UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

“APLICACIÓN DE LA TECNOLOGÍA FLEXIS DE FREESCALE EN LA IMPLEMENTACIÓN DE PRÁCTICAS PARA DISEÑO MECATRÓNICO”

T E S I S

QUE PARA OBTENER EL TíTULO DE:
INGENIERO MECÁNICO ELECTRICISTA
(ÁREA ELÉCTRICA-ELECTRÓNICA)

PRESENTA:
OSCAR GUADALUPE MORENO ESPINOZA

ASESOR:
M. EN C. ARTURO OCAMPO ÁLVAREZ

MÉXICO 2013

Agradecimientos a la DGAPA al permitir colaborar en el proyecto PAPIME103011

Agradecimientos

Primeramente a las Sagradas y Benditas Divinidades y a los Ángeles de Luz, que me han guiado en mi camino.

A mi mamá ***Susana Espinosa García****, por todo lo que has hecho por mí, te quiero mucho.*

También a mi hermano ***Cesar, mi tía Alicia, mis primos Alicia, David, José e Inés,*** *por todo lo que hemos pasado.*

En especial a ***Iris María Morales Ortega****, tu valioso trabajo quedará conjunto a este. Gracias por tus palabras, las veces que me has escuchado y la de tu compañía es algo que me gusta mucho, aunque a veces me dejas sin ideas... Iris eres alguien a quien quiero mucho y al igual muy especial para mí.*

Mis amigos ***María del Carmen Velasco Cárdenas, Esteban Jiménez Rodríguez, Roció Flores Cruz y Raúl Iván Campos Delgadillo*** *por su gran apoyo que en las buenas y en las malas he contado con ustedes.*

Mi Maestro y asesor ***M. en C. Arturo Ocampo Álvarez,*** *gracias por todo el apoyo, paciencia mostrados en el tiempo que hemos compartido y sus conocimientos que me ha transmitido en el desarrollo de mi formación profesional, además de ser una persona valiosísima de calidad humana.*

Mtro. Juan Gastaldi Pérez, Dr. Alejandro Antonio Vega Ramírez y Dr. Ismael Díaz Rangel *por compartir sus conocimientos y sus valiosas observaciones para que este trabajo fuera realizado.*

No solo no hubiéramos sido nada sin ustedes, sino con toda la gente que estuvo a nuestro alrededor desde el comienzo, algunos siguen hasta hoy.

¡¡GRACIAS…TOTALES!!

Gustavo Cerati, músico, cantautor y compositor Argentino.

I N D I C E

INTRODUCCIÓN

Este trabajo de tesis se enfoca en la realización de un manual de prácticas con la tarjeta DEMOJM de Freescale, el alumno será capaz de diseñar y programar prácticas con la tarjeta antes mencionada. Además, será una guía útil en el laboratorio de Diseño Mecatrónico, correspondiente a la materia del mismo nombre de la carrera de Ingeniería Mecánica con plan de estudios 2007, que es impartida en el laboratorio L-3 de la Facultad de Estudios Superiores Aragón. En este manual se trata de explicar la programación básica en el lenguaje “C” de los microcontroladores de la familia Flexis de Freescale, así como la información necesaria para activar los módulos entrada y salida de la tarjeta DEMOJM, realizar las conexiones de los circuitos externos, y la comunicación de este dispositivo a hacia la computadora personal, para el desempeño de las prácticas a realizar.

Las prácticas de este trabajo se basan en la programación de lenguaje C, el cual es ampliamente estandarizado desde el momento de su creación por los ingenieros Ritchie y Kernighan. Su popularidad ha superado las expectativas y más ahora que entran en el mercado de los procesadores embebidos su crecimiento es aún mayor. Este hecho es tal vez el más importante de resaltar, ya que garantiza el encontrar con facilidad programadores en el ámbito laboral que puedan realizar una aplicación y no estar limitados a los expertos que solo conocen determinado lenguaje ensamblador y/o arquitectura.

En el primer capítulo se describe el marco histórico de los microcontroladores y sus respectivas clasificaciones, así como las ventajas que ofrece la tecnología Flexis de Freescale con sus correspondientes campos de aplicación.

El segundo capítulo se tratan los objetivos y justificación para la elaboración de prácticas de diseño mecatrónico. Se comenta también la metodología para desarrollo de productos mecatrónicos y los componentes que intervienen en el diseño, como los son sensores, actuadores e interfaz hombre-máquina. Al igual se describe en forma general lo que es un microcontrolador y se hace una comparación entre un microcontrolador y microprocesador.

Por último, en el tercer capítulo se desarrolla las prácticas que conforman este trabajo de tesis. Los temas fueron elegidos debido a la importancia que tienen para lograr la comprensión de la teoría básica de estos dispositivos además que desde la primera práctica el alumno trabaja con la tarjeta DEMOJM y programa el microcontrolador, lo que permite el poder familiarizarse cuando se trabaja con este tipo de dispositivos (diseñar el código fuente, depurar, compilar y grabar, etc.).

Además, permite la motivación e interés por parte del alumno ya que puede comprobar de manera teórica-práctica los conceptos de los microcontroladores.

El temario de prácticas propuesto es el siguiente:

- *Práctica Número 1: "Conceptos del Microcontrolador MC9S08JM60 (Freescale) y su entorno de programación CodeWarrior".*
- *Práctica Número 2: "Configuración de los Puertos de E/S de la DEMOJM"*
- *Práctica Número 3: "Manejo de la interrupción externa IRQ".*
- *Práctica Número 4: "Control de Velocidad de un Motor de DC".*
- *Práctica Número 5: "Control de un Servomotor de DC".*
- *Práctica Número 6: "Convertidor Analógico Digital".*
- *Práctica Número 7: "Comunicación Serie".*

JUSTIFICACIÓN

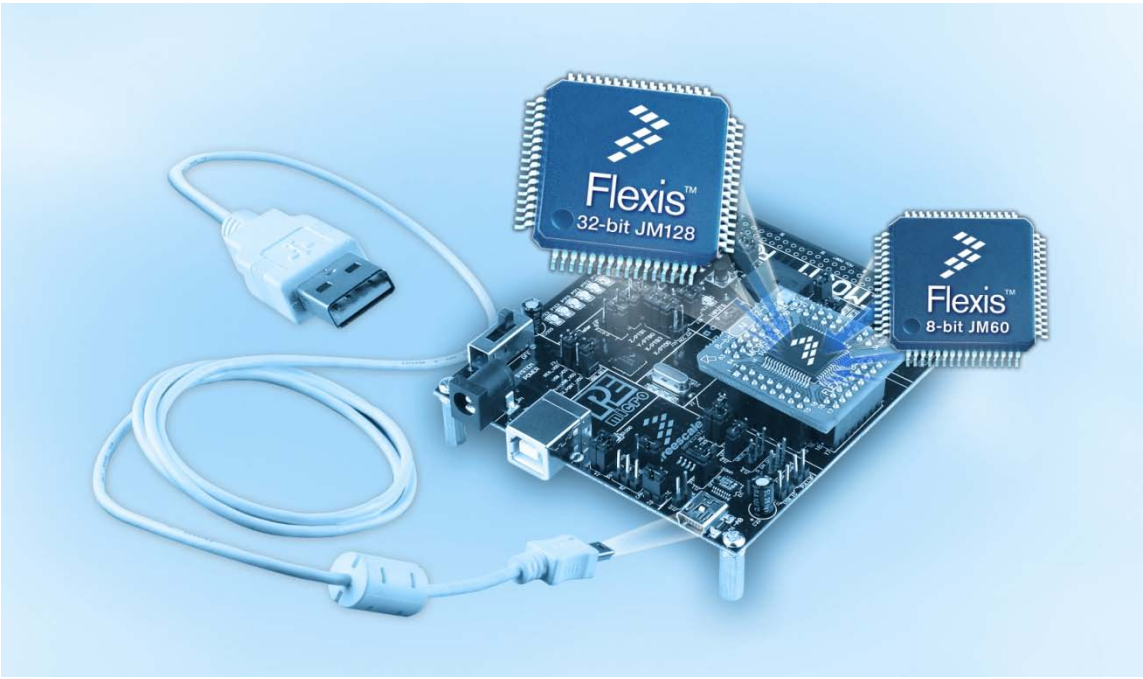
Este trabajo se desarrolla por la necesidad de no contar con un manual de prácticas para el laboratorio de Diseño Mecatrónico, de la carrera de Ingeniería Mecánica de la Facultad de Estudios Superiores Aragón, como apoyo didáctico utilizando microcontroladores de la familia Flexis de Freescale. Con el uso de la tarjeta DEMOJM se espera que los estudiantes logren adentrar más a la programación de sistemas embebidos sin mayor problema y así poder realizar sus prácticas, tanto escolares como profesionales.

Se busca que este trabajo de tesis sea incorporado como material de apoyo en el laboratorio de Diseño Mecatrónico, que se imparte actualmente en el plan de estudios de la carrera de Ingeniería Mecánica, así como proporcionar información sobre la aplicación y programación de sistemas embebidos.

OBJETIVOS

- Obtener un manual de prácticas como material de apoyo en el laboratorio de Diseño Mecatrónico.
- Mostrar las características generales de la familia Flexis de Freescale.
- Proporcionar una herramienta teórica y práctica de control con la tarjeta de desarrollo DEMOJM de Freescale.
- Aplicación de cualquier microcontrolador en la solución de problemas prácticos.
- Mejor comprensión de la programación de microcontroladores, por parte de los estudiantes.

CAPÍTULO 1: VENTAJAS DE LA TECNOLOGÍA FLEXIS



1.1 Marco y evolución histórica

Pocas personas son conscientes de lo importante que se han vuelto los microcontroladores en su vida diaria. En nuestro hogar, hay decenas de sistemas embebidos “escondidos” en casi todos los electrodomésticos de la cocina, en sistemas de alarmas, en televisión, en los reproductores de audio y video, cámara digitales, en celulares, en video juegos y en un sinfín de aparatos más. En el automóvil puede llegar a tener, hoy en día, hasta un centenar de microprocesadores especializados para controlar y monitorizar el motor, los frenos, la estabilidad y el cinturón de seguridad por citar algunos.

En la calle, los encontramos en semáforos, cajeros automáticos de los bancos, expendedores de boletos o de refrescos, etc. En el trabajo rodeados de teléfonos, impresoras, fax, fotocopiadoras, sistemas de acceso y de vigilancia, lectores de código y muchos sistemas que también incluyen procesamiento digital.

Finalmente, la industria hace muchos años que se “rindió” a la potencia de los microcontroladores que le permiten automatizar procesos y mejorar el rendimiento. Ya nada escapa a esta tecnología que es capaz de codificar en una secuencia de datos binarios para poder almacenarlos, duplicarlos, procesarlos, transformarlos e incluso transportarlos a gusto.

Esta revolución empezó hace cerca de cuarenta años, cuando los fabricantes de semiconductores consiguieron integrar en un chip todos los elementos de una unidad central de proceso. Así nació el microprocesador y desde entonces la evolución ha sido realmente espectacular.

La figura 1.1 muestra el crecimiento exponencial del número de transistores por chip con el curso de los años, que ha pasado de miles a cientos de millones.

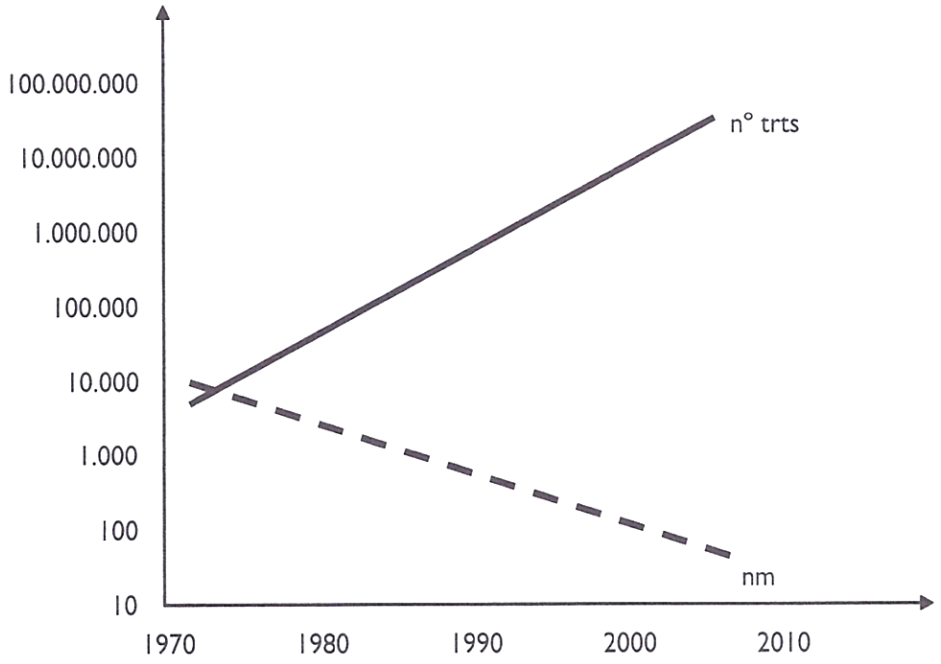


Figura 1.1 Evolución del número de transistores/chip y de la tecnología de proceso.

La clave ha estado en la evolución de la microelectrónica y en la filosofía de diseño por niveles de abstracción, que han permitido desarrollar circuitos integrados cada vez más veloces y complejos, y han dado lugar a diferentes implementaciones en función de los criterios de diseño.

Cuando se habla de sistemas digitales programables mucha gente piensa sólo en las computadoras personales. Sin embargo, éstos la punta del iceberg, del orden del 1%. El resto son sistemas de propósito específico que cuentan con elevadas capacidades de adquisición y control de datos, proceso y conectividad.

1.2 Clasificación

Una clasificación general de sistemas digitales que puede dar una idea de la variedad de implementaciones existentes es la que se muestra en la figura 1.2:

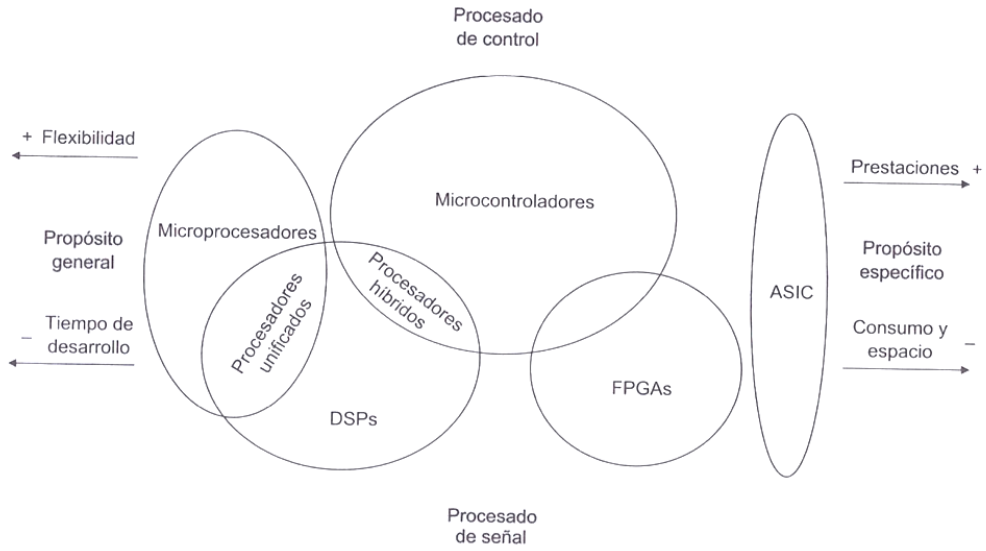


Figura 1.2 Clasificación de los sistemas digitales.

Se distingue entre dispositivos diseñados para sistemas de propósito general y de propósito específico. Los primeros buscan flexibilidad, escalabilidad, altas prestaciones y tiempos cortos de desarrollo del hardware. Los segundos se especializan en aplicaciones concretas y, a diferencia de los anteriores, suelen optimizar espacio y consumo. También se distingue entre proceso de control, donde el objetivo básico es arbitrar muchos eventos en tiempo real, y procesamiento de señales, que se centra en la computación intensiva de flujos de datos.

- Microprocesadores

  El microprocesador es el procesador de propósito general por excelencia; consiste en una Unidad Central de Proceso (CPU) integrada que conforma el núcleo de un sistema digital programable. Su función básica es la de ejecutar las instrucciones de un programa almacenado en memoria, para lo que hace uso de unidades aritméticas lógicas y de un conjunto de registros para direccionamiento de instrucciones y datos, almacenamiento de resultados y control y estado del sistema, figura 1.3:

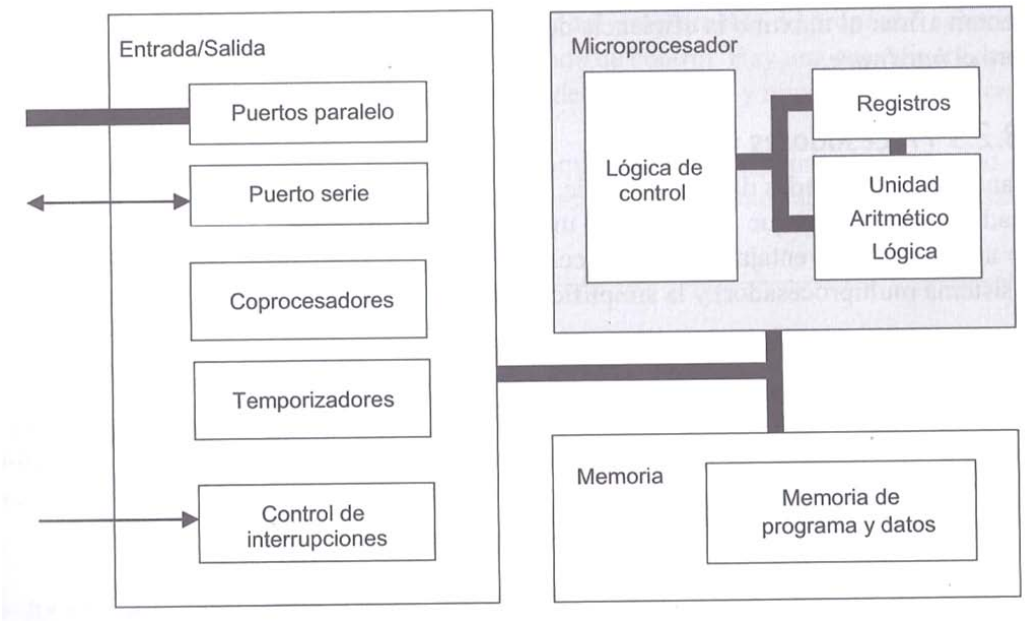


Figura 1.3 Ejemplo de sistema digital programable en un microprocesador.

El microprocesador es capaz de ejecutar cualquier tipo de programa a la máxima velocidad posible. Esta flexibilidad ha requerido un conjunto de instrucciones de propósito general cada vez más potente y variado, grandes cantidades de memoria externa, dispositivos estándar de entrada/salida, un incremento constante de la frecuencia de reloj de KHz a GHz y el tamaño de sus registros de 4 a 64 bits, pasando por 8, 16 y 32.

Los sistemas basados en microprocesadores usan potentes y complejos sistemas operativos que les permiten independizar el hardware de la aplicación, las computadoras personales son ejemplo más característico de este tipo de sistemas.

- DSP

Un DSP (*Digital Signal Processor, Procesador Digital de Señal*) es un microprocesador especializado en el procesamiento de señal con una arquitectura optimizada para llevar a cabo una computación intensiva de flujos de datos en tiempo real. Los DSP trabajan con aritmética fraccionaria, disponen de direccionamientos específicos para buffers circulares e implementación de FFT (*Fast Fourier Transform, Transformada Rápida de Fourier*). También incluyen módulos específicos para mantener la precisión de los algoritmos, así como unidades MAC (*Multiply-ACcumulate, Acumulador de Multiplicación*) capaces de realizar multiplicaciones y sumas en un ciclo de reloj.

No suelen usar sistemas operativos ya que intentan afinar al máximo la eficiencia del código, por lo que se programan directamente sobre el hardware.

- Procesadores unificados

Cuando las necesidades de procesado de señal no son muy exigentes se pueden usar procesadores unificados, que combinan en un único motor de ejecución un microprocesador con un DSP. Sus ventajas son la reducción del costo y complejidad del sistema.

- ASIC

El diseño ideal de un sistema de propósito específico sería un circuito integrado hecho a medida ASIC (*Application Specific Integrated Circuit, Circuito Integrado de Aplicaciones Específicas)*, pero esta solución no es rentable cuando el volumen de producción es bajo o cuando los tiempos y costos de desarrollo son muy justos.

- FPGA

Un componente en auge son la FPGA (*Field Programmable Gate Array, Arreglo de Compuertas Programable*). Es un dispositivo lógico programable y reconfigurable adecuado para bajos volúmenes de producción y muy útil como coprocesador, ya que permite integrar módulos complejos como procesadores, controladores y periféricos estándar.

- Microcontroladores

Un microcontrolador es un pequeño sistema digital programable de propósito específico integrado en un chip especializado en el procesado de control. Hay una gran diversidad de microcontroladores para cubrir la amplia gama de aplicaciones y requerimientos de costo, consumo y espacio que pueden soportar. En la figura 1.4 muestra un diagrama de bloques de un microcontrolador básico que incluye una CPU, memoria de programa y de datos, y diversos interfaces de entrada/salida, tanto analógicos como digitales.

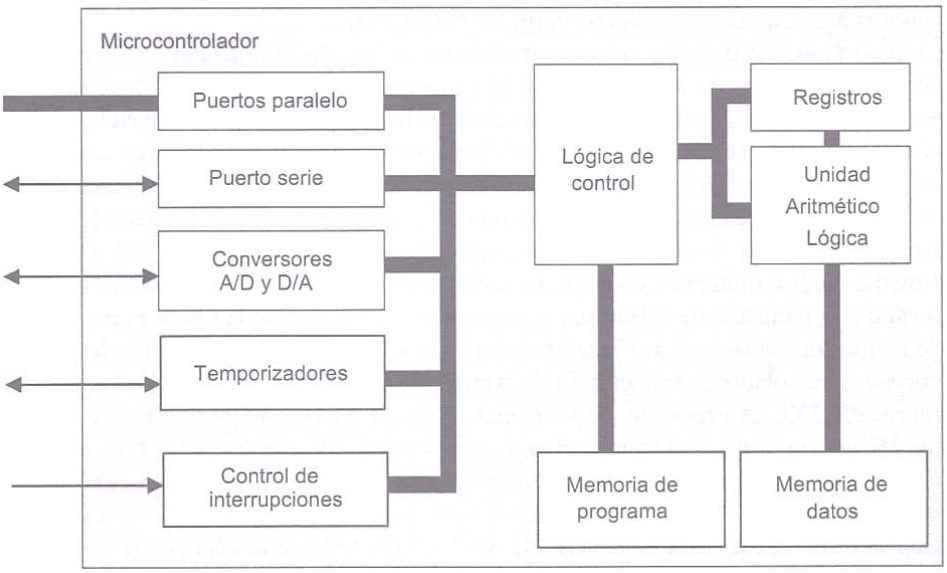


Figura 1.4 Ejemplo de sistema digital programable basado en un microcontrolador.

Otra característica de los microcontroladores es que están capacitados para reducir al mínimo su gasto energético, conmutando entre modos de bajo consumo e incluso cambiado con la frecuencia de trabajo.

El número de microcontroladores en el mercado es enorme, además, los fabricantes suelen proporcionar decenas de versiones del mismo modelo con el objeto de ajustarse lo más posible a los requerimientos del cliente; hay microcontroladores de 4, 8,16 y 32 bits.

Los microcontroladores de 4 bits son muy económicos y adecuados para aplicaciones sencillas, sin embargo, su arquitectura de estos los limita computacionalmente, por lo que están siendo sustituidos por los de 8 bits, que proporcionan alta velocidad.

1.3 Ventajas de la Tecnología Flexis de Freescale

1.3.1 Freescale Semiconductor, Inc.

Es un fabricante estadounidense de semiconductores, creado a partir de la división de semiconductores de Motorola en 2004. Freescale se centra en el mercado de los sistemas integrados y las comunicaciones. Forma parte del top 20 mundial de empresas de semiconductores. Motorola anunció su creación el 6 de Octubre del 2003 y completó su oferta pública inicial el 16 de Julio del 2004.

Freescale también se ha encargado de los procesadores PowerPC para los Apple PowerBook y Mac mini, hasta la transición de Apple a Intel en 2006. La compañía forma parte desde 2006 de Power.org como miembro fundador de esta asociación para el desarrollo y promoción de la arquitectura Power.

En 2006 la empresa desarrolló un microchip que almacena información como si de un disco duro se tratara. El funcionamiento del chip, denominado MRAM (*Magnetoresistive Random-Access Memory, Memoria de Acceso Aleatorio Magnética*), se basa en principios magnéticos en lugar de eléctricos. Freescale comenzó los envíos comerciales de chips MRAM de 4 Mbits el 10 de Julio del 2006, valorando cada chip en 25 dólares.

1.3.2 Familia ColdFire

Desde sus comienzos en 1995, la familia ColdFire se diseñó para proporcionar un conjunto de núcleos procesadores de 32 bits compatibles, 100% sintetizables y con un diseño independiente de la tecnología. El objetivo era facilitar su reutilización y su integración en un solo chip.

La familia ColdFire, compatible con el popular microprocesador 68000, tiene 5 versiones: V1, V2, V3, V4, V5. La versión V1 es la más reciente y fue desarrollada con el propósito de facilitar la migración de los microcontroladores de 8 bits a 32 bits. El resto de las versiones son más potentes en capacidad de proceso y complejidad, las prestaciones van aumentando conforme pasan de la segunda a la quinta versión, pero incompatibles con los microcontroladores de 8 bits [1].

Todos los núcleos comparten la misma arquitectura RISC (*Reduced Instruction set Computing, Computador con Conjunto de Instrucciones Reducidas*); ofrecen múltiples opciones en costo, prestaciones y funcionalidad. Hay modelos con diferentes tamaños de caché o memoria local, punto flotante, gestión de memoria virtual, etc. En la tabla 1.1 se presenta una comparación entre versiones.

Tabla 1.1 Comparativa entre versiones:

ColdFire	***V1***	***V2***	***V3***	***V4***	***V5***
Frecuencia máx. de reloj (MHz)	50	166	240	266	300
Compatible S08	X				
Bus externo (dirección de datos)		X	X	X	X
Módulo de depurado integrado	X	X	X	X	X
Unidad MAC (acumulador múltiple)		X	X	X	X
Unidad de aceleración criptográfica		X	X	X	X
Unidad de punto flotante				X	X

Además, gracias a su diseño SoC (*System on a Chip , Sistema en un solo Chip)*, el núcleo se puede rodear de un conjunto muy variado de módulos periféricos como conmutadores ***crossbar***, controladores ***DMA***, controlador de interrupciones, interfaz a bus externo, controladores ***Ethernet***, controladores ***SDRAM*** externa, puertos serie, etc. La figura 1.5 se muestra un ejemplo de un microcontrolador ColdFire V2, diseñado con módulos específicos para implementar un reproductor de ***MP3***.

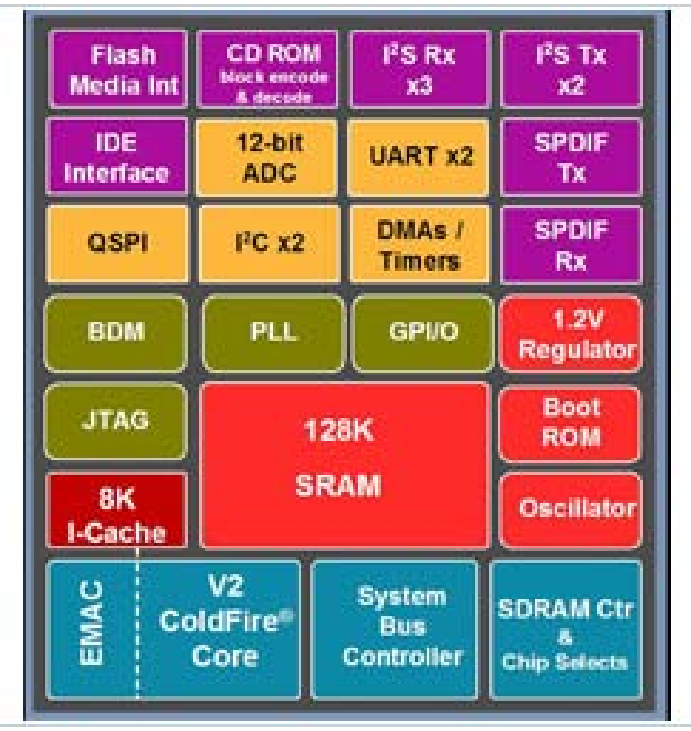


Figura 1.5 Diagrama de bloques del SCF5250.

La familia ColdFire está optimizada para diseño de bajo consumo. Algunos de sus módulos incluyen características avanzadas, tales como sistema de reloj distribuido por módulos y de actividad programable[1]. Otras opciones únicas e innovadoras son ***SRAM*** de doble puerto que habilita concurrencia, temporizadores de 32 bits, convertidores analógicos-digitales de 12 bits, interrupciones vectorizadas, capacidad de control en tiempo real, capacidad de trazas ***BDM***, etc. [1].

1.3.3 Aplicaciones

Las aplicaciones de la familia ColdFire se centran en el control, conectividad y seguridad, para un amplio rango de prestaciones, precio y opciones de integración en aplicaciones tanto industriales como de consumo.

Unos ejemplos son:

- Las aplicaciones de control hacen uso de buses estándar.
- La instrumentación biomédica y las terminales de venta añaden pantallas ***LCD*** de alta resolución y soporte criptográfico.

[1] Permite a cada módulo detener su reloj o modificar su frecuencia de trabajo

- La automatización en fábrica requiere tiempo real y manejo de señales analógicas.
- Los sistemas de seguridad y televigilancia deben ser fiables y económicos.
- La ***VoIP*** precisa de procesado en tiempo real.
- La monitorización remota y recogida de datos que necesitan diagnósticos inmediatos y seguros.

1.3.4 Control Continuo (Controller Continum)

Freescale ha redefinido la compatibilidad entre microcontroladores de 8 y 32 bits mediante su iniciativa denominada controller continum. Esta permite migrar paso a paso entre las arquitecturas de los S08 de 8 bits y los ColdFire de 32 bits, pasando desde los RS08 de ultra bajo consumo hasta los de ColdFire V4 de muy altas presentaciones. Esta migración puede ser en los dos sentidos, figura 1.6:

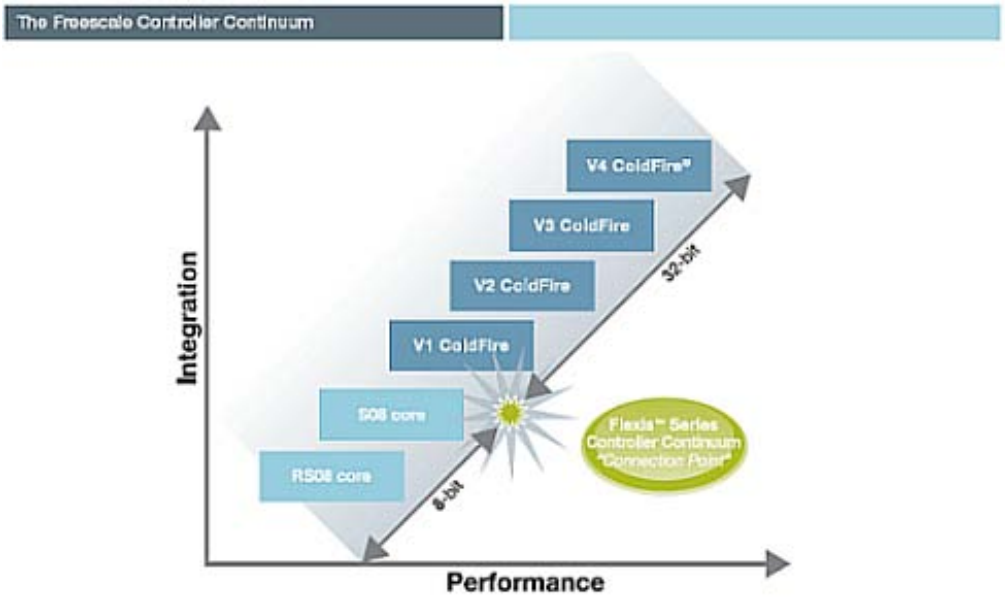


Figura 1.6 Control Continuo de Freescale.

En el punto de conexión entre 8 y 32 bits del controller continum se encuentra la serie Flexis basada en el microcontrolador ColdFire V1 [1].

1.3.5 Migración entre S08 y ColdFire V1

La migración entre S08 y ColdFire V1 se debe tener presente que los núcleos son completamente diferentes: sus buses, conjunto de instrucciones, proceso de interrupciones, acceso a memoria o capacidad de depurado, la tabla 1.2 se pueden apreciar algunas de las diferencias más significativas.

Tabla 1.2 Diferencias más significativas entre S08 y ColdFire V1

Características	*S08*	*ColdFire V1*	*Comentarios*
Bus de direcciones	16	32	
Bus de datos	8	32	
Conjunto de instrucciones	S08	ColdFire rev. C	Con instrucciones call y return adicionales
Número de interrupciones	32	256	Excepciones de reset
Interrupciones anidadas	No	Sí	
Puerto paralelo rápido	No	Sí	
Paginación	Sí	No	
Flash	2 x 64K x 8 en serie	2 x 32K x1 6 en paralelo	

Conviene destacar algunas características compatibles, como por ejemplo, el uso de los mismos periféricos (con excepción del puerto paralelo rápido), casi todos los modos de bajo consumo, la tensión de alimentación, la interfaz de depurado BDM o las mismas herramientas de desarrollo [1].

1.3.6 Migración entre ColdFire V1 y ColdFire V2

En la migración entre ColdFire V1 y V2 los núcleos son muy parecidos, ambos tienen la misma microarquitectura, modelo de programación, conjunto de instrucciones y modos de direccionamiento. Sin embargo, el resto de los módulos puede variar considerablemente. Además, no son chips compatibles pin a pin y usan sistemas de desarrollo diferentes. La tabla 1.3 se pueden ver algunas de las diferencias más significativas [1].

Tabla 1.3 Diferencias más significativas entre ColdFire V1 y ColdFire V2.

Características	***ColdFire V1***	***ColdFire V2***	***Comentarios***
Bus de direcciones	24	32	
Bus de datos	32	32	
Conjunto de instrucciones	ColdFire rev. a+	ColdFire rev. C	
Buffer del puerto Serie	No	Sí	
Temporizador	16	32	
Unidad de PWM	No	Sí	Integrada en el Temporizador
Flash	<128K	<152K	

1.3.7 Series Flexis

La serie Flexis está formada por pares de microcontroladores prácticamente idénticos, con la única diferencia que en la pareja uno lleva un núcleo de 8 bits (S08) y el otro un núcleo de 32 bits (ColdFire V1), siendo el resto del chip exactamente igual, figura 1.7. A esto se le añade una herramienta común de desarrollo "***CodeWarrior"*** para microcontroladores V6.2.

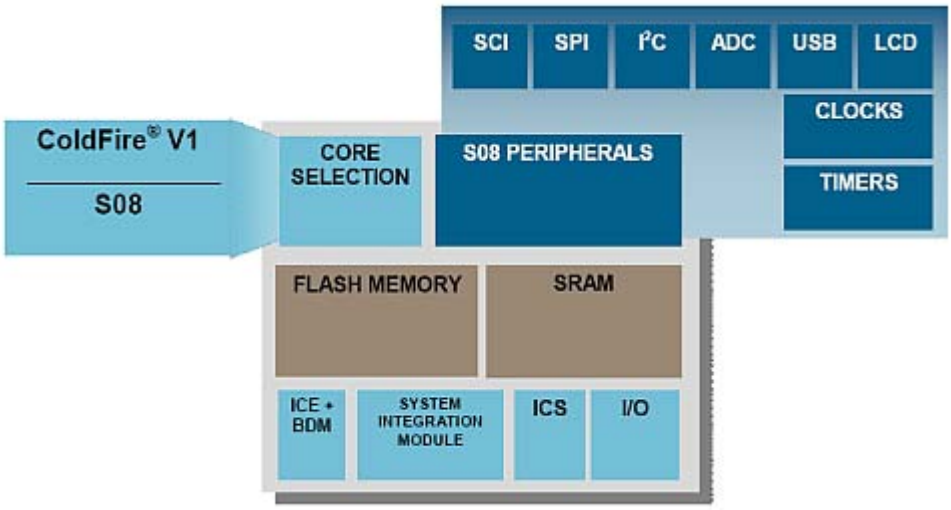


Figura 1.7 Serie Flexis de Freescale.

Esta filosofía de incorporar dos núcleos diferentes tiene varias ventajas, como se usan los mismos periféricos de 8 bits se puede reutilizar el software asociado, por otro lado, como los microcontroladores S08 y ColdFire V1 son compatibles pin a pin se puede reutilizar la misma plataforma hardware.

Esto facilita el poder probar, simplemente cambiando el chip del procesador múltiples configuraciones y niveles de prestaciones. Si se precisa velocidad de proceso extra, se puede migrar a 32 bits, pero si necesita reducir el consumo, se puede ir a 8 bits. Y lo que es más importante, los tiempos de desarrollo debido a la migración se puede reducir drásticamente.

La primera pareja de microcontroladores de la serie Flexis es el dúo QE128 formada por los chips MC9S08QE128 (S08) y MCF51QE128 (ColdFire V1).

Una de las características de esta nueva familia Flexis de Freescale es el "Controller Continum", que es la compatibilidad total entre pins, periféricos y herramientas de desarrollo entre los microcontroladores de 8-bits (arquitectura HCS08) y los microcontroladores de 32-bits (arquitectura ColdFire V1) [1].

El hardware utilizado es la tarjeta de demostración DEMOJM60, es una tarjeta que permite hacer experimentos usando comunicación serial, ***SPI, I^2C***, generar ***PWM***, así como el convertidor analógico digital y además, de interactuar con dispositivos externos al microcontrolador como son:

- 1 acelerómetro de 3 ejes,
- 8 Leds,
- 4 botones,
- 1 buzzer,
- 1 potenciómetro,
- Interfaz ***USB***,
- Cristal externo

La tarjeta DEMOJM60 se muestra en la figura 1.7:

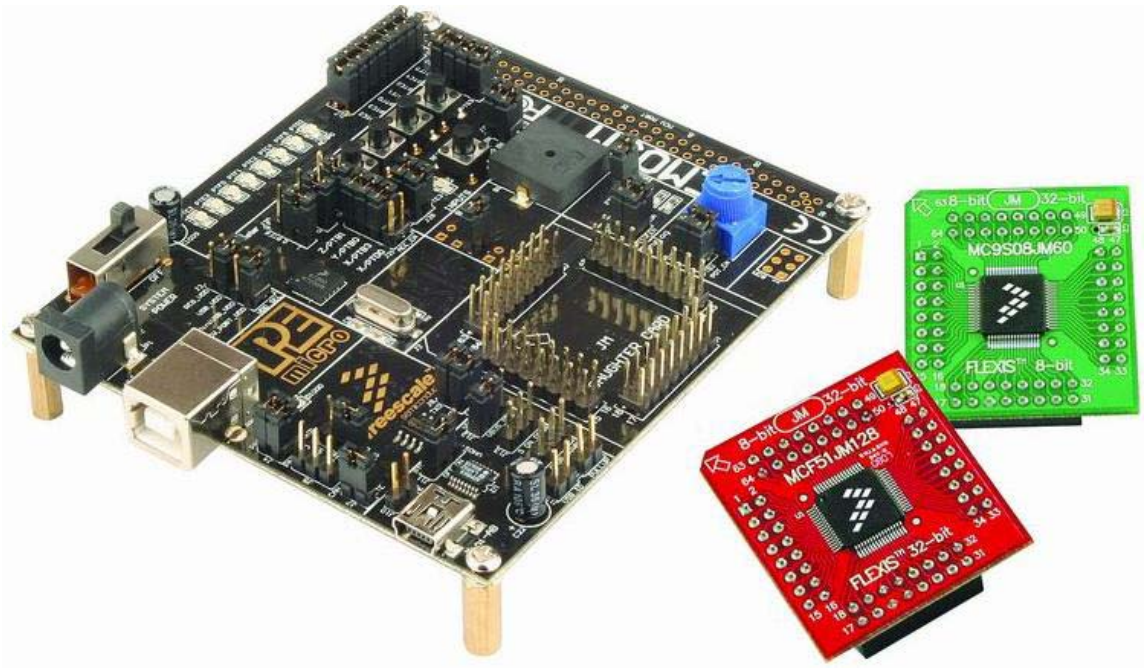
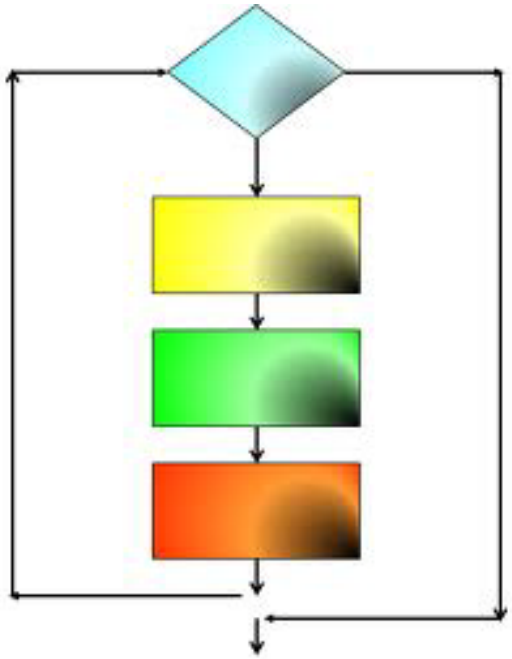


Figura 1.7 Tarjeta de desarrollo JM60 de Freescale.

CAPÍTULO 2: METODOLOGÍA PARA EL DISEÑO MECATRÓNICO



2.1 Planteamiento, objetivos y justificación para la elaboración de prácticas de diseño mecatrónico

La realización de estas prácticas es debido a que la asignatura optativa de Diseño Mecatrónico, de la carrera de ingeniería Mecánica, es de índole teórica-práctica y no cuenta con prácticas correspondientes para dicho laboratorio. Por lo cual, se pretende que al alumno le sirva como manual de prácticas para el laboratorio y aplicar los conceptos que se aprenden en la teoría.

La materia de Diseño Mecatrónico de la carrera de Ingeniería Mecánica tiene como objetivo del curso (ver anexo 1):

El alumno aplicará los principios de operación de los sistemas mecatrónicos a través del estudio de los microprocesadores y su aplicación en el diseño de sistemas industriales que integran elementos mecánicos, electrónicos y de programación.

El tema I correspondiente con los apartados 2.2 y 2.3: de este capítulo se pretende que el alumno enuncie la importancia de la mecatrónica y sus aplicaciones en la industria.

En tema II correspondiente con los apartados 2.4 y 2.5: el alumno analizará los métodos utilizados en el diseño de sistemas mecatrónicos.

El tema III correspondiente con los apartados 2.6: el alumno analizará la arquitectura, funcionamiento y programación de un microprocesador.

El tema IV correspondiente con los apartados 2.7, 2.8 y 2.9: se describirán los tipos de sensores, actuadores e interfaces hombre-máquina en un sistema mecatrónico.

2.2 Antecedentes de la mecatrónica

Hoy en día el avance de la tecnología se encuentra en constante crecimiento. El desarrollo tecnológico es un aspecto importante para los países en vías de crecimiento. La trascendencia del desarrollo científico no se limita a sus consecuencias económicas, también contribuye a elevar la vida política y social, y por lo tanto, la capacidad del país para dirigir su propio destino.

La mecatrónica como parte de la automatización y la robótica generan tecnología, siendo un área estratégica para los países, ya que su impacto no sólo repercute en aspectos políticos y económicos, sino también forma parte de la vida cotidiana, educación, cultura y sociedad.

A mediados de los años cuarenta del siglo pasado, la introducción del transistor hecho con material semiconductor inicia la segunda revolución industrial, hacia la miniaturización de los componentes electrónicos acoplados en circuitos integrados, para generar la computadora digital como producto que cambió la mentalidad en la industria y en la sociedad.

En esas dos épocas, los países que emplearon pero, especialmente que produjeron las tecnologías se pusieron a la vanguardia de la sociedad. En la actualidad, la mecatrónica es un concepto nuevo en torno a las tecnologías que concita los productos específicos en esas dos revoluciones: la integración de las máquinas a las computadoras digitales, para crear un nuevo ambiente en el tercer milenio.

