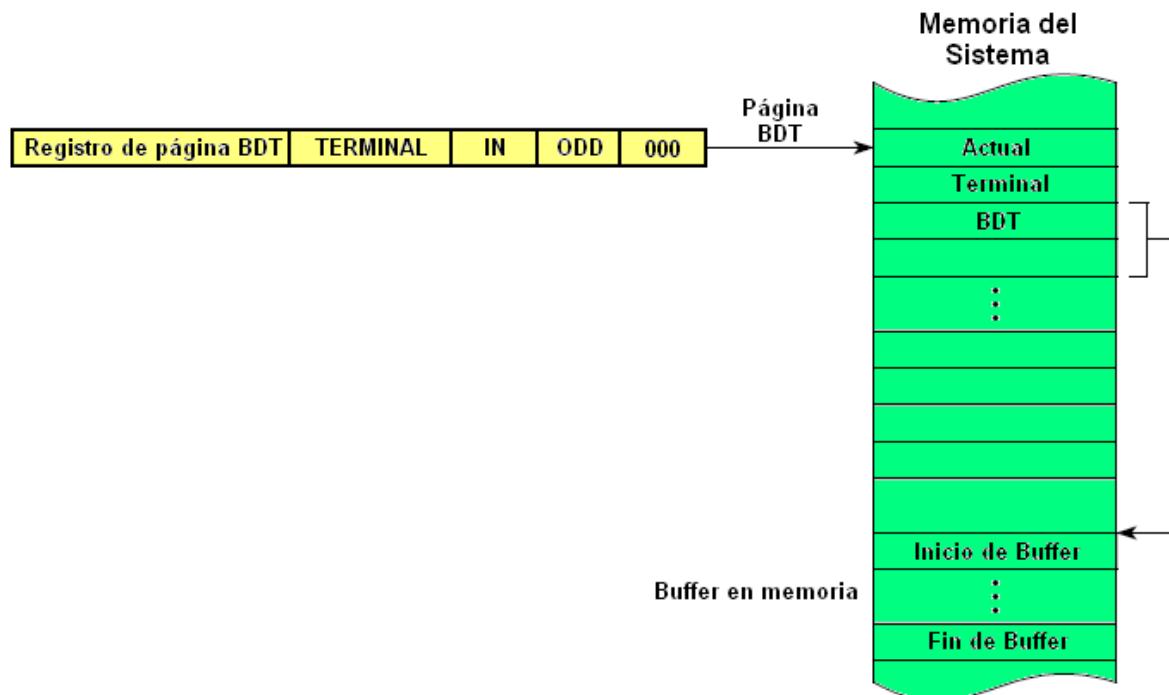
- **Estructura de los datos e interfase con el programador**

La transferencia de información desde o hacia un terminal es manipulada por una estructura llamada la tabla de descriptores de *buffer* (BDT: *Buffer Descriptor Table*), la cual reserva un espacio de 512 bytes en la memoria y que utiliza para su direccionamiento registros punteros a BDT.

Cada dirección de un terminal requiere un registro descriptor de 2 bytes, de tal manera que para 16 posibles terminales son necesarios 256 bytes. Pero como el sistema permite que tanto el procesador (MCF51JM128) como el módulo USB-FS, manipulen de manera independiente un descriptor de *buffer* (*Double Buffer*). A uno de ellos se le asignarían las direcciones pares y al otro las impares, entonces son necesarios 512 bytes de memoria y no 256.

Para determinar quién tiene el mecanismo de BDT en un momento determinado, el sistema implementa un bit llamado semáforo. Cuando el bit del semáforo vale “0” es el procesador quien tiene el BDT, pero si el bit de semáforo vale “1” es el módulo USB-FS quien lo tiene.

El mecanismo de descriptores de buffer utiliza direccionamiento indirecto, con sus respectivos punteros. La Figura 18.12 detalla un bloque de memoria utilizado en el mecanismo de BDT.



**Figura 18.12. Memoria dedicada al mecanismo BDT.**

Para cada dirección de un terminal son necesarios 16 bytes (bloque) y es el registro de página del BDT quien apunta a la dirección de inicio del bloque. De tal manera que si un dispositivo USB recibe el testigo (TOKEN), el sistema USB-FS deberá leer el correspondiente BD (*Buffer Descriptor*) y determinar si se refiere al propio.

Para calcular la dirección de entrada a la tabla de *buffer's* descriptores, el registro de página BDT es concatenado con el terminal actual, la señal IN (1: Tx y 0: Rx) y el campo ODD. Con esto se conforma una gran dirección de 32 bits, que se presenta en la Figura 18.13 y cuyos campos son explicados en la Tabla 18.3.



**Figura 18.13. Cálculo de la dirección BDT.**

**Tabla 18.3. Componentes de la dirección BDT.**

Campo	Descripción
PÁGINA XX BDT	Registros de página BDT en el Bloque de Registros de Control
TERMINAL	Campo del terminal con el testigo (TOKEN) USB
IN	Si IN = 1 se va a transmitir (TX), Si IN = 0 se va a recibir (RX)
ODD	Este bit es mantenido junto con el bit USB-FS SIE e indica el buffer que está siendo usado

- **Formato del descriptor de *buffer* (BD)**

Este descriptor contiene información de control del *buffer*, tanto para el módulo USB-FS como para el microcontrolador. El BD tiene diferentes maneras de ser interpretado, dependiendo de quién lo está leyendo.

Si quien lo interpreta es el módulo USB-FS, los datos allí almacenados determinan:

- Quién es el propietario del *buffer* en el sistema de memoria.
- PID para dato tipo 0 o dato tipo 1.
- Liberación del *buffer* ante la finalización de transferencia de paquete.
- Sin incremento de memoria (modo FIFO).
- Habilitación de la sincronización del cambio de estado (Toggle) de los datos.
- Cuantos datos se han recibido o transmitido.
- Donde se encuentra el *buffer* en el sistema de memoria.

Si quien lo interpreta es el microcontrolador, los datos allí almacenados determinan:

- Quién es el propietario del *buffer* en el sistema de memoria.
- PID para dato tipo 0 o dato tipo 1.
- El PID del testigo (TOKEN) recibido.
- Cuantos datos se han recibido o transmitido.
- Donde se encuentra el *buffer* en el sistema de memoria.

La Figura 18.14 ilustra sobre el formato del BD.

31:26	25:16	15:8	7	6	5	4	3	2	1	0
RSVD	BC (10-Bits)	RSVD	OWN	DATA0/1	KEEP/ TOK_PID[3]	NINC/ TOK_PID[2]	DTS/5 TOK_PID[1]	BDT_STALL/ TOK_PID[0]	0	0
Dirección del Buffer (32-Bits)										

**Figura 18.14. Formato del BD.**

En donde:

- **RSVD:** Reservado.
- **BC (Byte Count):** Contador de bytes. La máquina USB-FS SIE cambia el contenido de este campo, al completarse una recepción y almacena allí el número de bytes recibidos.
- **OWN:** Si OWN = 1, el USB-FS tiene acceso exclusivo al BD. Si OWN = 0, el microcontrolador tiene acceso exclusivo al BD. Por lo general la máquina USB-FS SIE escribe “0” a este bit al completarse el paquete del testigo (TOKEN PACKET), excepto cuando el bit KEEP = 1. El USB-FS ignora los demás campos del BD cuando OWN = 0, caso en el que el microprocesador tiene pleno acceso.
- **DATA0/1:** Define cuando un campo de dato es tipo 0 (dato transmitido) o tipo 1 (dato recibido). Este bit no puede ser modificado por el USB-FS.
- **KEEP/TOK\_PID[3]:** Si KEEP = 1 , después de que el bit OWN sea “1” el BD permanecerá como USB-FS siempre. Si KEEP = 0 el BD puede ser liberado por el USB-FS, cuando un testigo (TOKEN) ha sido procesado. Por lo general este bit permanece en “1”, siempre que se tengan terminales trabajando bajo el modo FIFO. El microcontrolador no es informado cuando un testigo está siendo procesado, el dato es simplemente transferido desde o hacia la lista FIFO. Normalmente el bit NINC es “1” cuando KEEP = 1, para prevenir incrementos en las direcciones del *buffer*. Mientras KEEP es “1”, éste no podrá ser cambiado por el USB-FS, de otra manera el bit 3 del actual PID de testigo (TOKEN) es escrito, después, por el USB-FS dentro del BD.
- **NINC/TOK\_PID[2]:** El bit de no incremento, deshabilita el incremento de dirección de la máquina DMA. Esta acción fuerza a la DMA a leer o escribir sobre la misma dirección. Esta opción es útil para terminales en donde los datos se deben leer o escribir en una sola localización, como en un *buffer* FIFO. Este bit, junto con el bit KEEP, son puestos a “1” para terminales que trabajan con lista FIFO. Cuando KEEP = 1, el bit NINC no puede ser cambiado por el USB-FS, de otra manera el bit 2 del actual PID de testigo (TOKEN) es escrito, después, por el USB-FS dentro del BD.
- **DTS/TOK\_PID[1]:** Poniendo este bit a “1” se habilita al USB-FS para hacer la sincronización del cambio de estado (Toggle) de los datos. Mientras KEEP es “1”, el bit DTS no podrá ser cambiado por el USB-FS, de otra manera el bit 1 del actual PID de testigo (TOKEN) es escrito, después, por el USB-FS dentro del BD.
- **BDT\_STALL/TOK\_PID[0]:** Poniendo a “1” este bit se causa que el USB-FS emita una señal de STALL, si se ha recibido el testigo (TOKEN) por el SIE, el cual podrá usar el BDT en esta localización. El BDT no es consumido por el SIE,

- cuando el bit STALL es “1”. Si KEEP es “1”, el bit STALL no podrá ser cambiado por el USB-FS, de otra manera el bit 0 del actual PID de testigo (TOKEN) es escrito, después, por el USB-FS dentro del BD.
- **TOK\_PID[n]:** Estos bits pueden representar el actual PID del TOKEN. Al terminarse una transferencia el actual PID TOKEN es escrito, por el USB-FS dentro del BD. Los valores que se escriben después de la transferencia son los valores del PID de TOKEN yson: 0x1 para especificar un TOKEN tipo OUT, 0x9 para especificar un TOKEN tipo IN o 0xD para especificar un TOKEN tipo SETUP.

En modo HOST este campo es usado para reportar el último PID devuelto o una indicación del estado de una transferencia. Para este caso se tienen los siguientes códigos:

- **0x3:** especifica un dato tipo 0 (RX).
  - **0xB:** especifica un dato tipo 1 (TX).
  - **0x2:** especifica un ACK (acknowledge).
  - **0xE:** especifica una condición de STALL.
  - **0xA:** especifica un NAK (BUSY).
  - **0x0:** especifica un vencimiento de tiempo en el bus USB (*Bus Timeout*).
  - **0xF:** especifica un dato con error.
- **ADDR[31:0]:** Representa la dirección en 32 bits del BD en el sistema de memoria y estos bits no pueden ser cambiados por el USB-FS.

- **Secuencia de una transacción USB para la máquina MCF51JM128**

- **Cálculo de la dirección del BDT:** Al transmitir o recibir datos, el USB-FS calcula la dirección del BDT.
- **Tranferencia del dato vía SIE:** Después de que el BDT ha sido leído y con el bit OWN =1, la SIE transfiere el dato vía DMA o lista FIFO, desde o hacia el *buffer* apuntado en el campo ADDR del BD.
- **Actualización del BDT:** Cuando el TOKEN se ha completado, el USB-FS actualiza el BDT, cambiando el bit OWN a”0”, siempre y cuando el bit KEEP sea “0”. El registro STAT (registro de estado que se verá más adelante) es actualizado y la interrupción TOK\_DNE (el TOKEN se ha completado) es generada. Cuando el microcontrolador procesa la interrupción TOK\_DNE, este lee el registro de estado para obtener toda la información necesaria para procesar el terminal. En este punto, el procesador localiza un nuevo BD como un dato adicional que puede ser transmitido o que se podrá recibirse del terminal.

La Figura 18.15 ilustra el proceso de un TOKEN.

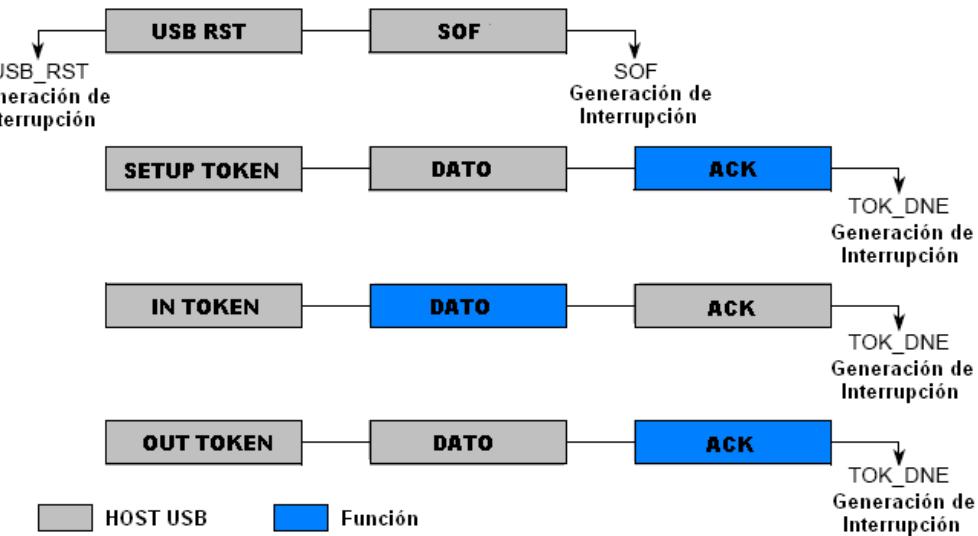


Figura 18.15. Transacción del testigo en el bus USB.

Después de haberse dado una transacción del TOKEN, puede ocurrir un error por sobremarcha (*overrun*) en el DMA. La primera causa de este posible error es debido a los retardos (*latency*) en la memoria sobre el interface iniciador del BVCI (*Basic Virtual Component Interface*), que podría hacer que el FIFO receptor tenga un sobreflujo (*overflow*). La segunda causa es que el paquete recibido esté por fuera del tamaño negociado, lo que podría acasionar una pérdida del flujo del *software* (*BUG*).