La palabra Mechatronic fue compuesta por el veterano ingeniero japonés Yaskawa en 1969, como una combinación de “Mecha” de Mechanisms y “tronics” de electronics. La nueva palabra muy pronto ganó aceptación y empezó a usarse desde 1982 por la industria moderna. En sentido amplio, mecatrónica es una jerga técnica que describe la filosofía en la tecnología de la ingeniería, en lugar de un simple término técnico.

Muchas definiciones se han propuesto para la mecatrónica, pero su amplitud conceptual no ha permitido normalizar ninguna de ellas. Las definiciones más comunes enfatizan en la asociación: ***La mecatrónica es la integración de la ingeniería mecánica con la ingeniería eléctrica y electrónica basada en control inteligente computarizado para el diseño y manufactura de productos y procesos***.

Históricamente, el desarrollo de la mecatrónica se ha realizado en tres etapas:

- La primera corresponde a la introducción de la palabra en el medio industrial y su aceptación. Durante esta las tecnologías que la integran se desarrollaron independientemente.

- La segunda etapa se inicia a comienzos de los años 80 y se caracterizó por la integración sinérgica de sus diferentes tecnologías, como la integración de la óptica a la electrónica para conformar la optoelectrónica y el diseño integrado de hardware/software.

- En tercera etapa puede considerarse como la que inicia la era de la mecatrónica y se basa en el desarrollo de la inteligencia computacional y los sistemas de información. Una característica importante de paso es la miniaturización de los componentes en forma de microactuadores y microsensores integrados en la micromecatrónica.

Un robot es un ejemplo de tecnología mecatrónica en acción. La mecánica contribuye en el diseño y selección de componentes para la estructura del robot, como: materiales, mecanismos, articulaciones, transmisiones, actuadores, análisis cinemático, análisis dinámico, análisis de cargas, momentos de inercia, confiabilidad y seguridad.

La electricidad y electrónica contribuyen en el diseño y selección de componentes, como: sensores, transductores, circuitos, redes, servomecanismos, interfaces, amplificadores, convertidores de señales, acondicionadores de señales, sistemas de potencia y sistemas de visión.

La ciencia computarizada contribuye con el controlador especialmente. Los sistemas de información permiten la integración de los componentes de la mecatrónica y contribuyen con software para la simulación, modelado, supervisión, diseño de sistemas de control, programación de trayectorias, optimización, dibujo y diseño asistidos por computador *CAD* (*Computer-aided design, Diseño asistido por computadora*) de la estructura del robot. La figura 2.1 muestra los componentes de un sistema mecatrónico.

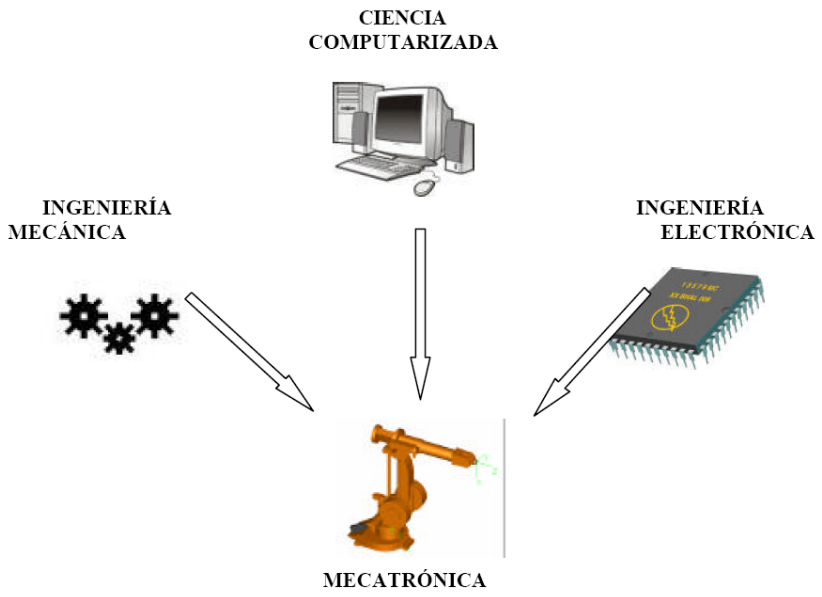


Figura 2.1 Componentes de un sistema mecatrónico.

2.3 Sistemas integrados

El término se utiliza cuando los microprocesadores son construidos dentro de los sistemas, y éste es el tipo que por lo general interesa en la mecatrónica. Un microprocesador puede considerarse básicamente como una colección de compuertas lógicas y elementos de memoria que no están comunicados como componentes individuales, pero cuyas funciones lógicas se implementan mediante software.

Con el objetivo de que se utilice un microprocesador en un sistema de control, necesita chips adicionales para dar memoria al almacenaje de datos y para puertos entrada/salida, con el fin de habilitarlos en las señales de proceso desde y para el mundo externo. Los microcontroladores son microprocesadores con estas instalaciones extra, todas ellas integradas en un solo chip.

Un sistema integrado está basado en un microprocesador que está diseñado para controlar una gama de funciones y no está diseñado para que el usuario final lo programe de la misma forma que una computadora. Por lo tanto, con un sistema de este tipo, el usuario no puede cambiar lo que ya se encuentra, añadir o reemplazar el software.

En un automóvil moderno tendrá microprocesadores que controlen funciones como el sistema antibloqueo de frenos y el sistema de mando del motor. Se muestra en la figura 2.2 el esquema a bloques de una computadora del sistema *ABS (Antilock Braking System, Sistema de Antibloqueo de Frenado*).

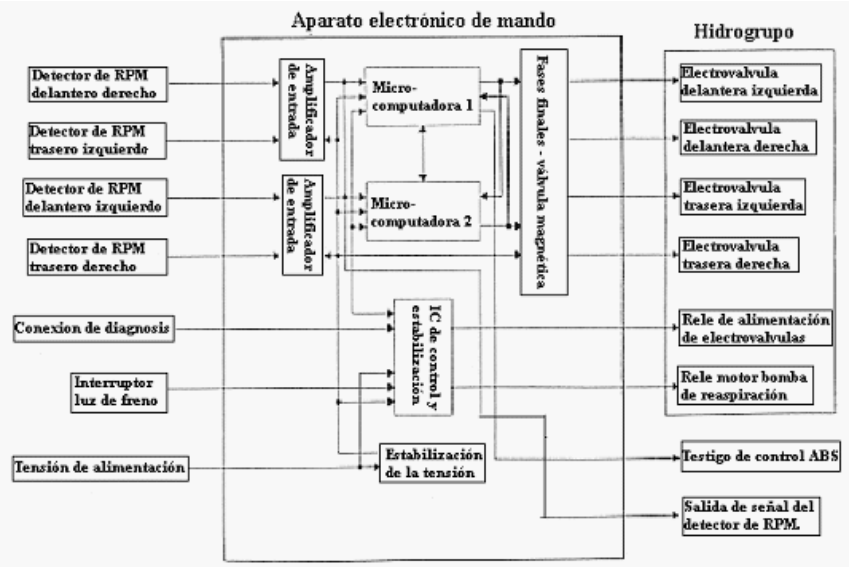


Figura 2.2 Esquema a bloques de una computadora del sistema ABS de un automóvil.

2.4 Metodología en el desarrollo de productos

2.4.1 Diseño mecatrónico

El proceso de diseño para cualquier sistema puede considerarse como el que involucra las siguientes etapas:

- **La necesidad**

El proceso de diseño comienza con una necesidad, quizá del consumidor o cliente. Esto se puede detectar en la investigación del mercado, que se lleva a cabo para establecer las necesidades de clientes potenciales.

- **Análisis del problema**

El primer paso en el desarrollo de un diseño es investigar la naturaleza verdadera del problema. Esta es una etapa importante en cuando a que si el problema no se define con exactitud, puede ocasionar pérdida de tiempo en los diseños.

- **Preparación de una especificación**

En el planteamiento del problema se deberán especificar todas las funciones requeridas del diseño, junto con cualquier otra característica deseable. De esta manera puede haber una exposición del volumen, dimensiones, precisión de requerimientos de entrada y salida de los elementos, interfaces, potencia y entorno operativo.

- **Generación de soluciones posibles**

A esto se le califica por lo general como etapa conceptual. Los bosquejos de soluciones se preparan, mismos que funcionan con los detalles suficientes que indican los medios para obtener cada una de las funciones requeridas, por ejemplo: tamaños aproximados y muestras de materiales y costos. También significa investigar lo que se ha hecho anteriormente ante problemas similares.

- **Selecciones de una solución apropiada**

Las diversas soluciones se evalúan y la más apropiada es la que se selecciona, la evaluación a menudo incluye la representación de un sistema mediante un modelo para luego llevar a cabo una simulación, con el objetivo de establecer cómo puede reaccionar a las entradas.

- **Producción de un diseño detallado**

El detalle de un diseño seleccionado debe funcionar para este momento del proceso. Éste puede requerir la producción de prototipos o maquetas de tamaño natural para determinar los detalles óptimos de un diseño.

- **Producción de dibujos de trabajo**

El diseño seleccionado se traduce entonces en dibujos de trabajo y diagramas de circuitos, de manera que se pueda elaborar.

Cada etapa del proceso de diseño no se debe considerar como algo independiente, y a menudo se necesitará regresar a una etapa previa y darle mayor consideración. Así, cuando se presente un problema, puede haber la necesidad de reconsiderar el análisis del mismo en la etapa de generación de soluciones posibles.

2.5 Diseños tradicionales y mecatrónicos

El diseño de ingeniería es un proceso complejo que implica interacciones entre varias habilidades y disciplinas. Con el diseño tradicional, la propuesta era que el ingeniero mecánico diseñara los elementos mecánicos, luego el ingeniero de control progresara y diseñara el sistema de control. Esto da como resultado lo que se conoce como enfoque secuencial para el diseño. Sin embargo, la base del enfoque de la mecatrónica se considera que yace en la inclusión concurrente de las disciplinas de la ingeniería mecánica, electrónica, computación e ingeniería de control en el enfoque de diseño. La concurrencia inherente de este enfoque depende mucho del modelo del sistema de simulación, de la manera en la que el modelo reacciona a las entradas y por consiguiente, cómo puede reaccionar el sistema real.

Tradicionalmente, el diseño de los sistemas multidisciplinarios ha sido secuencial, o diseño por disciplina. Por ejemplo, el diseño de un sistema electromecánico a menudo se desarrolla en tres etapas:

- Primero, el diseño mecánico.

- Segundo, cuando el diseño mecánico se ha realizado, se continúa con el sistema de potencia y de los dispositivos microelectrónicos.

- Tercero, es donde el diseño del sistema de control contribuye con restricciones que se han acumulado a lo largo del desarrollo de las dos etapas anteriores.

La principal desventaja del diseño por disciplina es que en algunas etapas de la secuencia emergen parámetros y restricciones muy rígidas, con lo que el diseño no es muy flexible.

2.5.1 Ingeniería concurrente

La metodología del diseño mecatrónico se basa en la CE (*Concurrent Engineering, Ingeniería Concurrente*) en lugar del método secuencial, resultando una aproximación al diseño de productos más sinérgicos.

A principio de los años 90, la ingeniería concurrente emergió como una tendencia en el diseño y manufactura automatizados de nuevos productos en FMS (*Flexible Manufacturing Systems, Sistemas de Manufactura Flexibles*). No se conoce una definición normalizada de la ingeniería concurrente por lo novedoso del término; algunos autores homologan la ingeniería concurrente y la SE (*Simultaneous Engineering, Ingeniería Simultánea*). Se considera que la implementación de la CE es con base en el empleo de la computadora en el diseño y la manufactura. Se afirma que la CE generalmente se ha reconocido como una práctica de diseño y manufactura, que considera varios valores de ciclos de vida para un producto desde su etapa de diseño y abarca otros aspectos, como facilidad de manufactura, facilidad de ensamblaje, y funcionalidad.

Se define CE como la integración más temprana posible de todos los conocimientos de la fábrica en recursos, experiencia en el diseño, investigación, mercadeo, manufactura y ventas para la creación de un nuevo producto, con alta calidad y bajo costo para complacer las expectativas de los clientes.

En la actualidad, hay investigadores que piensan que la CE es sinónimo de un buen equipo de trabajo interdisciplinario, con buenas relaciones y comunicaciones entre las funciones en cada departamento o especialidad.

El objetivo de la CE es reducir los tiempos en el diseño y la manufactura por superposición de actividades. En la ingeniería secuencial, cada proceso de diseño y manufactura se debe cumplir antes de pasar a la siguiente etapa. Por ejemplo, el diseño del producto debe completarse antes de entrar a la etapa del proceso de manufactura, de esta forma el tiempo de todo el proceso es mayor en comparación con el lapso que cuando se aplica la CE.

Para crear un ambiente en donde la CE sea puesta en práctica es necesario considerar los siguientes atributos:

- Primero. Tener un equipo de trabajo.

- Segundo. El equipo de trabajo debe tener comunicación intensiva y cooperativa entre sus funciones computarizadas en red.

- Tercero. El equipo de trabajo debe tener el poder de tomar decisiones de acuerdo a su conocimiento y experiencia.

- Cuarto. Lo anterior es solamente posible en una organización horizontal en donde la integración es posible, las comunicaciones son claras, y el personal está directamente involucrado en la ejecución del trabajo y programas.

En el diseño mecatrónico de robots, hay una integración sinérgica de los sistemas, mecánicos, sistemas electrónicos, y sistemas computarizados como un todo, en donde los diseñadores de cada área intervienen en todas las etapas del diseño. Esta sinergia es generada por la combinación correcta de los parámetros, de tal manera, que el diseño final del robot es tan bueno como la suma de las partes que lo componen.

Un sistema mecatrónico no es un simple sistema electromecánico y es más que un sistema de control. Los sensores y actuadores en un sistema mecatrónico convierten energía de alta potencia, usualmente del lado mecánico, a baja potencia, del lado eléctrico y computador.

Además de los componentes mecánicos del lado mecánico también hacen parte, los sistemas neumáticos, los sistemas hidráulicos, los sistemas térmicos, los sistemas acústicos, y otras disciplinas afines. El control automático en robótica implica que la máquina llamada robot es controlada por otra máquina llamada computador.

La mecatrónica hace posible la combinación de los actuadores, sensores, sistemas de control y computadores en una estructura mecánica en el proceso de diseño. La figura 2.3 ilustra los fundamentos de la mecatrónica.

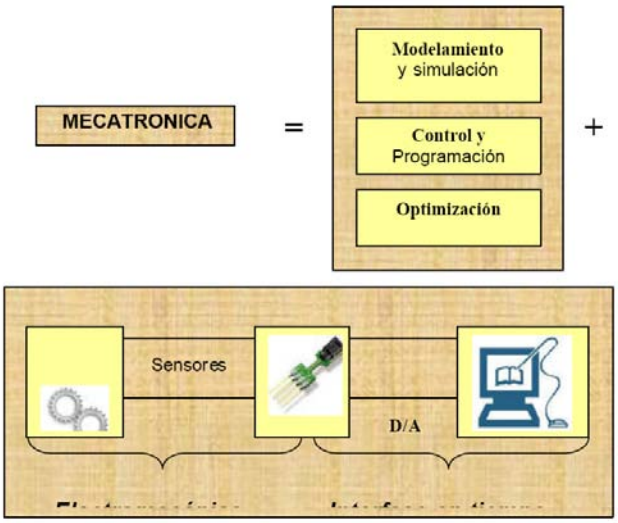


Figura 2.3 fundamentos de la mecatrónica.

A partir de un diseño básico de la estructura, y progresando a través de la manufactura del robot, el diseño mecatrónico optimiza sus parámetros en cada fase del diseño para lograr un producto de calidad, en un corto periodo de tiempo.

El diseño mecatrónico se apoya en el concepto de ingeniería concurrente, y es necesario que el conocimiento y la información requerida sean coordinados por diferentes grupos de expertos. La CE es una aproximación al diseño, en el cual la manufactura del producto se fusiona de una manera especial. Las barreras tradicionales entren ambas desaparecen, particularmente influidos por el reconocimiento que los altos costos de manufactura se deciden en la etapa de diseño del producto.

Durante la etapa de diseño, el producto mecatrónico es regulado por requerimientos tales como que sea manufacturado y de fácil ensamblaje, fácil de inspeccionar, de fácil mantenimiento, comportamiento óptimo durante su ciclo de vida, calidad y confiabilidad.

El empleo eficiente de la ingeniería concurrente en el proceso de diseño y manufactura permite el logro de varios objetivos, entre ellos se tienen:

- Diseño para la manufactura y el ensamblaje.
- Diseño para la calidad.
- Diseño para un ciclo de vida predeterminado.
- Diseño para minimizar costos.

2.6 Computadoras

La revolución en las tecnologías computacionales de las últimas décadas ha cambiado el panorama de todas las disciplinas en ingeniería. La integración de las computadoras ha dado origen a una nueva variedad de esquemas en la práctica de la ingeniería.

La CAD (*Computer Aided Design, Diseño Asistido por Computadora*) define una tecnología en donde el computador es herramienta fundamental para la solución de tareas en ingeniería mecánica como análisis, diseño, producción y control.

Los sistemas ***CAD/CAM*** (*Computer Aided Design and Computer Aided Manufacturing, Diseño Asistido por Computadora y Manufactura Asistida por Computadora*) son tecnologías computarizadas que han permitido la automatización del dibujo, diseño y manufactura en la ingeniería mecánica. La computación gráfica es otra tecnología que ha permitido a los investigadores la visualización y animación de modelos.

La mecánica computacional es empleada en la investigación como el arte y la ciencia de simulación de problemas mecánicos en el computador; su aplicación se extiende a problemas de las industrias aeroespaciales, automotrices, termofluidos, biomédicos e ingeniería electromagnética. Una de las formas de aplicación de la mecánica computacional es el software de análisis de elementos finitos.

En automatización y control, las computadoras digitales realizan una importante función de control de sistemas mecatrónicos. Hoy en día tratar de solucionar problemas de ingeniería en forma aislada de las otras disciplinas es difícil y complejo.

- **Programación**

En robótica el computador es el controlador fundamental, pero además, es la herramienta mediante la cual se programan las tareas a desarrollar por el robot en su espacio de trabajo. Programar un robot consiste en indicar paso a paso las diferentes acciones que deberá realizar durante su funcionamiento automático.

La flexibilidad en la utilización del robot dependerá en gran medida en las características del sistema de programación. Cada fabricante desarrolla su método particular y aplicable sólo a sus propios robots. Sin embargo, existen algunos métodos que han servido de modelo para el desarrollo de otros.

Los primeros robots industriales fueron simples máquinas secuenciales controladas por una combinación de servomotores, detención mecánica ajustable, interruptores de límite y PLC *(Programmable Logic Controller, Controladores Lógicos Programables).* Estas máquinas eran programadas por discos, el usuario se comunicaba y operaba al robot por medio de un módulo de enseñanza manual para moverlo en la ruta deseada.

Durante el año 1970 también se desarrolló el lenguaje MINI en el MIT (*Massachusetts Institute of Technology, Instituto Tecnologico de Massachusetts*) como una extensión del lenguaje LISP (*LISt Processing, Proceso de LIStas*) y se empleó en un robot de coordenadas cartesianas.

En la década de los 80 en la Universidad de Stanford se desarrolló el lenguaje AL basado en programación ALGOL y PASCAL. En Unimation Inc., la empresa que creó el robot PUMA, y se desarrollaron los lenguajes VAL y VALII en 1984. El lenguaje AML fue desarrollado por la empresa IBM para un robot ensamblador en 1982.

La compañía McDonnell Douglas desarrolló el lenguaje MCL como una extensión del APT (originalmente escrito para el control de máquinas NC). Automatix Inc. desarrolló el lenguaje RAIL como una extensión del PASCAL y lo empleó en robots que desempeñaban funciones de soldadura. V+ es un lenguaje de programación de robots moderno desarrollado por Adept Technologies Inc. en 1989, que contiene cientos de instrucciones. La Universidad de Grenoble desarrolló el lenguaje LM en 1981. Estos son algunos de los lenguajes de programación de robots industriales que se han desarrollado en el mundo.

Los investigadores los clasifican en tres niveles:

- La programación por aprendizaje consiste en hacer realizar al robot la tarea conduciéndolo manualmente y registrando las configuraciones adoptadas, para posteriormente repetirlas de manera automática.

- La programación textual permite indicar la tarea al robot mediante el uso de un lenguaje de programación específico, con una serie de instrucciones escritas en un computador digital.

- La programación off line se realiza mediante un software especializado, que permite programar y simular antes de implantar en un robot industrial.

- **Modelado**

Es el proceso de representar el comportamiento de un sistema robótico por medio de un conjunto de ecuaciones matemáticas y lógicas en un proyecto de un robot. Los modelos pueden ser estáticos o dinámicos.

- En un modelo estático no hay transferencia de energía.

- En un modelo dinámico hay transferencia de energía, que produce el movimiento.

Los fenómenos observados son señales, generalmente modelados respecto al tiempo. Actualmente hay software en el mercado para el modelamiento en una estación computarizada: Matrixx, Easy5, SimuLink, VisSim, LabView, y otros.

- **Simulación**

Es el proceso de resolver el modelo de un proyecto y se visualiza en una computadora. El proceso de simulación puede dividirse en tres etapas: iniciación, iteración y finalización. La iniciación puede ser el diagrama de bloques con sus correspondientes ecuaciones. La iteración es la resolución de las ecuaciones diferenciales del modelo por medio de la integración o la diferenciación.

Las ecuaciones diferenciales que representan el modelo generalmente son no lineales que contienen derivadas en función de la variable independiente, tiempo. La ecuación diferencial tiene el grado del término de más alta derivación.

En programación off line, el modelado y la simulación se realizan mediante un software especializado, en donde se puede seleccionar el robot de una base de datos para analizar su comportamiento en su ambiente de trabajo virtual, antes de ser implantado en su ambiente real.

Con esta simulación de la programación se ahorra tiempo y dinero, porque toda la programación puede efectuarse en una estación de computadora, mientras los robots industriales reales continúan trabajando con la programación anterior. Generalmente este software de modelamiento y simulación tiene un sistema CAD para diseñar el layout (capas) de las facilidades en donde se desempeñará el robot. Este software también se emplea en el proyecto de un nuevo robot industrial.

- **Control automático**

El control automático moderno es una disciplina que apareció después que Maxwell descubrió la estabilidad de un sistema de primero, segundo o tercer orden y que depende de las raíces de la ecuación característica y del método de Ruth, el cual suministra una herramienta de evaluación para sistemas de más alto orden. Esto ocurrió en la última década del siglo XIX.

El modelo matemático de la dinámica de la estructura de un robot, es una ecuación diferencial de segundo orden. Extendiendo el modelo dinámico a los actuadores y transmisiones, este modelo puede ser mayor del tercer orden. La computadora es la herramienta más importante, hoy en día es empleado en el diseño e implementación del sistema de control de robots industriales.

- **Optimización**

En mecatrónica, la optimización se emplea para establecer la configuración óptima del sistema robótico cuando se desarrolla un proyecto, pero también se emplea para la identificación de las trayectorias óptimas, diseño del sistema de control e identificación de los parámetros del modelo, haciendo uso de software de modelamiento y simulación.

2.7 Sensores

Para que un sistema electrónico pueda contralar un proceso o un producto industrial es necesario obtenga información de la evolución de determinadas variables físicas del mismo, en su mayoría no son eléctricas, como puede ser la temperatura, presión, fuerza, la radiación luminosa, posición, aceleración y desplazamiento de un objeto, etc. Por ello, el acoplamiento entre el sistema electrónico y el proceso productivo se debe realizar a través de dispositivos que convierten las variables no electrónicas.

Dichos dispositivos reciben diversos nombres, como:

- Captador
- Detector
- Transductor
- Transmisor
- Sensor

Aunque este último es el más utilizado por los fabricantes de sistemas electrónicos de control en general, no existe una única definición de sensor aceptada de manera universal. Se considera que, en general, un sensor es un dispositivo que, situado en un cierto medio, genera una señal (función de alguna característica del medio) de una determinada forma física.

El elemento que realiza conversión se suele denominar transductor y cada vez es más usual denominar sensor al conjunto formado por el dispositivo anteriormente descrito, y al transductor acoplado a él. Por otro lado, la existencia de sistemas que memorizan, amplifican, y en general procesan señales eléctricas que hacen que la mayoría de los transductores conviertan variables no eléctricas en eléctricas.

El tipo de señal eléctrica portadora de información y sus parámetros varían de un dispositivo sensor a otro, y por ello, es necesario acoplar la salida de éste a un circuito que, de acuerdo con las características de aquélla, realice al menos una de las siguientes operaciones:

- Amplificación
- Filtrado
- Corrección
- Conversión

Dicho circuito recibe el nombre de circuito acondicionador o de acondicionamiento y su utilización conjunta con el elemento sensor da lugar a un sistema como el representado en la figura 2.4.

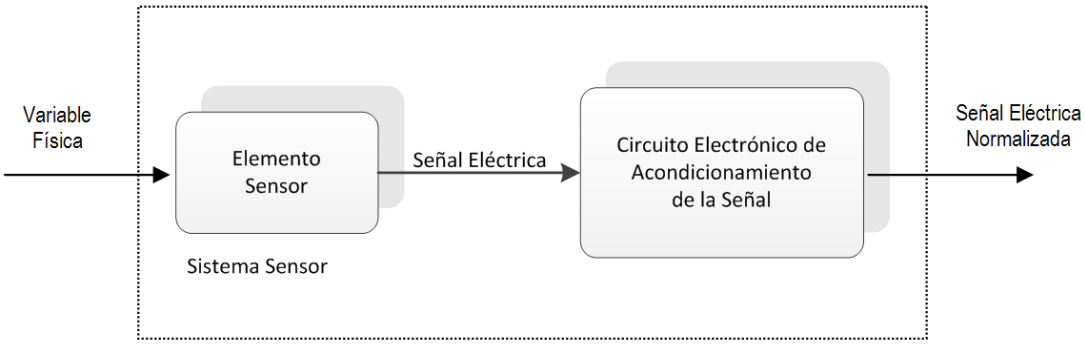


Figura 2.4 Componentes básicos de un sensor

Se denominada sensor a dicho sistema que genera una señal normalizada, ya sea el fabricante o siguiendo pautas establecidas por organismos de normalización, como por ejemplo la *IEEE* (*Institute of Electrical and Electronics Engineers, Instituto de Ingenieros en Electricidad y Electrónica)*.

De todo lo expuesto se concluye que el análisis de los sensores es una tarea compleja y la selección del más adecuado para una implementación determinada obliga a tener en cuenta múltiples factores. Los sensores se pueden clasificar de acuerdo con un conjunto de características diferentes y no excluyentes, que se indican en la figura 2.5.

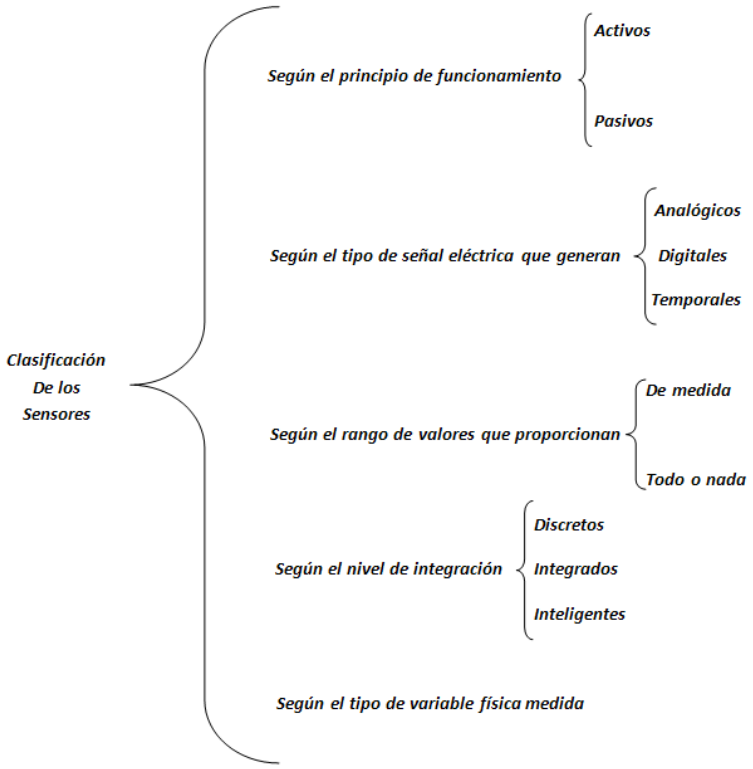


Figura 2.5 Clasificación de los sensores.

2.8 Actuadores

Un actuador es un dispositivo capaz de utilizar energía hidráulica, neumática o eléctrica para la activación de un proceso, con la finalidad de generar un efecto sobre un proceso. Este recibe la orden de un regulador o controlador y en función a ella genera la parte activa en un elemento final de control como, por ejemplo, una válvula.

Existen varios tipos de actuadores como son:

- Eléctricos
- Hidráulicos
- Neumáticos
- Electrónicos

Los actuadores hidráulicos, neumáticos y eléctricos son usados para manejar aparatos mecatrónicos. Por lo general, los actuadores hidráulicos se emplean cuando lo que se necesita es potencia y los neumáticos son simples posicionamientos [5].

2.9 Interfaz hombre-máquina

Los conceptos de visualización y actuación surgieron de forma natural cuando el ser humano comenzó, a principios del siglo XX, a realizar maquinas electromagnéticas gobernadas mediante un sistema de control; recibía órdenes o consignas de un operador y generaba señales que actuaban sobre la citada máquina.

Los sistemas de control lógico se realizaban en ese entonces con relés y los procesos con válvulas, y las órdenes o consignas las tenía que dar el usuario, para lo cual utilizaba elementos cuyo funcionamiento era fácil de entender. Se trataba de interruptores y pulsadores, en el caso de variables lógicas o todo-nada, y potenciómetros, para generar variables analógicas.

Para visualizar las variables analógicas se utilizaban, cuando era necesario, galvanómetros convertidos en amperímetros o voltímetros, y para visualizar las variables todo-nada, pilotos que eran lámparas de incandescencia o neón de reducido tamaño, gobernadas mediante relés [5]. Por lo tanto, la interfaz usuario-máquina estaba formada por un conjunto de los citados, elementos como se muestra en la figura 2.6.

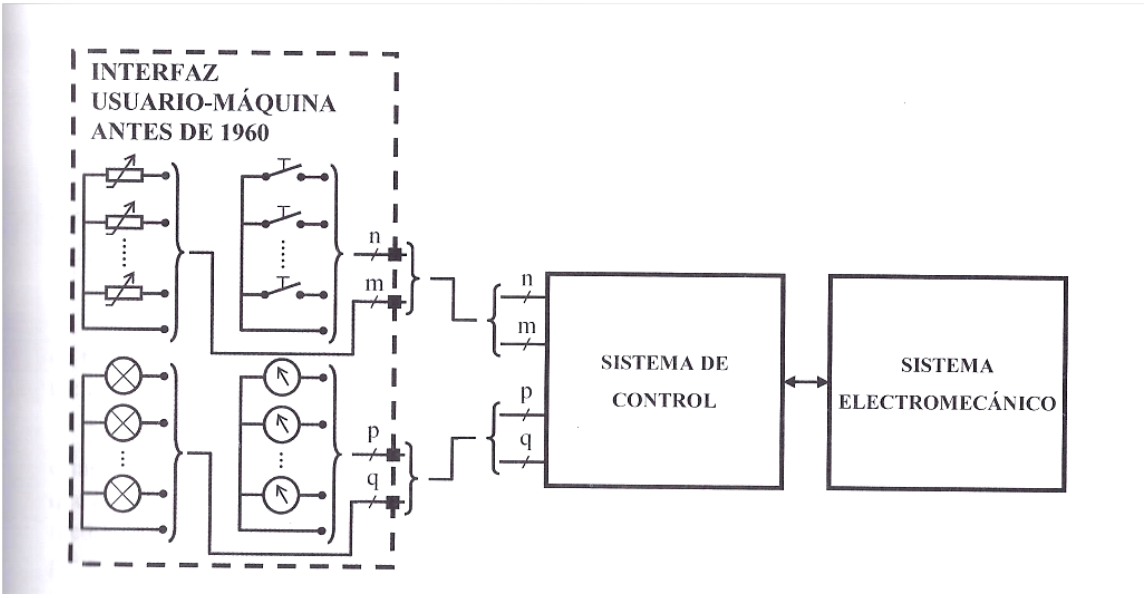


Figura 2.6 Diagrama de bloques de una unidad HMI anterior a 1960.

El desarrollo de los transistores hizo en la década de 1960 se utilizaran puertas lógicas con transistores, para sustituir a los relés en los sistemas de control lógico, y controladores *PID* (*Proportional-Integral-Derivative Proporcional Integral Derivativo)* con amplificadores operacionales, para sustituir a los implementados con válvulas en los sistemas de control continuo.

La sustitución de los relés por transistores hizo que se tuviesen que utilizar indicadores luminosos para conocer si una determinada salida estaba en nivel cero o en uno, lo cual en los circuitos implementados con relés se comprobaba viendo los contactos.

Esta fue una de las razones que impulsó la investigación básica para conocer las propiedades de emisión de luz de ciertos compuestos de semiconductores, como por ejemplo, el arseniuro de galio, lo que trajo como resultado el desarrollo de los diodos luminiscentes, conocidos con el acrónimo *LED* (*Light Emitting Diode, Diodo Emisor de Luz*), y su incorporación como dispositivos de visualización de las variables de salida de los primeros controladores lógicos electrónicos digitales cableados.

Además, el progreso de la electrónica de estado sólido no solo hizo cambiar la forma de construir lo sistemas electrónicos de control, sino que mejoró las prestaciones y redujo del costo de las computadoras de aplicación general.

2.9.1 Características de los equipos HMI

Los equipos de visualización y actuación reciben actualmente, tal como se indica en el apartado anterior, la denominación de interfaz máquina-usuario y se les suele conocer por el acrónimo *HMI* (*Human Machine Interface, Interface Hombre Máquina*).

Los paneles de operación, conocidos por las siglas *OP* (*Operation Panel, Panel de Operación*), están formados por una pantalla gráfica y un conjunto de pulsadores de membrana asociados, que constituyen un teclado. Ambos se controlan mediante un procesador especializado a través de la correspondiente interfaz y un ejemplo de panel de operación. Tal como se muestra en la figura 2.7.

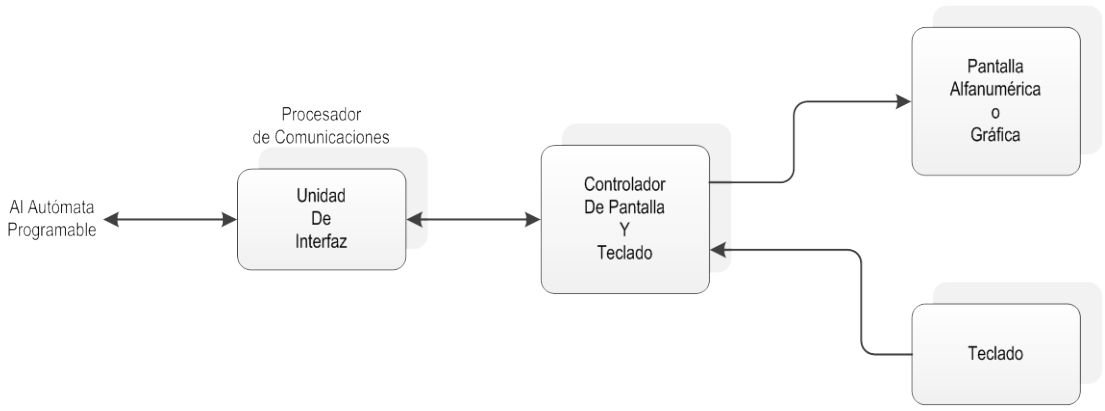


Figura 2.7 Diagrama de bloque de un panel de operación.

Los paneles o pantallas táctiles, conocidas por las siglas *TP* (Touch *Panel, Pantalla Táctil*), poseen elementos sensores sensibles al tacto. De esta forma, la pantalla realiza la función de entrada y salida, eliminado el teclado. Su diagrama de bloques se representa en la figura 2.8 y es un ejemplo de panel táctil.

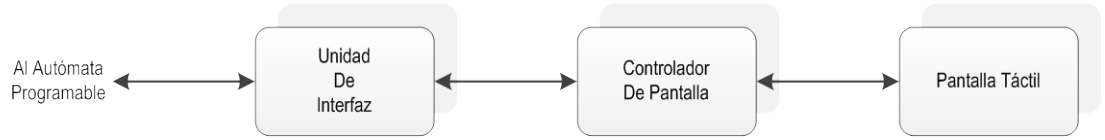
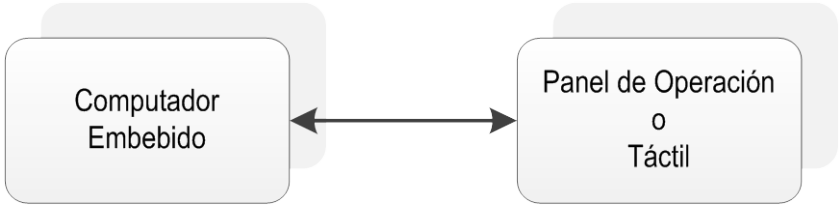


Figura 2.8 Diagrama de bloques de un panel táctil.

Existen ocasiones en la que es conveniente que el sistema electrónico de control esté embebido o empotrado en las mismas, para lo cual la unidad HMI debe ocupar el mínimo espacio posible y proporcionar más funciones que un panel de operación o un panel táctil, como por ejemplo elevada capacidad de memoria, funciones de autómata programable, funciones de supervisión y adquisición de datos conocidas como ***SCADA*** (*Supervisory Control And Data Acquisition, Supervisión, Control y Adquisición de Datos*).

A fin de entender esta necesidad, los fabricantes de sistemas de automatización comercializan equipos, cuyo diagrama de bloques se representa en la figura 2.9, que incluye en una sola carcasa una unidad HMI y un computador industrial embebido. Se pueden dividir en dos clases:



2.9 Diagrama de bloques de una unidad HMI con computador embebido

- **Paneles con computador industrial embebido y arquitectura cerrada**

Este tipo de paneles utilizan un computador embebido con un sistema físico no ampliable, un sistema operativo empotrado y recursos de programación, entre los que se puede incluir un emulador de autómata programable.

- **Paneles con computador industrial embebido y arquitectura abierta**

Para mejorar las presentaciones de los paneles de arquitectura cerrada, los fabricantes de autómatas programables utilizan un computador embebido implementado con un microprocesador de elevada capacidad de cálculo.

El desarrollo de la microelectrónica ha proporcionado que los fabricantes de unidades HMI las doten de capacidad de comunicación a través de una o más redes de control normalizadas, para facilitar así su ubicación en un lugar más adecuado de la instalación [5].

2.10 ¿Qué es un microcontrolador?

Un microcontrolador es un pequeño sistema digital programable de propósito específico, integrado en un chip especializado en el proceso de control. En su memoria sólo reside un programa destinado a controlar una aplicación determinada; sus líneas de entrada/salida soportan la conexión de sensores y dispositivos de control que permiten efectuar el proceso deseado.

Resumiendo, un microcontrolador es un microprocesador optimizado, utilizado para controlar equipos electrónicos, de sistemas de comunicación, monitoreo y adquisición de señales físicas, procesamiento y administración señales analógicas y digitales. Una vez programado y configurado el microcontrolador, solamente sirve para controlar la tarea asignada.

2.10.1 Procesador

La función básica del procesador es ejecutar la secuencia de instrucciones del programa que está almacenado en memoria. Cada procesador tiene su propio conjunto de instrucciones (código máquina) que conforma el lenguaje con el que se escribirá el programa que se requiere ejecutar.

Cada instrucción tiene un código de operación y opcionalmente operandos de entrada (sobre los que realiza la operación) y de salida (resultado de dicha operación). Las instrucciones más comunes son las de transferencia de datos (entre el procesador y memoria o entrada/salida), aritméticas, lógicas y de salto de secuencia, figura 2.10.

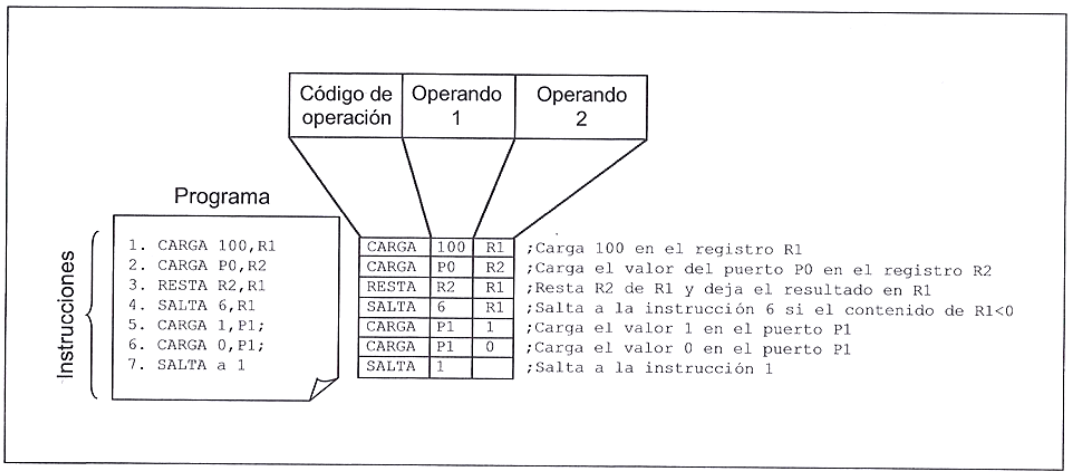


Figura 2.10 Ejemplo de programa.

2.10.2 Programación

El código máquina es idóneo para el hardware del microcontrolador pero completamente inadecuado para el usuario que tenga que programarlo. Para resolver este problema, el programador escribe su programa en un lenguaje más apropiado a su forma de expresarse y luego hace uso de un traductor para convertirlo al código máquina del microcontrolador; se usa fundamentalmente el lenguaje ensamblador y el lenguaje C.

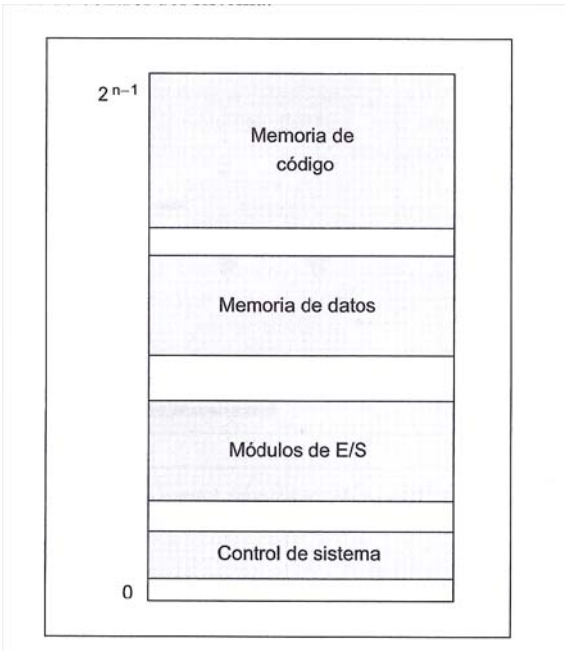
El lenguaje ensamblador proporciona un lenguaje simbólico del código máquina, aunque no deja de ser un lenguaje de bajo nivel (tedioso de programar) y especifico de un microcontrolador.

El lenguaje C es un lenguaje de más alto nivel, menos eficiente que el lenguaje ensamblador pero más fácil de programar e independiente del procesador utilizado.

En ocasiones los microcontroladores hacen uso de sistemas operativos en tiempo real *RTOS (Real Time Operating System, Sistema Operativo de Tiempo Real*). RTOS es un conjunto de funciones de bajo nivel que ejecutan tareas rutinarias del sistema y que liberan al programador de aplicaciones de tener que programar. Sus características de tiempo real se refieren a que garantizan su respuesta en los plazos de tiempo programados.

2.10.3 Espacio de direcciones

Se define como el conjunto de direcciones de memoria y entrada/salida a las que se puede acceder. Los microcontroladores pueden tener uno o varios espacios de direcciones. La figura 2.11 muestra un ejemplo de mapa de memoria de un microcontrolador con un único espacio de direcciones, donde hay zonas asignadas para cada elemento del microcontrolador: memoria de código y memoria de datos, módulos entrada/salida y registros de control de sistema.



2.11 Ejemplo de espacio de direcciones de un microcontrolador.

La zona de memoria de código almacena las instrucciones y los datos de solo lectura. Mientras que la memoria de datos guarda las variables del programa, el en resto se almacenan los registros de configuración, estado y control de los módulos entrada/salida y de control de sistema.

Para acceder a un espacio de 2^{n-1} direcciones se requiere un bus de direcciones de *n* bits.

2.10.4 Excepciones

Hay ocasiones en las que se producen ciertos eventos denominados excepciones que requieren, en muchos casos de forma inmediata, del procesador y que alteran la ejecución normal del programa. Las excepciones provocan la ejecución de una rutina de servicio de excepción que se ejecutan de forma casi transparente al programa principal.

Pueden ser internas o externas al procesador. Las excepciones internas se provocan por errores en la ejecución de instrucciones o en el acceso de memoria, por no tener derechos de acceso a ciertos recursos del microcontrolador, por instrucciones específicas programadas por el usuario o por estar en un modo especial de depuración del código. Las externas se denominan interrupciones y son solicitudes de atención por parte de los módulos entrada/salida y de control del sistema.

2.10.5 Memoria

El elemento básico de memoria es el ***bit***, sin embargo la memoria de los microcontroladores se suele organizar en ***byte***. En la figura 2.12 se puede observar un ejemplo de una memoria de tamaño 2^{n-1} bytes, donde cada byte tiene su propia dirección (0,1, 2…2^{n-1}). El bus de direcciones de la memoria deberá tener *n* bits para poder acceder a todos los bytes, el tamaño de la memoria se suele medir en bytes o múltiplos de bytes.

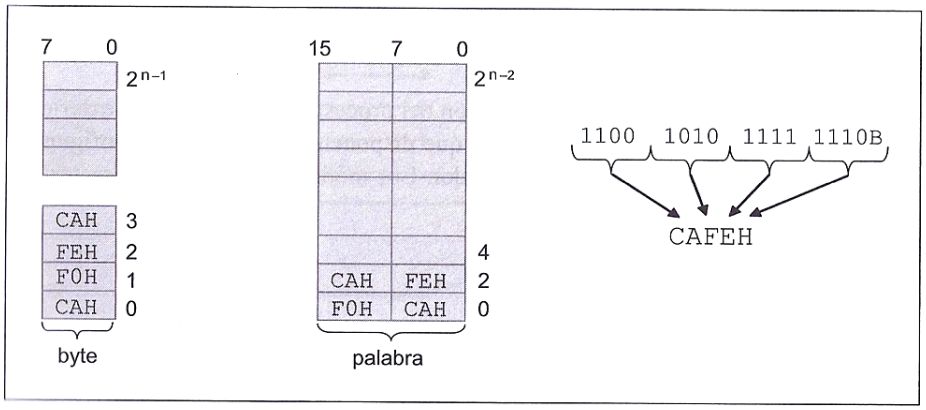


Figura 2.12 Ejemplo de organización de memoria.

Por ejemplo:

- 1 Kbyte = 2^{10} bytes
- 1 Mbyte = 2^{20} bytes
- 1 Gbyte = 2^{30} bytes

No obstante, los microcontroladores también pueden acceder directamente a bloques de bits más grandes:

- Word (2 bytes consecutivos)
- Long Word (4 bytes consecutivos)

En este caso, la dirección de estos bloques es la de uno de sus byte. Se dice que es una organización ***little-endian*** cuando la dirección del bloque de bits es la de su byte menos significativo, por lo contrario se dice que es ***big-endian*** cuando la dirección del bloque de bits es la de su byte más significativo.

También es común la posibilidad de acceder directamente a los bits de algunas zonas de la memoria de datos o de los registros de configuración, estado y de control de algunos módulos entrada/salida, en este caso cada bit tiene su propia dirección.

2.10.6 Tipos de memorias

En los microcontroladores se usan memorias no volátiles para almacenar instrucciones y datos de sólo lectura y volátiles para variables del programa. La memoria ***ROM*** conserva la información almacenada aunque se retire la alimentación del sistema.

- *EPROM*
- *EEPROM*
- *FLASH*

Actualmente la memoria flash es la más utilizada ya que aventaja a todas las anteriores en velocidad, costo, consumo y tensión de programación, su único inconveniente es que el borrado se hace por bloques y no byte a byte.

Las memorias volátiles pierden la información almacenada cuando se retira la alimentación; son muy rápidas de lectura y escritura. Se denominan memorias ***RAM*** ya que son de acceso aleatorio, los microcontroladores suelen integrar memorias ***SRAM***.

2.10.7 Módulos entrada/salida

Permiten la comunicación del procesador con el conjunto exterior, estos módulos suelen ser programables, por lo que disponen de registro de configuración, estado y control a los que accede el procesador.

Los módulos de entrada/salida más característicos de un microcontrolador son:

- Puertos entrada/salida
- Comunicación serial
- Contadores
- Moduladores de Ancho de Pulso
- Interfaz Analógica

2.10.8 Control de sistema

El control de sistema lleva a cabo varias funciones:

- Reinicialización (Reset) del sistema
- Control de interrupciones
- Interfaz de control y depurado
- Protección del sistema
- Seguridad del sistema

Los microcontroladores actuales implementan varios modos de operación relacionados con la seguridad del sistema, al ahorro de energía y el desarrollo de la aplicación. En el modo seguro el microcontrolador impide el acceso no autorizado a sus recursos, en los modos ahorro energético reduce su consumo disminuyendo la tensión de alimentación y/o frecuencia de reloj de sus diferentes módulos. Finalmente, en modo depurado permite el acceso a todos sus recursos mientras está corriendo el programa, esto es de gran utilidad en las fases de desarrollo del sistema. La reinicialización del sistema es un mecanismo drástico pero imprescindible en los microcontroladores. Su función es la de llevar al sistema a una situación inicial con condiciones conocidas de funcionamiento. Esto es necesario, hay ocasiones en las que el microcontrolador cae en estados indeseables de los que no se puede recuperar por otros medios.

Los microcontroladores suelen incluir mecanismos de protección contra malos funcionamientos del sistema. Por ejemplo, un comportamiento anómalo del programa, una caída de tensión, un acceso a zonas inválidas de memoria, etc.

Un típico mecanismo de protección es el ***watchdog timer,*** también llamado contador ***COP*** (*Computer Operating Properly, Computador Operando Apropiadamente*), que reinicializa al microcontrolador, al detectar si el programa no sigue el curso establecido.

Los microcontroladores también disponen mecanismos de seguridad, evitando accesos no autorizados a la memoria y a los registros del microcontrolador. En la memoria flash, donde se ubica el código, se pueden proteger los sectores más sensibles y permitir el acceso externo sólo si se conoce una clave de seguridad.

2.10.9 Arquitectura básica de microcontroladores

- Arquitectura CISC (Von Neumann)

La arquitectura CISC (*Complex Instruction Set Computer, Computador con Conjunto de Instrucciones Complejas)* es una estructura interna de diseño de procesadores en la que el bus de datos está compartido con el bus de direcciones.

Esta arquitectura permite tener instrucciones más elaboradas que realizan operaciones aritméticas más complejas*.* Sin embargo, por tener un solo bus compartido, el tiempo de ejecución se puede ver un poco alterado. Por su parte, existe un único mapa de datos continuo para los datos del programa (código software) y para los datos de almacenamiento temporal RAM (datos del programa), que permite direccionar de igual forma y mediante las misma instrucciones los datos de la memoria Flash y RAM, incluso permitir la ejecución del programa en la memoria de almacenamiento figura 2.13.

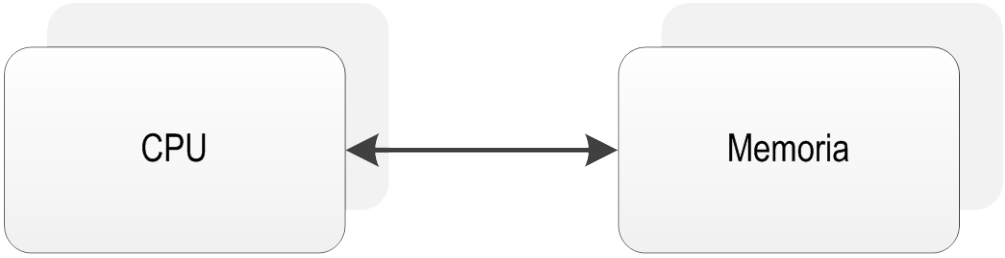


Figura 2.13 Arquitectura Von Neumann

- Arquitectura RISC (Harvard)

RISC (*Reduced Instruction Set Computer, Computador con Conjunto de Instrucciones Reducidas)* Esta arquitectura contiene buses separados para la codificación de instrucciones y datos, y es denominada arquitectura Harvard. Este doble bus permite la gran mayoría de instrucciones sean ejecutadas en un solo ciclo máquina. El procesador puede direccionar los datos de forma directa o indirecta, figura 2.14.

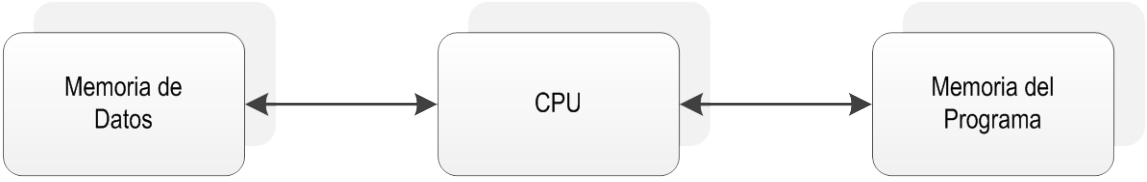


Figura 2.14 Arquitectura Harvard

En los microprocesadores de arquitectura Harvard cuando una instrucción está siendo ejecutado, la siguiente es extraída a la memoria de programa, con lo cual, se mejora su desempeño.

2.11 Microcontrolador vs microprocesador

Un microcontrolador difiere de una microprocesador normal debido a es más fácil convertirla en un computador en funcionamiento, con un mínimo de chips externos de apoyo. Un microprocesador tradicional no le permitirá hacer esto, ya que espera que todas estas tareas sean manejadas por otros chips.

Un microcontrolador típico tendrá un generador de reloj integrado y una pequeña cantidad de memoria RAM y ROM/EPROM/EEPROM, lo que significa que para hacerlo funcionar se necesita solamente unos pocos programas de control y un cristal de sincronización. Los microcontroladores disponen generalmente también de una gran variedad de dispositivos de entrada/salida, como convertidores de analógico a digital, temporizadores, UART, y buses de interfaz serie especializados como ***I^2C*** y ***CAN***.

2.12 ¿Qué es un sistema embebido?

Se conoce como sistema embebido a un circuito electrónico computarizado que está diseñado para cumplir una labor especifica en un producto. La inteligencia artificial, secuencias y algoritmos de un sistema embebido se encuentran dentro de la memoria de una pequeña computadora denominada microcontrolador. A diferencia de los sistemas computacionales de oficina y laptops, estos solucionan un problema específico y están dispersos en todos los ambientes posibles de la vida cotidiana. Es común encontrar circuitos embebidos en los vehículos, por ejemplo, controlando el sistema de inyección de combustible, en los sistemas de frenado ABS, en el control de espejos, sistemas de protección contra impactos (*Airbag*), alarmas contra robo, sistemas de ubicación, entre otros. También en los electrodomésticos de uso diario: controlando la temperatura en refrigeradores, estufas, hornos microondas y planchas, el motor de licuadoras, lavadoras de ropa, lavaplatos, aspiradoras y juguetes, en los equipos celulares, agendas de bolsillo, ***PDA***, cajeros automáticos, cámaras fotográficas, reproductores de música MP3 y video, equipos de gimnasio, equipo médico y una gran cantidad de dispositivos de uso diario.

Se sabe que un consumidor promedio interactúa con alrededor de 400 microcontroladores por día. Este número tiende a crecer significativamente para los próximos años, considerando que los procesadores son cada vez más pequeños, consumen menos energía y el precio es menor gracias a la economía de escala aplicada en su fabricación, aspectos que ayudan a reemplazar en mayor proporción los sistemas lógicos, los equipos electromecánicos y, en el futuro, se podrán incorporar en equipos desechables, figura 2.15.

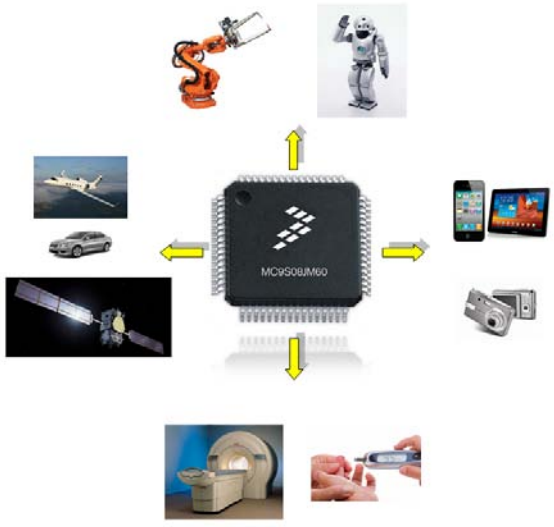


Figura 2.15. Campo de aplicación de los microcontroladores.

CAPÍTULO 3: IMPLEMENTACIÓN DE PRÁCTICAS PARA DISEÑO MECATRÓNICO

3.1 Práctica Número 1:

"Conceptos del Microcontrolador MC9S08JM60 (Freescale) y su Entorno de Programación CodeWarrior"

Objetivo

Comprender los elementos fundamentales del microcontrolador y su entorno de programación.

Introducción

La situación actual en el campo de los microcontroladores se ha producido gracias al desarrollo de la tecnología de fabricación de los circuitos integrados. Este desarrollo ha permitido construir las centenas de miles de transistores en un chip, esto fue una condición previa para la fabricación de un microprocesador. Las primeras microcomputadoras se fabricaron al añadirles periféricos externos, tales como: memoria, líneas de entrada/salida, temporizadores y otros.

El incremento posterior de la densidad de integración permitió crear un circuito integrado que contenía tanto al procesador como periféricos. Así es cómo fue desarrollada la primera microcomputadora en un solo chip, denominada más tarde microcontrolador.

Los principiantes en electrónica creen que un microcontrolador es igual a un microprocesador. Esto no es cierto, difieren uno del otro en muchos sentidos. La primera y la más importante diferencia es su funcionalidad, para utilizar al microprocesador en una aplicación real se debe de conectar con componentes, tales como memoria o componentes y buses de transmisión de datos.

Aunque el microprocesador se considera una máquina de computación poderosa, no está preparado para la comunicación con los dispositivos periféricos que se le conectan. Para que se comunique con algún periférico, se deben utilizar los circuitos especiales. Así era en el principio y esta práctica sigue vigente en la actualidad.

Funcionamiento

1.-Explicación teórica de las características fundamentales de la tarjeta DEMOJM de FREESCALE, figura 3.1.

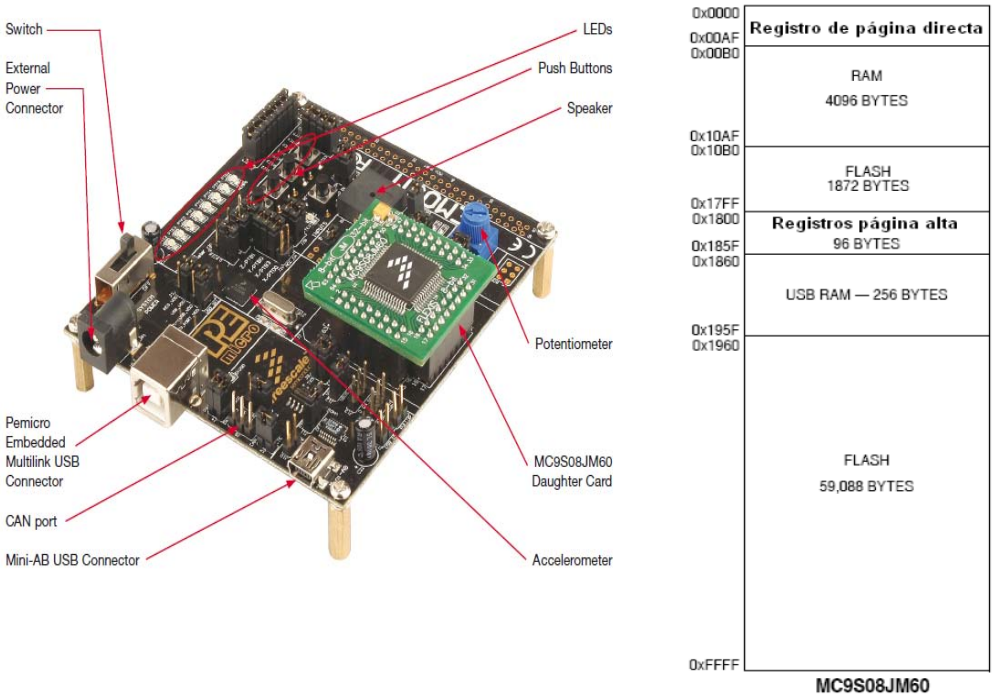


Figura 3.1 Tarjeta DEMOJM y mapa de memoria del MC9S08JM60.

2.- Ejecute los siguientes pasos para realizar el primer programa en C (se debe de tener el programa CodeWarrior instalado en la pc):

Para iniciar un proyecto nuevo en el *CodeWarrior,* teniendo en cuenta que utilizaremos un entorno *Windows,* debe abrirse el programa a través de un acceso directo o siguiendo la siguiente ruta:

Inicio > Todos los programas > Freescale Codewarrior > CodeWarrior Development Studio for Microcontrollers V6.3 > CodeWarrior IDE.

Lo que se muestra en la figura 3.2.

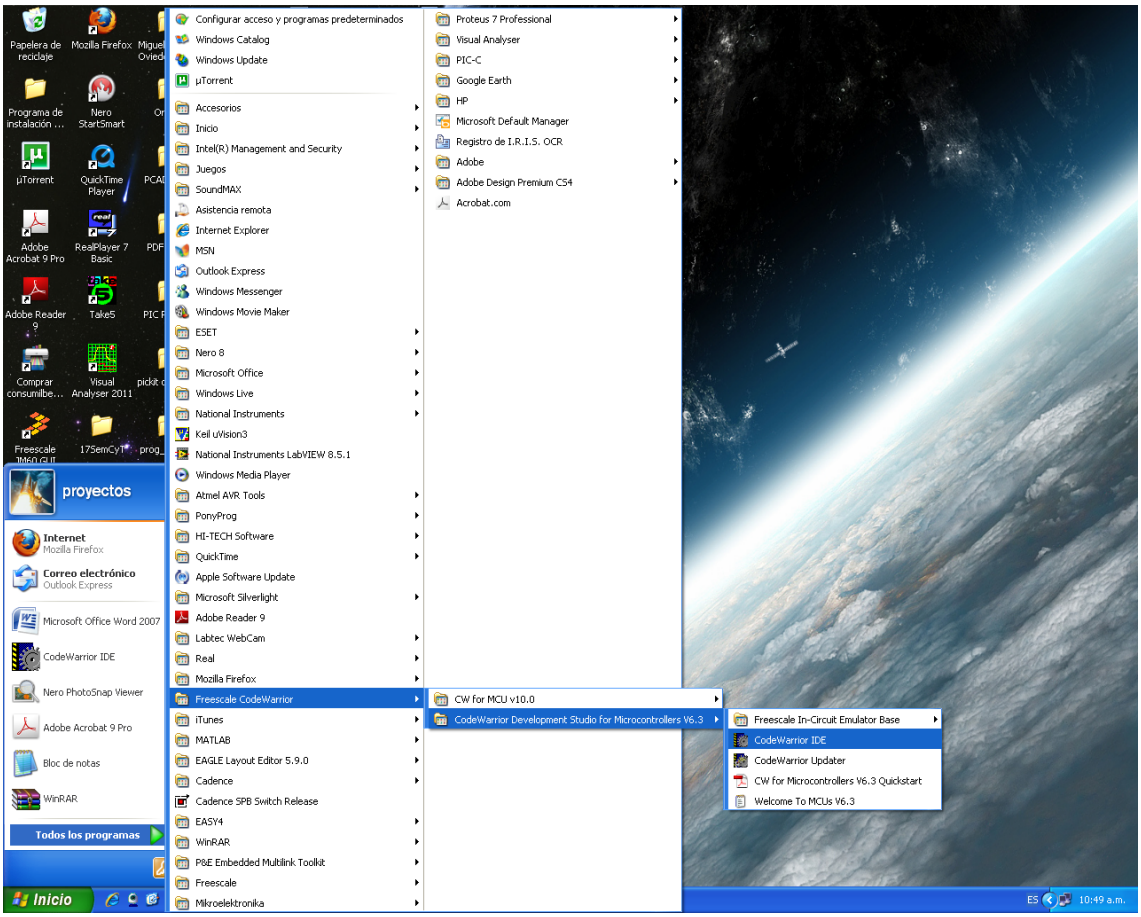


Figura 3.2 Ruta para abrir el CodeWarrior IDE.

Una vez hecho esto, aparecerá la pantalla en la figura 3.3.

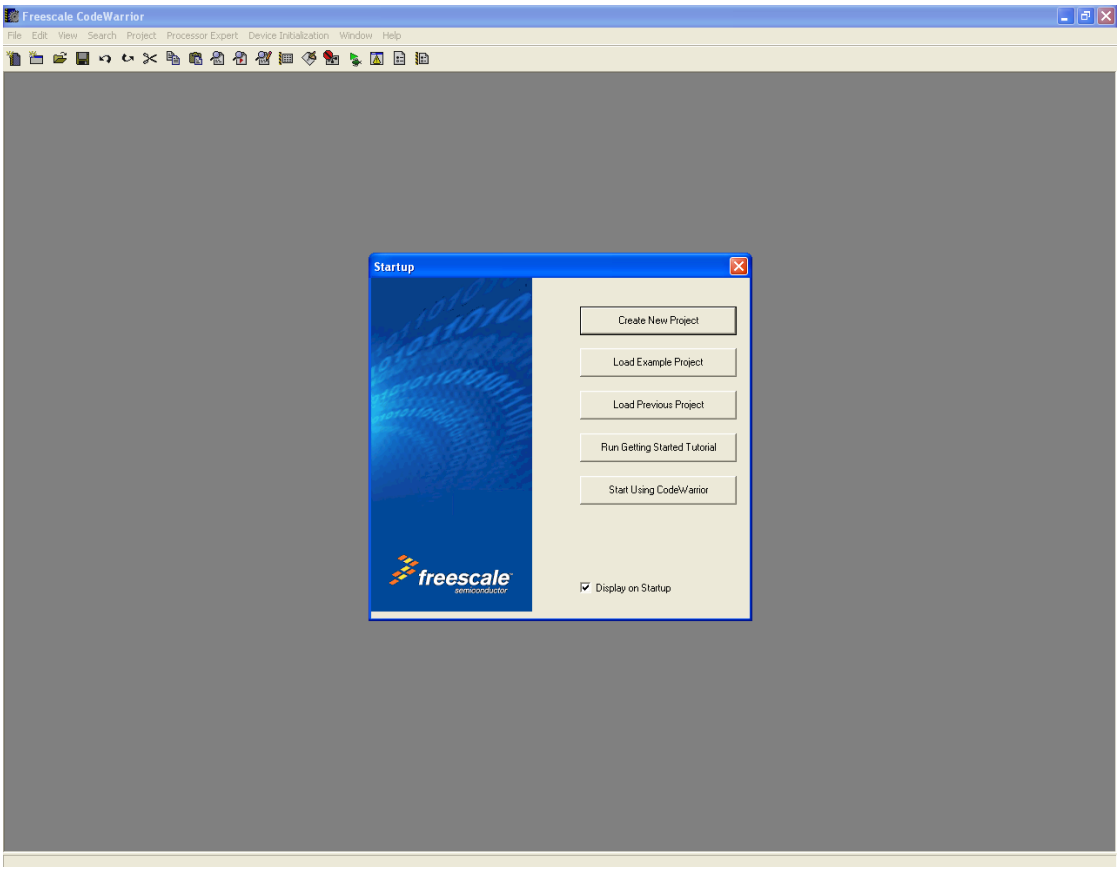


Figura 3.3 Ventana inicial del CodeWarrior IDE.

Ya que ha seleccionado la opción ***"Create New Project o Crear Nuevo Proyecto",*** aparecerá un nuevo cuadro figura 3.4. Aquí podrá seleccionar el tipo de arquitectura que desea utilizar, las cual son: *HC08, HCS08, RS08, ColdFire V1,* y *Flexis.* Desglose el árbol para encontrar el microcontrolador que empleará, en este caso ***MC9S08JM60***, este se encuentra en el grupo de los pertenecientes a la arquitectura ***HCS08*** en la familia ***JM***.

También podrá seleccionar el tipo de conexión que quiera utilizar, como lo son: ***Full Chip Simulation*** o ***P&E Multilink/Cyclone Pro***. Los otros dos tipos de conexión no interesan.

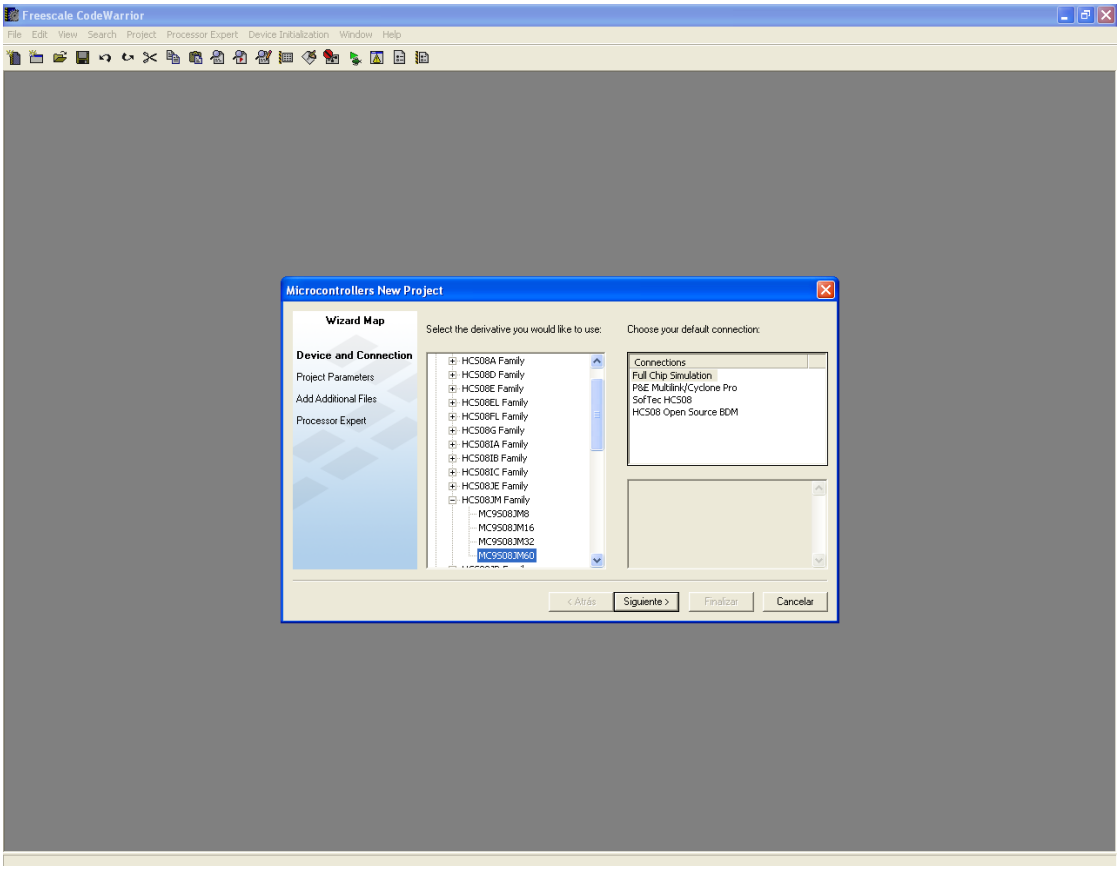


Figura 3.4 Selección del microcontrolador y tipo de conexión.

Full Chip Simulation:

En este tipo 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:

Para este tipo, el código máquina 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 pins del microcontrolador se realizan de la misma forma.

Después de presionar el botón *siguiente* en el cuadro anterior, aparecerán las opciones correspondientes ***parámetros del proyecto,*** mostrado en la figura 3.5. Aquí podrá usted seleccionar el tipo de lenguaje con el que desea programar así como el nombre del proyecto y el lugar donde será guardado el archivo, en este caso el proyecto se escribirá en ***lenguaje C*** y se nombrara ***Proyecto_1,*** por default la opción de lenguaje C se encuentra activa.

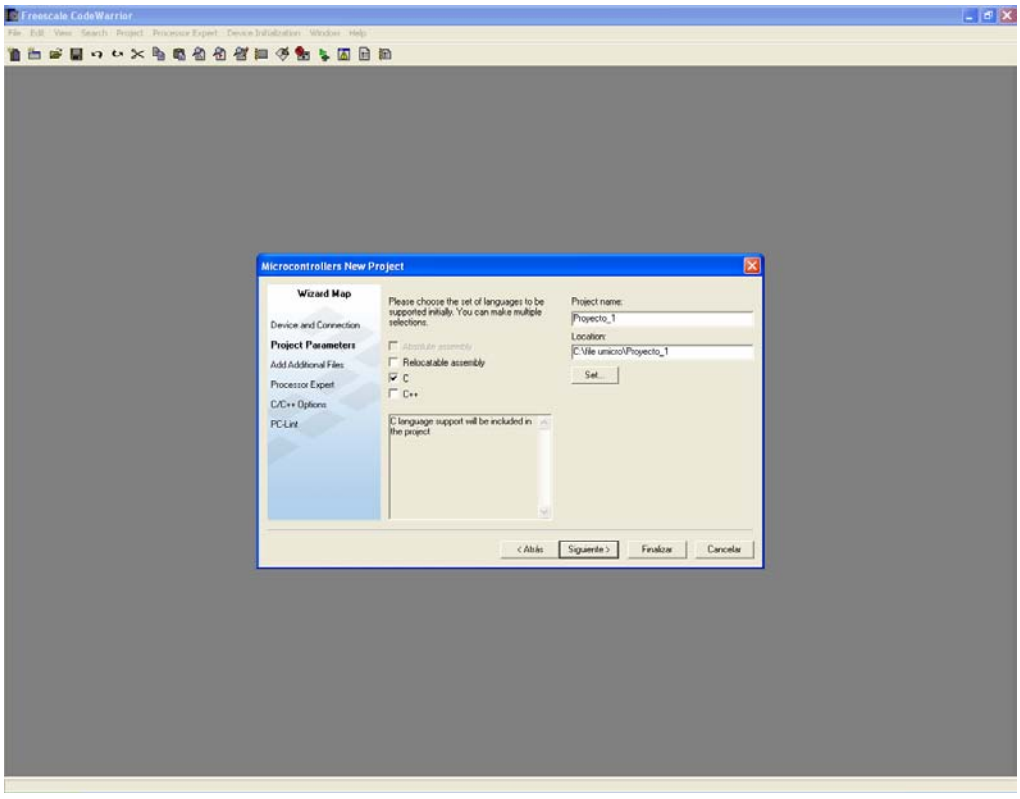


Figura 3.5 Selección de parámetros del proyecto.

Después de presionar el botón *siguiente* aparecerá el cuadro mostrado en la figura 3.6, con el cual podremos adicionar a nuestro proyecto programas existentes o subrutinas ya escritas. El botón ***Add*** es para adjuntar el archivo seleccionado en el árbol izquierdo y el botón ***Remove*** sirve para remover el archivo.

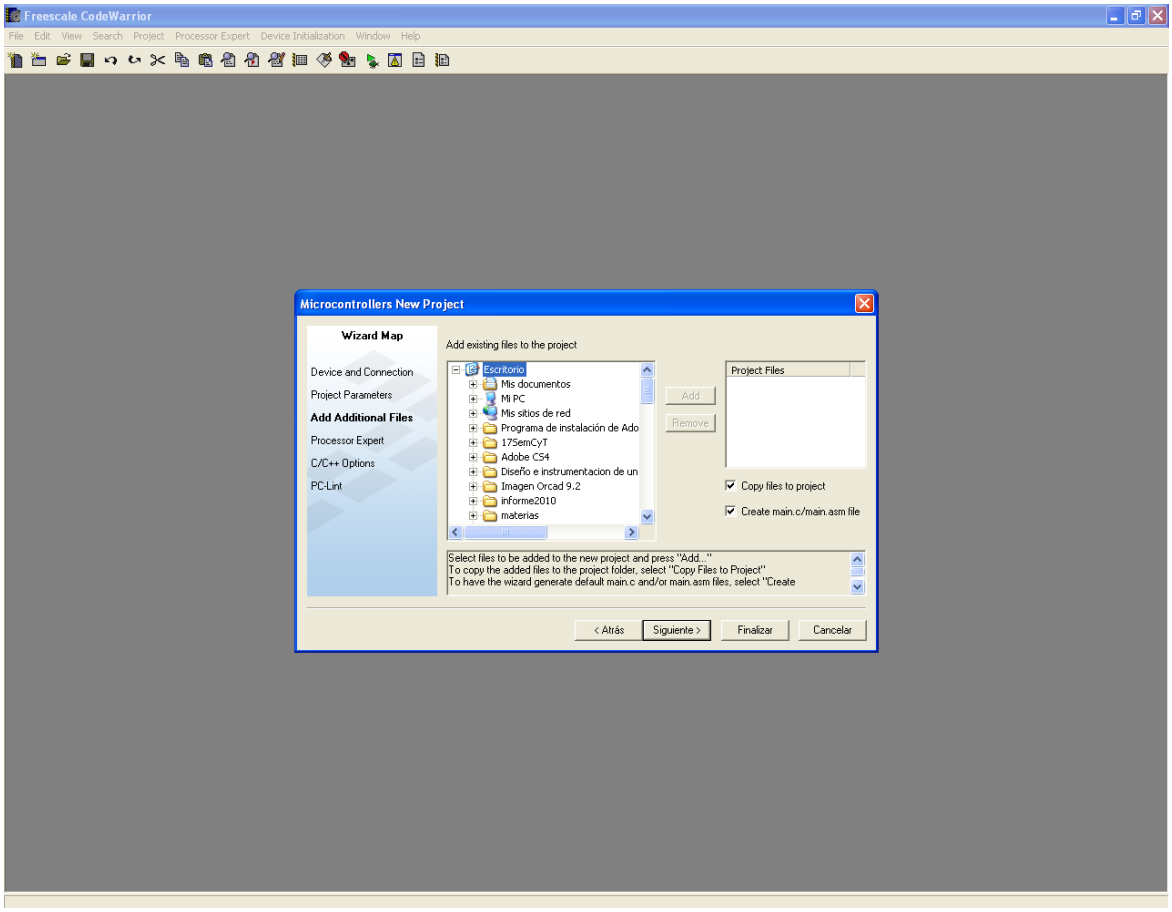


Figura 3.6 Adjuntar archivos adicionales.

Presione el botón ***siguiente*** para que aparezca el cuadro mostrado en la figura 3.7, donde podrá seleccionar ***None "para realizar una programación pura en lenguaje C"***, y posteriormente presionar el botón ***Finalizar*** para concluir la creación de un nuevo proyecto con el asistente ***CodeWarrior***.

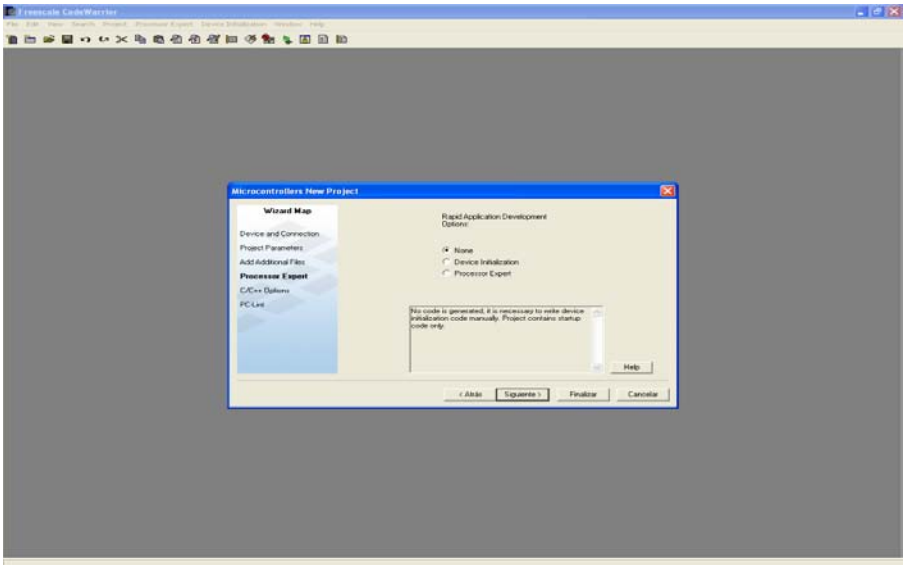


Figura 3.7 Activación del Processor Expert.

Ya inicializado un nuevo proyecto se abrirá la ventana mostrada en la figura 3.8, que será el entorno donde trabajará la mayor parte del tiempo.

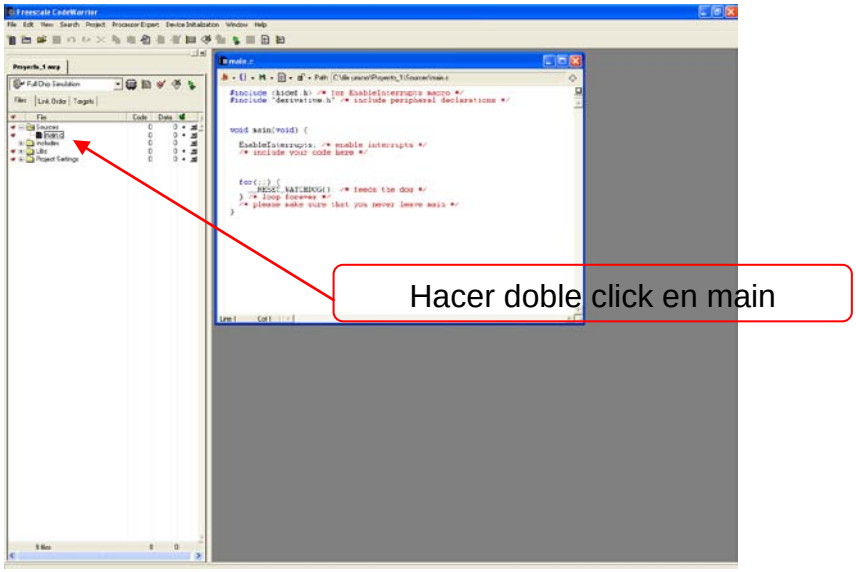


Figura 3.8 Ventana principal del CodeWarrior.

Desarrollo

3. Una vez abierta y maximizada la ventana de trabajo, se introducirá el siguiente código que realiza el encendido y apagado del led PF1 de la DEMOJM con una demora de tiempo, para poder observar la transición de encendido y apagado.

```
/*****************************************************************************************

Programa: Encender y apagar el led PTF1 de la tarjeta DEMOJM con el
microcontrolador MC9S08JM60 de 8 bits.

Fecha: Agosto del 2012

Autor: Oscar M. Espinoza

Version: 1.0

*****************************************************************************************/

#include <hidef.h>                  /* Macro para habilitar interrupciones */
#include "derivative.h"             /* Incluye declaración de periféricos */


#define LED_ON()  PTFD_PTFD1 = 0; PTFDD_PTFDD1 = 1  //Macro para enceder
                                                    //led
#define LED_OFF() PTFD_PTFD1 = 1; PTFDD_PTFDD1 = 1  //Macro para apagar
                                                     //led

#define Disable_COP() SOPT1 &= 0x3F             //Deshabilita el COP


/*****************************************************************************************/

//FUNCION DE RETARDO POR SOFTWARE

void Delay(void){

 unsigned int i;                    //Declaración de variable local tipo entero

   i = 40000;                       //Asigna a la variable i el valor de 40,000
```

```
   while(i > 0){                    //Mientras la variable i sea mayor

    i--;                            //Decrementa a la variable

   }

}
/*****************************************************************************************/

void main(void) {                   //Principal

  MCGC1 = 0x04;                     //Reloj en modo FEI y divisor por 1
  MCGC2 = 0x00,

  Disable_COP();                    //Deshabilita el COP

  EnableInterrupts;                 //Habilita interrupciones


  for(;;) {                         //Ciclo infinito

   LED_ON();                        //Enciende el led
   Delay();                         //Llama a la función de retardo
   LED_OFF();                       //Apaga el led
   Delay();                         //Llama a la función de retardo

  }

}
```

Se muestra el diagrama de flujo para encender y apagar un led con demora de tiempo por software:

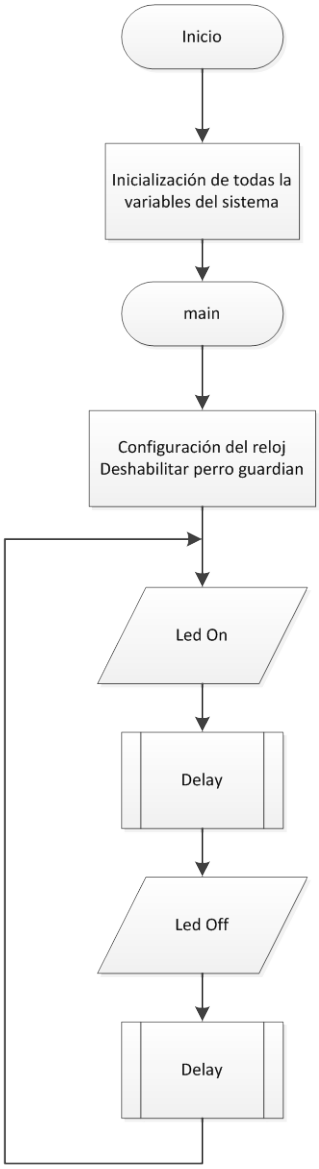


Figura 3.9 Diagrama de flujo de rutina para encender y apagar un led.

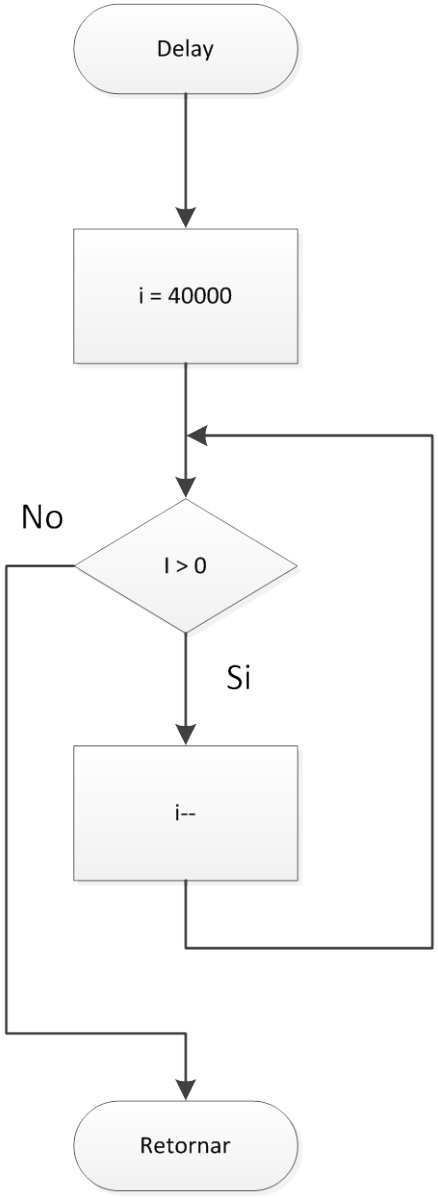


Figura 3.10 Diagrama de flujo para realizar retardos por medio de software.