Para el primer caso el USB responderá con una señal de NAK o bus *timeout* (BTO). El bit **ERR\_STAT[DMA\_ERR]** se pone a “1”, tanto para el HOST como para el terminal. Dependiendo de los valores de los registros **INT\_ENB** y **ERR\_ENB**, la CPU podría habilitar un estado de interrupción y notificar al procesador del error en el DMA. En el modo terminal, el BDT no se escribirá porque se ha dado una interrupción **TOK\_DNE**. Lo anterior es debido a que se asume que se dará un segundo intento y tendrá éxito en el futuro. En el modo HOST, la interrupción por **TOK\_DEN** ocurre y el campo **TOK\_PID** del BDT se pone a “1111” para indicar el error por retardos en el DMA. El software del HOST deberá decidir si intentar o moverse a otro asunto en el procedimiento.

Para el segundo caso, de paquetes muy grandes, la especificación de USB es ambigua. Una sobremarcha no es conveniente para los retardos (*latency*) en la memoria, pero es conveniente en la falta de espacio para poner datos de exceso. La generación de una señal NAK puede causar otra retransmisión del paquete, que se encuentra por fuera de tamaño. En respuesta a paquetes por fuera del tamaño, la CPU continúa emitiendo señales de ACK, cuando se trata de transferencias no isocrónicas (*Bulk* e Interrupción). El dato escrito en la memoria es marcado como un paquete de máximo tamaño y no como un dato corrupto. La CPU pone a “1” el bit

ERR\_STAT[DMA\_ERR] y un evento de interrupción tipo TOK\_DNE ocurrirá. El campo TOK\_PID del BDT no se pone a “1111” debido a que el evento de DMA\_ERR no fue debido al retardo (*latency*). El campo de longitud del paquete, escrito con anterioridad hacia el BDT, representa la longitud del dato actual escrito en la memoria. El *software* podrá decidir una serie de acciones, para evitar que futuras transferencias tengan: paros del terminal, cancelación de una transferencia, deshabilitar el terminal, etc

- **Registros asociados al módulo USB**

- **Registro del ID del periférico (PER\_ID):** La Figura 18.16 ilustra sobre el registro del ID del periférico del módulo USB.

**Registro de ID del periférico (PER\_ID): (FFFF9A00)**

	7	6	5	4	3	2	1	0
Lectura	0	0	ID5	ID4	ID3	ID2	ID1	ID0
Escritura								
Reset:	0	0	0	0	0	1	0	0

**Figura 18.16. Registro PER\_ID.**

**IDx:** Bits para la identificación del periférico. Por defecto siempre se lee como un periférico 0x04.

- **Registro de ID complementario (ID\_COMP):** La Figura 18.17 ilustra sobre el registro de ID complementario del módulo USB.

**Registro de ID complementario (ID\_COMP): (FFFF9A04)**

	7	6	5	4	3	2	1	0
Lectura	1	1	NID5	NID4	NID3	NID2	NID1	NID0
Escritura								
Reset:	1	1	1	1	1	0	1	1

**Figura 18.17. Registro ID\_COMP.**

**NIDx:** Bits como complemento a “1” del ID del periférico.

- **Registro de la revisión del periférico(REV):** La Figura 18.18 ilustra sobre el registro de revisión del módulo USB.

### Registro de revisión del USB (REV): (FFFF9A08)

	7	6	5	4	3	2	1	0
Lectura	REV7	REV6	REV5	REV4	REV3	REV2	REV1	REV0
Escritura								
Reset:	0	0	1	1	0	0	1	1

**Figura 18.18. Registro REV.**

**REVx:** Bits que indican el número de revisión del núcleo del USB.

- **Registro de información adicional del periférico (ADD\_INFO):** La Figura 18.19 muestra el registro de información adicional del módulo USB.

### Registro de información adicional del periférico (ADD\_INFO): (FFFF9A0C)

	7	6	5	4	3	2	1	0
Lectura	IRQ_NUM					0	0	IEHOST
Escritura								
Reset:	0	0	0	0	0	0	0	1

**Figura 18.19. Registro ADD\_INFO.**

- **IRQ\_NUM:** Número de interrupción asignada al periférico.
- **IEHOST:** Este bit es puesto a “1” cuando está habilitado el modo HOST.

- **Registro del estado de la interrupción en modo OTG (OTG\_INT\_STAT):** Este registro almacena los cambios en el ID y las señales del VBUS (Bus de alimentación USB). El software deberá leer este registro, para determinar qué evento ha generado esta interrupción. Sólo los bits que han cambiado desde la última lectura se ponen a “1”. Para borrar el estado de un bit, basta con escribir un “1” en este. La Figura 18.20 muestra el registro del estado de interrupción en modo OTG.

### Registro de estado de interrupción en modo OTG (OTG\_INT\_STAT): (FFFF9A10)

	7	6	5	4	3	2	1	0
Lectura	ID_CHG	1_MSEC	LINE_STATE_CHG	Reserved	SESS_VLD_CHG	B_SESS_CHG	Reserved	A_VBUS_CHG
Escritura				—			—	
Reset:	0	1	0	X	0	0	X	0

**Figura 18.20. Registro OTG\_INT\_STAT.**

- **ID\_CHG:** Este bit es puesto a “1” cuando se presenta un cambio de estado en el pin ID del conector USB.  
**0:** No se ha presentado un cambio en el pin ID del conector USB.  
**1:** Se ha presentado un cambio en el pin ID del conector USB.

- **1\_MSEG:** Este bit indica cuando ha pasado un milisegundo. Este bit debe ser aclarado cada que se necesite verificar el milisegundo.  
 0: No se ha completado un tiempo de 1ms.  
 1: Se ha completado un tiempo de 1ms.
  - **LINE\_STAT\_CHG:** Este bit indica cuando la línea de estado del USB cambia. La interrupción asociada a este bit se puede utilizar para detectar un estado de RESET, resumen, conexión y datos como señales pulsantes.  
 0: No se ha detectado un cambio en la línea de estado USB.  
 1: detectado un cambio en la línea de estado USB.
  - **SESS\_VLD\_CHG:** Este bit indica cuando se ha detectado un cambio en la línea VBUS indicando que hay una sesión válida.  
 0: No se ha detectado una sesión válida.  
 1: Se ha detectado una sesión válida.
  - **B\_SESS\_CHG:** Este bit indica cuando se ha detectado un cambio en la línea VBUS por un terminal tipo B.  
 0: No se ha detectado un cambio por un terminal tipo B.  
 1: Se ha detectado un cambio por un terminal tipo B.
  - **A\_SESS\_CHG:** Este bit indica cuando se ha detectado un cambio en la línea VBUS por un terminal tipo A.  
 0: No se ha detectado un cambio por un terminal tipo A.  
 1: Se ha detectado un cambio por un terminal tipo A.
- **Registro del control de la interrupción en modo OTG (OTG\_INT\_EN):** Este registro habilita bit a bit las posibles interrupciones del modo OTG. La Figura 18.21 muestra el registro de control de interrupción en modo OTG.

**Registro de control de interrupciones en modo OTG (OTG\_INT\_EN): (FFFF8014)**

Lectura Escritura	7	6	5	4	3	2	1	0
	ID_EN	1_MSEC_EN	LINE_STATE_EN	Reserved	SESS_VLD_EN	B_SESS_EN	Reserved	A_VBUS_EN
Reset:	0	0	0	X	0	0	X	0

**Figura 18.21. Registro OTG\_INT\_EN.**

- **ID\_EN:** Habilita la interrupción por evento ID.  
 0: La interrupción por evento ID está deshabilitada.  
 1: La interrupción por evento ID está habilitada.
- **1\_MSEG\_EN:** Habilita la interrupción por evento 1\_MSEG.  
 0: La interrupción por evento 1\_MSEG está deshabilitada.  
 1: La interrupción por evento 1\_MSEG está habilitada.
- **LINE\_STAT\_EN:** Habilita la interrupción por evento LINE\_STAT\_CHG.  
 0: La interrupción por evento LINE\_STAT\_CHG está deshabilitada.  
 1: La interrupción por evento LINE\_STAT\_CHG está habilitada.
- **SESS\_VLD\_CHG:** Habilita la interrupción por evento SESS\_VLD\_CHG.

- **0:** La interrupción por evento SESS\_VLD\_CHG está deshabilitada.  
**1:** La interrupción por evento SESS\_VLD\_CHG está habilitada.
  - **B\_SESS\_EN:** Habilita la interrupción por evento B\_SESS\_CHG.  
**0:** La interrupción por evento B\_SESS\_CHG está deshabilitada.  
**1:** La interrupción por evento B\_SESS\_CHG está habilitada.
  - **A\_SESS\_EN:** Habilita la interrupción por evento A\_SESS\_CHG.  
**0:** La interrupción por evento A\_SESS\_CHG está deshabilitada.  
**1:** La interrupción por evento A\_SESS\_CHG está habilitada.
- o **Registro de estado de interrupción (OTG\_STAT):** Este registro muestra el valor actual desde las salidas del comparador externo del pin ID y el VBUS. La Figura 18.22 muestra el registro de control de interrupción en modo USB A/B.

#### Registro de estado de interrupción (OTG\_STAT): (FFFF9A18)

Lectura Escritura	7	6	5	4	3	2	1	0
	ID	1_MSEC_EN	LINE_STATE_STABLE	Reserved	SESS_VLD	B_SESS_END	Reserved	A_VBUS_VLD
Reset:	0	0	0	X	0	0	X	0

Figura 18.22. Registro OTG\_STAT.

- **ID:** Este bit es puesto a “1” cuando se presenta un cambio de estado en el pin ID del conector USB.  
**0:** Indica que un cable tipo A ha sido conectado al conector USB.  
**1:** Indica que no se ha establecido una conexión o que un cable tipo B ha sido conectado al conector USB.
- **1\_MSEG\_EN:** Este bit no se usa.
- **LINE\_STATE\_STABLE:** Este bit indica que las líneas internas, que controlan el LINE\_STAT\_CHG se han estabilizado por al menos 1 milisegundo. Primero se lee el bit LINE\_STAT\_CHG y luego se lee el LINE\_STATE\_STABLE.  
**0:** La línea LINE\_STAT\_CHG no está estable.  
**1:** La línea LINE\_STAT\_CHG está estable.
- **SESS\_VLD:** Este bit indica cuando una sesión es válida.  
**0:** El voltaje VBUS está por debajo del umbral de una sesión válida tipo B.  
**1:** El voltaje VBUS corresponde al umbral de una sesión válida tipo B.
- **B\_SESS\_END:** Este bit indica cuando una sesión tipo B ha terminado.  
**0:** El voltaje VBUS corresponde al umbral de un final de una sesión tipo B.  
**1:** El voltaje VBUS está por debajo del umbral de un final de una sesión tipo B.
- **A\_VBUS\_VLD:** Este bit indica cuando el VBUS tipo A es válido.  
**0:** El voltaje VBUS está por debajo del umbral válido del VBUS tipo A.

**1:** El voltaje VBUS corresponde al umbral válido del VBUS tipo A.

- **Registro del control OTG (OTG\_CTRL):** Este registro controla la operación del VBUS y de las resistencias asociadas a la línea de datos del USB. La Figura 18.23 muestra el registro de control OTG.

#### Registro de control OTG (OTG\_CTRL): (FFFF9A1C)

Lectura Escritura	7	6	5	4	3	2	1	0
	DP_HIGH	RSVD	DP_LOW	DM_LOW	VBUS_ON	OTG_EN	VBUS_CHG	VBUS_DSCHG
Reset:	0	0	0	0	0	0	0	0

Figura 18.23. Registro OTG\_CTRL.

- **DP\_HIGH:** Habilita el *pullup* para la línea D+.  
**0:** El *pullup* para la línea D+ no está habilitado.  
**1:** El *pullup* para la línea D+ está habilitado.
- **DP\_LOW:** Habilita el *pulldown* para la línea D+.  
**0:** El *pulldown* para la línea D+ no está habilitado.  
**1:** El *pulldown* para la línea D+ está habilitado.
- **DM\_LOW:** Habilita el *pulldown* para la línea D-.  
**0:** El *pulldown* para la línea D- no está habilitado.  
**1:** El *pulldown* para la línea D- está habilitado.
- **VBUS\_ON:** Señal de encendido del VBUS.  
**0:** La señal de encendido del VBUS no está puesta .  
**1:** El VBUS está encendido.
- **OTG\_EN:** Habilita una resistencia pullup/pulldown para el modo OTG.  
**0:** Si el bit USB\_EN está en “1” y el bit HOST\_MODE es “0”, se habilita una resistencia de *pullup* para el pin D+. Si el bit HOST\_MODE es “1”, se ubican *pulldown*’s para las líneas D+ y D-.  
**1:** Los controles sobre los *pullup*’s y *pulldown*’s son usados.
- **VBUS\_CHG:** Inserta una resistencia a la señal de VBUS.  
**0:** No inserta una resistencia a la señal de VBUS.  
**1:** Inserta una resistencia a la señal de VBUS.
- **VBUS\_DSCHG:** Inserta una resistencia de descarga a la señal de VBUS.  
**0:** No inserta una resistencia de descarga a la señal de VBUS.  
**1:** Inserta una resistencia de descarga a la señal de VBUS.

- **Registro de estado de interrupción (INT\_STAT):** Contiene una serie de bits para cada evento de interrupción del módulo USB y pertenecen a una sola fuente de interrupción del microcontrolador ColdFire® V1. Para aclarar cada bit, es necesario escribirlo a “1” una vez se halla atendido el evento de interrupción. La Figura 18.24 muestra el registro de estado de interrupción.

### Registro de estado de interrupción (INT\_STAT): (FFFF9A80)

Lectura Escritura	7	6	5	4	3	2	1	0
	STALL	ATTACH	RESUME	SLEEP	TOK_DNE	SOF_TOK	ERROR	USB_RST
Reset:	0	0	0	0	0	0	0	0

Figura 18.24. Registro INT\_STAT.