4. Una vez escrito el programa, se compila como se muestra en la figura 3.10 Este programa funciona de forma correcta.

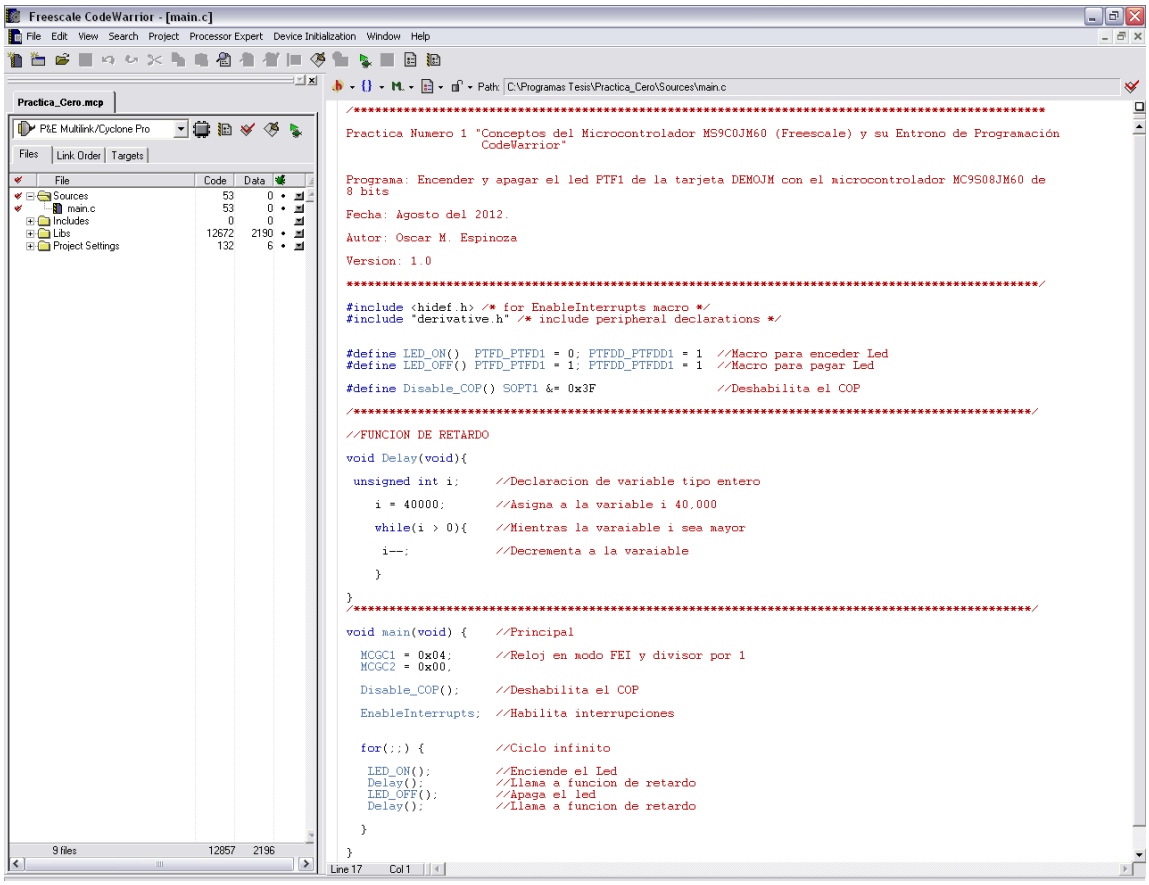


Figura 3.10 Ventana del CodeWarrior con el programa.

Si existen errores, nuevamente revisar la sintaxis del programa.

5. Para proceder a la depuración es necesario establecer la conexión de la Figura 3.11. La PC deberá instalar automáticamente el hardware del DEMOJM y, una vez reconocido, seguir los pasos que se detallan a continuación:

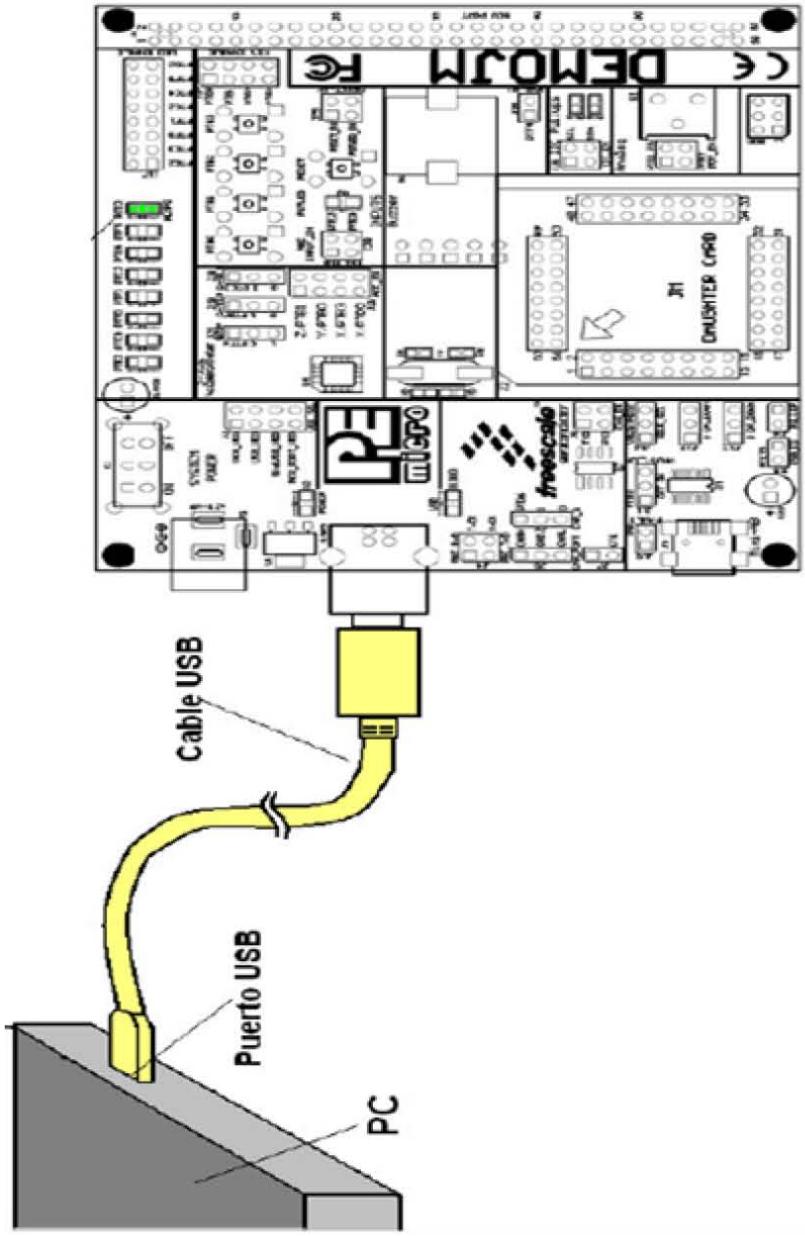


Figura 3.11 Conexión de la DEMOJM a la PC por medio del cable USB.

6. Transferir el programa que realizamos hacia la memoria del microcontrolador es de la siguiente manera, figura 3.12.

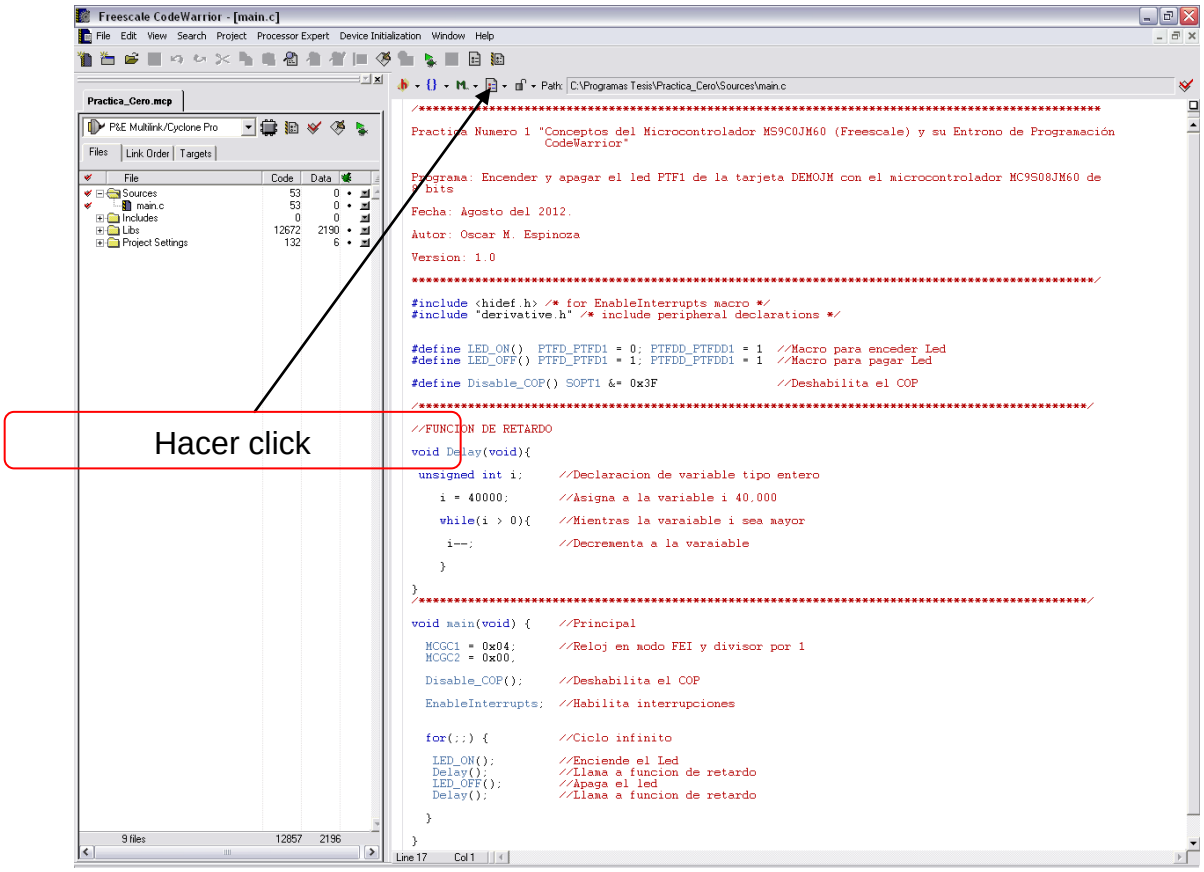


Figura 3.12 Icono Debug para transferir el programa al microcontrolador.

Despues de presionar el icono del ***Debug*** aparecerá la siguiente ventana, anteriormente a este paso se debe conectar la DEMOJM a la PC y encender la tarjeta.

7. Seleccionar ***Connect(Reset),*** figura 3.13.

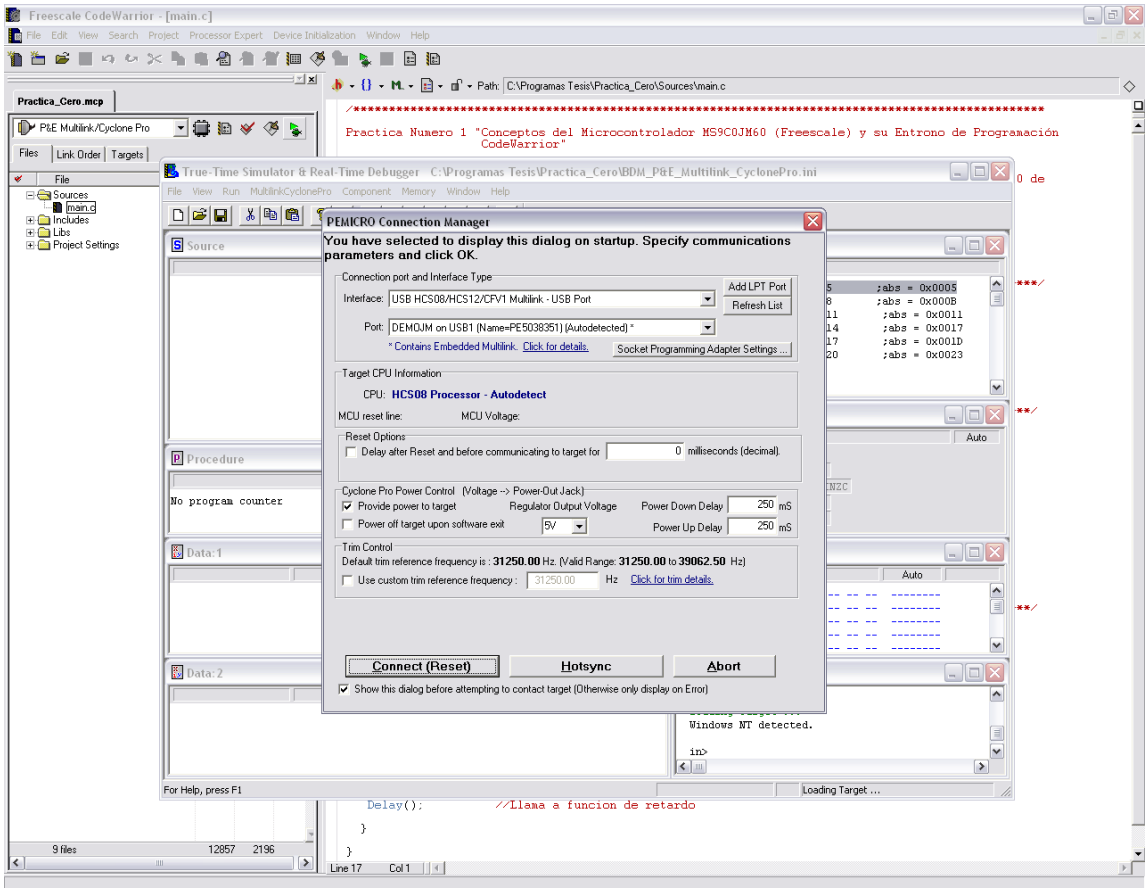


Figura 3.13 Ventana de que indica la comunicación, PC a la tarjeta DEMOJM.

8. Presionar ***Yes***, figura 3.14.

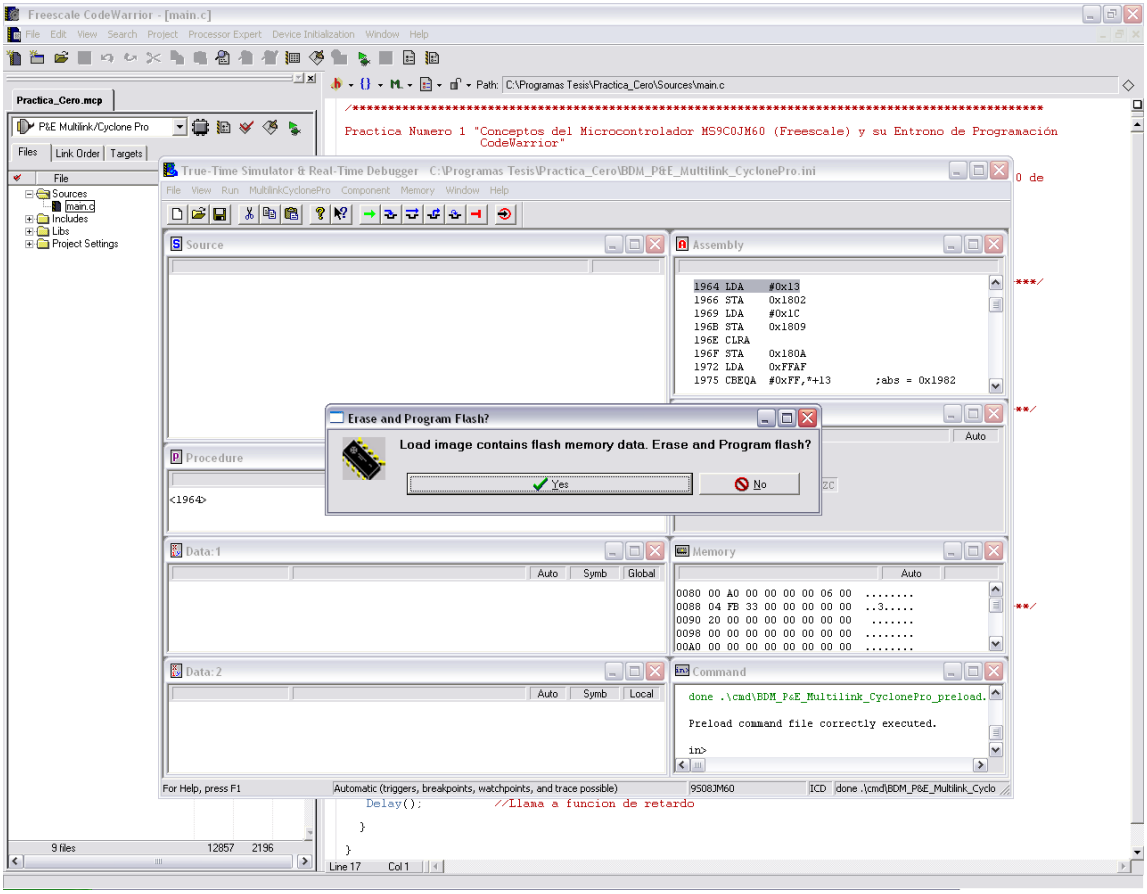


Figura 3.14 Programa transferido a la memoria del microcontrolador.

9. Al dar clic en ***Start*** el programa se ejecutará en tiempo real, observar el comportamiento, figura 3.15.

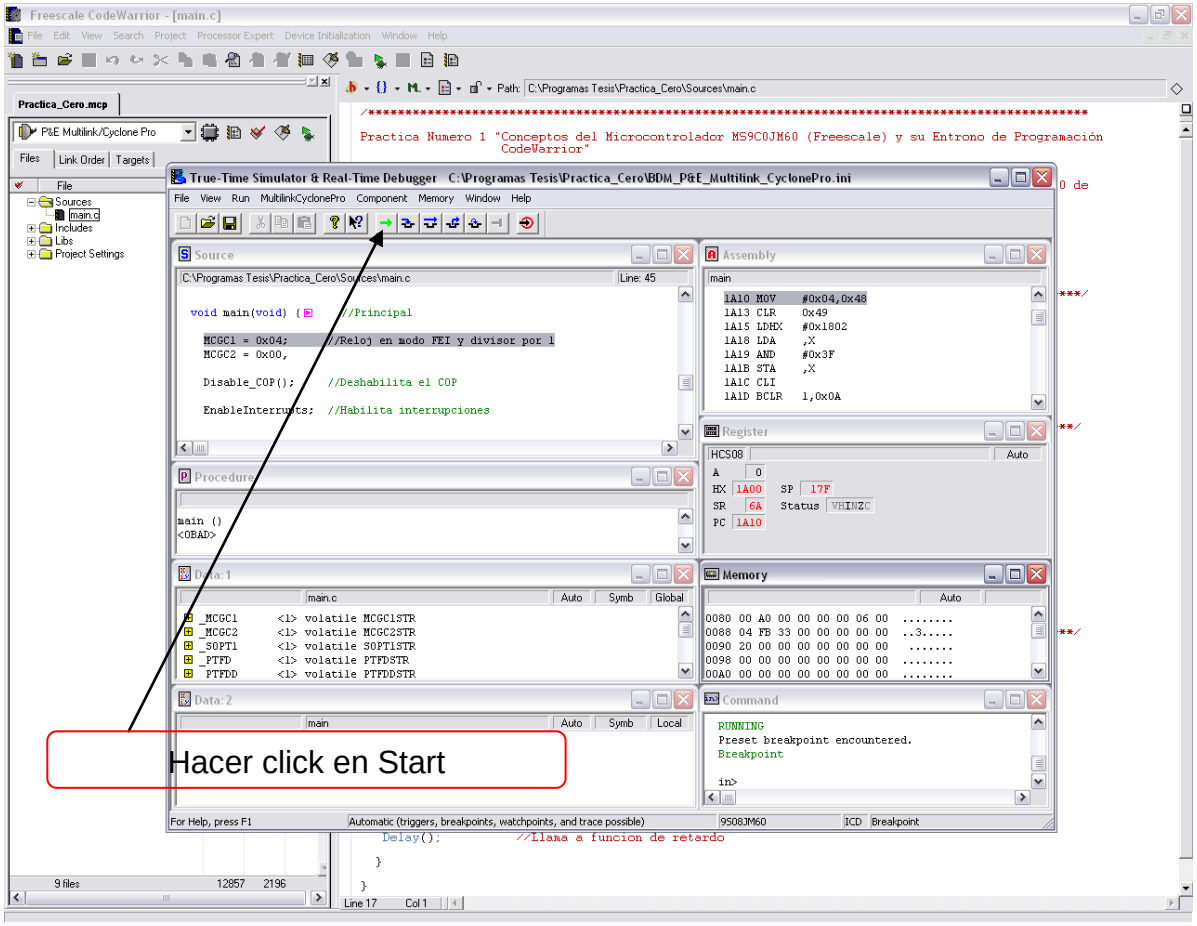


Figura 3.15 Para iniciar, la ejecución del programa en tiempo real en la memoria del microcontrolador.

El menú de botones para la depuración es el siguiente:

 **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 detenido en cualquier momento mediante el botón “**halt**”. 

 **Step-In:** Realiza la ejecución de una línea simple en el programa. 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 misma corresponde a un llamado de función, esta será invocada y su ejecución no será intervenida. La función se ejecuta en tiempo real y la computadora detiene su ejecución al regresar de la línea invocada.

 **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.

 **Single-Step**: Ejecuta la instrucción en ensamblador 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 ensamblador 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 a actualizar las subventanas del depurador.

Reset: Genera un restablecimiento 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).

Material y equipo para la práctica

1 Multímetro.
1 PC.
1 Tarjeta DEMOJM60.
1 Microcontrolador MC9S08JM60.

Cuestionario Preliminar

1.- Enuncie que es un microcontrolador.

2.- Enuncie que es un sistema embebido.

3.- Cuáles son los tipos de arquitectura con los que cuentan los microcontroladores y realice un esquema básico de cada una de ellas.

4.- Mencione que es el lenguaje máquina y el lenguaje ensamblador.

5.- Qué es un lenguaje de alto nivel, mencione algunos ejemplos.

6. - Qué significa: Bit, Byte, WordDouble y Word.

7.- ¿Qué es una memoria?

8.- ¿Qué es una memoria ROM, RAM, Flash?

9.- Enuncie que es el contador de programa (PC).

10.-Enuncie los registros con los que cuenta el microcontrolador MC9S08JM60.

3.2 Práctica Número 2:

"Configuración de los Puertos de E/S de la DEMOJM"

Objetivo

Comprender y aplicar la configuración de los puertos E/S del MC9S0JM60.

Introducción

Los puertos del microcontrolador son el punto de comunicación entre el microcontrolador y el mundo exterior, a través de ellos se puede efectuar procesos de control electrónico sobre dispositivos de potencia, instrumentación, telemetría, etc. Además, permiten también recibir señales del mundo exterior, como por ejemplo: señales provenientes de transductores, amplificadores, transmisores, actuadores, etc.

¿Qué son los pins (terminales) de entrada?

En los microcontroladores se usan dispositivos de entrada muy sencillos como, por ejemplo, interruptores simples, debido a que la mayoría de entradas pueden solamente procesar señales digitales con los mismos niveles de tensión que la fuente de alimentación.

Inicialmente se debe tener en cuenta lo siguiente:

- El nivel de cero volts o nivel de tierra se denomina $V_{ss.}$
- El nivel positivo o nivel de alimentación se denomina V_{dd}, cuyo valor por defecto son 5 volts de corriente continua.

Se sabe por circuitos digitales que un nivel de tensión de (0 - 0.8 Volts) se considera como "0" lógico y una tensión entre 2.5 y 5 Volts, se considera como "1" lógico.

Sin embargo, no todas las señales que se aplican al microcontrolador deben ser señales de tipo digital. En el mundo real existen señales analógicas, o señales que son de otros niveles de tensión. Algunos dispositivos tiene la propiedad de acondicionar las señales presentes en el medio a niveles de tensión dentro del

rango permitido para el microcontrolador. Por otra parte, existen otros dispositivos con la propiedad de convertir señales analógicas, las cuales el microcontrolador será capaz de procesar y manipular.

¿Qué son los pins de salida?

Así como se mencionó antes, que las entradas permiten recibir señales en la mayoría de los casos en formato digital, los dispositivos de salida permiten que el microcontrolador envíe información al mundo exterior, o bien realice acciones sobre éste. En una computadora, un dispositivo de salida puede ser el monitor, los microcontroladores se utilizan de igual modo dispositivos simples basados en mecanismos de conmutación o interruptores.

Cuando el microcontrolador envía un “0” lógico a través de alguno de los pins configurados como salida, a nivel de tensión se obtendrá externamente un valor de 0 Volts. De lo contrario, si se envía un “1” lógico a través de alguno de los pins configurados como salida, a nivel de tensión se obtendrá externamente un valor de 5 Volts.

Funcionamiento

En el microcontrolador MC9S08JM60, cuenta con 60 pins que pueden ser configurados de manera bidireccional (I/O) a través de siete puertos paralelos. Todos los pins pueden ser configurados como entrada o salida, figura 3.16.

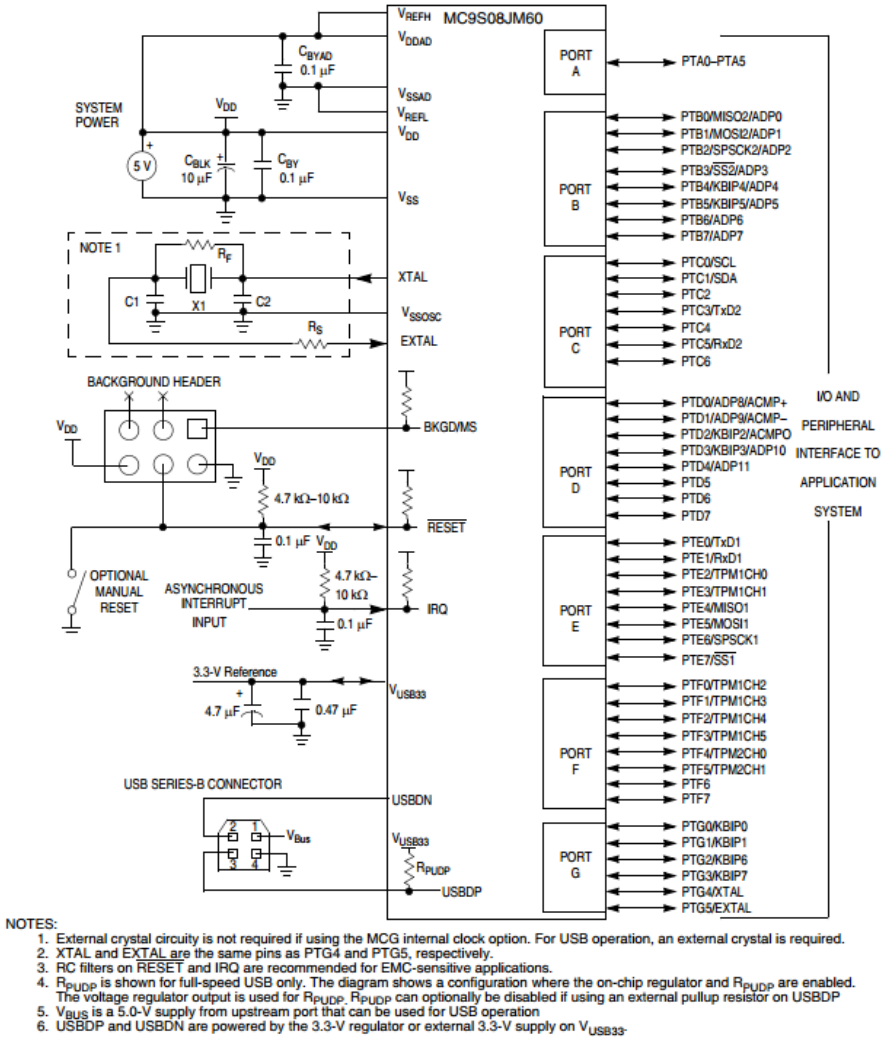
Se debe tener en cuenta que en la gran mayoría de familias de microcontroladores, los puertos de entrada/salida no solamente cumplen funciones de envío y recepción de señales digitales, sino que además comporten recursos internos, es decir, que si por un pin en especial se pueden manipular datos digitales (“1” y “0”), según la configuración I/O, también podría cumplir, por ejemplo, funciones de conversión ***A/D, PWM*** entre otras funciones.

La mayoría de los pins cumplen una doble y algunos una triple función, por ejemplo: el pin PTD3 trabaja como pin de propósito general (I/O), como una entrada de la función KBI (KBIP3) y una entrada del convertidor analógico digital ADC (ADP10).

Por defecto, cuando la máquina ha salido del estado de RESET, los pins 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 una diferencia de potencial e incurrir en daños de al microcontrolador.

NOTA: Todos aquellos pins 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 espurias*** o ruido ***EMI*** (*Electromagnetic Interference, Interferencia Electromagnética)* y/o consumos extras de corriente por pins flotantes.



NOTES:
1. External crystal circuity is not required if using the MCG internal clock option. For USB operation, an external crystal is required.
2. XTAL and EXTAL are the same pins as PTG4 and PTG5, respectively.
3. RC filters on RESET and IRQ are recommended for EMC-sensitive applications.
4. R_{PUDP} is shown for full-speed USB only. The diagram shows a configuration where the on-chip regulator and R_{PUDP} are enabled. The voltage regulator output is used for R_{PUDP}. R_{PUDP} can optionally be disabled if using an external pullup resistor on USBDP
5. V_{BUS} is a 5.0-V supply from upstream port that can be used for USB operation
6. USBDP and USBDN are powered by the 3.3-V regulator or external 3.3-V supply on V_{USB33}.

Figura 3.16 Puertos en el microcontrolador MC90S08JM60.

La figura 3.17 muestra un diagrama simplificado de la composición de un pin de un puerto.

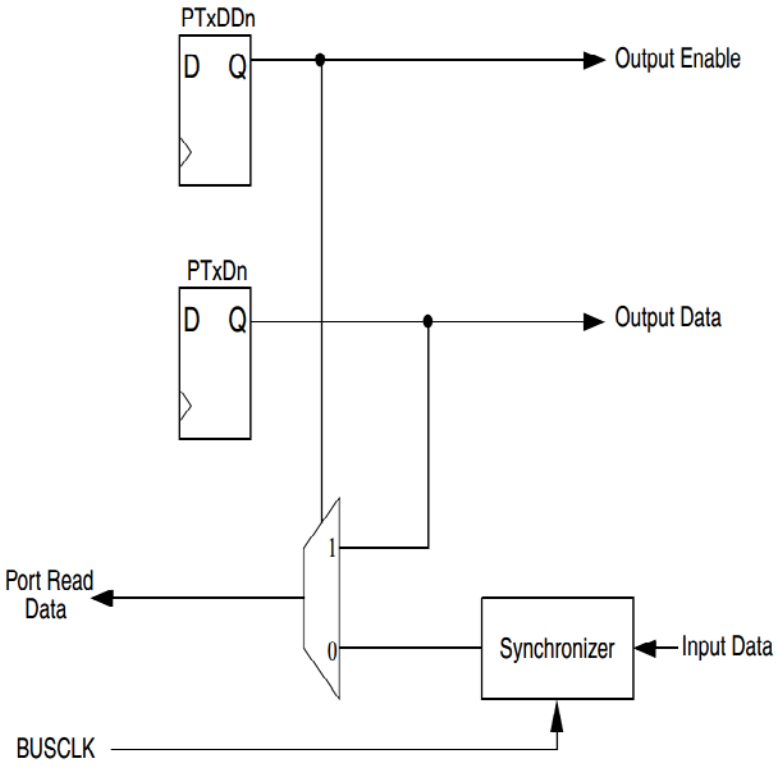


Figura 3.17 Diagrama a bloques simplificado de un pin.

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.

Las funciones básicas que se pueden definir alrededor de un puerto I/O son soportadas por los siguientes registros:

- ***Registro PTxD***

Registro de datos (PTxD): se muestra la configuración general de un registro de datos de un puerto x.

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	PTxD7	PTxD6	PTxD5	PTxD4	PTxD3	PTxD2	PTxD1	PTxD0
Reset	0	0	0	0	0	0	0	0

PTxDn: Establece el estado de cada pin I/O del correspondiente puerto. Al salir del estado de RESET, los pins son configurados como entradas en alta impedancia *pull-ups* deshabilitados. Los pins de salida se actualizan al valor de este registro, una vez se haya definido su dirección.

- ***Registro PTxDD***

Registro de dirección de pin (PTxDD): muestra la configuración general de un registro de dirección de pins de un puerto x.

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	PTxDD7	PTxDD6	PTxDD5	PTxDD4	PTxDD3	PTxDD2	PTxDD1	PTxDD0
Reset	0	0	0	0	0	0	0	0

- ***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

- ***Registro PTxPE***

Registro de habilitación de *pull-ups* (PTxPE): se muestra la configuración general de un registro de habilitación de *pull-ups* para los pins de entrada de un puerto x.

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	PTxPE7	PTxPE6	PTxPE5	PTxPE4	PTxPE3	PTxPE2	PTxPE1	PTxPE0
Reset	0	0	0	0	0	0	0	0

PTxPEn

Establece una resistencia de *pull-up* en el pin de entrada indicado.

0: El *pull-up* interno para este pin es deshabilitado (valor por defecto)
1: El *pull-up* interno para este pin es habilitado

NOTA: Los *pull-ups* son típicamente de 45KΩ

- ***Registro PTxSE***

Registro de habilitación del *slew rate* (PTxSE): se muestra la configuración general de un registro de habilitación del *slew rate* para los pins de salida de un puerto x.

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	PTxSE7	PTxSE6	PTxSE5	PTxSE4	PTxSE3	PTxSE2	PTxSE1	PTxSE0
Reset	0	0	0	0	0	0	0	0

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, Interferencia ElectroMagnetica*).

0: El *slew rate* interno para este pin es deshabilitado
1: El *slew rate* interno para este pin es habilitado (valor por defecto)

- ***Registro PTxDS***

Registro de habilitación del *drive strength* (PTxDS): se muestra la configuración general de un registro de habilitación del *drive strength* para los pins de salida de un puerto x.

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	PTxDS7	PTxDS6	PTxDS5	PTxDS4	PTxDS3	PTxDS2	PTxDS1	PTxDS0
Reset	0	0	0	0	0	0	0	0

PTxDSn

Habilita el drenaje o suministro de altas corrientes para un pin de salida. El efecto sobre la EMC (*ElectroMagnetic Compliance, Compatibilidad ElectroMagnetica, 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

NOTA: La máxima corriente que entrega el microcontrolador desde los puertos, sumadas todas las corrientes parciales por cada pin de salida, es de100mA @ 5V (60mA @ 3V). El usuario deberá tener este factor en cuenta, cuando esté utilizando la propiedad del *drive strength.*

La figura 3.18 distribución de pins (PINOUT), la mayoría de los ejercicios harán referencia de conexión sobre este conector.

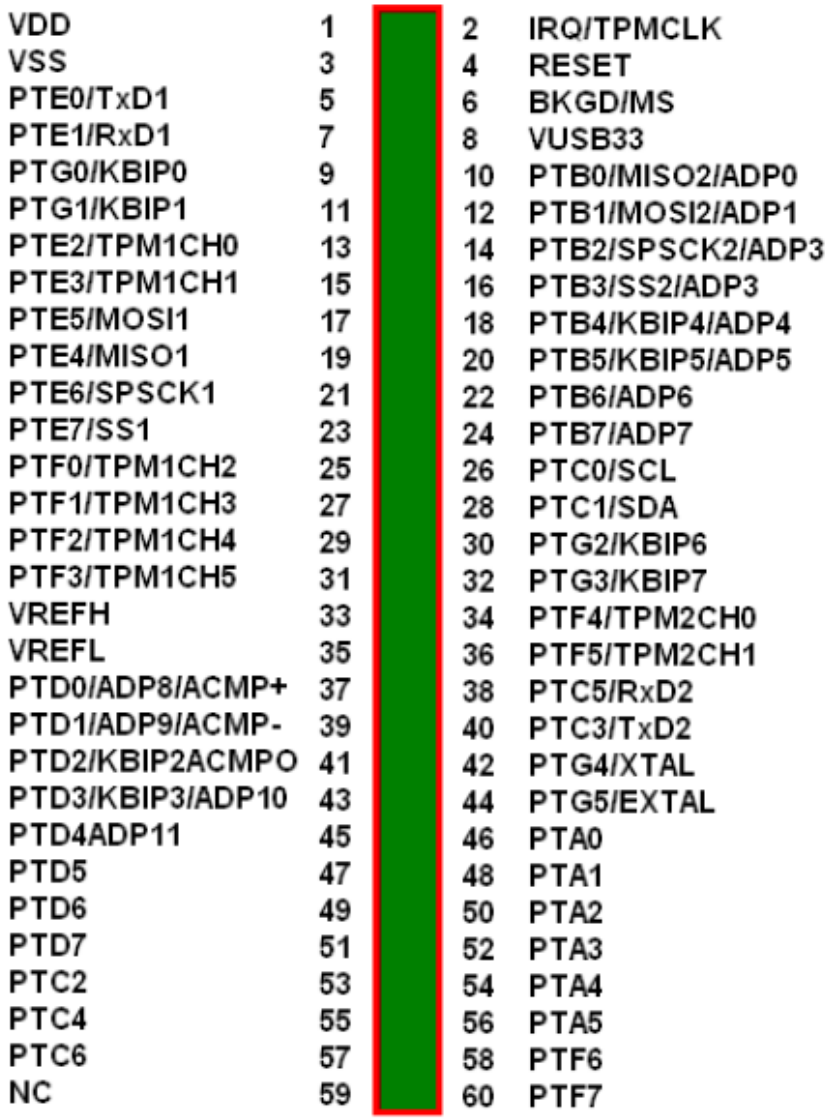


Figura 3.18 Distribución de los pin (PINOUT) en el circuito impreso de la DEMOJM.

Desarrollo

La figura 3.19 se muestra el alambrado a seguir, como ejercicio del manejo de los pins de puertos.

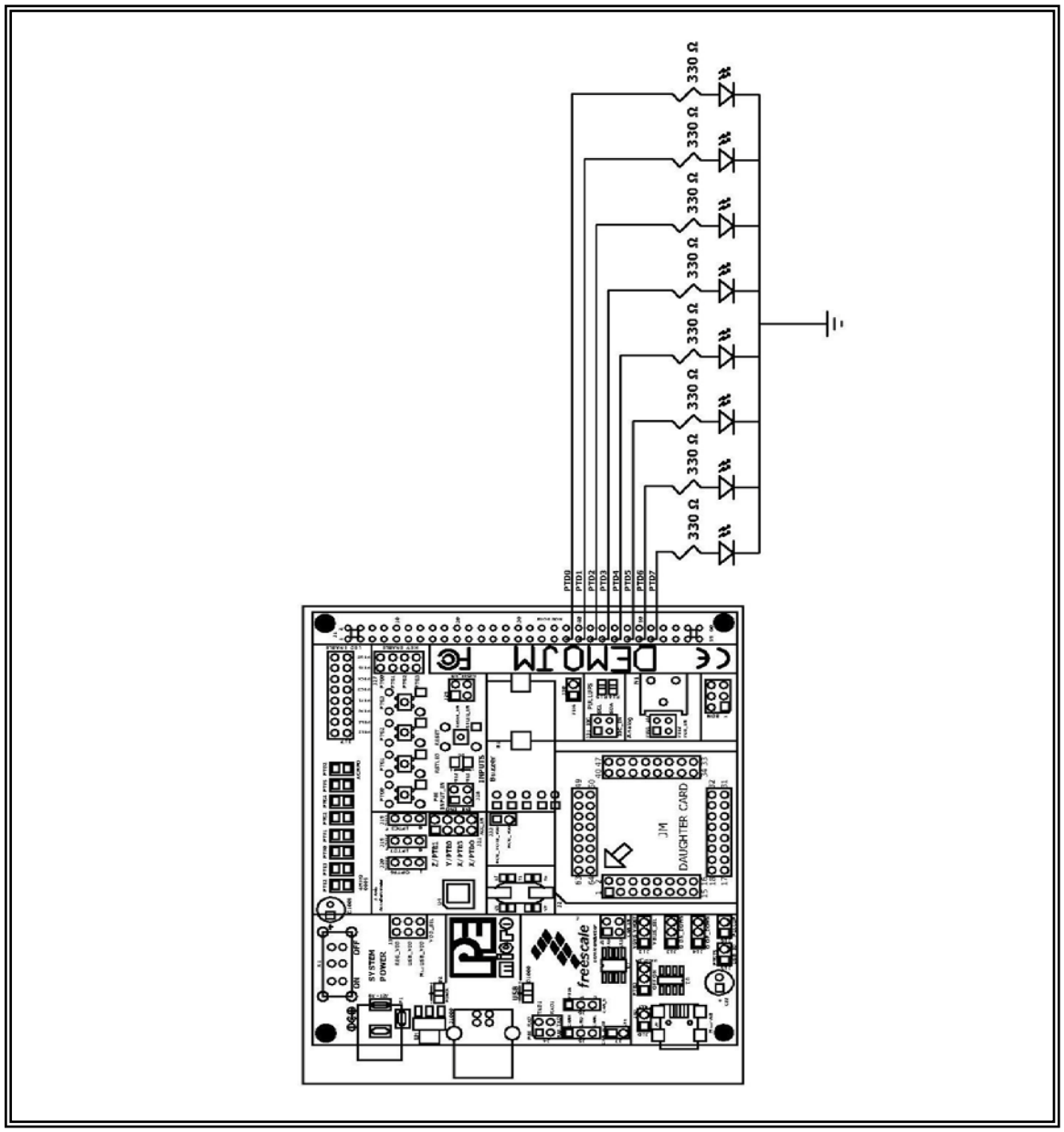


Figura 3.19 Circuito para el manejo de entrada/salida de datos.

1. Capturar el siguiente código en el IDE de CodeWarrior, que lee datos por el Puerto G (Entrada) y que sean desplegados en el Puerto D (Salida).

```
/*****************************************************************************************************
*************************/

//Programa que lee Puerto G (PTG0 a PTG5) y los datos adquiridos son mostrados en el Puerto D (PTD0 a PTD7)

//NOTA: LIBERAR EL JUMPER DE PTD1 PARA SU PERFECTO FUNCIONAMIENTO

//Agosto 19 del 2012

//Oscar M. Espinoza

/*****************************************************************************************************
*************************/

#include <hidef.h>                  /* Macro para habilitar interrupciones */
#include "derivative.h"             /* Incluye declaración de periféricos */

char PTDD_Config   = 0;
char PTDDD_Config = 0;
char PTDPE_Config = 0;
char PTDSE_Config = 0;
char PTDDS_Config = 0;

char PTGD_Config   = 0;
char PTGDD_Config = 0;
char PTGPE_Config = 0;
char PTGSE_Config = 0;
char PTGDS_Config = 0;

unsigned char i;                    //Declara variable

#define Disable_COP() SOPT1 &= 0x3F

////////////////////////////////////////////////////////////////////////////////////////////////////////////
```

```
void PTG_Init(char PTGD_Config,char PTGDD_Config,char PTGPE_Config,char
PTGSE_Config,char PTGDS_Config){

    PTGD = PTGD_Config;                    //Función que inicializa el puerto G
    PTGDD = PTGDD_Config;
    PTGPE = PTGPE_Config;
    PTGSE = PTGSE_Config;
    PTGDS = PTGDS_Config;
}

void PTD_Init(char PTDD_Config,char PTDDD_Config,char PTDPE_Config,char
PTDSE_Config,char PTDDS_Config){

    PTDD = PTDD_Config;                    //Función que inicializa el puerto D
    PTDDD = PTDDD_Config;
    PTDPE = PTDPE_Config;
    PTDSE = PTDSE_Config;
    PTDDS = PTDDS_Config;
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////

void main(void) {                          //Principal

  PTG_Init(0x00,0x00,0xFF,0x00,0x00);      //Carga los valores a los registros
  PTD_Init(0x00,0xFF,0x00,0xFF,0x00);

  Disable_COP();

 EnableInterrupts;                         /* habilita interrupciones */


  for(;;) {                                /*Ciclo infinito */

       i = PTGD;        //Lee el puerto G y deposita el contenido en la variable i
       PTDD = i;        //Lo que contenga la variable i muéstralo por el puerto D

  }

}
```

Se muestra el diagrama de flujo para la configuración de los puertos del microcontrolador:

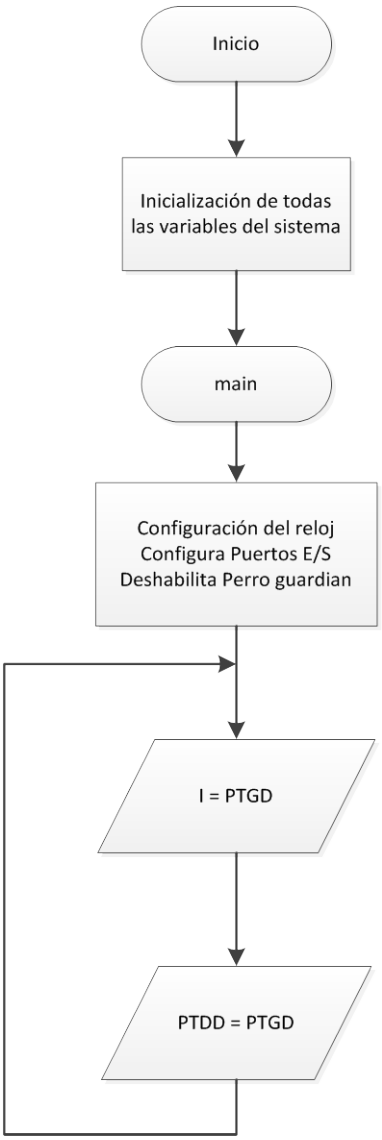


Figura 3.20 Configuración de puertos Entrada/Salida.

Material y equipo para la práctica

1 Multímetro.
1 PC.
1 Tarjeta DEMOJM60.
1 Microcontrolador MC9S08JM60.
1 DIP Switches.
4 Push Button.
8 Resistencias 330Ω.
1 Protoboard
8 Leds color rojo.
Cables.

Cuestionario Preliminar

1.- ¿Cuántos puertos E/S contiene el microcontrolador MC9S08JM60?

2.- ¿Cómo se configura un puerto como salida para el MC9S08JM60?

3.- ¿Cómo se configura un puerto como entrada para el MC9S08JM60?

4.- ¿Cómo se configuran los led E/S en la DEMOJM?

5.- ¿Qué es una resistencia de pull-up?

6.- Mencione qué función realizan los registros: PTxD, PTxDD, PTxPE, PTxSE, PTxDS.

7.- ¿Que es una transferencia en paralelo y una serial?

8.- Ventajas y desventajas del puerto paralelo.

3.3 Práctica Número 3:

"Manejo de la Interrupción Externa IRQ"

Objetivo

Comprender y aplicar el funcionamiento de la interrupción externa IRQ.

Introducción

Interrupción también conocida como interrupción de hardware o petición de interrupción es una señal recibida por el procesador de un ordenador, indicando que debe "interrumpir" el curso de ejecución actual y pasar a ejecutar código específico para tratar esta situación.

Las interrupciones, atendiendo a su duración, pueden dividirse en dos tipos claramente diferentes: excepciones que se producen a lo largo de un intervalo de tiempo, y excepciones puntuales. La forma en que se notifican estas excepciones hace que las denomine excepciones por flanco y por nivel, respectivamente:

- Excepciones por flanco: estas son puntuales, que se producen en un instante concreto y determinado. La línea asociada a estas indicará el instante en que se producen con un flanco activo (de bajada o de subida). Es importante tener en cuenta que el nivel de la línea de excepción no es relevante, solo un flanco activo marca la ocurrencia de una por flanco.

- Excepciones por nivel: estas no son puntuales, sino que se prolongan a lo largo de un intervalo de tiempo, dicho de otro modo, permanecen activas durante un cierto tiempo. A la hora de comprobar si la excepción se produce o no, se ha de considerar el nivel de la línea asociada ha dicho evento. Mientras esta línea esté activa, la excepción se estará produciendo.

El principal inconveniente de las excepciones por flanco es que, si se producen en un instante en que no pueden ser atendidas por ejemplo, porque se esté procesando otra excepción de más prioridad, requieren el uso de un hardware adicional que permita registrar dicha excepción, para procesarla posteriormente.

Las excepciones por nivel, como se producen a lo largo del intervalo de tiempo, son menos sensibles a este problema.

Funcionamiento

El registro IRQSC es el encargado de manipular la señal externa IRQ, que ocupa un lugar privilegiado dentro de los niveles y prioridades en la máquina. El procesador monitorea permanentemente la lógica del pin IRQ y puede validar tanto el flanco, y nivel de la señal presente en el pin.

La interrupción generada por el pin IRQ puede sacar a la máquina en el modo STOP, aún con el reloj suspendido. Un “0” lógico aplicado al pin de la interrupción externa produce un evento de interrupción. El pin IRQ es configurable por software, activado por flanco descendente o ascendente y nivel bajo.

Es necesario tener en cuenta que se debe atender la interrupción y salir de ella antes de que el pin de interrupción retorne a “1” lógico, siempre y cuando se tenga configurado como flanco y nivel lógico bajo.

- ***Registro IRQSC***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	0	IRQPDD	IRQEDG	IRQPE	IRQF	0	IRQIE	IRQMOD
Escritura						IRQACK		
Reset	0	0	0	0	0	0	0	0

= No implementado o reservado

IRQPDD (IRQ Pull Device Disable)

Deshabilita la resistencia de *pull-up/pull-down* asociada al pin IRQ.

IRQPDD = 0: La resistencia está habilitada.
IRQPDD = 1: La resistencia esta deshabilitada.

IRQEDG (IRQ Edge Select)

Configura el flanco que activará la interrupción externa IRQ.

IRQEDG = 0: El pin IRQ será sensible al flanco de bajada o flanco de baja y nivel bajo (de existir una resistencia asociada al pin, deberá ser de *pull-up*).

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 *pull-down*.

IRQPE (IRQ Pin Enable)

Habilita el funcionamiento del pin IRQ.

IRQPE = 0: El pin IRQ ha sido deshabilitado.
IRQPE = 1: El pin IRQ ha sido habilitado.

IRQF (IRQ bandera)

Bandera que indica que ha ocurrido un evento de interrupción externa IRQ.

IRQF = 0: No hay evento de IRQ.
IRQF = 1: Ha ocurrido un evento de IRQ.

IRQACK (IRQ Acknowledge)

Bit para el reconocimiento de un evento de IRQ.

Escribiendo un “1” en este bit, se pone a “0” la bandera IRQF. El usuario deberá poner a cero 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: bandera de IRQF con IRQMOD=0.

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.

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.
IRQMOD = 1: El pin de IRQ actúa tanto en el flanco como en el nivel de la señal eléctrica.

Desarrollo

La figura 3.21 detalla el circuito a implementar de la práctica número 3.

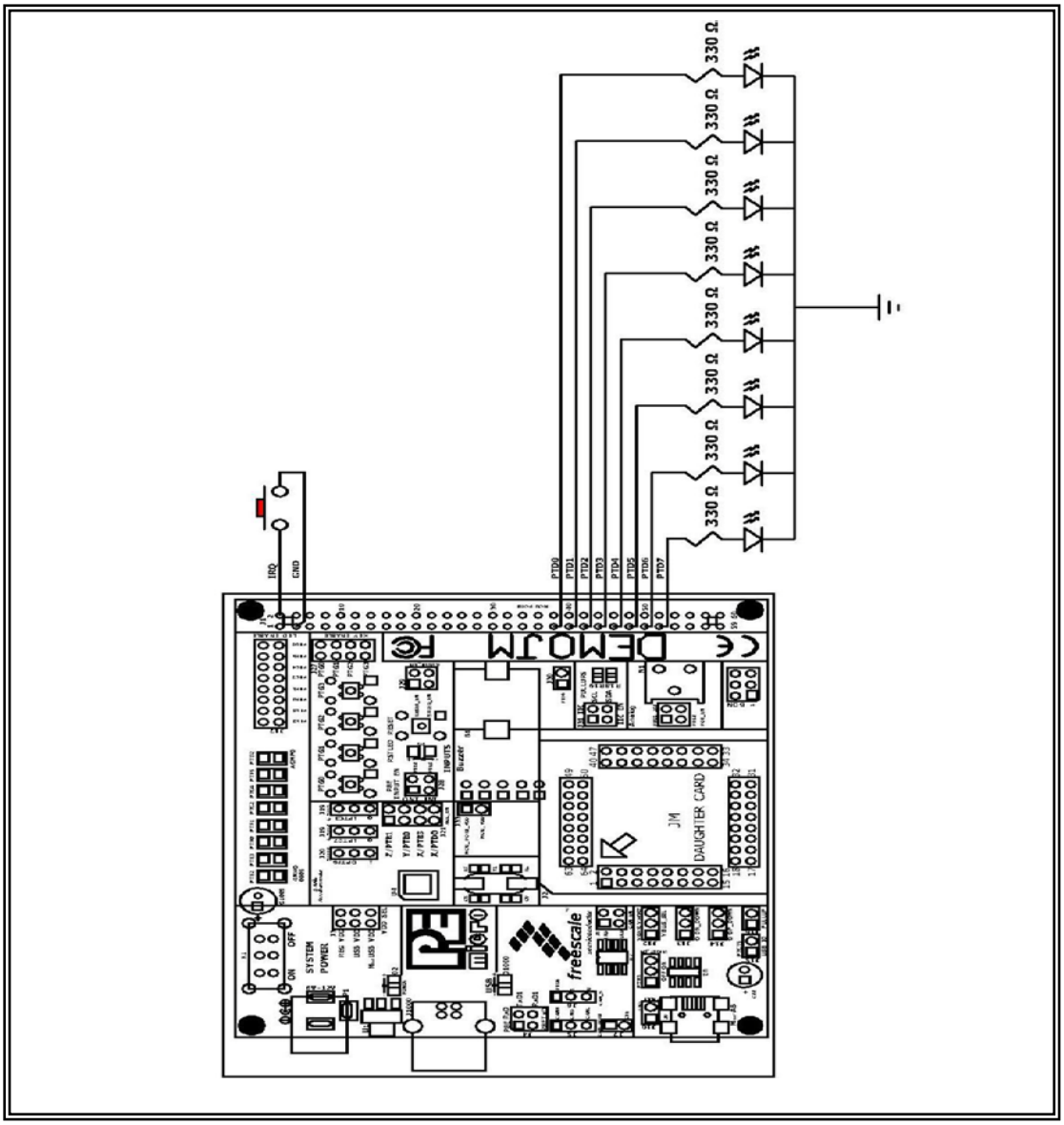


Figura 3.21 Circuito como ejercicio del manejo de la interrupción externa IRQ.

1.- Capturar el siguiente programa que cuenta del 0 al 7 en leguaje C por medio de la interrupción externa IRQ y los datos deben ser desplegados en puerto de D.

```
/*****************************************************************************************************
*********
Programa: Contador del 0 al 7 por medio de la interrupción externa IRQ.

Fecha: Agosto 19 del 2012

Autor: Oscar M. Espinoza

Versión: 1.0

*****************************************************************************************************
*********/

#include <hidef.h>                    /* Macro para habilitar interrupciones */
#include "derivative.h"               /* Incluye declaración de periféricos */

volatile unsigned char i;             //Declaración de variable

#define MAX 7                        //Macro para definir constante MAX

#define Disable_COP() SOPT1 &= 0x3F

/*****************************************************************************************************
********/

void Delay(void){                          //Rutina de retardo por Software

  unsigned int i = 40000;

    while(i > 0){

      i--;

    }

}
```

```
/*************************************************************************************************
*******/

void main(void) {

  MCGC1 = 0x04;                    //Frecuencia aproximada a 8 MHz
  MCGC2 = 0x40;
  MCGC3 = 0x01;

  Disable_COP();

  EnableInterrupts;                /* Habilita Interrupciones */


  PTDD  = 0x00;                    //Puerto D configurado  como salida
  PTDDD = 0xFF;


  IRQSC = 0x12;    //Configuración de la interrupción externa IRQ, por flanco de
subida

  for(;;) {                        //Ciclo infinito
  }
}

////////////////////////////////////////////////////////////////////////////////////////////////

interrupt VectorNumber_Virq void ISR_IRQ (void){    //Rutina de atención a la
interrupción externa IRQ

      IRQSC_IRQACK=1;          //Poner bandera a 1 de interrupción por IRQ

      Delay();                 //Llama a subrutina de retardo (rebotes)

      if(i >= MAX){            //Si i es menor a MAX entonces

       i = 0;                  //Variable a cero

      PTDD = 0;                //Puerto D a cero

      }else{                   //Sino haz

        i++;                   //Incrementa la variable i
```

```
        PTDD = i;              //Lo que contenga la variable i muéstralo por el puerto D
      }


}
```

Se muestra el diagrama de flujo para el manejo la interrupción externa IRQ:

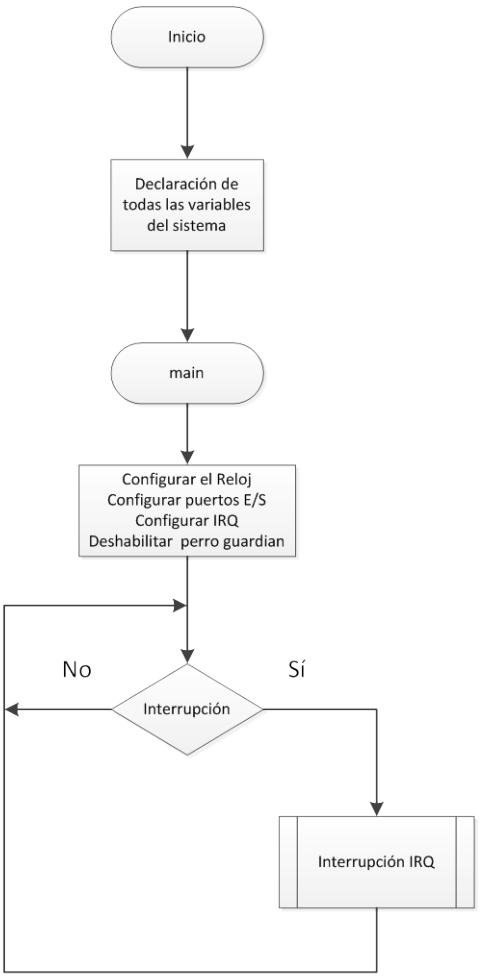


Figura 3.22 Configuración de módulos y periféricos.

Se muestra el diagrama de flujo rutina cuando es solicitada la interrupción externa IRQ:

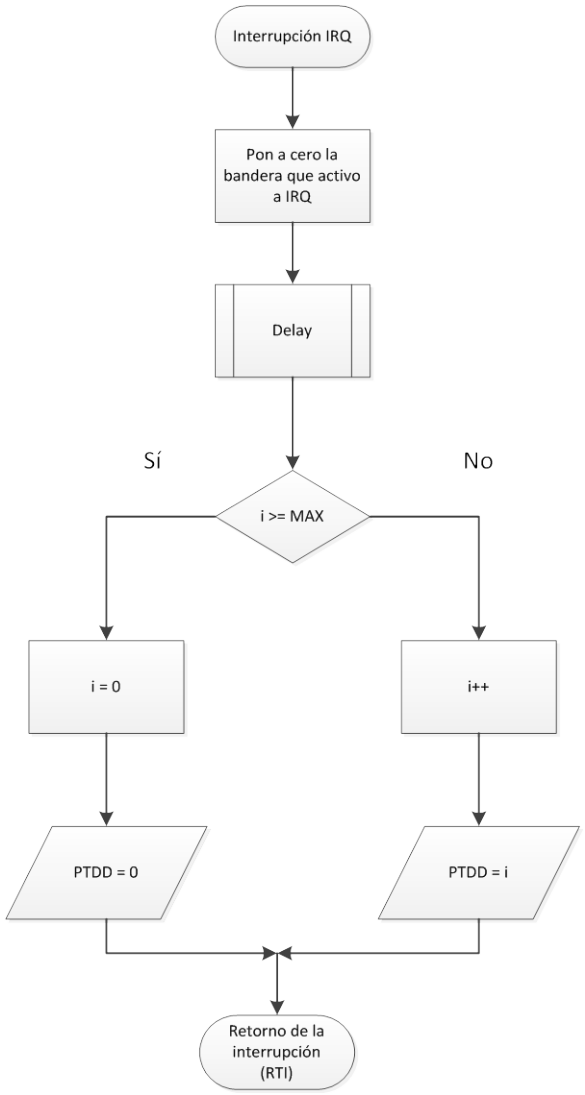


Figura 3.23 Atención a la interrupción externa IRQ.

Se muestra el diagrama de flujo cuando es solicitada la rutina de demora de tiempo por software:

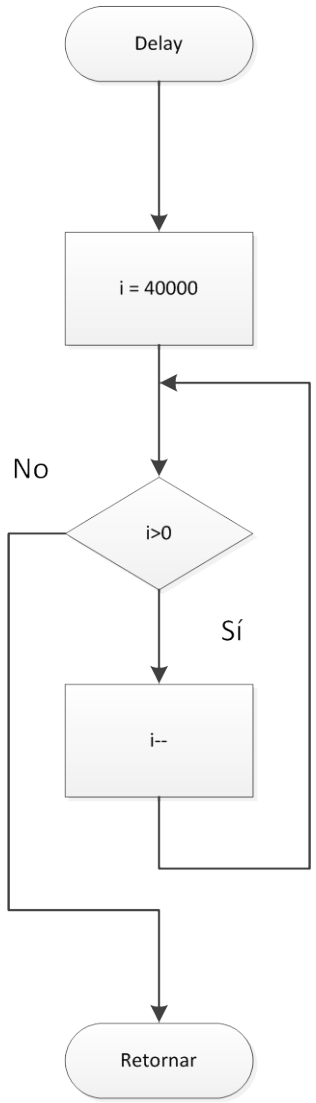


Figura 3.24 Rutina de demora por software.

Material y equipo para la práctica

1 Multímetro.
1 PC.
1 Tarjeta DEMOJM60.
1 Microcontrolador MC9S08JM60.
2 Push Button.
1 Protoboard.
8 Leds de color rojo.
8 Resistencias 330Ω
Cables.

Cuestionario Preliminar

1. ¿Qué es una interrupción por software?
2. ¿Qué es una interrupción por hardware?
3. ¿Qué es una bandera?
4. ¿En qué situaciones son requeridas las interrupciones externas?
5. ¿Cómo se atiende una interrupción en lenguaje C?
6. ¿Para qué sirve el modificador *volatile* en una interrupción?
7. ¿Qué sucede si no se declara una variable sin el modificador *volatile*?
8. ¿Qué sucede si no borramos la bandera que originó la atención de la interrupción?

3.4 Práctica Número 4:

"Control de Velocidad de un Motor de DC"

Objetivo

Controlar la velocidad de un motor de DC.

Introducción

Se tiene la impresión de que cuando se habla de motores eléctricos, se está haciendo referencia a grandes motores empleados principalmente en la industria, entonces, de acuerdo al conocimiento que se tiene de ellos por el tipo de corriente con la que operan, se piensa que la mayoría consume corriente alterna. Si se consideran las múltiples aplicaciones que tienen los motores eléctricos, tanto en el hogar, como en la oficina y en distintas aéreas de la industria, se encontrarán que los motores de corriente directa se utilizan, con pequeñas potencias, en gran variedad de casos, por ejemplo en juguetes, aparatos del hogar (licuadoras, batidoras, extractores, etc.), equipos de oficina y cómputo (impresoras de carro y de tipo láser, fotocopiadoras, etc.). En robótica se encuentra también un número importante de aplicaciones, y así como otros usos se tienen en medicina y en equipos dentales. En general, se puede establecer que en nuestra vida diaria usamos motores eléctricos grandes y pequeños; en particular, pequeños, los cuales se deben fabricar en gran cantidad.

Un motor de corriente directa cuenta con dos conexiones. La corriente eléctrica es proporcionada a través de estas, y por dentro fluye por cables que forman un electroimán. Este electroimán genera un campo magnético que reacciona contra imanes permanentes ubicados alrededor del cable, logrando que la armadura comience a girar.

La velocidad de un motor DC puede ser controlado, gracias a una técnica llamada Modulación por Ancho de Pulso. Esto se logra prendiendo y apagando el motor de forma rápida y repetitivamente. La clave es el ciclo de trabajo (duty

cycle), que es definido por el porcentaje de tiempo encendido contra el de tiempo apagado. Por ejemplo, si la corriente es proporcionada solo la mitad del tiempo, entonces el motor gira a sólo el 50% de su operación máxima. Al realizar estos cambios rápidamente el motor aparenta funcionará más lentamente sin detenerse.

Funcionamiento

Contadores y temporizadores

El microcontrolador MC09S08JM60 incorpora dos: módulos hardware para realizar las funciones de contador y temporizador estos son el Contador de Tiempo Real (Real Time Counter, RTC) y el Temporizador/Modulador de Anchura de Pulsos (Timer/Pulse-Width Modulator, TPM).

Temporizador/PWM

El microcontrolador MC09S08JM60 dispone de dos Temporizadores/PWM, TMP[1:2], de los cuales el TPM1 tiene 6 canales y el TPM2, poseen 2 canales CH[0:1].

Cada canal de un TPMx puede configurarse en diferentes modos:

- Modo de captura

Este modo permite obtener los momentos en los que se produce algún cambio en una determinada señal digital de entrada.

- Modo de comparación

Este modo permite generar una señal de salida digital cuyo nivel cambiará, según desee el programador, trascurrido un tiempo configurable.

- Modo de modulación por anchura de pulsos (PWM) alineados a flancos

Este modo permite generar señales digitales de periodo y ciclo de trabajo configurables. Este tipo de señal PWM generada se llama alineada a flancos, es decir, los primeros flancos de todas las señales se alinean con el comienzo del periodo.

- Modo de modulación por anchura del pulsos (PWM) centralmente alineados

También se pueden generar señales digitales con periodo y ciclos de trabajo configurables, pero ahora los centros de los periodos activos del ciclo de trabajo están con el centro de la señal de generada.

Inicialmente, se puede separar el funcionamiento de los canales de cada TPM en dos, dependiendo del estado en que se encuentre el bit 5, CPWMS, del registro de estado y control del TPMx, TPMxSC:

- ***Registro TPMxSC***

Registro de estado y control (TPMxSC): se muestra la configuración del registro de estado y control del TPM.

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
Escritura	0							
Reset	0	0	0	0	0	0	0	0

TOF

Bandera de sobre flujo del contador de 16 bits. Esta bandera se pone a “1” cuando se ha alcanzado el valor de 0x0000, superando el valor programado en el registro del módulo del contador.

Para poner el bit a cero de la bandera 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 sobre flujo
1: El contador ha alcanzado un sobre flujo

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 sobre flujo del contador del TPM.

0: Para detectar un evento de sobre flujo 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 el ruido en las conmutaciones del pin de salida y el contador que 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. Se muestran los registros que conforman el contador de 16 bits del TPM. La acción de escribir en cualquiera de los dos registros hace que se ponga a "0" el contador de 16 bits.

TPMxCNTH (Parte Alta)

	7	6	5	4	3	2	1	0
Lectura	15	14	13	12	11	10	9	Bit 8
Escritura	Escribir un valor en este registro pone a cero el contador de 16 bits							
Reset	0	0	0	0	0	0	0	0

TPMxCNTL(Parte Baja)

	7	6	5	4	3	2	1	0
Lectura	7	6	5	4	3	2	1	Bit 0
Escritura	Escribir un valor en este registro pone a cero el contador de 16 bits							
Reset	0	0	0	0	0	0	0	0

Los bits de los registros implicados en la configuración son el bit 5, CPWMS, del registro de estado y control el TPMx, TPMxSC, que como se observa en **la tabla 1.4** determina si el modo es PWM centralmente alineado (CPWMS = 1) o cualquier otro modo (CPWMS = 0), y los bits 5 y 4, MSnB y MSnA, así como los bits 3 y 2, ELSnB y ELSnA, del registro de estado y control del canal ***n*** del TPMx, TPMXCnSC.

Tabla 1.4 Configuración de los distintos tipos PWM.

CPWMS	*MSnB:MSnA*	*ELSnB:ELSnA*	*Modo*	*Configuración*
x	xx	00	Módulo TPM deshabilitado	
0	00	01	INPUT CAPTURE	Captura en flanco de subida
		10		Captura en flanco de bajada
		11		Captura en flanco de subida o bajada
	01	01	OUTPUT COMPARE	Cambio estado de pin en OUTPUT COMPARE
		10		Poner pin en 0 en OUT COMPARE
		11		Poner pin en 1 en OUT COMPARE
	1x	10	PWM ALINEADO AL FLANCO	Comienza en alto y cae en OUT COMPARE
		X1		Comienza en bajo y sube en OUT CAMPARE
1	xx	10	PWM ALINEADO AL CENTRO	Comienza en alto y cae en OUT COMPARE
		x1		Comienza en bajo y sube en OUT CAMPARE

Obsérvese que, independientemente de los valores que se hayan configurado para los bits CPWMS, MSnB y MSnA, cuando los bits ELSnB y ELSnA son cero, el canal ***n*** que se está usando del TPMx correspondiente no puede disponer del pin de entrada/salida que tiene asociado, TPMxCHn, que quedará libre para uso como entrada/salida de propósito general del microcontrolador.

Por otro lado, si CPWMS = 0, se puede seleccionar el canal para trabajar en:

- Modo captura
- Modo comparación
- Modo modulación por anchura de pulsos (PWM) alineados a flancos
- Modo PWM centralmente alineado

En el modo por anchura de pulsos alineado a flancos, para este modo, el bit 5, MSnB, del registro de estado y control del canal ***n*** del TPMx, TPMxCnSc, deberá estar a uno. Con la función de PWM alineado a flancos, el TPM puede generar señales digitales periódicas con una anchura de pulso en alto y bajo configurable. El periodo de la señal queda fijado por el dato escrito en el registro del módulo de 16 bits del TPMx, TPMxMOD, mientras que la anchura del pulso, y por lo tanto el ciclo de trabajo de la señal, queda fijado por el dato escrito en el registro de valor del canal ***n*** del TPMx, TPMxCnV.

Cálculos para generar una señal PWM para el control de velocidad de un motor de CD a una frecuencia de 60 Hz.

$$T = (Base\ de\ Tiempo)(Modulo) \quad ecuacion\ 1$$

Donde

T = [segundos]
Modulo = 0x0000….0xFFFF

$$Base\ de\ Tiempo = \frac{Preescalador}{Frecuencia\ Interna\ del\ uc} \quad ecuacion\ 2$$

Donde

Preescalador = [1, 2, 4, 8, 16, 32, 64, 128]

Frecuencia interna = 8 MHz

Estableciendo los valores:

Preescalador a 128
Frecuencia interna de 8MHz
Señal con una frecuencia de 60 Hz
Periodo de 16.66 milisegundos

Y sustituyéndolos en la ecuación 2

Se tienen:

$$Base\ de\ Tiempo = \frac{128}{8\ MHz} = 16\ us$$

Despejando la variable Módulo de la ecuación 1 y sustituyendo valores se tiene:

$$Modulo = \frac{16.66\ ms}{16\ us} = 1041$$

El valor obtenido es el que se carga en el registro TPMxMOD, que es valor del periodo del PWM. Se debe de configurar si la señal del PWM va a estar alineada al centro, alineada al flanco y habilitar la interrupción del canal por medio del registro TPMxCnSC.

Para obtener el 50% del ciclo de trabajo de la señal se procede de la siguiente manera:

$$D.C = (1041 * .50) = 520.5$$

$$520_d = 0780_h$$

El resultado obtenido se cargará al registro TPMxMOD = 0x0208 en su forma hexadecimal. El resultado se aprecia en la figura 3.25 donde se observa la forma de la señal, su frecuencia, periodo y su ciclo de trabajo, por el pin 13 de la tarjeta DEMOJM, por los cálculos realizados anteriormente.

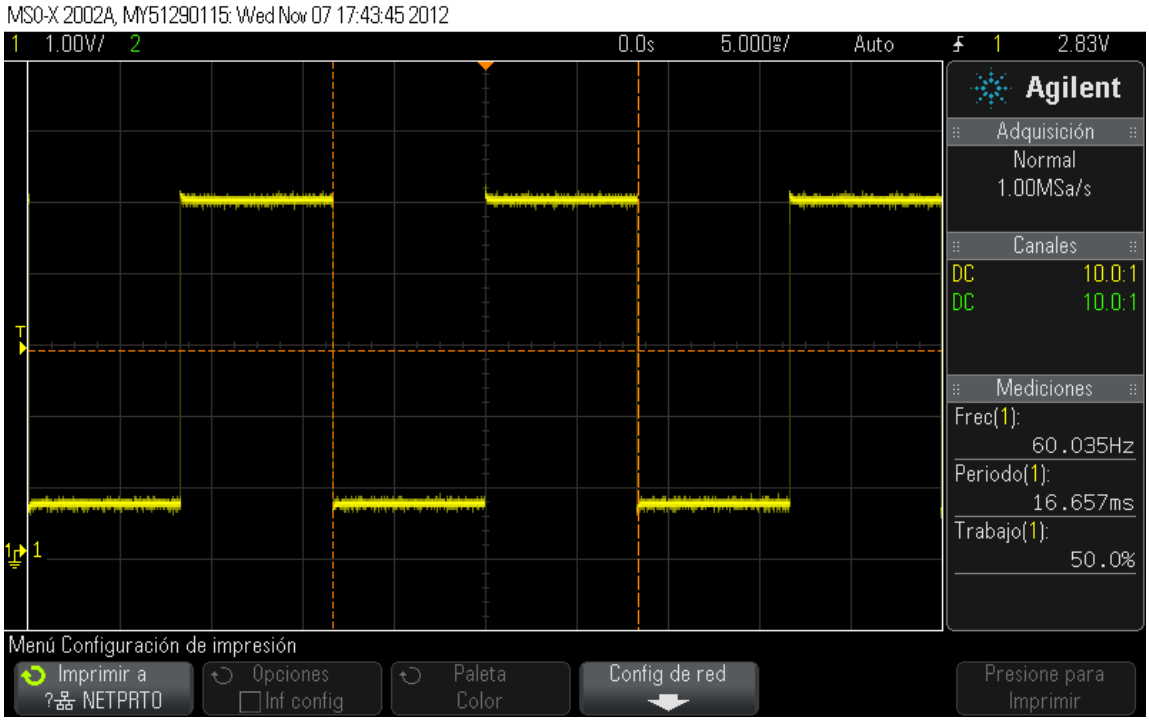


Figura 3.25 Muestra de la señal al 50% de trabajo.

Para realizar el cálculo del ciclo de trabajo del 75% se procede de la siguiente manera:

$$D.C = (1041 * .75) = 780.75$$

$$780_d = 030C_h$$

Se carga el valor nuevo al registro TPMxMOD = 0x030C en forma hexadecimal, pero se debe de considerar en la tarjeta DEMOJM, que en ella se obtiene a su

salida del canal CH[0] un valor inverso a la señal, por lo tanto, se debe de plantear la siguiente ecuación para que dé el valor correspondiente a 75% del ciclo de trabajo de la señal.

$$D.C = (1041 * .25) = 260.25$$

$$260_d = 0104_h$$

Cargado el valor correspondiente al 25% en el registro TPMxMOD, dará el resultado correcto como se muestra en la figura 3.26

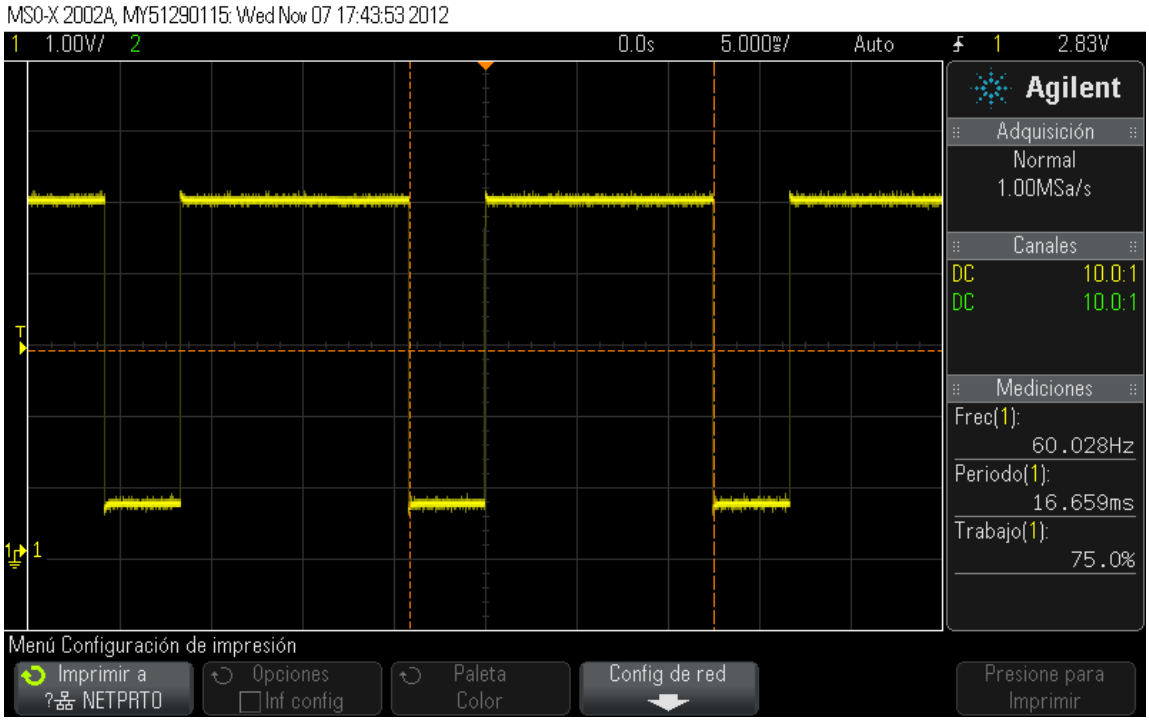


Figura 3.26 Señal al 75% de trabajo.

Desarrollo

El circuito de la figura 3.27 detalla la aplicación a implementar, para el controlar la velocidad de un motor de DC.

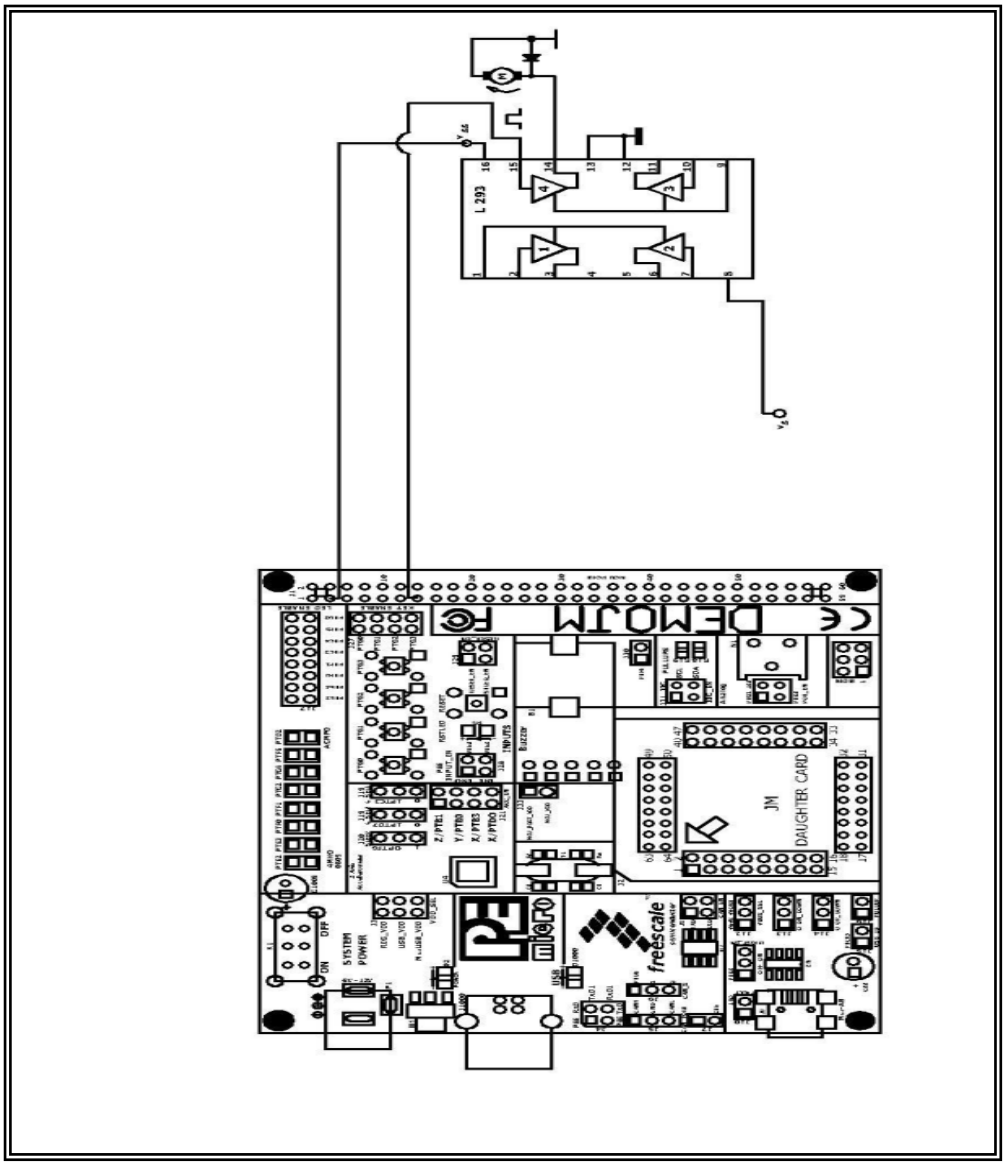


Figura 3.27 Circuito para el control de velocidad de un motor de DC.

1. Capture el siguiente programa para controlar un motor DC.

```
/*****************************************************************************************************
*********

Programa: Control de velocidad de un motor de DC a 60 Hz.

Autor: Oscar M. Espinoza

Fecha: Septiembre 1 del 2012

Versión: 1.0

*****************************************************************************************************
**********/

#include <hidef.h>                    /* Macro para habilitar interrupciones */
#include "derivative.h"               /* Incluye declaración de periféricos */

volatile unsigned char newL_DC = 0x0E; //Declaración de variable tipo char
volatile unsigned char newH_DC = 0x04; //Declaración de variable tipo char

#define SW_0() !(PTGD_PTGD0)          //Macro para definir switch 0
#define SW_1() !(PTGD_PTGD1)          //Macro para definir switch 1
#define SW_2() !(PTGD_PTGD2)          //Macro para definir switch 2
#define SW_3() !(PTGD_PTGD3)          //Macro para definir switch 3

#define Disable_COP() SOPT1 &= 0x3F

/*****************************************************************************************************
*********/
//Prototipo para función de retardo

 void Delay_ms(unsigned int del);

/*****************************************************************************************************
*********/

void main(void) {          //Principal
```

```
MCGC1 = 0x04;          //Configuración del reloj aproximadamente 8 MHz
MCGC2 = 0x40;
MCGC3 = 0x01;

Disable_COP();         //Deshabilita el COP

EnableInterrupts;      /* Habilita interrupciones */

PTGD  = 0x00;          //Configura el puerto G como entrada y activa la resistencia
PTGDD = 0x00;          //de pull-up
PTGPE = 0xFF;

TPM1SC   = 0x4F;       //Carga el valor de configuración para el registro TPM1SC
TPM1C0SC = 0x64;       //Carga el valor de configuración para el registro
TPM1MOD  = 1041;       //Valor del periodo del PWM (62500 * 16.16ms = 1041)
TPM1C0V  = 104;        //Valor equivalente 0º aproximadamente (1041 *.10 = 104)

TPM2MOD = 8000;        // El contador se desbordara cada milisegundo


TPM2SC = 0x00;         // Detener el timer 2 y selecciona el divisor = 1


for(;;) {


 while(SW_0()){          //Si es presionado el SW_0

   Delay_ms(20);         //Llama a la función de retardo (para contrarrestar efecto
                         // mecánico del pulsador)
   newL_DC = 0x04;       //Carga el valor del 75% de velocidad (1041 * .25 = 260)
   newH_DC = 0x01;

 }

 while(SW_1()){          //Si es presionado el SW_1

   Delay_ms(20);         //Llama a la función retardo (para contrarrestar efecto
                         // mecánico del pulsador)
   newL_DC = 0x08;       //Carga el valor del 50% de velocidad (1041 *.50 = 520)
   newH_DC = 0x02;

 }
```

```
  while(SW_2()){              //Si es presionado el SW_3

    Delay_ms(20);             //Llama a la función retardo (para contrarrestar efecto
                              // mecánico del pulsador)
    newL_DC = 0x0D;
    newH_DC = 0x03;     //Carga el valor del 25% de velocidad (1041 *.75 = 781)

  }

  while(SW_3()){            //Si es presionado el SW_4

    Delay_ms(20);           //Llama a la función retardo (para contrarrestar efecto
                            // mecánico del pulsador)
    newL_DC = 0xFC;     //Carga el valor del 2% de velocidad (1041 *.98 = 1020)
    newH_DC = 0x03;

  }

 }

}


/*************************************************************************************************
**********/

//FUNCION DE RETARDO POR MEDIO DEL TIMER_2

void Delay_ms(unsigned int del){

  TPM2SC = 0x48;            //Activa el Timer2

  while(del)                //Mientras igual a cero

    if(TPM2SC & 0x80)       //Activa y desactiva el timer por 20 veces
    {
      del--;                //Decrementa a la variable "del"

      TPM2SC &= ~0x80;      //Limpia la bandera TOF
    }

  TPM2SC = 0x00;            //Para el TIMER_2
```

```
}

////////////////////////////////////////////////////////////////////////////////////////////////////

//ATENCION A LA INTERRUPCION POR DESBORDAMIENTO DEL TIMER_0

interrupt VectorNumber_Vtpm1ovf void ISR_TPM_OVF1 (void){

  (void)TPM1SC;                 //Lee registro de estado y control
  TPM1SC_TOF = 0;               //Bandera del registro a cero

}

//ATENCION A LA INTERRUPCION POR TPM CANAL_0

interrupt VectorNumber_Vtpm1ch0 void ISR_TPM1_CH0 (void){

  (void)TPM1C0SC;               //Borrar bandera del canal
  TPM1C0SC_CH0F=0;              //Bandera a 0 del canal

  TPM1C0VL = newL_DC;           //Actualiza D.C. (Duty Cycle, Ciclo de trabajo)
  TPM1C0VH = newH_DC;

}

//ATENCION A LA INTERRUPCION POR DESBORDAMIENTO DEL TIMER_2

interrupt VectorNumber_Vtpm2ovf void ISR_TPM_OVF2 (void){

  (void)TPM2SC;                 //Lee registro de estado y control
  TPM2SC_TOF = 0;               //Bandera de sobre flujo a 0

}
```

Se muestra el diagrama de flujo el programa principal para variar la velocidad de un motor de dc:

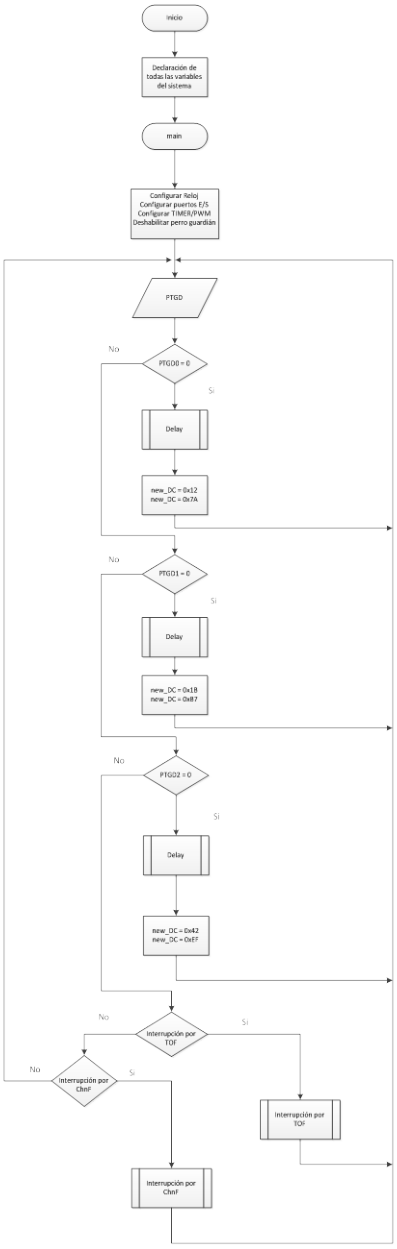


Figura 3.28 Configuración de módulos y periféricos para variar la velocidad del motor.

Se muestra el diagrama de flujo para cuando es invocada la interrupción por desbordamiento por TIMER:

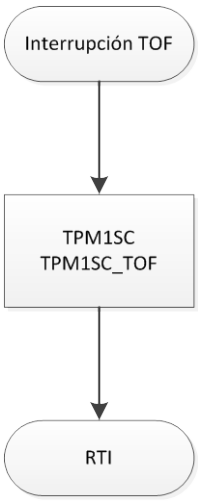


Figura 3.29 Atención a la interrupción por TIMER.