- **STALL:** Cuando el dispositivo está en modo objetivo (*Target*), este bit se pone a “1” cuando el SIE (*Serial Interface Engine*) envía una señal de STALL. Cuando se está en modo HOST , este bit se pone a ”1” cuando el módulo USB detecta una señal de STALL. Esta interrupción puede ser usada para determinar si la última transacción USB fue completada con éxito o si paró (STALL).
  - **ATTACH:** Interrupción por enganche. Este bit es puesto a “1” cuando el módulo USB detecta un enganche de un dispositivo USB. Esta señal sólo será valida si el bit HOST\_MODE\_EN =1. esta interrupción indica que un periférico está conectado y deberá ser configurado.
  - **RESUME:** Este bit es puesto a “1” dependiendo de las señales DP/DM y puede ser usado como una señal remota para despertar el dispositivo pegado al bus USB.
  - **SLEEP:** Este bit es puesto a “1” cuando el módulo USB detecta un evento de modo vago (IDLE), durante 3 milisegundos. Este temporizador es aclarado al detectarse actividad en el bus USB.
  - **TOK\_DNE:** Este bit se pone a “1” cuando se ha completado el proceso actual de testigo (TOKEN). El núcleo del ColdFire® V1 deberá leer, inmediatamente, el registro STAT para determinar el terminal y el BD usados para el TOKEN. Aclarar este bit causa que el registro STAT sea borrado o que el estado retenido del registro STAT sea almacenado en el registro STAT.
  - **SOFT\_TOK:** Este bit se pone a “1” cuando módulo USB recibe una señal de TOKEN de inicio de trama (*Start Of Frame*).
  - **ERROR:** Este bit es puesto a “1” cuando alguna de las condiciones de error en el registro ERR\_STAT ocurre. El núcleo del ColdFire® V1 deberá leer, inmediatamente, el registro ERR\_STAT para determinar la fuente del error.
  - **USB\_RST:** Este bit es puesto a “1” cuando el módulo USB a decodificado un RESET válido. La acción anterior informa al microcontrolador que podrá escribir un 0x00 dentro del registro de direcciones y habilitar el terminal 0.
- **Registro de habilitación de interrupciones (INT\_ENB):** Contiene una serie de bits para habilitar cada fuente de interrupción del módulo USB. La Figura 18.25 muestra el registro de habilitación de las interrupciones.

### Registro de habilitación de interrupciones (INT\_ENB): (FFFF9A84)

	7	6	5	4	3	2	1	0
Lectura	STALL_EN	ATTACH_EN	RESUME_EN	SLEEP_EN	TOK_DNE_EN	SOF_TOK_EN	ERROR_EN	USB_RST_EN
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 18.25. Registro INT\_ENB.

- **STALL\_EN:** Bit para habilitar interrupción por un evento tipo STALL.  
**0:** Inhibe interrupción por evento STALL.  
**1:** Habilita interrupción por evento STALL.
  - **ATTACH\_EN:** Bit para habilitar interrupción por un evento tipo ATTACH.  
**0:** Inhibe interrupción por evento ATTACH.  
**1:** Habilita interrupción por evento ATTACH.
  - **RESUME\_EN:** Bit para habilitar interrupción por un evento tipo RESUME.  
**0:** Inhibe interrupción por evento RESUME.  
**1:** Habilita interrupción por evento RESUME.
  - **SLEEP\_EN:** Bit para habilitar interrupción por un evento tipo SLEEP.  
**0:** Inhibe interrupción por evento SLEEP.  
**1:** Habilita interrupción por evento SLEEP.
  - **TOK\_DNE\_EN:** Bit para habilitar interrupción por un evento tipo TOK\_DNE.  
**0:** Inhibe interrupción por evento TOK\_DNE.  
**1:** Habilita interrupción por evento TOK\_DNE.
  - **SOF\_TOK\_EN:** Bit para habilitar interrupción por un evento tipo SOF\_TOK.  
**0:** Inhibe interrupción por evento SOF\_TOK.  
**1:** Habilita interrupción por evento SOF\_TOK.
  - **ERROR\_EN:** Bit para habilitar interrupción por un evento tipo ERROR.  
**0:** Inhibe interrupción por evento ERROR.  
**1:** Habilita interrupción por evento ERROR.
  - **USB\_RST\_EN:** Bit para habilitar interrupción por un evento tipo USB\_RST.  
**0:** Inhibe interrupción por evento USB\_RST.  
**1:** Habilita interrupción por evento USB\_RST.
- **Registro de estado de interrupción por error (ERR\_STAT):** Contiene una serie de bits para detectar cada fuente de interrupción por error en el módulo USB. Todos estos bits conforman una OR alambrada y esta a su vez afecta el bit INT\_STAT[ERROR]. Los bits pueden ser borrados escribiendo un “1” en estos. La Figura 18.26 muestra el registro de detección de las interrupciones.

### Registro de estado de interrupción por error (ERR\_STAT): (FFFF9A88)

	7	6	5	4	3	2	1	0
Lectura	BTS_ERR	Reserved	DMA_ERR	BTO_ERR	DFN8	CRC16	CRC5_EOF	PID_ERR
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 18.26. Registro ERR\_STAT.

- **BTS\_ERR:** Este bit se pone a “1” en la detección de error por evento de BIT STUFF (Bit de relleno para sincronía).
  - **DMA\_ERR:** Este bit se pone a “1” si el módulo USB hace un requerimiento al DMA de la lectura de un nuevo BDT, pero no fue posible obtener el dato desde el bus, tanto para lectura como para escritura. Si se está procesando una transferencia tipo TX (IN), esta puede causar una condición de bajo-flujo (*underflow*). Si se está procesando una transferencia tipo RX (IN), esta puede causar una condición de sobreflujo (*overflow*). Este tipo de condición es muy usada en el desarrollo de arbitramiento de dispositivos, tanto para el microprocesador como para el módulo USB. Lo anterior minimiza la demanda en el requerimiento de bus y tiempos de inactividad muy grandes (*grand latency*). Este bit también es puesto a “1” si un paquete de datos supera el tamaño máximo especificado en el BDT, para este caso la máquina trunca el paquete y lo almacena en el *buffer* de memoria.
  - **BTO\_ERR:** Este bit se pone a “1” en la detección de error por evento de un tiempo por fuera (*timeout*) en la devolución del bus (*bus turnaround*). El módulo USB contiene un temporizador para el bus *turnaround*.
  - **DFN8:** Este bit se pone a “1” si el campo del dato recibido no tiene 8 bits de longitud.
  - **CRC16:** Este bit se pone a “1” cuando un paquete es rechazado por un error en el chequeo cíclico redundante CRC16.
  - **CRC5\_EOF:** Este bit tiene dos funciones: La primera es que si el Módulo USB está operando en modo periférico (HOST\_MODE\_EN = 0) y se detecta un error de redundancia cíclica tipo CRC5, el bit se pone a “1”. La segunda función es que si el módulo está operando en modo HOST (HOST\_MODE\_EN = 1) y se detecta un error por fin de trama (EOF), este bit se pone a “1”. Este tipo de interrupción se usa en el *software* para el despacho de paquetes de información y garantizar que no ocurran cruces en las transacciones, al comienzo de una nueva trama.
  - **PID\_ERR:** Este bit se pone a “1” cuando el PID de una transacción falla.
- 
- **Registro de habilitación de interrupción por error (ERR\_ENB):** Contiene una serie de bits para habilitar cada fuente de interrupción por error en el módulo USB. Poniendo a “1” uno de estos bits, habilita la respectiva interrupción en el

registro ERR\_STAT. La Figura 18.27 muestra el registro de habilitación de las interrupciones.

#### Registro de habilitación de interrupciones por error (ERR\_ENB): (FFFF9A8C)

Lectura Escritura	7	6	5	4	3	2	1	0
	BTS_ERR _EN	Reserved —	DMA_ERR _EN	BTO_ERR _EN	DFN8_EN	CRC16_EN	CRC5_EOF _EN	PID_ERR _EN
Reset:	0	0	0	0	0	0	0	0

Figura 18.27. Registro ERR\_ENB.

- **BTS\_ERR\_EN:** Bit para habilitar la interrupción por evento de BTS\_ERR.  
 0: Inhibe interrupción por evento BTS\_ERR.  
 1: Habilita interrupción por evento BTS\_ERR.
  - **DMA\_ERR\_EN:** Bit para habilitar la interrupción por evento de DMA\_ERR.  
 0: Inhibe interrupción por evento DMA\_ERR.  
 1: Habilita interrupción por evento DMA\_ERR.
  - **BTO\_ERR\_EN:** Bit para habilitar la interrupción por evento de BTO\_ERR.  
 0: Inhibe interrupción por evento BTO\_ERR.  
 1: Habilita interrupción por evento BTO\_ERR.
  - **DFN8\_EN:** Bit para habilitar la interrupción por evento de DFN8.  
 0: Inhibe interrupción por evento DFN8\_EN.  
 1: Habilita interrupción por evento DFN8\_EN.
  - **CRC16\_EN:** Bit para habilitar la interrupción por evento de CRC16.  
 0: Inhibe interrupción por evento CRC16\_EN.  
 1: Habilita interrupción por evento CRC16\_EN.
  - **CRC5\_EOF\_EN:** Bit para habilitar la interrupción por evento de CRC5\_EOF.  
 0: Inhibe interrupción por evento CRC5\_EOF.  
 1: Habilita interrupción por evento CRC5\_EOF.
  - **PID\_ERR\_EN:** Bit para habilitar la interrupción por evento de PID\_ERR.  
 0: Inhibe interrupción por evento PID\_ERR.  
 1: Habilita interrupción por evento PID\_ERR.
- o **Registro de estado (STAT):** Este registro reporta el estado de las transacciones del módulo USB. Cuando el núcleo del procesador ColdFire® V1 ha recibido una interrupción por TOK\_DNE, el registro de estado puede leerse para determinar el estado de la comunicación con el terminal previo. Cuando se aclara el bit TOK\_DNE, ocasiona que se actualice el registro STAT desde el SIE. La Figura 18.28 muestra el registro de estado.

### Registro de estado (STAT): (FFFF9A90)

	7	6	5	4	3	2	1	0
Lectura	ENDP				TX	ODD	Reserved	Reserved
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 18.28. Registro STAT.

- **ENDP[3:0]:** Estos 3 bits codifican la dirección del terminal que ha recibido o transmitido el TOKEN anterior.
  - **TX:** Bit para indicar que tipo de transacción fue la más reciente.  
 0: La transacción más reciente fue una operación de recepción.  
 1: La transacción más reciente fue una operación de transmisión.
  - **ODD:** Este bit se pone a “1” cuando la última actualización del descriptor de buffer fue en el banco impar de la BDT.
- o **Registro de control (CTL):** Es utilizado para suministrar funciones de control sobre el módulo USB. La Figura 18.29 muestra el registro de control.

### Registro de control (CTL): (FFFF9A94)

	7	6	5	4	3	2	1	0
Lectura	JSTATE	SE0	TXSUSPEND/ TOKENBUSY	RESET	HOST_MODE_EN	RESUME	ODD_RST	USB_EN/ SOF_EN
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 18.29. Registro CTL.

- **JSTATE:** Señal para permitir recepción en modo diferencial. La polaridad de esta señal es afectada por el estado actual de la línea LS\_EN.
- **SE0:** Señal para permitir modo SEZ (*Single Ended Zero*).
- **TXSUSPEND/TOKENBUSY:** Cuando el USB está en modo HOST y está en la etapa de ejecución del testigo (TOKE), TOKEN\_BUSY es puesta a “1”. Esto se hace para inhibir que más comandos tipo TOKEN sean escritos en el registro de TOKEN. El software deberá comprobar este bit, para evitar la pérdida de comandos del tipo TOKEN. En modo terminal el bit TXD\_SUSPEND en puesto a “1” cuando el SIE ha deshabilitado la transmisión y recepción de paquetes. Aclarando este bit hace que el SIE continúe con el proceso de TOKEN.
- **RESET:** Este bit en “1” permite al módulo USB generar un RESET al sistema y a los periféricos. Esta señal de control es únicamente válida en modo HOST (HOST\_MODE\_EN = 1). El software debe controlar el tiempo que se requiera que la señal de RESET esté puesta.
- **HOST\_MODE\_EN:** Bit para habilitar el sistema como HOST.  
 0: Módulo USB está en modo terminal.  
 1: Módulo USB está en modo HOST.

- **RESUME:** Este bit habilita el sistema para hacer señalización de resumen, permitiendo al módulo hacer despertar remoto (*remote wakeup*). El software debe controlar el tiempo que se requiera que la señal de RESUME esté puesta.
- **ODD\_RST:** Este bit aclara todos los bits ODD de las BDT, haciendo que todos los bancos queden como EVEN.
- **USB\_EN/SOF\_EN:** Bit para habilitar la operación del BDT. Cuando este bit es puesto a “1” causa que el SIE aclare todos los bits ODD (de los BDT) y por otro lado, causa que la lógica del SIE reinicialice. Aclarando este bit, en el modo HOST, causa que el SIE pare el envío de TOKEN'S tipo SOF.  
**0:** Módulo USB está deshabilitado.  
**1:** Módulo USB está habilitado.
- o **Registro de dirección (ADDR):** Este registro retiene la única dirección USB, que el módulo USB decodifica, cuando se encuentra en modo terminal (HOST\_MODE\_EN = 0). Cuando se opera en modo HOST (HOST\_MODE\_EN = 1) el módulo USB transmite esta dirección con un paquete tipo TOKEN, esto habilita el módulo USB para direccional un único periférico. En cualquiera de los modos, el bit USB\_EN deberá estar en “1”. Observe como ante un RESET el registro ADDR se pone al valor de 0x00, según la especificación USB. La Figura 18.30 muestra el registro de dirección.

#### Registro de dirección (ADDR): (FFFF9A98)

Lectura Escritura	7	6	5	4	3	2	1	0
	LS_EN				ADDR			
Reset:	0	0	0	0	0	0	0	0

Figura 18.30. Registro ADDR.

- **LS\_EN:** Bit para selección de baja velocidad. Este bit informa al módulo USB que el próximo comando de TOKEN, escrito en el registro de TOKEN, será ejecutado a baja velocidad. Esto habilita al módulo USB para hacer el preámbulo necesario requerido para transmisión de datos a baja velocidad.
- **ADDR:** Dirección del módulo USB. Define la dirección del USB en modo terminal (periférico) o la dirección de transmisión en modo HOST.
- o **Registro de página 1 del BDT (BDT\_PAGE\_01):** Registro de página 1 de la tabla de descriptores de *buffer*, conteniendo la dirección de la actual tabla de descriptores de *buffer* en la memoria. La Figura 18.31 muestra el registro de página 1 del BDT.