Se muestra el diagrama de flujo para cuando es invocada la interrupción por TIMER_PWM:

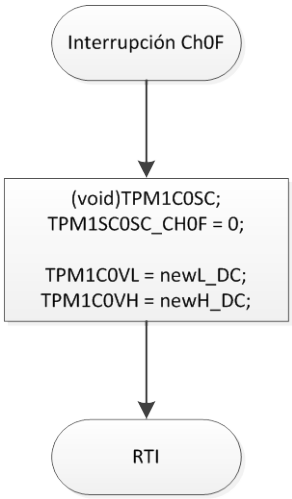


Figura 3.30 Atención a la interrupción por TIMER_PWM

Se muestra el diagrama de flujo para cuando es invocada la rutina para realizar retardos por desbordamiento por TIMER:

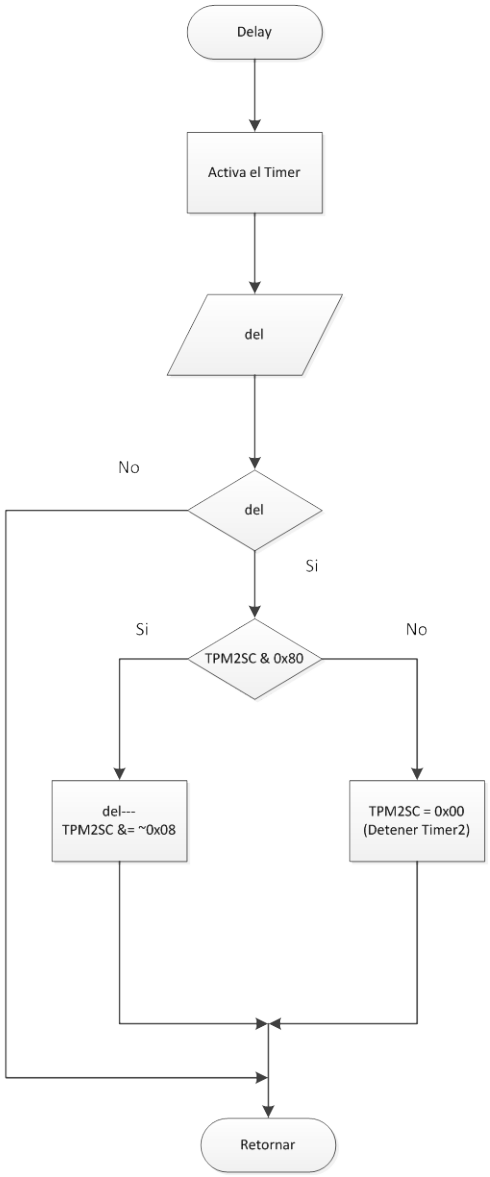


Figura 3.31 Rutina para genera retardos por medio del TIMER.

Material y equipo para la práctica

1 Multímetro.
1 PC.
1 Tarjeta DEMOJM60.
1 Microcontrolador MC9S08JM60.
1 Circuito integrado L293D o L293B.
4 Diodos 1N4007.
1 Motor de CD de 9 Volts.
1 Protoboard.
1 Fuente de alimentación CD a 24 Volts.
Cables.

Cuestionario Preliminar

1. ¿Qué tipos de motores eléctricos conoce?
2. ¿Qué es una señal PWM?
3. ¿Qué es ciclo de trabajo (Duty cicle)?
4. ¿Qué es periodo de una señal?
5. ¿Qué es frecuencia de una señal?

3.5 Práctica Número 5:

"Control de un Servomotor de DC"

Objetivo

Controlar el posicionamiento de servomotores por PWM.

Introducción

Un servomotor es básicamente un actuador mecánico basado en un motor y un conjunto de engranajes que permiten multiplicar el torque del sistema final, el cual posee elementos de control para monitorear de manera constante la posición de un elemento mecánico, que será el enlace con el mundo exterior. Es decir, ante una acción inducida electrónicamente a un servomotor, obtendremos por resultado una respuesta mecánica controlada. Por ejemplo, los motores que forman parte de una impresora, junto a los sistemas de control de avance o retroceso del papel, forman un servomotor. Las aplicaciones de estos sistemas se pueden observar mayormente en aeromodelismo y robótica, pero no son exclusivos de estos usos. Cualquier sistema que requiera un posicionamiento mecánico preciso y controlado dependerá de un servosistema o servomecanismo, actuado, por supuesto, por un servomotor. El zoom de una cámara, el autoenfoque de un conjunto óptico, un sistema de movilización de cámaras de vigilancia y hasta las puertas automáticas de un ascensor son sencillos ejemplos de su aplicación.

El motor posee la característica de girar a una buena velocidad, la cual disminuye por los juegos de engranajes de la caja reductora que aprovechan esta velocidad para transformarla en fuerza de trabajo, en la figura 3.32 muestra el diagrama de bloques de un servomotor. Al girar el último engranaje acoplado al eje de salida obtenemos una velocidad notablemente reducida, a pesar de que, dentro del sistema, el motor está girando a altas velocidades.

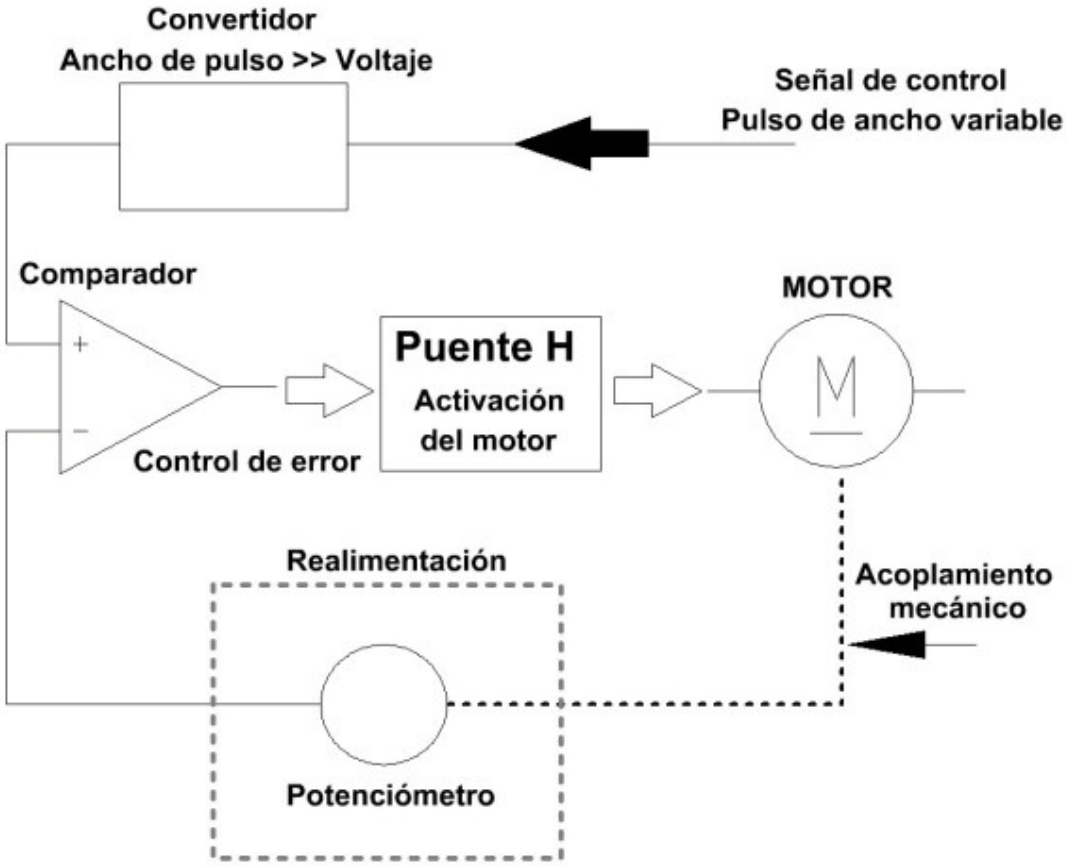


Figura 3.32 Diagrama a bloques de la estructura interna de un servomotor.

Además, en esta última rueda de acoplamiento encontraremos topes o límites de recorrido para entregarnos en la salida final un giro de 180° del brazo actuador. En la mayoría de los servomotores, este desplazamiento angular es “copiado” por un potenciómetro incorporado al sistema de control en forma mecánica al eje externo. Este sensor resistivo se encargará de informarle al sistema la posición que posee el actuador exterior, para así controlar con exactitud que la instrucción de posicionamiento enviada esté siendo ejecutada fielmente.

Funcionamiento

Las conexiones son muy sencillas y se basan en una normativa de colores muy elementales que involucran al rojo como positivo de la alimentación principal, junto a otro cable que puede ser de color negro o marrón y que, por lógica, podemos deducir que se trata del negativo de alimentación. Un tercer cable, correspondiente al control de posicionamiento del actuador mecánico, es amarillo, naranja o blanco.

La tensión de trabajo de los servomotores suele estar comprendida entre los 3 y los 7 Volts, siendo 5 Volts la tensión que se utiliza en la mayoría de las aplicaciones fijas donde interviene una fuente de alimentación conectada a la red de energía domiciliaria, y 6 Volts para los casos de alimentación a baterías cuando se trata de equipos móviles. En todos los casos, siempre se requiere de una señal de control de 5 Volts de amplitud.

La señal de control del servo, como mencionamos al principio, se realizará mediante impulsos de ancho variable que deben refrescarse periódicamente. Esto significa que, si dejamos de enviar la señal de control en el tiempo en el que el servomotor lo necesita, éste a pesar de estar energizado dejará de mantenerse en la posición preestablecida y adoptará cualquier orientación regida por el esfuerzo al que esté sometido. Si no mantenemos la señal de control en forma efectiva todo el tiempo que sea necesario, el sistema quedará a merced de las fuerzas externas a la que sea sometido. Por ejemplo, un brazo de palanca dejará de sostener un objeto y se dejará caer todo el trayecto mecánico que pueda recorrer, o un sistema erguido en vertical se caerá hacia atrás, o hacia adelante, al momento en el que el servomotor deje de “sostener” la aplicación en la posición preestablecida.

Para bloquear al servomotor en una posición es necesario, entonces, enviarle continuamente la señal con la posición deseada. De esta forma, el sistema de control seguirá operando y el servo conservará su posición y se resistirá a las fuerzas externas que intenten cambiarlo de posición.

Registros que se deben configurar:

- ***Registro TPMxSC***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	TOF	TOIE	CPWMS	CLKSB	CLKSA	PS2	PS1	PS0
Escritura	0							
Reset	0	0	0	0	0	0	0	0

- ***TPMxCNTH (Parte Alta)***

	7	6	5	4	3	2	1	0
Lectura	15	14	13	12	11	10	9	Bit 8
Escritura	Escribir un valor en este registro pone a cero el contador de 16 bits							
Reset	0	0	0	0	0	0	0	0

- ***TPMxCNTL (Parte Baja)***

	7	6	5	4	3	2	1	0
Lectura	7	6	5	4	3	2	1	Bit 0
Escritura	Escribir un valor en este registro pone a cero el contador de 16 bits							
Reset	0	0	0	0	0	0	0	0

Desarrollo

La figura 3.33 detalla la aplicación a implementar, como ejercicio de control del posicionamiento de un servomotor.

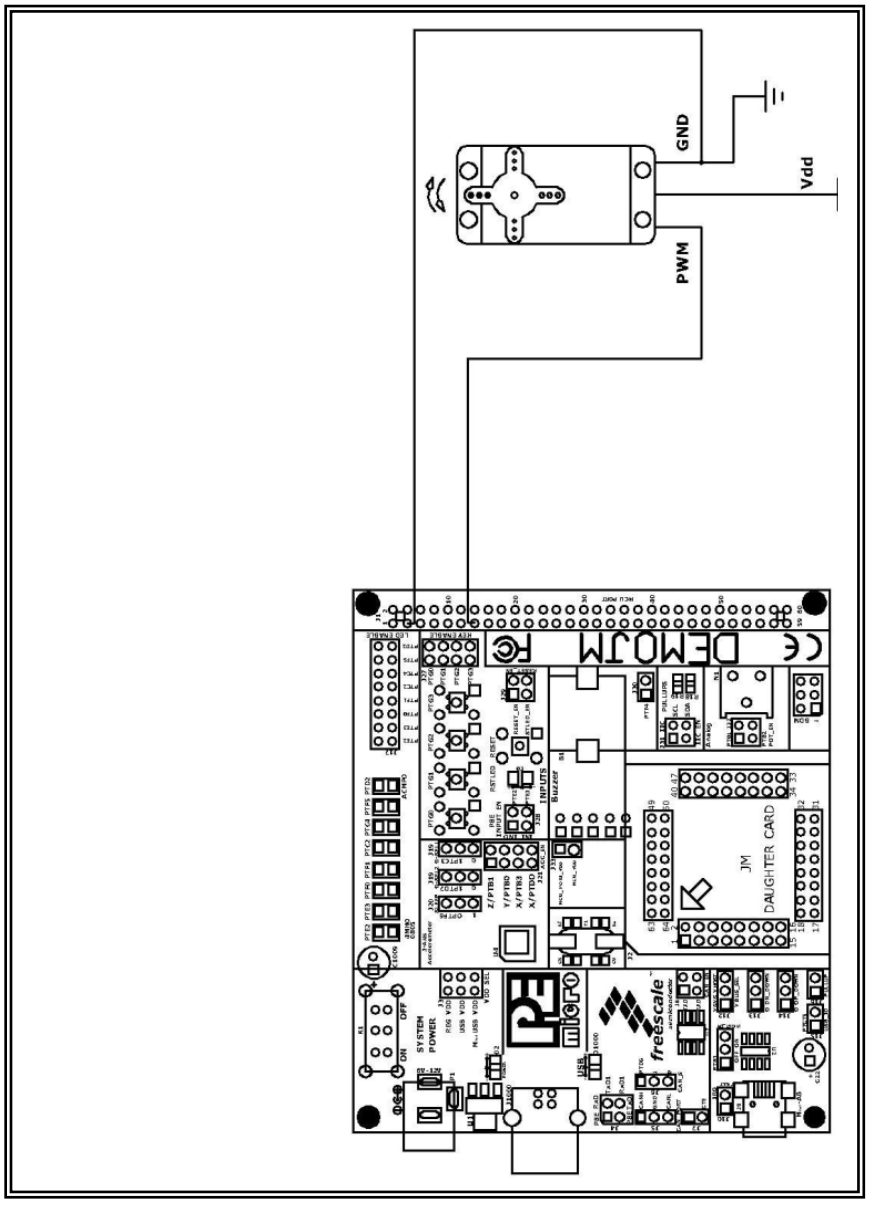


Figura 3.33 Circuito para controlar el posicionamiento de un servomotor.

1. Capture el siguiente programa para controlar el posicionamiento de un servomotor.

```
/*****************************************************************************************************
*********

Programa: Para controlar servomotores utilizando PWM a una frecuencia de
aproximada de 50 Hz para un óptimo desempeño.

Autor: Oscar M. Espinoza

Fecha: Octubre 2 2012

Versión: 1.0

Nota: Retirar el jumper correspondiente y revisar las conexiones correctamente


*****************************************************************************************************
*********/

#include <hidef.h>                      /* Macro para habilitar interrupciones */
#include "derivative.h"                 /* Incluye declaración de periféricos */


unsigned int new_duty = 39600;              //Ciclo de trabajo por default

volatile byte bandera = 0;

#define INPUT_0() !(PTGD_PTGD0)             //Macros para entradas
#define INPUT_1() !(PTGD_PTGD1)
#define INPUT_2() !(PTGD_PTGD2)


#define KEY_0     0                         //Constantes
#define KEY_90    1
#define KEY_180 2

#define KEY_INVAL 3

#define Disable_COP() SOPT1 &= 0x3F
```

```
/*****************************************************************************************************
**********

 //*PROTOTIPO DE FUNCIONES***//

void Perif_Init(void);
void Key_Run(void);
char Key_Read(void);

/*****************************************************************************************************
*********/

void main(void) {


  MCGC1 = 0x04;                    //Reloj aproximadamente a 8 MHz
  MCGC2 = 0x40;
  MCGC3 = 0x01;

  Disable_COP();


  EnableInterrupts;                /* Habilita interrupciones */
  /* include your code here */


  Perif_Init();                    //Llama función para inicializar periféricos


  for(;;) {                        //Ciclo infinito

    Key_Run();                     //Lee interruptor presionado

  }

}
```

```
/*************************************************************************************************
*********/

void Perif_Init(void){

  PTGD  = 0x00;                  //Configuración del Puerto A como entrada y salida
  PTGDD = 0xF0;
  PTGPE = 0xFF;                  //Activa resistencias de pull-up

  TPM1MOD = 40625;               //Timer cuenta hasta 40625 al 100% de la señal
                                 // Modulo = tiempo de señal / Base de tiempo
                                // 20ms / .5us = 40000
   //NOTA:
   //Para un mejor desempeño del servomotor utilizar el valor 40625


  TPM1SC   = 0x4A;       //Habilita interrupción por desbordamiento, divisor por 4
  TPM1C0SC = 0x7C;     //Canal_0, flanco inicia de bajada y no alineado

  TPM1CNT = 0;                  //Contador a 0

  TPM1C0V = new_duty;     //Carga al registro el valor por default "posición cero"

}

void Key_Run(void){

 char nroTecla;                  //Declaración de variable local

 nroTecla = Key_Read();   //Carga el resultado de la tecla presionada

 while(!bandera);              //Espera a que se produzca la interrupción por PWM

   bandera = 0;                 //Bandera a cero

   switch(nroTecla){          //Lee que tecla fue presiona

    case KEY_0:

      new_duty = 39600;          //Carga el valor de posición 0º al 97% o 2.52%

    break;
```

```
    case KEY_90:

      new_duty = 37900;          //Carga el valor de posición 90º al 93% o 7%

    break;


    case KEY_180:

      new_duty = 36000;          //Carga valor de posición 180º al 87% o 11%

    break;


    default:                     //Default
    break;

   }

}

char Key_Read(void){

 if(INPUT_0()) return KEY_0;     //Si es presionado el interruptor 0 entonces KEY_0
 if(INPUT_1()) return KEY_90;    //Si es presionado el interruptor 1 entonces KEY_1
 if(INPUT_2()) return KEY_180;

 return KEY_INVAL;

}

////////////////////////////////////////////////////////////////////////////////////////////////////
//Atención a la interrupción por desbordamiento

interrupt VectorNumber_Vtpm1ovf void ISR_TPM_OVF (void){

  (void)TPM1SC;                  //Lee registro de estado y control
  TPM1SC_TOF=0;                  //Bandera del registro a cero

}
```

```
//Atención a  la interrupción TPM1 CANAL 0
interrupt VectorNumber_Vtpm1ch0 void ISR_TPM1_CH0(void){

  (void)TPM1C0SC;                 //Lee registro de estado y control
  TPM1C0SC_CH0F = 0;              //Bandera del registro a cero

  TPM1C0V = new_duty;             //Actualiza el Duty cycle (ciclo de trabajo)

  bandera = 1;                    //Asigna 1 a la variable bandera
}
```

Material y equipo para la práctica

1 Multímetro.
1 PC.
1 Tarjeta DEMOJM60.
1 Microcontrolador MC9S08JM60.
1 Servomotor de 4.8 ~ 6.0 Volt de DC.
1 Protoboard.
1 Fuente de alimentación CD a 24 Volts.
Cables.

Cuestionario Preliminar

1. ¿Qué es un servomotor?
2. ¿Qué componentes internos componen a un servomotor?
3. ¿Cómo funciona un servomotor?
4. ¿Cuántos grados puede moverse un servomotor?
5. ¿Cuáles son las frecuencias a las que se debe configurar un servomotor para ser posicionado a 0º, 90º y 180º?

3.6 Práctica Número 6:

"Convertidor Analógico Digital"

Objetivo

Comprender el funcionamiento del convertidor A/D del MC9S08JM60.

Introducción

En muchas situaciones se requiere controlar un proceso mediante un sistema digital. Las señales provenientes del mundo exterior son de tipo analógico, como por ejemplo presión, temperatura, velocidad, voltaje, etc., en las cuales mediante el uso de un transductor se realiza el proceso de conversión de un tipo de señal física a una señal eléctrica. Pero, un microcontrolador solamente reconoce señales de tipo digital, y por ello se requiere convertir las señales analógicas a digitales.

Los convertidores analógico/digital (A/D) y digital/analógico (D/A) son elementos que permiten a los sistemas digitales "comunicarse" con sistemas analógicos, tanto para recibir como para suministrar información. Se debe de tener en cuenta que una señal digital es aquella que toma sólo dos valores lógicos (0,1) traducidos a valores discretos de voltaje, en cambio una señal analógica es aquella que varía en forma continua desde un mínimo hasta un máximo de corriente o voltaje.

La conversión de una señal analógica a una digital puede realizarse mediante diferentes formas:

- Convertidores con salida en paralelo

Como su nombre lo indica, entregan la señal binaria en paralelo, equivalente al valor de la variable analógica de entrada. Son los convertidores comúnmente usados por su fácil acoplamiento a procesos digitales.

- Convertidores de salida en serie

También codifican la señal de entrada en una digital, entregándola por única salida serie.

Cuando se desea realizar un diseño haciendo uso de un convertidor analógico digital, hay necesidad de conocer como mínimo algunos parámetros propios del convertidor, que pueden suministrar información valiosa en el momento de decidir si es adecuado o no para su implantación. Estos parámetros son:

- Características de diseño

Constituyen los parámetros básicos de los convertidores A/D

- Código binario

Corresponden al tipo de código que obtiene en la salida del convertidor: binario natural o BDC

- Resolución

Se define como el mínimo incremento necesario en la entrada analógica para que se produzca un cambio en la combinación binaria.

- Tiempo de conversión

Es el intervalo de tiempo necesario para que se produzca una conversión.

- Características de funcionamiento

Las características reales de un convertidor A/D también difieren de las ideales, y corresponden a las mismas definidas.

Funcionamiento

De manera resumida, el proceso completo de conversión es como sigue: dado que se disponen de 28 canales de entrada en el microcontrolador MC9S08JM60, antes de realizar una conversión, se debe seleccionar el canal a convertir. A continuación se genera la orden de disparo que desencadena el proceso de conversión en el ***SAR*** (*Successive Approximation Register, Registro de Aproximaciones Sucesivas*) y, una vez ejecutado el algoritmo de aproximaciones sucesivas, el resultado o muestra digital es ubicado en los registros de datos, indicándose también en una bandera de estado el hecho de que dicho proceso de conversión ha finalizado. Este evento de fin de conversión puede ir acompañado o no de la activación de una interrupción hacia el MC9S08JM60.

El ADC pueden estar realizando conversiones en dos modos: modo simple o continuo, el ADC vuelve al estado de reposo (idle) cuando termina una conversión y queda en espera de otra indicación de disparo. En modo continuo, sin embargo, cuando finaliza una conversión, de manera automática, comienza una nueva.

Desde el punto de vista de la potencia consumida, cuando el módulo ADC está habilitado, éste puede encontrarse en dos estados diferentes: a) realizando una conversión b) en una situación de reposo esperando la orden de convertir otra muestra. Obviamente, en estado de reposo el consumo de potencia es mucho menor que cuando se encuentra convirtiendo.

Un aspecto de vital importancia en la configuración de este módulo es la elección de la frecuencia interna del funcionamiento del ADC (f_{ADCK}), que debe de ajustarse a las especificaciones del sistema y de la señal o señales a monitorizar y controlar. Para ello, además de poder elegir la fuente para generar dicha frecuencia, esta se puede dividir por 1, 2, 3, 4 y 8. Esta frecuencia influye en el tiempo de conversación, muestra la rapidez que el ADC realizar una conversión y es importante no confundirla con la frecuencia de muestreo de los canales.

Antes de abordar todas las opciones de funcionamiento del ADC se debe de resaltar que, en esta familia de microcontroladores, este módulo dispone de capacidad de realizar automáticamente una comparación entre el resultado obtenido de la conversión y un valor almacenado en los registros de comparación. Esta función de comparación permite que se pueda monitorizar el

voltaje de un canal, mientras el microcontrolador se encuentra en estado de espera (*wait*) o de parada (*stop3*).

La ***Resolución*** de un convertidor 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 convertidor A/D que tiene una resolución de 12 bits, el número de valores discretos en los cuales se puede valorar a una señal será 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}).

Se recomienda que el usuario aproveche al máximo la resolución del sistema, adecuando la señal análoga para que incursionara 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 3 Volts, 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 convertidor de12 bits de resolución, un V_{min} = 0V y un V_{max} = 5V.

El paso mínimo de conversión está dado por:

$$Paso\ minimo = \frac{V_{max} - V_{min}}{Resolucion}$$

Ejemplo:

$$Paso\ minimo = \frac{5V - 0V}{2^{12}}$$

$$Paso\ minimo = 1.22\ mV$$

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 convertidor A/D ignora información del mundo análogo.

El ***Muestreo*** (*sample rate*) es otra característica importante de un convertidor A/D y se refiere a la cantidad de muestras, en la unidad de tiempo, que se puede procesar y convertir a cantidades discretas.

La figura 3.34 presenta una señal análoga continua entre los puntos T0 y T1, 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.

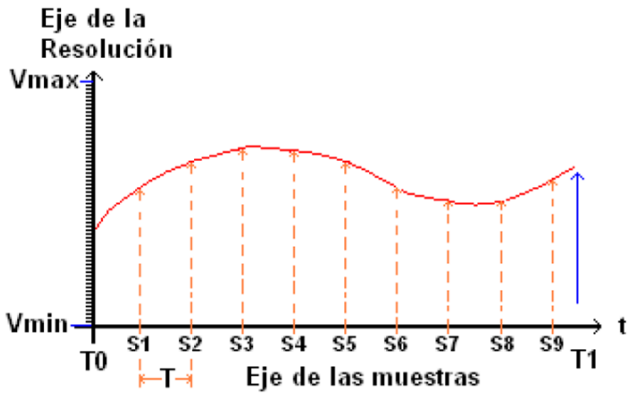


Figura 3.34 Representación de una señal analógica muestreada.

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. El usuario deberá siempre conocer las limitaciones de velocidad del convertidor y aplicar el teorema de Nyquist-Shannon a la hora de capturar el programa.

Registros que se deben configurar:

- ***Registro de estado y control 1 (ADCSC1)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	COCO	AIEN	ADCO	ADCH				
Escritura								
Reset	0	0	0	1	1	1	1	1

COCO

Bandera de conversión completa.

Cuando COCO = 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: Deshabilita 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.
1: Habilita conversión continua. La conversión es iniciada cuando se escribe sobre el registro ADCSC1 o cuando se trabaja con señal de disparo externa.

ADCH

Bits para seleccionar el canal a convertir (multiplexor). En la Tabla 1.5 siguiente se detalla la combinación de bits según el canal.

Tabla 1.5 selección del canal ADC

ADCH	***Selección del Canal***
00000 - 01111	ADC0 - 15
10000 – 11011	ADC16 – 27
11100	Reservado
11101	V_{REFH}
11110	V_{REFL}
11111	Módulo Deshabilitado

- ***Registro de estado y control 2 (ADCSC2)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	ADACT	ADTRG	ACFE	ACFGT	0	0	R^2	R^1
Escritura								
Reset	0	0	0	0	0	0	0	0

ADACT

Bit que indica que 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. Cuando se elige la conversión por software, la simple escritura sobre el registro ADCSC1 inicializa una 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)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	0	0	0	0	ADR11	ADR10	ADR9	ADR8
Escritura								
Reset	0	0	0	0	0	0	0	0

Se 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 resultado bajo de la conversión (ADCRL)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	ADR7	ADR6	ADR5	ADR4	ADR3	ADR2	ADR1	ADR0
Escritura								
Reset	0	0	0	0	0	0	0	0

Se 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 de 8 bits.

- ***Registro alto de comparación (ADCCVH)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	0	0	0	0	ADCV11	ADCV10	ADCV9	ADCV8
Escritura								
Reset	0	0	0	0	0	0	0	0

Se 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 bajo de comparación (ADCCVL)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	ADCV7	ADCV6	ADCV5	ADCV4	ADCV3	ADCV2	ADCV1	ADCV0
Reset	0	0	0	0	0	0	0	0

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 de configuración (ADCCFG)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	ADLPC	ADIV		ADLMSP	MODE		ADICLK	
Reset	0	0	0	0	0	0	0	0

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

ADSMP

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 cortó 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)***

Los registros de control de pins del ADC, todo canal que se desee habilitar necesita ser confirmado escribiendo un "1" en el respectivo bit ADPCx.

	Bit 7	6	5	4	3	2	1	Bit 0
APCTL1	ADPC7	ADPC6	ADPC5	ADPC4	ADPC3	ADPC1	ADPC1	ADPC0
APCTL2	-	-	-	-	ADPC11	ADPC10	ADPC9	ADPC8
Reset	0	0	0	0	0	0	0	0

Cálculo del tiempo de una conversión.

Para efectos del cálculo del tiempo que toma una conversión en ejecutarse, la Tabla 1.6 relaciona el tiempo dependiendo del reloj elegido:

$$ADCLK = \frac{ADICLK}{ADIV}$$

y el bit ADSMP (tiempo corto o largo).

Tabla 1.6 relación de tiempo de conversión.

Tipo de conversión	*ADICLK*	*ADLSMP*	*Máximo tiempo de conversión*
Simple o primera en modo continuo (8 bits)	0x, 10	0	20 ADCK ciclos + 5 ciclos de reloj del BUS
Simple o primera en modo continuo (10 bits y 12 bits)	0x, 10	0	23 ADCK ciclos + 5 ciclos de reloj del BUS
Simple o primera en modo continuo (8 bits)	0x, 10	1	40 ADCK ciclos + 5 ciclos de reloj del BUS
Simple o primera en modo continuo (10 bits y 12 bits)	0x, 10	1	43 ADCK ciclos + 5 ciclos de reloj del BUS
Simple o primera en modo continuo (8 bits)	11	0	5 µs + 20 ADCK + 5 ciclos de reloj
Simple o primera en modo continuo (10 bits y 12 bits)	11	0	5 µs + 23 ADCK + 5 ciclos de reloj

Simple o primera en modo continuo (8 bits)	11	1	5 µs + 40 ADCK + 5 ciclos de reloj
Simple o primera en modo continuo (10 bits y 12 bits)	11	1	5 µs + 43 ADCK + 5 ciclos de reloj
Las siguientes en modo continuo (8bits)	xx	0	17 ADCK ciclos
Las siguientes en modo continuo (10 bits y 12 bits)	xx	0	20 ADCK ciclos
Las siguientes en modo continuo (8bits)	xx	1	37 ADCK ciclos
Las siguientes en modo continuo (10 bits y 12 bits)	xx	1	40 ADCK ciclos

Continuación Tabla 1.6 relación de tiempo de conversión.

Desarrollo

El circuito de la figura 3.35 detalla la aplicación a implementar, como ejercicio del manejo del convertidor analógico-digital.

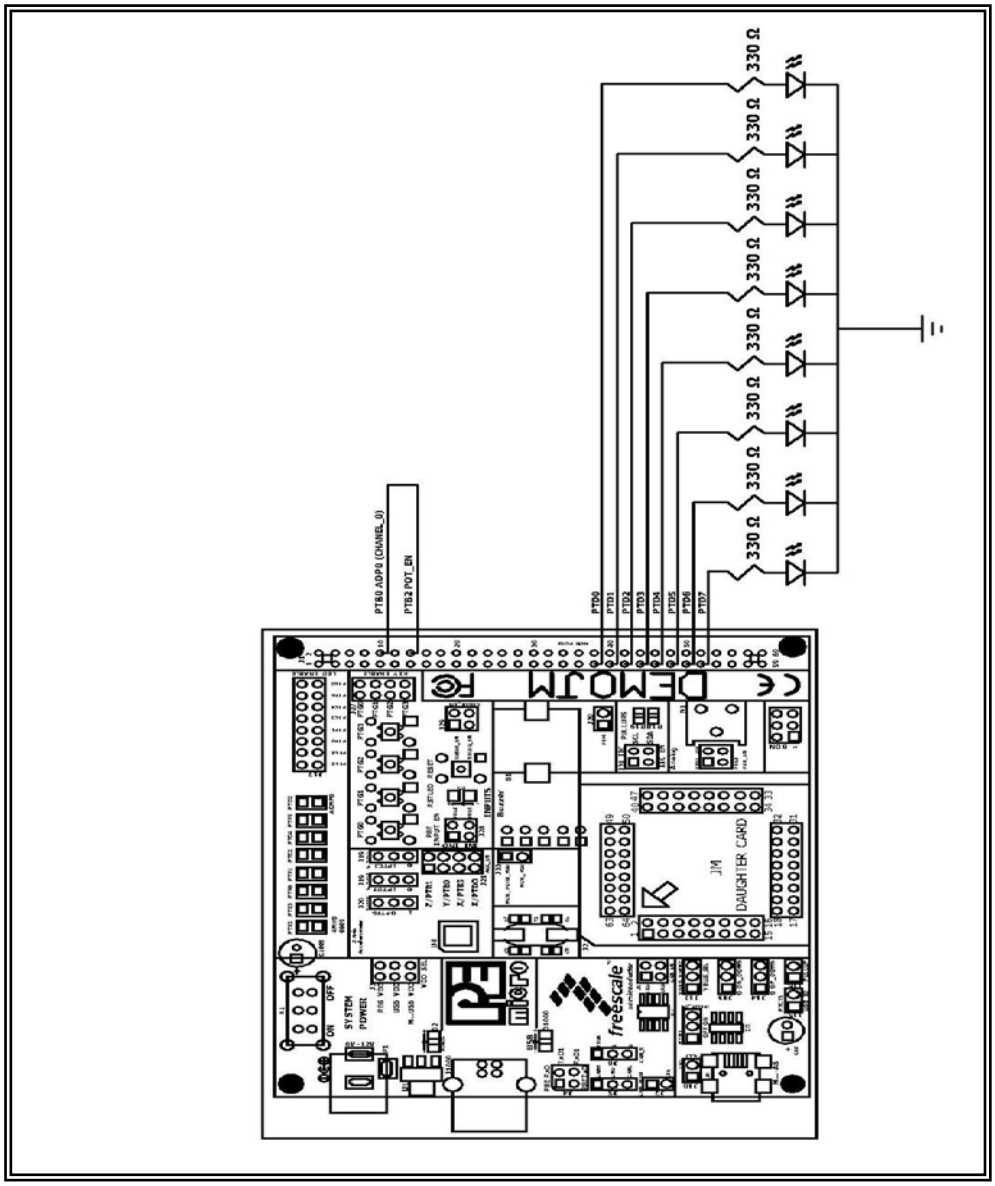


Figura 3.35 Circuito para conversión de 8 bits.

1. Capture el siguiente programa para utilizar el convertidor analógico-digital.

```
/*****************************************************************************************************
**********

Programa: Funcionamiento del convertidor analógico-digital de 8 bits

Autor: Oscar M. Espinoza

Fecha: Septiembre 14 2012

Versión: 1.0

NOTA: LIBERAR LOS JUMPERS CORREPONDIENTES.

*****************************************************************************************************
**********/

#include <hidef.h>                          /* Macro para habilitar interrupciones */
#include "derivative.h"                     /* Incluye declaración de periféricos */

volatile unsigned char valor_convL = 0;      //Declaración de variables
volatile byte bandera = 0;

#define Disable_COP() SOPT1 &= 0x3F

/*****************************************************************************************************
*********/
 /* PROTOTIPO DE FUNCIONES */

void ADC_Init(void);
void Delay(unsigned int dato);

/*****************************************************************************************************
********/

void main(void) {                    //Principal

  MCGC1 = 0x04;
  MCGC2 = 0x40;                     /* Reloj a 8 MHz              */
  MCGC3 = 0x01;
```

```
  Disable_COP();

  EnableInterrupts;              /* Habilita interrupciones*/

/* include your code here */


  ADC_Init();                    //Llama a inicializar el convertidor

  TPM1MOD = 8000;                //Timer contará hasta 8000 (1 milisegundo)
  TPM1SC = 0x00;                 //Timer deshabilitado

  PTDD  = 0x00;                  //Puerto D configurado como salida
  PTDDD = 0xFF;

  for(;;) {

    while(!bandera);             //Espera a interrupción por convertidor

      ADCSC1 = 0x00;             //Deshabilita convertidor

      bandera = 0;               //bandera a 0

      Delay(50);                 //Retardo para poder ver las muestras obtenidas

      PTDD = valor_convL;

      ADCSC1 = 0x40;             //Habilita de nuevo al convertidor


  }

}

/*************************************************************************************************
**********/

//Función que inicializa al convertidor analógico digital
void ADC_Init(void){

  ADCCFG = 0x90;    //Bajo consumo, factor de división es 1, conversión lenta, 12
                    //bit 0x94, reloj de bus
```

```
  ADCSC2 = 0x00;   // Default
  ADCSC1 = 0x40;   // Interrupción habilitada, modo de conversión simple, canal_0
}

//Rutina de retardo por Timer

void Delay(unsigned int dato){

  TPM1SC = 0x08;

  TPM1CNT = 0;

    while(dato){

      if(TPM1SC & 0x80){

         dato--;

         TPM1SC &= ~0x80;
      }


    }

         TPM1SC = 0x00;

}
////////////////////////////////////////////////////////////////////////////////////////////////////

//Atención a la interrupción por conversión

interrupt VectorNumber_Vadc void ISR_ADC (void){


  valor_convL = ADCRL;          //El valor de la conversión asígnalo a la variable

  bandera = 1;                  //Asigna a uno a la variable bandera

}
```

Se muestra el diagrama de flujo para emplear el convertidor analógico-digital:

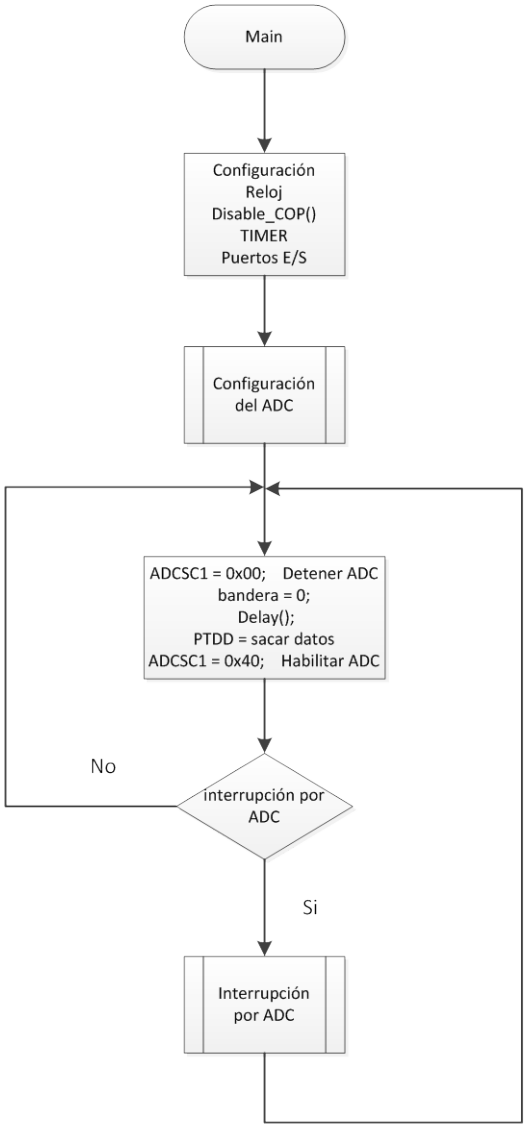


Figura 3.36 Configuración para la conversión analógica digital.

Se muestra el diagrama de flujo para atender la interrupción del convertidor analógico-digital:

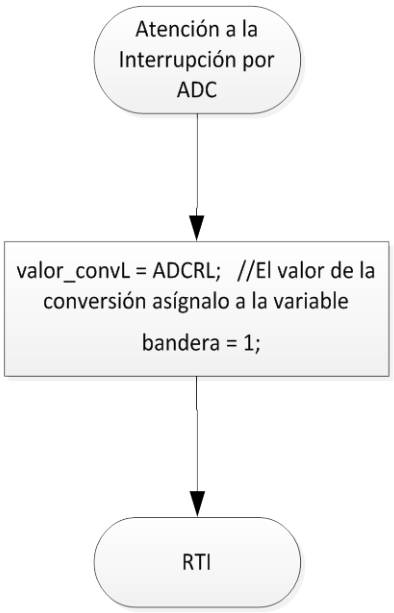


Figura 3.37 Atención a la interrupción por ADC.

Material y equipo para la práctica

1 Multímetro.
1 PC.
1 Tarjeta DEMOJM60.
1 Microcontrolador MC9S08JM60.
8 Led color rojo.
8 Resistencias de 330 Ω.
1 Protoboard.
Cables para alambrar.

Cuestionario preliminar

1. ¿Qué es una señal analógica?
2. ¿Qué es una señal continua?
3. ¿Las magnitudes físicas, con qué tipo de señal se representan?
4. ¿Qué tipo de convertidor analógico-digital dispone el microcontrolador MC9S08JM60?
5. ¿Cuál es el teorema de Nyquist-Shannon?

3.7 Práctica Número 7:

"Comunicación Serie"

Objetivo

Comprender el principio de funcionamiento del puerto serie asíncrono y los modos de configuración.

Introducción

El puerto serie es uno de los módulos más habituales que integran los microcontroladores. Su función básica es la de establecer una comunicación serie asíncrona con otros sistemas digitales, tanto para intercambiar información, como para volcar el código en la memoria flash del microcontrolador.

Funcionamiento

En la comunicación serie los datos se envían y se reciben uno a uno de forma independiente, ver figura 3.38.

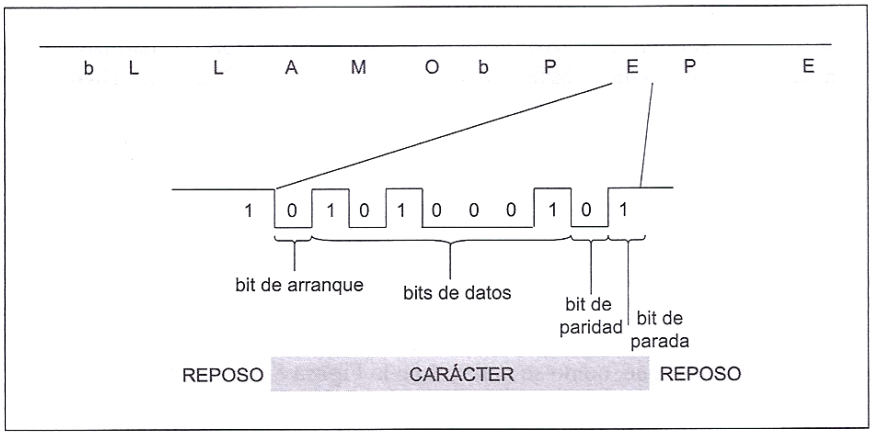


Figura 3.38 Ejemplo de una comunicación serial.

En reposo, el transmisor envía “unos” binarios continuamente, cuando requiere transmitir un caracter envía un bit de arranque (“cero” binario) y, a continuación, los bits de datos correspondientes al caracter codificado en ASCII. El número de datos depende de la codificación utilizada (7 u 8 son los valores habituales). También se suele incluir un bit adicional de paridad para detectar errores de transmisión. Cuando el receptor recibe el bit de arranque activa un reloj que muestra los bits de datos. Cuando termina la transmisión, el transmisor envía 1 o 2 bits de parada (unos binarios) y vuelve a la condición de reposo. En el ejemplo de la figura 3.38 este procedimiento se repite caracter a caracter hasta que se envía el mensaje entero.

La figura 3.39 muestra un esquema simplificado de un sistema de comunicación asíncrona, hay un transmisor, un canal de comunicaciones y un receptor. En el transmisor, el dato a transmitir se carga en el buffer de transmisión. A continuación, se encapsula en una trama o paquete que añade los bits de control (arranque, paridad y parada) y se transfiere a un registro de desplazamiento, que va generando una secuencia de bits a la velocidad establecida por el reloj de transmisión. Los bits generados se envían a la interfaz de comunicaciones que suele hacer una conversión de niveles de tensión para mejorar las condiciones de transmisión.

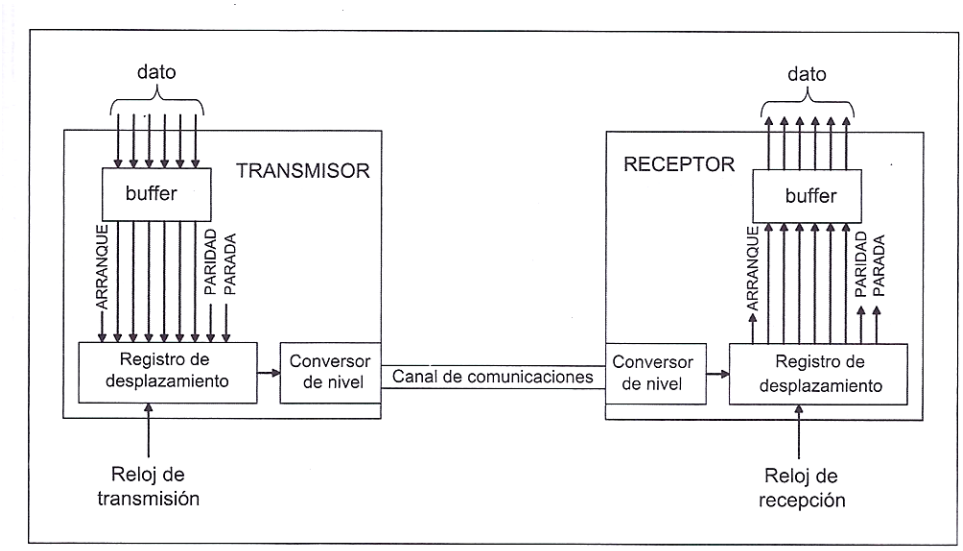


Figura 3.39 Esquema de un sistema de comunicación asíncrona.

En el receptor, se deshace la conversión de nivel y se almacena la trama en el registro de desplazamiento al ritmo de la frecuencia del reloj de recepción. Transmisor y receptor usan el mismo formato de trama y la misma frecuencia nominal de reloj. Cuando el dato es completamente recibido se extrae del registro de desplazamiento y se transfiere al buffer de recepción.

Un aspecto muy importante es que, en todo este proceso, los bits adicionales de arranque y parada son fundamentales para establecer y mantener una frecuencia de tiempo común entre transmisor y receptor. Esto es necesario para realizar correctamente la comunicación puesto que los relojes de transmisión y recepción, aunque tienen la misma frecuencia nominal, son independientes entre sí. La sincronización de los relojes se lleva a cabo tomando como referencia el flanco de bajada que se produce cuando comienza el bit de arranque, como se muestra en la figura 3.40.

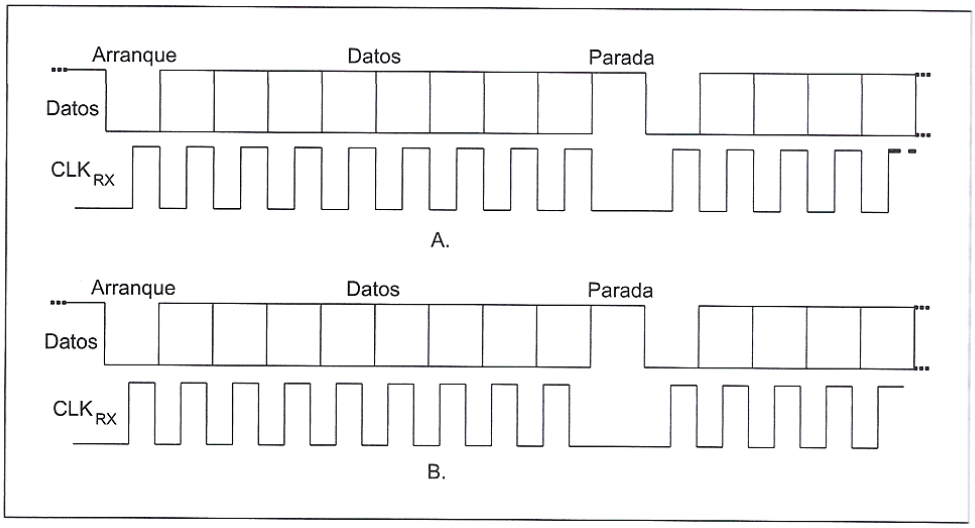


Figura 3.40 Sincronización en una comunicación asíncrona.

En una comunicación asíncrona, es posible que exista una cierta deriva entre los relojes del transmisor y receptor debido a que pueden sufrir cierta desviación con respecto a la frecuencia nominal de funcionamiento. Esto obliga a resincronizar los relojes de forma periódica, si se quiere garantizar un funcionamiento correcto, ya que, de lo contrario, se producirán errores. Un ejemplo en el que el reloj del receptor es ligeramente más rápido que el del transmisor se muestra en la figura 3.40, y como puede observarse, transcurrido

un cierto tiempo desde la sincronización, la captura de los bits de datos se realizaría instantes de tiempo en los que su valor es inestable. La longitud de las tramas de datos está pues limitada si se quiere permitir cierta tolerancia en las frecuencias de los relojes de transmisión y recepción, siendo lo habitual utilizar 7 y 8 bits.

La inserción de bits de arranque y parada cada cierto número de bits garantiza la existencia de flancos de bajada independientemente de los datos transmitidos, de forma que el receptor puede mantenerse sincronizado con el trasmisor.

En algunos protocolos de comunicaciones serie asíncrona se utiliza un caracter especial denominado *break,* que habitualmente se utiliza para llamar la atención del receptor. La condición de break se produce cuando la línea permanece al valor del bit de arranque por un tiempo superior al necesario para la transmisión de un caracter, lo cual es imposible en condiciones normales ya que hay al menos un bit con el valor del bit de parada por caracter transmitido.

En resumen, la necesidad de introducir bits adicionales en las comunicaciones asíncronas hace que éstas no sean adecuadas para altas velocidades, ya que son poco eficientes. Además, como ya se ha comentado, al ser los relojes de transmisión y recepción independientes, el tamaño de los caracteres no puede ser muy largo ya que conforme se incrementa el tamaño del caracter aumenta un posible desfase entre los relojes y, en consecuencia, la dificultad de recibir datos válidos.

La velocidad de transmisión hace uso de valores estándar, algunas velocidades típicas son 110, 300, 1200, 1400, 4800, 9600, 19200 bits por segundo. La unidad usada es bits por segundo, aunque a veces se puede encontrar otra muy utilizada en módems: el baudio. Se entiende por baudio el número de cambios de la señal por segundo; es importante saber que no siempre coincide el número de baudios con el de bits por segundo ya que, en ciertas modulaciones, se transfieren varios bits en un cambio de señal.

El puerto clásico, también llamado UART (*Universal Asynchronos Receiver/Transmitter, Transmisor-Receptor Asíncrono Universa*l), implementa un transmisor, un receptor, generadores de reloj, registros para configurar, entre otras cosas. El MC9S08JM60 está dotado de dos módulos hardware denominados SCI que facilitan la implementación de interfaces de comunicaciones serie asíncronas y ofrecen soporte para los protocolos y modos

de operación más habitualmente implementados sobre este tipo de interfaces. Ambos módulos SCI son iguales y pueden ser utilizados de manera independiente, disponiendo cada uno de ellos de un conjunto separado de registros operacionales para su control y configuración.

La figura 3.41, cada módulo SCI está constituido por tres bloques bien diferenciados: el bloque transmisor, receptor y el generador de baudios. Tanto el transmisor como el receptor pueden operar independientemente uno del otro, pero comparten el mismo generador de baudios, por lo que la velocidad (o tasa binaria) será la misma en transmisión y recepción.

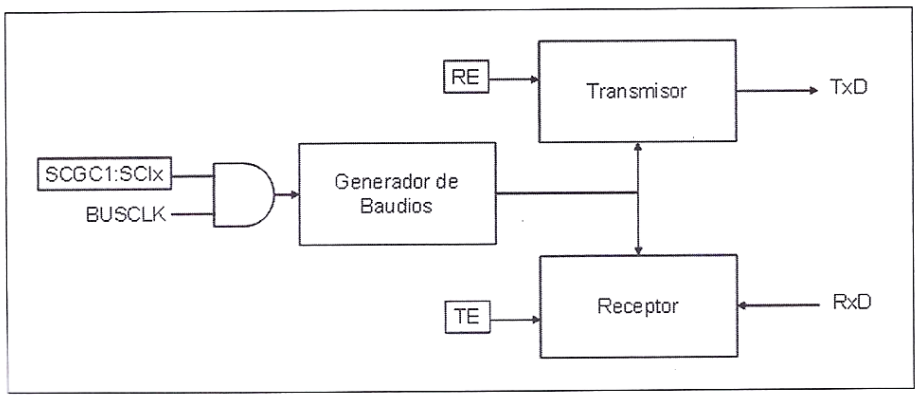


Figura 3.41 Diagrama de bloques del módulo de comunicación serie SCI.

Para el control de estos tres bloques, el microcontrolador dispone de 8 registros que permiten configurar su funcionamiento (registro de control), realizar operaciones (registro de datos) y verificar los datos (registro de estado). Estos 8 registros están duplicados para el módulo SCI1 y para el SCI2.

Generador de baudios

El generador de baudios es el subsistema encargado de crear los relojes que gobiernan la transmisión y recepción de bits, permitiendo de esta manera configurar la tasa binaria a la que operarán el transmisor y el receptor. Como se muestra en el diagrama de bloques de la figura 3.41**,** básicamente consiste en un contador/divisor de 13 bits programable que divide la frecuencia del reloj BUSCLK por un valor configurable, de forma que pueda ajustarse la tasa de bits que utilizan transmisor y receptor, lo que implica que una modificación en la

frecuencia del BUSCLK afecta a la velocidad con la que operan los módulos SCI.

La tasa binaria se ajusta mediante 13 bits situados en los registros SCIxBDH y SCIxBDL, viene dada por la siguiente expresión:

$$Baud\ Rate = \frac{f_{BUSCLK}}{SBR * 16}$$

Dónde:

f_{BUSCLK} : es la frecuencia configurada para el reloj BUSCLK en el subsistema de generación de reloj.

SBR es el valor de 13 bits configurado para el contador/divisor programable, que puede estar comprendido entre 1 y 8191. El valor 0 es especial, ya que detiene el funcionamiento del contador/divisor y, por lo tanto, el funcionamiento de los bloques transmisor y receptor. SCIxSBDH y SCIxSBDL deben ser escritos en este orden, ya que el valor del contador/divisor se actualiza cuando se realiza la escritura de SCIxSBDL.

Puesto que el registro SBR tiene una resolución limitada de 13 bits, es evidente que no pueda conseguirse de manera exacta cualquier valor de tasa binaria deseada. No obstante, el mecanismo de sincronización mediante bits de arranque y parada, utilizados en las interfaces serie asíncronos, permite la existencia de cierta tolerancia en la tasa de los extremos transmisor y receptor sin que se produzcan errores en la comunicación. La tolerancia permitida en el ajuste de la tasa binaria es aproximadamente de un 3 o 4%, que debe incluir tanto las posibles desviaciones de la frecuencia del reloj respecto a la ideal, como error cometido por la resolución del SBR.

Bloque transmisor

El bloque es independiente de los otros dos bloques aunque comparte alguno de los parámetros de configuración con el bloque receptor y puede activarse y desactivarse mediante el bit TE del registro SCIxC2. Una vez habilitado, el transmisor permanece en estado de espera, manteniendo la línea en condición de parada hasta que se envíen datos al buffer de transmisión.

Los datos son enviados al buffer de transmisión, por el código que se ejecuta en el microcontrolador mediante escrituras en el registro SCIxD.

El elemento central es un registro de desplazamiento (registro de transmisión) en el que, en cada transmisión, se almacenan los bits que se van a transmitir por cada caracter a enviar:

- Bit de arranque
- Bits de datos
- Bit de paridad (si procede)
- Bit de parada

El número de bits que se envían por cada caracter depende de las diferentes opciones de configuración seleccionadas mediante los correspondientes registros de control. Cuando el registro de transmisión se encuentra disponible para enviar un nuevo caracter, el valor almacenado en el registro de datos (SCIxD) se transfiere al de desplazamiento, y comienza un ciclo de operación en el cual se va desplazando sincronizadamente con el reloj suministrado por el generador de baudios, de forma que en la línea TxD van apareciendo sucesivamente el bit de arranque, los bits de datos (primero el LSB), y finalmente el bit de parrada, con lógica positiva o negativa, en función del bit TXINV del registro SCIxC3.

Una vez finalizada la operación, se transfiere un nuevo valor desde el registro de datos SCIxD y se comienza nuevamente un ciclo. Cada vez que se transfiere un valor disponible en el SCIxD al registro de desplazamiento, se activa el bit de estado TDRE del registro SCIxS1 para indicar que se puede escribir un nuevo valor en el registro de datos. Si cuando finaliza la transmisión en curso no se ha escrito un nuevo valor en el registro de datos, entonces el transmisor entra en estado de espera y se activa la bandera TC del registro SCIxS1 para indicar que la transmisión ha finalizado. Ambos eventos registro de datos vacío y transmisión finalizada pueden producir la interrupción asociada al transmisor si ella está habilitada mediante los bits TIE y TCIE del registro SCIxC2 respectivamente.

El módulo transmisor del SCI permite configurar el modo de transmisión a 8 o 9 bits por caracter, lo que se controla mediante la bandera M del registro de control SCIxC1. En el caso de transmitir 9 bits, el bit más significativo tiene que ser escrito en el bit T8 del registro SC1xC3 y, posteriormente, los 8 bit menos significativos deben ser escritos en el registro de datos (SCIxD). El transmisor del SCI incluye la posibilidad de generar automáticamente un bit de paridad para mejorar la fidelidad de las comunicaciones asíncronas, permitiendo detectar errores en la transmisión. Para habilitar la paridad es necesario activar el bit PE del registro SCIxC1. El tipo de paridad utilizado (par o impar) puede ajustarse mediante el bit PT del mismo registro. La configuración seleccionada mediante ambos bit afectará tanto al comportamiento del transmisor como al del receptor. En el caso de habilitar la paridad, ésta se generará automáticamente en el bit más significativo, por lo tanto, perderá su capacidad de llevar información, de forma que según se haya seleccionado el modo de transmisión con 8 o 9 bits, se tendrá 7 u 8 bits de datos respectivamente.

Cuando el transmisor es deshabilitado poniendo a 0 el bit TE del registro SCIxC2, no se desactiva inmediatamente, sino que primero finalizará la transmisión del caracter en curso para evitar la transmisión de un caracter erróneo. Una vez finalizada, la línea de salida pasa al estado de espera.

El transmisor del SCI también incluye facilidades para generar y detectar el caracter especial *break*, que fuerza al nivel de parada en la línea durante un periodo de tiempo superior al de un caracter. La generación de la condición *break* por el transmisor es disparada poniendo a 1 el bit SBK del registro SCIxC2. Para enviar un *break*, el programa normalmente verificara que el registro de datos se encuentre vacío mediante el bit TDRE y, tras la activación de SBK, un caracter de *break* quedará en la cola para ser enviado en cuando finalice la transmisión en curso.

Si en ese momento SBK permanece aún activo, afecta a la longitud del caracter de *break* que será de 10 y 11 bits respectivamente. Adicionalmente se pueden generar caracteres de *break* de 13 o 14 bits si se activa el bit BRK13 del registro SCIxS2.

Bloque receptor

Puede activarse o desactivarse independientemente de los otros dos bloques mediante el bit RE del registro SCIxC2. Al igual que el transmisor, puede operar con caracteres de 8 o 9 bits. Los valores recibidos por la entrada RxD pueden ser enviados mediante el bit RXINV del registro SCIxS2.

El bloque receptor detecta el flanco provocado en línea por la condición de arranque y lee los bits sucesivos a la velocidad marcada por el generador de baudios, mediante el registro de desplazamiento. Una vez capturado el número adecuado de bits en el registro de desplazamiento, el caracter recibido es transferido al registro datos SCIxD, y el bit RDRF del registro de estado SCIxS1 es puesto a uno para indicar que existe un dato válido en SCIxD que puede ser leído por el programa en ejecución. Si el registro SCIxD ya había un dato disponible de una recepción anterior que no ha sido leído todavía RDRF estaría ya activo antes de recibir el caracter, entonces el nuevo dato recibido se pierde y se pone a uno el bit de estado OR del registro SCIxS1, que indica que se ha producido una pérdida de un caracter. El bit RDRF se desactiva automáticamente cuando se leen SCIxS1 y SCIxD desde el programa en ejecución, lo cual implica que se ha leído el caracter recibido. La activación del bit RDRF puede dar lugar a una interrupción del receptor si se habilita mediante el correspondiente bit RIE en el registro SCIxC2.

El receptor incluye los mecanismos necesarios para conseguir una buena sincronización con los bits que están siendo recibidos, lo cual es importante para garantizar la integridad de la comunicación. Si además de la condición de arranque se detectan algunos flancos de bajada adicionales, el receptor los aprovecha para mantener la sincronización. También se incluyen mecanismos para la detención de flancos espurios cercanos al instante en el que se capturan los bits de entrada, indicando su presencia mediante la activación del bit NF del registro SCIxS1. De la misma forma, se verifica automáticamente la correcta recepción del bit de parada, indicando un error de trama mediante el bit FE de SCIxS1 en caso de que no se reciba correctamente el bit, que indicará un problema con el sincronismo entre el transmisor y el receptor que están llevando a cabo la comunicación. Cuando se produce un error de trama, el receptor es deshabilitado hasta que se borre el bit FE por programa.

El receptor del SCI incluye el hardware necesario para la comprobación del bit de paridad, lo que permite detectar errores en la transmisión de los bits. Para habilitar la verificación de paridad es necesario activar el bit PE del registro SCIxC1. El tipo de paridad utilizando par o impar puede ajustarse mediante el bit PT del mismo registro. Como se mencionó, estos bits afectan tanto a la generación automática del bit de paridad en el transmisor como para sondear el mismo en el bloque receptor. En caso en que se detecte un error de paridad, el receptor activa la bandera PF del registro SCIxS1, pudiendo dar lugar también a una interrupción por hardware.

- ***Registros de tasa de baudios (SCIxBDH:SCIxBDL)***

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 SCIxBDH. Para el caso de programación en C, es posible escribir sobre el registro SCIxBD, el cual agrupa los registros anteriores.

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	LBKDIE	RXEDGIE	0	SBR[12:8]				
Reset	0	0	0	0	0	0	0	0

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	SBR[7:0]							
Reset	0	0	0	0	0	0	0	0

LBKDIE

Bit para habilitar un evento de interrupción debido a la recepción de un caracter 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 tasa de baudios.

Cuando el valor de estos bits es cero, el generador de la tasa de baudios se detiene con el fin de ahorrar energía. La ecuación para el cálculo de la tasa de baudios es:

$$Tasa\ de\ Baudios = \frac{Reloj\ de\ BUS}{SBR * 16}$$

Por ejemplo, si se necesita una tasa de baudios de 9600 y se está trabajando con un reloj de BUS interno a 8MHZ, el SBR será de:

$$SBR = \frac{8\ MHZ}{16 * 9600\ Hz}$$

$$SBR = 52.08$$

El valor anterior se puede aproximar a 52, sin ningún deterioro del sincronismo de la comunicación.

- ***Registro de control 1 del SCI (SCIxC1)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	LOOPS	SCISWAI	RSRC	M	WAKE	ILT	PE	PT
Reset	0	0	0	0	0	0	0	0

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 pins 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 del SCI podría despertar la máquina
1: El módulo SCI está detenido en modo WAIT

RSRC

La operación con este bit tiene sentido si el bit LOOP está en “1”. En este modo 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 + bit 9 + 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 y 11 bits necesarios como nivel IDLE para la lógica de detección

0: El contador de bits, del caracter para la condición de IDLE, comienza después del bit de start
1: El contador de bits, del caracter 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 muestrar.

0: Paridad par (Even)
1: Paridad impar (Odd)

- ***Registro de control 2 del SCI (SCIxC2)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura Escritura	TIE	TCIE	RIE	ILIE	TE	RE	RWU	SBK
Reset	0	0	0	0	0	0	0	0

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: Deshabilita 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: Deshabilita 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: Deshabilita 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: Deshabilita 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: Deshabilita la operación del transmisor
1: Habilita la operación del transmisor

RE

Bit para habilitar la operación del receptor

0: Deshabilita 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 caracter *break*. Un caracter *break* consiste en el apilamiento de “0” 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)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	TDRE	TC	RDRF	IDLE	OR	NF	FE	PF
Escritura								
Reset	0	0	0	0	0	0	0	0

TDRE

Bandera que indica un dato del *buffer* de transmisión ha pasado al *shift register*. Este evento revela 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, 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 caracter de *break* escribiendo un “1” en el bit SBK.

0: Transmisor enviando un dato
1: Transmisor en modo IDLE

RDRF

Bandera que indica, el *buffer* de recepción está lleno. El evento revela cuándo un dato recibido en el *shift register* ha pasado al registro de datos de SCI (SCIxD). Para poner el bit a cero, 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 caracter de sólo “1’s”. Para poner le bit a cero la bandera de IDLE es necesario leer el registro SCIxS1 y luego leer el registro de datos SCIxD.

0: No se ha detectado un caracter IDLE
1: Se ha detectado un caracter IDLE

OR

Bandera de sobre carrera del receptor (*Receiver Overrun*). Esta bandera indica cuándo 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 para poner el bit a cero el bit OR es necesario leer el registro SCIxS1 y luego leer el registro de datos SCIxD.

0: No se ha detectado una sobre carrera en el receptor
1: Se ha detectado una sobre carrera en el receptor y se ha perdido el último dato

NF

Todo bit de start es muestreado siete veces 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 poner el bit a cero 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 poner el bit a cero 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 cuándo el bit de paridad no coincide con el valor de la paridad pactada. Para poner el bit a cero 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

- ***Registro de estado 2 del SCI (SCIxS2)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	LBKDIF	RXEDGIF	0	RXINV	RWUID	BRK_{13}	LBKDE	RAF
Escritura								
Reset	0	0	0	0	0	0	0	0

LBKDIF

Bandera que indica cuándo se ha detectado un caracter de *break* en la línea. Esta bandera se puede regresar a su valor original escribiendo un "1" en ella.

0: No se ha detectado un caracter de *break* en la línea
1: Se ha detectado un caracter de *break* en la línea

RXEDGIF

Bit para indicar cuándo ha ocurrido un evento de interrupción por un flanco recibido en el pin de RxD. Puede ser de bajada (RXINV = 0) o de subida (RXINV = 1). Esta bandera se puede regresar a su valor original 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 *Idle Wakeup*. Indica cuando un caracter IDLE ha puesto la bandera 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 caracter IDLE
1: Durante el estado de atento del receptor (RWU = "1"), el bit IDLE ha sido puesto a "1" en la detección de un caracter IDLE

BRK_{13}

Bit para seleccionar la longitud de un caracter de *break*.

0: El caracter de *break* será de 10 bits (11 si M =1)
1: El caracter de *break* será de 13 bits (14 si M =1)

LBKDE

Bit para habilitar un caracter *break* en la línea. Mientras este bit sea "1", el sistema previene que el bit FE (*Famming Error*) y RDRF sean puestos a "1".

0: Un caracter de *break* de 10 bits es detectado (11 si M =1)
1: Un caracter 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 verificar cuándo un caracter 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)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	R_8	T_8	TXDIR	TXINV	ORIE	NEIE	FEIE	PEIE
Escritura								
Reset	0	0	0	0	0	0	0	0

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 SCIxD y luego definir el bit T8.

TXDIR

Bit para definir la dirección del pin de TxD, en modo de un solo 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: Deshabilita interrupción por *Receiver Overrun*
1: Habilita interrupción por *Receiver Overrun*

NEIE

Bit para habilitar un evento de interrupción por *Noise Error*.

0: Deshabilita interrupción por *Noise Error*
1: Habilita interrupción por *Noise Error*

FEIE

Bit para habilitar un evento de interrupción por *Framming Error*.

0: Deshabilita interrupción por *Framming Error*
1: Habilita interrupción por *Framming Error*

PEIE

Bit para habilitar un evento de interrupción por *Parity Error*.

0: Deshabilita interrupción por *Parity Error*
1: Habilita interrupción por *Parity Error*

- ***Registro de datos del SCI (SCIxD)***

	Bit 7	6	5	4	3	2	1	Bit 0
Lectura	R_7	R_6	R_5	R_4	R_3	R_2	R_1	R_0
Escritura	T_7	T_6	T_5	T_4	T_3	T_2	T_1	T_0
Reset	0	0	0	0	0	0	0	0

Rn: Dato Recepción
Tn: Dato Transmisión

Desarrollo

El circuito de la figura 3.42 detalla la aplicación a implementar, como ejercicio del manejo de la comunicación serial con una PC.

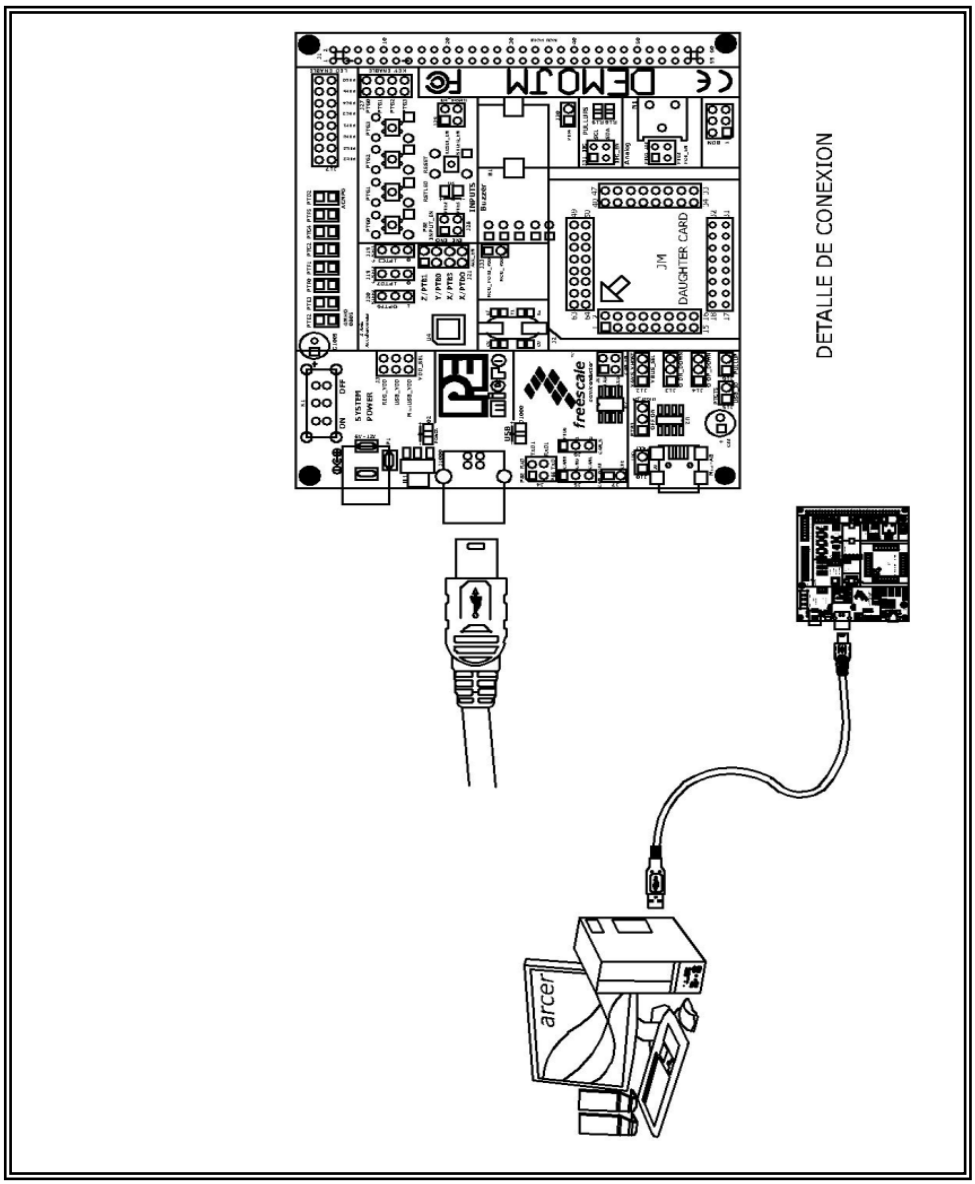


Figura 3.42 Circuito para transmitir vía serial a la computadora.

1. Capturar el siguiente programa para realizar la comunicación serial entre el microcontrolador y la PC, atra vez de la herramienta *P&E Embedded Multilink Toolkit.*

```
/*****************************************************************************************************
*********

Programa: HOLA MUNDO (Transmisión serial ocupando las herramientas de
Freescale)

Autor: Oscar M. Espinoza

Fecha: Diciembre 12 2012

Versión: 1.0


*****************************************************************************************************
**********/

#include <hidef.h>                  /* Macro para habilitar interrupciones */
#include "derivative.h"             /* Incluye declaración de periféricos */

//Inicialización del cola circular o FIFO

#define SERIAL_BUFFER_SIZE 32


volatile unsigned char _serial_rx_buffer[SERIAL_BUFFER_SIZE];     //Buffer de
recepción
volatile unsigned char _serial_tx_buffer[SERIAL_BUFFER_SIZE];     //Buffer de
transmision

static volatile unsigned char _serial_rx_nxt=0;
static volatile unsigned char _serial_rx_rd=0;

static volatile unsigned char _serial_tx_rd=0;
static volatile unsigned char _serial_tx_nxt=0;

static volatile unsigned char _serial_tx_active=0;
static volatile unsigned char _serial_tx_buffer_full=0;
static volatile unsigned char _serial_rx_buffer_overflow=0;
```

```
#define Disable_COP() SOPT1 &= 0x3F

/*****************************************************************************************************
*********/
/***Prototipo de Funciones***/

void Init_SCI(unsigned char baudrate);                  //Inicializa SCI

void serial_send(unsigned char *ptrtochar);             //Manda información

/*****************************************************************************************************
********/

void main(void) {

  MCGC1 = 0x04;                          //Frecuencia aproximada a 8 MHz
  MCGC2 = 0x40;
  MCGC3 = 0x01;

  Disable_COP();

  EnableInterrupts;                     /* Habilita interrupciones */
  /* include your code here */

  Init_SCI(52);                         // baud 9600

  for(;;) {

    serial_send("\r\nHola! ***** Diciembre 9 2012 ;-)\r\n");  //Envío a la PC


  }

}
```

```
/*****************************************************************************************************
*********/

//Inicializa transmision serial

void Init_SCI(unsigned char baudrate){

  _serial_rx_nxt=0;
  _serial_rx_rd=0;
  _serial_tx_rd=0;
  _serial_tx_nxt=0;
  _serial_tx_active=0;
  _serial_tx_buffer_full=0;

  SCI1BD = baudrate;

  SCI1C1 = 0;
  SCI1C3 = 0;
  SCI1S2 = 0;

  SCI1C2 = 0x2C;

}


//Llena buffer circular

void serial_send(unsigned char *ptrtochar)
{

  unsigned char i;

  i=0;

  while((!_serial_tx_buffer_full))
  {
     //Si está inactivo debe haber espacio de sobra (de hecho debe estar vacio),
    // luego no se bloqueará

     _serial_tx_buffer[_serial_tx_nxt]=ptrtochar[i];
     _serial_tx_nxt++;
```

```
    if (_serial_tx_nxt>=SERIAL_BUFFER_SIZE)

      _serial_tx_nxt=0;

    i++;

    if (_serial_tx_nxt==_serial_tx_rd)

      _serial_tx_buffer_full=1;

  }

  if (!_serial_tx_active){

    _serial_tx_active=1;

    if (SCI1S1_TDRE){

      //Si el transmisor estaba vacío, transmite el primer caracter.
      // Escribe el caracter. Al hacerlo, se pone a cero la bandera TDR

      SCI1D= _serial_tx_buffer[_serial_tx_rd];
      _serial_tx_rd++;

      if (_serial_tx_rd>=SERIAL_BUFFER_SIZE)

        _serial_tx_rd=0;

        _serial_tx_buffer_full=0; //Se ha enviado un caracter, luego ya no está
vacío
    }
      SCI1C2_TIE=1;
  }


}
```

```
/*****************************************************************************************************
********/

//Atención a la interrupción por transmisión y envía los datos almacenados por el
buffer circular

interrupt VectorNumber_Vsci1tx void SCI_TX_ISR_Handler(void)
{
  if ((SCI1C2_TIE)&&(SCI1S1_TDRE)) {

     if ((!_serial_tx_buffer_full)&&(_serial_tx_nxt==_serial_tx_rd))
     {
              SCI1C2_TCIE=1;                  //Habilita la interrupción
              SCI1C2_TIE=0;                   //Deshabilita más interrupciones
              _serial_tx_active=0;
     }
     else {



// Escribe el caracter. Al hacerlo, se pone a cero la bandera TDR

            SCI1D = _serial_tx_buffer[_serial_tx_rd];
            _serial_tx_rd++;

              if (_serial_tx_rd>=SERIAL_BUFFER_SIZE)

                _serial_tx_rd=0;

                _serial_tx_buffer_full=0;
     }
  }

}
```

Material y equipo para la práctica

1 Multímetro.
1 PC.
1 Tarjeta DEMOJM60.
1 Microcontrolador MC9S08JM60.

Cuestionario Preliminar

1. ¿Qué es una comunicación serial?
2. ¿Qué es una comunicación en paralelo?
3. ¿Qué es una comunicación asíncrona?
4. ¿Qué es un canal de comunicaciones?
5. ¿Qué es un baudio?
6. ¿Qué es un DCE en comunicaciones?
7. ¿Qué es un DTE en comunicaciones?
8. ¿Cuáles son las velocidades estándar para transmisión serial?
9. ¿Qué es un bit de paridad?
10. ¿Qué es un buffer circular?
11. ¿Qué es el código ASCII?
12. ¿Qué es el estándar RS-323?

Conclusiones

Las conclusiones a las que se ha llegado al diseñar e implementar las prácticas de esta tesis usando la tarjeta DEMOJM y usando la tecnología Flexis de Freescale, son las siguientes:

- Las prácticas desarrolladas lograron cumplir con todos los objetivos especificados, para el laboratorio de Diseño Mecatrónico tomando en cuenta el plan de estudios 2007 correspondiente a la misma materia de la carrera de Ingeniería Mecánica de la Fes Aragón-UNAM.

- Las características pertenecientes a la familia Flexis de Freescale que ha lanza al mercado un nuevo conjunto de microcontroladores basados en arquitecturas de 8 y 32 bits ya existentes como son S08 y Coldfire con la particularidad fundamental de ser compatibles entre ellos. Esto permite la migración de aplicaciones de 8 a 32 bits o viceversa con sólo unos 'clicks' del ratón, posibilitando la reutilización completa del código desarrollado.

  El programador/diseñador podrá utilizar la misma herramienta de desarrollo hardware DEMOJM y el software CodeWarrior para microcontroladores V6.x en su aplicación gracias a la compatibilidad de encapsulado y pines, y con mínimos cambios o en casos ninguno en el código, gracias a la total compatibilidad de periféricos.

  La tecnología Flexis abre las puertas a la generación de aplicaciones que podrán ser ejecutadas en dos procesadores muy diferentes, en función de sus necesidades y con tamaños de memoria similares, a pesar del cambio de una arquitectura de 8 a 32 bits, o viceversa. Esto es posible gracias a las características de los procesadores.

  También ofrece poder mantener los métodos de trabajo y gestión de proyectos de una manera cómoda y eficaz, permitiéndole la migración de proyectos de 8 a 32 bits y viceversa o simplemente de cambio de tamaño de memoria de una manera sencilla, rápida y eficiente.

- La de DEMOJM como tarjeta de control:

  La activación de dispositivos, todo o nada, correspondiente a la práctica número dos, configuración de puertos entrada/salida. Incluyendo también la configuración de interrupciones externas IRQ de la práctica tres.

Control de motores y servomotores de corriente directa, para el direccionamientos de elementos mecatrónicos, articulaciones de robots móviles, abertura de compuertas eléctricas, etc. para la cual fueron desarrolladas las practicas número 4 y 5.

La digitalización de señales analógicas por medio del convertidor del microcontrolador MC9S08JM60 correspondiente a la práctica número 6.

La comunicación serie entre el microcontrolador y la computadora, envió y transmisión de datos entre ellos, ejemplo de ello la practica número 7.

- Las prácticas desarrolladas con el MC9S08JM60 de la familia Flexis de Freescale pueden ser adaptadas a otros microcontroladores, ya que la metodología de programación es retomada para la programación de microcontroladores en general en lenguaje de programación de alto nivel.

- El manual de prácticas fue desarrollado de tal forma que el alumno de la materia de Diseño Mecatrónico que no está familiarizado con la programación de sistemas embebidos y electrónica digital, pueda desarrollar sistemas de control, mecatrónico y comunicación de una manera lógica y secuencial.

- Trabajo a futuro implementar prácticas de comunicaciones entre un microcontrolador y display LCD, I^2C y SPI así como la introducción de sistemas de tiempo real (RTOS).

Glosario

ABS: *Es un dispositivo utilizado en aviones y en automóviles, para evitar que los neumáticos pierdan la adherencia con el suelo durante un proceso de frenado. El sistema fue desarrollado inicialmente para los aviones, los cuales acostumbran a tener que frenar fuertemente una vez han tomado tierra. En 1978, Bosch hizo historia cuando introdujo el primer sistema electrónico de frenos antibloqueo. Esta tecnología se ha convertido en la base para todos los sistemas electrónicos que utilizan de alguna forma el ABS, como por ejemplo los controles de tracción y de estabilidad.*

A/D: *Es un dispositivo electrónico capaz de convertir una entrada analógica de voltaje en un valor binario, Se utiliza en equipos electrónicos como computadora, grabadores de sonido y de vídeo, y equipos de telecomunicaciones. La señal analógica, que varía de forma continua en el tiempo, se conecta a la entrada del dispositivo y se somete a un muestreo a una velocidad fija, obteniéndose así una señal digital a la salida del mismo.*

BDM: *Es un programador BDM de código abierto, el cual se utiliza para programar microcontroladores de Freescale de las familias S08, y RS08.*

BIT: *Un bit es un dígito del sistema de numeración binario (1 o 0). El bit es la unidad mínima de información empleada en informática, en cualquier dispositivo digital, o en la teoría de la información.*

BIG ENDIAN: *En el formato "Big Endian", el byte de mayor peso se almacena en la dirección más baja de memoria y el byte de menor peso en la dirección más alta.*

BYTE: *Es una secuencia de bits contiguos, cuyo tamaño depende del código de información o código de caracteres en que sea definido. Los bytes de 8 bits a menudo se llaman "octetos" en contextos formales como los estándares industriales, así como en redes informáticas y telecomunicaciones para evitar confusiones sobre el número de bits implicados. La mitad de un byte de ocho bits se llama nibble o un dígito hexadecimal.*

CAD: *El diseño asistido por computadora, más conocido por sus siglas en inglés (computer-aided design), es el uso de un amplio rango de herramientas computacionales que asisten a ingenieros, arquitectos y a otros profesionales del diseño en sus respectivas actividades.*

CAM: *Fabricación asistida por computadora, también conocida por las siglas en inglés (computer-aided manufacturing), implica el uso de computadoras y tecnología de cómputo para ayudar en la fase directa de la manufactura del producto, es un puente entre el Diseño Asistido por Computadora CAD y el lenguaje de programación de las máquinas herramientas con una intervención mínima del operario.*

CPU: *Unidad Central de Proceso o simplemente, es el componente principal del ordenador y oros dispositivos programables, que interpreta las instrucciones contenidas en los programas y procesa los datos.*

COP: *Módulo del Adecuado Funcionamiento de la Computadora o Watchdog. El módulo del adecuado funcionamiento de la computadora (COP), también conocido como watchdog, es un módulo que tiene como función básica provocarle un reset al microcontrolador, en caso de que no reciba una notificación antes de un tiempo determinad. Este módulo se encuentra ubicado en el control del sistema del HCS08.*

DEDUG: *Depuración de programas es el proceso de identificar y corregir errores de programación.*

DMA: *El acceso directo a memoria (DMA, del inglés direct memory access) permite a cierto tipo de componentes de una computadora acceder a la memoria del sistema para leer o escribir independientemente de la unidad central de procesamiento (CPU) principal. Muchos sistemas hardware utilizan DMA, incluyendo controladores de unidades de disco, tarjetas gráficas y tarjetas de sonido. DMA es una característica esencial en todos los ordenadores modernos, ya que permite a dispositivos de diferentes velocidades comunicarse sin someter a la CPU a una carga masiva de interrupciones.*

EMBEBIDO: *Se conoce como sistema embebido a un circuito electrónico computarizado que está diseñado para cumplir una labor especifica en el producto.*

EEPROM: *(Electrical Erasable Programable Read Only Memory): tipo de memoria de almacenamiento que no pierde su contenido al ser desenergizada, borrable y programable por medio eléctrico.*

EPROM: *Son las siglas de Erasable Programmable Read-Only Memory (ROM programable borrable). Es un tipo de chip de memoria ROM no volátil. Se programan mediante un dispositivo electrónico que proporciona voltajes superiores a los normalmente utilizados en los circuitos electrónicos. Una vez programada, una EPROM se puede borrar solamente mediante exposición a una fuerte luz ultravioleta.*

FLASH: *Tipo de memoria no volátil que se borra y programa eléctricamente; es un tipo EEPROM de menor precio que puede ser borrada en grandes bloques.*

ETHERNET: *Es un estándar de redes de área local para computadores con acceso al medio por contienda CSMA/CD. CSMA/CD (Acceso Múltiple por Detección de Portadora con Detección de Colisiones), es una técnica usada en redes Ethernet para mejorar sus prestaciones.*

HARDWARE: S*e refiere a todas las partes tangibles de un sistema informático; sus componentes son: eléctricos, electrónicos, electromecánicos y mecánicos.*

HMI: *Interfaz de usuario por sus siglas en idioma inglés, (Human Machine Interface) que se usa para referirse a la interacción entre humanos y máquinas; Aplicable a sistemas de Automatización de procesos.*

IEEE: *Corresponde a las siglas de (Institute of Electrical and Electronics Engineers) en español Instituto de Ingenieros Eléctricos y Electrónicos, una asociación técnico-profesional mundial dedicada a la estandarización, entre otras cosas. Con cerca de 400.000 miembros y voluntarios en 160 países,es la mayor asociación internacional sin ánimo de lucro formada por profesionales de las nuevas tecnologías, como ingenieros eléctricos, ingenieros en electrónica, científicos de la computación, ingenieros en informática, matemáticos aplicados, ingenieros en biomédica, ingenieros en telecomunicación e ingenieros en Mecatrónica.*

I^2C: *(Inter-Integrated Circuit); estándar de comunicaciones serial síncrona entre periféricos, que usa dos líneas SDA y SCL.*

MP3: *MPEG-1 Audio Layer III o MPEG-2 Audio Layer III, más comúnmente conocido como MP3, es un formato de compresión de audio digital patentado que usa un algoritmo con pérdida para conseguir un menor tamaño de archivo. Es un formato de audio común usado para música tanto en ordenadores como enreproductores de audio portátil.*

LABVIEW: *(acrónimo de Laboratory Virtual Instrumentation Engineering Workbench) es una plataforma y entorno de desarrollo para diseñar sistemas, con un lenguaje de programación visual gráfico. Recomendado para sistemas hardware y software de pruebas, control y diseño, simulado o real y embebido, pues acelera la productividad. El lenguaje que usa se llama lenguaje G, donde la G simboliza que es lenguaje Gráfico.*

LITTLE ENDIA: *Significa que el byte de menor peso se almacena en la dirección más baja de memoria y el byte de mayor peso en la más alta.*

LCD: *Una pantalla de cristal líquido o LCD (sigla del inglés liquid crystal display) es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora. A menudo se utiliza en dispositivos electrónicos de pilas, ya que utiliza cantidades muy pequeñas de energía eléctrica.*

LED: *Un led (de la sigla inglesa LED: Light-Emitting Diode: 'diodo emisor de luz', también 'diodo luminoso') es un diodo semiconductor que emite luz. Se usan como indicadores en muchos dispositivos y en iluminación.*

PID: *Un PID es un mecanismo de control por realimentación que calcula la desviación o error entre un valor medido y el valor que se quiere obtener, para aplicar una acción correctora que ajuste el proceso. Elalgoritmo de cálculo del control PID se da en tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor Proporcional determina la reacción del error actual. El Integral genera una corrección proporcional a la integral del error, esto nos asegura que aplicando un esfuerzo de control suficiente, el error de seguimiento se reduce a cero. El Derivativo determina la reacción del tiempo en el que el error se produce. La suma de estas tres acciones es usada para ajustar al proceso vía un elemento de control.*

PLC: *Los controladores lógicos programables o PLC (programmable logic controller en sus siglas en inglés) son dispositivos electrónicos muy usados en automatización industrial. Los PLC sirven para realizar automatismos; son dispositivos electrónicos que reproducen programas informáticos, que permiten controlar procesos. Estos equipos pueden contar tanto con salidas como entradas del tipo Analógico y/o Digital. Su costo tiende a ser moderado para sus grandes aplicaciones y suplantan completamente a la lógica cableada. Dejando de esta manera solo elementos de potencia.*

PWM: *La modulación por ancho de pulsos (también conocida como PWM, siglas en inglés de pulse-width modulation) de una señal o fuente de energía es una técnica en la que se modifica el ciclo de trabajo de una señal periódica (una senoidal o una cuadrada, por ejemplo), ya sea para transmitir información a través de un canal de comunicaciones o para controlar la cantidad de energía que se envía a una carga.*

POLLING: *Es una técnica software en la que el microcontrolador pregunta constantemente al periférico si necesita ser atendido.*

RAM: *La memoria de acceso aleatorio (en inglés: random-access memory) se utiliza como memoria de trabajo para el sistema operativo, los programas y la mayoría del software. Es allí donde se cargan todas las instrucciones que ejecutan el procesador y otras unidades de cómputo. Se denominan «de acceso aleatorio» porque se puede leer o escribir en una posición de memoria con un tiempo de espera igual para cualquier posición, no siendo necesario seguir un orden para acceder a la información de la manera más rápida posible.*

RESOLUCION: *Indica el número de valores discretos que este puede producir sobre un rango de valores de voltaje. Generalmente es expresado en bits. Por ejemplo, un convertidor que codifica una entrada analógica de 1 a 256 valores discretos (0… 255) tiene una resolución de 8 bits: o sea, 2 elevado a 8.*

ROM: *La memoria de sólo lectura, conocida también como ROM (acrónimo en inglés de read-only memory), es un medio de almacenamiento utilizado en ordenadores y dispositivos electrónicos, que permite sólo la lectura de la información y no su escritura, independientemente de la presencia o no de una fuente de energía.*

RTOS: *(Real Time Operating System), programa residente que controla el funcionamiento de varias tareas, dando a cada una la sensación de simultaneidad.*

SDRAM: *Synchronous Dynamic Random Access Memory (SDRAM) es una memoria dinámica de acceso aleatorio DRAM que tiene una interfaz síncrona. Tradicionalmente, la memoria dinámica de acceso aleatorio DRAM tenía una interfaz asíncrona, lo que significaba que el cambio de estado de la memoria se efectúa un cierto tiempo (marcado por las características de la memoria) desde que cambian sus entradas. En cambio, en las SDRAM el cambio de estado tiene lugar en un momento señalado por una señal de reloj y, por lo tanto, está sincronizada con el bus de sistema del ordenador. El reloj también permite controlar una máquina de estados finitos interna que controla la función de "pipeline" de las instrucciones de entrada.*

SRAM: *Static Random Access Memory (SRAM), o Memoria Estática de Acceso Aleatorio es un tipo de memoria basada en semiconductores que a diferencia de la memoria DRAM, es capaz de mantener los datos, mientras esté alimentada, sin necesidad de circuito de refresco. Sin embargo, sí son memorias volátiles, es decir que pierden la información si se les interrumpe la alimentación eléctrica.*
No debe ser confundida con la SDRAM (Syncronous DRAM).