#### Registro de página 01 del BDT (BDT\_PAGE\_01): (FFFF9A9C)

	7	6	5	4	3	2	1	0
Lectura	BDT_BA15	BDT_BA14	BDT_BA13	BDT_BA12	BDT_BA11	BDT_BA10	BDT_BA9	NOT USED
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 18.31. Registro BDT\_PAGE\_01.

**BDT\_BA[15:9]:** Bits para determinar la dirección base del BDT, la cual define dónde se encuentra la tabla de descriptores de buffer en la memoria.

- **Registro de número de trama baja/alta (FRM\_NUML, FRM\_NUMH):** Registro que contiene los 8 bits bajos y altos, usados para el cálculo de la dirección de la actual tabla de descriptores de *buffer*. La Figura 18.32 muestra el registro de número de trama baja/alta.

#### Registros de número de trama (FRM\_NUML:FRM\_NUMH): (FFFF9AA0:FFFF9AA4)

	7	6	5	4	3	2	1	0
Lectura	FRM7	FRM6	FRM5	FRM4	FRM3	FRM2	FRM1	FRM0
Escritura								
Reset:	0	0	0	0	0	0	0	0
	7	6	5	4	3	2	1	0
Lectura	0	0	0	0	0	FRM10	FRM9	FRM8
Escritura								
Reset:	0	0	0	0	0	0	0	0

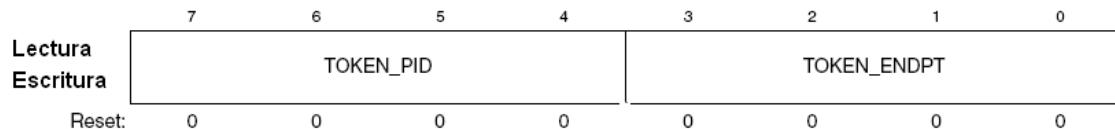
Figura 18.32. Registros FRM\_NUML y FRM\_NUMH.

**FRMx:** Bits para representar el número de la trama.

- **Registro de funciones del testigo (TOKEN):** Este registro es utilizado para realizar transacciones del tipo USB cuando se está en modo HOST. Cuando el procesador ColdFire® V1 desea ejecutar una transacción hacia un terminal, este escribe el tipo de TOKEN y el número de terminal en este registro. Después de que este registro ha sido escrito el módulo USB comienza la transacción especificada, hacia la dirección contenida por el registro de dirección (ADDR). El núcleo del procesador ColdFire® V1 deberá, siempre, verificar que no esté en “1” el bit de TOKEN\_BUSY en el registro de control (CTL). Esto se debe hacer antes de escribir al registro TOKEN. El registro de dirección (ADDR) y el registro de control 0 del terminal (ENDPT0), son también utilizados cuando se está realizando un comando TOKEN y por lo tanto deberán ser escritos antes que el registro de TOKEN. El registro de direcciones (ADDR) es usado para hacer la selección correcta de la dirección del periférico, transmitida por el comando del

TOKEN. El registro de control del terminal determina la señalización y las directivas de reintentos durante la transferencia. La Figura 18.33 muestra el registro de TOKEN.

#### Registro del testigo (TOKEN): (FFFF9AA8)



**Figura 20.33. Registro TOKEN.**

- **TOKEN\_PID[3:0]:** Estos bits representan el tipo de TOKEN ejecutado por el módulo USB y estos son:
  - TOKEN\_PID = 0001:** TOKEN tipo OUT, el módulo USB está realizando una transacción de transmisión (TX).
  - TOKEN\_PID = 1001:** TOKEN tipo IN, el módulo USB está realizando una transacción de recepción (RX).
  - TOKEN\_PID = 1101:** TOKEN tipo SETUP, el modulo USB está realizando una transacción de SETUP (TX).
- **TOKEN\_ENDPT:** Estos 4 bits retienen la dirección del terminal para el comando TOKEN.
- o **Registro de umbral de inicio de trama (SOF\_THLD):** Este registro es usado sólo en modo HOST. El registro es un contador de 14 bits, que cuenta el intervalo entre tramas SOF. Una trama SOF deberá ser transmitida cada milisegundo. El contador del SOF es cargado con un valor de 12000 y cuando este alcanza el valor de 0 el TOKEN tipo SOF es transmitido.

Este registro deberá ser programado a un valor que asegure que otros paquetes no estén en actividad de transmisión, cuando el temporizador SOF\_THLD ha llegado a cero. Cuando el contador de SOF alcanza el valor de umbral, no se transmitirán más TOKEN's hasta que el SOF haya sido transmitido.

El valor programado dentro del SOF\_THLD deberá reservar suficiente tiempo para asegurar que en el peor de los casos, la transacción se lleve a cabo. Unos valores típicos para el SOF\_THLD son:

- Para un paquete de 64 bytes: SOF\_THLD = 74
- Para un paquete de 32 bytes: SOF\_THLD = 42
- Para un paquete de 16 bytes: SOF\_THLD = 26

La Figura 18.34 muestra el registro SOF\_THLD.

#### Registro de umbral del SOF (SOF\_THLD): (FFFF9AAC)

	7	6	5	4	3	2	1	0
Lectura	CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 18.34. Registro SOF\_THLD.

**CNT[7:0]:** Estos bits representan el contador del SOF en términos de tiempos de byte.

- **Registro de página 2 del BDT (BDT\_PAGE\_02):** Registro de página 2 de la tabla de descriptores de *buffer*, conteniendo la dirección de la actual tabla de descriptores de *buffer* en la memoria. La Figura 18.35 muestra el registro de página 2 del BDT.

#### Registro de página 2 del BDT (BDT\_PAGE\_02): (FFFF9AB0)

	7	6	5	4	3	2	1	0
Lectura	BDT_BA23	BDT_BA22	BDT_BA21	BDT_BA20	BDT_BA19	BDT_BA18	BDT_BA17	BDT_BA16
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 18.35. Registro BDT\_PAGE\_02.

**BDT\_BA[23:16]:** Bits para determinar la dirección base del BDT, la cual define dónde se encuentra la tabla de descriptores de buffer en la memoria.

- **Registro de página 3 del BDT (BDT\_PAGE\_03):** Registro de página 3 de la tabla de descriptores de *buffer*, conteniendo la dirección de la actual tabla de descriptores de *buffer* en la memoria. La Figura 18.36 muestra el registro de página32 del BDT.

#### Registro de página 3 del BDT (BDT\_PAGE\_03): (FFFF9AB4)

	7	6	5	4	3	2	1	0
Lectura	BDT_BA31	BDT_BA30	BDT_BA29	BDT_BA28	BDT_BA27	BDT_BA26	BDT_BA25	BDT_BA24
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 18.36. Registro BDT\_PAGE\_03.

**BDT\_BA[31:24]:** Bits para determinar la dirección base del BDT, la cual define dónde se encuentra la tabla de descriptores de buffer en la memoria.

- **Registros de control de terminales 0 - 15 (ENDPT0-15):** Contiene los bits de control para cada una de las 16 terminales. La Figura 18.37 ilustra sobre un registro típico de control de terminal y el listado que relaciona la dirección de todos estos registros.

#### Registros de control de terminales (ENDPT0-15)

Lectura Escritura		7	6	5	4	3	2	1	0
		HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
Reset:		0	0	0	0	0	0	0	0

0x(FF)FF_9AC0	ENDPT0	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AC4	ENDPT1	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AC8	ENDPT2	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9ACC	ENDPT3	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AD0	ENDPT4	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AD4	ENDPT5	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AD8	ENDPT6	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9ADC	ENDPT7	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AE0	ENDPT8	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AE4	ENDPT9	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AE8	ENDPT10	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AEC	ENDPT11	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AF0	ENDPT12	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AF4	ENDPT13	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AF8	ENDPT14	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK
0x(FF)FF_9AFC	ENDPT15	HOST_WO_HUB	RETRY_DIS	0	EP_CTL_DIS	EP_RX_EN	EP_TX_EN	EP_STALL	EP_HSHK

Figura 18.37. Registros ENDPT0-15.

- **HOST\_WO\_HUB:** Este es un bit sólo para el modo HOST y aplica para el terminal 0 (ENDPT0). Cuando este bit se pone a “1” permite una conexión a baja velocidad. Al ser aclarado, el HOST emite un PRE\_PID y se conmuta a

baja velocidad, como requerimiento para comunicarse con un dispositivo en iguales condiciones utilizando un HUB.

- **RETRY\_DIS:** Este es un bit sólo para el modo HOST y aplica para el terminal 0 (ENDPT0). Cuando este bit es “1”, causa que el HOST no reintente transmitir una señal del tipo NAK. Cuando una transacción es del tipo NAK, el campo PID del BDT es actualizado con el NAK\_PID y una interrupción del tipo TOKEN\_DNE es generada. Cuando este bit es aclarado una transacción tipo NAK es reintentada. Este bit deberá ser llevado a “1” cuando el HOST está haciendo un proceso de *polling* sobre una interrupción de un terminal.
- **EP\_CTL\_DIS:** Cuando este bit es “1”, deshabilita el control de transferencias (SETUP). Cuando es borrado, el control de transferencias es habilitado. Lo anterior aplica únicamente cuando los bits EP\_RX\_EN y EP\_TX\_EN están en “1”.
- **EP\_RX\_EN:** Cuando este bit está en “1” habilita al terminal para transferencias del tipo RX.
- **EP\_TX\_EN:** Cuando este bit está en “1” habilita al terminal para transferencias del tipo TX.
- **EP\_STALL:** Cuando este bit es “1”, indica que el terminal es llamado. Este bit tiene prioridad sobre los demás bits de este registro, pero es únicamente válido si los bits EP\_RX\_EN y EP\_TX\_EN están en “1”. Cualquier acceso a este terminal causa que el módulo USB retorne de una condición de STALL. Después de que un terminal entre en la condición de STALL, este requiere una intervención desde el controlador del HOST.
- **EP\_HSHK:** Cuando este bit es “1”, habilita a un terminal para realizar el proceso de entendimiento durante la transacción. Este bit es generalmente “1”, a menos que el terminal esté en modo isocrónico.

La Tabla 18.4 resume el control de la dirección de las transacciones con un terminal.

**Tabla 18.4. Dirección de las transferencias y control sobre las terminales.**

EPL_CTL_DIS	EP_RX_EN	EP_TX_EN	Habilitación Terminal / Control de Dirección
X	0	0	Deshabilita la terminal
X	0	1	Habilita la terminal para transferencias tipo sólamente
X	1	0	Habilita la terminal para transferencias tipo RX sólamente
1	1	1	Habilita la terminal para transferencias tipo TX y RX
0	1	1	Habilita la terminal para transferencias de SETUP en TX y RX

- o **Registro de control USB (USB\_CTRL):** La Figura 18.38 muestra el registro de control USB.

### Registro de control USB (USB\_CTRL): (FFFF9B00)

	7	6	5	4	3	2	1	0
Lectura	SUSP	PDE	—	—	—	—	CLK_SRC	
Escritura	0	1	0	0	0	0	1	1
Reset:								

Figura 18.38. Registro USB\_CTRL.

- **SUSP:** Ubica el transceiver USB en estado de suspensión.  
 0: El módulo USB no está en estado de suspensión.  
 1: El módulo USB entra en estado de suspensión.
  - **PDE:** Habilita los *pulldown* débiles al transceiver USB  
 0: Deshabilita los *pulldown* débiles de los pines D+ y D-.  
 1: Habilita los *pulldown* débiles de los pines D+ y D-.
  - **CLK\_SRC:** Bits para determinar la fuente de reloj para el módulo USB.  
 00: Habilita el pin USB\_ALT\_CLK como fuente de reloj (PTG0).  
 01: Reservado.  
 10: Reservado.  
 11: Reloj del sistema (MCGPLLCLK).
- **Registro de observación del USB OTG (USB\_OTG\_OBSERVE):** La Figura 20.39 muestra el registro de observación del USB OTG.

### Registro de observación del USB OTG (USB\_OTG\_OBSERVE): (FFFF9B04)

	7	6	5	4	3	2	1	0
Lectura	DP_PU	DP_PD	0	DM_PD	—	—	—	1
Escritura	1	1	0	0	0	0	1	1
Reset:								

Figura 18.39. Registro USB\_OTG\_OBSERVE).

- **DP\_PU:** Suministra información de pin de salida D+ (en *pullup*) desde el módulo USB OTG. Este bit se usa cuando se tiene una interfase con un módulo en modo USB OTG.  
 0: El *pullup* del pin D+ está deshabilitado.  
 1: El *pullup* del pin D+ está habilitado.
- **DP\_PD:** Suministra información de pin de salida D+ (en *pulldown*) desde el módulo USB OTG. Este bit se usa cuando se tiene una interfase con un módulo en modo USB OTG.  
 0: El *pulldown* del pin D+ está deshabilitado.  
 1: El *pulldown* del pin D+ está habilitado.
- **DM\_PD:** Suministra información de pin de salida D- (en *pulldown*) desde el módulo USB OTG. Este bit se usa cuando se tiene una interfase con un módulo en modo USB OTG.  
 0: El *pulldown* del pin D- está deshabilitado.

- 1:** El *pulldown* del pin D- está habilitado.
- **Registro de control del USB OTG (USB\_OTG\_OBSERVE):** La Figura 18.40 muestra el registro de control del USB OTG.
- Registro de control del USB OTG (USB\_OTG\_CONTROL): (FFFF9B08)**
- |           |   |   |   |                 |    |         |         |        |
|-----------|---|---|---|-----------------|----|---------|---------|--------|
| Lectura   | 7 | 6 | 5 | 4               | 3  | 2       | 1       | 0      |
| Escritura | — | — | — | DPPULLUP_NONOTG | ID | VBUSVLD | SESSVLD | SESEND |
| Reset:    | 0 | 0 | 0 | 0               | 0  | 0       | 0       | 0      |
- Figura 18.40. Registro USB\_OTG\_OBSERVE).**
- **DPPULLUP\_NONOTG:** Suministra control sobre el *pullup* del pin D+ del USB OTG, cuando el USB no está configurado en modo OTG.  
**0:** Deshabilita el *pullup* del pin D+ en operación no OTG.  
**1:** Habilita el *pullup* del pin D+ en operación no OTG.
  - **ID:** Suministra control sobre la línea ID en el modo USB OTG, si el pin no ha sido configurado para esta función. Esta función es usada cuando se establece una interfase con un módulo USB OTG.  
**0:** Niega la señal ID.  
**1:** Confirma la señal ID.
  - **VBUSVLD:** Suministra control sobre la línea VBUS válida (*VBUS valid*) dentro del módulo USB OTG, si el pin no ha sido configurado para esta función. Esta función es usada cuando se establece una interfase con un módulo USB OTG.  
**0:** Niega la función válida VBUS.  
**1:** Confirma la función válida VBUS.
  - **SESSVLD:** Suministra observabilidad a la señal de sesión válida (*Session Valid*) dentro del módulo USB OTG. Esta función es usada cuando se establece una interfase con un módulo USB OTG.  
**0:** Niega la función de sesión válida.  
**1:** Confirma la función de sesión válida.
  - **SESEND:** Suministra observabilidad a la señal de sesión terminada (*Session End*) dentro del módulo USB OTG. Esta función es usada cuando se establece una interfase con un módulo USB OTG.  
**0:** Niega la función de sesión terminada.  
**1:** Confirma la función de sesión terminada.
- **Registro de control del transceiver y regulador del USB (USBTRC0):** La Figura 18.41 muestra el registro de control del transceiver y regulador del USB.

### Registro de control del transceiver y regulador USB (USBTRC0): (FFFF9B0C)

	7	6	5	4	3	2	1	0
Lectura	USBRESET	USBPU	USBRESME N	—	—	USBVREN	—	USB_RESU ME_INT
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 18.41. Registro USBTRC0.

- **USBRESET:** Este bit genera un RESET profundo al módulo USB. Este bit es automáticamente aclarado después que el sistema haya salido de ese estado.  
**0:** Operación normal del USB.  
**1:** Ubica al módulo USB en estado de RESET profundo.
  - **USBPU:** Este bit determina la fuente de la resistencia de *pullup* en la línea D+.  
**0:** Inhibe *pullup* interno a la línea D+. Se puede usar un *pullup* externo.  
**1:** Habilita *pullup* interno a la línea D+.
  - **USBRESMEN:** Cuando este bit se pone a “1”, permite que el módulo USB envíe un *wakeup* asíncrono hacia el microcontrolador, sobre la detección de una señal de resumen (*resume signalling*). El microcontrolador rehabilita el reloj hacia el módulo USB. Esto es usado para suspender el modo de bajo consumo, cuando el reloj del módulo ha sido parado.  
**0:** Deshabilita un *wakeup* asíncrono.  
**1:** Habilita un *wakeup* asíncrono.
  - **USBVREN:** Suministra observabilidad a la señal de sesión válida (*Session Valid*) dentro del módulo USB OTG. Esta función es usada cuando se establece una interfase con un módulo USB OTG.  
**0:** Niega la función de sesión válida.  
**1:** Confirma la función de sesión válida.
  - **USB\_RESUME\_INT:** Este bit indica si se ha generado una interrupción asíncrona en el módulo USB.  
**0:** No se ha generado una interrupción.  
**1:** Se ha generado una interrupción asíncrona en el módulo USB.
- **Registro de control del pin OTG (OTGPIN):** La Figura 18.42 muestra el registro de control del pin OTG, referido al DEMOJM en cuanto al *hardware*.

### Registro de control del pin OTG (OTGPIN): (FFFF9B10)

	7	6	5	4	3	2	1	0
Lectura	—	USPID	DMDOWN	DPPDOWN	PULLUP	VBUSVLD	SESSEND	SESSVLD
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 18.42. Registro OTGPIN.

- **USBID:** Este bit permite que el pin de entrada USB\_ID sea suministrado externamente.
    - 0:** El pin USB\_ID no está conectado externamente.
    - 1:** El pin USB\_ID está conectado al puerto PTC5.
  - **DMDOWN:** Este bit permite que el pin de entrada DM\_DOWN sea suministrado externamente, permitiendo al usuario conectar un *pulldown* externo a la señal D-. Este pin trabaja conjuntamente con el bit OTG\_CTRL[DM\_LOW].
    - 0:** El pin DM\_DOWN no está conectado externamente.
    - 1:** El pin DM\_DOWN está conectado al puerto PTA5.
  - **DPDOWN:** Este bit permite que el pin de entrada DP\_DOWN sea suministrado externamente, permitiendo al usuario conectar un *pulldown* externo a la señal D+. Este pin trabaja conjuntamente con el bit OTG\_CTRL[DP\_LOW].
    - 0:** El pin DP\_DOWN no está conectado externamente.
    - 1:** El pin DP\_DOWN está conectado al puerto PTA4.
  - **PULLUP:** El pin USB\_PULLUP es una salida del módulo USB, el cual trabaja conjuntamente con el bit USBTRC0[USBPU]. Si el bit USBTRC0[USBPU] es “0”, no se conecta un *pullup* interno en el pin D+. Poniendo a “1” el bit OTGCTRL[DP\_HI] pone los 3.3V del regulador interno al pin USB\_PULLUP. Poniendo a “0” el bit OTGCTRL[DP\_HI] pone pone en estado de alta impedancia al pin USB\_PULLUP.
    - 0:** El pin USB\_PULLUP está conectado externamente.
    - 1:** El pin DP\_DOWN está conectado al puerto PTA3.
  - **VBUSVLD:** El pin VBUS\_VLD es una entrada del módulo USB, el cual trabaja conjuntamente con el bit OTGSTAT[VBUS\_VLD].
    - 0:** El pin VBUS\_VLD no está conectado externamente.
    - 1:** El pin VBUS\_VLD está conectado al puerto PTA2.
  - **SESEND:** El pin SESS\_END es una entrada del módulo USB, el cual trabaja conjuntamente con el bit OTGSTAT[SESS\_END].
    - 0:** El pin SESS\_END no está conectado externamente.
    - 1:** El pin SESS\_END está conectado al puerto PTA1.
  - **SESSVLD:** El pin SESS\_VLD es una entrada del módulo USB, el cual trabaja conjuntamente con el bit OTGSTAT[SESS\_VLD].
    - 0:** El pin SESS\_VLD no está conectado externamente.
    - 1:** El pin SESS\_VLD está conectado al puerto PTA0.
- **Secuencia de pasos para una operación en modo HOST**

La siguiente secuencia de pasos ilustra la forma de colocar al USB en modo HOST, usando el núcleo del USB-FS. La secuencia hace referencia al *hardware* del DEMOJM.

    - **Habilitar modo HOST y descubrir un dispositivo conectado:**

- **Paso 1:** Habilitar modo HOST y parar la generación de paquetes SOF (*Start Of Frame*) con el fin de eliminar ruido en el bus. Las acciones para realizar lo anterior son:
  - (a) CTL[HOST\_MODE\_EN] = 1
  - (b) CTL[ESB\_EN] = 0
- **Paso 2:** Habilitar una posible interrupción por evento ATTACH y suministrar el VBUS al periférico. Las acciones para realizar lo anterior son:
  - (a) INT\_ENB[ATTACH] = 1
  - (b) PTBD[PTBD3] = 1
  - (c) Hacer un retardo de 100ms para estabilización de la alimentación del periférico.
- **Paso 3:** Esperar una interrupción por evento de ATTACH. Las acciones para realizar lo anterior son:

Hacer *polling* sobre la bandera INT\_STAT[ATTACH] o atender evento de interrupción verificando la misma bandera. El usuario debe recordar escribir un “1” en INT\_STAT[ATTACH], para aclarar el estado de la bandera.

- **Paso 4:** Verificar el estado de los bits CTL[JSTATE] y CTL[SE0]. Las acciones para realizar lo anterior son::

Si CTL[JSTATE] = 0 y CTL[SE0] = 1, entonces el dispositivo detectado requiere una conexión de baja velocidad y se debe poner ADDR[LS\_EN] = 1. Adicionalmente hacer el bit EP\_CTL0[HOST\_WO\_HUB] = 1.

- **Paso 5:** Generar señal de RESET por 10ms. Las acciones para realizar lo anterior son:

- (a) CTL[RESET] = 1
- (b) Hacer un retardo de 10ms
- (c) CTL[RESET] = 0

- **Paso 6:** Habilitar el envío de paquetes SOF. Las acciones para realizar lo anterior son:

CTL[ESB\_EN] = 1

- **Completar las transacciones de control:**

- **Paso 1:** Habilitar al periférico para realizar transacciones bidireccionales. Las acciones para realizar lo anterior son:  
EP\_CTL0[4 :0] = 0x0D

- **Paso 2:** Habilitar al periférico para realizar transacciones bidireccionales. Las acciones para realizar lo anterior son:  
EP\_CTL0[4 :0] = 0x0D

**NOTA:** En la página [www.freescale.com](http://www.freescale.com) y en el CD del DEMOJM se encuentran ejercicios para el módulo USB, que el lector podrá implementar fácilmente y apropiar para sus proyectos.

### **18.3. REFERENCIAS**

- MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual. Freescale Semiconductor. Rev 1, 2008.
- Microcontroller MCF51JM128 ColdFire Data Sheet. Freescale Semiconductor. Rev 0, 2008.
- Axelson, Jan. USB Complete. Segunda edición, USA. 2001.

# CAPÍTULO 19

## Otros Módulos del MCF51JM128

19.1. Diagrama en bloques, explicación del funcionamiento y registros asociados al módulo CMT

19.2. Diagrama en bloques, explicación del funcionamiento y registros asociados al módulo CAN

19.3. Referencias

## 19.1 DIAGRAMA EN BLOQUES, EXPLICACIÓN DEL FUNCIONAMIENTO Y REGISTROS ASOCIADOS AL MÓDULO CMT

- Breve descripción del módulo CMT

Este módulo esta compuesto de un generador de señal portadora, un circuito modulador y un circuito transmisor que adecúa la señal al dispositivo conectado a la salida del módulo. Las características más relevantes del módulo son:

- Cuatro modos de operación:
  - Temporizado, con independencia de los tiempos asignados para el “0” y el “1”.
  - Banda Base.
  - FSK (*Frequency Shift Key*).
  - Control directo del pin IRO (*Infra Red Output*).
- Tiempo del espacio (“0”) extensible.
- Generación de interrupción al final del ciclo.
- El temporizador puede ser utilizado como un generador de interrupciones periódicas, sin necesidad de afectar el pin IRO.

La Figura 19.1 ilustra el diagrama de bloques del módulo CMT, en donde el circuito (A) configura el reloj del sistema y de allí se produce el reloj base de para el circuito generador de portadora, que es de 125ns (derivado de un reloj de 8MHz). El circuito (B) contiene el generador de portadora y que puede manipular señales entre 4MHz y 7.84KHz y en pasos de 125ns, como se ilustra en la Tabla 19.1.

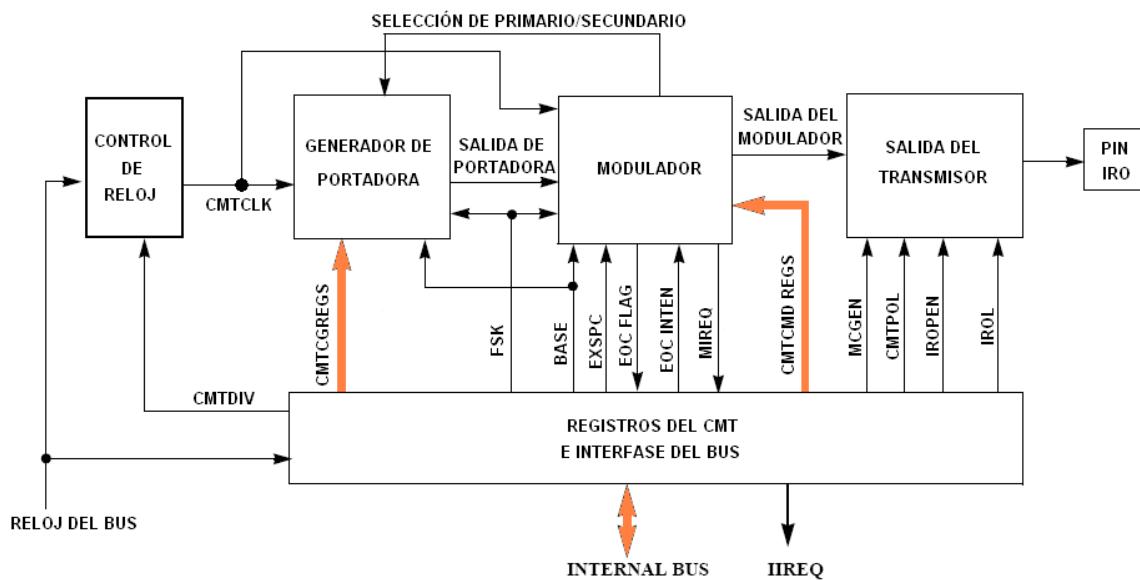


Figura 19.1. Diagrama en bloques del módulo CMT.

La resolución del generador de portadora es de 125ns, con reloj de operación de 8MHz. El generador puede manipular señales entre 4MHz y 7.84KHz y en pasos de 125ns, como se ilustra en la Tabla 19.1.

**Tabla 19.1. Frecuencias de la portadora y modulador del CMT.**

Reloj del BUS MHz	CMTDIV1:CMTDIV 0	Resolución del Generador de Portadora (us)	Período min del Generador de Portadora (us)	Período min del Modulador (us)
8	0:0	0.125	0.25	1.0
8	0:1	0.25	0.5	2.0
8	1:0	0.5	1.0	4.0
8	1:1	1.0	2.0	8.0

El ciclo de trabajo está relacionado con el número de conteos requerido para completar un período de la señal portadora. Por ejemplo: una señal de 2MHz tiene un período de 500ns, que requerirían 4 conteos de 125ns. Estos conteos pueden ser repartidos entre tiempos abajo y arriba: 1 arriba y 3 abajo, 2 arriba y 2 abajo, 3 arriba y 1 abajo.

## 19.2. DIAGRAMA EN BLOQUES, EXPLICACIÓN DEL FUNCIONAMIENTO Y REGISTROS ASOCIADOS AL MÓDULO CAN

Conservando la definición del controlador MSCAN de la familia M68HC12, este módulo está implementado en el protocolo CAN 2.0 A/B y que fue desarrollado y especificado por la compañía Bosch en Septiembre de 1991. Es altamente recomendable que el usuario lea la especificación, para un mayor entendimiento del módulo y que encontrará en el documento de la página:

<http://www.semiconductors.bosch.de/pdf/can2spec.pdf>.

El módulo CAN está enfocado como bus estándar para las comunicaciones en sistemas electrónicos en el área automotriz, presentando gran robustez en el tratamiento de señales EMI, gran ancho de banda y bajo costo. Las características más relevantes de este módulo son:

- CAN en protocolo 2.0 A/B.
  - Manejo de tramas estándar y ampliadas.
  - Longitud de datos desde 0 hasta 8 bytes.
  - Velocidad de hasta 1Mbps.
  - Soporte de tramas remotas.
- Cinco *buffers* FIFO para recepción con esquema de almacenamiento.
- Tres *buffers* de transmisión con esquema de priorización.
- Dos filtros de máscara de 32 bits (4 de 16 bits u 8 de 8 bits), de identificación flexibles.
- Función de *wakeup* con un filtro integrado pasa bajos.
- Soporta *loopback* para auto chequeo.
- Modo programable de sólo escucha, para monitoreo del bus CAN.
- Modo de recuperación de estado de off programable.
- Separación de interrupciones del transmisor, receptor y estados de error del bus.
- Fuente de reloj del CAN programable.
- Temporizador interno para *time stamping* y mensajes de transmisión.
- Tres modos de bajo consumo.
- Registros para inicialización global del periférico CAN.

El módulo can trabaja en tres modos:

- Modo de sólo escucha.
- Modo SLEEP.
- Modo de inicialización.
- Modo de POWER DOWN.

La Figura 19.2 muestra un diagrama en bloques de módulo CAN.

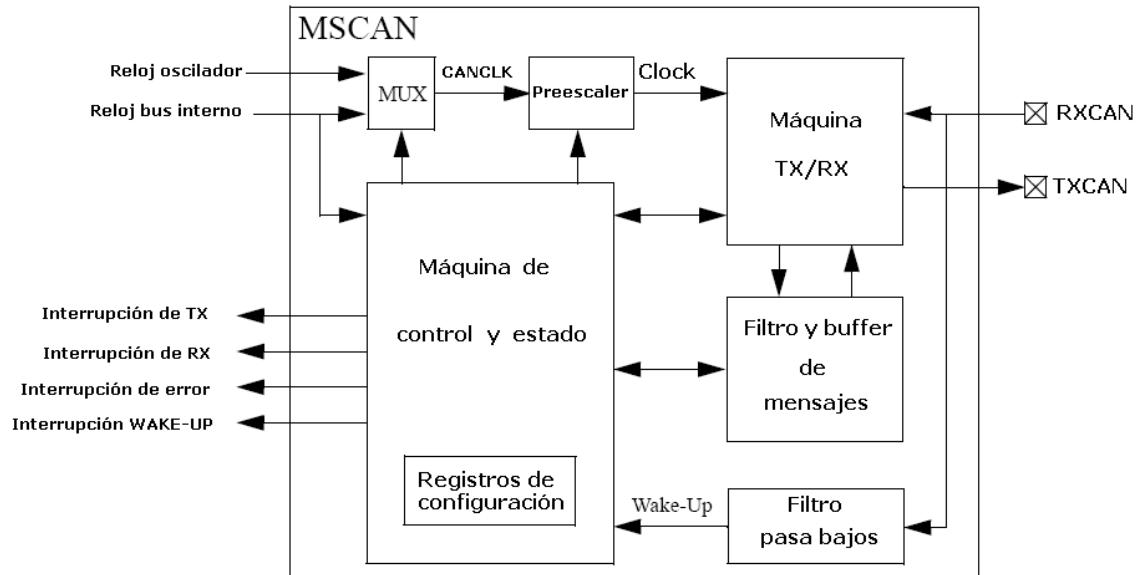


Figura 19.2. Diagrama en bloques módulo CAN.

La conexión típica de un bus CAN es como se ilustra en la figura 19.3.

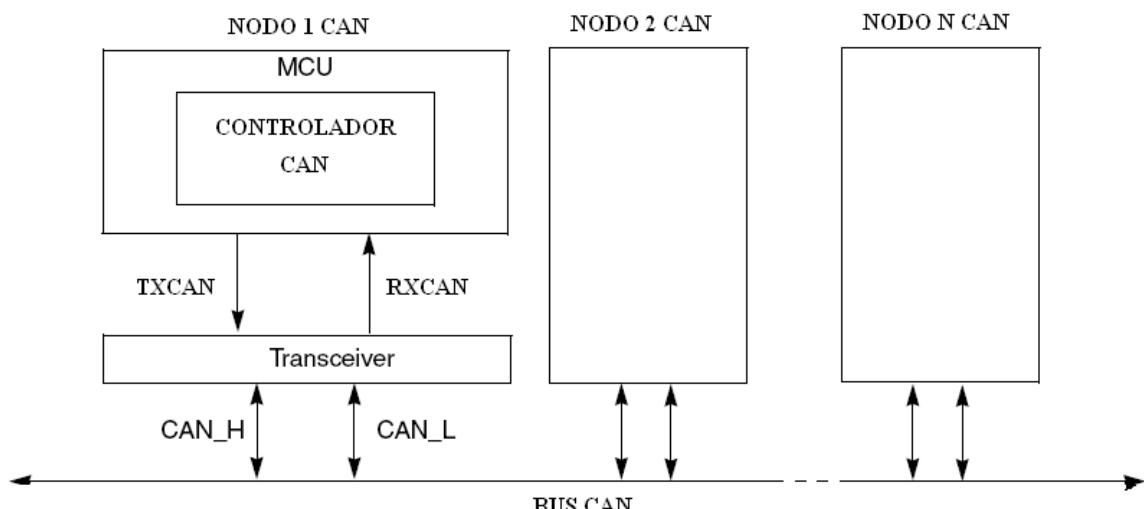


Figura 19.3. Conexión típica BUS CAN.

#### • Registros del módulo CAN

- **Registro de control 0 (CANCTL0):** La Figura 19.4 muestra el registro de control 0 del módulo CAN.

### Registro de control 0 (CANCTL0): (FFFF9880)

	7	6	5	4	3	2	1	0
Lectura	RXFRM	RXACT	CSWAI	SYNCH	TIME	WUPE	SLPRQ	INITRQ
Escritura								
Reset:	0	0	0	0	0	0	0	1

Figura 19.4. Registro de control 0 del CAN.

- **RXFRM:** Bandera de frame recibido. Este bit es de sólo lectura y borrado, que cuando se pone a “1” indica que se ha recibido un mensaje válido, independientemente de la configuración del filtro. Este bit permanece en “1” hasta tanto no sea borrado (escribiendo un “1”). Este bit no tiene sentido en modo *loopback*.
  - 0:** No se ha recibido un mensaje válido.
  - 1:** Se ha recibido un mensaje válido.
- **RXACT:** Estado activo de recepción. Indica que un mensaje está siendo recibido y es controlado por el frente final de recepción. Este bit no tiene sentido en modo *loopback*.
  - 0:** El módulo CAN está transmitiendo o está en modo IDLE.
  - 1:** El módulo CAN está recibiendo un mensaje.
- **CSWAI:** No puede parar en modo STOP. Permite entrar a modo de bajo consumo, deshabilitando todas las fuentes de reloj del módulo CAN.
  - 0:** El módulo no es afectado en modo WAIT.
  - 1:** El módulo CAN se detiene estando la MCU en modo WAIT.
- **SYNCH:** Estado de sincronismo. Indica cuando el MSCAN está sincronizado al bus CAN y disponible para participar en el proceso de comunicación.
  - 0:** El módulo no está sincronizado.
  - 1:** El módulo está sincronizado.
- **TIME:** Habilitador del temporizador CAN. Habilita un temporizador tipo *free running* de 16 bits. Una vez el temporizador está activo, el sistema activa un sello (*stamp*) para cada mensaje junto con el *buffer* de RX/TX. Tan pronto como el mensaje ha sido reconocido en el bus CAN, el sello es escrito en los bytes más altos (0x000E, 0x000F) en el *buffer* apropiado. El temporizador es inicializado cuando se desactiva el CAN.
  - 0:** Deshabilita el temporizador interno del CAN.
  - 1:** Habilita el temporizador interno del CAN.
- **WUPE:** Habilitador de WAKE-UP. Este bit permite al MSCAN despertar estando en modo SLEEP, cuando hay tráfico en el bus.
  - 0:** Deshabilita el modo WAKE-UP.
  - 1:** Habilita el modo WAKE-UP.
- **SLPRQ:** Requerimiento de modo SLEEP. Conduce al módulo CAN a entrar en modo de bajo consumo SLEEP. Este modo es servido cuando el módulo está en estado IDLE, por ejemplo cuando el módulo no está recibiendo un mensaje y los *buffer* de transmisión están vacíos. Poniendo el bit SLPAK = 1,

el módulo podrá entrar en SLEEP. El bit SLPRQ no puede ser puesto a “1” mientras el bit WUPIF esté en “1”. El modo SLEEP permanecerá activo hasta tanto la CPU no lo apague.

**0:** El MSCAN está corriendo normalmente.

**1:** Se hace una solicitud de modo SLEEP al CAN.

- **INITRQ:** Requerimiento de modo de inicialización. Cuando este bit es puesto en “1” el MSCAN escapa del módulo de inicialización. Cualquier proceso de transmisión o recepción es abortado y el sincronismo del bus CAN se pierde. El módulo indica entrada al modo de inicialización poniendo el bit INITAK en “1”.
  - 0:** El MSCAN en modo normal de operación.
  - 1:** El MSCAN está en modo de inicialización.
- **INITRQ:** Requerimiento de modo de inicialización. Cuando este bit es puesto en “1” el MSCAN escapa del módulo de inicialización. Cualquier proceso de transmisión o recepción es abortado y el sincronismo del bus CAN se pierde. El módulo indica entrada al modo de inicialización poniendo el bit INITAK en “1”.
  - 0:** El MSCAN en modo normal de operación.
  - 1:** El MSCAN está en modo de inicialización.

- **Registro de control 1 (CANCTL1):** La Figura 19.5 muestra el registro de control 1 del módulo CAN.

#### Registro de control 1 del CAN (CANCTL1): (FFFF9881)

	7	6	5	4	3	2	1	0
Lectura	CANE	CLKSRC	LOOPB	LISTEN	BORM	WUPM	SLPAK	INITAK
Escritura	0	0	0	1	0	0	0	1
Reset:	0	0	0	1	0	0	0	1

Figura 19.5. Registro de control 1 del CAN.

- **CANE:** Habilita el módulo MSCAN.  
**0:** Inhabilita el módulo MSCAN.  
**1:** Habilita el módulo MSCAN.
- **CLKSRC:** Fuente de reloj del MSCAN. Este bit define la fuente de reloj del MSCAN. Este bit no tiene sentido en modo *loopback*.  
**0:** El oscilador externo es la fuente de reloj del MSCAN.  
**1:** El reloj interno es la fuente de reloj del MSCAN.
- **LOOPB:** Autochequeo en modo *loopback*. Cuando este bit se pone a “1” el MSCAN realiza un *loopback* interno que puede ser usado como operación de autochequeo. El bit que se sirve en el transmisor es realimentado hacia el receptor internamente.  
**0:** Autochequeo por *loopback* deshabilitado.

- 1:** Autochequeo por *loopback* habilitado.
  - **LISSEN:** Modo de sólo escucha. Configura el MSCAN como un monitor de bus CAN. Una vez habilitado, todos los mensajes con el ID del MSCAN son recibidos, pero no se devuelven los *acknowledgement* ni los mensajes de error, en consecuencia el contador de errores es congelado. El modo LISTEN soporta aplicaciones de conexión en caliente (*hot plugging*) o análisis en línea (*throughput analysis*). El MSCAN no está habilitado para transmitir ningún mensaje cuando el modo LISTEN esté habilitado.
  - 0:** MSCAN en operación normal.
  - 1:** MSCAN en modo LISTEN.
  - **TIME:** Habilitador del temporizador CAN. Habilita un temporizador tipo *free running* de 16 bits. Una vez el temporizador está activo, el sistema activa un sello (*stamp*) para cada mensaje junto con el *buffer* de RX/TX. Tan pronto como el mensaje ha sido reconocido en el bus CAN, el sello es escrito en los bytes más altos (0x000E, 0x000F) en el *buffer* apropiado. El temporizador es inicializado cuando se desactiva el CAN.
  - 0:** Deshabilita el temporizador interno del CAN.
  - 1:** Habilita el temporizador interno del CAN.
  - **BORM:** Recuperación desde apagado del bus CAN. Este bit configura el modo de recuperación del estado de apagado del MSCAN, cuando hay tráfico en el bus.
  - 0:** Modo de recuperación automática.
  - 1:** Recuperación a voluntad del usuario.
  - **WUPM:** Modo de despertar del MSCAN. Si el bit WUPE es “1”, este bit define si se aplica un filtro pasa-bajos para proteger el MSCAN de despertar debido a señales espúreas.
  - 0:** El MSCAN despierta la MCU después de un flanco dominante en el bus CAN .
  - 1:** EL MSCAN despierta la MCU sólo en el caso de un pulso en el bus CAN con una longitud de Twup.
  - **SLPAK:** Reconocimiento del modo SLEEP. Este bit indica cuando el módulo MSCAN ha entrado en modo SLEEP. El modo SLEEP está activo cuando los bits SLPRQ = 1 and SLPAK = 1. Dependiendo de si WUPE es “1” el MSCAN borra esta bandera si se detecta actividad en el bus mientras se está en modo SLEEP. El MCU aclara los bits SLRPQC y SLPAK.
  - 0:** El MSCAN en modo normal de operación.
  - 1:** El MSCAN está en modo SLEEP.
  - **INITAK:** Reconocimiento del modo de inicialización. Este bit indica cuando el módulo MSCAN ha entrado en modo de inicialización. El modo SLEEP está activo cuando los bits INITRQ = 1 and INITAK = 1..
  - 0:** El MSCAN en modo normal de operación.
  - 1:** El MSCAN está en modo de inicialización.
- **Registro 0 de temporización de bus del MSCAN (CANBTR0):** La Figura 19.6 muestra el registro 0 de temporización de bus del MSCAN.

### Registro 0 de temporización de bus del MSCAN (CANBTR0): (FFFF9882)

	7	6	5	4	3	2	1	0
Lectura	SJW1	SJW0	BRP5	BRP4	BRP3	BRP2	BRP1	BRP0
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

Figura 19.6. Registro 0 de temporización de bus del MSCAN.

- **SJW0,SJW1:** Ancho del salto de sincronización. Define el valor máximo de tiempo en ciclos de reloj ( $T_q$ ), que un bit puede ser recortado o alargado para obtener re-sincronización entre transiciones de datos del bus MSCAN (ver Tabla 19.1).
  - 0:** Inhabilita el módulo MSCAN.
  - 1:** Habilita el módulo MSCAN.

Tabla 9.1. Ancho del salto de transición.

SJW1	SJW0	Ancho pulso sincronización
0	0	1 $T_q$ ciclos de reloj
0	1	2 $T_q$ ciclos de reloj
1	0	3 $T_q$ ciclos de reloj
1	1	4 $T_q$ ciclos de reloj

- **BRP0-BRP5:** Selección de rata de baudios del MSCAN. Bits utilizados para definir la rata de baudios ( $T_q$ ) (ver Tabla 19.2).

Tabla 9.2. Preescaler baudios MSCAN.

BRP5	BRP4	BRP3	BRP2	BRP1	BRP0	Valor del preescaler (P)
0	0	0	0	0	0	1
0	0	0	0	0	1	2
0	0	0	0	1	0	3
0	0	0	0	1	1	4
:	:	:	:	:	:	:
1	1	1	1	1	1	64

- o **Registro 1 de temporización de bus del MSCAN (CANBTR1):** La Figura 19.7 muestra el registro 1 de temporización de bus del MSCAN.

### Registro 1 de temporización de bus del MSCAN (CANBTR1): (FFFF9883)

	7	6	5	4	3	2	1	0
Lectura	SAMP	TSEG22	TSEG21	TSEG20	TSEG13	TSEG12	TSEG11	TSEG10
Escritura	Reset:	0	0	0	0	0	0	0

Figura 19.7. Registro 1 de temporización de bus del MSCAN.

- **SAMP:** Muestreo. Determina el número de muestras por bit del bus CAN.  
**0:** Una muestra por bit..  
**1:** Tres muestras por bit.
- **TSEG20-TSEG22:** Segmento de tiempo 2. Junto con el tiempo de bit fija el número de ciclos de reloj por tiempo de bit y la localización del punto de la muestra (ver Tabla 9.3).  
**0:** Inhabilita el módulo MSCAN.  
**1:** Habilita el módulo MSCAN.

Tabla 9.3. Segmento de tiempo 2.

TSEG22	TSEG21	TSEG20	Segmento de tiempo 2
0	0	0	1 Tq ciclo de reloj
0	0	1	2 Tq ciclo de reloj
:	:	:	:
1	1	0	7 Tq ciclo de reloj
1	1	1	8 Tq ciclo de reloj

- **TSEG10-TSEG13:** Segmento de tiempo 1. Junto con el tiempo de bit fija el número de ciclos de reloj por tiempo de bit y la localización del punto de la muestra (ver tabla 9.4).  
**0:** Inhabilita el módulo MSCAN.  
**1:** Habilita el módulo MSCAN.

Tabla 9.4. Segmento de tiempo 1.

TSEG13	TSEG12	TSEG11	TSEG10	Segmento de tiempo 1
0	0	0	0	1 Tq ciclo de reloj
0	0	0	1	2 Tq ciclo de reloj
0	0	1	0	3 Tq ciclo de reloj
0	0	1	1	4 Tq ciclo de reloj
:	:	:	:	:
1	1	1	0	15 Tq ciclo de reloj
1	1	1	1	16 Tq ciclo de reloj

Donde:

$$\text{Tiempo de bit} = ((\text{Valor del preescaler}) / \text{FCANCLK}) \times (1 + \text{Segmento tiempo 1} + \text{Segmento tiempo 2})$$

- **Registro de banderas del receptor del MSCAN (CANRFLG):** La Figura 19.8 muestra el registro de banderas del receptor del MSCAN.

#### **Registro de banderas del receptor MSCAN (CANRFLG): (FFFF9884)**

Lectura Escritura	7	6	5	4	3	2	1	0
	WUPIF	CSCIF	RSTAT1	RSTAT0	TSTAT1	TSTAT0	OVRIF	RXF
Reset:	0	0	0	0	0	0	0	0

**Figura 19.8. Registro de banderas del receptor MSCAN.**

- **WUPIF:** Bandera de interrupción por WAKE-UP. Se pone a “1” cuando se detecta actividad en el bus CAN cuando el MSCAN está en modo de SLEEP.  
**0:** No hay actividad en el bus durante modo de SLEEP.  
**1:** Hay actividad en el bus durante modo de SLEEP y se requiere WAKE-UP.
- **CSCIF:** Bandera de interrupción por cambio de estado del CAN. Esta bandera se pone a “1” cuando el MSCAN cambia el estado actual del bus CAN.  
**0:** No ha habido cambios en el estado del bus CAN desde la última interrupción.  
**1:** El reloj interno es la fuente de reloj del MSCAN.
- **RSTAT0,RSTAT1:** Estado de los bits del receptor. El valor del contador de errores controla el estado actual del bus CAN.  
**00:** Rx OK ( $0 \leq$  contador de errores del receptor  $\leq 96$ ).  
**01:** Rx Warning ( $96 <$  contador de errores del receptor  $\leq 127$ ).  
**10:** Rx error ( $127 <$  contador de errores del receptor  $\leq 255$ ).  
**11:** Bus CAN (contador de errores del receptor  $> 255$ ).
- **TSTAT0,TSTAT1:** Estado de los bits del transmisor. El valor del contador de errores controla el estado actual del bus CAN.  
**00:** Tx OK ( $0 \leq$  contador de errores del transmisor  $\leq 96$ ).  
**01:** Tx Warning ( $96 <$  contador de errores del transmisor  $\leq 127$ ).  
**10:** Tx error ( $127 <$  contador de errores del transmisor  $\leq 255$ ).  
**11:** Bus CAN (contador de errores del transmisor  $> 255$ ).
- **OVRIF:** Bandera de interrupción por overrun.  
**0:** No hay condición de overrun.  
**1:** Se ha detectado un overrun en los datos.
- **RXF:** Bandera de *buffer* de recepción lleno. Esta bandera se pone a “1” cuando un mensaje nuevo es almacenado en el *buffer* FIFO de recepción y ha llegado con la dirección apropiada, libre de errores y CRC validado. Una vez se ha leído el mensaje, la bandera deberá ser aclarada para liberar el *buffer*.  
**0:** No hay mensaje disponible en el *buffer* de Rx.

- 1:** El *buffer* FIFO de Rx no está vacío. Hay un nuevo mensaje disponible.
- **Registro de habilitación de interrupciones del receptor del MSCAN (CANRIER):** La Figura 19.9 muestra el registro de banderas del receptor del MSCAN.

**Registro de habilitación de interrupciones del receptor MSCAN (CANRIER): (FFFF9885)**

Lectura Escritura	7	6	5	4	3	2	1	0
	WUPIE	CSCIE	RSTATE1	RSTATE0	TSTATE1	TSTATE0	OVRIE	RXFIE
Reset:	0	0	0	0	0	0	0	0

**Figura 19.9. Registro de habilitación de interrupciones del receptor del MSCAN.**

- **WUPIE:** Bit para habilitar interrupción por WAKE-UP.  
**0:** No se genera interrupción por este evento.  
**1:** Un evento de WAKE-UP causa interrupción.
- **CSCIE:** Bit para habilitar interrupción por cambio de estado del bus CAN.  
**0:** No se genera interrupción por este evento.  
**1:** Un evento de WAKE-UP causa interrupción.
- **RSTATE0,RSTATE1:** Bit para habilitar interrupción por cambio de estado del receptor. Bits para controlar el nivel de sensibilidad en el cual el cambio de estado del receptor está causando interrupciones del tipo CSCIF. Independiente del nivel de sensibilidad seleccionado las banderas RSTAT continúan indicando el estado actual del receptor y serán actualizadas mientras no existan eventos de CSCIF pendientes.  
**00:** No se genera ningún evento CSCIF causado por cambios en el estado del receptor.  
**01:** Se generan eventos de CSCIF sólo por entrar o salir del estado de *bus-off*.  
**10:** Se generan eventos de CSCIF por entrar o salir de los estados de error o de *bus-off*.  
**11:** Se generan eventos de CSCIF para todos los cambios de estado.
- **TSTATE0,TSTATE1:** Bit para habilitar interrupción por cambio de estado del transmisor. Bits para controlar el nivel de sensibilidad en el cual el cambio de estado del transmisor está causando interrupciones del tipo CSCIF. Independiente del nivel de sensibilidad seleccionado las banderas TSTAT continúan indicando el estado actual del transmisor y serán actualizadas mientras no existan eventos de CSCIF pendientes.  
**00:** No se genera ningún evento CSCIF causado por cambios en el estado del transmisor.  
**01:** Se generan eventos de CSCIF sólo por entrar o salir del estado de *bus-off*.  
**10:** Se generan eventos de CSCIF por entrar o salir de los estados de error o de *bus-off*.  
**11:** Se generan eventos de CSCIF para todos los cambios de estado.
- **OVRIE:** Bit para habilitar interrupción evento de *overrun*.  
**0:** No se genera interrupción por este evento.

- **1:** Un evento de *overrun* causa interrupción.
- **RXFIE:** Bit para habilitar interrupción evento de *buffer* de receptor lleno.
- **0:** No se genera interrupción por este evento.
- **1:** Un evento de *buffer* de receptor lleno causa interrupción.
- **Registro de banderas del transmisor del MSCAN (CANTFLG):** La Figura 19.10 muestra el registro de banderas del transmisor del MSCAN.

#### Registro de banderas del transmisor MSCAN (CANTFLG): (FFFF9886)

	7	6	5	4	3	2	1	0
Lectura	0	0	0	0	0	TXE2	TXE1	TXE0
Escritura								
Reset:	0	0	0	0	0	1	1	1

Figura 19.10. Registro de banderas del transmisor del MSCAN.

- **TXE0-TXE2:** Indica que el *buffer* de transmisión asociado está vacío. La MCU deberá borrar la bandera una vez el mensaje ha sido puesto en el *buffer* de transmisión. El MSCAN pone la bandera a “1” una vez el mensaje ha sido transmitido con éxito. La bandera también se pone a “1” cuando se aborta un proceso de transmisión de mensaje. Aclarando la bandera TXEx, también se aclara la correspondiente ABTAKx y cuando es puesta a “1” la correspondiente ABTRQx es aclarada. Cuando el modo LISTEN está activo la bandera TXEx no podrá ser borrada y ninguna transmisión puede ser llevada a cabo. Los accesos de escritura y lectura del *buffer* de transmisión son bloqueados si el correspondiente bit TXEx es aclarado y el *buffer* será reprogramado para transmisión.
- **0:** El *buffer* del mensaje asociado está lleno (tiene un mensaje pendiente o en proceso de transmisión).
- **1:** El *buffer* de transmisión está vacío.
- **Registro de habilitación de interrupciones del transmisor del MSCAN (CANTIER):** La Figura 19.11 muestra el registro de banderas del transmisor del MSCAN.

#### Registro de habilitación de interrupciones del transmisor del MSCAN (CANTIER): (FFFF9887)

	7	6	5	4	3	2	1	0
Lectura	0	0	0	0	0	TXEIE2	TXEIE1	TXEIE0
Escritura								
Reset:	0	0	0	0	0	0	0	0

Figura 19.11. Registro de habilitación de interrupciones del transmisor del MSCAN.

- **TXEIE0-TXEIE2:** Bit para habilitar interrupción por transmisor vacío.  
 0: No se habilita interrupción por este evento.  
 1: Un evento de *buffer* vacío de transmisión causa interrupción.
- o **Registro de requerimiento de aborto de mensaje a transmitir del MSCAN (CANTARQ):** La Figura 19.12 muestra el registro para el requerimiento de aborto de mensaje en proceso de transmisión del MSCAN.

**Registro de requerimiento de aborto del mensaje en transmisión del MSCAN (CANTARQ): (FFFF9888)**

	7	6	5	4	3	2	1	0
Lectura	0	0	0	0	0	ABTRQ2	ABTRQ1	ABTRQ0
Escritura								
Reset:	0	0	0	0	0	0	0	0

**Figura 19.12. Registro de requerimiento de aborto del mensaje en transmisión del MSCAN.**

- **ABTRQ0-ABTRQ2:** Requerimiento de aborto. El MCU impide que un mensaje programado en el *buffer* de transmisión sea abortado (TXEx = 0).  
 0: No se hay requerimiento de aborto de mensaje.  
 1: Requerimiento de aborto pendiente.
- o **Registro de reconocimiento de requerimiento de aborto de mensaje a transmitir del MSCAN (CANTAAK):** La Figura 19.13 muestra el registro de reconocimiento del requerimiento de aborto de mensaje en proceso de transmisión del MSCAN.

**Registro de reconocimiento de aborto de mensaje a transmitir del MSCAN (CANTAAK): (FFFF9889)**

	7	6	5	4	3	2	1	0
Lectura	0	0	0	0	0	ABTAK2	ABTAK1	ABTAK0
Escritura								
Reset:	0	0	0	0	0	0	0	0

**Figura 19.13. Registro de reconocimiento del requerimiento de aborto del mensaje en transmisión del MSCAN.**

- **ABTAK0-ABTAK2:** Reconocimiento de aborto. Indica que un mensaje de transmisión fue abortado con éxito.  
 0: El mensaje no ha sido abortado.  
 1: El mensaje fue abortado.

- **Registro de selección de *buffer* de transmisión del MSCAN (CANTBSEL):** La Figura 19.14 muestra el registro de selección del *buffer* actual de transmisión del MSCAN.

**Registro de selección del buffer de transmisión del MSCAN (CANTBSEL): (FFFF988A)**

	7	6	5	4	3	2	1	0
Lectura	0	0	0	0	0	TX2	TX1	TX0
Escritura						0	0	0
Reset:	0	0	0	0	0	0	0	0

**Figura 19.14. Registro de selección del *buffer* de transmisión del MSCAN.**

- **TX0-TX2:** Ubica el respectivo *buffer* de transmisión, de acuerdo al bit de más bajo peso en “1”, en el registro de espacio CANTXFG. Por ejemplo si TX1 = 1 y TX0 = 1 se elige el *buffer* de transmisión TX0, si TX1 = 1 y TX0 = 0 se elige el *buffer* de transmisión TX1.
- 0: El *buffer* de mensaje asociado no se ha seleccionado.
- 1: El *buffer* de mensaje asociado se ha seleccionado (de acuerdo al bit de más bajo peso en “1”).
- **Registro de control de aceptación del identificador del MSCAN (CANIDAC):** La Figura 19.15 muestra el registro que trabaja como filtro de aceptación de identificador del MSCAN.

**Registro de control de aceptación de identificación del MSCAN (CANIDAC): (FFFF988B)**

	7	6	5	4	3	2	1	0
Lectura	0	0	IDAM1	IDAM0	0	IDHIT2	IDHIT1	IDHITO
Escritura					0	0	0	0
Reset:	0	0	0	0	0	0	0	0

**Figura 19.15. Registro de control de aceptación del identificador del MSCAN.**

- **IDAM0, IDAM1:** Modo de aceptación de identificación. El MCU pone tres banderas para definir el filtro de aceptación del identificador, de acuerdo a la Tabla 19.5.

**Tabla 19.5. Modos de aceptación del identificador.**

IDAM1	IDAM0	Modo de aceptación del identificador
0	0	Dos filtros de 32 bits de aceptación
0	1	Cuatro filtros de 16 bits de aceptación
1	0	Ocho filtros de 8 bits de aceptación
1	1	Filtro cerrado

- **IDHIT0-IDHIT2:** Indicador de aceptación de identificador logrado. Según la Tabla 19.6, estos bit indican cual filtro se a logrado.

**Tabla 19.6. Tabla de logro de filtro.**

IDHIT2	IDHIT1	IDHIT0	Filtro de aceptación logrado
0	0	0	Logro en el filtro 0
0	0	1	Logro en el filtro 1
0	1	0	Logro en el filtro 2
0	1	1	Logro en el filtro 3
1	0	0	Logro en el filtro 4
1	0	1	Logro en el filtro 5
1	1	0	Logro en el filtro 6
1	1	1	Logro en el filtro 7

- o **Registro misceláneo del MSCAN (CANMISC):** La Figura 19.16 muestra el registro misceláneo del MSCAN.

**Registro misceláneo del MSCAN (CANMISC): (FFFF988D)**

Lectura	7	6	5	4	3	2	1	0
Escritura	0	0	0	0	0	0	0	BOHOLD
Reset:	0	0	0	0	0	0	0	0

**Figura 19.16. Registro misceláneo del MSCAN.**

- **BOHOLD:** Retiene el estado de *bus off* hasta que el usuario lo requiera. Indica que el bus ha entrado al estado de *bus off*. Aclarando este bit se hace un requerimiento de abandono del estado de *bus off*.
- 0:** El módulo no está en estado de *bus off* o se ha hecho una solicitud de recuperación del estado *bus off* por el usuario.
- 1:** El módulo está en estado de *bus off* y de allí no sale hasta tanto el usuario no lo pida.
- o **Registro contador de errores del receptor del MSCAN (CANRXERR):** La Figura 19.17 muestra el registro contador de errores del receptor del MSCAN. Este registro sólo podrá ser leído en modo SLEEP o en modo de inicialización.

**Registro contador de errores del receptor del MSCAN (CANRXERR): (FFFF988E)**

Lectura	RXERR7	RXERR6	RXERR5	RXERR4	RXERR3	RXERR2	RXERR1	RXERRO
Escritura	0	0	0	0	0	0	0	0
Reset:	0	0	0	0	0	0	0	0

**Figura 19.17. Registro contador de errores del receptor del MSCAN.**

**RXERR0-RXERR7:** Bits de estado del contador de errores del receptor del MSCAN.

- **Registro contador de errores del transmisor del MSCAN (CANTXERR):** La Figura 19.18 muestra el registro contador de errores del transmisor del MSCAN. Este registro sólo podrá ser leído en modo SLEEP o en modo de inicialización.

**Registro contador de errores de transmisor del MSCAN (CANTXERR):  
(FFFF988F)**

Lectura	TXERR7	TXERR6	TXERR5	TXERR4	TXERR3	TXERR2	TXERR1	TXERR0
Escritura								
Reset:	0	0	0	0	0	0	0	0

**Figura 19.18. Registro contador de errores del transmisor del MSCAN.**

**TXERR0-TXERR7:** Bits de estado del contador de errores del transmisor del MSCAN.

- **Registro identificador de aceptación del MSCAN (CANIDAR0-7):** En estado de recepción cada mensaje es escrito en el *buffer background* de recepción. El MCU sólo es señalado para leer el mensaje si este aprueba el criterio de aceptación de identificación y los registros de identificación de máscara, de otra manera el mensaje será sobrescrito por el siguiente mensaje.

Los registros de aceptación del MSCAN son aplicados sobre los registros IDR0-IDR3.

**Registros de aceptación de identificador del MSCAN (CANIDAR0 - CANIDAR7):  
(FFFF9890 - FFFF9893) (FFFF9898 - FFFF989B)**

	7	6	5	4	3	2	1	0
Lectura	AC7	AC6	AC5	AC4	AC3	AC2	AC1	AC0
Escritura	0	0	0	0	0	0	0	0
Reset	0	0	0	0	0	0	0	0

**Figura 19.19. Registro contador de errores del transmisor del MSCAN.**

- **AC0-AC7:** Bits de aceptación de identificador del MSCAN. Comprenden una secuencia de bit definida por el usuario contra las cuales los correspondientes bits del registro de identificación relacionado (IDRn) del buffer del mensaje recibido es comparado. El resultado de esta comparación es enmascarado con el registro de máscara de identificador.
- **Registros máscara del identificador MSCAN (CANIDMR0 - 7):** Especifica cuales de los correspondientes bits en el registro de aceptación de identificador son relevantes como filtro de aceptación. Para recibir identificadores estándar en el modo de 32 bits se requiere el uso de los bits AM0 – AM2, ubicados en el

registro de máscara CANIDMR1. Para recibir identificadores estándar en el modo de 16 bits se requiere el uso de los bits AM0 – AM2, ubicados en los registros de máscara CANIDMR1, CANIDMR2 y CANIDMR5.

**Registros máscara del identificador del MSCAN (CANIDMR0 - 7): (FFFF9894 - FFFF9897) (FFFF989C - FFFF989F)**

Lectura Escritura	7 AM7	6 AM6	5 AM5	4 AM4	3 AM3	2 AM2	1 AM1	0 AM0
Reset	0	0	0	0	0	0	0	0

**Figura 19.20. Registros máscara del identificador del MSCAN.**

- **AM0-AM7:** Bits máscara de aceptación. Si un bit en particular de este registro es aclarado, esto indicará que el bit correspondiente en el registro de aceptación de identificador deberá ser el mismo, como el bit de identificación después de detectarse una coincidencia. El mensaje será aceptado si todos los bits involucrados coinciden. Si un bit en el registro de máscara es puesto a “1”, esto indica que el bit en la posición no se tendrá en cuenta para la aceptación.  
**0:** Acierto en el correspondiente código de aceptación y bits de identificación.  
**1:** Ignore el correspondiente bit en el código de aceptación.

- **Modelo de programación para el almacenamiento de mensajes**  
 Un buffer de mensajes está formado según la Tabla 19.7.

**Tabla 19.7. Buffer de mensaje.**

Offset de Dirección	Registro
0x00X0	Registro de identificación 0
0x00X1	Registro de identificación 1
0x00X2	Registro de identificación 2
0x00X3	Registro de identificación 3
0x00X4	Registro de segmento de datos 0
0x00X5	Registro de segmento de datos 1
0x00X6	Registro de segmento de datos 2
0x00X7	Registro de segmento de datos 3
0x00X8	Registro de segmento de datos 4
0x00X9	Registro de segmento de datos 5
0x00XA	Registro de segmento de datos 6
0x00XB	Registro de segmento de datos 7
0x00XC	Registro de longitud de datos
0x00XD	Registro de buffer de prioridad de transmisión
0x00XE	Registro de marca de tiempo (alto)
0x00XF	Registro de marca de tiempo (bajo)

- **Registros de identificación del MSCAN (IDR0 – IDR3):** Especifica cuales de los correspondientes bits en el registro de aceptación de identificador son relevantes como filtro de aceptación. Para recibir identificadores estándar en el modo de 32 bits se requiere el uso de los bits AM0 – AM2, ubicados en el registro de máscara CANIDMR1. Para recibir identificadores estándar en el modo de 16 bits se requiere el uso de los bits AM0 – AM2, ubicados en los registros de máscara CANIDMR1, CANIDMR2 y CANIDMR5.

#### **Registros de identificación del MSCAN (IDR0 - IDR3): (FFFF98A0 - FFFF98A3)**

	Bit 7	6	5	4	3	2	1	Bit0	
IDR0	R W	ID28	ID27	ID26	ID25	ID24	ID23	ID22	ID21
IDR1	R W	ID20	ID19	ID18	SRR <sup>(1)</sup>	IDE <sup>(1)</sup>	ID17	ID16	ID15
IDR2	R	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7
IDR3	R W	ID6	ID5	ID4	ID3	ID2	ID1	ID0	RTR <sup>2</sup>

**Figura 19.21. Registros de identificación del MSCAN.**

- **IDx:** Bits de identificación del MSCAN. El bit más significativo es transmitido primero en el bus CAN durante el proceso de arbitramiento. La prioridad de un identificador está definida del más alto al más bajo número binario.
- **SRR:** Requerimiento de sustitución remota. Usado en formato de dirección extendida y es “1” en el *buffer* de transmisión.
- **IDE:** Dirección extendida. Define cuando el identificador es el estándar o extendido y es aplicado al buffer en cuestión.  
**0:** Formato estándar de identificación (11 bits).  
**1:** Formato extendido de identificación (29 bits).
- **RTR:** Requerimiento de transmisión remota. Refleja el estado de requerimiento de transmisión remota..  
**0:** Trama d datos.  
**1:** Trama remota.

- **Registros de segmento de datos del MSCAN (DSR0 – DSR7):** Contienen los datos a ser transmitidos o recibidos. El número de datos a manipular se especifica en el registro DLR correspondiente.

### Registros de segmento de datos (DSR0 - DSR7): (FFFF98A4 - FFFF98AB)

DSR0	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR1	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR2	R	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR3	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR4	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR5	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR6	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
DSR7	R W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0

Figura 19.22. Registros segmento de datos del MSCAN.

- **Registro de longitud de datos del MSCAN (DLR):** La Figura 19.23 muestra el registro de longitud de datos de una transacción CAN.

### Registro de longitud de datos de una trama CAN (DSR): (FFFF98AC)

	7	6	5	4	3	2	1	0
Lectura					DLC3	DLC2	DLC1	DLC0
Escritura	x	x	x	x	x	x	x	x

Figura 19.23. Registro de longitud de datos de una trama CAN.

- **DLC0 –DLC3:** Contiene el numero de bytes del mensaje actual.
- **Registro de prioridad de *buffer* de transmisión (CANTBPR):** La Figura 19.24 muestra el registro de prioridad de *buffer* de transmisión. Usado para el proceso interno de priorización y está definida del más alto al más bajo nivel de prioridad de acuerdo al número binario. Todos los *buffers* de transmisión con la bandera TXEx = 0 participan en el proceso de priorización después de un inicio de trama. El *buffer* de transmisión con la más baja prioridad local gana la priorización. En caso de que más de un buffer tengan el mismo nivel bajo de prioridad, el mensaje con el más bajo índice gana.

### Registro de prioridad de buffer de transmisión (CANTBPR): (FFFF98BD)

	7	6	5	4	3	2	1	0
Lectura	PRI07	PRI06	PRI05	PRI04	PRI03	PRI02	PRI01	PRI00
Escritura	0	0	0	0	0	0	0	0
Reset:	x	x	x	x	x	x	x	x

Figura 19.24. Registro de prioridad de buffer de transmisión.

- **Registro de marca de tiempo (CANTSR):** La Figura 19.25 muestra el registro de marca de tiempo. Si el bit TIME está habilitado el MSCAN escribe una marca de tiempo en los registros respectivos de los *buffer* de transmisión y recepción, tan pronto como el mensaje ha sido reconocido en el bus CAN.

La marca de tiempo es escrita sobre el punto de bit de muestra para el bit recessivo del bit delimitador de *acknoledge* en la trama de datos CAN. En el caso de una transmisión el MCU puede leer únicamente la marca de tiempo después de que el respectivo buffer de transmisión ha sido aclarado en sus banderas.

### Registro de marca de tiempo del MSCAN (CANTSR):(FFFF98AE,FFFF98AF)

	7	6	5	4	3	2	1	0	
TSRL	R	TSR15	TSR14	TSR13	TSR12	TSR11	TSR10	TSR9	TSR8
	W								
TSRH	R	TSR7	TSR6	TSR5	TSR4	TSR3	TSR2	TSR1	TSR0
	W								
Reset:	x	x	x	x	x	x	x	x	

Figura 19.25. Registro de marca de tiempo.

**NOTA:** Se recomienda al usuario hacer una lectura juiciosa sobre el funcionamiento del módulo CAN y consultar en la página [www.freescale.com](http://www.freescale.com) sobre algunas notas de aplicación con éste módulo.

### **19.3. REFERENCIAS**

- MCF51JM128 ColdFire® Integrated Microcontroller Reference Manual. Freescale Semiconductor. Rev 1, 2008.
- Microcontroller MCF51JM128 ColdFire Data Sheet. Freescale Semiconductor. Rev 0, 2008.