USART: *(Universal Asynchonous Receiver Transmitter), estándar de intercambio de datos que permite transmisión y recepción de datos de forma full-duplex a diferentes velocidades de configuración.*

USB: *El puerto USB o Universal Serial Bus (Bus Universal en Serie) es un puerto diseñado para conectar varios periféricos a una computadora. El puerto USB se encuentra en todas las computadoras modernas. Hay algunos conectores diferentes que se usan para conectar los dispositivos.*

SCADA: *(Acrónimo de Supervisory Control And Data Acquisition (Supervisión, Control y Adquisión de Datos) es un software para ordenadores que permite controlar y supervisar procesos industriales a distancia. Facilita retroalimentación en tiempo real con los dispositivos de campo (sensores y actuadores) y controlando el proceso automáticamente. Provee de toda la información que se genera en el proceso productivo (supervisión, control calidad, control de producción, almacenamiento de datos, etc.) y permite su gestión e intervención.*

SEÑALES ESPURIAS: *Energía de la señal no deseado en cualquier frecuencia a la salida de un dispositivo que no estuvo presente en la entrada.*

SOFTWARE: *Se conoce como software al equipamiento lógico o soporte lógico de un sistema informático, comprende el conjunto de los componentes lógicos necesarios que hacen posible la realización de tareas específicas, en contraposición a los componentes físicos, que son llamados hardware.*

SPI: *El Bus SPI (del inglés Serial Peripheral Interface) es un estándar de comunicaciones, usado principalmente para la transferencia de información entre circuitos integrados en equipos electrónicos. El bus de interfaz de periféricos serie o bus SPI es un estándar para controlar casi cualquier dispositivo electrónico digital que acepte un flujo de bits serie regulado por un reloj.*

Vmax: *Voltaje máximo.*

Vmin: *Voltaje mínimo.*

VOIP: *Voz sobre Protocolo de Internet, también llamado Voz sobre IP, Voz IP, VozIP, (VoIP por sus siglas en inglés, Voice over IP), es un grupo de recursos que hacen posible que la señal de voz viaje a través de Internet empleando un protocolo IP (Protocolo de Internet). Esto significa que se envía la señal de voz en forma digital, en paquetes de datos, en lugar de enviarla en forma analógica a través de circuitos utilizables sólo por telefonía convencional como las redes PSTN (sigla de Public Switched Telephone Network, Red Telefónica Pública Conmutada).*

Bibliografía

[1] Díaz Estrella Antonio, Teoría y Diseño con Microcontroladores de Freescale.

[2] Vesga Ferreira Juan Carlos, Microcontroladores Motorola–Freescale Programación, familias y sus distintas aplicaciones en la industria.

[3] Enríquez Harper Gilberto, Control de Motores Eléctricos.

[4] Enríquez Harper Gilberto, Motores Eléctricos en la Industria.

[5] Mandonado Pérez Enrique, Autómatas Programables y Sistemas de Automatización.

[6] Galeano Gustavo, Programación de Sistemas Embebidos en C.

[7] Stan Kelly-Bootle y Bob Foeler, 68000 68010/68020 Arquitectura y programación en ensamblador.

[8] Valvano, Jonathan W., Introducción a los Sistemas de microcomputadora embebido: Simulación de Motorola 6811 y 6812.

Páginas Electrónicas

http://cache.freescale.com/files/microcontrollers/doc/data_sheet/MC9S08JM60.pdf?fsrch=1&sr=2
Consultado en 25 de Junio del 2012, 14:00 hrs.

http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=DEMOJM
Consultado en 2 de Septiembre del 2012, 13:00 hrs.

http://www.cursomicros.com/
Consultado en 14 de Septiembre del 2012, 09:00 hrs.

http://www.edudevices.com.ar/
Consultado en 12 de Septiembre del 2012, 22:00 hrs.

http://www.datasheetcatalog.com/
Consultado en 14 de Agosto del 2012, 07:00 hrs.

http://www.wikipedia.org/
Consultado en 12 de Enero del 2013, 13:00 hrs.

Anexos

Anexo I

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
Facultad de Estudios Superiores Aragón
Ingeniería Mecánica
Programa de Asignatura

NOMBRE DE LA ASIGNATURA:	DISEÑO MECATRÓNICO (L)				
PLAN 2007	Tipo de asignatura: Teórico-Práctica				
Clave:	**Créditos:**	10	**Carácter:** Optativa	**Semestre:**	Octavo
Duración del Curso	Semanas:	16	**Área de Conocimiento:**	Mecatrónica	
	Horas:	96			
Horas/Semana	Teoría:	4.0			
	Práctica:	2.0			
MODALIDAD: CURSO-LABORATORIO					
SERIACIÓN INDICATIVA PRECEDENTE:	Ninguna.				
SERIACIÓN INDICATICA SUBSECUENTE:	Ninguna.				

OBJETIVO DEL CURSO:

El alumno aplicará los principios de operación de los sistemas mecatrónicos a través del estudio de los microprocesadores y su aplicación en el diseño de sistemas industriales que integran elementos mecánicos, eléctricos, electrónicos y de programación.

No.	Nombre	Horas Teoría	Horas Práctica
I	INTRODUCCIÓN	4.0	2.0
II	METODOLOGÍA EN EL DESARROLLO DE PRODUCTOS	8.0	2.0
III	MICROPROCESADORES Y MICROCONTROLADORES	28.0	16.0
IV	SENSORES, ACTUADORES E INTERFASES HOMBRE-MÁQUINA	12.0	6.0
V	SISTEMA MECÁNICO	12.0	6.0
	Total de Horas Teóricas:	64.0	
	Total de Horas Prácticas:		32.0
	TOTAL:	96.0	

OBJETIVOS Y CONTENIDO DE LOS TEMAS

TEMA I "INTRODUCCIÓN"

Objetivo: El alumno enunciará la importancia de la mecatrónica y sus aplicaciones en la industria.

Contenido:

I.1 Definición de Mecatrónica y su evolución.

I.2 La mecatrónica en la automatización de las fábricas, oficinas, el hogar y los productos.

TEMA II "METODOLOGÍA EN EL DESARROLLO DE PRODUCTOS"

Objetivo: El alumno analizará los métodos utilizados en el diseño de sistemas mecatrónicos.

Contenido:

II.1 Elementos constitutivos de un sistema mecatrónico.

II.2 Definición de: método de diseño, procedimiento de diseño y modelos.

II.3 Comparación de las características metodológicas del diseño mecánico, electrónico y de programación.

TEMA III "MICROPROCESADORES"

Objetivo: El alumno analizará la arquitectura, funcionamiento y programación de un microprocesador.

Contenido:

III.1 Los microprocesadores en los sistemas mecatrónicos y el diseño de sistemas con microprocesador.

III.2 El microprocesador: componentes, memorias, bus de direcciones, bus de datos, bus de control, mapas de memoria y decodificación de memoria.

III.3 Funcionamiento general del hardware y del software.

III.4 La unidad de procesamiento central CPU

III.4.1 La fase de BUSQUEDA: El apuntador de programa, el apuntador de pila, decodificación y control de la instrucción.

III.4.2 La fase de EJECUCION: La unidad aritmética lógica, el acumulador, el registro de estados, el banco de registros.

III.4.3 Procesamiento de interrupciones: Salvamento de registros, sistemas con interrupciones, múltiples, interrupciones no vectorizadas, interrupciones vectorizadas, procesamiento de interrupciones múltiples, interrupciones no enmascarables y restablecimiento.

III.5 Instrucciones del microprocesador: Códigos de operación, operados y conjunto de instrucciones.

OBJETIVOS Y CONTENIDO DE LOS TEMAS

III.6 Modos de direccionamiento: Direccionamiento inmediato, directo, paginado, indirecto, indexado, relativo, pila, etc.

III.7 Ensamblador: Etiquetas, mnemónicos, comentarios, seudo-instrucciones, editor, ensamblador, ligador.

III.8 Puertos de entrada/salida: Latches, Transeivers, puertos paralelos y seriales, contadores y temporizadores, convertidores digitales -analógicos y analógicos-digitales.

III.9 Compiladores en lenguajes de alto nivel.

TEMA IV "SENSORES, ACTUADORES E INTERFASES HOMBRE-MAQUINA"

Objetivo: El alumno aplicará los tipos de sensores, actuadores e interfases hombre-máquina en un sistema mecatrónico.

Contenido:

IV.1 Sensores: Clasificación por su función, su desempeño y su salida, de estado sólido, ópticos, piezoeléctricos, ultrasónicos.

IV.2 Actuadores: Actuadores lineales, rotacionales, neumáticos, hidráulicos y eléctricos.

IV.3 Interfases hombre-máquina.

TEMA V "SISTEMA MECÁNICO"

Objetivo: El alumno enunciará los beneficios que se tienen al diseñar sistemas que operan con principios mecatrónicos y realizará un proyecto donde se integren los conocimientos de la asignatura.

Contenido:

V.1 Características de los sistemas mecatrónicos comparados con los sistemas tradicionales.

V.2 Aspectos del control en los sistemas mecatrónicos.

V.3 Mecanismos y estructuras en los sistemas mecatrónicos.

V.4 Proyecto integrador.

BIBLIOGRAFÍA

Bibliografía Básica	Temas para los que se recomienda.
Mecatrónica. Sistemas de control electrónico en ingeniería mecánica y eléctrica. México. Ed. Alfaomega. 2001.	TODOS
Bradley, D.A., Dawson, D. *Mechatronics, Electronics in Products and Processes.* Gran Bretaña. Chapman and Hall. 1991.	I Y II
A Theoretical Approach to Mechatronics Desig". Institute for Engineering Design, Dinamarca. Technical University of Denmark. 1990.	TODOS
The Engineering Design Process EE.UU. Edit. John Wiley & Sons 1996.	TODOS

Bibliografía Complementaria	Temas para los que se recomienda.
Bolton, William. *Mecatrónica. Sistemas De Control Electrónico En Ingeniería Mecánica y Eléctrica* México. Alfaomega. 2001.	TODOS
Popovic, Dobrivoje *Mechatronics In Engineering Design And Product Development.* Usa. Book News, Inc. 1998.	TODOS
Seward, Derek. *Mechatronics And The Design Of Intelligent Mechines And Systems.* USA. Book News, Inc. 2001.	TODOS
Burr, J. *A Theoretical Approach To Mechatronics Design Denmark* Institute For Engineering Design, Technical University Of Denmark. 1990.	TODOS
Hunt, V. D. *Mechatronics: Japan's Newest Treat.* Great Britain. Chapman And Hall. 1988.	TODOS

Anexo II

SGS-THOMSON MICROELECTRONICS

L293D
L293DD

PUSH-PULL FOUR CHANNEL DRIVER WITH DIODES

- 600mA OUTPUT CURRENT CAPABILITY PER CHANNEL
- 1.2A PEAK OUTPUT CURRENT (non repetitive) PER CHANNEL
- ENABLE FACILITY
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)
- INTERNAL CLAMP DIODES

SO(12+4+4) Powerdip (12+2+2)

ORDERING NUMBERS:

L293DD L293D

DESCRIPTION

The Device is a monolithic integrated high voltage, high current four channel driver designed to accept standard DTL or TTL logic levels and drive inductive loads (such as relays solenoides, DC and stepping motors) and switching power transistors.

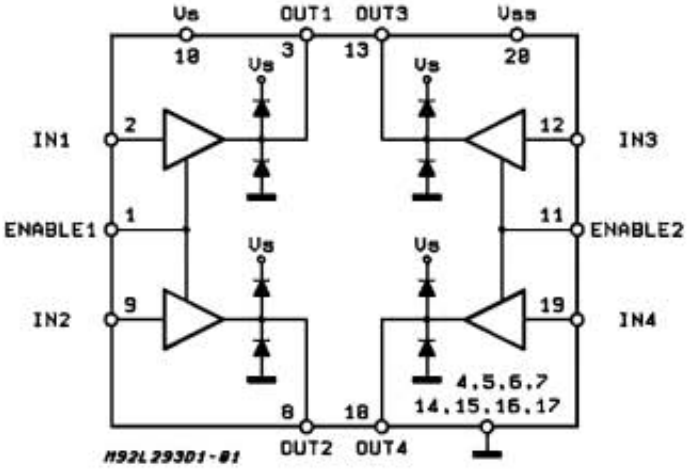
To simplify use as two bridges each pair of channels is equipped with an enable input. A separate supply input is provided for the logic, allowing operation at a lower voltage and internal clamp diodes are included.

This device is suitable for use in switching applications at frequencies up to 5 kHz.

The L293D is assembled in a 16 lead plastic packaage which has 4 center pins connected together and used for heatsinking

The L293DD is assembled in a 20 lead surface mount which has 8 center pins connected together and used for heatsinking.

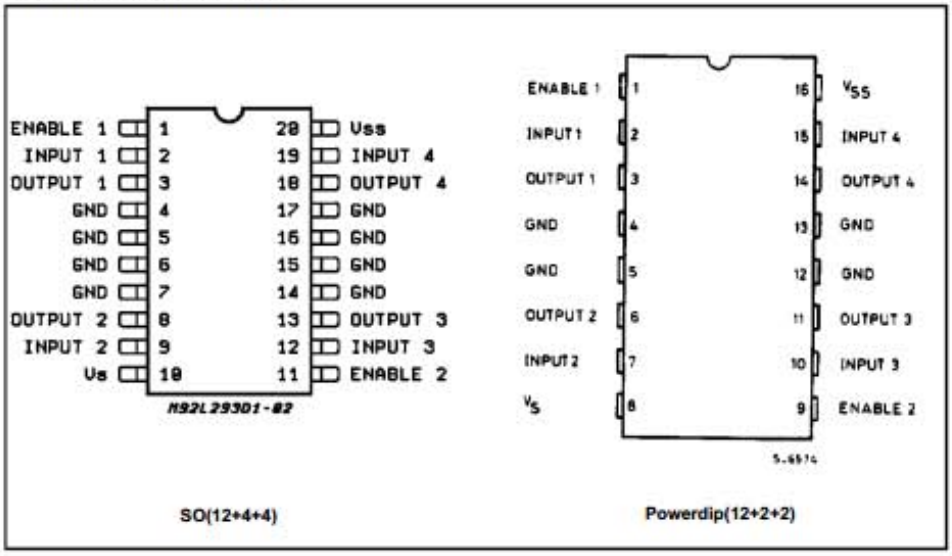
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Supply Voltage	36	V
V_{SS}	Logic Supply Voltage	36	V
V_i	Input Voltage	7	V
V_{en}	Enable Voltage	7	V
I_o	Peak Output Current (100 µs non repetitive)	1.2	A
P_{tot}	Total Power Dissipation at T_{pins} = 90 °C	4	W
T_{stg}, T_j	Storage and Junction Temperature	– 40 to 150	°C

PIN CONNECTIONS (Top view)



SO(12+4+4) Powerdip(12+2+2)

THERMAL DATA

Symbol	Decription		DIP	SO	Unit
$R_{th\ j\text{-}pins}$	Thermal Resistance Junction-pins	max.	–	14	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance junction-ambient	max.	80	50 (*)	°C/W
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	max.	14	–	

(*) With 6sq. cm on board heatsink.

ELECTRICAL CHARACTERISTICS (for each channel, Vs = 24 V, Vss = 5 V, T_{amb} = 25 °C, unless otherwise specified)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 10)		V_{SS}		36	V
V_{SS}	Logic Supply Voltage (pin 20)		4.5		36	V
I_S	Total Quiescent Supply Current (pin 10)	V_i = L ; I_O = 0 ; V_{en} = H		2	6	mA
		V_i = H ; I_O = 0 ; V_{en} = H		16	24	mA
		V_{en} = L			4	mA
I_{SS}	Total Quiescent Logic Supply Current (pin 20)	V_i = L ; I_O = 0 ; V_{en} = H		44	60	mA
		V_i = H ; I_O = 0 ; V_{en} = H		16	22	mA
		V_{en} = L		16	24	mA
V_{IL}	Input Low Voltage (pin 2, 9, 12, 19)		– 0.3		1.5	V
V_{IH}	Input High Voltage (pin 2, 9, 12, 19)	$V_{SS} \leq 7$ V	2.3		V_{SS}	V
		$V_{SS} > 7$ V	2.3		7	V
I_{IL}	Low Voltage Input Current (pin 2, 9, 12, 19)	V_{IL} = 1.5 V			– 10	µA
I_{IH}	High Voltage Input Current (pin 2, 9, 12, 19)	$2.3\ V \leq V_{IH} \leq V_{SS} - 0.6\ V$		30	100	µA
$V_{en\,L}$	Enable Low Voltage (pin 1, 11)		– 0.3		1.5	V
$V_{en\,H}$	Enable High Voltage (pin 1, 11)	$V_{SS} \leq 7$ V	2.3		V_{SS}	V
		$V_{SS} > 7$ V	2.3		7	V
$I_{en\,L}$	Low Voltage Enable Current (pin 1, 11)	$V_{en\,L}$ = 1.5 V		– 30	– 100	µA
$I_{en\,H}$	High Voltage Enable Current (pin 1, 11)	$2.3\ V \leq V_{en\,H} \leq V_{SS} - 0.6\ V$			± 10	µA
$V_{CE(sat)H}$	Source Output Saturation Voltage (pins 3, 8, 13, 18)	I_O = – 0.6 A		1.4	1.8	V
$V_{CE(sat)L}$	Sink Output Saturation Voltage (pins 3, 8, 13, 18)	I_O = + 0.6 A		1.2	1.8	V
V_F	Clamp Diode Forward Voltage	I_O = 600nA		1.3		V
t_r	Rise Time (*)	0.1 to 0.9 V_O		250		ns
t_f	Fall Time (*)	0.9 to 0.1 V_O		250		ns
t_{on}	Turn-on Delay (*)	0.5 V_i to 0.5 V_O		750		ns
t_{off}	Turn-off Delay (*)	0.5 V_i to 0.5 V_O		200		ns

(*) See fig. 1.

TRUTH TABLE (one channel)

Input	Enable (*)	Output
H	H	H
L	H	L
H	L	Z
L	L	Z

Z = High output impedance
(*) Relative to the considered channel

Figure 1: Switching Times

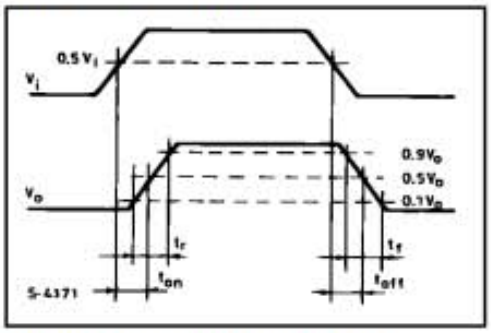
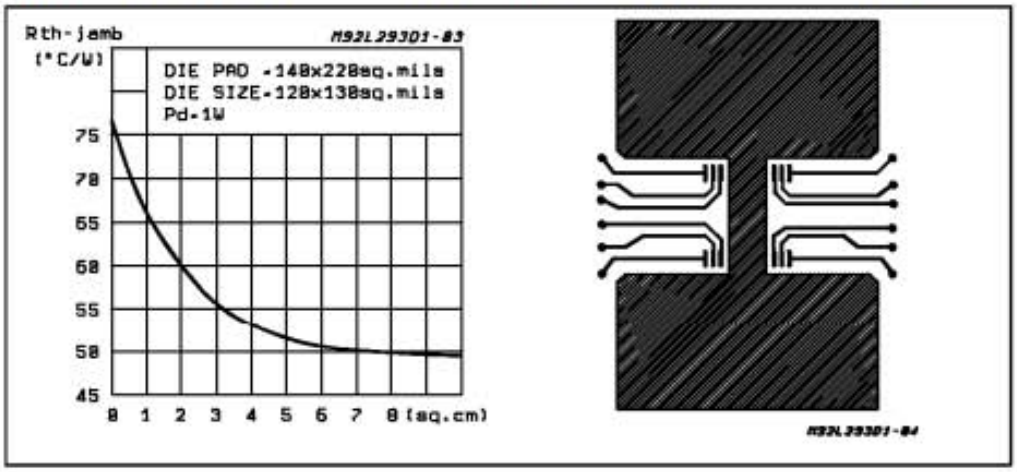
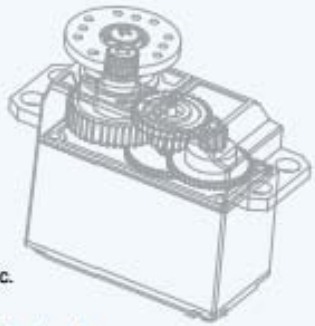


Figure 2: Junction to ambient thermal resistance vs. area on board heatsink (SO12+4+4 package)



Anexo III

VIGOR PRECISION LTD.
力嘉精密有限公司



VS-2A SERVO

FOR REFERENCE

TECHNICAL DATA

Control System:	Pulse width control, 1500μs neutral	
Operating Voltage:	4.8V~6.0V (DC)	
STD Direction:	Counter clockwise/pulse travelling 800 to 2200 μsec.	
Test Voltage:	at 4.8V	at 6.0V
Operating Speed:	0.20sec/60°at no load	0.17sec/60° at no load
Stall Torque:	≥4.5kgf.cm(62.49 oz/in)	≥5.0kgf.cm(69.44 oz/in)
Running Current:	~0.2A	~0.25A
Stall Current:	~1.2A	~1.5A
Output Angle:	≥170°	
Dead Band Width:	5μsec	

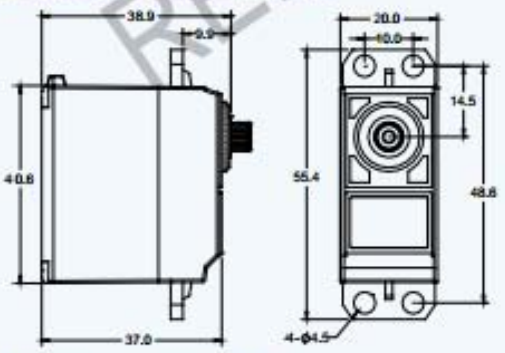
FEATURES

Motor Type:	3poles metal brush motor (Heavy-duty Motor)
Potentiometer Life:	1,000,000 cycles
Bearing:	None
Gear Material:	Resin
Spline Horns:	Spline type for FUTABA style
Connector Wire Length:	250mm(9.84in),26AWG(Connector for FUTABA or JR style)

DIMENSIONS

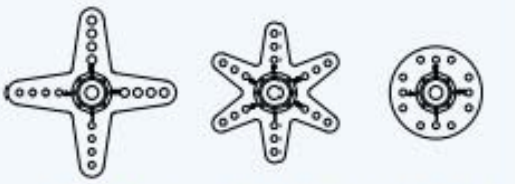
Size :	40.6×20.0×38.9mm(1.6×0.79×1.53in)
Weight:	37g (1.30oz)

CASE AND HORN DRAWING



STD ACCESSORIES:

Horn Screw × 1pc
Cross Horn "A" × 1pc
Star Horn "A" × 1pc
Circular Horn "A" × 1pc



Cross Horn "A" Star Horn "A" Circular Horn "A"

OPTION ACCESSORIES:

Servo Mounter "Set-C": Rubber Mounter Plate × 2pcs and Small Metal Sleeve × 4pcs

REMARKS:

Anexo IV

MC9S08JM60 Series Features

8-Bit HCS08 Central Processor Unit (CPU)

- 48-MHz HCS08 CPU (central processor unit)
- 24-MHz internal bus frequency
- HC08 instruction set with added BGND instruction
- Background debugging system
- Breakpoint capability to allow single breakpoint setting during in-circuit debugging (plus two more breakpoints in on-chip debug module)
- In-circuit emulator (ICE) debug module containing two comparators and nine trigger modes. Eight deep FIFO for storing change-of-flow addresses and event-only data. Debug module supports both tag and force breakpoints.
- Support for up to 32 interrupt/reset sources

Memory Options

- Up to 60 KB of on-chip in-circuit programmable flash memory with block protection and security options
- Up to 4 KB of on-chip RAM
- 256 bytes of USB RAM

Clock Source Options

- Clock source options include crystal, resonator, external clock
- MCG (multi-purpose clock generator) — PLL and FLL; internal reference clock with trim adjustment

System Protection

- Optional computer operating properly (COP) reset with option to run from independent 1-kHz internal clock source or the bus clock
- Low-voltage detection with reset or interrupt
- Illegal opcode detection with reset
- Illegal address detection with reset

Power-Saving Modes

- Wait plus two stops

Peripherals

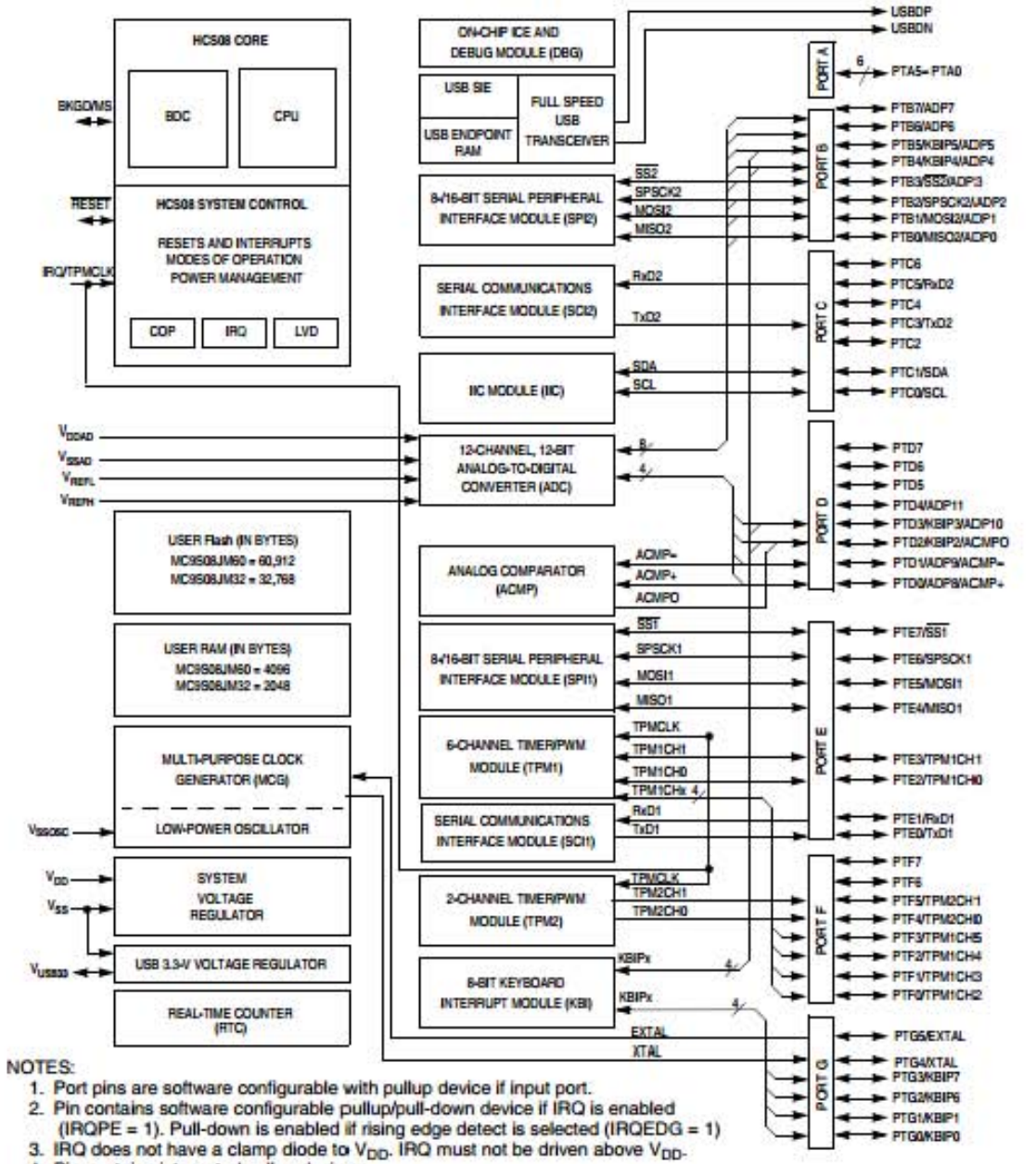
- **USB** — USB 2.0 full-speed (12 Mbps) device controller with dedicated on-chip USB transceiver, 3.3-V regulator and USBDP pull-up resister; supports control, interrupt, isochronous, and bulk transfers; supports endpoint 0 and up to 6 additional endpoints; endpoints 5 and 6 can be combined to provide double buffering capability
- **ADC** — 12-channel, 12-bit analog-to-digital converter with automatic compare function; internal temperature sensor
- **ACMP** — Analog comparator with option to compare to internal reference; operation in stop3 mode
- **SCI** — Two serial communications interface modules with optional 13-bit break LIN extensions
- **SPI** — Two 8- or 16-bit selectable serial peripheral interface modules with a receive data buffer hardware match function
- **IIC** — Inter-integrated circuit bus module to operate at up to 100 kbps with maximum bus loading; multi-master operation; programmable slave address; interrupt-driven byte-by-byte data transfer; 10-bit addressing and broadcast modes support
- **Timers** — One 2-channel and one 6-channel 16-bit timer/pulse-width modulator (TPM) modules: Selectable input capture, output compare, and edge-aligned PWM capability on each channel. Each timer module may be configured for buffered, centered PWM (CPWM) on all channels
- **KBI** — 8-pin keyboard interrupt module
- **RTC** — Real-time counter with binary- or decimal-based prescaler

Input/Output

- Up to 51 general-purpose input/output pins
- Software selectable pullups on ports when used as inputs
- Software selectable slew rate control on ports when used as outputs
- Software selectable drive strength on ports when used as outputs
- Master reset pin and power-on reset (POR)
- Internal pullup on $\overline{\text{RESET}}$, IRQ, and BKGD/MS pins to reduce customer system cost

Package Options

- 64-pin quad flat package (QFP)
- 64-pin low-profile quad flat package (LQFP)
- 48-pin quad flat no-lead (QFN)
- 44-pin low-profile quad flat package (LQFP)



NOTES:
1. Port pins are software configurable with pullup device if input port.
2. Pin contains software configurable pullup/pull-down device if IRQ is enabled (IRQPE = 1). Pull-down is enabled if rising edge detect is selected (IRQEDG = 1)
3. IRQ does not have a clamp diode to V_{DD}. IRQ must not be driven above V_{DD}.
4. Pin contains integrated pullup device.
5. When pin functions as KBI (KBIPEn = 1) and associated pin is configured to enable the pullup device, KBEDGn can be used to reconfigure the pullup as a pull-down device.

Figure 1-1. MC9S08JM60 Series Block Diagram

2.2 Device Pin Assignment

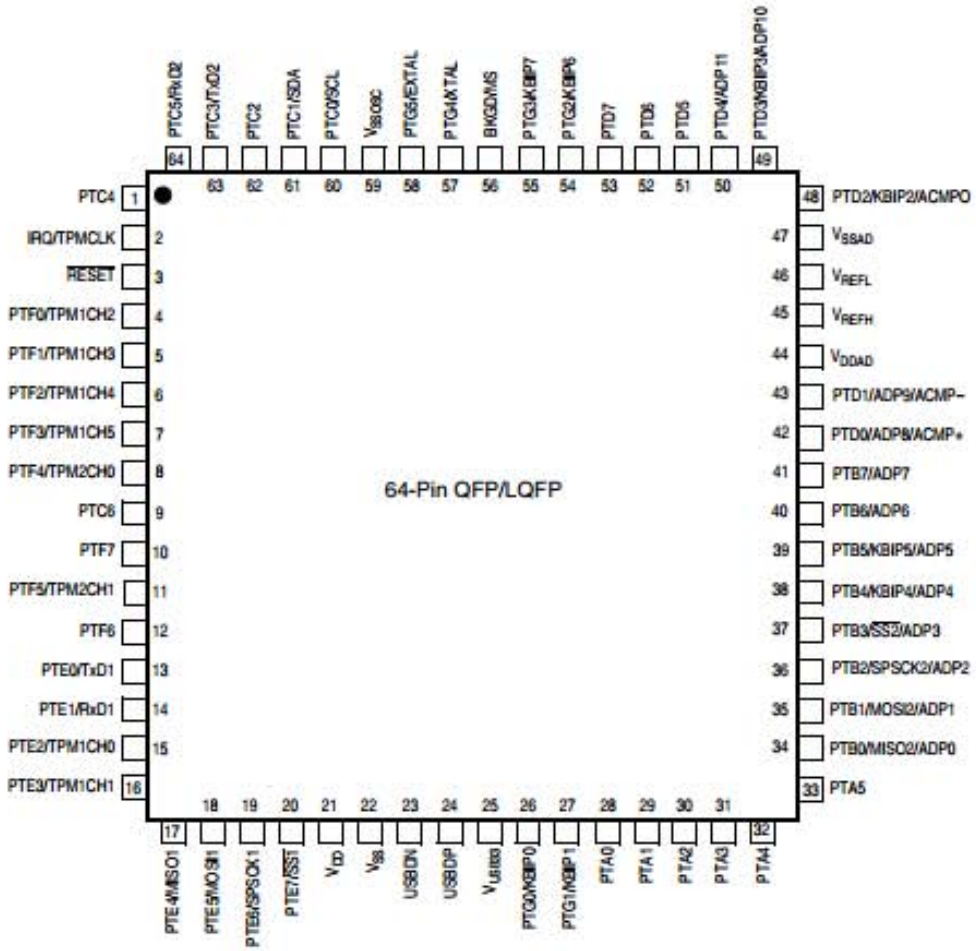


Figure 2-1. MC9S08JM60 in 64-Pin QFP/LQFP Package

Anexo V

1 INTRODUCTION

1.1 Overview

The DEMOJM is a low cost development system supporting Freescale MC9S08JM60 and MCF51JM128 64LQFP microcontrollers. It consists of a DEMOJM Base Board, a DC9S08JM60 Daughter Card and a DC51JM128 Daughter Card. P&E's Embedded Multilink circuitry on the DEMOJM board allows the processor connected to the DEMOJM to be debugged and programmed via USB from a PC. In addition, the demo board can be powered using the USB bus.

1.2 Package Contents

The DEMOJM package includes the following items:

- DEMOJM Base Board with a DC9S08JM60 Daughter Card installed
- DC51JM128 Daughter Card
- Getting Started DVD - Getting started with the series of microcontrollers
- USB A-to-B Cable
- Mini-AB USB Kit
- USB Thumb Drive
- Quick Start Guide
- Freescale Warranty Card

1.3 Supported Devices

The DEMOJM supports the following devices:

- MC9S08JM60CLH

- MCF51JM128VLH

1.4 Recommended Materials On The Getting Started DVD-ROM

- Freescale MC9S08JM60 reference manual and datasheet
- Freescale MCF51JM128 reference manual and datasheet
- DEMOJM Base Board and Daughter Card schematic
- P&E Embedded Multilink Toolkit applications
- P&E Embedded Multilink driver installation guide and resources

1.5 Handling Precautions

Take care to handle the package contents, including the DEMOJM Base Board, DC9S08JM60 Daughter Card, and DC51JM128 Daughter Card, in a manner such as to prevent electrostatic discharge.

2 HARDWARE FEATURES

The DEMOJM is a demonstration and development system for Freescale's MC9S08JM60 and MCF51JM128 microcontrollers. Application development is quick and easy using P&E's Embedded Multilink circuitry and the included software tools and examples. An optional BDM port is provided to allow the use of an external BDM interface such as P&E's Cyclone PRO automated programmer or USB Multilink. The USB Multilink is functionally comparable to the DEMOJM's Embedded Multilink circuitry.

Note: The DEMO board's onboard Embedded Multilink circuitry is intended to function with the onbaord processor and any daughter cards that may be included. It cannot be used to communicate with other devices.

2.1 DEMOJM Base Board Features

- On-board Logic Analyzer
- On-board Virtual Serial Port
- Four (4) asymmetrically positioned 8x2 male connectors for interchangeable daughter cards
- P&E's Embedded Multilink circuitry populated on the underside

- SCI signals connected to P&E's Embedded Multilink through jumpers
- ON/OFF Power Switch w/ LED indicator
- A 6VDC - 8VDC power supply input barrel connector

Note: The DEMOJM board power connector is incorrectly labelled as 6-12VDC. Maximum voltage is 8VDC.

- Power Input Selection Jumpers for selecting the input voltage source:
  - Power Input from Embedded Multilink to LDO regulator
  - Power Input from DC Power Jack to LDO regulator
  - Power Input from Mini-AB connector
  - Power Input from MCU_PORT connector
- RESET Push Button and LED indicator w/ Enable
- User Features:
  - USB device mode and host mode support with Mini-AB USB connector
  - CAN Module w/Enable
  - 3-axis Accelerometer w/Enable
  - 8 User LED's w/ Enable
  - 4 User Push Buttons w/ Enable
  - 1 Piezo Buzzer w/ Enable
  - IIC Pullups w/ Enable
  - 10K Ohm POT w/ Enable
- Specifications:
  - Board Size 3.5 x 4.0
  - Daughter Card Size 1.4 x 1.5
  - Power Input:
    - USB Cable: 5VDC, 500mA max
    - DC Power Jack: 2.5/5.5mm barrel connector, 6VDC to 8VDC Center Positive

Note: The DEMOJM board power connector is incorrectly labelled as 6-12VDC

Maximum voltage is 8VDC.

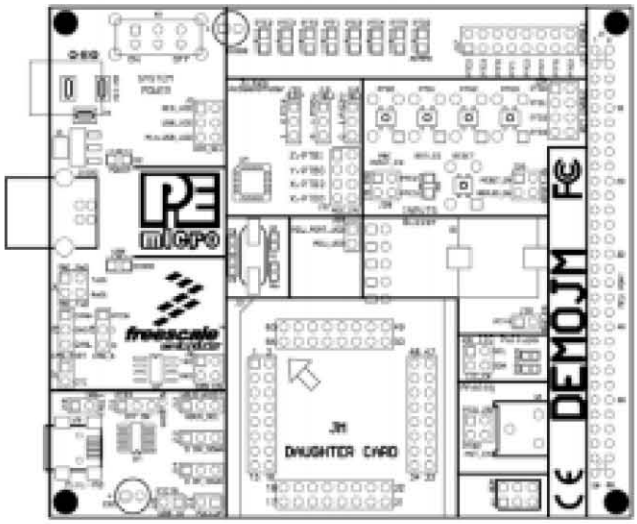


Figure 2-1: DEMOJM Top Component Placement

2.2 On-Board Logic Analyzer

The DEMOJM board has a built-in 2-channel logic analyzer which may be used to display captured data in real-time on a host PC. The logic analyzer channels (IN0/IN1) are connected to the PTE2 and PTE3 signals on the DEMOJM board by default via the J28 jumpers. The channels may be connected to any of the processor pins via wire jumpers (not included).

The Logic Analyzer Utility, included in the P&E Embedded Multilink Toolkit on the accompanying DVD-ROM, displays the logic analyzer signals on a PC.

2.3 On-Board Virtual USB Port

The DEMOJM board has a built-in virtual serial port which may be connected to the JM processor's SCI RXD/TXD. This allows certain PC applications to be able to connect in a serial fashion to the microcontroller without the actual use of serial port hardware.

The Terminal Window Utility, included in the P&E Embedded Multilink Toolkit on the accompanying DVD-ROM, is a generic serial port utility which works with the DEMOJM virtual serial port or actual serial port hardware.

2.4 DEMOJM Daughter Card Features

- Four (4) bottom-mounted asymmetrically positioned 8x2 female connectors to mate with the DEMOJM Base Board
- A top-mounted MC9S08JM60CLH or MCF51JM128VLH chip

2.5 DEMOJM Jumper/Connector Quick Reference

Default Jumper Settings

The following is a list of default jumper settings for DEMOJM board. The settings listed indicate the "on" (or installed) position.

Default Jumper Settings

JUMPERS	SETTINGS
J3	3&4
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	ALL ON
J18	2&3

Default Jumper Settings

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
J33	1&2

MCU Port Connector Pinout

The following is the pinout for the MCU Port connector on the DEMOJM board.

Figure 2-2: MCU Port Connector Pinout

3 GETTING STARTED WITH THE DEMOJM

The DEMOJM is a low-cost board targeting quick microcontroller evaluation. The board includes two plug-in daughter cards to demonstrate the ease of migration between the Flexis JM60, 8-bit S08 and Flexis JM128, 32-bit ColdFire V1 microcontrollers. The board also includes a power terminal to measure the ultra-low power consumption of the JM devices.

Please refer to the DEMOJM Quick Start Guide and Labs for instructions on how to install software, connect the DEMOJM to your PC, and run quick demonstrations.

4 SYSTEM SETUP

4.1 Overview

P&E's Embedded Multilink driver is required to operate the DEMOJM using a PC. The Embedded Multilink driver should be installed with the CodeWarrior Development Studio software or from the DEMOJM Resources in the Getting Started DVD-ROM before the PC is connected to the DEMOJM.

4.2 Operating System Requirements

The following are the resources required to run the CodeWarrior Development Studio and the DEMOJM:

- A PC-compatible system running Windows 2000, Windows XP, or Windows Vista
- 128MB of available system RAM, and 1GB of available hard disk space
- A DVD-ROM drive for software installation
- A USB port

4.3 Software Setup

4.3.1 Installing CodeWarrior Development Studio

To install the CodeWarrior Development Studio, follow the instructions on the DVD-ROM.

4.3.2 Installing P&E Resources

Use the DEMOJM Resources in the DVD-ROM to access and install P&E resources for the DEMOJM. These materials are not required for operation. The DEMOJM Resources in the Getting Started DVD-ROM contains the following support materials:

- DEMOJM Embedded Multilink hardware interface driver

- DEMOJM User Manual (this document)
- DEMOJM Base Board and Daughter Cards Schematics
- DEMOJM Component Breakdown List
- P&E Embedded Multilink Toolkit PC Applications
- P&E Evaluation Software
- Links to Freescale documentation, P&E Discussion Forums, and DEMOJM FAQs.

4.4 Quick Startup

Only a few steps are required to get the DEMOJM up and running. Please reference the Quick Start Guide.

4.5 Hardware Setup

4.5.1 First-Time Connection

The DEMOJM may be connected to a PC through a USB port. Connection steps are listed below in typical order:

1. Install the required software, as described in the previous section.
2. Make sure the jumper USB_VDD for VDD_SEL is installed.
3. Plug the USB cable A-M connector into a free USB port of the PC.
4. Plug the USB cable B-M connector into the USB connector on the DEMOJM Base Board.
5. The operating system will recognize P&E's Embedded Multilink circuitry and P&E's USB to Serial circuitry. Depending on the operating system, you may see the "Found New Hardware Wizard" dialog to assist you with software installation for "PEMicro USB Multilink (i0)." On Windows XP (SP2), the following dialog will appear: