UNIVERSIDAD NACIONAL AUTÓNOMA

DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES ARAGÓN

Diseño de sistemas embebidos con microcontroladores hc08

T E S I S

Para obtener el titulo de:
Ingeniero mecánico electricista
Área: eléctrica electrónica
P R E S E N T A:
Jose Luis Escutia Sánchez

Asesor:
Arturo Ocampo Álvarez

México **MMX**

A mis padres Ignacio y Martha
con amor y cariño

Puesto que yo soy imperfecto y necesito la tolerancia y la bondad de los demás, también he de tolerar los defectos del mundo hasta que pueda encontrar el secreto que me permita ponerles remedio.

Mahatma Gandhi

Agradecimientos

Dios.
Gracias Padre

Martha Sánchez Vázquez ✞
Mi madre, quien me dio la vida, ella me enseñó el lado bueno y divertido de todas las cosas, a ponerme en lo zapatos de mi prójimo y que servir es un privilegio, en paz descanse.

Ignacio Escutia Rubio
Mi padre, mi mentor, quien me enseñó que solo existen dos formas de hacer las cosas, sin contar su paciencia extraordinaria que sin duda me heredó, gracias por ser lo que eres para con migo.

Beatriz Guadalupe Espinal Mendoza.
Mi querida y amada esposa, gracias por tu apoyo y por realizar el trabajo más difícil del mundo…….. Soportarme

Eduardo Gaitan Mendoza y Ma. Del Rosario Vallejo Cadena.
Mis amigos, quienes me brindaron ayuda invaluable e incondicional sin solicitar nada a cambio, espero en Dios algún día poder retribuirles todo lo que por mi hicieron…… Gracias.

Ing. Arturo Ocampo Álvarez
Gracias Profe por todas sus enseñanzas y sobre todo su paciencia, tal vez yo no sea su alumno favorito pero usted si que es mi profesor preferido.

Mis hermanos Ignacio y Martha
Gracias por hacerme reír en momentos difíciles

COMECYT CONACYT
Gracias por su apoyo "becas a tesis de licenciatura"

A mi suegra Patricia Mendoza Espinosa y a todas las personas que de una u otra forma me ayudaron….. Gracias.

Jose Luis Escutia Sánchez

INDICE

PLANTEAMIENTO DEL PROBLEMA

Desde la revolución Industrial la tecnología ha evolucionado de forma exponencial, en nuestros días la tecnología más reciente se vuelve obsoleta en cuestión de meses, sin embargo, lo que en algunos casos es obsoleto en otros continúa siendo mejor que lo nuevo, esto depende en gran medida del ingenio, habilidad y formación de los diseñadores, así, la educación y actualización toman gran importancia, ya que en cuestión de unos cuantos semestres las materias dedicadas al estudio de alguna tecnología en particular deben ser actualizadas, aunque para un ingeniero recién egresado esta actualización depende enteramente de él.

PERO ¿QUE ES UN SISTEMA EMBEBIDO? Es un sistema de computación basado en microprocesadores o microcontroladores (dispositivo que controla o monitorea) con un propósito específico, entiéndase como un sistema incrustado que resuelve una o varias tareas específicas dentro de un sistema más grande. Algunos ejemplos de sistemas embebidos son los sistemas de información integrados en automóviles, trenes o aviones, controladores de procesos en sistemas de producción industrial, sistemas de monitoreo o de adquisición de datos hasta los sistemas de control en una lavadora, horno de microondas o refrigerador.
El diseño de un producto que incorpora uno o más sistemas embebidos generalmente está orientado a minimizar los costos y maximizar la confiabilidad. Los sistemas embebidos a menudo operan en un ambiente dedicado con condiciones operacionales y escenarios muy específicos.

Existen muy pocos ingenieros especializados en el diseño de sistemas embebidos, otra parte del problema es que la industria nacional no apoya el consumo de diseños Mexicanos, seguramente por que se cuestiona mucho la calidad de éstos. Así que ¿Qué tan difícil es que un alumno se convierta por si solo en un buen diseñador de sistemas embebidos? En general sólo depende de él, aunque debe invertir una gran cantidad de tiempo incursionando de forma autónoma en el estudio y asimilación de las tecnologías y software de diseño que necesita para estar a la par con los mejores diseñadores, una forma de acelerar el proceso es que el alumno recurra a costosos cursos pero, como sabemos, a la mayoría de los alumnos mexicanos les resulta difícil costear estos cursos por lo que terminan desertando de su deseo de ser diseñadores versátiles.

Un diseñador de sistemas embebido debe ser capaz de resolver problemas de la manera mas práctica, eficiente y económicamente posible pero para esto el diseñador debe ser experto en por lo menos la programación de un familia de microcontroladores, además de tener amplios conocimientos de electricidad, electrónica, electrónica de potencia, control digital, control analógico, diseño lógico, comunicaciones, protocolos de comunicación, etc. Saber elaborar esquemas eléctricos y electrónicos, diseñar circuitos impresos, tener conocimiento de variables físicas como lo son presión temperatura luminosidad humedad y además conocer la tecnología a su alcance que permita sensar estas variables tomando en cuenta que debe ser capaz de interconectar entre si todas estas ramas y muchas más de manera eficiente.

JUSTIFICACIÓN DEL ESTUDIO

La mayoría de los "conocimientos" mencionados anteriormente deben ser aprendidos por el alumno durante el transcurso de su formación universitaria, debido a las deficiencias de los sistemas educativos, los estudiantes no reciben la orientación correcta que les permita integrar sus conocimientos en un sistema funcional y eficiente.
Un sistema funcional y eficiente es lo que si sabe hacer un diseñador de sistemas embebidos y por tal motivo me di a la tarea de generar este trabajo que muestra al estudiante los conocimientos mínimos necesarios que le permitan visualizar el proceso y lo necesario para el diseño de sistemas embebidos.

OBJETIVO GENERAL.

Establecer una serie de conceptos básicos y necesarios para que el alumno pueda apoyarse en la introducción al ramo del diseño embebido, además de tutoriales sobre la paquetería (software) necesaria para sus diseños y así, con un poco práctica el alumno pueda desarrollar su propia metodología a la par de sus habilidades. Cabe mencionar que el alumno puede usar los softwares de diseño de circuitos y de PCB´s de su conveniencia, los MCU´s que mas le agraden o se adapten a sus necesidades de diseño ya que este manual presenta un solo software para cada etapa del diseño y puede darse el caso en el que el alumno conozca o prefiera algún otro programa.
Este trabajo de investigación y compilación tiene el fin de mostrarle al alumno los softwares mínimos necesarios para el diseño electrónico además de abrirle un panorama para reconocer necesidades, problemas y posibles soluciones para así convertirse en un diseñador de sistemas embebidos. **Al final la metodología que se aplique al momento de diseñar depende particularmente del tipo de problema al que se enfrenta y es algo muy particular de cada diseñador.**

OBJETIVO PARTICULAR

El aprendizaje de la programación del microcontrolador, a pesar de que los microcontroladores HC08 son de los mas fáciles de programar gracias a su ambiente de desarrollo Codewarrior existe muy poca documentación en español al respecto de estos, recientemente apareció un libro en español llamado: "MICROCONTROLADORES MOTOROLA - FREESCALE", libro que se enfoca en la programación de los microcontroladores HC08 con algunas aplicaciones que se muestran de una manera didáctica, además de no explicar el funcionamiento del programa Codewarrior (que facilita enormemente la programación) ya que solo se explica la manera de escribir el programa principal para el ambiente de desarrollo WINIDE (programa antecesor al codewarrio).

INTRODUCCION

El siguiente trabajo de tesis se divide en cinco capítulos, en el primero se introduce al lenguaje ensamblador y se describe a conciencia cada una de las instrucciones, además se explica cómo interpretar el set (conjunto) de instrucciones del MCU, sus características y

registros implícitos en la unidad aritmética lógica (ALU) y los modos de direccionamiento con los que es compatible el MCU.

En el segundo capitulo se explica la estructura de funcionamiento del software de programación que se usará para programar estos MCUs, el programa Codewarrior incluye un simulador, un sistema de emulación sobre el circuito además de una interfaz para la programación con un hardware de licencia libre llamado MON08, todas estas características sumado al grabador universal MON08 dan como resultado a un sistema de desarrollo integral y de fácil adquisición similar al sistema ARDUINO (Sistema de desarrollo de hardware y software libre usado en la programación de los microcontroladores ATMEGA8 y ATMEGA168) solo que a diferencia del sistema arduino el sistema basado en la interfaz MON08 es compatible con los aproximadamente 90 modelos de microcontroladores HC08, también cabe mencionar que cada uno de éstos MCUs cuenta con diferentes presentaciones de encapsulado por lo que el diseñador obtiene una amplia gama de opciones con un gasto mínimo en el sistema de desarrollo.

En el tercer capitulo se introduce al alumno en el diseño de diagramas electrónicos utilizando el programa Multisim 9, para migrar después el diseño al programa Ultiboard 9 en el cual se puede generar el PCB a partir del diseño hecho en Multisim 9.

El cuarto capitulo muestra ejemplos de sistemas pequeños y simples, además de una tentativa de pasos a seguir para el diseño de un sistema embebido complejo.

El quinto capitulo muestra la fabricación de un programador llamado OSBDM para la familia HCS08 ya que las herramientas comerciales de estos MCUs son muy costosas para el alumno promedio mexicano, éstos son de reciente aparición, más rápidos y más baratos, además el código de los MCUs HC08 es compatible totalmente con los HCS08.

Al final de este trabajo de tesis se incluye un DVD con todos los archivos necesarios para la elaboración del programador OSBDM además del programa Codewarrior 6.3, así como el software Multisim 9 y Ultiboard 9 y diferentes proyectos de circuitos, PCB´s y programas Codewarrior ya terminados. También contiene diferentes manuales de componentes mencionados en este trabajo.

CAPÍTULO

1

INTRODUCCION A LOS MICROCONTROLADORES "HC08"

OBJETIVOS

Después de estudiar este capítulo el lector podrá ser capaz de:

- ***Identificar las ventajas y desventajas de los microcontroladores HC08.***
- ***Mencionar las características y arquitectura de los microcontroladores HC08.***
- ***Describir y manipular los principales registros de la CPU HC08***
- ***Indicar los diferentes tipos de direccionamientos compatibles con la CPU HC08.***
- ***Interpretar el set de instrucciones HC08.***
- ***Definir el funcionamiento de cada una de las instrucciones del HC08.***
- ***Manipular cada uno de los modos de direccionamiento para cada una de las instrucciones***

Los microcontroladores Freescale son de los más flexibles en el mercado, y la pregunta obligada ¿por que basar nuestro estudio en los microcontroladores HC08 y no en los HCS08? Siendo éstos más rápidos, más baratos y de más resiente aparición por lo cual están actualmente en auge. La respuesta es sencilla ya que al aprender a programar MCUs HC08 se aprende simultáneamente a programar MCUs HCS08 (mismo set de instrucciones), otro motivo es la mayor presentación en encapsulados tipo PDIP en los HC08 que facilitan enormemente la experimentación y por ultimo un programador HCS08 es costoso y fabricarlo no es rápido ni sencillo.

1.0 El MICROCONTROLADOR ADECUADO----------

Una ves que se han delimitado las características y determinado las necesidades de la aplicación (esto se verá en capítulos posteriores) se procede a la elección de un microcontrolador apropiado para el diseño, para esto el primer paso es elegir una arquitectura, dentro de esta arquitectura se busca una familia de dispositivos que serán

consultadas y comparadas entre las soluciones que ofrecen los fabricantes y dentro de la familia el modelo adecuado que se adapte mejor a las características propias del diseño, cabe también mencionar que el mejor microcontrolador no siempre es el mismo, esto depende enteramente de las necesidades del diseño o aplicación.

1.0.1 EL FABRICANTE

En el mercado existe una gran cantidad de fabricantes, la elección de un microcontrolador de Freescale frente a otros más conocidos como los de Intel, Microchip, Atmel, Nacional Semiconductors, etc. se debe a características como su bajo precio, reducido consumo de energía, tamaño, facilidad de uso, fácil programación y la gran cantidad de recursos con los que cuenta cada microcontrolador al momento de diseñar alguna aplicación. Por ello los microcontroladores Freescale se encuentran hoy en día en una gran variedad de aplicaciones industriales, de comunicaciones y control. Al respecto se podría decir como ejemplo que en el caso de la industria automotriz, que actualmente es una de las que mayor precisión requiere en el desarrollo de procesos de control, instrumentación, entre otras, se encuentra que casi el 90% de sus componentes son gobernados por microcontroladores Freescale, debido a sus características de estabilidad, inmunidad al ruido y otros factores importantes que hacen su elección decisiva ante otras marcas, además de contar con elementos suplementarios como dispositivos de radiofrecuencia, sensores de presión, acelerómetros, sensores de proximidad etc.

1.0.2 ELECCION DE LA ARQUITECTURA

Después de elegir al fabricante nos damos a la tarea de elegir una arquitectura dependiendo de que tanto y que tipo de proceso se deba realizar, de acuerdo a las necesidades de la aplicación, para esto Freescale tiene microcontroladores de 8 bits, 16 bits, 32 bits y controladores digitales de señales (DSC´s Digital Signal Controller) de 16 bits los cuales son una fusión entre las bondades de los microcontroladores y los procesadores digitales de señales (DSP Digital Signal Processor). Si se desea controlar un proceso sencillo tal como controlar un display de cristal liquido (LCD) o tal vez diseñar una chapa electrónica o simplemente un sensor de cualquier tipo de baja resolución, un microcontrolador de 8 bits bastaría para realizar la tarea. Pero si se desea realizar un sensor con un poco mas de resolución por ejemplo uno que involucre en su funcionamiento un convertidor analógico digital de 10, 12, o 16 bits implica un microcontrolador de 16 bits para facilitar el diseño. Ahora bien si es necesario realizar complejos procesos matemáticos como la elaboración de un filtro digital de audio o aplicaciones por el estilo se recomienda usar un DSC ya que este cuenta con un amplio set (conjunto) de instrucciones matemáticas además de tener en su interior dos acumuladores es decir algo así como dos unidades funcionales ALU trabajando al mismo tiempo. Por último si se desean controlar una gran cantidad de procesos muy complejos y a gran velocidad como es el caso de los dispositivos portátiles, celulares, Palm´s, videojuegos, sistemas de comunicación de banda ancha, o inalámbricos como el wi-fi son necesarios microcontroladores o microprocesadores de 32 bits.
Para un fin más bien educativo es necesario empezar por lo más sencillo eligiendo una arquitectura de 8 bits, siendo ésta mucho más versátil y sencilla de usar con respecto a las

demás y no existe mejor opción como experiencia inicial en los microcontroladores de Freescale como lo es elegir los de 8 bits y su ambiente de desarrollo Codewarrior además de que su limitación en cuanto a 8 bits es solo aparente ya que con un poco de imaginación se pueden lograr cosas sorprendentes, Freescale cuenta con tres arquitecturas de 8 bits comerciales en producción, estas son HC08, HCS08 y RS08. Descartamos la arquitectura RS08 de ultra bajo costo por ser de muy reciente aparición por lo cual solo cuenta con seis microcontroladores en la actualidad además de no estar a la venta en México, también descartamos la arquitectura HCS08 debido a que sus sistemas de desarrollo son costosos, por lo cual elegimos la arquitectura HC08 ya que su sistema de desarrollo es de muy fácil fabricación además de que con un microcontrolador de esta arquitectura se puede fabricar un programador para los HCS08, el fin es desarrollarse en HC08 para poder migrar hacia HCS08 de manera económica es decir sin la necesidad de comprar un costoso sistema de desarrollo sin dejar aun lado que todo lo aprendido en HC08 será utilizado en HCS08 ya que utilizan el mismo conjunto de instrucciones.

1.0.3 ELECCION DE LA FAMILIA, MCU Y ENCAPSULADO

La arquitectura HC08 cuenta con una gran cantidad de familias entre las que inicialmente se recomiendan las JK, JL, GP y MR debido a que pueden conseguirse de manera gratuita a través de Freescale o simplemente están a la venta en México además de contar con presentación en encapsulados del tipo PDIP para implementaciones en laboratorio en una PROTO BOARD.

Los Microcontroladores MC68HC908MR8, MC68HC908JL3E, MC68HC908JL8, MC68HC908JL16, MC68HC908JK1, MC68HC908JK3, MC68HC908JK8 se pueden conseguir en México o gratuitamente a través de Freescale en encapsulados tipo PDIP, para mayor información puede visitar la tienda virtual de AG Electrónica en su sección de microcontroladores Motorota (esto se debe a que antes Freescale era parte de Motorola), también se puede visitar Freescale.com en su sección de muestras (samples), se requiere dar de alta el correo electrónico como alumno UNAM (??????@escolar.unam.mx), las muestras están sujetas a disponibilidad y el flete es gratuito, es muy importante que en los datos personales y dirección de envío se coloque a la FES Aragón como empresa a la que se pertenece, por ejemplo si se inscribe solo como UNAM su paquete será absorbido por el departamento aduanal de la UNAM y será difícil y tardado recuperarlo.

EL microcontrolador MC68HC908GP32 perteneciente a la Familia GP cuenta con 32 KiloBytes de memoria y lo encontramos en una presentación PDIP de 40 pines como MC68HC908GP32CP, a la venta en México en AG Electrónica a un precio accesible en relación con el número de terminales y tamaño de memoria.

Se eligió éste último por su encapsulado PDIP-40 con gran cantidad de puertos de propósito general (a diferencia de los microcontroladores anteriormente mencionados que tan solo cuentan con 20 o 28 pines en su presentación PDIP) ya que será útil a la hora de conectar varios dispositivos al mismo tiempo como teclados, LCD´s, display de 7 segmentos etc.

Otra característica que motivó la elección de este microcontrolador fue la cantidad de memoria que posee.

Resumiendo para este trabajo se ha seleccionado el microcontrolador MC68HC908GP32CP de encapsulado PDIP-40 perteneciente a la familia GP la cual cuenta con un núcleo HC08 fabricado por Freescale.

1.1 DESCRIPCION GENERAL DEL MCU MC68HC908GP32----------

1.1.1 CARACTERISTICAS

El microcontrolador MC68HC908GP32 es un miembro de bajo costo y muy alto rendimiento de 8 bits HC08, estos microcontroladores tiene una arquitectura interna del tipo Von Neumann, la arquitectura del procesador es del tipo CISC, entre sus principales características tenemos:

- Alto rendimiento, arquitectura HC08 optimizada para compiladores C
- Totalmente compatible con el código objeto de sus arquitecturas predecesoras M6805, M146805, y M68HC05.
- Velocidad máxima interna del bus de 8 MHz.
- Reset por operación a bajo voltaje a 5V o 3V.
- Reset por COP (computer operating properly).
- Reset por detección de código ilegal.
- Reset por detección de dirección ilegal.
- Bajo consumo de poder con sus modos STOP y WAIT.
- Pin de Master reset y Master reset automático al encender.
- 32 Kilobytes de memoria Flash para programa.
- 512 bytes de memoria RAM.
- Serial peripheral interface module (SPI).
- Serial Communications Interface module (SCI).
- Dos temporizadores (Timer) de 16 bits con opciones de: captura de entrada, comparación de salida y PWM.
- Convertidor Analógico – Digital de 8 bits con 8 canales de entrada.
- Resistencias internas de pull up en -RST e -IRQ, reduce el costo del sistema.
- Resistencias de pull up ajustables en modo entrada para los puertos A, C, y D en conjunto o individualmente. En modo salida la resistencia de pull up se deshabilita automáticamente.
- Capacidad de alta corriente de 10 mA en todos los puertos en modo fuente y sumidero.
- Capacidad de muy alta corriente de 15 mA en los puertos PTC0 - PTC4 en modo fuente y sumidero.
- Interrupción por Timer con circuito preescalador de reloj, con opción de despertar periódico después de un STOP.
- Opción a desactivar o habilitar el oscilador en el modo STOP.
- Teclado de interrupciones de 8 bits

- En el encapsulado PDIP–40 el puerto C es de solo 5 bits PTC0-PTC4 y el puerto D es de solo 6 bits PTD0-PTD5.
- 16 modos de direccionamiento.
- Registro de índice X y puntero de pila (stack pointer) de 16 bits.
- Transferencia de datos de memoria a memoria.
- Rápida instrucción de multiplicación de 8 * 8 bits.
- Rápida instrucción de de división de 16 / 8 bits.
- Instrucción (BCD) binario a código decimal.
- Optimizado para aplicaciones de control.
- Soporta eficientemente el lenguaje C.

1.1.2 DIAGRAMA A BLOQUES DEL MCU

En la figura 1.1 se muestra la estructura del MC68HC908GP32, la cruz denota que el pin puede configurarse con una resistencia interna de pull up a través de software en su funcionamiento como entrada, la doble cruz indica que el pin es configurable a salida en alta corriente y por ultimo el asterisco señala que el pin tiene por defecto una resistencia interna de pull up.

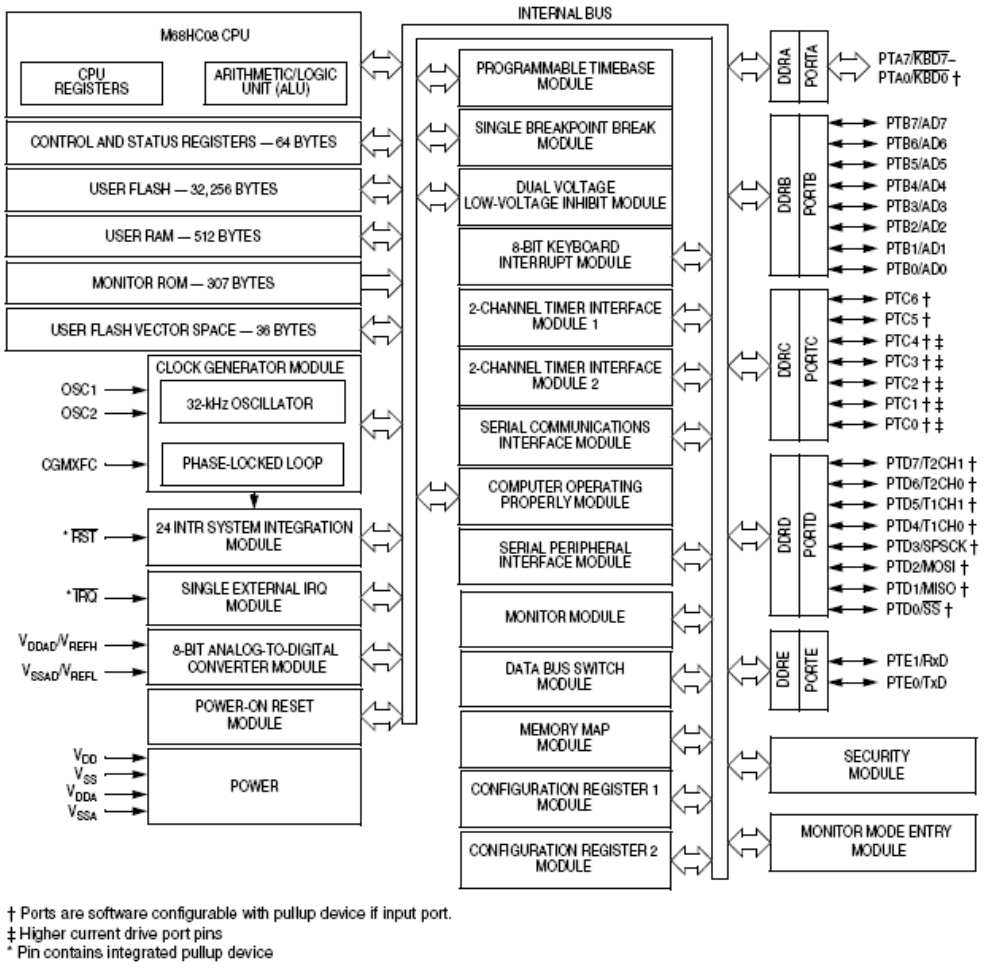


Figura 1.1: Diagrama a bloques del MCU

1.1.3 ASIGNACIÓN DE PINES PARA EL ENCAPSULADO PDIP-40

El núcleo del MCU cuenta con 4 puertos mas: PTC5, PTC6, PTD6/T2CH0, PTD7/T2CH1. Los dos primeros están conectados internamente a tierra y los dos últimos están desconectados. En la figura 1.2 puede verse la configuración de los pines del MCU.

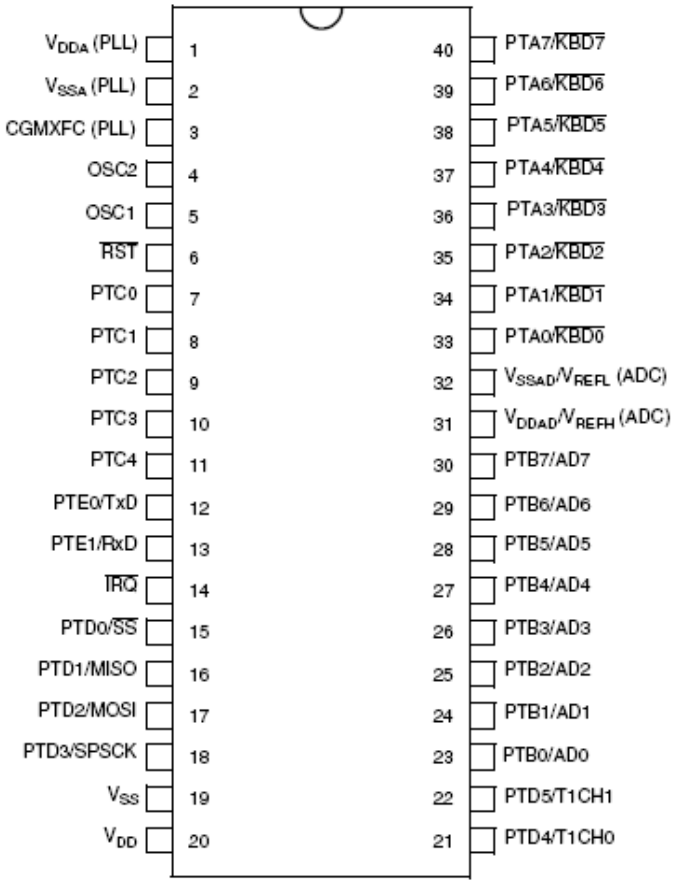


Figura 1.2: Asignación de pines 1

1.1.4 POLARIZACIÓN DEL MCU (VDD Y VSS)

EL voltaje V_{DD} corresponde al de polarización positiva el cual puede ser de 5v o 3v, V_{SS} corresponde a GND o 0v, el fabricante recomienda colocar entre estos dos pines un capacitor de bypass cerámico de 0.1µF. Opcionalmente se puede utilizar un capacitor electrolítico en el caso que se utilicen los puertos del MCU en su modo de alta corriente, en la figura 1.3 A) puede verse claramente la forma de conexión.

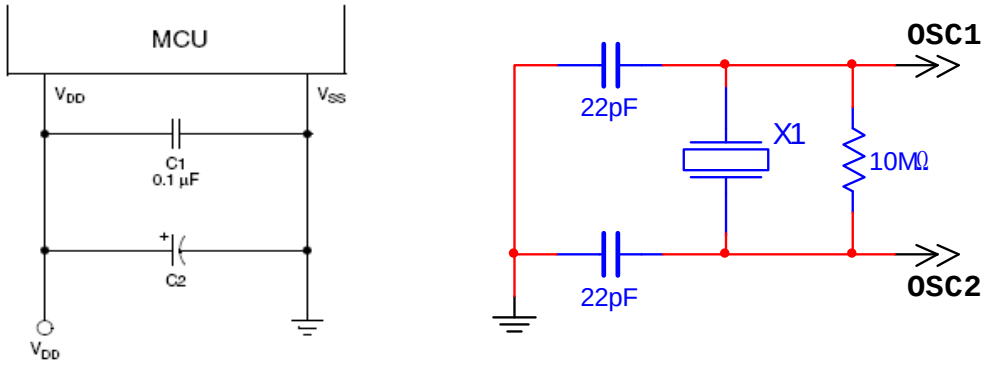


Figura 1.3: A) Polarización y bypass. B) Conexión del cristal

1.1.5 OSCILADOR Y CICLO MAQUINA

El ciclo maquina esta generado por cuatro periodos de reloj, el reloj es generado por un oscilador de cristal de cuarzo.
La velocidad máxima del BUS de este microcontrolador es de 8 MHz lo que quiere decir que para obtener esta frecuencia se necesita un cristal de 32 MHz.
En la figura 1.3 B) se observa el cristal X1 el cual puede ser de máximo 32 MHz, los capacitares pueden variar de 6pF a 30pF, este circuito oscilador debe ir conectado al los pines OSC1 y OSC2 para que el MCU genere el ciclo máquina.
Si el ciclo máquina es de 8 MHz entonces la duración de cada ciclo será 125ns.
Sabiendo la velocidad en ciclos de cada instrucción puede calcularse el tiempo de ejecución de cualquier programa.

Los MCU HC08 tienen opción a dos tipos de oscilador, al anteriormente mencionado y a un oscilador RC el cual solo necesita de un pin y así poder liberar al pin OSC2 y poder utilizarlo como un puerto de propósito general, si se desea colocar un oscilador RC refiérase al manual del fabricante.

Los MCU HC08 tienen dos modos de operación en cuanto a la generación del ciclo maquina se refiere, el primero se llama **modo usuario** el cual es el funcionamiento en que trabaja normalmente el MCU y la generación del ciclo máquina se basa en dividir entre 4 la velocidad del cristal y como velocidad máxima de cristal se puede emplear uno de 32 MHz dando así una velocidad de BUS máxima de 8 MHz, esto ya se a explicado, pero existe un segundo modo de operación llamado **modo monitor** el cual debe funcionar con un cristal de 4.9152 MHz o uno de 9.8304 MHz, si el cristal es de 4.9152 MHz este se dividirá entre 2 para obtener la velocidad del BUS , pero si el cristal usado es de 9.8304 MHz éste se divide entre 4 para obtener el ciclo maquina o velocidad del BUS, en cualquiera de los dos casos la velocidad de bus obtenida es de 2.4576 MHz esta es la velocidad que el Codewarrior[1] requiere para comunicarse a través del puerto serial con el MCU, además se puede hacer funcionar al MCU mientras esta conectado a la computadora después de ser grabado para reducir el tiempo de depuración del programa.

1 El Codewarrior es el software integrado de desarrollo que requieren los microcontroladores de Freescale el cual se puede descargar de http://freescale.com

Sin embargo si el programa esta diseñado para alguna otra velocidad de cristal y parte de su funcionamiento requiere precisas rutinas de retardo este no funcionará correctamente en modo monitor por lo que se recomienda probar el programa conectando al MCU al circuito final en el que trabajará con el cristal que fueron calculadas las rutinas y los demás procesos.

1.2 MAPA DE MEMORIA DEL MC68HC908GP32 ----------

1.2.1 DISTRIBUCIÓN DE LA MEMORIA

La arquitectura HC08 cuenta asta con 64 Kbytes de espacios de memoria, la memoria del MC68HC908GP32 se encuentra distribuida de la siguiente forma:

- 32256 Bytes de memoria flash.
- 512 Bytes de memoria RAM.
- 36 Bytes ocupados para la definición de vectores.
- 307 Bytes para el modo monitor.
- 32192 Bytes no implementados

Consulte la figura 1.4 para el mayor entendimiento del mapeo de la memoria, nótese que la memoria Flash, RAM y prácticamente todas las memorias se encuentran en una sola, esto quiere decir que el microcontrolador tiene una arquitectura interna del tipo Von Neumann.

1.2.2 SECCIÓN DE ENTRADA-SALIDA Y OTROS REGISTROS DE CONTROL

La sección de entrada y salida está comprendida de la localidad $0000 a la localidad $003F (el símbolo $ denota que se está hablando de un número Hexadecimal), los registros que se comprenden en esta sección están encargados del estatus y control del MCU.
Otros registros de control son:

- $FE00; SIM break status register, SBSR
- $FE01; SIM reset status register, SRSR
- $FE02; reserved, SUBAR
- $FE03; SIM break flag control register, SBFCR
- $FE04; interrupt status register 1, INT1
- $FE05; interrupt status register 2, INT2
- $FE06; interrupt status register 3, INT3
- $FE07; reserved
- $FE08; FLASH control register, FLCR
- $FE09; break address register high, BRKH
- $FE0A; break address register low, BRKL
- $FE0B; break status and control register, BRKSCR
- $FE0C; LVI status register, LVISR
- $FF7E; FLASH block protect register, FLBPR
- $FFFF; COP control register, COPCTL

NOTAS:

Si se accede a la memoria no implementada causará un reset por acceso a dirección de memoria ilegal.

Si se accede a localidades de memoria reservadas puede causar efectos impredecibles en el MCU.

$0000 ↓ $003F	I/O Registers 64 Bytes
$0040 ↓ $023F	RAM 512 Bytes
$0240 ↓ $7FFF	Unimplemented 32,192 Bytes
$8000 ↓ $FDFF	FLASH Memory 32,256 Bytes
$FE00	SIM Break Status Register (SBSR)
$FE01	SIM Reset Status Register (SRSR)
$FE02	Reserved (SUBAR)
$FE03	SIM Break Flag Control Register (SBFCR)
$FE04	Interrupt Status Register 1 (INT1)
$FE05	Interrupt Status Register 2 (INT2)
$FE06	Interrupt Status Register 3 (INT3)
$FE07	Reserved
$FE08	FLASH Control Register (FLCR)
$FE09	Break Address Register High (BRKH)
$FE0A	Break Address Register Low (BRKL)
$FE0B	Break Status and Control Register (BRKSCR)
$FE0C	LVI Status Register (LVISR)
$FE0D ↓ $FE0F	Unimplemented 3 Bytes
$FE10 ↓ $FE1F	Unimplemented 16 Bytes Reserved for Compatibility with Monitor Code for A-Family Parts
$FE20 ↓ $FF52	Monitor ROM 307 Bytes
$FF53 ↓ $FF7D	Unimplemented 43 Bytes
$FF7E	FLASH Block Protect Register (FLBPR)
$FF7F ↓ $FFDB	Unimplemented 93 Bytes
$FFDC ↓ $FFFF	FLASH Vectors 36 Bytes

Figura 1.4: Mapeo de la memoria

1.3 ARQUITECTURA INTERNA HC08----------

La arquitectura interna del HC08 es del tipo Von Neumann es decir la CPU ingresa a una sola memoria en la que las instrucciones y los datos, la RAM y la Flash, los registros de control y de estados están todos juntos en una sola memoria tal y como se muestra en la figura 1.4.

1.3.1 ARQUITECTURA VON NEUMANN

La arquitectura tradicional de computadores y microprocesadores está basada en la arquitectura Von Neumann, (figura 1.5) en la cual la unidad central de procesamiento (CPU) está conectada a una memoria única donde se guardan las instrucciones del programa y los datos.

El tamaño de la unidad de datos o instrucciones está fijado por el ancho del bus que comunica la memoria con la CPU. Así, un microprocesador de 8 bits con un bus de 8 bits tendrá que manejar datos e instrucciones de una o más unidades de 8 bits de longitud. Si tiene que acceder a una instrucción o dato de más de un byte de longitud, tendrá que realizar más de un acceso a la memoria.

El tener un único bus permite que la estructura interna sea más sencilla de implementar, y con ello un requerimiento de menor cantidad de silicio, lo que permite que estos dispositivos sean fáciles de construir, eficientes y a costos de fabricación y comercialización relativamente bajos, contrario a lo que ocurre con los dispositivos que poseen arquitectura Harvard. Por estas poderosas razones de estabilidad, simplicidad y bajo costo, la mayoría de fabricantes prefieren utilizar esta arquitectura en la gran mayoría de dispositivos que fabrican.

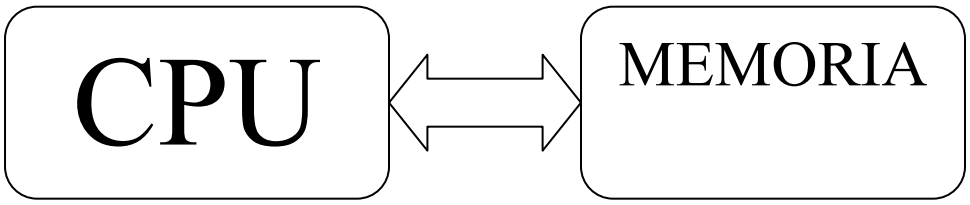


Figura 1.5: Arquitectura Von Neumann.

1.3.2 ARQUITECTURA HARVARD

Tiene la unidad central de procesamiento (CPU) conectada a dos memorias (una con las instrucciones y otra con los datos) por medio de dos buses diferentes (figura 1.6).

Una de las memorias contiene solamente las instrucciones del programa (memoria de programa), y la otra sólo almacena datos (memoria de datos).

Ambos buses son totalmente independientes y pueden ser de distintos anchos. Para un procesador de Conjunto de Instrucciones Reducido, o RISC (Reduced Instruction Set Computer), el conjunto de instrucciones y el bus de memoria de programa pueden diseñarse

de tal manera que todas las instrucciones tengan una sola posición de memoria de programa y la misma longitud.
Además, al ser independientes los buses, la CPU puede acceder a los datos para completar la ejecución de una instrucción y al mismo tiempo leer la siguiente instrucción a ejecutar.

Ventajas de esta arquitectura:

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

Entre las principales desventajas de los procesadores que tienen arquitectura Harvard están:
Se debe poseer instrucciones especiales para acceder a tablas de valores constantes que pueda ser necesario incluir en los programas, ya que estas tablas se encontrarán físicamente en la memoria de programa (por ejemplo, en la EPROM de un microprocesador).
Debido a que la arquitectura Harvard es más compleja que la arquitectura Von Neumann, requiere mayor cantidad de silicio para su construcción y con ello, un mayor costo de producción. La relación aproximada de costo entre un dispositivo con arquitectura Harvard es aproximadamente 50% más costoso con relación a un dispositivo de arquitectura Von Neumann.

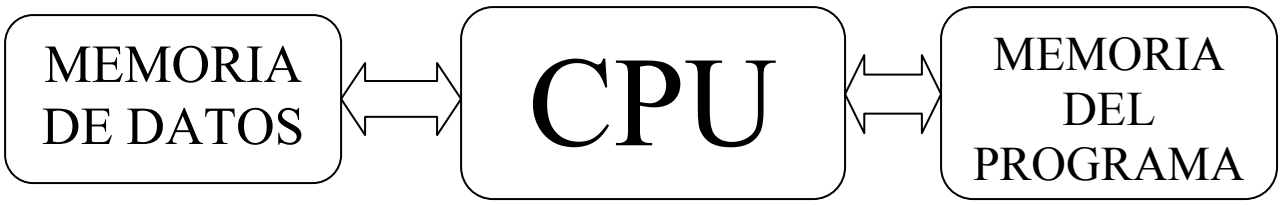


Figura 1.6: Arquitectura Harvard.

Para el caso de los microcontroladores Freescale, la arquitectura implementada es la Von Neumann.
Independientemente del MCU HC08 seleccionado sin importar su tamaño o conjunto de características estos implementan el mismo modelo de CPU.

1.4 FILOSOFIA DE LA ARQUITECTURA DEL MCU HC08----------

Según la filosofía de la arquitectura del procesador, se puede clasificar en:

- Microcontroladores CISC.
- Microcontroladores RISC.
- Microcontroladores SISC.

1.4.1 ARQUITECTURA CISC

Un microcontrolador basado en la filosofía CISC (computadores de juego de instrucciones complejo) dispone de más de 80 instrucciones máquina en su repertorio, algunas de las cuales son muy sofisticadas y potentes, requiriendo muchos ciclos para su ejecución.
Una ventaja de los procesadores CISC es que ofrecen al programador instrucciones complejas que actúan como macros. La desventaja de esta arquitectura es que debido al gran número de instrucciones no se considera en muchos casos llamativa para el ingeniero diseñador. Los microcontroladores HC08 son del tipo CISC.

1.4.2 ARQUITECTURA RISC

La siguiente arquitectura, conocida como arquitectura RISC (computadores de juego de instrucciones reducido) se considera como la más llamativa para el diseño de aplicaciones debido a su juego de instrucciones un poco más moderado en cuanto a cantidad que los CISC. Tanto la industria de los computadores comerciales como la de los microcontroladores están enfocándose hacia esta filosofía. En estos procesadores el repertorio de instrucciones es muy reducido y las instrucciones son simples y, generalmente, se ejecuta en un solo ciclo.
La sencillez y rapidez de las instrucciones permiten optimizar el hardware y el software del procesador.

1.4.3 ARQUITECTURA SISC

En los microcontroladores destinados a aplicaciones muy concretas, el juego de instrucciones, además de ser reducido, es específico, es decir, las instrucciones se adaptan a las necesidades de la aplicación prevista. Esta filosofía se ha bautizado con el nombre de SISC (computadores con conjunto de instrucciones específico).

1.5 REGISTROS DE USO GENERAL----------

Existen algunos registros que se encuentran interactuando constantemente con el CPU, realizando tareas de acuerdo con las instrucciones que se estén ejecutando. Uno de los detalles a tener en cuenta es que a pesar de que estos registros hacen parte constante en la programación de microcontroladores, son registros propios e internos de la CPU y por tal razón no se encuentran mapeados en la memoria. Estos registros son:

1.5.1 ACUMULADOR (A)

Es un registro de 8 bits para todo uso, utilizado en las operaciones aritméticas y lógicas. El acumulador se manipula normalmente para almacenar operandos, resultados de cálculos aritméticos, y de manipulación de datos, además del hecho de ser directamente accesible a la CPU para operaciones no aritméticas. El Acumulador no es afectado por el reset.

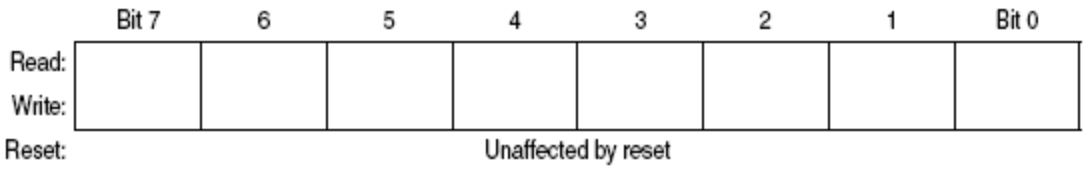


Figura 1.7: Acumulador

1.5.2 REGISTROS DE INDICE “H” E INDICE “X”

El registro de índice H:X admite 16 bits siendo H el byte mas significativo y X el byte menos significativo, esto nos da como resultado el registro H:X para poder así acceder a los 64 kilobytes de memoria del MCU a través del direccionamiento indexado, no obstante H y X son dos registros que dependiendo de la instrucción o circunstancia pueden actuar como dos registros independientes de 8 bits o como uno solo de 16 bits.

El registro X se emplea en los modos de direccionamiento indexado o puede trabajar como un acumulador secundario o auxiliar ya que hay diversas instrucciones que lo emplean como fuente o destino igual que al acumulador, esto es útil para disminuir la carga de trabajo del acumulador, de modo que se requiere de menos movimientos de datos entre memoria y acumulado, acelerando así la ejecución del programa o aplicación.
El H:X se utiliza en los modos de direccionamiento indexado y sirve como extensión, siendo capaz de direccionar todo el mapa de memoria. El valor que adquiere el registro H después de un reset es de $00. El valor que adquiere el registro X después de un reset es indeterminado.

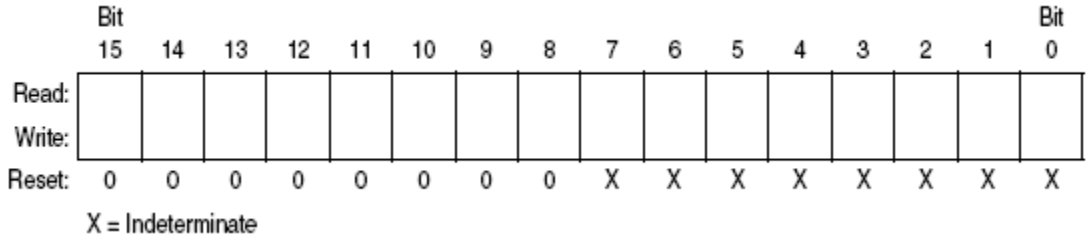


Figura 1.8: Registro H:X

1.5.3 STACK POINTER (SP)

El Stack Pointer o puntero de pila en español es un registro de 16 bits que contiene la dirección de la posición disponible en el stack. El stack pointer puede funcionar como un registro para direccionamiento indexado para acceder a datos en el stack.

La pila o stack se ubica en un rango de localidades de la memoria RAM esta se usa como una memoria del tipo LIFO (del inglés Last In First Out) que significa que el último dato que entra en la pila será el primero en salir mediante las instrucciones PUSH que coloca un dato sobre la pila y POP que retira el último dato colocado en la pila, tal cual sucede al apilar un

montón de platos es decir la pila sirve para almacenar objetos y luego recuperarlos de manera inversa al que fueron colocados.

Para insertar un objeto se utiliza la instrucción PUSH y para retirar un objeto se usa la instrucción POP.
El stack pointer es la punta de la pila es decir en el se encuentra la dirección siguiente al ultimo objeto colocado en la pila. El valor que adquiere el Stack Pointer después de un reset es de $00FF.

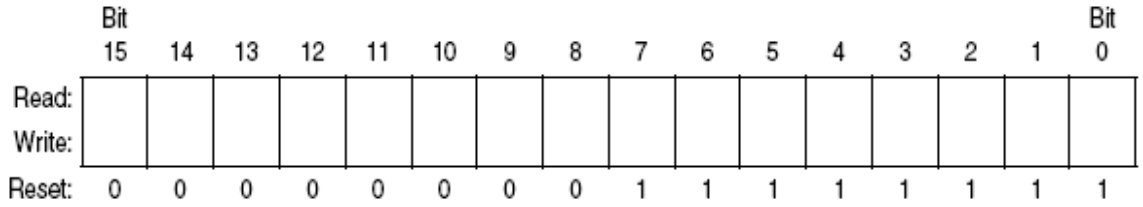


Figura 1.9: Stack Pointer

1.5.4 CONTADOR DE PROGRAMA (PC)

El Contador de Programa o PC (Program Counter) es un registro de 16 bits que contiene la dirección de la siguiente instrucción a ejecutar. A su vez el contador de programa incrementa la dirección de memoria de forma secuencial y automáticamente según el código de operación siguiente de instrucciones como saltar, bifurcar, e interrupciones.
En otras palabras, se podría decir que el registro contador de programa (PC) es usado por la CPU para no perder de vista la dirección de la próxima instrucción a ejecutar.
El valor que adquiere el contador de programa después de un reset o encender es el que se encuentra almacenado en el Vector de Reset. El vector de reset es un dato de 16 bits que se encuentra almacenado en las localidades de memoria $FFFE y $FFFF, este dato no es mas que la dirección en donde se encuentra la primera instrucción del programa grabado en el microcontrolador.
En muchos de los microcontroladores HC08, algunos de los bits más significativos del contador de programa no son usados y siempre están en cero esto se debe a que el número de bits utilizados en el contador de programa coinciden con el número de líneas de direcciones de memoria implementadas en el sistema del microcontrolador.

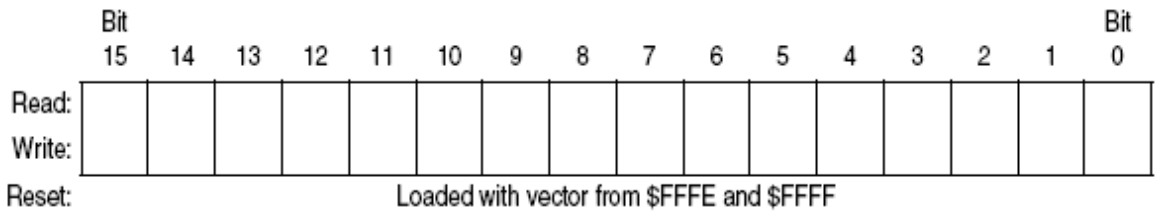


Figura 1.10: Contador de programa

1.5.5 REGISTRO DE BANDERAS (CCR)

El registro de banderas (Condition Code Register) esta formado de 8 bits entre los cuales uno de ellos es el bit de interrupción enmascarada, otros 5 son banderas de estado y los dos bits restantes el 5 y el 6 siempre permanecen en alto aun después de un reset. Las cinco banderas de estado reflejan el resultado de operaciones aritméticas y ciertas condiciones generadas por las instrucciones previas y ejecutadas. Las cinco banderas son bandera de rebosamiento (V), bandera de medio acarreo (H), bandera de negativo (N), bandera de cero (Z) y bandera de acarreo o préstamo (C); además del bit de mascara de interrupción o interrupción enmascarada (I).

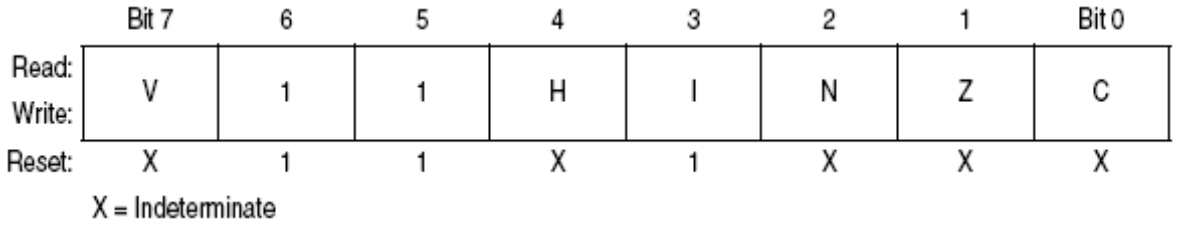


Figura 1.11: Registro de banderas.

La mayoría de las instrucciones de salto utilizan estas banderas como condición, a continuación se explican cada una de ellas.

1.5.6 V: BANDERA DE DESBORDAMIENTO O SOBRE FLUJO (OVERFLOW)

Esta bandera es el equivalente de C para operaciones con signo. Se activa si el resultado sale del rango de -128 a 127. Este bit puede ser modificado por instrucciones no aritméticas. La CPU coloca esta bandera en 1 cuando al efectuar el complemento a dos ocurre un rebosamiento.

1: Rebosamiento
0: No rebosamiento

1.5.7 H: BANDERA DE MEDIO ACARREO

Indica si existió un desbordamiento en los 4 primeros bits del resultado. Funciona de la misma manera que C para instrucciones aritméticas, pero considerando solo los 4 primeros bits del resultado. La CPU coloca este bit en 1 cuando ocurre un acarreo entre los bits 3 y 4 durante una suma con o sin acarreo. El medio acarreo es empleado cuando se utiliza codificación en BCD.

1: Acarreo entre los bits 4 y 3
0: No acarreo entre los bits 4 y 3

1.5.8 I: MASCARA DE INTERRUPCION

El bit I no es una bandera de estado, es un bit de mascara de interrupción. Este bit es el habilitador global de interrupciones, si este bit se encuentra en 1 todas las interrupciones enmascaradas quedan deshabilitadas, al atenderse una interrupción este bit pasa automáticamente a 1 impidiendo que se ejecute alguna otra interrupción, cuando se retorna de la interrupción este bit pasa a ser cero nuevamente. En el caso de que se esté atendiendo una interrupción, es decir el bit I esté en 1 y ocurra otra interrupción, esta ultima se almacenará para ser atendida en el momento en que el bit I regrese a ser cero (retorne de la interrupción), es decir la CPU puede mantener una interrupción en cola.
Luego de cualquier reset el bit I estará en 1y solo podrá llevarse a cero por medio de una instrucción ya que hay 2 instrucciones que pueden manipular directamente a este bit.

1.5.9 N: BANDERA DE VALOR NEGATIVO

Esta bandera muestra el estado del bit de signo (bit7) del resultado anterior. Este bit se pone a 1 cuando el resultado de una operación aritmética es negativa.

1: El resultado es negativo
0: El resultado es positivo

1.5.10 Z: BANDERA DE CERO

Este bit se pone en 1 cuando el resultado de una operación aritmética o lógica es cero. Como esta bandera no solo es modificada por operaciones aritméticas se activará siempre que el valor del registro de destino de la instrucción sea igual a cero.

1: El resultado es cero
0: El resultado no es cero

1.5.11 C: BANDERA DE ACARREO

En instrucciones aritméticas sin signo, esta bandera indica si ocurrió un desbordamiento en el rango de 0 a 255. Las instrucciones de corrimiento y rotación pueden hacer uso de este bit. Este bit se coloca a uno cuando el resultado de una operación aritmética produce acarreo después del bit 7. La bandera C se utiliza para indicar si ha habido o no acarreo de una suma o pedido de préstamo como resultado de una resta. Las instrucciones de desplazamiento y rotación operan sobre y a través del bit C para facilitar operaciones de desplazamiento de múltiples bytes. El bit C es afectado además durante las instrucciones de evaluación de bit y de bifurcación.

1.6 MODOS DE DIRECCIONAMIENTO----------

En todo proceso de programación se requiere realizar constantemente operaciones, asignación de valores a registros y extraer valores existentes en otros registros para transferirlos y/o procesarlos con otros datos; este proceso de extracción y adjudicación se conoce como **direccionamiento**.
Una de las propiedades que brindan actualmente los microcontroladores consiste en la habilidad para acceder a la memoria; aprovechando tal cualidad, los modos de direccionamiento existentes en la CPU proveen esta capacidad. Existe una gran variedad de formas de direccionar los datos, las cuales se clasifican según la forma en que una instrucción obtendrá el valor requerido para su ejecución. Debido a los diferentes modos de direccionamiento, una instrucción puede acceder al operando en una de las diversas maneras. Cada variante del modo de direccionamiento de una instrucción debe tener un único código de operación de instrucción. Los microcontroladores Freescale usan seis modos de direccionamiento, que son:

- Inherente
- Inmediato
- Extendido
- Directo
- Indexado sin desplazamiento
  Indexado con desplazamiento de 8 bits
  Indexado con desplazamiento de 16 bits
- Relativo

En los microcontroladores HC08 las localidades de memoria donde se encuentran los registros utilizados en el programa para el uso de variables y los registros de entrada – salida pueden variar, para los microcontroladores pequeños como el JK1 van de $0000 a $00FF mientras que en el GP32 van de $0000 a $003F, en estos registros el modo de direccionamiento mas usado es el directo ya que la dirección de estos registros se puede representar con un solo byte.

Antes de comenzar existe la necesidad exponer ciertos conceptos sobre el uso adecuado del sistema numérico en el que se desea expresar una cantidad en un momento dado. El espacio en blanco o ningún símbolo, o terminando la expresión con la letra T, indica que el número es decimal. Este número será trasladado a un valor binario antes de ser almacenado en memoria para ser usado por la CPU.

El símbolo $ precediendo a un número indica que el número es hexadecimal y hace referencia a una localidad de memoria; por ejemplo, $0F es la representación en hexadecimal de la localidad de memoria número 15 en decimal o base 10. Si al símbolo $ lo precede el símbolo # indica que el valor que lo acompaña es un valor constante es decir que se conoce a la hora de escribir el programa y además esta en hexadecimal; por ejemplo, #$0F es la representación de una constante con valor de 15 en decimal o base 10. Se debe tener en cuenta que existe una gran variedad de símbolos y expresiones que pueden utilizarse siguiendo al carácter #. Debido a que no todos los compiladores de ensamblador usan las mismas reglas de sintaxis ni los mismos caracteres

especiales, es necesario referirse a la documentación del ensamblador en particular que se esté utilizando en un modelo específico.

En la figura 1.12 puede verse el listado de prefijos para representar la cantidad en el sistema numérico deseado.

PREFIJO	Tipo de valor que representa
t	Decimal
$	Hexadecimal
@	Octal
%	Binario
Apóstrofe '	Carácter ASCII

Figura 1.12: Prefijos de representaciones numéricas.

1.6.1 MODO DE DIRECCIONAMIENTO INMEDIATO

En el modo de direccionamiento inmediato, el operando está contenido en el byte inmediato siguiente al código de operación. Este modo se usa cuando se requiere un valor o constante conocido en el momento de escribir el programa y que cumple con el hecho de que no cambiará durante la ejecución del programa.

Esta es una instrucción de dos bytes, uno para el código de operación y otro para el byte de dato inmediato. El símbolo "#" se antepone al operando el cual es un dato constante. Cualquier cosa que tenga ante puesto un ";" será ignorado y considerado tan solo como un comentario.

Ejemplos:

```
LDA    #$3F            ; A = $3F
```

Explicación:
Los pasos que se ejecutan en el momento de evaluar la anterior instrucción son los siguientes:

- La CPU almacena el nuevo valor en el acumulador, en este caso se almacena el valor de $3F o 63 en decimal y se ajustan las banderas requeridas según la operación.

```
LDX    #10T            ; X = 10
```

Explicación:
Los pasos que se ejecutan en el momento de evaluar la anterior instrucción son los siguientes:

- La CPU almacena el nuevo valor en el registro índice (X); en este caso se almacena el valor 10 y se ajusta las banderas requeridas según la operación.

A continuación se muestra el listado de instrucciones que permiten la ejecución del direccionamiento inmediato.

Instrucción	Mnemónico
Suma con acarreo	ADC
Suma sin acarreo	ADD
Función lógica AND	AND
Comparar el acumulador con	CMP
Comparar el registro índice con	CPX
Función OR exclusiva	EOR
Cargar el acumulador con memoria	LDA
Cargar el registro índice con	LDX
Función OR	ORA
Resta con acarreo	SBC

Figura 1.13: Instrucciones con modo de direccionamiento inmediato

1.6.2 MODO DE DIRECCIONAMIENTO INHERENTE

Este modo de direccionamiento se caracteriza porque toda la información requerida para la operación ya es implícitamente conocida por la CPU y no es necesario utilizar valores adicionales para su ejecución. En caso de requerirse algún operando en particular, son sólo los registros de la CPU o bien valores de datos almacenados en la pila.

Ejemplos:

```
INCA        ; Incrementar el acumulador
```

Explicación:

Los pasos que se ejecutan en el momento de evaluar la anterior instrucción son los siguientes:

- La CPU lee el valor que se encuentra almacenado en el registro Acumulador.
- La CPU le suma uno al valor actual del acumulador.
- La CPU almacena el nuevo valor en el acumulador y ajusta las banderas requeridas según la operación.

```
CLRA        ; Borrar el Acumulador
```

Los pasos que se ejecutan en el momento de evaluar la anterior instrucción son los siguientes:

- La CPU almacena el valor $00 en el acumulador (A), borrando toda información que se encontrara almacenada previamente en este registro y ajusta las banderas requeridas según la operación.

A continuación se listan las instrucciones que pueden usar el modo de direccionamiento inherente.

Instrucción	**Mnemónico**
Desplazamiento aritmético a la izquierda	ASLA, ASLX
Desplazamiento aritmético a la derecha	ASRA, ASRX
Borra el bit de carga	CLC
Borra el bit de mascara de interrupción	CLI
Limpiar	CLRA, CLRX
Complemento a uno	COMA, COMX
Decrementar en uno	DECA, DECX
Incrementar en uno	INCA, INCX
Desplazamiento lógico a la izquierda	LSLA, LSLX
Desplazamiento lógico a la derecha	LSRA, LSRX
Multiplicar	MUL
Negar	NEGA, NEGX
No operación	NOP
Rotar a la izquierda a través del bit de carga	ROLA, ROLX
Rotar a la derecha a través del bit de carga	RORA, RORX
Restaurar la pila	RSP
Retorno de interrupción	RTI
Retorno de subrutina	RTS
Colocar el bit de carga en 1	SEC
Colocar el bit de máscara de interrupción en 1	SEI
Habilitar IRQ, detener el oscilador	STOP
Interrupción por software	SWI
Transferir el acumulador al registro X	TAX
Probar que no sea negativo o cero	TSTA, TSTX
Transferir el registro X al acumulador	TXA

Figura 1.14: Instrucciones con modo de direccionamiento inherente

1.6.3 MODO DE DIRECCIONAMIENTO EXTENDIDO

Uno de los modos de direccionamiento más importantes dentro de la programación de los microcontroladores es el modo de direccionamiento extendido, que consiste en extraer la información almacenada en una dirección de memoria que para representarla requiere 2 bytes (16 bits), y se escribe su valor seguido de la instrucción. Este modo se emplea para hacer referencia a cualquier posición de memoria dentro del espacio de memoria del MCU, incluyendo direcciones de puertos de entrada/salida, direcciones de la memoria RAM, ROM, EPROM, Flash. En general, se podría decir que esta forma de direccionamiento se encuentra constituida por tres bytes, un primer byte para la instrucción a utilizar y otros dos para la dirección del operando. A continuación se ilustra un ejemplo que explica con mayor detalle este modo especial de direccionamiento:

Ejemplo:

Supongamos que en la posición de memoria $0367 se encuentra almacenado el valor $0F, entonces:

```
LDA   $0367                ;A = $0F,
```

La anterior instrucción extrae el valor que se encuentra almacenado en la dirección extendida $0367 y lo almacena en el Acumulador.

Explicación:

- La CPU lee la instrucción que significa cargar el acumulador usando el modo de direccionamiento extendido.
- La CPU lee el valor $03, el cual es interpretado como el valor correspondiente a los 8 bits de mayor peso de la dirección a leer.
- La CPU lee el valor $67, el cual es interpretado como el valor correspondiente a los 8 bits de menos peso de la dirección a leer.
- La CPU arma la dirección extendida completa $0367 con los dos valores previamente leídos, esta dirección es colocada en el bus de direcciones y la CPU leerá el valor almacenado en la posición de memoria $0367 almacenándolo en el Acumulador.
- En el Acumulador quedará almacenado el valor 0Fh para el caso del ejemplo.

A continuación se ilustra las instrucciones que permiten el uso del modo de direccionamiento extendido.

Instrucción	**Mnemónico**
Suma con acarreo	ADC
Suma sin acarreo	ADD
Función lógica AND	AND
Comparar el acumulador con memoria	CMP
Comparar el registro índice con memoria	CPX
Función lógica OR exclusiva	EOR
Saltar	JMP
Saltar a subrutina	JSR
Cargar el acumulador desde memoria	LDA
Cargar el registro índice desde memoria	LDX
Función lógica OR	ORA
Resta con acarreo	SBC
Resta sin acarreo	SUB

Figura 1.15: Instrucciones con modo de direccionamiento Extendido.

1.6.4 MODO DE DIRECCIONAMIENTO DIRECTO

Uno de los modos de direccionamiento mas utilizados en la programación de microcontroladores es el modo de direccionamiento directo; este es muy similar al modo de direccionamiento extendido, con la diferencia de que el byte correspondiente a la parte alta de la dirección del operando se asume con el valor $00, de tal forma que sólo es necesario incluir el byte de menos peso de la dirección del operando en la instrucción a ejecutar
El hecho de considerar que la parte alta de la dirección de memoria a acceder se considere como 00h delimita en cierta manera los registros y recursos del microcontrolador a acceder mediante este modo direccionamiento. Esta área de memoria en especial se denomina página directa, la cual incluye parte de los registros de la memoria RAM y puertos de entrada/salida del interior del chip.
En general, se podría decir que este modo de direccionamiento es muy eficiente en factores fundamentales a la hora de programar y desarrollar aplicaciones como son: menor espacio de memoria de programa, menor tiempo de ejecución, entre otros factores, debido a que ésta es una instrucción de dos bytes, el primero utilizado para la instrucción a ejecutar y el otro para el byte de menos peso de la dirección de memoria a acceder.
A continuación se ilustra un ejemplo que explica con mayor detalle este modo especial de direccionamiento:

Ejemplo:

Supongamos que en la posición de memoria $80 se encuentra almacenado el valor $0F, entonces:

```
LDA $80     ; A=$0F          ;es decir el valor 0Fh (15 en base 10)
```

La anterior instrucción extrae el valor que se encuentra almacenado en la dirección 80h y lo almacena el registro Acumulador (A).

Explicación:

- La CPU lee la instrucción que significa cargar el acumulador usando el modo de direccionamiento directo.
- La CPU lee el valor $80, el cual es interpretado como el valor correspondiente a los 8 bits de menos peso de una dirección de pagina directa (desde $0000 hasta $00FF).
- Los 8 bits de más peso de la dirección a acceder se establecen como $00.
- La CPU arma la dirección completa de página directa $0080 con los dos valores previamente mencionados, colocando esta dirección en el bus de direcciones y la CPU leerá el valor almacenado en la posición de memoria $0080 almacenándolo en el registro Acumulador (A).
- En el Acumulador quedará almacenado el valor 0Fh para el caso del ejemplo.

La siguiente figura muestra las instrucciones que emplean el modo de direccionamiento directo.

Instrucción	Mnemónico
Desplazamiento aritmético a la izquierda	ASL
Desplazamiento aritmético a la derecha	ASR
Limpiar	CLR
Complementar	COM
Decrementar	DEC
Incrementar	INC
Desplazamiento Lógico a la izquierda	LSL
Desplazamiento Lógico a la derecha	LSR
Negar	NEG
Rotar a la izquierda a través del carry	ROL
Rotar a la derecha a través del carry	ROR
Probar que no sea negativo o cero	TST
Almacenar el valor del acumulador	STA
Almacenar el valor del registro de índice X	STX
Suma con acarreo	ADC
Suma sin acarreo	ADD
Función lógica AND	AND
Compara el acumulador con memoria	CMP
Compara el registro índice con memoria	CPX
Función lógica OR exclusiva	EOR
Saltar	JMP
Saltar a subrutina	JSR
Cargar el acumulador desde memoria	LDA
Cargar el registro índice desde memoria	LDX
Función lógica OR	ORA
Resta con acarreo	SBC
Resta sin acarreo	SUB
Saltar si el bit n es 1	BRSET
Saltar si el bit n es 0	BSCLR
Colocar en uno el bit indicado en memoria	BSET
Limpiar el bit indicado en memoria	BCLR

Figura 1.16: Instrucciones con modo de direccionamiento Directo.

1.6.5 MODO DE DIRECCIONAMIENTO INDEXADO

Uno de los problemas que surgen en el momento de requerir almacenar 10, 20 o más valores en direcciones de memoria consecutivas utilizando los modos de direccionamiento anteriores sería el hecho de tener que escribir el código de instrucciones de almacenamiento tantas veces como valores se desee almacenar, lo cual generaría un trabajo bastante tedioso; por esta razón se planteó el modo de direccionamiento indexado. Este modo particular de direccionamiento consiste en que la dirección efectiva del operando es variable y depende de dos factores:

- El valor almacenado actualmente en el registro índice (X)
- El desplazamiento contenido en el byte o bytes siguientes al código de operación.

Este modo de direccionamiento se puede clasificar de tres maneras diferentes:

- sin desplazamiento
- con desplazamiento de 8 bits
- con desplazamiento de 16 bits

1.6.6 INDEXADO SIN DESPLAZAMIENTO

La primera subclase del modo de direccionamiento indexado denominada indexado sin desplazamiento consiste en que la dirección efectiva del operando para la instrucción está contenida en los 16 bits del registro índice H:X. De tal forma que este modo de direccionamiento puede acceder a todas las posiciones de memoria (desde $0000 hasta $FFFF), considerando el hecho de que la dirección que se tomará como punto de partida o de desplazamiento con relación al valor almacenado en el registro índice será $00 (00h). Hay que tener en cuenta que esta instrucción es de un solo byte.

A continuación se ilustra un ejemplo que explica con mayor detalle este modo especial de direccionamiento.

Ejemplo:

Supongamos que en la posición de memoria $80 se encuentra almacenado el valor $0F y se desea acceder a esta posición utilizando el modo de direccionamiento indexado sin desplazamiento (H es igual a 0); entonces:

```
LDX   #$80   ; X = $80 o (80h) dirección a ser apuntada por el registro X

LDA   0,X
```

;Carga el registro A con el valor almacenado previamente en la dirección 80h de la memoria que para el ejemplo corresponde al valor $0F (0Fh).

La anterior secuencia de instrucciones ilustra la forma de almacenar en el registro Acumulador (A), el valor almacenado en la dirección 80h, apuntada por el registro índice (X).

Explicación:

- La CPU lee la instrucción que permitirá almacenar un valor existente en una posición de memoria en particular en el registro acumulador (A), utilizando el modo de direccionamiento indexado sin desplazamiento.

- La CPU construye la dirección completa a acceder sumando $0000 al contenido del registro índice H:X de 16 bits, registro que actualmente, para el caso del ejemplo, presenta un valor almacenado de $0080 (80h).
- La dirección resultante es colocada en el bus de direcciones y la CPU lee el valor del registro contenido en esa posición de memoria y lo carga en el registro acumulador.

1.6.7 INDEXADO CON DESPLAZAMIENTO DE 8 BITS

La segunda subclase del modo de direccionamiento indexado denominada indexado con desplazamiento de 8 bits consiste en que la dirección efectiva es la suma del contenido del registro índice de 16 bits (H:X) y el byte de desplazamiento siguiente a la instrucción a ejecutar. Hay que tener en cuenta que el byte de desplazamiento suministrado en la instrucción es un número entero no signado de 8 bits. La dirección que se tomará como punto de partida o de desplazamiento con relación al valor almacenado en el registro índice será la estipulada por el valor del byte de desplazamiento siguiente a la instrucción a ejecutar.
A continuación se ilustra un ejemplo que explica con mayor detalle este modo especial de direccionamiento.

Ejemplo:

Supongamos que en la posición de memoria $85 se encuentra almacenado el valor $0F y se desea acceder a esta posición utilizando el modo de direccionamiento indexado con desplazamiento de 8 bits (H es igual a 0); entonces:

```
LDX   #$80      ;X = $80 o (80h) dirección a ser apuntada por el registro X

LDA   5,X
```

;Carga el registro A con el valor almacenado en la dirección de memoria correspondiente a la suma del valor del registro índice H:X ($0080) y el valor constante que acompaña la instrucción (5); en otras palabras, $0080+$5=$85, lo cual produce que el valor almacenado en la dirección $85 sea almacenado en el registro Acumulador, que para el ejemplo corresponde al valor $0F (0Fh).

La anterior secuencia de instrucciones ilustra la forma de almacenar en el registro Acumulador (A) el valor almacenado en la dirección 85h, apuntada por el registro índice (X) y complementada por un valor de corrimiento constante.

Explicación:

- La CPU lee la instrucción que permitirá almacenar un valor existente en una posición de memoria en particular en el registro acumulador (A), utilizando el modo de direccionamiento indexado con desplazamiento de 8 bits.
- La CPU construye la dirección completa a acceder sumando el valor constante que acompaña la instrucción (5), estableciéndose como punto de partida la $0005 con el

contenido del registro índice H:X de 16 bits, registro que actualmente, para el caso del ejemplo, presenta un valor almacenado de $0080 (80h).

- La dirección resultante ($0080+$5=$85) es colocada en el bus de direcciones y la CPU lee el valor del registro contenido en esa posición de memoria y lo carga en el registro acumulador (A).

Instrucción	**Mnemónico**
Suma con acarreo	ADC
Suma sin acarreo	ADD
Función lógica AND	AND
Desplazamiento aritmético a la izquierda	ASL
Desplazamiento aritmético a la derecha	ASR
Limpiar	CLR
Comparar el acumulador con memoria	CMP
Complementar	COM
Comparar el registro índice con memoria	CPX
Decrementar	DEC
Función lógica OR exclusiva	EOR
Incrementar	INC
Saltar	JMP
Saltar a subrutina	JSR
Cargar el acumulador desde memoria	LDA
Cargar el registro índice desde memoria	LDX
Desplazamiento lógico a la izquierda	LSL
Desplazamiento lógico a la derecha	LSR
Negar	NEC
Función lógica OR	ORA
Rotar a la izquierda a través del carry	ROL
Rotar a la derecha a través del carry	ROR
Resta con acarreo	SBC
Almacenar el valor del acumulador	STA
Almacenar el valor del registro de índice	STX

Figura 1.17: Instrucciones con modo de direccionamiento indexado sin desplazamiento y con desplazamiento de 8 bits.

1.6.8 INDEXADO CON DESPLAZAMIENTO DE 16 BITS

La tercera subclase del modo de direccionamiento indexado es la denominada indexado con desplazamiento de 16 bits; la dirección efectiva es la suma del contenido del registro índice H:X de 16 bits y los de 2 bytes de desplazamiento siguientes a la instrucción a ejecutar. Se debe tener en cuenta que el byte de desplazamiento suministrado en la instrucción es un número entero sin signo de 16 bits. En resumen, se puede decir que este modo especial de direccionamiento constituye una instrucción de tres bytes, uno para la instrucción a ejecutar *y* los otros dos bytes son para el desplazamiento.

Una de las cosas que se deben tener en cuenta es que el hecho de contar con la suma de dos registros, uno de 8 bits (si solo se utiliza X y H es igual a 0) y el otro de 16 bits, generará un resultado de 16 bits, provocando que se pueda acceder a posiciones de memoria en modo extendido.

A continuación se muestra un ejemplo que explica este modo especial de direccionamiento.

Ejemplo:

Supongamos que en la posición de memoria $0315 se encuentra almacenado el valor $0F y se desea acceder a esta posición utilizando el modo de direccionamiento indexado con desplazamiento de 16 bits (H es igual a 0); entonces:

```
LDX   #$5          ; X = $5 o (5h) dirección a ser apuntada por el registro X

LDA   $0310,X
```

; Carga el registro A con el valor almacenado en la dirección de memoria correspondiente a la suma del valor del registro índice H:X ($0005) y el valor constante de 16 bits que acompaña la instrucción ($0310); en otras palabras, $0310+$0005=$0315, lo cual produce que el valor almacenado en la dirección $0315 sea almacenado en el registro Acumulador, que para el ejemplo corresponde al valor $0F (0Fh).
La anterior secuencia de instrucciones ilustra la forma de almacenar en el registro Acumulador (A) el valor almacenado en la dirección 0315h, apuntada por el registro índice (X) y complementada por un valor de corrimiento constante de 16 bits.

Explicación:

- La CPU lee la instrucción que permitirá almacenar un valor existente en una posición de memoria en particular en el registro acumulador (A), utilizando el modo de direccionamiento indexado con desplazamiento de 16 bits.
- La CPU construye la dirección completa a acceder sumando el valor constante que acompaña la instrucción ($0310), estableciéndose como punto de partida la $0310 con el contenido del registro índice de 8 bits (X), registro que actualmente, para el ejemplo, presenta un valor almacenado de $05 (05h).
- La dirección resultante ($0310+$5=$0315) es colocada en el bus de direcciones y la CPU lee el valor del registro contenido en esa posición de memoria y lo carga en el registro acumulador (A).

Instrucción	Mnemónico
Suma con acarreo	ADC
Suma sin acarreo	ADD
Función lógica AND	AND
Desplazamiento aritmético a la izquierda	ASL
Desplazamiento aritmético a la derecha	ASR
Comparar el acumulador con memoria	CMP
Comparar el registro índice con memoria	CPX
Función lógica OR exclusiva	EOR
Saltar	JMP
Saltar a subrutina	JSR
Cargar el acumulador desde memoria	LDA
Cargar el registro índice desde memoria	LDX
Función lógica OR	ORA
Rotar a la izquierda a través del carry	ROL
Rotar a la derecha a través del carry	ROR
Resta con acarreo	SBC
Almacenar el valor del acumulador	STA
Almacenar el valor del registro de índice X	STX
Resta sin acarreo	SUB

Figura 1.18: Instrucciones con modo de direccionamiento indexado con desplazamientode 16 bits.

NOTAS:

Los pocos autores que manejan microcontroladores Freescale y el manual mismo de los MCU no mencionan esto pero la localidad a la que se accede en el modo de direccionamiento indexado (en sus tres modalidades) no esta designada por el registro índice X si no por el registro índice H:X es decir si H posee algún valor este será parte de la dirección a acceder es decir:

```
LDHX   #$0200
LDX    #$40
LDA    0,X
```

¿Desde que localidad de memoria se cargó el acumulador?
Si creíste que desde la localidad $0240 estas en lo correcto. El ignorar el registro de índice H es un error común en programadores que se inician en los HC08, HCS08 y RS08 es decir en microcontroladores Freescale.
Por lo cual recomiendo siempre limpiar el registro H antes de usar este modo de direccionamiento ya que hay operaciones como la división y multiplicación que utilizan este registro para los resultados, es mas, cuando se utiliza el codewarrior para la programación este graba automáticamente unas instrucciones que inicializan el stack pointer en el caso del MC68HC908GP32 queda almacenado en el registro H:X el valor $0240 que corresponde al inicio del stack. Hay solo una forma manual de modificar el registro H desafortunadamente también afecta al registro X: LDHX #$0000
Para limpiar el registro H basta con la instrucción: CLRH

1.6.9 MODO DE DIRECCIONAMIENTO RELATIVO

Uno de los modos de direccionamiento especiales y que se considera de gran ayuda a la hora de programar microcontroladores es el llamado modo de direccionamiento relativo, el cual es usado solamente por instrucciones de bifurcación (saltos condicionados). Se debe tener en cuenta que las instrucciones de bifurcación, a excepción de las bifurcaciones en su versión de manipulación de bits, generan dos bytes de código máquina: el primer byte se utiliza para la instrucción y otro byte para el desplazamiento relativo. Como muchos casos se desea que el salto o bifurcación sea en cualquier sentido (hacia arriba o hacia abajo), el byte de desplazamiento es un número que puede presentar bit de signo, oscilando en el rango entre -128 ha +127 bytes (respecto a la dirección de la instrucción inmediata posterior a la instrucción de bifurcación). Si la condición de salto es verdadera, el contenido de los 8 bits del byte con signo siguiente al código operación (desplazamiento) es sumado al contenido del contador de programa para formar la dirección de bifurcación efectiva; de otro modo, el control continúa en la siguiente instrucción inmediata posterior a la instrucción de bifurcación. A continuación se ilustra un ejemplo para explicar con mayor detalle el modo de direccionamiento relativo.

Ejemplo:

```
LDA    #40T      ; A = 40, Acumulador = 40
CMP    #30T      ; Compara A con 30
BNE    Saltol
```

; Si no son iguales salta a la etiqueta "Saltol", de lo contrario sigue en la siguiente línea; como en este caso no son iguales, se produce el salto.

Explicación:

- La CPU lee la instrucción de saltar hacia la etiqueta "Saltol" en caso de que al efectuar el proceso de comparación el bit Z = 0.
- La CPU lee la dirección de memoria denominada en este caso $XX, en donde la dirección $XX es interpretada como el valor de desplazamiento relativo. Después de realizado este proceso, el contador de programa apunta al primer byte de la próxima instrucción a ejecutar.
- Si el bit Z =1, no se produce ninguna bifurcación o salto y el programa debe continuar con la próxima instrucción. De lo contrario, si el bit Z = 0, la CPU generará la dirección completa sumando el desplazamiento con signo mencionado anteriormente ($XX) con el valor existente en el contador de programa para obtener la dirección destino de la bifurcación o salto, provocando que la ejecución del programa continúe desde otro punto del programa (Saltol).

La figura siguiente incluye una lista de todas las instrucciones que se pueden usar en el modo de direccionamiento relativo.

Instrucción	Mnemónico
Saltar a la etiqueta si el bit de carry es 0	BCC
Saltar a la etiqueta si el bit de carry es 1	BCS
Saltar a la etiqueta si es igual (bit Z = 1)	BEQ
Saltar a la etiqueta si el bit de carry medio es	BHCC
Saltar a la etiqueta si el bit de carry medio es	BHCS
Saltar a la etiqueta si es mayor	BHI
Saltar si es mayor o igual	BHS
Saltar si el Pin IRQ está en alto	BIH
Saltar si el pin IRQ está en bajo	BIL
Saltar a la etiqueta si es menor	BLO
Saltar a la etiqueta si es menor o igual	BLS
Saltar si la bandera de interrupción está en 0	BMC
Saltar si el resultado de una operación es	BMI
Saltar si la bandera de interrupción está en	BMS
Saltar a la etiqueta si no es igual	MNE
Saltar si el resultado de una operación es	BPL
Saltar a la etiqueta siempre	BRA
Saltar si el bit N del registro OPR está en 0	BRCLR
Nunca saltar	BRN
Saltar si el bit N del registro OPR está en 1	BRSET
Poner en 1 el bit N del registro OPR	BSET
Saltar a subrutina	BSR

Figura 1.19: Instrucciones con modo de direccionamiento Relativo.

NOTAS:

En el caso de tener que bifurcar saliendo del rango de -128 a + 127 será imposible por lo cual tendrá que re-direccionar mandando su bifurcación hacia la instrucción JMP la cual genera saltos con direccionamiento directo o extendido por lo que puede abarcar todo el mapa de memoria. Que quede claro que por convención hablamos de "saltar" en la explicación del direccionamiento relativo pero en realidad es muy diferente saltar que bifurcar puesto que al bifurcar se cuentan desplazamientos en la memoria y al saltar se apunta directo en el contador de programa la dirección a la que se desea ir. Por ejemplo:

```
ATRÁS:    CBEQA #$FF, SALTO     ; Bifurca hacia salto cuando A sea igual a $FF
          INCA                  ; Incrementa en uno al acumulador
          BRA   ATRÁS           ; Bifurca hacia ATRÁS
SALTO:    JMP   RE_SALTO        ; Salta asta RE_SALTO
          ..... ¿?              ; CONJUNTO DE INSTRUCCIONES QUE SUPERAN
          ..... ¿?              ; EN CANTIDAD DE BYTES LOS +127 DADOS POR
          ..... ¿?              ; EL RANGO DE DESPLAZAMIENTO DEL MODO DE
          ..... ¿?              ; DIRECCIONAMIENTO RELATIVO.
RE_SALTO: STOP
```

1.6.10 MODO DE DIRECCIONAMIENTO DE MEMORIA A MEMORIA

Este modo de direccionamiento exclusivo de la instrucción MOV tiene la particularidad de que por cada vez que la instrucción se ejecuta la CPU utiliza dos direccionamientos diferentes.

- Mover de: Inmediato – Directo

```
MOV     #$55 ,$40      ; Mueve el dato #$55 a la localidad de
                       ; Memoria $40
```

- Mover de: Directo – Directo

```
MOV     $40 ,$41       ; mueve el dato contenido en la localidad $40
                       ; a la localidad $41
```

- Mover de: Directo – Indexado

```
MOV     $40,X+         ;Mueve el dato contenido en la localidad $40
                       ; y ponlo en localidad apuntada en el registro
                       ; H:X
```

- Mover de: Indexado a Directo

```
MOV     X+,$40         ; Mueve el dato que se encuentra en la
                       ; localidad de memoria apuntada en el
                       ; registro H:X y ponlo en la localidad $40
```

1.7 COMO LEER EL COJUNTO DE INSTRUCCIONES----------

El 68HC05 tiene un total de 85 instrucciones que en la actualidad todavía forman un conjunto básico de instrucciones funcional y potente. El conjunto de instrucciones del 68HC08 añade 28 instrucciones al conjunto de instrucciones del 68HC05.

Se explica detalladamente todo el conjunto de instrucciones HC08 y no se limita esta explicación a lo que brinda el manual otorgado por el fabricante el cual puede consultarlo en el **anexo 1** al final de este trabajo, que aun que es muy completo es difícil de entender. Se debe tener en cuenta que solo se utilizan datos en Hexadecimal.

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				

Figura 1.20: Títulos en el set de instrucciones.

1.7.1 SOURCE FORM

CÓDIGO FUENTE. En esta columna puede verse como se escribe el Mnemónico correspondiente a la instrucción deseada y el tipo de operando con el que es compatible, debe recordarse que las instrucciones que trabajan en modo de direccionamiento Inherente no utilizan operando.

DESCRIPCION DE LOS OPERANDOS. Los operandos nos indican que tipo de dato manipulará la instrucción, y debido a esto el operando, también define el modo de direccionamiento que se esta empleando. Los operandos se encuentran a la derecha del mnemónico en la columna SOURCE FORM.

- **#opr:** Indicativo de un dato conocido a la hora de escribir el programa o constante propio del direccionamiento inmediato.

```
LDA    #$88        ;Carga en el acumulador el dato $88
```

- **opr:** Indicativo de que el dato se encuentra en una localidad de memoria directa o extendida, esto ultimo depende de la columna **ADRESS MODE** ya que hay instrucciones que manejan los modos de direccionamiento directo y extendido y otras que solo el directo.

```
LDA    $88         ;Carga en el acumulador el dato contenido en la
                   ;dirección de memoria directa $88.

LDA    $0100       ;Carga en el acumulador el dato contenido en la
                   ;dirección de memoria extendida 0100.

LDA    REG_A       ;Carga en el acumulador el valor contenido en el
                   ;registro A. Que el registro A se encuentre en una
                   ;localidad de memoria directa o extendida depende
                   ;de el programador.
```

- **opr,X:** Indicativo de modo de direccionamiento indexado con desplazamiento de 8 bits o 16 bits, esto ultimo depende de la columna **ADRESS MODE** ya que hay instrucciones que manejan los modos de direccionamiento indexados con desplazamiento de 8 y 16 bits y otras que solo lo manejan con el desplazamiento de 8 bits.

```
LDA    $0100,X     ;Si H = $00 y X = $02 entonces:
                   ;Carga en el acumulador el dato contenido en la
                   ;dirección de memoria $0102.

LDA    $80,X       ;Si H = $00 y X = $08 entonces:
                   ;Carga en el acumulador el dato contenido en la
                   ;dirección de memoria $88.
```

```
LDA     REG_A,X   ;Si H = $00,  X = $08 y el registro A se encuentra
                   :en la localidad de memoria directa $80 entonces:
                  ;Carga en el acumulador el dato contenido en la
                  ;dirección de memoria $88.

LDA     REG_B,X   ;Si H = $00, X = $08 y el registro B se encuentra
                   :en la localidad de memoria extendida
                   ;$0100entonces:
                  ;Carga en el acumulador el dato contenido en la
                  ;dirección de memoria extendida $0108
```

- **,X:** Indicativo de modo de direccionamiento indexado sin desplazamiento.

```
LDA     $0,X      ;Si H = $00 y X = $88 entonces:
                  ;Carga en el acumulador el dato contenido en la
                   ;dirección de memoria directa $88
```

- **opr,SP:** Indicativo de modo de direccionamiento indexado a través del Stack Pointer, con desplazamiento de 8 y 16 bits esto ultimo depende de la columna **ADRESS MODE** ya que hay instrucciones que manejan los modos de direccionamiento con desplazamiento de 8 y 16 bits y otras que solo lo manejan con el desplazamiento de 8 bits. Este modo de direccionamiento es idéntico al indexado solo que este en lugar de usar el registro H:X utiliza el Stack Pointer.

- **opr,opr:** Indicativo del modo de direccionamiento de directo a directo, este modo de direccionamiento es exclusivo de la instrucción MOV.

```
MOV     $40,$41          ;mueve el contenido de la localidad de
                         ;memoria $40 a la localidad de memoria
                         ;$41.

MOV     REG_A,REG_B      ;si y solo si los 2 registros están en
                          ;localidades de memoria directas.
                          ;mueve el contenido del registro A al B
```

- **#opr,opr:** Indicativo del modo de direccionamiento de inmediato a directo, este modo de direccionamiento es exclusivo de la instrucción MOV.

```
MOV     #$55,$40         ;lleva el dato $55 a la localidad de memoria
                          ;$40

MOV     #$55,REG_A       ;si y solo si el registro A se encuentra en una
                          ;localidad de memoria directa
```

- **opr,X+:** Indicativo del modo de direccionamiento de directo a indexado, este modo de direccionamiento es exclusivo de la instrucción MOV.

```
MOV     $40,X+      ;lleva el dato contenido en la localidad de
                    ;memoria $40 a la localidad de memoria
                    ;apuntada en el registro H:X

MOV     REG_A,X+    ;si y solo si el registro A se encuentra en una
                    ;localidad de memoria directa.
                    ;el dato contenido en el registro es colocado en la
                    ;dirección apuntada en el registro H:X
```

- **X+,opr:** Indicativo del modo de direccionamiento de indexado a directo, este modo de direccionamiento es exclusivo de la instrucción MOV.

```
MOV     X+,$40      ;lleva el dato contenido en la localidad de
                    ;memoria apuntada en el registro H:X a la
                    ;localidad de memoria $40

MOV     X+,REG_A    ;si y solo si el registro A se encuentra en una
                    ;localidad de memoria directa.
                    ;lleva el dato contenido en la localidad de
                    ;memoria apuntada en el registro H:X al
                    ;registro A
```

- **n,opr:** Este tipo de operando hace referencia a un solo bit de alguna localidad de memoria directa por ejemplo, pon a 0 el bit 0 del puerto B

```
BCLR    0,PTB   ; pon a 0 el bit 0 del puerto B
```

- **rel:** Indicativo del modo de direccionamiento relativo. Este tipo de operando se utiliza para bifurcaciones, el dato del operando se refiere al número de desplazamiento que recibirá el contador del programa para ubicarse en su nueva posición.
  En el momento en que nosotros escribimos el programa colocamos etiquetas para las direcciones de las bifurcaciones sin embargo, el programa compilador se encarga de calcular los desplazamientos para colocarlo en el código objeto como datos de desplazamiento en lugar de etiquetas.

Ejemplo:

¿Que es lo que en realidad esta escrito en memoria suponiendo que la subrutina siguiente esta escrita en la localidad de memoria ROM $8100?

```
Subrutina_1:
            BRSET   7,PTA,etiqueta_1    ;Bifurca si el bit 7 del puerto A esta en 1
            NOP                         ;No operar
            NOP                         ;No operar
            NOP                         ;No operar
            NOP                         ;No operar
            STOP                        ;Detener
etiqueta_1: RTS                         ;Retorna de subrutina
```

Explicación:

Localidad de memoria	Código Objeto	Explicación
$8100	$0E	Código de operación para la instrucción BRSET del bit 7
$8101	$00	Dirección del puerto A
$8102	$05	Número de desplazamientos
$8103	$9D	Código de no operación
$8104	$9D	Código de no operación
$8105	$9D	Código de no operación
$8106	$9D	Código de no operación
$8107	$8E	Código de operación para Detener al micro
$8108	$81	Código de operación para la instrucción Retorna de subrutina

En un principio el contador de programa tiene un valor de $8100 que corresponde a la primera instrucción de la subrutina. Si la condición no se cumple la CPU toma el valor del contador de programa y le suma $03 (debido a los tres bytes que se necesitan para escribir esta instrucción) lo que da un resultado de $8103 esto se almacena en el contador de programa lo que genera un desplazamiento de 3 localidades de memoria por consiguiente, la siguiente instrucción a ejecutarse es el primer NOP.

Si la condición se cumple entonces para generar el desplazamiento la CPU toma el valor del contador de programa y le suma $03 además de $05 dándonos como resultado $8108, esto se coloca en el contador de programa lo que provoca que sean saltadas las cuatro instrucciones NOP y la instrucción STOP llegando asta donde esta grabada la instrucción RTS. El dato $05 a sido calculado por el programa compilador nosotros escribimos etiquetas para facilitar el proceso de programación.

- **n,opr,rel:** Una simple combinación entre los operandos “n,opr” y “rel” que francamente es predecible el resultado como ejemplo tenemos el siguiente. Bifurca si el bit 4 del puerto A esta puesto en 1.

```
BRSET   4,PTA, etiqueta_1
```

- **opr,rel**: Una simple combinación entre los operandos "opr" y "rel". El resultado de esto es sencillo, una combinación entre el modo de direccionamiento directo con el relativo, como ejemplo: compara y bifurca si el dato contenido en la localidad de memoria es igual al acumulador.

```
CBEQ    $40, etiqueta_1
```

Pueden utilizarse etiquetas siempre y cuando el registro este en una localidad de memoria directa:

```
CBEQ    REG_A, etiqueta_1
```

- **#opr,rel**: Una simple combinación entre los operandos "#opr" y "rel". El resultado de esto es sencillo, una combinación entre el modo de direccionamiento inmediato con el relativo, como ejemplo: compara y bifurca si el dato inmediato es igual al registro X.

```
CBEQX     #$FE, etiqueta_1
```

- **opr,X+,rel:** Una simple combinación entre los operandos "opr,X" y "rel". El resultado de esto es sencillo, una combinación entre el modo de direccionamiento indexado con desplazamiento de 8 bits y el relativo, como ejemplo: compara y bifurca si el dato contenido en la localidad de memoria apuntada en el registro índice H:X mas el desplazamiento es igual al acumulador.

```
CBEQ     $05,X+,  etiqueta_1
```

- **X+,rel:** Una simple combinación entre los operandos "X" y "rel". El resultado de esto es sencillo, una combinación entre el modo de direccionamiento indexado sin desplazamiento y el relativo, como ejemplo: compara y bifurca si el dato contenido en la localidad de memoria apuntada en el registro índice H:X es igual al acumulador.

```
CBEQ     X+,  etiqueta_1
```

- ***opr,SP,rel:*** Una simple combinación entre los operandos "opr,SP" y "rel". El resultado de esto es sencillo, una combinación entre el modo de direccionamiento indexado a través del stack pointer con desplazamiento de 8 bits y el relativo, como ejemplo: compara y bifurca si el dato contenido en la localidad de memoria apuntada en el stack pointer más el desplazamiento es igual al acumulador.

```
CBEQ     $05,SP,  etiqueta_1
```

1.7.2 OPERATION

OPERACIÓN: Esta columna se encarga de mostrar la función u operación de la instrucción correspondiente. Posteriormente se explicará la función de cada instrucción.

1.7.3 DESCRIPTION

DESCRIPCION: Esta columna se encarga de mostrar una descripción breve de que camino toma la CPU para llegar al resultado. Esta columna se lee por pequeños bloques separados por flechas cada bloque se lee de izquierda a derecha pero el orden de los bloques es de derecha a izquierda.

Ejemplo:

ADC #$40 $A \leftarrow (A) + (M) + (C)$

Explicación:

Se le suma al acumulador, el operando el cual es $40, mas el bit de carga y el resultado se coloca en el acumulador.
El primer bloque es: (A) + (M) + (C), el bloque se lee de izquierda a derecha
El segundo bloque es : (A), el orden en que se toman en cuenta los bloques es de derecha a izquierda.
En resumen se leería así: $A + M + C \rightarrow A$

1.7.4 EFFECT ON CCR

EFECTOS EN EL REGISTRO DE BANDERAS: Aquí se muestran los efectos que tendrán las banderas de estado del microcontrolador después de haber sido ejecutada la instrucción en cuestión.
Indica que la bandera no es afectada (-)
Indicativo de que la bandera es puesta a cero (0)
Indicativo de que la bandera es puesta a uno (1)
Indicativo de que la bandera puede ser puesta a uno o a cero dependiendo del resultado (↕)
Indica que el valor no puede ser determinado (U)

1.7.5 ADRESS MODE

MODO DE DIRECCIOAMIENTO: Aquí se indican los modos de direccionamiento en los que trabaja la instrucción en cuestión.

- IMM: Inmediato
- DIR: Directo
- EXT: Extendido
- IX2: Indexado con desplazamiento de 16 bits

- IX1: Indexado con desplazamiento de 8 bits
- IX: Indexado sin desplazamiento
- SP2: Indexado a través del stack pointer con desplazamiento de 16 bits
- SP1: Indexado a través del stack pointer con desplazamiento de 8 bits
- INH: Inherente
- DD: De directo a directo
- IMD: De inmediato a directo
- DIX+: De directo a indexado
- IX+D: De indexado a directo

1.7.6 OPCODE

CODIGO DE OPERACIÓN: Indica el código de operación de la instrucción en cuestión, cada instrucción tiene un código de operación diferente para cada uno de los modos de direccionamiento con los que sea compatible. Los códigos de operación pueden ocupar uno o 2 bytes de espacio.

1.7.7 OPERAND

OPERANDO: Indica la estructura del operando

- ii: Operando de 8 bits inmediato
- dd: Operando de 8 bits que indica una dirección de memoria directa
- hh ll: Operando de 16 bits que indica una dirección de memoria extendida
- ee ff: Desplazamiento de 16 bits en el direccionamiento indexado
- ff: Desplazamiento de 8 bits en el direccionamiento indexado
- rr: Byte de desplazamiento en el modo de direccionamiento relativo

1.7.8 CYCLES

CICLOS: Número de ciclos que la instrucción tarda en ser ejecutada por la CPU

Para obtener el número de bytes de espacio en memoria que utiliza la instrucción en cuestión solo hay que sumar el número de bytes del código de operación con el número de bytes del operando.

1.8 SET DE INSTRUCCIONES----------

En esta sección se explicará el funcionamiento de todas las instrucciones. Puesto que la forma de escribir un operando en especial es la misma en cada instrucción solo se explicará a fondo la forma de hacerlo una sola ves por eso la primera instrucción a exponer es la mas cargada de información ya que maneja 8 operandos diferentes pero al ser estos ya explicados no se repetirán en las siguientes instrucciones.

1.8.1 ADC *(suma con carry)*

Esta instrucción se encarga de realizar una simple suma aritmética en donde interfieren el acumulador, el operando y una unidad otorgada si el bit de carga o carry esta puesto en uno, el resultado se coloca en el acumulador.

Como puede verse en la figura esta instrucción según el resultado puede afectar todas las banderas de estados, es decir;

- Si la respuesta es mayor a FF el bit C se pondrá a 1
- Si la respuesta es 00 el bit Z se pondrá a 1
- Si la respuesta es un número negativo N será puesto a 1
- Si existe un medio acarreo H será puesto a 1
- Si existe desbordamiento en operaciones con signo V será puesto a 1

Si cada una de las condiciones anteriores no son ciertas las banderas se pondrán en cero

Source Form	Operation	Description	Effect on CCR						Address Mode	Opcode	Operand	Cycles
			V	H	I	N	Z	C				
ADC #*opr*	Add with Carry	A ← (A) + (M) + (C)	↕	↕	–	↕	↕	↕	IMM	A9	ii	2
ADC *opr*									DIR	B9	dd	3
ADC *opr*									EXT	C9	hh ll	4
ADC *opr*,X									IX2	D9	ee ff	4
ADC *opr*,X									IX1	E9	ff	3
ADC ,X									IX	F9		2
ADC *opr*,SP									SP1	9EE9	ff	4
ADC *opr*,SP									SP2	9ED9	ee ff	5

Figura 1.21: Instrucción ADC

ADC *#opr (inmediato)*

Esta instrucción en su modo de direccionamiento inmediato gasta 2 bytes de memoria, un byte para el código y otro para el dato inmediato, tiene un código de operación "A9" y una duración de 2 ciclos maquina.

Ejemplo: ¿Cuál es el valor Hexadecimal que adquiere el acumulador?

```
LDA     #$FF            ;C = ¿?   A = $FF
ADD     #$01            ;C = 1,   A = $00
ADC     #$20            ;C = 0,   A = $21
```

Explicación:

- Se carga en el acumulador el valor de $FF
- Se le suma al acumulador un 1 lo que provoca un rebosamiento que manda el bit de carga a 1 y el acumulador en $00
- Se hace una suma con carga. Se suma el valor del acumulador, el valor del dato inmediato y el bit de carga el cual solo tiene 2 posibles valores 1 y 0, el resultado se guarda en el acumulador y es $21.

(IMM) + (A) + (C) = (A)
($20) + ($00) + (1) = ($21)

ADC ***opr (directo)***

Esta instrucción en su modo de direccionamiento directo gasta 2 bytes de memoria, un byte para el código y otro para la localidad de memoria directa, tiene un código de operación "B9" y una duración de 3 ciclos maquina.

Ejemplo: ¿Cuál es el valor Hexadecimal que adquiere el acumulador? Suponiendo que hemos creado un registro llamado "REG_A" el cual se encuentra ubicado en la localidad de memoria directa $40.

```
LDA     #$01            ;C = ¿?,   A = $01
MOV     #$20, REG_A     ;C = ¿?,   REG_A = $20
SEC                     ;C = 1
ADC     REG_A           ;C = 0,    A = $22 .
```

También puede representarse de la siguiente manera sin la utilización de etiquetas.

```
LDA     #$01            ;C = ¿?,   A = $01
MOV     #$20,$40        ;C = ¿?,   REG_A = $20
SEC                     ;C = 1
ADC     $40             ;C = 0,    A = $22.
```

Explicación:

- El acumulador adquiere el valor de $01
- El registro adquiere el valor de $20
- El bit de carga se pone a 1

- Se hace una suma con carga. Se suma el valor del acumulador, el valor del dato almacenado en el registro y el bit de carga el cual solo tiene 2 posibles valores 1 y 0, el resultado se guarda en el acumulador y es $22.

(REG_A) + (A) + (C) = (A)
($20) + ($01) + (1) = ($22)

ADC *opr (extendido)*

Esta instrucción en su modo de direccionamiento extendido gasta 3 bytes de memoria, un byte para el código y dos para la localidad de memoria extendida, tiene un código de operación “C9” y una duración de 4 ciclos maquina.

Ejemplo: ¿Cuál es el valor Hexadecimal que adquiere el acumulador? Suponiendo que hemos creado un registro llamado “REG_B” el cual se encuentra ubicado en la localidad de memoria extendida $0100.

```
LDA      #$25          ;C = ¿?,   A = $25
STA      REG_B         ;C = ¿?,   REG_B = $25
SEC                    ;C = 1
ADC      REG_B         ;C = 0,    A = $4B .
```

Explicación:

- El acumulador adquiere el valor de $25
- El registro adquiere el valor de $25
- El bit de carga se pone a 1
- Se hace una suma con carga. Se suma el valor del acumulador, el valor del dato almacenado en el registro y el bit de carga el cual solo tiene 2 posibles valores 1 y 0, el resultado se guarda en el acumulador y es $4B.

(REG_B) + (A) + (C) = (A)
($25) + ($25) + (1) = ($4B)

ADC *opr,X (Indexado con desplazamiento de 16bits)*

Esta instrucción en su modo de direccionamiento indexado con desplazamiento de 16 bits gasta 3 bytes de memoria, un byte para el código y dos para el desplazamiento de 16 bits, tiene un código de operación “D9” y un duración de 4 ciclos maquina.

Ejemplo: ¿Cuál es el valor Hexadecimal que adquiere el acumulador? Suponiendo que hemos creado un registro llamado “REG_A” el cual se encuentra ubicado en la localidad de memoria extendida $0125.

```
LDA     #$20            ;C = ¿?,    A = $20
STA     $0125           ;REG_A = $20
LDHX    #$0025          ;H = 00,    X = 25
SEC                     ;C = 1
ADC     $0100,X         ;C = 0,     A = $41
```

Explicación:

- El acumulador adquiere el valor de $20
- El valor del acumulador es almacenado en el registro REG_A o localidad $0125
- El registro H:X adquiere el valor de $0025
- Se pone a uno el bit de carry
- Se hace una suma con carga. Se suma el valor del acumulador, el valor del bit de carga y el valor contenido en la dirección apuntada en el registro H:X mas el desplazamiento de 16 bits.

(Desplazamiento + X = $0240 = REG_A) + (A) + (C) = (A)
($20) + ($20) + (1) = ($41)

También puede representarse de la siguiente manera utilizando etiquetas.

```
LDA     #$20            ;C = ¿?,    A = $20
STA     REG_A           ;REG_A = $20
LDHX    #$0025          ;H = 00,    X = 25
SEC                     ;C = 1
ADC     $0100,X         ;C = 0,     A = $41
```

También es valido hacer lo siguiente:

```
LDA     #$20            ;C = ¿?,    A = $20
STA     REG_A           ;REG_A = $20
LDHX    #$0000          ;H = 00,    X = 00
SEC                     ;C = 1
ADC     REG_A,X         ;C = 0,     A = $41
```

En el siguiente caso el REG_A se encuentra en la localidad de memoria extendida $0225:

```
LDA     #$20            ;C = ¿?,    A = $20
STA     REG_A           ;REG_A = $20
LDHX    #$0125          ;H = 01,    X = 25
SEC                     ;C = 1
ADC     $0100,X         ;C = 0,     A = $41
```

Nótese que en este ultimo caso la dirección se obtiene de la conjunción de los registros H:X y el desplazamiento: H = $01, X = $25, Desplazamiento = $0100.

Por lo tanto: (H:X = $0125) + ($0100) = $0225; En esta localidad de memoria se encuentra almacenado el dato $20.

ADC *opr,X (Indexado con desplazamiento de 8 bits)*

Esta instrucción en su modo de direccionamiento indexado con desplazamiento de 8 bits gasta 2 bytes de memoria, un byte para el código y uno para el desplazamiento de 8 bits, tiene un código de operación “E9” y una duración de 3 ciclos maquina.

Ejemplo: ¿Cuál es el valor Hexadecimal que adquiere el acumulador? Suponiendo que hemos creado un registro llamado “REG_A” el cual se encuentra ubicado en la localidad de memoria directa $40.

```
LDA     #$20            ;C = ¿?,     A = $20
STA     $40             ;REG_A = $20
LDHX    #$0020          ;H = 00,     X = 20
SEC                     ;C = 1
ADC     $20,X           ;C = 0,      A = $41
```

Explicación:

- El acumulador adquiere el valor de $20
- El valor del acumulador es almacenado en el registro REG_A o localidad $40
- El registro H:X adquiere el valor de $0020
- Se pone a uno el bit de carry
- Se hace una suma con carga. Se suma el valor del acumulador, el valor del bit de carga y el valor contenido en la dirección apuntada en el registro H:X mas el desplazamiento de 8 bits.

(Desplazamiento + X = $0040 = REG_A) + (A) + (C) = (A)
($20) + ($20) + (1) = ($41)

Puede accederse con este tipo de direccionamiento a una localidad de memoria extendida tomando en cuenta que podemos modificar el valor de H pero si este permanece en $00 la localidad mas alta que podemos alcanzar corresponde al valor máximo del desplazamiento mas el valor máximo que puede alcanzar el registro X, esto es $FF + $FF = $01FE

ADC *opr,X (Indexado sin desplazamiento)*

Esta instrucción en su modo de direccionamiento indexado sin desplazamiento gasta 1 byte de memoria, que es usado para el código de operación “F9” con una duración de 3 ciclos maquina.

Ejemplo: ¿Cuál es el valor Hexadecimal que adquiere el acumulador? Suponiendo que hemos creado un registro llamado "REG_A" el cual se encuentra ubicado en la localidad de memoria directa $40.

```
LDA     #$20            ;C = ¿?,     A = $20
STA     $40             ;REG_A = $20
LDHX    #$0040          ;H = 00,     X = 40
SEC                     ;C = 1
ADC     0,X             ;C = 0,      A = $41
```

Explicación:

- El acumulador adquiere el valor de $20
- El valor del acumulador es almacenado en el registro REG_A o localidad $40
- El registro H:X adquiere el valor de $0040
- Se pone a uno el bit de carry
- Se hace una suma con carga. Se suma el valor del acumulador, el valor del bit de carga y el valor contenido en la dirección apuntada en el registro H:X.

$$(X = \$0040 = REG\_A) + (A) + (C) = (A)$$
$$(\$20) + (\$20) + (1) = (\$41)$$

ADC *opr,SP (Indexado a través del stack pointer y con desplazamiento de 16 bits)*

Esta instrucción en su modo de direccionamiento indexado a través del stack pointer con desplazamiento de 16 bits gasta 4 bytes de memoria, 2 bytes para el código y 2 para el desplazamiento de 16 bits, tiene un código de operación "9ED9" y una duración de 5 ciclos maquina.

Ejemplo: ¿Cuál es el valor Hexadecimal que adquiere el acumulador?

Si se ha reservado una pagina de la memoria ROM para crear una EEPROM que va de $8000 a $807F, suponiendo que hemos creado un registro llamado "REG_A" el cual se encuentra ubicado en la localidad de memoria extendida $8000 que a través de un programa especial hemos ejecutado desde la memoria RAM para almacenar en el REG_A el número $20.

```
LDA     #$20            ;C = ¿?,     A = $20
SEC                     ;C = 1
ADC     $7DC1,SP        ;C = 0,      A = $41
```

Explicación:

- El acumulador adquiere el valor de $20
- Se pone a uno el bit de carry
- Se hace una suma con carga. Se suma el valor del acumulador, el valor del bit de carga y el valor contenido en la dirección apuntada en el stack pointer más el desplazamiento de 16 bits.
- El valor del stack pointer que corresponde al MC68HC908GP32 cuando se esta en el programa principal y no en alguna subrutina es el número de localidad a la que pertenece el fin de la memoria RAM, este es $023F

(Desplazamiento + SP = $7DC1 + $023F) + (A) + (C) = (A)
(El valor contenido en la localidad $8000) + ($20) + (1) = ($41)

ADC *opr,SP (Indexado a través del stack pointer y con desplazamiento de 16 bits)*

Esta instrucción en su modo de direccionamiento indexado a través del stack pointer con desplazamiento de 8 bits gasta 3 bytes de memoria, 2 bytes para el código y 1 para el desplazamiento de 8 bits, tiene un código de operación “9EE9” y una duración de 4 ciclos maquina.

Solo suponga que el valor contenido la localidad de memoria es de $20 y el valor en el stack es el fin de la RAM

```
LDA     #$20            ;C = ¿?,    A = $20
SEC                     ;C = 1
ADC     $00,SP          ;C = 0,     A = $41
```

(Desplazamiento + SP = $00 + $023F) + (A) + (C) = (A)
($20) + ($20) + (1) = ($41)

UN EJEMPLO PRÁCTICO:

Se desea detener el microcontrolador cuando el llamado de una subrutina se llegue a cumplir 65,025 veces o en hexadecimal $FE01, utilizando el REG_A (localidad $40), el REG_B (localidad $41) y por supuesto la instrucción ADC. El valor inicial de los registros A y B es de $00

```
Subrutina_1:
            LDA     REG_B           ; Carga en el acumulador el registro B
            ADD     #$01            ; Suma al acumulador un 1
            STA     REG_B           ; Almacena el acumulador en el registro B
            CLRA                    ; Limpia el acumulador
            LDX     REG_A           ; Carga en X el registro A
            CBEQX   #$FF, etiqueta_1 ; Bifurca a etiqueta_1 si X es igual a $FF
            ADC     REG_A           ; Suma al registro A, al acumulador y C
            STA     REG_A           ; Almacena el acumulador en  registro A
            RTS                     ; Retorna de subrutina
etiqueta_1: STOP                    ; Detén al microcontrolador
```

Solución:

Para llegar al número $FE01 que es un número de 16 bits con un microcontrolador de 8 bits es necesario contar asta $FF y repetir esta cuenta $FF veces por eso, le sumaremos un 1 al registro B cada ves que la subrutina es llamada. Después de ser llamada $FF veces ocurrirá un desbordamiento que pondrá a uno el bit de carry. Al registro A se le será sumado cada ves que sea llamada esta subrutina el bit de carga que casi siempre será un 0 a menos que un desbordamiento ocurra esto provocará que se le sume un 1. Cuando el registro A llegue a $FF el microcontrolador será detenido con la instrucción STOP

Explicación:

- El acumulador adquiere el valor que posee el registro A, que al iniciarse la ejecución del programa este es 0.
- Se le suma 1 al acumulador, esto sucederá cada ves que sea llamada esta subrutina, pero por que utilizamos ADD que gasta 2 bytes de memoria uno para el código de operación y otro para el dato a sumar que en este caso es un 1, en ves de utilizar simplemente la instrucción INCA que incrementa en 1 al acumulador. Esto es por que necesitamos que ocurra un desbordamiento en el acumulador para que el bit de carga se ponga a 1, suponiendo que cuando el registro B llegue al valor de $FF y este sea llevado al acumulador ocurrirá un desbordamiento con la instrucción ADD #$01 que pondrá a 1 el bit de carga y el acumulador quedará en $00 esto no ocurre cuando el acumulador se desborda con la instrucción INCA ya que el bit de carga permanece tal cual se encontrara antes deque ocurriese el desbordamiento.
- El acumulador es almacenado en el registro B
- Se pone a $00 el acumulador
- Si el registro A es $FF se dirige el programa hacia la instrucción STOP.
- Suma el acumulador (el cual siempre es cero en este punto) con el registro A y el carry
- Retorna al programa principal.

1.8.2 ADD *(suma sin carry)*

Esta instrucción se encarga de realizar una simple suma aritmética en donde interfieren el acumulador y el operando, en este caso el bit de carga no es tomado en cuenta, el resultado se coloca en el acumulador.
Puesto que esta instrucción es similar a ADC puede utilizar los ejemplos de esta para una mejor comprensión, solo recuerde que el bit C no interfiere en ADD.

ADD #*opr* ADD *opr* ADD *opr* ADD *opr*,X ADD *opr*,X ADD ,X ADD *opr*,SP ADD *opr*,SP	Add without Carry	A ← (A) + (M)	↕	↕	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	AB BB CB DB EB FB 9EEB 9EDB	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

Figura 1.22: Instrucción ADC

1.8.3 AIS *(suma al stack el valor inmediato signado)*

Esta instrucción le suma al stack pointer el valor inmediato signado y el resultado lo guarda en el stack pointer.
Cuando se habla de un valor signado se hace la referencia de que el bit 7 se considera el signo de la palabra es decir el dato ya no es mas una palabra de 8 bits si no una de 7 bits con signo. Si el bit 7 es 0 será un número positivo, si es 1 será negativo. Puesto que el signo es el bit mas significativo con un número signado de 8 bits pueden escribirse el siguiente rango de números:

Hexadecimal	Decimal
00	00
01	01
↓	↓
7E	126
7F	127
80	-128
81	-127
↓	↓
FE	-02
FF	-01

Figura 1.23: Tabla de conversión de un número signado

El número signado de $00 a $7F no difiere en nada al número no signado pero a partir de $80 se debe obtener el complemento a 2 para saber su valor.
Para una mayor comprensión puede consultar los ejemplos mostrados en la instrucción AIX.

AIS #*opr*	Add Immediate Value (Signed) to SP	SP ← (SP) + (16 « M)	–	–	–	–	–	–	IMM	A7	ii	2

Figura 2.24: Instrucción AIS

1.8.4 AIX *(suma a H:X el valor inmediato signado)*

Esta instrucción le suma al registro de índice H:X el valor inmediato signado y el resultado lo guarda en el registro H:X.

AIX #*opr*	Add Immediate Value (Signed) to H:X	H:X ← (H:X) + (16 « M)	–	–	–	–	–	–	IMM	AF	ii	2

Figura 1.25: Instruccion AIX.

Ejemplo:

¿Cuál es el valor Hexadecimal que adquiere el Registro H:X si su valor inicial es $0240?

Solución:

Con números signados positivos:

```
AIX     #$00    ;H:X es igual a $0240
AIX     #$50    ;H:X es igual a $0290
AIX     #$65    ;H:X es igual a $02A5
AIX     #$7F    ;H:X es igual a $0367
```

Solución:

Con números signados negativos:

```
AIX     #$80    ;H:X es igual a $01C0
AIX     #$B5    ;H:X es igual a $01F5
AIX     #$DF    ;H:X es igual a $021F
AIX     #$FF    ;H:X es igual a $023F
```

Si se pregunta por que no resulta su resta. Solo tiene que obtener el complemento a 2 del operando por ejemplo $B5 con lo cual obtendrá $4B que es lo que en realidad resta la CPU al registro de índice H:X.

1.8.5 AND *(operación lógica AND)*

Esta instrucción realiza una operación lógica AND entre el acumulador y el operando, el resultado es almacenado en el acumulador.

AND #*opr* AND *opr* AND *opr* AND *opr*,X AND *opr*,X AND ,X AND *opr*,SP AND *opr*,SP	Logical AND	A ← (A) & (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A4 B4 C4 D4 E4 F4 9EE4 9ED4	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

Figura 1.26: Instruccion AND.

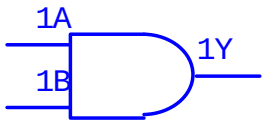
Ejemplo:

¿Qué debe hacerse para eliminar los 4 bits menos significativos del acumulador sin afectar los bits más significativos?

Solución:

Si el acumulador es $55 la respuesta es: AND #$F0 ;A = $50

Recordemos la tabla de verdad de la operación lógica AND:



A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

Figura 1.27: Tabla de verdad de la operación logica AND.

Si el acumulador es igual a: $55 = 0101 0101
Para eliminar los bits menos significativos: $F0 = 1111 0000
Después de la operación AND nos queda: $50 = 0101 0000

Ejemplo:
¿Qué debe hacerse para eliminar los 4 bits más significativos del acumulador sin afectar los bits menos significativos?

Solución:
Si el acumulador es $55 la respuesta es: AND #$0F ;A = $05

1.8.6 ASL, ASLA, ASLX *(deslizamiento aritmético a la izquierda)*

Esta instrucción realiza un deslizamiento aritmético a la izquierda al acumulador, al registro X o a algún tipo de operando, como puede verse en la figura un cero es colocado en el bit menos significativo, el contenido del bit 0 pasa a ser el contenido del bit 1, el contenido del bit 1 pasa a ser el contenido del bit 2, y así asta llegar al contenido del bit 7 que pasa a ser el contenido del bit de carga, el contenido de la carga se borra. Esta instrucción es idéntica a la instrucción LSL.
ASLA se utiliza en el caso de querer aplicar esta instrucción al acumulador.
ASLX se utiliza en el caso de querer aplicar esta instrucción al registro X.
ASL se utiliza en el caso de querer aplicar esta instrucción a algún tipo de operando.

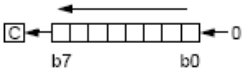
ASL *opr*			↕	–	–	↕	↕	↕	DIR	38	dd	4
ASLA									INH	48		1
ASLX	Arithmetic Shift Left								INH	58		1
ASL *opr*,X	(Same as LSL)								IX1	68	ff	4
ASL ,X									IX	78		3
ASL *opr*,SP									SP1	9E68	ff	5

Figura 1.28: Instrucción ASL.

Ejemplo:

Se desea realizar un deslizamiento aritmético a la izquierda en el acumulador el cual contiene el dato $55

Solución:

Si el acumulador es $55 la respuesta es: ASLA ; A = AA

	C	ACUMULADOR
	¿?	0 1 0 1 0 1 0 1 = $55
Después de ASLA	0	1 0 1 0 1 0 1 0 = $AA

Ejemplo:

Se desea realizar un deslizamiento aritmético a la izquierda al registro A el cual se encuentra en una localidad de memoria directa, el dato contenido en el registro es $8F.

Solución:
Si el registro A es $8F la respuesta es: ASL REG_A ; A = 1E

	C	ACUMULADOR
	¿?	1 0 0 0 1 1 1 1 = $8F
Después de ASL REG_A	1	0 0 0 1 1 1 1 0 = $1E

1.8.7 ASR, ASRA, ASRX *(deslizamiento aritmético a la derecha)*

Esta instrucción realiza un deslizamiento aritmético a la derecha al acumulador, al registro X o a algún tipo de operando, como puede verse en la figura el bit 7 no cambia pero su contenido pasa a ser el contenido de bit 6, el contenido del bit 6 pasa a ser el contenido del bit 5 y así asta llegar al bit 0 cuyo contenido pasa a ser el contenido del bit de carga y por ultimo el contenido de la carga se borra.
ASRA se utiliza en el caso de querer aplicar esta instrucción al acumulador.
ASRX se utiliza en el caso de querer aplicar esta instrucción al registro X.
ASR se utiliza en el caso de querer aplicar esta instrucción a algún tipo de operando.

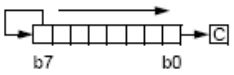
ASR *opr* ASRA ASRX ASR *opr*,X ASR *opr*,X ASR *opr*,SP	Arithmetic Shift Right		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	37 47 57 67 77 9E67	dd ff ff	4 1 1 4 3 5

Figura 1.29: Instrucción ASR.

Ejemplo:

Se desea realizar un deslizamiento aritmético a la derecha en el acumulador el cual contiene el dato $55

Solución:

Si el acumulador es $55 la respuesta es: ASRA ; A = $2A

		ACUMULADOR	C
	$55 =	0 1 0 1 0 1 0 1	¿?
Después de ASRA	$2A =	0 0 1 0 1 0 1 0	1

Ejemplo:

Se desea realizar un deslizamiento aritmético a la derecha al registro A el cual se encuentra en una localidad de memoria directa, el dato contenido en el registro es $8E.

Solución:

Si el registro A es $8E la respuesta es: ASR REG_A ; A = C7

		ACUMULADOR	C
	$8E =	1 0 0 0 1 1 1 0	¿?
Después de ASR REG_A	$C7 =	1 1 0 0 0 1 1 1	0

1.8.8 BCC *(bifurca si el bit de carry esta puesto a 0)*

Esta instrucción realiza una bifurcación asía la dirección indicada por la etiqueta colocada por el programador siempre y cuando el bit de carga este puesto a 0. Recuerde que este es modo de direccionamiento relativo por lo cual solo se puede desplazar el contador de programa de 127 a -128 localidades de memoria.

BCC *rel*	Branch if Carry Bit Clear	PC ← (PC) + 2 + rel ? (C) = 0	–	–	–	–	–	–	REL	24	rr	3

Figura 1.30: Instrucción BCC.

Ejemplo:

```
Subrutina:
            BCC     ET_01
            STOP
ET_01:      RTS
```

Explicación:

Cuando el contador de programa llega a la subrutina la primera instrucción a ejecutar es BCC, si la condición se cumple es decir si el bit de carga esta puesto a 0 el programa bifurcará asía la etiqueta 01, en este caso el contador de programa regresará al programa principal a través de la instrucción RTS. En caso de que el bit de carga este puesto a 1 el programa no bifurcará por lo cual se ejecutará la instrucción siguiente a BCC en este caso STOP por lo que se detendrá el microcontrolador.

1.8.9 BCLR, BSET *(pon a 0 o a 1 el bit n en M)*

Estas instrucciones se encargan de poner a 0 o a 1 respectivamente un solo bit de cualquier localidad de memoria directa. Puesto que los registros de datos de los puertos se encuentran en localidades de memoria directa puede manipularse individualmente el estado de algún pin del microcontrolador para el control de procesos en tiempo real.
BCLR se utiliza para poner a 0 el bit n (cualquier bit) de la localidad de memoria directa.
BSET se utiliza para poner a 1 el bit n (cualquier bit) de la localidad de memoria directa.

BCLR *n, opr*	Clear Bit n in M	Mn ← 0	–	–	–	–	–	–	DIR (b0)	11	dd	4
									DIR (b1)	13	dd	4
									DIR (b2)	15	dd	4
									DIR (b3)	17	dd	4
									DIR (b4)	19	dd	4
									DIR (b5)	1B	dd	4
									DIR (b6)	1D	dd	4
									DIR (b7)	1F	dd	4

Figura 1.31: Instrucción BCLR.

BSET *n,opr*	Set Bit *n* in M	Mn ← 1	–	–	–	–	–	–	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	10 12 14 16 18 1A 1C 1E	dd dd dd dd dd dd dd dd	4 4 4 4 4 4 4 4

Figura 1.32: Instrucción SET.

Ejemplo:

Un LED se encuentra conectado en el puerto PTA 0. Según la figura, si el LED se encuentra apagado deduzca en que estado se encuentra el bit 0 del puerto A y que instrucción es necesaria para encenderlo:

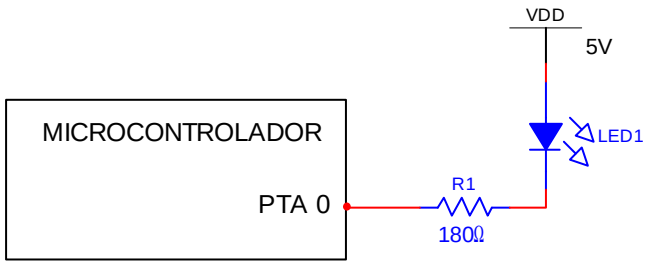


Figura 1.33: LED configurado en lógica negativa

Explicación:

Como sabemos en la lógica tradicional (Lógica positiva) las cosas se activan con un 1 lógico o nivel de voltaje alto y se apagan con un 0 o nivel de voltaje bajo, en la lógica negativa esto sucede al revés, las cosas se activan con un cero como el caso de la figura.

El microcontrolador puede suministrar en la mayoría de sus puertos un máximo de corriente de 10 mA en modo fuente o en modo sumidero es decir estando en 1 lógico o en 0 lógico.

Para que el LED se encuentre apagado es necesario que el puerto este en 1 lógico esto se debe a que los pines del microcontrolador en 1 lógico tienen un nivel de voltaje casi igual a VDD en este caso 5V por lo que no existe diferencia de potencial entre los extremos del LED.

Si el voltaje de polarización es: VDD = 5V
Entonces el voltaje del pin en 1 lógico será: Vpin ≈ 5V
La diferencia de los voltajes en las terminales del LED es: VDD – Vpin ≈ 0V

Al no existir una diferencia de potencial mayor al voltaje del LED la corriente no fluirá y por lo tanto el LED no prenderá.
Conclusión: El LED se encuentra apagado

Solución:

Por lo tanto si el LED se encuentra apagado se puede deducir que el pin 0 del puerto A se encuentra en un nivel lógico alto. Para que el LED prenda es necesario llevarlo a un nivel lógico bajo puesto que así el voltaje del pin será casi igual a 0V.

Si el voltaje de polarización es: VDD = 5V
Entonces el voltaje del pin en 0 lógico será: Vpin ≈ 0V
La diferencia de los voltajes en las terminales del LED es: VDD – Vpin ≈ 5V

Al existir una diferencia de potencial mayor al voltaje del LED la corriente fluirá y por lo tanto el LED prenderá. Por lo cual para que el LED prenda debe utilizarse la instrucción:

```
BCLR    0,PTA
```

DATOS A TOMAR ENCUENTA POR EL DISEÑADOR EXPERTO

Se desea prender un LED rojo conectado al microcontrolador en lógica positiva. ¿De que valor deberá ser la resistencia para que la corriente por el LED sea igual a 10mA?

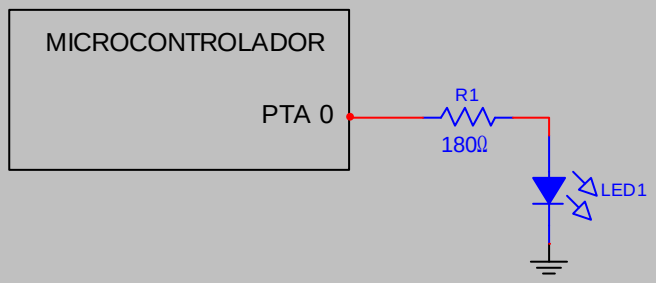


Figura 1.34: LED configurado en lógica positiva

Solución: El fabricante menciona que si un puerto estando en 1 lógico (5V) se le coloca una carga de10 mA el voltaje se caerá en 1.5V y si esta puesto a 0 lógico (0V) el voltaje aumentará en 1.5V. Estas características son para el funcionamiento a 5V ya que en su funcionamiento a 3V estas difieren. También debe tomarse en cuenta el voltaje del LED en el caso de uno rojo este es de 1.7V, en el caso de uno verde este es de 2.1V.
Aplicando kirchhoff de voltajes para la malla del LED:

$$\Sigma = V_{pin} + V_{R1} + V_{LED} = 0$$

Donde: Vled = 1.7V;
Vpin = 5V–1.5V = 3.5V; caída de 1.5V debido a la carga de 10mA
$V_{R1} = I/R$

Despejando R:

$$R = \frac{5V - 1.5V - 1.7V}{10mA} = 180\Omega$$

CURIOSIDADES

Por que el estudiante funde más rápido un LED rojo que uno verde:

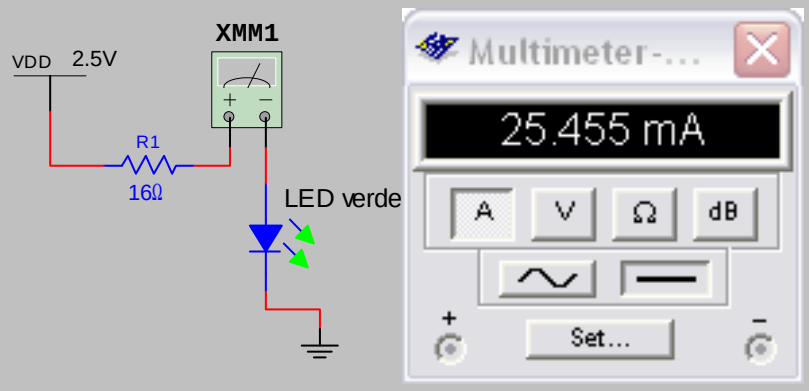


Figura 1.35: Corriente máxima del LED.

Supóngase un circuito simple como el de la figura 1.35 con un LED verde con V_{LED}=2.1V polarizado con una fuente de 2.5V. ¿Qué valor de resistencia sería necesario para que el LED operara a su corriente máxima de 25 mA?

$$R = \frac{2.5V - 2.1V}{25mA} = 16\Omega$$

Ahora supóngase que en el circuito de la figura 1.35 es reemplazado el LED verde por un LED rojo con un V_{LED} = 1.7V y corriente máxima de 25 mA. ¿Cuál seria la corriente que circula por el LED rojo?

$$I = \frac{2.5V - 1.7V}{16\Omega} = 50\ mA$$

Las consecuencias serian que la corriente se duplicaría y sobrepasaría por el doble la corriente máxima del LED acortando sin duda la vida útil de este. Por lo tanto en un circuito idéntico en voltaje y resistencia un LED rojo permitirá un mayor paso de corriente debido a su menor caída de voltaje que un LED verde.

NOTAS:

Cuando utilice LED´s en circuitos polarizados con fuentes pequeñas de entre 2.5V y 5.5V (rango de operación de la mayoría de los microcontroladores) procure calcular la resistencia para el LED que en cuyo caso valla a utilizar. Mientras mas aumente el voltaje de polarización (9V, 12V, 24V etc.) más despreciable será este concepto (Intercambio de LED´s) debido al incremento de la resistencia limitadora del LED.

1.8.10 BCS *(bifurca si el bit de carry esta puesto a 1)*

Esta instrucción se encarga de bifurcar a la dirección indicada por la etiqueta apuntada por el programador siempre y cuando el bit de carga este puesto a 1. Modo de direccionamiento Relativo, recuerde el desplazamiento máximo de 127 a -128 localidades de memoria. Esta instrucción tiente el mismo efecto que la instrucción BLO.

BCS *rel*	Branch if Carry Bit Set (Same as BLO)	PC ← (PC) + 2 + *rel* ? (C) = 1	–	–	–	–	–	–	REL	25	rr	3

Figura 1.36: Instrucción BCS.

1.8.11 CMP, *(compara al acumulador con el operando)*

Esta instrucción se encarga de comparar al acumulador con algún tipo de operando, la acción de esta instrucción es modificar las banderas de estados pertinentes según el resultado de la comparación. La comparación no es mas que una resta entre el acumulador y el operando, el resultado de esta resta no es almacenado en ningún sitio pero modifica las banderas de estado.

CMP #*opr*	Compare A with M	(A) – (M)	↕	–	–	↕	↕	↕	IMM	A1	ii	2
CMP *opr*									DIR	B1	dd	3
CMP *opr*									EXT	C1	hh ll	4
CMP *opr*,X									IX2	D1	ee ff	4
CMP *opr*,X									IX1	E1	ff	3
CMP ,X									IX	F1		2
CMP *opr*,SP									SP1	9EE1	ff	4
CMP *opr*,SP									SP2	9ED1	ee ff	5

Figura 1.37: Instrucción CMP.

Ejemplo:

Si el acumulador es mayor al operando: En este caso ninguna bandera es afectada puesto que la CPU hace una resta de la siguiente manera: A – M = $55 – $4B = $0A. Esto permite que instrucciones como BHI (bifurca si el acumulador es mayor que el operando) realice su función de bifurcación.

```
LDA     #$55
CMP     #$4B
```

Si el acumulador es igual al operando: En este caso la bandera afectada es Z puesto que la CPU hace una resta de la siguiente manera: A – M = $55 – $55 = $00. Debido a este resultado ($00) la bandera Z se pone a 1. Esto permite que instrucciones como BEQ (bifurca si el acumulador es igual al operando) realice su función de bifurcación.

```
LDA     #$55
CMP     #$55
```

Si el acumulador es menor al operando: En este caso las banderas afectadas son C y N puesto que la CPU hace una resta de la siguiente manera A – M = $55 – $65 = $F0, esto es lo que arroja la CPU pero sacando su complemento a 2 la respuesta es $10 negativo. Como los bit´s C y N se ponen a uno indican que el número es negativo lo que quiere decir que el operando es mayor que el acumulador.
Si la respuesta de la resta va de $01 negativo a $80 negativo ($FF a $80 lo que en realidad aparece en el acumulador en caso de una resta con resultado negativo) los bit´s C y N se ponen a 1. Si la respuesta de la resta va de $81 negativo a $FF negativo ($7F a $01 lo que en realidad aparece en el acumulador en caso de una resta con resultado negativo) solo el bit C se pone a 1. En cual quiera de los 2 casos instrucciones como BLO (bifurca si es menor) ejecutan la bifurcación ya que en este tipo de instrucciones el bit C puesto a 1, significa que hubo un pedido de préstamo en la resta por lo que da un resultado negativo.

```
        LDA     #$55
        CMP     #$65
```

1.8.12 BEQ *(bifurca si es igual)*

Esta instrucción se encarga de bifurcar a la dirección indicada en la etiqueta apuntada por el programador siempre y cuando se hubiese ejecutado una comparación anterior a la instrucción BEQ y el resultado de la comparación hubiese sido que el acumulador y el operando sean iguales o que es lo mismo que el bit Z sea puesto a 1.

BEQ *rel*	Branch if Equal	PC ← (PC) + 2 + *rel* ? (Z) = 1	–	–	–	–	–	–	REL	27	rr	3

Figura 1.38: Instrucción BEQ.

Ejemplo:

Se hace una resta entre el acumulador y el registro A el cual se encuentra en una localidad de memoria directa de la siguiente forma: A – M = $55 – $¿? = $¿?.

```
        LDA     REG_A
        CMP     #$55
        BEQ     ET_01
        STOP
ET_01:  RTS
```

Explicación:
Si el registro A contiene el dato $55 la resta dará como resultado $00 lo cual pondrá a 1 el bit Z lo que indica que la condición de: “bifurca si el acumulador es igual al operando” sea verdadera, por lo que el programa bifurcará hacia la instrucción RTS. Con cualquier otro valor en el registro A la respuesta de la resta no será $00 por lo que no habrá bifurcación y el microcontrolador se detendrá con la instrucción STOP.

1.8.13 BGE *(bifurca si es mayor o igual [con operandos signados])*

Esta instrucción se encarga de bifurcar a la dirección indicada en la etiqueta apuntada por el programador siempre y cuando se hubiese ejecutado una comparación anterior a la instrucción BGE y el resultado de la comparación hubiese sido que el acumulador es mayor o igual que el operando solo que en este caso el operando y el acumulador se toman como números signados tal y como ya se ha visto en la tabla 1.23. Para que esto se ejecute la CPU se vale de las banderas N y V aplicando una OR exclusiva entre ellas y si el resultado de esta operación es 0 quiere decir que alguna de las 2 condiciones que demanda esta instrucción para su ejecución se esta cumpliendo.

BGE *opr*	Branch if Greater Than or Equal To (Signed Operands)	PC ← (PC) + 2 + *rel* ? (N ⊕ V) = 0	–	–	–	–	–	–	REL	90	rr	3

Figura 1.39: Instrucción BGE.

Ejemplo:

Supóngase que al entrar a la subrutina el acumulador es igual a $7F.

```
Subrutina:  CMP     #$80
            BGE     ET_01
            STOP
Et_01:      RTS
```

Explicación:

Esta instrucción bifurcará siempre y cuando el acumulador sea igual o mayor al operando a simple vista se podría decir que no bifurcará debido a que el acumulador es menor que el operando, pero recordemos que esta instrucción toma en cuenta al acumulador y al operando como números signados es decir que según la tabla 1.23 $7F es 127 y $80 es -128 por lo tanto el acumulador si es mayor al operando.

Supóngase que al entrar a la subrutina el acumulador es igual a $80.
El programa si bifurcara por que el acumulador y el operando son iguales.

Supóngase que al entrar a la subrutina el acumulador es igual a $81.
El programa si bifurcara ya que $81 es igual a -127 mientras que $80 es igual -128, por lo tanto -127 es mayor a -128, el acumulador es mayor

Supóngase que el acumulador es $80 y el operando es $85.
El programa no bifurcara ya que $80 es igual a -128 mientras que $85 es igual -123, por lo tanto -128 es menor a -123, el acumulador es menor

1.8.14 BGT *(bifurca si es mayor [con operandos signados])*

Esta instrucción se encarga de bifurcar si el acumulador es mayor que el operando pero con la particularidad de que el acumulador y el operando son tomados como números signados tal y como se muestra en la tabla 1.23.
Las banderas que interfieren en la ejecución de esta instrucción son Z, N y V; primero se hace una OR exclusiva entre N y V, posteriormente al resultado de esto se le aplica una OR con el bit Z y si esto da como resultado 0 se ejecuta la bifurcación de lo contrario no.

BGT *opr*	Branch if Greater Than (Signed Operands)	PC ← (PC) + 2 + *rel* ? (Z) I (N ⊕ V) = 0	–	–	–	–	–	–	REL	92	rr	3

Figura 1.40: Instrucción BGT.

Ejemplo:

Supóngase que al entrar a la subrutina el acumulador es igual a $7F.

```
Subrutina:  CMP     #$80
            BGT     ET_01
            STOP
Et_01:      RTS
```

Explicación:

El programa bifurcara debido a que en números signados $7F es mayor a $80

Supóngase que al entrar a la subrutina el acumulador es igual a $80.
El programa no bifurcara por que el acumulador y el operando son iguales.

Supóngase que al entrar a la subrutina el acumulador es igual a $81.
El programa si bifurcara ya que $81 es igual a -127 mientras que $80 es igual -128, por lo tanto -127 es mayor a -128, el acumulador es mayor.

1.8.15 BHCC *(bifurca si el bit de medio acarreo esta puesto a 0)*

Sencilla instrucción en la que el programa bifurcara a la etiqueta indicada siempre y cuando la bandera H este puesta a 0.

BHCC *rel*	Branch if Half Carry Bit Clear	PC ← (PC) + 2 + *rel* ? (H) = 0	–	–	–	–	–	–	REL	28	rr	3

Figura 1.41: Instrucción BHCC.

1.8.16 BHCS *(bifurca si el bit de medio acarreo esta puesto a 1)*

Sencilla instrucción en la que el programa bifurcara a la etiqueta indicada siempre y cuando la bandera H este puesta a 1.

BHCS *rel*	Branch if Half Carry Bit Set	PC ← (PC) + 2 + *rel* ? (H) = 1	–	–	–	–	–	–	REL	29	rr	3

Figura 1.42: Instrucción BHCS.

1.8.17 BHI *(bifurca si es mayor)*

Instrucción en la que se bifurca si después de una comparación resulta que el acumulador es mayor al operando. Para ejecutar esta instrucción la CPU se vale de las banderas C y Z, entre las cuales se ejecuta una operación lógica OR, si el resultado de esto es 0 significa que el acumulador es mayor.

BHI *rel*	Branch if Higher	PC ← (PC) + 2 + *rel* ? (C) I (Z) = 0	–	–	–	–	–	–	REL	22	rr	3

Figura 1.43: Instrucción BHI.

1.8.18 BHS *(bifurca si es mayor o igual)*

Instrucción en la que se bifurca después de una comparación siempre y cuando el acumulador sea igual o mayor al operando. Para ejecutar esta instrucción el acumulador se vale del bit de carga, si este es 0 se ejecuta la bifurcación. Esta instrucción es igual a utilizar BCC.

BHS *rel*	Branch if Higher or Same (Same as BCC)	PC ← (PC) + 2 + *rel* ? (C) = 0	–	–	–	–	–	–	REL	24	rr	3

Figura 1.44: Instrucción BHS

1.8.19 BIH *(bifurca si el pin –IRQ esta puesto en 1)*

Si en el momento en que se ejecuta esta instrucción el pin –IRQ esta puesto a 1 el programa bifurcará.
El pin –IRQ trae por default una resistencia de pull up es decir que siempre esta puesto a un nivel de voltaje alto, para que el pin cambie de estado es necesario hacerlo a través de Hardware tal como puede verse en la figura 1.46.

BIH *rel*	Branch if $\overline{\text{IRQ}}$ Pin High	PC ← (PC) + 2 + *rel* ? $\overline{\text{IRQ}}$ = 1	–	–	–	–	–	–	REL	2F	rr	3

Figura 1.45: Instrucción BIH.

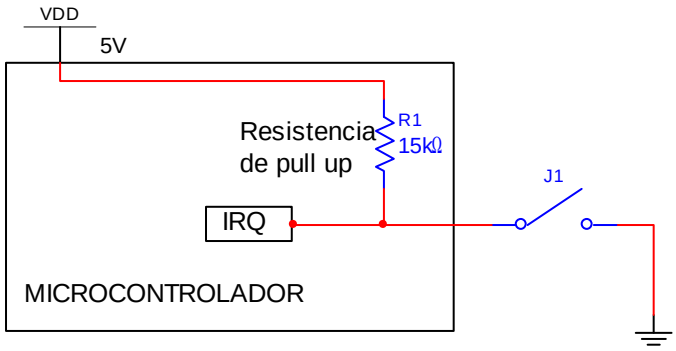


Figura 1.46: Manda a 0 al pin IRQ.

1.8.20 BIL *(bifurca si el pin –IRQ esta puesto a 0)*

Si en el momento en que se ejecuta esta instrucción el pin –IRQ esta puesto a 0 el programa bifurcará.

BIL *rel*	Branch if $\overline{\text{IRQ}}$ Pin Low	PC ← (PC) + 2 + *rel* ? $\overline{\text{IRQ}}$ = 0	–	–	–	–	–	–	REL	2E	rr	3

Figura 1.47: Instrucción BIH.

1.8.21 BIT *(prueba de bit)*

Esta instrucción es la que se utiliza para preguntar por el estado de algún bit en específico de cualquier localidad del mapa de memoria, en si misma no es mas que una simple operación lógica AND.

Esta instrucción pone siempre a 0 la bandera V y modifica las banderas N y Z según sea el caso, nosotros pondremos especial atención a la bandera Z. La instrucción por si sola no prueba ningún bit, existen diferentes formas de lograr esto, aquí se recomienda cargar en el acumulador el dato especial correspondiente para el bit determinado a probar, consulte la figura 1.49, después se coloca la instrucción BIT junto con el operando a analizar según los modos de direccionamientos con los que es compatible esta instrucción.

Si la bandera Z se pone a 0 significara que el bit probado esta puesto a 1, y si la bandera Z se pone a 1 significa que el bit probado esta puesto a 0.

BIT #*opr* BIT *opr* BIT *opr* BIT *opr*,X BIT *opr*,X BIT ,X BIT *opr*,SP BIT *opr*,SP	Bit Test	(A) & (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A5 B5 C5 D5 E5 F5 9EE5 9ED5	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

Figura 1.48: Instrucción BIT.

Ejemplo:

Podemos valernos de la instrucción BNE que provoca una bifurcación cuando el bit Z es igual a 0 o mejor dicho "bifurca si no es igual". Supóngase que al entrar a la subrutina el puerto A es igual a $F5 (PTA = F5).

```
Subrutina:  LDA     #$01        ;se analiza el bit 0
            BIT     PTA         ;del puerto A (0,PTA)
            BNE     ET_01       ;bifurca si Z = 0 a consecuencia de que el bit
                                ;testado es 1
            STOP
Et_01:      RTS
```

Explicación:

A = $01 = 0 0 0 0 0 0 0 1
PTA = $F5 = 1 1 1 1 0 1 0 1
Después de BIT PTA = $01 = 0 0 0 0 0 0 0 1
Por lo tanto Z = 0

Como el resultado de la operación es 1 el bit Z se mantiene en 0 y por lo tanto el programa bifurca.

Ejemplo:

Supóngase que el puerto A es igual a $F4 (PTA = F4).

```
Subrutina:  LDA     #$01        ;se analiza el bit 0
            BIT     PTA         ;del puerto A (0,PTA)
            BNE     ET_01       ;bifurca si Z = 0 a consecuencia de que el bit
                                ;testado es 1
            STOP
Et_01:      RTS
```

Explicación:

A = $01 = 0 0 0 0 0 0 0 1
PTA = $F4 = 1 1 1 1 0 1 0 0
Después de BIT PTA = $01 = 0 0 0 0 0 0 0 0
Por lo tanto Z = 1

Como el resultado de la operación es 0 el bit Z cambia a 1 y por lo tanto el programa no bifurca y se detiene debido a la instrucción STOP.

BIT	DATO
0	$01
1	$02
2	$04
3	$08
4	$10
5	$20
6	$40
7	$80

Figura 1.49: Códigos para probar Bit´s.

1.8.22 BLE *(bifurca si es menor o igual [con operandos signados])*

Esta instrucción se encarga de bifurcar a la dirección indicada en la etiqueta apuntada por el programador siempre y cuando se hubiese ejecutado una comparación anterior a la instrucción BLE y el resultado de la comparación hubiese sido que el acumulador es menor o igual que el operando solo que en este caso el operando y el acumulador se toman como números signados tal y como ya se ha visto en la tabla 1.23. Para ejecutar esta instrucción la CPU se vale de los bit´s Z, N y V; la CPU ejecuta una OR exlusiva entre el bit N y V, y el resultado de esto es afectado por una opeeración OR con el bit Z, si todo esto es igual a 1 la bifurcación se ejecuta.

Para un mayor entendimiento de esta instrucción consulte la instrucción BGE que se ha visto en páginas anteriores.

BLE *opr*	Branch if Less Than or Equal To (Signed Operands)	PC ← (PC) + 2 + *rel* ? (Z) I (N ⊕ V) = 1	–	–	–	–	–	–	REL	93	rr	3

Figura 1.50: Instrucción BLE.

1.8.23 BLO *(bifurca si es menor)*

Esta instrucción se encarga de bifurcar si el resultado de una comparación resulta que el acumulador es menor que el operando. La CPU se vale del bit de carga, si este es igual a 1 entonses la instrucción se ejecuta.
Esta instrucción es idéntica que BCS

BLO *rel*	Branch if Lower (Same as BCS)	PC ← (PC) + 2 + *rel* ? (C) = 1	–	–	–	–	–	–	REL	25	rr	3

Figura 1.51: Instrucción BLO

1.8.24 BLS *(bifurca si es menor o igual)*

Esta instrucción se encarga de bifurcar si el resultado de una comparación resulta que el acumulador es menor o igual que el operando. Para esto la CPU se vale de los bits C y Z con los cuales ejecuta una operación lógica OR, si el resultado es 1 significa que la condición "bifurca si es menor o igual" es cierta por lo que la instrucción se ejecuta.

BLS *rel*	Branch if Lower or Same	PC ← (PC) + 2 + *rel* ? (C) I (Z) = 1	–	–	–	–	–	–	REL	23	rr	3

Figura 1.52: Instrucción BLS.

1.8.25 BLT *(bifurca si es menor [con operandos signados])*

Esta instrucción se encarga de bifurcar si el resultado de una comparación resulta que el acumulador es menor que el operando, solo que en este caso el operando y el acumulador se toman como números signados tal y como ya se ha visto en la tabla 1.23. La CPU realiza una OR exclusiva entre los bit´s N y V, si esto resultra 1 quiere decir que la condicion es cierta y por lo tanto el programa bifurca.

BLT *opr*	Branch if Less Than (Signed Operands)	PC ← (PC) + 2 + *rel* ? (N ⊕ V) =1	–	–	–	–	–	–	REL	91	rr	3

Figura 1.53: Instrucción BLT.

1.8.26 BMC *(bifurca si la interrupción enmascarada esta puesta a 0)*

Sencilla instrucción en la que el programa bifurcará si el bit I (interrupción enmascarada) esta puesto a 0, este 0 significa que las interrupciones están habilitadas.

BMC *rel*	Branch if Interrupt Mask Clear	PC ← (PC) + 2 + *rel* ? (I) = 0	–	–	–	–	–	–	REL	2C	rr	3

Figura 1.54: Instrucción BMC.

1.8.27 BMI *(bifurca si es negativo)*

Con esta instrucción el programa bifurcará a la dirección indicada por la etiqueta siempre y cuando el bit N este puesto a 1, es decir después de cualquier comparación donde resulten términos negativo o en el caso en que mas se usa: una resta con resultado negativo de $FF a $80, en decimal que sería de -1 a -128. Como se a mencionado antes el bit N estando puesto a 1 indica que el bit 7 es el signo que por obvio resultado de la resta esta en 1, por lo tanto solo se aprovechan 7 bits con los que solo se puede representar asta el número 128 pero que pasa si restamos $00 - $FE, esto seria igual a -254 resultado que no puede ser visualizado con 7 bits por lo que el bit N no se pone a 1, en este caso el que entra a ayudar es el bit C indicando no un desbordamiento como en la suma con resultado mayor a 255 sino un número negativo menor a -128 como podrían ser desde -129 asta -255. No confunda esto con

el número signado. Debido a esto esta instrucción se nombraría mejor: bifurca si N esta puesto a 1.

BMI *rel*	Branch if Minus	PC ← (PC) + 2 + *rel* ? (N) = 1	–	–	–	–	–	–	REL	2B	rr	3

Figura 1.55: Instrucción BMI

1.8.28 BMS *(bifurca si la interrupción enmascarada esta puesta a 1)*

Sencilla instrucción en la que el programa bifurca si las interrupciones enmascaradas están deshabilitadas es decir, que el bit I del registro CCR esta puesto a 1.

BMS *rel*	Branch if Interrupt Mask Set	PC ← (PC) + 2 + *rel* ? (I) = 1	–	–	–	–	–	–	REL	2D	rr	3

Figura 1.56: Instrucción BMS

1.8.29 BNE *(bifurca si no es igual)*

Instrucción en la que se bifurca si en alguna comparación el acumulador y el operando no resultan ser iguales, o lo que es lo mismo el bit Z es igual a 0, otro caso sería que el resultado de alguna operación difiere de 0. Esta instrucción también podría llamarse “Bifurca si no es cero”.

BNE *rel*	Branch if Not Equal	PC ← (PC) + 2 + *rel* ? (Z) = 0	–	–	–	–	–	–	REL	26	rr	3

Figura 1.57: Instrucción BNE.

1.8.30 BPL *(bifurca si es positivo)*

Instrucción en la que se bifurca cuando el resultado de una operación o comparación resulta ser positivo es decir que la bandera N es igual a 0, aunque no se cumple en todos los caso debido a lo ya visto en la instrucción BMI ya que en una resta en la que el resultado es menor a -128, o mejor dicho resultados que van de -129 a -255 el bit N se pone a 0 por lo tanto a pesar de que es un número negativo marcado por el bit C el programa no bifurca. Esta instrucción se nombraría mejor: bifurca si N esta puesto a 0.

BPL *rel*	Branch if Plus	PC ← (PC) + 2 + *rel* ? (N) = 0	–	–	–	–	–	–	REL	2A	rr	3

Figura 1.58: Instrucción BPL.

1.8.31 BRA *(bifurca siempre)*

Con esta instrucción se bifurca siempre, es una bifurcación incondicional pero no deja de ser modo de direccionamiento relativo por lo que está limitada a un cierto número de desplazamientos como ya se ha mencionado antes.

BRA *rel*	Branch Always	PC ← (PC) + 2 + *rel*	–	–	–	–	–	–	REL	20	rr	3

Figura 1.59: Instrucción BRA.

1.8.32 BRCLR *(bifurca si esta puesto a 0 el bit n en M)*

Instrucción en la que el programa pregunta si esta puesto a 0 algún bit en particular de alguna localidad de memoria directa, si la condición se cumple el programa bifurca.

BRCLR *n,opr,rel*	Branch if Bit *n* in M Clear	PC ← (PC) + 3 + *rel* ? (Mn) = 0	–	–	–	–	–	↕	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	01 03 05 07 09 0B 0D 0F	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5

Figura 1.60: Instrucción BRCLR.

Ejemplo:

Supóngase que el puerto PTA 5 está configurado como entrada

```
Subrutina:
            BRCLR     5,PTA, ET_01
            NOP
            STOP

ET_01:      RTS
```

Explicación:

Si al entrar a la subrutina el PTA 5 tiene un nivel de voltaje bajo el programa bifurcará hacia la dirección indicada por la etiqueta 1 (ET_01) en este caso es la instrucción RTS por esto el programa regresará al programa principal, pero si el PTA 5 tiene un nivel de voltaje alto el programa no bifurcará y se ejecutará la instrucción STOP por lo cual el microcontrolador se detendrá.

1.8.33 BRSET *(bifurca si esta puesto a 1 el bit n en M)*

Instrucción en la que el programa pregunta si esta puesto a 1 algún bit en particular de alguna localidad de memoria directa, si la condición se cumple el programa bifurca.

BRSET *n,opr,rel*	Branch if Bit *n* in M Set	PC ← (PC) + 3 + *rel* ? (Mn) = 1	–	–	–	–	–	t	DIR (b0) DIR (b1) DIR (b2) DIR (b3) DIR (b4) DIR (b5) DIR (b6) DIR (b7)	00 02 04 06 08 0A 0C 0E	dd rr dd rr dd rr dd rr dd rr dd rr dd rr dd rr	5 5 5 5 5 5 5 5

Figura 1.61: Instrucción BRSET.

Ejemplo:

Supóngase que el puerto PTB 2 está configurado como entrada

```
Subrutina:
            BRSET     2,PTB, ET_01
            NOP
            STOP

ET_01:      RTS
```

Explicación:

Si al entrar a la subrutina el PTB 2 tiene un nivel de voltaje alto el programa bifurcará hacia la dirección indicada por la etiqueta 1 (ET_01) en este caso es la instrucción RTS por esto el programa regresará al programa principal, pero si el PTB 2 tiene un nivel de voltaje bajo el programa no bifurcará y se ejecutará la instrucción STOP por lo cual el microcontrolador se detendrá.

1.8.34 BSR *(bifurca hacia la subrutina)*

Esta instrucción hace bifurcar al programa hacia la subrutina indicada por el programador.
NOTA: Mire la instrucción JSR

BSR *rel*	Branch to Subroutine	PC ← (PC) + 2; push (PCL) SP ← (SP) – 1; push (PCH) SP ← (SP) – 1 PC ← (PC) + *rel*	–	–	–	–	–	–	REL	AD	rr	4

Figura 1.62: Instrucción BSR

1.8.35 CBEQ, CBEQA, CBEQX *(compara y bifurca si son iguales)*

CBEQ: Esta instrucción se encarga de comparar al acumulador con algún operando compatible y bifurcar si estos 2 son iguales.
CBEQA: Esta instrucción se encarga de comparar al acumulador con un operando inmediato es decir, que el operando se conoce a la hora de escribir el programa o mejor dicho es una constante.
CBEQX: Esta instrucción se encarga de comparar al registro de índice X con un operando inmediato es decir, que el operando se conoce a la hora de escribir el programa o mejor dicho es una constante.

CBEQ *opr,rel*	Compare and Branch if Equal	PC ← (PC) + 3 + rel ? (A) – (M) = $00	–	–	–	–	–	–	DIR	31	dd rr	5
CBEQA #*opr,rel*		PC ← (PC) + 3 + rel ? (A) – (M) = $00							IMM	41	ii rr	4
CBEQX #*opr,rel*		PC ← (PC) + 3 + rel ? (X) – (M) = $00							IMM	51	ii rr	4
CBEQ *opr,X+,rel*		PC ← (PC) + 3 + rel ? (A) – (M) = $00							IX1+	61	ff rr	5
CBEQ X+,*rel*		PC ← (PC) + 2 + rel ? (A) – (M) = $00							IX+	71	rr	4
CBEQ *opr*,SP,*rel*		PC ← (PC) + 4 + rel ? (A) – (M) = $00							SP1	9E61	ff rr	6

Figura 1.63: Instrucción CBEQ.

1.8.36 CLC *(pon a 0 el bit de carry)*

Esta instrucción se encarga de limpiar el bit de carga es decir lo pone a 0.

CLC	Clear Carry Bit	C ← 0	–	–	–	–	–	0	INH	98		1

Figura 1.64: Instrucción CLC.

1.8.37 CLI *(pon a 0 el bit I)*

Esta instrucción se encarga de habilitar las interrupciones enmascaradas es decir que pone a 0 al bit I del registro CCR.

CLI	Clear Interrupt Mask	I ← 0	–	–	0	–	–	–	INH	9A		2

Figura 1.65: Instrucción CLI

1.8.38 CLR, CLRA, CLRX, CLRH *(limpia)*

CLR: limpia (pone a $00) la localidad de memoria.
CLRA: limpia al acumulador.
CLRX: limpia al registro X.
CLRH: limpia al registro H.

CLR *opr*	Clear	M ← $00	0	–	–	0	1	–	DIR	3F	dd	3
CLRA		A ← $00							INH	4F		1
CLRX		X ← $00							INH	5F		1
CLRH		H ← $00							INH	8C		1
CLR *opr*,X		M ← $00							IX1	6F	ff	3
CLR ,X		M ← $00							IX	7F		2
CLR *opr*,SP		M ← $00							SP1	9E6F	ff	4

Figura 1.65: Instrucción CLR

1.8.39 COM, COMA, COMX *(complemento a 1)*

COM: Se obtiene el complemento a 1 del contenido de la localidad de memoria, el resultado se almacena en la misma localidad de memoria.
COMA: Se obtiene el complemento a 1 del acumulador el resultado se almacena en el mismo acumulador.
COMX: Se obtiene el complemento a 1 del registro índice X, el resultado se almacena en el mismo registro.

COM *opr*	Complement (One's Complement)	M ← (M) = $FF – (M)	0	–	–	↕	↕	1	DIR	33	dd	4
COMA		A ← (A) = $FF – (M)							INH	43		1
COMX		X ← (X) = $FF – (M)							INH	53		1
COM *opr*,X		M ← (M) = $FF – (M)							IX1	63	ff	4
COM ,X		M ← (M) = $FF – (M)							IX	73		3
COM *opr*,SP		M ← (M) = $FF – (M)							SP1	9E63	ff	5

Figura 1.66: Instrucción COM

1.8.40 CPHX *(compara al registro de índice H:X)*

Esta instrucción compara al registro índice H:X con algún operando, este puede ser solo directo o inmediato, esta instrucción solo afecta a las banderas de estado tal y cual lo ase la instrucción CMP solo que en este caso los operandos son de 16 bits

Ejemplo:

```
CPHX    #$8050    ; direccionamiento inmediato

CPHX    $40       ; direccionamiento directo
```

En el segundo ejemplo se esta comparando al registro H:X con las localidades de memoria $40 y $41 siendo la $40 la parte mas significativa y $41 la parte menos significativa de la palabra de 16 bit´s con la que se esta comparando al registro H:X.

Ahora bien si el registro A esta en la localidad de memoria $40 y el registro B en la localidad $41.

```
CPHX        REG_A
```

Sucede lo mismo, el registro A es la parte mas significativa y el registro B la menos significativa de la palabra de 16 bit´s con la que se esta comparando al registro H:X

CPHX #*opr* CPHX *opr*	Compare H:X with M	(H:X) – (M:M + 1)	↕	–	–	↕	↕	↕	IMM DIR	65 75	ii ii+1 dd	3 4

Figura 1.67: Instrucción CPHX.

1.8.41 *CPX (compara al registro índice X con el operando)*

Instrucción que su funcionamiento es idéntico a la instrucción CMP solo que esta vez se compara con el registro X en ves del acumulador. Consulte la instrucción CMP.

CPX #*opr* CPX *opr* CPX *opr* CPX ,X CPX *opr*,X CPX *opr*,X CPX *opr*,SP CPX *opr*,SP	Compare X with M	(X) – (M)	↕	–	–	↕	↕	↕	IMM DIR EXT IX2 IX1 IX SP1 SP2	A3 B3 C3 D3 E3 F3 9EE3 9ED3	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

Figura 1.68: Instrucción CPX.

1.8.42 DAA *(ajuste decimal)*

Instrucción que convierte el valor del acumulador de hexadecimal a decimal del número $0A al $0F tal y como se muestra en la tabla 1.70, los valores siguientes necesitan factores de corrección por lo cual no recomiendo su uso, los valores que no aparecen en la tabla es por que no producen cambio alguno en el acumulador; en capítulos posteriores se mostrará como convertir números en hexadecimal a números en decimales

DAA	Decimal Adjust A	$(A)_{10}$	U	–	–	↕	↕	↕	INH	72		2

Figura 1.69: Instrucción DAA

Hexa	ADD	Hexa	ADD	Hexa	ADD	Hexa	ADD
$0A	10	$4A	50	$8A	90	$CA	30
$0B	11	$4B	51	$8B	91	$CB	31
$0C	12	$4C	52	$8C	92	$CC	32
$0D	13	$4D	53	$8D	93	$CD	33
$0E	14	$4E	54	$8E	94	$CE	34
$0F	15	$4F	55	$8F	95	$CF	35
$1A	20	$5A	60	$9A	00	$DA	40
$1B	21	$5B	61	$9B	01	$DB	41
$1C	22	$5C	62	$9C	02	$DC	42
$1D	23	$5D	63	$9D	03	$DD	43
$1E	24	$5E	64	$9E	04	$DE	44
$1F	25	$5F	65	$9F	05	$DF	45
$2A	30	$6A	70	$AA	10	$EA	50
$2B	31	$6B	71	$AB	11	$EB	51
$2C	32	$6C	72	$AC	12	$EC	52
$2D	33	$6D	73	$AD	13	$ED	53
$2E	34	$6E	74	$AE	14	$EE	54
$2F	35	$6F	75	$AF	15	$EF	55
$3A	40	$7A	80	$BA	20	$FA	60
$3B	41	$7B	81	$BB	21	$FB	61
$3C	42	$7C	82	$BC	22	$FC	26
$3D	43	$7D	83	$BD	23	$FD	63
$3E	44	$7E	84	$BE	24	$FE	64
$3F	45	$7F	85	$BF	25	$FF	65

Figura 1.70: Tabla del ajuste decimal.

1.8.43 DBNZ, DBNZA, DBNZX *(decrementa y bifurca si no es cero)*

Instrucción que su primera acción es decrementar en 1 al operando y si después de esto el operando es diferente de cero bifurca hacia la etiqueta apuntada.
La forma de acceder al los datos de esta instrucción puede ser inherente, directo o indexado pero al momento de bifurcas se usa el modo de direccionamiento relativo.

DBNZ: es utilizado para los diferentes operandos con los que es compatible la instrucción.
DBNZA: decrementa y bifurca si el acumulador es 0.
DBNZX: decrementa y bifurca si el registro de índice X es 0.

DBNZ *opr,rel*	Decrement and Branch if Not Zero	A ← (A) – 1 or M ← (M) – 1 or X ← (X) – 1 PC ← (PC) + 3 + *rel* ? (result) ≠ 0	–	–	–	–	–	–	DIR	3B	dd rr	5
DBNZA *rel*		PC ← (PC) + 2 + *rel* ? (result) ≠ 0							INH	4B	rr	3
DBNZX *rel*		PC ← (PC) + 2 + *rel* ? (result) ≠ 0							INH	5B	rr	3
DBNZ *opr,X,rel*		PC ← (PC) + 3 + *rel* ? (result) ≠ 0							IX1	6B	ff rr	5
DBNZ X,*rel*		PC ← (PC) + 2 + *rel* ? (result) ≠ 0							IX	7B	rr	4
DBNZ *opr,SP,rel*		PC ← (PC) + 4 + *rel* ? (result) ≠ 0							SP1	9E6B	ff rr	6

Figura 1.71: Instrucción DBNZ .

Ejemplo:

Se desea detener el microcontrolador cuando el registro A llegue a $00, el valor inicial del registro es $FF.

```
Subrutina:
            DBNZ        REG_A, ET_01
            STOP
ET_01:      RTS
```

Explicación:

La primera ves que sea llamada esta subrutina el registro A sufrirá un decremento debido a la instrucción DBNZ para obtener el nuevo valor de $FE; después el microcontrolador verifica que este valor sea diferente de 0 y como esta condición se cumple el programa bifurca hacia la etiqueta 1. Cuando esta subrutina sea llamada por 255ª ves la instrucción DBNZ habrá decrementado $FF veces al registro A lo que le dará el valor de $00 por lo que el programa no bifurca, lo que provocará que la instrucción STOP se ejecute.

1.8.44 DEC, DECA, DECX *(decrementa)*

Instrucción en la que la CPU decrementa al operando en 1 el resultado lo guarda en el mismo operando.
DEC: decrementa los operandos con los que es compatible la instrucción.
DECA: decrementa al acumulador
DECX: decrementa al registro de índice X.

DEC *opr*	Decrement	M ← (M) – 1	↕	–	–	↕	↕	–	DIR	3A	dd	4
DECA		A ← (A) – 1							INH	4A		1
DECX		X ← (X) – 1							INH	5A		1
DEC *opr*,X		M ← (M) – 1							IX1	6A	ff	4
DEC ,X		M ← (M) – 1							IX	7A		3
DEC *opr*,SP		M ← (M) – 1							SP1	9E6A	ff	5

Figura 1.72: Instrucción DEC.

1.8.45 DIV *(divide)*

Poderosísima instrucción con la que se ahorran muchísimos pasos con relación a otros microcontroladores que no poseen la división como instrucción, por lo que es necesario programar paso a paso esta operación.

Esta instrucción realiza una división tomando como dividendo a un registro auxiliar de 16 bits armado por la CPU entre el registro H y el acumulador siendo H el byte de mayor peso; el divisor debe ser colocado en el registro de índice X. El resultado de la división se guarda en el acumulador, el residuo se almacena en el registro H y el registro X no es afectado.

$$\frac{H:A}{X} = A \quad ;RESIDUO = H$$

DIV	Divide	A ← (H:A)/(X) H ← Remainder	–	–	–	–	↕	↕	INH	52		7

Figura 1.73: Instrucción DIV.

Ejemplo:
Divide el número $80 entre $06

```
LDHX    #$0006
LDA     #$80        ; H:A = $0080, X = $06

DIV                 ; A = $15
                    ; H = $02
                    ; X = $06,
```

Explicación:

Cuando se ejecuta la instrucción DIV el resultado se guarda en el acumulador el cual es $15 mientras el residuo que es $02 se almacena en H y como ya se menciono, esta instrucción no afecta al registro X.

Ejemplo:

Divide el número $0180 entre $10

```
LDHX    #$0110
LDA     #$80        ; H:A = $0180, X = $10

DIV                 ; A = $18
                    ; H = $00
                    ; X = $10,
```

Explicación:

Cuando se ejecuta la instrucción DIV el resultado se guarda en el acumulador el cual es $18 mientras el residuo que es $00 se almacena en H y como ya se mencionó, esta instrucción no afecta al registro X.

1.8.46 EOR *(or exclusiva)*

Instrucción en la que se ejecuta una operación OR exclusiva entre el acumulador y el operando, el resultado se almacena en el acumulador.

EOR #*opr*	Exclusive OR M with A	A ← (A ⊕ M)	0	–	–	↕	↕	–	IMM	A8	ii	2
EOR *opr*									DIR	B8	dd	3
EOR *opr*									EXT	C8	hh ll	4
EOR *opr*,X									IX2	D8	ee ff	4
EOR *opr*,X									IX1	E8	ff	3
EOR ,X									IX	F8		2
EOR *opr*,SP									SP1	9EE8	ff	4
EOR *opr*,SP									SP2	9ED8	ee ff	5

Figura 1.74: Instrucción EOR.

1.8.47 INC, INCA, INCX *(incrementa)*

Instrucción que incrementa en 1 al operando, al registro de índice X o al acumulador, esta instrucción no pone a 1 el bit de carga a causa de un desbordamiento del registro u operando, tampoco afecta al registro H en caso de desbordamiento del registro X.

INC: incrementa los operandos con los que es compatible la instrucción.
INCA: incrementa al acumulador
INCX: incrementa al registro de índice X.

INC *opr*	Increment	M ← (M) + 1	↕	–	–	↕	↕	–	DIR	3C	dd	4
INCA		A ← (A) + 1							INH	4C		1
INCX		X ← (X) + 1							INH	5C		1
INC *opr*,X		M ← (M) + 1							IX1	6C	ff	4
INC ,X		M ← (M) + 1							IX	7C		3
INC *opr*,SP		M ← (M) + 1							SP1	9E6C	ff	5

Figura 1.75: Instrucción INC. 1

1.8.48 JMP *(brinco incondicional)*

Instrucción que genera un brinco incondicional, esta instrucción al no ser modo direccionamiento relativo puede abarcar saltos en todo el mapa de memoria, si se utilizan etiquetas para las direcciones de salto el programa ensamblador (codewarrior) calcula la dirección y por lo tanto también escoge el código de operación ya sea directo o extendido, como el programa se escribe dentro de localidades de memoria extendidas siempre se utilizará el direccionamiento extendido, salvo en condiciones muy especiales como el uso de bloques de la memoria FLASH para el almacenamiento de datos no volátiles. También es posible saltar a la dirección apuntada en el registro índice H:X con o sin desplazamiento de 8 o 16 bits es decir es compatible con los tres modos de direccionamiento indexado.

JMP *opr*	Jump	PC ← Jump Address	–	–	–	–	–	–	DIR	BC	dd	2
JMP *opr*									EXT	CC	hh ll	3
JMP *opr*,X									IX2	DC	ee ff	4
JMP *opr*,X									IX1	EC	ff	3
JMP ,X									IX	FC		2

Figura 1.76: Instrucción JMP.

1.8.49 JSR *(brinco a subrutina)*

Instrucción que realiza un salto incondicional hacia una subrutina

JSR *opr* JSR *opr* JSR *opr*,X JSR *opr*,X JSR ,X	Jump to Subroutine	PC ← (PC) + *n* (*n* = 1, 2, or 3) Push (PCL); SP ← (SP) – 1 Push (PCH); SP ← (SP) – 1 PC ← Unconditional Address	–	–	–	–	–	–	DIR EXT IX2 IX1 IX	BD CD DD ED FD	dd hh ll ee ff ff	4 5 6 5 4

Figura 1.77: Instrucción JSR.

En los saltos o bifurcaciones hacia una subrutina la dirección en la que se encuentra el contador de programa es almacenada en el STACK para su futuro retorno al programa principal al concluir la subrutina. Se recomienda ampliamente salir de la subrutina a través de la instrucción RTS ya que si sale de la subrutina a través de un BRA o JMP con el transcurso de repeticiones de esta subrutina terminara por saturar al stack y como el stack esta en una parte de la memoria RAM continuará saturando esta ultima perdiendo así todo lo que este almacenado en ella para después afectar a los registros de control de la CPU provocando así un fallo total en el funcionamiento en el CPU solucionable solo a través de un reset manual por hardware.
Si se desea hacer un retorno de subrutina redirigido a través de la instrucción BRA o JMP (o cualquier otra instrucción de bifurcación) es decir una dirección diferente de la que se partió en el programa principal, se recomienda sacar la ultima dirección apuntada en el stack antes del salto para esto nos valemos de instrucciones como PULA, PULH, y PULX

Ejemplo:

```
Subrutina:      LDA     REG_A
                CMP     #$80
                BHI     ET_01
                RTS

ET_01:          PULA                ; se saca la parte mas alta del ultimo byte
                                    ; colocado en el stack debido a JSR o BSR
                PULA                ; se saca la parte mas baja  de la dirección
                JMP     retorno_dirigido
```

Explicación:

Cuando se entra en la subrutina el acumulador se carga con el contenido del registro A por lo tanto, si este registro es menor a $80 el programa retornará de manera normal a una instrucción después de donde el programa partió hacia la subrutina ya que la instrucción RTS saca automáticamente la ultima dirección apuntada en el stack.

Ahora bien si el registro A es mayor a $80 se ejecutara un retorno de subrutina dirigido para esto es necesario sacar manualmente una dirección del stack, si se desea guardar este dato puede recurrir al siguiente ejemplo:

Ejemplo:

```
Subrutina:      LDA     REG_A
                CMP     #$80
                BHI     ET_01
                RTS

ET_01:          PULH
                PULX
                STHX    $82
JMP             retorno_dirigido
```

Explicación:

La parte alta de la dirección se almacena en H, la parte baja en X, para después H quedar almacenado en la localidad $82 y X en la $83.

1.8.50 LDA *(carga en el acumulador)*

Instrucción que coloca en el acumulador el dato contenido en el operando.

LDA #*opr* LDA *opr* LDA *opr* LDA *opr*,X LDA *opr*,X LDA ,X LDA *opr*,SP LDA *opr*,SP	Load A from M	A ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	A6 B6 C6 D6 E6 F6 9EE6 9ED6	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

Figura 1.78: Instrucción LDA.

Ejemplos:

Almacena una constante: dato conocido a la hora de escribir el programa.

```
LDA     #$80    ; A = 80
```

Si el registro A se encuentra en la localidad de memoria $80 y tiene contenido el dato $55.

```
LDA     $80     ; A = $55
```

Idéntico a:

```
LDA     REG_A   ; A = $55
```

1.8.51 LDHX *(carga en H:X)*

Carga en el registro H:X un dato inmediato o directo

LDHX #*opr* LDHX *opr*	Load H:X from M	H:X ← (M:M + 1)	0	–	–	↕	↕	–	IMM DIR	45 55	ii jj dd	3 4

Figura 1.79: Instrucción LDHX.

Ejemplos:

Almacena una constante: dato conocido a la hora de escribir el programa.

```
LDHX        #$8569        ; H:X = $8569
```

Si el registro A de 2 bytes se encuentra en las localidades de memoria $80 y $81 y tiene contenido el dato $65F6. La parte alta del byte se guarda en la localidad $80 mientras que la parte baja se almacena en la localidad $81.

```
LDHX        $80           ; H:X = $65F6
```

Idéntico a:

```
LDHX        REG_A         ; H:X = $65F6
```

1.8.52 LDX *(carga en X)*

Instrucción que coloca en el registro X el dato correspondiente al operando utilizado
NOTA: Mire la instrucción LDA para una mayor comprensión.

LDX #*opr* LDX *opr* LDX *opr* LDX *opr*,X LDX *opr*,X LDX ,X LDX *opr*,SP LDX *opr*,SP	Load X from M	X ← (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	AE BE CE DE EE FE 9EEE 9EDE	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

Figura 1.80: Instrucción LDX.

1.8.53 LSL, LSLA, LSLX *(deslizamiento lógico a la izquierda)*

Instrucción que realiza un deslizamiento lógico a la izquierda. Esta instrucción carga un 0 en el bit de menor peso y el de mayor peso se traslada al bit de carga, lo que estaba en la carga antes de la instrucción LSL se elimina. Esta instrucción es idéntica a ASL.

LSL: afecta al operando, el resultado se almacena en la localidad del operando.
LSLA: afecta al acumulador, el resultado se almacena en el acumulador.
LSLX: afecta al registro de índice X, el resultado se almacena en el registro X.

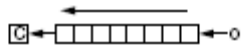
LSL *opr* LSLA LSLX LSL *opr*,X LSL ,X LSL *opr*,SP	Logical Shift Left (Same as ASL)		↕	–	–	↕	↕	↕	DIR INH INH IX1 IX SP1	38 48 58 68 78 9E68	dd ff ff	4 1 1 4 3 5

Figura 1.81: Instrucción LSL.

1.8.54 LSR, LSRA, LSRX *(deslizamiento lógico a la derecha)*

Instrucción que realiza un deslizamiento lógico a la derecha. Esta instrucción carga un 0 en el bit de mayor peso y el de menor peso se traslada al bit de carga, lo que estaba en la carga antes de la instrucción LSR se elimina.

LSR: afecta al operando, el resultado se almacena en la localidad del operando.
LSRA: afecta al acumulador, el resultado se almacena en el acumulador.
LSRX: afecta al registro de índice X, el resultado se almacena en el registro X.

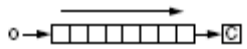
LSR *opr* LSRA LSR*X* LSR *opr*,X LSR ,X LSR *opr*,SP	Logical Shift Right		↕	–	–	0	↕	↕	DIR INH INH IX1 IX SP1	34 44 54 64 74 9E64	dd ff ff	4 1 1 4 3 5

Figura 1.82: Instrucción LSR.

1.8.55 MOV *(mueve)*

Instrucción que mueve el dato contenido en alguna localidad de memoria a otra.
NOTA: Para una mayor comprensión consulte modo de direccionamiento de memoria a memoria.

MOV *opr*,*opr* MOV *opr*,X+ MOV #*opr*,*opr* MOV X+,*opr*	Move	$(M)_{Destination} \leftarrow (M)_{Source}$ H:X ← (H:X) + 1 (IX+D, DIX+)	0	–	–	↕	↕	–	DD DIX+ IMD IX+D	4E 5E 6E 7E	dd dd dd ii dd dd	5 4 4 4

Figura 1.83: Instrucción MOV

1.8.56 MUL *(multiplicación sin signo)*

Instrucción que realiza una multiplicación entre el contenido del registro de índice X y el contenido del acumulador para después colocar el resultado de esta operación en los registros X:A, siendo X la parte mas significativa y A la menos significativa. El resultado se expresará en dos bytes tornando a $00 al registro X en los resultados pertinentes.

MUL	Unsigned multiply	X:A ← (X) × (A)	–	0	–	–	–	0	INH	42		5

Figura 1.84: Instrucción MUL.

Ejemplos:

```
LDA     #$02
LDX     #$2F
MUL                     ; $2F x $02 = $005E
                        ; 47 x 2 = 94
```

Explicación:

El resultado se almacena en los registros X:A, por lo tanto X es igual a $00 y A es igual a $5E.

Ejemplos:

```
LDA     #$FF
LDX     #$FF
MUL                     ; $FF x $FF = $FE01

                        ; 255 x 255 = 65025
```

Explicación:
El resultado se almacena en los registros X:A, por lo tanto X es igual a $FE y A es igual a $01.

1.8.57 NEG, NEGA, NEGX *(complemento a dos)*

Instrucción que obtiene el complemento a dos del operando, el acumulador o el registro X, el resultando se almacena en el operando, el acumulador o el registro X respectivamente

NEG: afecta al operando, el resultado se almacena en la localidad del operando.
NEGA: afecta al acumulador, el resultado se almacena en el acumulador
NEGX: afecta al registro de índice X, el resultado se almacena en el registro X.

NEG *opr*	Negate (Two's Complement)	M ← –(M) = $00 – (M)	↕	–	–	↕	↕	↕	DIR	30	dd	4
NEGA		A ← –(A) = $00 – (A)							INH	40		1
NEGX		X ← –(X) = $00 – (X)							INH	50		1
NEG *opr*,X		M ← –(M) = $00 – (M)							IX1	60	ff	4
NEG ,X		M ← –(M) = $00 – (M)							IX	70		3
NEG *opr*,SP		M ← –(M) = $00 – (M)							SP1	9E60	ff	5

Figura 1.85: Instrucción NEG.

1.8.58 NOP *(no operación)*

Instrucción que no realiza ninguna operación, es utilizada para ajustar retardos y hacerlos así mas exactos debido a que esta instrucción solo gasta en tiempo un ciclo maquina, es decir con un cristal de 32 MHz el ciclo maquina tiene un periodo de 125 nanosegundos y este es el tiempo que tarda en ejecutarse esta instrucción

NOP	No Operation	None	–	–	–	–	–	–	INH	9D		1

Figura 1.86: Instrucción NOP.

1.8.59 NSA *(cambio departe)*

Instrucción que cambia los 4 bits más significativos del acumulador por los 4 bits menos significativos y viceversa el resultado se almacena en el acumulador

Ejemplos:

Si el acumulador es igual a $7F tendremos:

```
NSA                    ; A = $F7
```

NSA	Nibble Swap A	A ← (A[3:0]:A[7:4])	–	–	–	–	–	–	INH	62		3

Figura 1.87: Instrucción NSA.

1.8.60 ORA *(operación lógica OR)*

Instrucción que realiza una operación lógica OR entre el acumulador y algún tipo de operando, el resultado se almacena en el acumulador.

Ejemplos:

Se desea ejecutar una operación lógica OR entre el acumulador el cual es igual a $DE y el dato inmediato $55.

```
LDA     #$DE
ORA     #$55          ; A = DF
```

A = $DE = 1 1 0 1 1 1 1 0
M = $55 = 0 1 0 1 0 1 0 1
A = $DF = 1 1 0 1 1 1 1 1

ORA #*opr* ORA *opr* ORA *opr* ORA *opr*,X ORA *opr*,X ORA ,X ORA *opr*,SP ORA *opr*,SP	Inclusive OR A and M	A ← (A) | (M)	0	–	–	↕	↕	–	IMM DIR EXT IX2 IX1 IX SP1 SP2	AA BA CA DA EA FA 9EEA 9EDA	ii dd hh ll ee ff ff ff ee ff	2 3 4 4 3 2 4 5

Figura 1.88: Instrucción ORA.

1.8.61 PSHA *(introduce el valor del acumulador en el stack)*

Instrucción que apila el valor del acumulador en el STACK

PSHA	Push A onto Stack	Push (A); SP ← (SP) – 1	–	–	–	–	–	–	INH	87		2

Figura 1.89: Instrucción PSHA.

1.8.62 PSHH *(introduce el valor del registro H en el stack)*

Instrucción que apila el valor del registro H en el STACK

PSHH	Push H onto Stack	Push (H); SP ← (SP) – 1	–	–	–	–	–	–	INH	8B		2

Figura 1.90: Instrucción PSHH.

1.8.63 PSHX *(introduce el valor del registro X en el stack)*

Instrucción que apila el valor del registro X en el STACK

PSHX	Push X onto Stack	Push (X); SP ← (SP) – 1	–	–	–	–	–	–	INH	89		2

Figura 1.91: Instrucción PSHX.

1.8.64 PULA *(saca un dato de la pila y almacénalo en el acumulador)*

Instrucción que saca el último dato colocado en el stack y lo almacena en el acumulador.

PULA	Pull A from Stack	SP ← (SP + 1); Pull (A)	–	–	–	–	–	–	INH	86		2

Figura 1.92: Instrucción PULA.

1.8.65 PULH *(saca un dato de la pila y almacénalo en el registro H)*

Instrucción que saca el último dato colocado en el stack y lo almacena en el registro H.

PULH	Pull H from Stack	SP ← (SP + 1); Pull (H)	–	–	–	–	–	–	INH	8A		2

Figura 1.93: Instrucción PULH.

1.8.66 PULX *(saca un dato de la pila y almacénalo en el registro X)*

Instrucción que saca el último dato colocado en el stack y lo almacena en el registro X.

PULX	Pull X from Stack	SP ← (SP + 1); Pull (X)	–	–	–	–	–	–	INH	88		2

Figura 1.94: Instrucción PULX.

1.8.67 ROL, ROLA, ROLX *(rota completamente a la izquierda con carry)*

Instrucción que rota a la izquierda a través del carry, observe la figura y notará que si repite 9 veces esta instrucción recuperará el valor inicial del registro.
Cuando el registro rota a la izquierda el bit 7 se desplaza al bit de carga, el bit de carga se desplaza al bit 0, el bit 0 se desplaza al bit 1 y así sucesivamente asta que el bit 6 se desplaza al bit 7.

ROL: afecta al operando, el resultado se almacena en la localidad del operando.
ROLA: afecta al acumulador, el resultado se almacena en el acumulador
ROLX: afecta al registro de índice X, el resultado se almacena en el registro X.

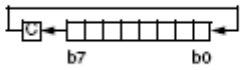
ROL *opr*	Rotate Left through Carry		↕	–	–	↕	↕	↕	DIR	39	dd	4
ROLA									INH	49		1
ROLX									INH	59		1
ROL *opr*,X									IX1	69	ff	4
ROL ,X									IX	79		3
ROL *opr*,SP									SP1	9E69	ff	5

Figura 1.95: Instrucción ROL.

1.8.68 ROR, RORA, RORX *(rota completamente a la derecha con carry)*

Instrucción que rota a la derecha a través del carry, observe la figura y notará que si repite 9 veces esta instrucción recuperará el valor inicial del registro.
Cuando el registro rota a la derecha el bit 0 se desplaza al bit de carga, el bit de carga se desplaza al bit 7, el bit 7 se desplaza al bit 6 y así sucesivamente asta que el bit 1 se desplaza al bit 0.

ROR: afecta al operando, el resultado se almacena en la localidad del operando.
RORA: afecta al acumulador, el resultado se almacena en el acumulador
RORX: afecta al registro de índice X, el resultado se almacena en el registro X.

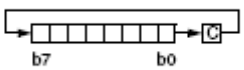
ROR *opr*	Rotate Right through Carry		↕	–	–	↕	↕	↕	DIR	36	dd	4
RORA									INH	46		1
RORX									INH	56		1
ROR *opr*,X									IX1	66	ff	4
ROR ,X									IX	76		3
ROR *opr*,SP									SP1	9E66	ff	5

Figura 1.96: Instrucción ROR.

1.8.69 RSP *(reinicia la parte baja del stack pointer)*

Instrucción que reinicia la parte baja del stack pointer, puesto que el puntero de pila es un registro de dos bytes si se desea reiniciarlo totalmente debe recurrirse a otra instrucción.

En el caso del microcontrolador MC68HC908GP32 esta instrucción es prácticamente inútil ya que RSP apunta le valor $FF en la parte baja del stack pointer y para que este sea realmente inicializado se le debe cargar el valor $023F.

NOTA: Cada valor de inicialización del stack pointer cambia a razón de cada microcontrolador y esto depende de la dirección de la última localidad de la memoria RAM. Por ejemplo para los microcontroladores MC68HC908JL3, MC68HC908JK3, MC68HC908JK1 que tienen una memoria RAM muy pequeña de 128 BYTES la cual va de la localidad $80 a la localidad $FF la instrucción RSP si es útil para la inicialización del stack pointer ya que como se ha mencionado antes el valor apuntado en el stack pointer corresponde a la ultima localidad de la memoria RAM, en este caso el valor con que se inicializa el stack pointer es el de $00FF por lo tanto con la simple ejecución de la instrucción RSP el stack pointer puede ser inicializado.

RSP	Reset Stack Pointer	SP ← $FF	–	–	–	–	–	–	INH	9C		1

Figura 1.97: Instrucción RSP.

Regresando a nuestro microcontrolador elegido el MC68HC908GP32 la primera localidad de memoria del stack es el fin de la memoria RAM es decir la localidad $023F por lo tanto el puntero de pila debe tener apuntada esta dirección, por lo que hay que llevar el dato $023F al stack pointer de la siguiente forma:

```
LDHX    #$023F
TXS     ; Después de esta instrucción el stack pointer será igual a
        ; $023F
```

Como el stack se almacena de manera descendente en la memoria es decir de la localidad $023F asta el inicio de la RAM que esta ubicada en la localidad $40 (para el caso del microcontrolador MC68HC908GP32), en el caso de haber entrado a un número extenso de subrutinas sin retornar de ellas y por consiguiente se desea reiniciar el stack pointer debe observar los dos siguientes casos para reinicia el stack pointer.

Ejemplos:

CASO 1: Se ha entrado a un pequeño número de subrutinas sin retornar de ellas en un microcontrolador MC68HC908JL3 (fin de la RAM $00FF) y se desea reiniciar al stack pointer, la dirección apuntada en el stack pointer es $0090.

Para que el stack pointer sea reiniciado independientemente de la dirección que tenga apuntada solo es necesaria la instrucción:

```
RSP     ; reinicia el stack pointer
        ; SP = $00FF
```

CASO 2: Se ha entrado a un gran número de subrutinas sin retornar de ellas en un microcontrolador MC68HC908GP32 (fin de la RAM $023F) por lo que la parte alta del stack pointer ha cambiado a $01 o a $00. Como ejemplo el stack pointer tiene apuntada la localidad $0124.

Para que el stack pointer sea reiniciado son necesarias las instrucciones:

```
LDHX    #$023F
TXS     ; Después de esta instrucción el stack pointer será igual a
        ; SP = $023F
```

Si utilizase el método del caso 1 el stack pointer tendría como resultado apuntada la dirección $01FF por lo cual el stack pointer no sería reiniciado correctamente.

1.8.70 RTI *(retorna de interrupción)*

Instrucción que debe ejecutarse al final de una interrupción para retornar al programa principal o al lugar de donde procede el programa antes de que se ejecutase la interrupción para no tener la necesidad de reiniciar el stack pointer.

RTI	Return from Interrupt	SP ← (SP) + 1; Pull (CCR) SP ← (SP) + 1; Pull (A) SP ← (SP) + 1; Pull (X) SP ← (SP) + 1; Pull (PCH) SP ← (SP) + 1; Pull (PCL)	↕	↕	↕	↕	↕	↕	INH	80		7

Figura 1.98: Instrucción RTI.

1.8.71 RTS *(retorna de subrutina)*

Instrucción que debe ejecutarse al final de una subrutina para retornar al programa principal o al lugar de donde procede el programa antes de que se ejecutase la subrutina para no tener la necesidad de reiniciar el stack pointer.

RTS	Return from Subroutine	SP ← SP + 1; Pull (PCH) SP ← SP + 1; Pull (PCL)	–	–	–	–	–	–	INH	81		4

Figura 1.99: Instrucción RTS.

1.8.72 SBC *(resta con carga)*

Instrucción que realiza una resta aritmética entre el operando y el acumulador además de restar el valor del bit de carga ya sea este 0 o 1. El resultado es almacenado en el acumulador.

SBC #*opr*	Subtract with Carry	A ← (A) – (M) – (C)	↕	–	–	↕	↕	↕	IMM	A2	ii	2
SBC *opr*									DIR	B2	dd	3
SBC *opr*									EXT	C2	hh ll	4
SBC *opr*,X									IX2	D2	ee ff	4
SBC *opr*,X									IX1	E2	ff	3
SBC ,X									IX	F2		2
SBC *opr*,SP									SP1	9EE2	ff	4
SBC *opr*,SP									SP2	9ED2	ee ff	5

Figura 1.100: Instrucción SBC.

1.8.73 SEC *(pon a 1 el bit de carga)*

Instrucción que pone a 1 al bit de carga

SEC	Set Carry Bit	C ← 1	–	–	–	–	–	1	INH	99		1

Figura 1.101: Instrucción SEC.

1.8.74 SEI *(pon a 1 el bit I)*

Esta instrucción se encarga de deshabilitar las interrupciones enmascaradas es decir que pone a 1 al bit I del registro CCR.

SEI	Set Interrupt Mask	I ← 1	–	–	1	–	–	–	INH	9B		2

Figura 1.102: Instrucción SEI.

1.8.75 STA *(almacena al acumulador)*

Instrucción que almacena el contenido del acumulador en la localidad de memoria indicada por el operando.

STA *opr*	Store A in M	M ← (A)	0	–	–	↕	↕	–	DIR	B7	dd	3
STA *opr*									EXT	C7	hh ll	4
STA *opr*,X									IX2	D7	ee ff	4
STA *opr*,X									IX1	E7	ff	3
STA ,X									IX	F7		2
STA *opr*,SP									SP1	9EE7	ff	4
STA *opr*,SP									SP2	9ED7	ee ff	5

Figura 1.103: Instrucción STA.

1.8.76 STHX *(almacena al registro H:X)*

Instrucción que almacena al registro H:X en la localidad de memoria indicada por el operando.

STHX *opr*	Store H:X in M	(M:M + 1) ← (H:X)	0	–	–	↕	↕	–	DIR	35	dd	4

Figura 1.104: Instrucción STHX.

Ejemplo:

Se desea almacenar al registro H:X el cual es igual a $F125 en la localidad de memoria $40.

```
STHX    #$40    ; Localidad $40 = $F1
                ; Localidad $41 = $25
```

Por lo tanto la localidad $40 tendrá contenido el valor $F1 y el valor $25 será almacenado en la localidad $41.

Ejemplo:

Se desea almacenar al registro H:X el cual es igual a $F125 en el registro REG_A el cual se encuentra ubicado en la localidad de memoria $0100.

```
STHX    REG_A   ; Localidad $0100 = $F1
                ; Localidad $0101= $25
```

NOTA: Si creía que esto es correcto esta equivocado ya que esta instrucción esta limitada al modo de direccionamiento directo (observe la figura 1.104) por lo cual trabaja con localidades de memoria directas y como el REG_A esta ubicado en una localidad de memoria extendida ($0100) la CPU no puede procesar tal petición, para solucionar esto solo hay que reubicar al REG_A en una localidad de memoria directa.

1.8.77 STOP *(detiene al CPU)*

Instrucción que detiene el reloj de la CPU es decir, apaga todos los módulos del microcontrolador y opcionalmente desactiva el reloj del BUS. Se sale del modo Stop por un reset, por alguna interrupción externa o por alguna interrupción TBM (interrupción que será vista en capítulos posteriores). Al ejecutarse la instrucción STOP se pone a 1 el bit I.

NOTA: para salir del modo Stop a través de una interrupción TBM es necesario que el reloj del BUS este activado.

STOP	Enable Interrupts, Stop Processing, Refer to MCU Documentation	I ← 0; Stop Processing	–	–	0	–	–	–	INH	8E		1

Figura 1.105: Instrucción STOP.

1.8.78 STX *(almacena al registro X)*

Instrucción que almacena el valor contenido en el registro de índice X en la localidad de memoria indicada por el operando.

STX *opr*	Store X in M	M ← (X)	0	–	–	↕	↕	–	DIR	BF	dd	3
STX *opr*									EXT	CF	hh ll	4
STX *opr*,X									IX2	DF	ee ff	4
STX *opr*,X									IX1	EF	ff	3
STX ,X									IX	FF		2
STX *opr*,SP									SP1	9EEF	ff	4
STX *opr*,SP									SP2	9EDF	ee ff	5

Figura 1.106: Instrucción STX.

1.8.79 SUB *(resta sin carga)*

Esta instrucción se encarga de realizar una simple resta aritmética en donde interfieren el acumulador y el operando, en este caso el bit de carga no es tomado en cuenta, el resultado se coloca en el acumulador.

SUB #*opr*	Subtract	A ← (A) – (M)	↕	–	–	↕	↕	↕	IMM	A0	ii	2
SUB *opr*									DIR	B0	dd	3
SUB *opr*									EXT	C0	hh ll	4
SUB *opr*,X									IX2	D0	ee ff	4
SUB *opr*,X									IX1	E0	ff	3
SUB ,X									IX	F0		2
SUB *opr*,SP									SP1	9EE0	ff	4
SUB *opr*,SP									SP2	9ED0	ee ff	5

Figura 1.107: Instrucción SUB.

1.8.80 SWI *(interrupción por software)*

La instrucción SWI es una interrupción no enmascarable es decir que no es afectada por el bit I del registro CCR por lo tanto sin importar el estado de este bit esta interrupción será llevada a cabo al momento de que se ejecute la instrucción SWI.

SWI	Software Interrupt	PC ← (PC) + 1; Push (PCL) SP ← (SP) – 1; Push (PCH) SP ← (SP) – 1; Push (X) SP ← (SP) – 1; Push (A) SP ← (SP) – 1; Push (CCR) SP ← (SP) – 1; I ← 1 PCH ← Interrupt Vector High Byte PCL ← Interrupt Vector Low Byte	–	–	1	–	–	–	INH	83		9

Figura 1.108: Instrucción SWI.

1.8.81 TAP *(transfiere al acumulador hacia el registro CCR)*

Instrucción que transfiere el dato contenido en el acumulador y lo coloca en el registro CCR. Observe la figura 1.11 para conocer al registro CCR.

TAP	Transfer A to CCR	CCR ← (A)	↕	↕	↕	↕	↕	↕	INH	84		2

Figura 1.109: Instrucción TAP.

1.8.82 TAX *(transfiere al acumulador hacia el registro X)*

Instrucción que transfiere el dato contenido en el acumulador y lo coloca en el registro X.

TAX	Transfer A to X	X ← (A)	–	–	–	–	–	–	INH	97		1

Figura 1.110: Instrucción TAX.

1.8.83 TPA *(transfiere al registro CCR hacia el acumulador)*

Instrucción que transfiere el dato contenido en el registro CCR y lo coloca en el acumulador.

TPA	Transfer CCR to A	A ← (CCR)	–	–	–	–	–	–	INH	85		1

Figura 1.111: Instrucción TPA.

1.8.84 TST, TSTA, TSTX *(prueba por negativo o por 0)*

Instrucción que prueba al operando, al registro X o al acumulador. Si el operando es $00 se pone a 1 el bit Z, y si el operando tiene puesto a uno el bit 7el bit N se pone a 1.

TST: la prueba se le realiza al operando.
TSTA: la prueba se le realiza al acumulador.
TSTX: la prueba se le realiza al registro X.

TST *opr* TSTA TSTX TST *opr*,X TST ,X TST *opr*,SP	Test for Negative or Zero	(A) – $00 or (X) – $00 or (M) – $00	0	–	–	↕	↕	–	DIR INH INH IX1 IX SP1	3D 4D 5D 6D 7D 9E6D	dd ff ff	3 1 1 3 2 4

Figura 1.112: Instrucción TST.

1.8.85 TSX *(transfiere al stack pointer hacia el registro H:X)*

Instrucción que toma la dirección apuntada en el stack pointer y le suma 1 para después ser almacenada en el registro H:X.

TSX	Transfer SP to H:X	H:X ← (SP) + 1	–	–	–	–	–	–	INH	95		2

Figura 1.113: Instrucción TSX.

1.8.86 TXA *(transfiere al registro X hacia el acumulador)*

Instrucción que toma el contenido del registro X y lo almacena en el acumulador.

TXA	Transfer X to A	A ← (X)	–	–	–	–	–	–	INH	9F		1

Figura 1.114: Instrucción TXA.

1.8.87 TXS *(transfiere al registro H:X hacia el stack pointer)*

Toma el dato contenido en el registro H:X y le resta 1, esto se almacena en el stack pointer.

TXS	Transfer H:X to SP	(SP) ← (H:X) – 1	–	–	–	–	–	–	INH	94		2

Figura 1.115: Instrucción TXS.

1.8.88 WAIT *(habilita interrupciones y espera por interrupciones)*

Instrucción que habilita las interrupciones (I = 0) apagando el reloj de la CPU, llevando así al microcontrolador a un estado de bajo consumo de energía mientras que el reloj del bus sigue funcionando junto con casi todos los módulos del microcontrolador en espera de alguna interrupción compatible con el modo de operación wait.

WAIT	Enable Interrupts; Wait for Interrupt	I bit ← 0; Inhibit CPU clocking until interrupted	–	–	0	–	–	–	INH	8F		1

Figura 1.116: Instrucción WAIT.

.

1.9 PROGRAMACION----------

En esta sección se explica donde deben escribirse los códigos del programa principal, los códigos de las subrutinas y los códigos de las interrupciones.

1.9.1 DECLARACION DE REGISTROS

La programación se facilita enormemente con la herramienta Codewarrior, en especial por que no es necesario declarar ningún registro de control del MCU como ejemplo podríamos mencionar que el puerto A el cual se encuentra localizado en la localidad de memoria $00 esta declarado con el nombre de "PTA" tal cual se nombra en el manual del fabricante, pasa exactamente lo mismo con todos los otros registros mencionados por el fabricante como lo son los registros de control entrada-salida de los puertos, los registros de control del Timer, el convertidor analógico digital, la interrupción en tiempo real, los módulos de comunicación serial etc.
En la mayoría de los programas es necesario declarar registros para almacenar variables aunque esto no es totalmente necesario ya que podemos utilizar las localidades de memoria directamente utilizando los números de localidad aunque es mucho mas sencillo recordar una etiqueta que en número en específico por lo tanto veremos como etiquetar o declarar localidades de memoria. Lo primero a tomar en cuenta es en que localidades de memoria podemos almacenar datos, esto se hace en la memoria RAM la cual va de la localidad $40 a la $023F, esto es en total 512 bytes, pero es necesario tomar en cuenta como mapea la memoria el Codewarrior y como etiqueta todos los registro de control ya que estos puntos

pueden variar un poco en cada MCU, para poder ver esto abrimos un proyecto Codewarrior lo cual se explicará en otro capítulo, observe la figura 1.117

Se puede observar en el árbol que se encuentra en la parte izquierda de la figura 1.117 que dentro de la carpeta **Includes** se encuentra un archivo llamado **MC68HC908GP32.inc** el cual se muestra en la figura 1.118

En la parte superior de la figura 1.118 se muestra el mapeo de la memoria, en la parte intermedia los vectores de interrupción y abajo se ve como se declara la primera localidad de memoria como PTA, este archivo es algo extenso por lo que no se muestra completo.

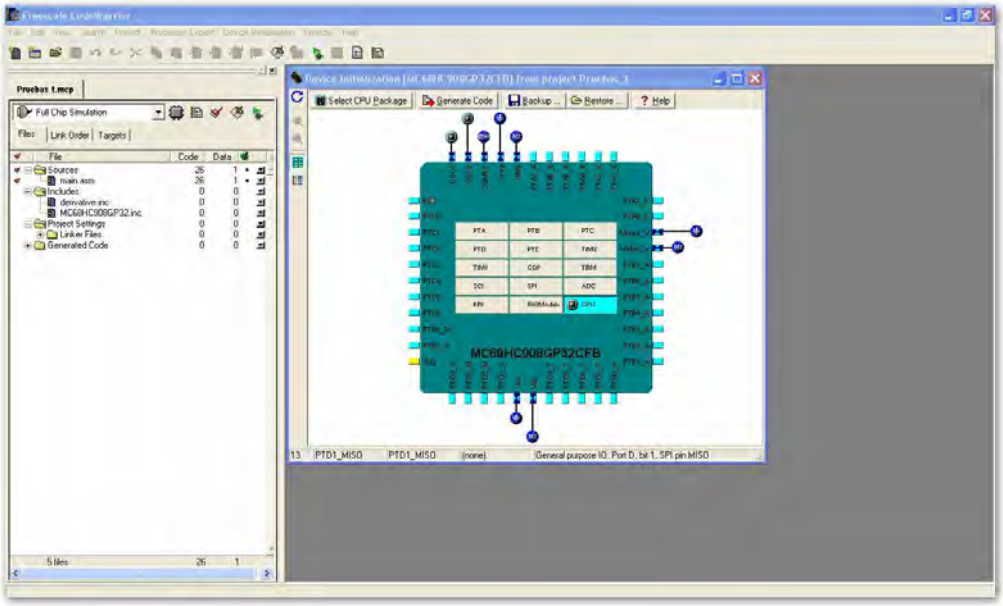


Figura 1.117: Proyecto Codewarrior.

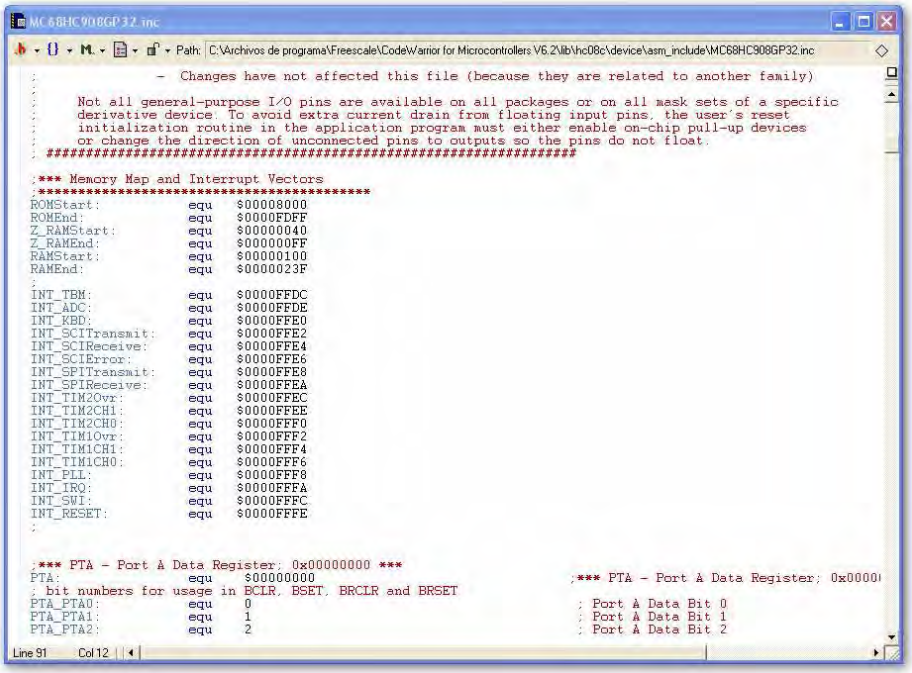


Figura 1.118: Archivo MC68HC908GP32.inc.

De lo primero que hace mención el archivo **MC68HC908GP32.inc** es de donde inicia y acaba la memoria ROM que se encarga de almacenar el programa y datos no volátiles, después viene la memoria RAM la cual una pequeña parte se encuentra en localidades de memoria directas y la mayor parte en localidades de memoria extendida por lo que este archivo nombra a la memoria RAM con localidades de memoria directa Z_RAM, y a la memoria RAM con localidades de memoria extendida simplemente como RAM, esto cambia con cada micro puesto que no todos tienen la misma cantidad de memoria RAM.
Se podrán crear registros en una localidad de de memoria directa o extendida, aunque recomiendo que esto se haga en localidades de memoria directas debido a lo siguiente:

```
CLR     REG_A     ; El registro A se encuentra en una localidad de memoria directa

CLR     REG_B     ; El registro B se encuentra en una localidad de memoria extendida
```

La primera instrucción se ejecuta normalmente mientras que la segunda causa un error en el compilador que no permitirá la simulación o grabación del MCU debido a que esta instrucción funciona con modo de direccionamiento directo y no con extendido, esto pasa con todas las instrucciones que pueden manipular localidades de memoria ya sean operaciones lógicas o aritméticas, claro que no es imposible modificar localidades de memoria extendidas para ello simplemente utilizamos modo de direccionamiento indexado, que todas las instrucciones de este tipo poseen pero siempre es mas rápido utilizar direccionamiento directo. Para declarar un registro abra el archivo **main.asm** que puede observarse en la figura 1.117, este se muestra ya abierto en la figura 1.119
Puede observarse la instrucción **“ORG RAMStart”**, ORG significa origen, y RAMStart es la etiqueta que hace referencia al mapeo de memoria que indica que RAMStart se encuentra en la localidad de memoria $0100 tal como se puede observar en la figura 1.118.
Debajo de **ORG RAMStart**, se encuentra **“ExampleVar: DS.B 1”**, esto significa que la localidad de memoria $0100 será nombrada como ExampleVar.
Observe la figura 1.120 la cual muestra el archivo **main.asm** en donde se han creado diferentes registros

- **El REG_A** mide dos bytes, la parte mas significativa se encuentra en la localidad $40 y la menos significativa en la localidad $41, nótese el número 2 al final de la instrucción es el indicativo de que el registro mide dos bytes.
- **EL REG_B** es de solo un byte y se encuentra en la localidad de memoria $42, nótese el numero 1 al final de la instrucción indicativo de que el registro mide un byte.
- **ExampleVar** se encuentra en la localidad de memoria $0100.
- Se ha almacenado una tabla de la localidad de memoria $8000 a la localidad $8009 con los datos del $30 al $39 que representan en código ASCII a los números decimales del 0 al 9 respectivamente, nótense las comas y la instrucción **FCB**.
- De la localidad de memoria $800A a la localidad $8010 se han almacenado los códigos en ASCII que representan las letras **MENSAGE**, nótense las comillas y la instrucción **FDB**.

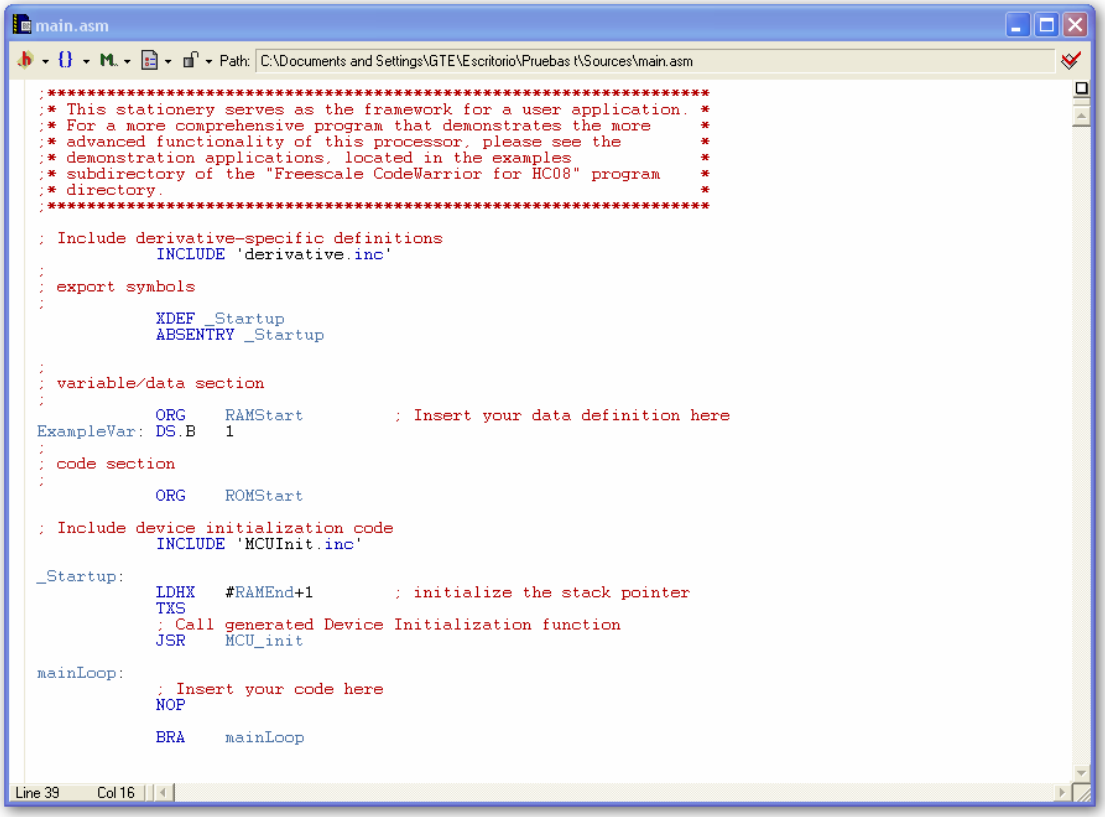


Figura 1.119: Archivo main.asm.

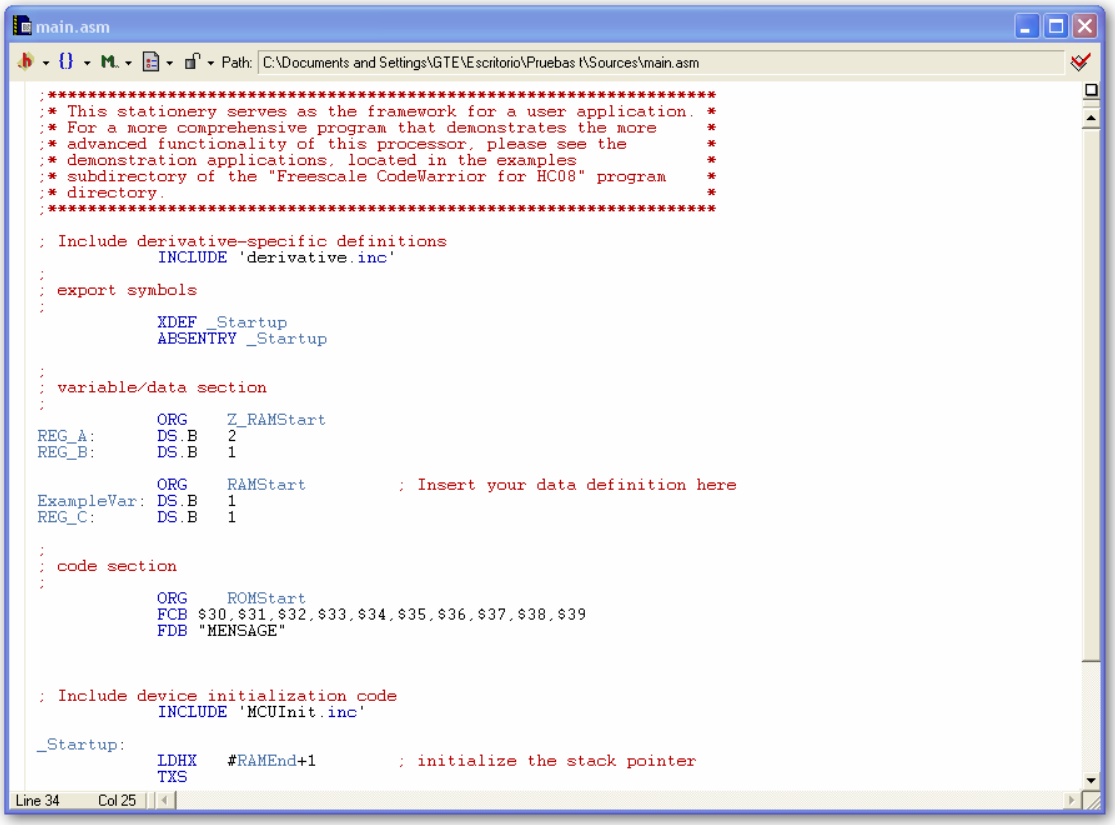


Figura 1.120: Creación de diferentes registros.

1.9.2 ¿DONDE SE ESCRIBE EL PROGRAMA PRINCIPAL?

Para saber donde se escribe el programa principal observe la figura 1.119 en la parte inferior donde puede observarse:

```
mainLoop:
            ;Inserte su código aquí
            NOP

            BRA     mainLoop
```

Básicamente el programa principal puede escribirse entre la etiqueta **mainLoop** y la instrucción **BRA mainLoop**, la instrucción NOP puede borrarse con toda libertad, si el programa principal se extiende mas de 128 bytes la instrucción **BRA mainLoop** no funcionará debido a lo ya visto en el direccionamiento relativo, para solucionar el problema solo cambie **BRA** por **JMP.**

1.9.3 ¿DONDE SE ESCRIBEN LAS SUBRUTINAS?

Las subrutinas deben ser escritas debajo de la instrucción **BRA mainLoop** o **JMP mainLoop** es decir después del programa principal.

1.9.4 ¿DONDE SE ESCRIBEN LAS INTERRUPCIONES?

Cuando una interrupción a sido activada e incluida en el código usted escribirá su interrupción (se verá en un capítulo posterior) en el archivo **MCUinit.inc** el cual esta dentro de la carpeta **Generated code** que puede verse en la figura 1.117.
La figura 1.121 muestra parte de un archivo **MCUinit.inc** donde pueden observarse las interrupciones de Timer 1 y del convertidor analógico digital activadas.
Escriba su código de interrupción para el convertidor analógico digital entre la etiqueta **isrINT_ADC** y la instrucción RTI.

```
isrINT_ADC:
        ; Escriba su código de ininterrupción aquí…

            RTI
```

Escriba su código de interrupción para el Timer 1 entre la etiqueta **isrINT_TIM1Ovr** y la instrucción RTI.

```
isrINT_TIM1Ovr:
        ; Escriba su código de interrupción aquí…

            RTI
```

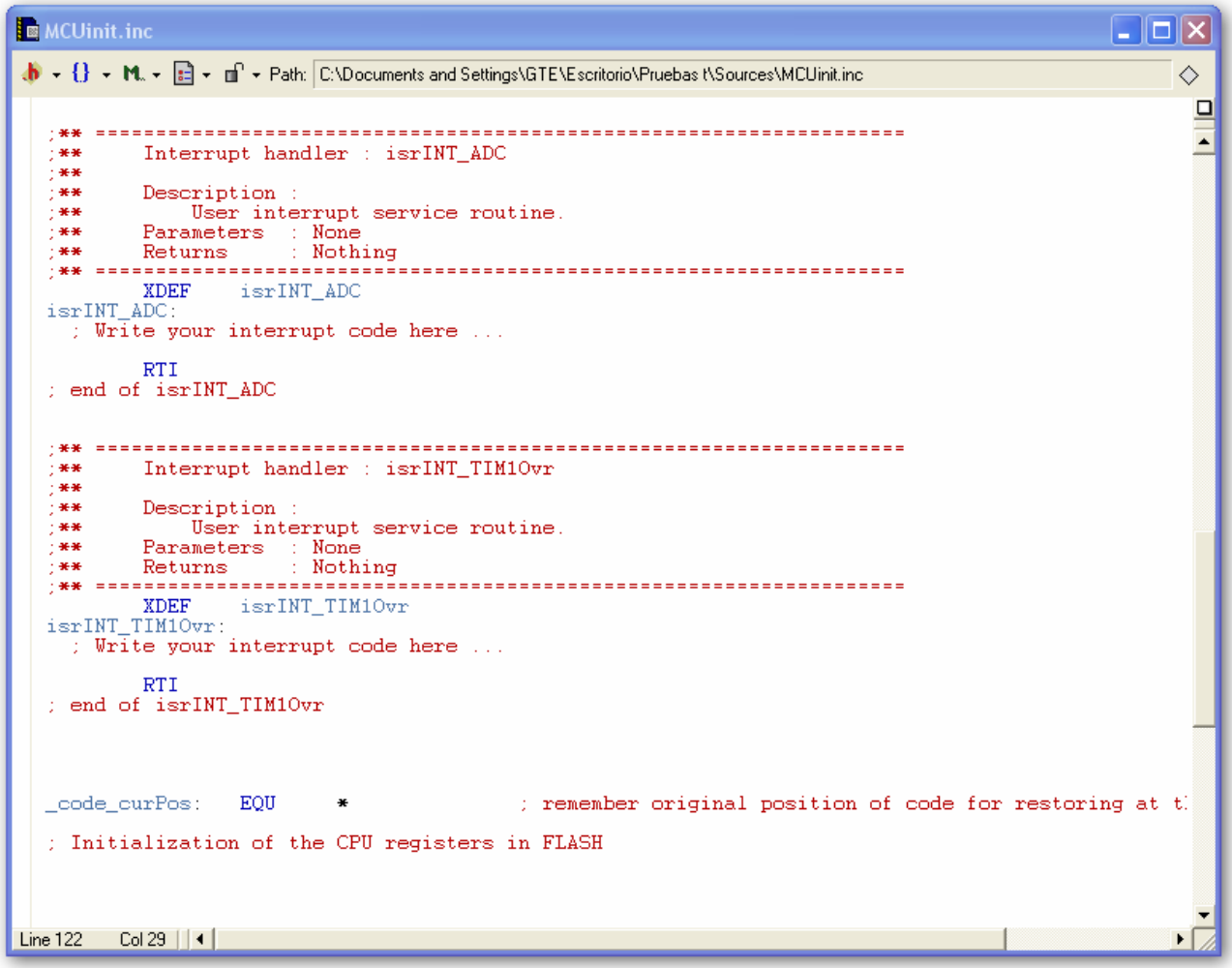


Figura 1.121: Archivo MCUinit.inc.

1.10 RESUMEN----------

Los MCU de Freescale son dispositivos de muy bajo costo y alto rendimiento con alta inmunidad al ruido, además existe una gran diversidad de ellos para cumplir con los gustos y necesidades que presenten los usuarios en cada uno de sus diseños.
Podemos resumir que los MCU Freescale cuentan con toda la arquitectura de un sencillo pero completo computador.
Los MCU disminuyen el numero de componentes a utilizar por lo que reducen el costo del sistema enormemente
Los MCU cuentan con una gran variedad de componentes como lo son la CPU, líneas de entrada y salida para comunicarse o controlar otros dispositivos, memoria ROM para guardar el programa o datos no volátiles, memoria RAM para el almacenamiento de datos temporales, módulos de comunicación serial, ADC, Timer, COP, teclado de interrupciones etc.
Los MCU pueden ser utilizados en una gran variedad de aplicaciones como lo son la robótica, la automatización industrial, el control de procesos, los electrodomésticos, sistemas de alarma, de instrumentación etc.
Estos MCU cuentan con un conjunto de instrucciones complejo que al conocer todas las instrucciones se facilita enormemente la programación a diferencia de los microcontroladores de conjunto de instrucciones reducido.
Gran parte de la habilidad del programador reside en como accede a memoria y manipula los datos es decir que tan hábil es decidiendo que instrucción y que modo de direccionamiento utilizar en cada situación que se le presente para así diseñar programas funcionales y eficientes. Para lograr esto es necesario conocer el potencial que puede entregar cada instrucción y modo de direccionamiento.

CAPÍTULO

2

HERRAMIENTAS DE DESARROLLO. "CODEWARRIOR" Y "MON08"

OBJETIVOS

Después de estudiar este capítulo usted podrá ser capaz de:

- ***Iniciar un nuevo proyecto con el software Codewarrior para microcontroladores de 8 bits en lenguaje ensamblador.***
- ***Usar el asistente iniciador de componente.***
- ***Describir cada uno de los módulos existentes en el iniciador de componente.***
- ***Manipular cada uno de los módulos y generar un código iniciador de componente.***
- ***Crear registros en memoria RAM, ROM y diversas formas de hacer tablas en memoria ROM.***
- ***Usar full chip simulation para la simulación del programa.***
- ***Usar MON08 para la grabación del programa en el MCU.***

Codewarrior de Freescale es uno de los ambientes de programación más amigables que existen, su interfaz simple y su modo intuitivo lo convierten en una herramienta apta para la introducción del estudiante, ingeniero o diseñador, en el mundo de los MCUs. También cuenta con poderosas herramientas llamadas "full chip simulation" y la depuración "in circuit" o MON08 que aceleran la depuración del programa de manera excepcional.

2.0 NUEVO PROYECTO

Existen versiones gratuitas del programa Codewarrior (el usado en este trabajo es: **Special Edition: CodeWarrior for Microcontrollers V6.2 Code Size Restriction: RS08/HC(S)08 32Kb; CFv1 - 64Kb**) y se pueden obtener descargándolo de la página oficial de Freescale (www.freescale.com) solo se necesita un registro previo, el programa esta contenido en un archivo de 350 MB y por ser una versión gratuita está limitado en la cantidad de líneas de código en lenguaje C es decir, si se desea escribir un programa muy grande para cualesquiera de los MCUs con suficiente memoria no será permitido. No existen limitaciones para el lenguaje ensamblador con el que se ha trabajado; si se escribe un

programa en ensamblador cuyo código ocupa el total de la memoria de un MCU por ejemplo el MC908GP32 no existirá ninguna limitación en el código del programa.
Para iniciar un proyecto nuevo en el Codewarrior teniendo en cuenta que se usará un entorno ***Windows*** debe abrirse el programa a través de un acceso directo que ejecuta el software de la siguiente ruta: ***Inicio _ Todos los programas_ Freescale Codewarrior _ CW for Microcontrollers V6.2 _ Codewarrior IDE,*** una vez hecho esto aparecerá la ventana mostrada en la figura 2.1, la parte de CW for Microcontrollers V6.2 de la ruta puede variar según la versión Codewarrior que se encuentre instalada.
Nótese que aparece un cuadro con cinco opciones que son:

- ***Create New Project (Crear Nuevo Proyecto),***
- ***Load Example Project (Cargar Proyecto de Ejemplo),***
- ***Load Previous Project (Cargar Proyecto Anterior),***
- ***Run Getting Started Tutorial (Ejecutar Tutorial de Inicio),***
- ***Starat Using CodeWarrior (Comenzar a Usar Codewarrior),***

Que indican su función por sí mismas, nosotros dedicaremos nuestra atención en la opción: ***crear un nuevo proyecto***, las opciones que más se utilizaran en el futuro serán ésta última y ***cargar proyecto anterior***.

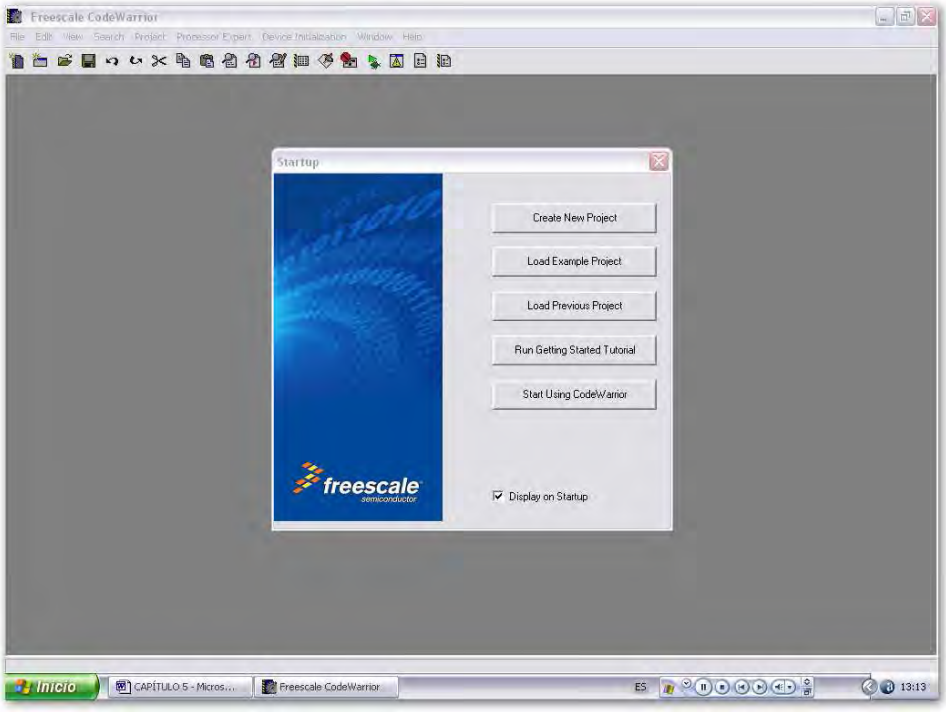


Figura 2.1: Ventana inicial del Codewarrior

Ya que se ha seleccionado la opción **“crear nuevo proyecto”** aparecerá una nueva ventana mostrada en la figura 2.2 aquí se debe seleccionar el tipo de arquitectura que se desea usar las cuales son: **HC08, HCS08, RS08, ColdFire V1, y Flexis,** es necesario desglosar el menú para encontrar el MCU que se empleará, en este caso se debe buscar el MCU MC908GP32,

este se encuentra en el grupo de MCUs pertenecientes a la arquitectura HC08 en la familia G. También se puede seleccionar el tipo de conexión que se pretenda usar como lo son ***Full Chip Simulation*** o ***MON08 intertace,*** los otros tres tipos de conexión no interesan ahora ya que son dispositivos de evaluación comerciales, mientras que MON08 puede ser construido fácilmente o simular el funcionamiento del MCU con la función **full chip simulation**, para empezar se selecciona **full chip simulation.**

Figura 2.2: Selección de componente y conexión.

Después de presionar el botón **siguiente** en el cuadro anterior, aparecerán las opciones correspondientes a **parámetros del proyecto** mostrado en la figura 2.3, aquí se puede seleccionar el tipo de lenguaje con el que se desea programar así como el nombre del proyecto y el directorio o ruta donde se guardará, en este caso el proyecto se escribirá en lenguaje ensamblador y será nombrado *Inicio_gp32;* es necesario deseleccionar la opción correspondiente a lenguaje **C** para seleccionar la opción ***Absolute Assembly*** que hace referencia al lenguaje ensamblador.

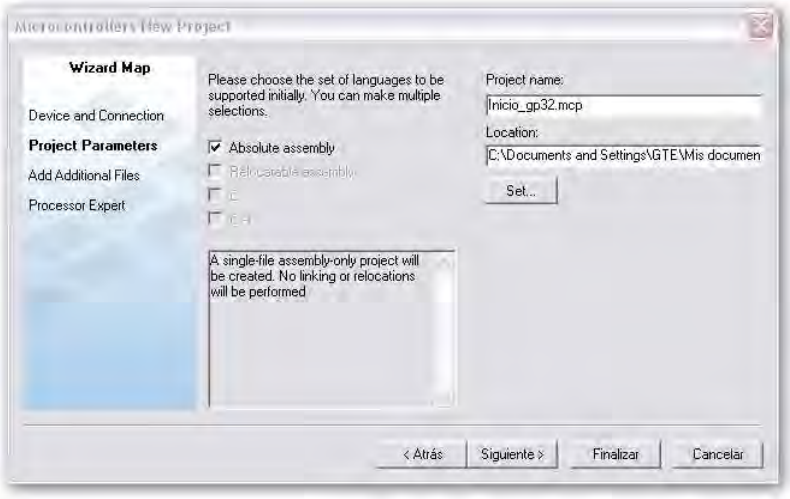


Figura 2.3: Selección de parámetros del proyecto.

Después de presionar el botón **Siguiente** aparecerá la ventana mostrada en la figura 2.4 con la cual se podrá adicionar al proyecto programas existentes o subrutinas ya escritas; el botón ***Add*** es para adjuntar el archivo seleccionado en el árbol izquierdo y el botón ***Remove*** es para remover el archivo seleccionado en el recuadro derecho.

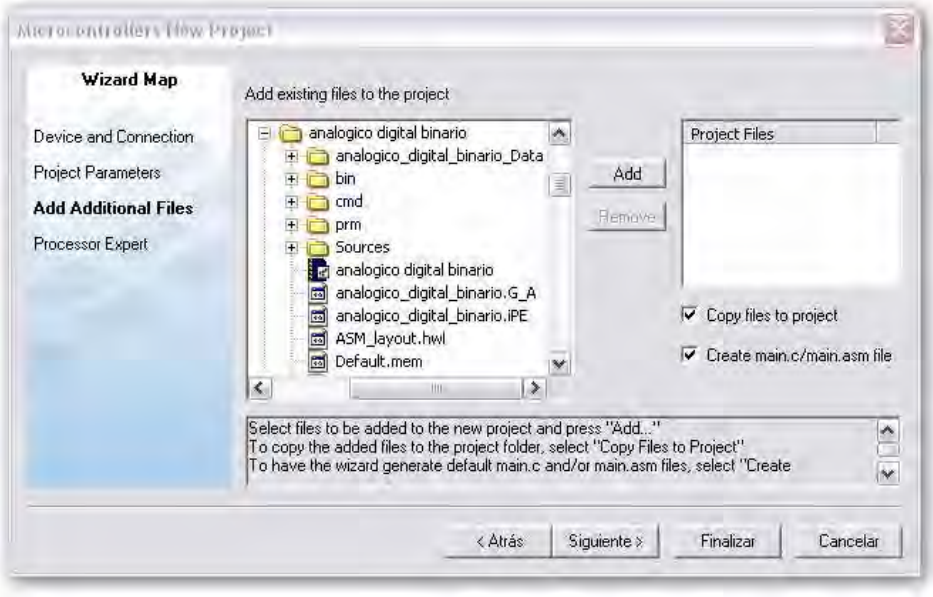


Figura 2.4: Adjuntar archivos adicionales

Es necesario presionar el botón **Siguiente** para que aparezca la ventana mostrada en la figura 2.5 donde se podrá seleccionar ***Device Initialization*** o inicializador de componente en español, esto activará automáticamente un subprograma que es parte de Codewarrior llamado ***Processor Expert*** es decir ***Device Initialization*** es una de las opciones de ***Processor Expert.*** Y así finaliza la creación de un nuevo proyecto con el asistente codewarrior.

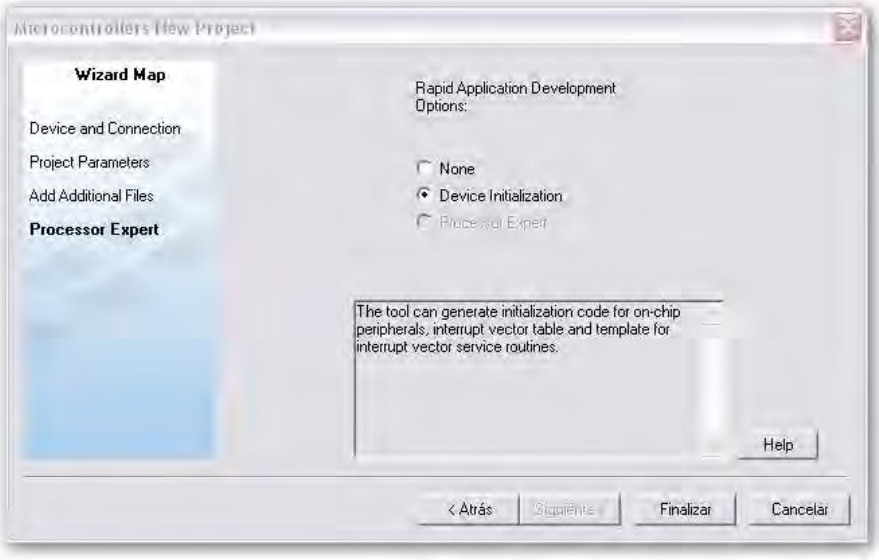


Figura 2.5: Activar Device Initialization

2.1 ENTORNO CODEWARRIOR ----------

El Codewarrior tiene una utilidad en la cual siempre que iniciemos un nuevo proyecto nos libra del engorroso proceso de establecer una estructura de programación, esto se menciona para hacer referencia que en la mayoría de los softwares de desarrollo de los diferentes fabricantes de MCUs incluido el antecesor al Codewarrior llamado WINIDE no cuentan con una herramienta similar.

El establecer una estructura de programa se refiere a la inclusión de bibliotecas especiales para la configuración de bits, registros propios del MCU a programar, como podrían ser: definir donde empieza la memoria ROM, indicar la dirección para el RESET, indicar donde se localiza la memoria RAM, nombrar registros debidamente es decir nombrar a la localidad de memoria $01 como PTB la cual funciona como un registro en el cual aparecen los datos presentes en el puerto B esto claro para adquisición de datos si estuviera configurado como entrada, o para control de dispositivos externos si está activado como salida.

La configuración de puertos es otro proceso que se evitará casi en su totalidad con el iniciador de componente es decir, establecer si funcionará como entrada de datos o salida y demás elementos como el COP, el convertidor analógico digital, el temporizador, la modulación por ancho de pulso, los puertos para la comunicación serial, interrupciones y demás opciones disponibles en el MCU seleccionado, sin embargo no se debe dejar de lado lo visto en capítulos anteriores y lo que se verá mas adelante referente a la función de cada registro y bits de propósito específico pertenecientes al MCU ya que en ocasiones y según lo exija el programa se tendrán que modificar.

Por ejemplo, si se inicializa el convertidor analógico digital a través del iniciador de componente de tal manera que realice una conversión con el valor obtenido en el canal uno donde se ha conectado un sensor de presión el canal uno quedará permanentemente configurado de esta forma pero, si a medida que avanza el programa se tiene la necesidad de cambiar el canal de la señal a convertir, para llevar acabo esto es necesario reiniciar el convertidor para poder realizar el cambio, esto se hace manipulando algunos de los bits del registro de control del convertidor analógico digital, lo mismo sucedería si se genera un PWM con un ciclo de trabajo del 50% y posteriormente es necesario modificarlo por otro que trabaje al 90%, esto solo por mencionar algunos ejemplos.

Ya inicializado un nuevo proyecto se abrirá la ventana mostrada en la figura 2.6 que será el entorno donde se trabajará la mayor parte del tiempo.

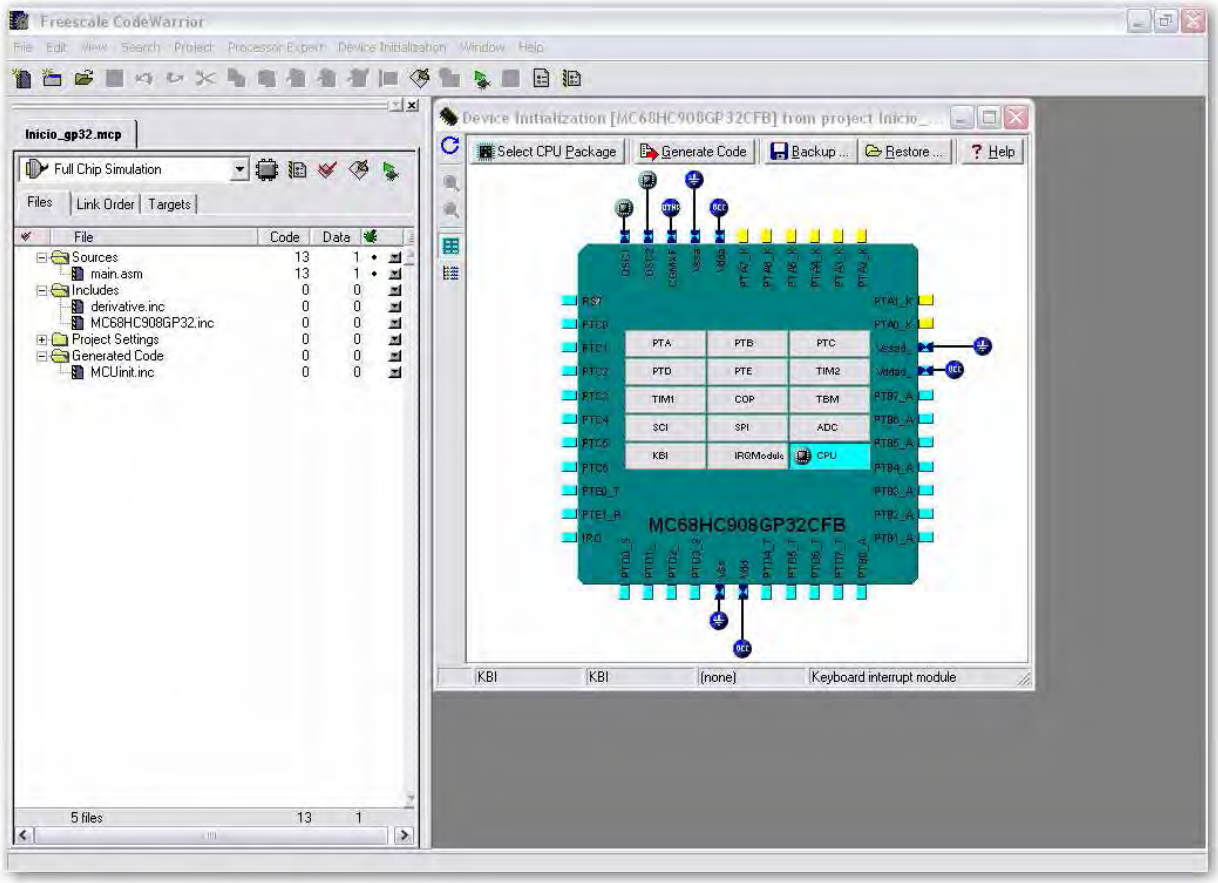


Figura 2.6: Entorno Codewarrior.

2.1.1 SELECCIÓN DE CONEXIÓN

Esta opción se encuentra en un pequeño recuadro en la parte superior izquierda de la ventana principal mostrada en la figura 2.7, sirve para cambiar la conexión con la que se trabajará. Las opciones que mas se utilizarán son ***Full Chip Simulation*** con la que podrá simular el funcionamiento del MCU y ***MON08 Interface*** con la cual se podrá grabar o emular el funcionamiento del MCU, aunque la emulación no es en tiempo real se podrá llevar al MCU por cada uno de sus estados acelerando así el tiempo de depuración del programa.

Figura 2.7: Selección de conexión.

2.1.2 MAKE Y DEBUG

Estas opciones se encuentran a la derecha del recuadro de selección de conexión mostradas en la figura 2.8. ***Make*** se utiliza para verificar que no existan errores de escritura en el programa, etiquetas repetidas, registros que aun no se han generado, modos de

direccionamiento erróneos, por ejemplo, utilizar una instrucción como "INC REGISTRO_A" la cual esta destinada para direcciones de memoria directas es decir si "REGISTRO_A" se encuentra en alguna localidad a partir de $0100 las cuales son localidades extendidas existirá un error en el modo de direccionamiento otro ejemplo, si utilizando el modo de direccionamiento relativo el lugar al cual debe desplazarse el contador de programa queda fuera de rango es decir con una desplazamiento mayor a -128 o +127 localidades de memoria con respecto a la ubicación en memoria de la instrucción ejecutada esto será detectado como error por lo se deberá usar un salto, esto son algunos de los errores que detecta Make, cabe mencionar que no detectará errores de programación como bucles infinitos, errores en la estructura del programa, etc. ***Debug*** hace exactamente lo mismo que ***Make*** solo que al terminar la revisión y si todo está correcto continuará con la conexión seleccionada en el recuadro de selección de conexión.

Figura 2.8: Make y Debug.

2.1.3 DOCUMENTOS DEL PROYECTO

La figura 2.9 muestra el cuadro ubicado en la parte izquierda de la ventana principal en el que se muestran todos los documentos del proyecto.

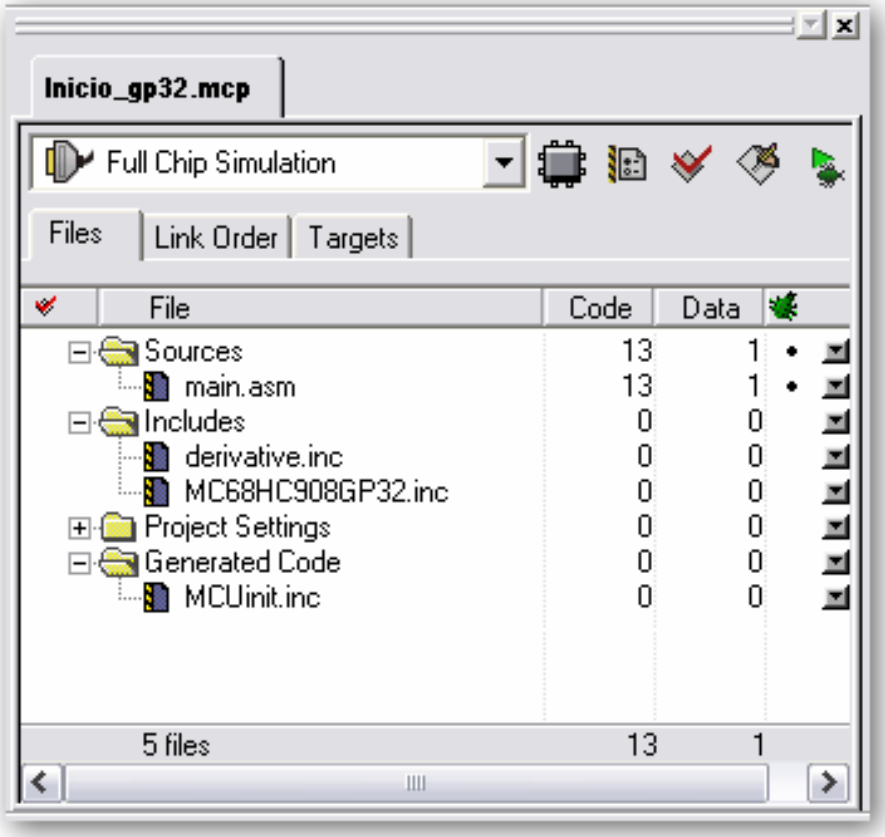


Figura 2.9: Documentos del proyecto.

2.1.4 DOCUMENTO MC68HC908GP32.inc

En el documento mostrado en la figura 2.10 ya se encuentran declarados todos los registros pertenecientes a este MCU, cada uno de los bits de esos registros, todos los vectores para cada una de las interrupciones y el mapeo de la memoria, tal cual se nombran en el manual del MCU.

Este documento es muy extenso así que solo se muestra la primera parte correspondiente al mapeo de la memoria, los vectores de cada interrupción y la declaración del puerto A.

NOTA: No intente modificar en nada este documento ya que cualquier cambio quedará guardado y permanecerá así cuando inicie un nuevo proyecto. En ocasiones se puede modificar tan solo el mapeo de memoria para generar páginas que puedan ser implementadas como una memoria EEPROM, que almacenará datos no volátiles cuando una aplicación así lo requiera..

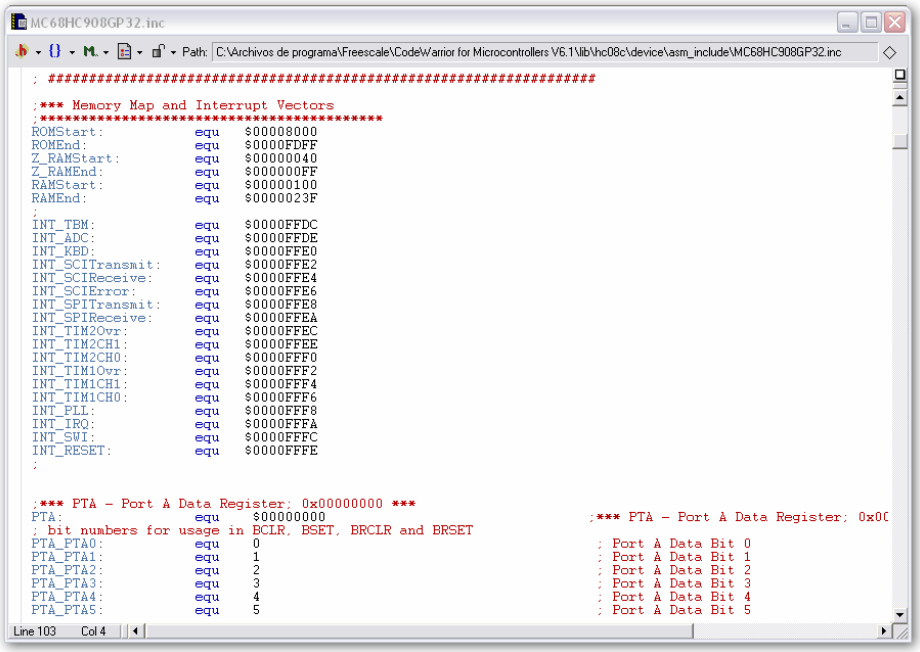


Figura 2.10: Documento MC68HC908GP32.inc.

2.1.5 DOCUMENTO MCUinit.inc

Todas las modificaciones que se realicen en el iniciador de dispositivo serán reflejadas en este archivo, aquí aparecerán espacios designados para cada una de las interrupciones si es que son activadas por el usuario, de lo contrario no aparecerán. MCUinit.inc a pesar de ser un documento aparte no es más que una subrutina ligada directamente al documento principal ***“main.asm”.***

En la figura 2.11 se muestra el documento ***MCUinit.inc*** con una interrupción activada (por que puede estar desactivada para después activarla moviendo un solo bit) perteneciente al temporizador 1, con el inicio del contador activado a 65535 ciclos maquina es decir cada vez que este número de ciclos transcurra se llamará a la subrutina de interrupción, el mismo documento le indica donde escribir la rutina lo cual se hace debajo del mensaje ***Write your interrupt code here*****,** en el documento puede observarse como se configuran los cuatro bits más significativo del PTA como salida poniendo a PTA7 y PTA6 en un estado alto mientras que PTA5 y PTA4 los coloca en un estado bajo además los 4 bits menos significativos de este mismo puerto los asigna como entrada pero a los puerto PTA3 y PTA2 se les activa la resistencia de pull up mientras que al PTA1 y PTA0 no.
El orden del documento es el siguiente:

- Aparece la etiqueta que llama a la subrutina de inicialización desde el programa principal esta es:

```
MCU  init:
```

- Este movimiento apaga el COP y habilita el reset por voltaje menor a 3v .

```
MOV    #$01,CONFIG1
```

- Deshabilita el oscilador mientras el MCU este en modo STOP.

```
CLR    CONFIG2
```

- Las siguientes instrucciones se encargan de generar los estados altos y bajos de los puertos PTA7:PTA4.

```
LDA    PTA
AND    #$CF
ORA    #$C0
STA    PTA
```

- Estas instrucciones activan la resistencia de pull up de los puertos PTA3 y PTA2.

```
LDA    PTAPUE
ORA    #$0C
STA    PTAPUE
```

- Esta se encarga de activar a los puertos como entrada y salida.

```
MOV     #$F0,DDRA
```

- Estas últimas activan la interrupción de temporizador tal y como fue mencionado anteriormente.

```
MOV     #$30,T1SC
LDHX    #$FFFF
STHX    T1MOD
LDA     T1SC
MOV     #$40,T1SC
LDA     #$FF
STA     COPCTL
CLI
RTS
```

- Esta sección es la correspondiente a la subrutina de interrupción.

```
;** ===================================================================
;**     Interrupt handler : isrINT_TIM1Ovr
;**
;**     Description :
;**         User interrupt service routine.
;**     Parameters  : None
;**     Returns     : Nothing
;** ===================================================================
       XDEF    isrINT_TIM1Ovr
isrINT_TIM1Ovr:
  ; Write your interrupt code here ...

       RTI
; end of isrINT_TIM1Ovr
```

También se encuentran los vectores que determinan que tipo de interrupción se ha presentado, (éstos no aparecen en la figura).

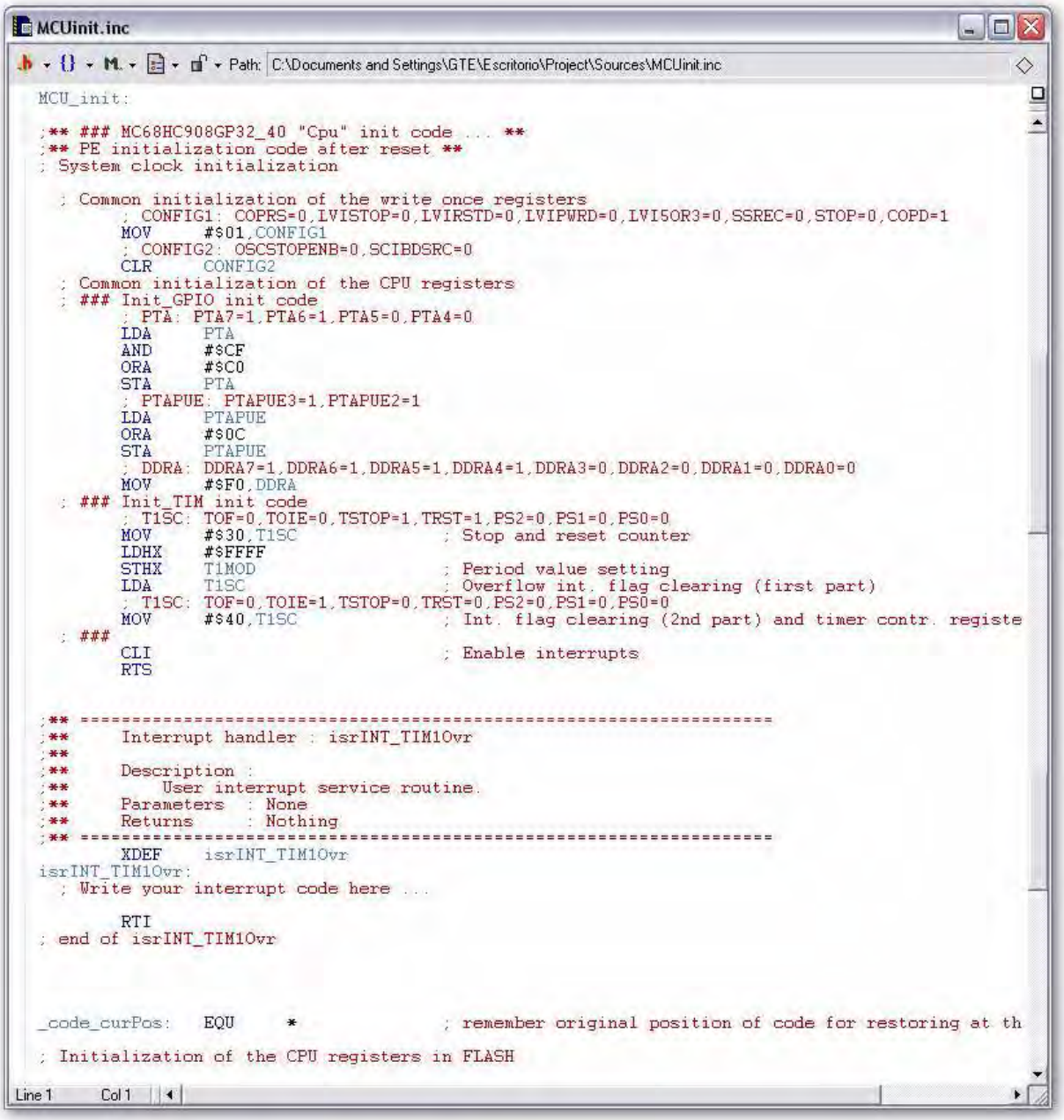


Figura 2.11: Documento MCUinit.inc.

2.1.6 DOCUMENTO MAIN.ASM

En el archivo main.asm se escribe el código del programa, al dar doble clic sobre el aparecerá la ventana ilustrada en la figura 2.13.

- Para crear registros en la memoria Z_RAM debemos escribir lo siguiente justo como se ve en la figura 2.13.

```
                ORG   Z_RAMStart
Registro_A:     DS.B  1
```

- El documento main.asm entrega por defecto un ejemplo de cómo generar un registro llamado ExampleVar, se a agregado un registro llamado Registro_B, nótese que al final de la línea hay un número dos mientras que en los otros registros hay un número 1 esto representa el tamaño en Bytes que tendrá el registro; en la memoria Z_RAM también pueden crearse registros de dos Bytes.

```
            ORG   RAMStart
ExampleVar: DS.B  1
Registro_B: DS.B  2
```

- Para generar tablas que contengan datos que serán ocupados durante el programa solo se deve escribir debajo de la instrucción **ORG ROMStart** justo como se muestra en la figura 2.13.
- TABLA: En este caso el codewarrior asignará en la primera localidad de la memoria ROM el dato correspondiente a “D” solo que en código ASCII y así sucesivamente para las otras letras, esto se usa especialmente cuando se trabaja con LCD´s los cuales responden al código ASCII, esto debe implementarse en mensajes que sean utilizados constantemente por el programa. En lugar de la instrucción FCB puede usarse FDB pero solo para este tipo de tabla.
- TABLA1: En este caso el codewarrior grabará los datos tal cual estén escritos a partir de la localidad siguiente de donde terminó de escribirse la tabla anterior, esto se usa principalmente cuando se programan displays de 7 segmentos, teclados matriciales, una matriz de led´s o LCD´s solo que en este último caso no colocamos la letra o el número sino directamente su equivalente en ASCII esto se debe a que no siempre se usan mensajes definidos, si no que podría ser un número que varíe constantemente, como ejemplo se hablará de desplegar el valor de temperatura de un sensor con rango de -25 a 120 grados centígrados esto se hace para ahorrar código ya que si usamos el método de la tabla anterior tendríamos que escribir 146 mensajes diferentes, es decir 146 tablas.
- TABLA2: En este último caso se escribirán los datos a partir de donde termina la tabla anterior; los datos tienen un tamaño de dos Bytes. Para una mayor comprensión observe la figura 2.12 que muestra como queda grabada la memoria; observe bien, los datos posteriores al dato #$6F42 ubicado en la localidad de memoria $801C pertenecen a las instrucciones que serán vistas en el punto siguiente, exactamente en la localidad $801D empieza a escribirse lo referente a **INCLUDE 'MCUInit.inc'.**

```
            ORG   ROMStart
TABLA:      FCB   "Datos Para LCD"
TABLA1:     FCB   $AC,$3E,$20,$25,$1F,$75,$36
TABLA2:     DCW   $9F2B,$25CD,$2530,$6F42
```

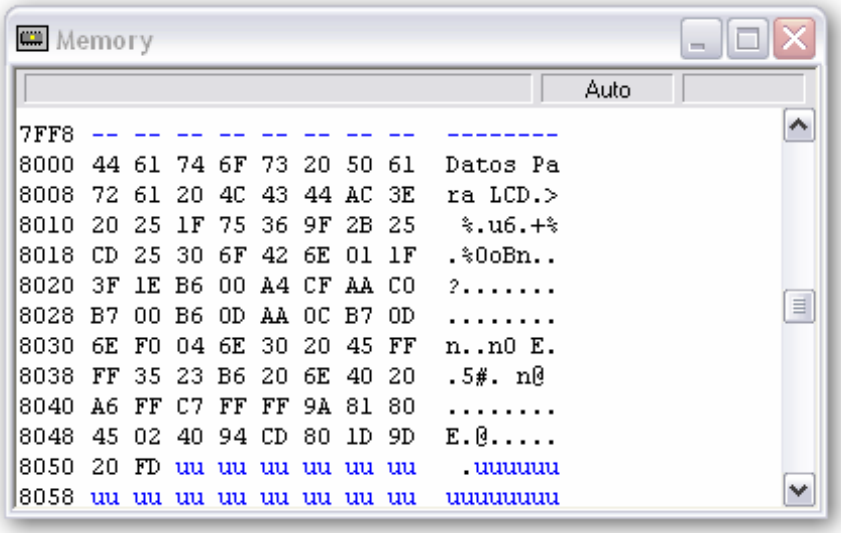


Figura 2.12: Estado de la memoria ROM.

- La siguiente parte del documento inicializa el **Stack Pointer** y brinca a la subrutina de inicialización que ya se ha visto en el punto anterior, ahora solo se debe introducir el código debajo de la instrucción **NOP**, puede eliminase la etiqueta **mainLoop** y la instrucción **BRA mainLoop** si no es necesario regresar al inicio del programa, tampoco es necesario conservar la instrucción **NOP** por lo cual puede borrarse, **Insert your code here** es solo un mensaje, así que se puede escribir el código a partir de **JSR MCU_init.**

```
INCLUDE 'MCUInit.inc'

_Startup:
        LDHX   #RAMEnd+1       ; initialize the stack pointer
        TXS
        ; Call generated Device Initialization function
        JSR    MCU_init

mainLoop:
        ; Insert your code here
        NOP

        BRA    mainLoop
```

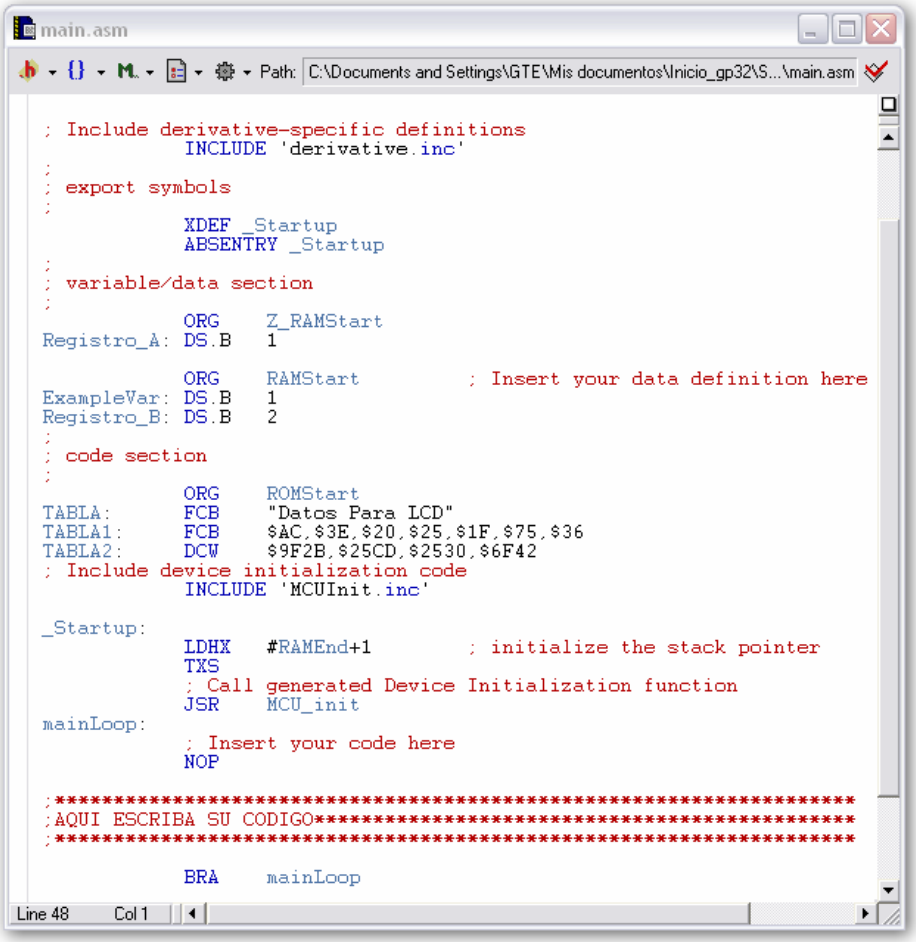


Figura 2.13: Documento main.asm.

2.2 INICIADOR DE COMPONENTE ----------

El Iniciador de Componente es una utilidad del **Processor Expert** un programa que trabaja como parte del Codewarrior el cual mediante una interfaz simple e intuitiva ayuda a inicializar puertos y modificar registros de control automáticamente para así no tener que modificar estos registros manualmente.
Lo primero a elegir en el iniciador de componente es el tipo de encapsulado a utilizar, la figura 2.14 muestra que se ha seleccionado el encapsulado PDIP de 40 pines (MC68HC908GP32CP) esto se hace con el botón ***Select CPU Package***, este botón también esta visible en la figura 2.14.

A continuación se muestran los módulos del MCU que se a usado hasta ahora, los módulos aquí mostrados pueden variar entre cada MCU, existen otros módulos disponibles para los HC08 pero solo están disponibles en otros modelos de MCUs.

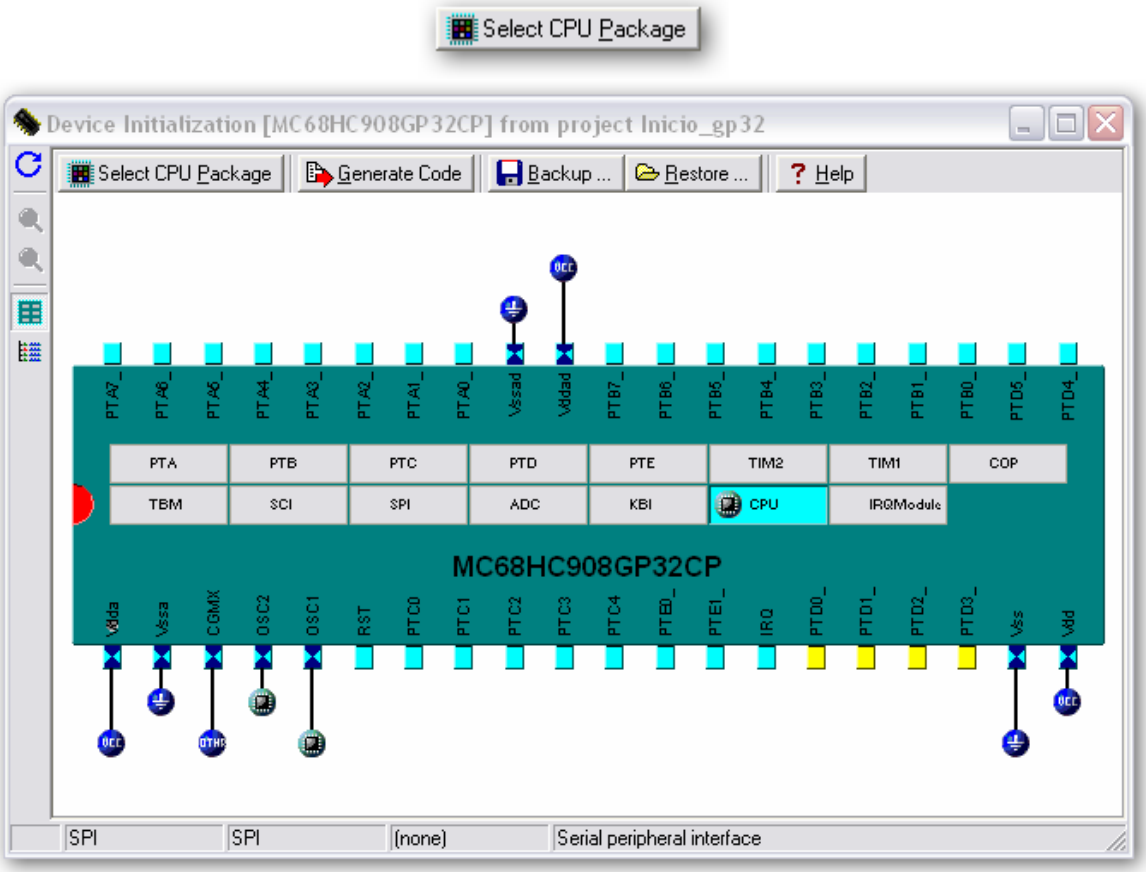


Figura 2.14: Device Initialization.

2.2.1. MÓDULO CPU

Esta sección permite declarar algunos parámetros de funcionamiento del MCU como son la velocidad del cristal, siendo esta solo informativa ya que al declarar este punto se evita el trabajo en otros módulos, un ejemplo de esto es que dentro del módulo del Timer aparece la duración del periodo del temporizador en segundos este cálculo nos lo evita el iniciador de componente y depende enteramente del valor del cristal, otros ejemplos son el modulo PWM y el convertidor analógico solo por mencionar algunos.

Para generar la frecuencia del convertidor analógico digital debe pasar la frecuencia del cristal por un preescalador (preescaler) por lo tanto se necesita indicar la frecuencia del cristal.

Además de establecer la frecuencia del cristal se puede indicar si el MCU funcionará en modo usuario o en modo monitor, por que, como se ha visto en el capítulo anterior, el modo usuario solo divide la frecuencia del cristal entre 4, mientras que en modo monitor se divide la frecuencia entre 2 si PTC3 esta puesto en un nivel lógico bajo y entre 4 si esta puesto a un nivel lógico alto, esto, para poder obtener la frecuencia de comunicación en **Monitor Rom,** esta frecuencia ira directo a los cálculos que realiza automáticamente el iniciador de componente para los módulos ya mencionados.

En la figura 2.15 se muestra el módulo correspondiente al CPU, a continuación se explican algunas de las opciones más usadas.

- **Clock Settings**: se puede hacer referencia al cristal que se empleará en el MCU escribiendo el valor exacto.
- **Stop instruction enabled**: si se habilita esta opción la instrucción STOP queda habilitada abriendo dos opciones más.
- Oscilador habilitado en el modo STOP.
- Número de ciclos necesarios para recuperarse del modo STOP.
- **CPU mode selection**: selecciona el modo monitor rom o el modo usuario.
- **PTC3 pin Level**: puesto a uno la frecuencia del cristal se divide entre cuatro, puesto a cero se divide entre dos (opción no disponible en **User mode**).
- **LVI module**: habilita o deshabilita el monitoreo del voltaje presente en el pin Vdd.
- **LVI reset**: deshabilita o habilita el reset por nivel de voltaje alto.
- **LVI Operating mode**: cambia el nivel de voltaje de reset. Nivel de voltaje bajo para 5v y 3v.
- **LVI in stop mode**: Habilita o deshabilita el reset por nivel bajo de voltaje en el modo stop.
- **ISR name**: No modifique la etiqueta de dirección para el vector de interrupción por reset.

Nota: Para un trabajo más sencillo se debe colocar el modo de la CPU en **User mode** y la velocidad del cristal a utilizar en MHz, no confundir con el cristal de grabación de 4.9152 MHz el cual se discute más adelante.

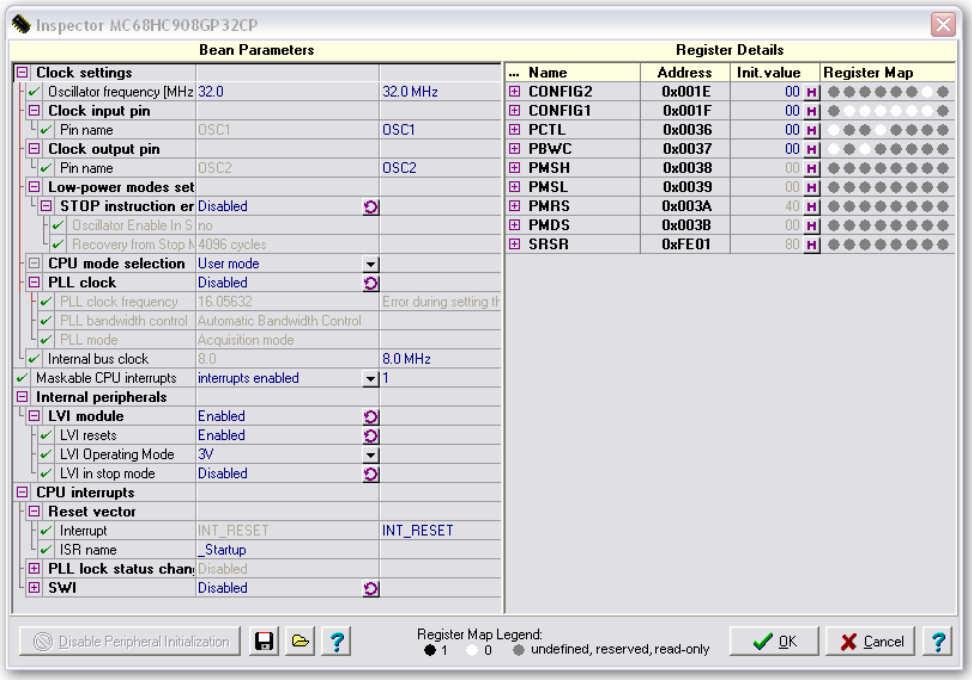


Figura 2.15: Módulo CPU.

2.2.2 MÓDULO IRQ

En este módulo se puede configurar el pin IRQ, el cual tiene muy pocas opciones a modificar; este MCU cuenta con solo 1 pin IRQ, el módulo se muestra en la figura 2.16.

- **PIN IRQ1**.
- **Pull resistor**: habilita o deshabilita la resistencia de pull up. La resistencia de pull up para este pin en este MCU no se puede deshabilitar.
- **Triggering sensitiviti**: activa la interrupción por flanco de bajada o por flanco de bajada y nivel bajo.
- **IRQ interrupt**: habilita o deshabilita la interrupción por IRQ.
- **ISR name**: no modifique este parámetro ya que es la etiqueta del vector que llama a la interrupción

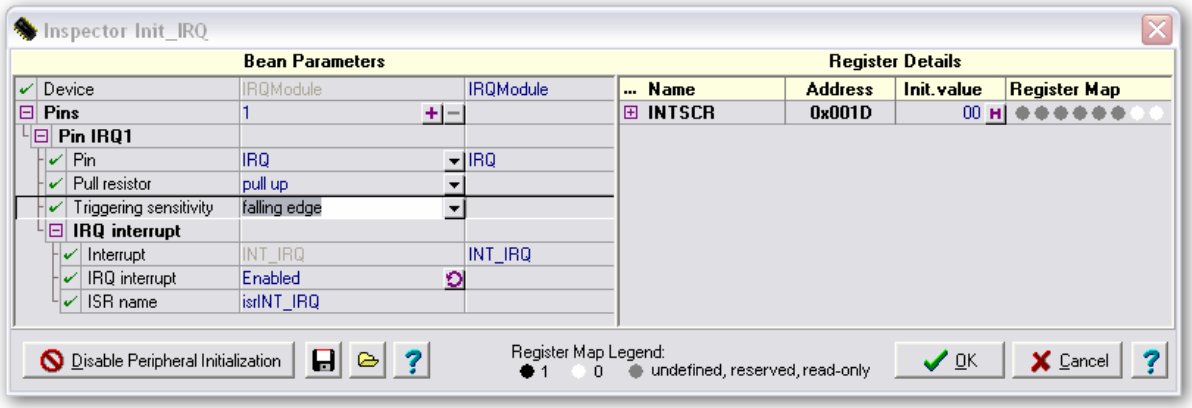


Figura 2.16: MóduloIRQ.

2.2.3. MÓDULO COP

En este módulo solo se puede habilitar o deshabilitar el COP o Wachdog, además de poder modificar el periodo time-out entre sus dos posibles opciones tal y como se muestra en la figura 2.17.

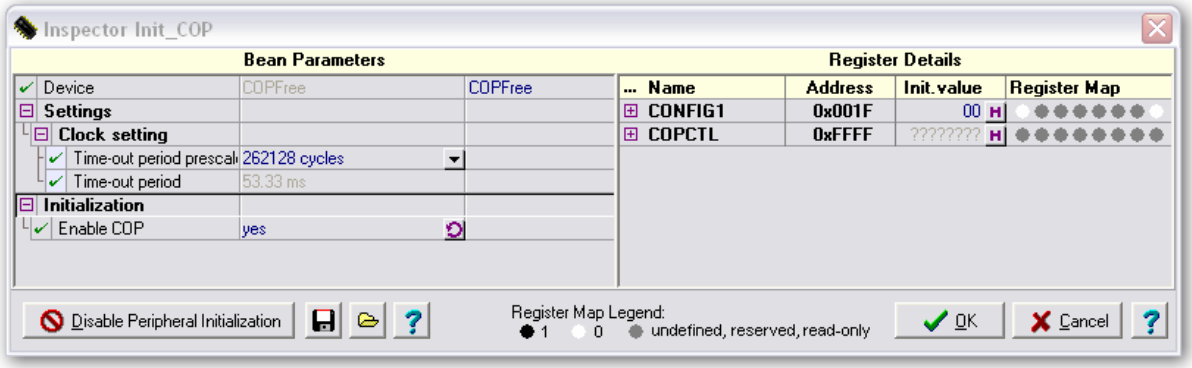


Figura 2.17: MóduloCOP.

2.2.4 MÓDULO TIMER 1 y 2

Para la manipulación de este módulo es preferible que se estudie a fondo la sección del Timer en el manual del MCU empleado para poder manejar apropiadamente las opciones input compare, output compare y PWM.
La figura 2.18 muestra un ejemplo de cómo configurar el Timer en el modo que usted mas utilizará. La interrupción por Timer se usa especialmente en sistemas de control temporizados por ejemplo un simple reloj.
Obsérvese que al poner el preescaler a 32 la interrupción se ejecutará cada segundo esto se debe a que en el módulo CPU e declarado un cristal de 8.0 MHz este al ser dividido entre cuatro provoca un ciclo maquina de 2.0 MHz ahora si por cada 32 ciclos maquina incremento en uno el módulo contador se obtiene que:

1/Bus = Tiempo de ejecución de un ciclo……1
1/2MHz = 500ns

preescaler (módulo contador) = número de ciclos……..2
32 (62500) = 2000000 ciclos

número de ciclos (tiempo de ejecución) = periodo del timer……..3
2000000 (500ns) = 1.0s

Sustituyendo 1 y 2 en 3 se obtiene 4 para después despejar “módulo contador” y obtener 5

periodo del timer = {preescaler (módulo contador)}(1/Bus)….4

$$\text{Módulo contador} = \frac{\text{Periodo del timer\_(Bus)}}{\text{Preescaler}} \quad \ldots.5$$

Se debe seleccionar el periodo del Timer requerido y escoger el preescaler adecuado, si el resultado de “módulo contador” es mayor a 65535 o $FFFF se debe seleccionar un preescaler mayor, si se llega a usar el preescaler 64 y a un el resultado es mayor a 65535 se debe dividir el tiempo de la interrupción deseada en dos o más partes según sea el caso y contabilizar la sucesión de interrupciones asta alcanzar el tiempo deseado a través de una rutina con esto se quiere decir que si se requiere de una interrupción cada dos segundos la solución sería generar una interrupción con un periodo de un segundo y a su vez cada que se contabilicen dos interrupciones ejecutar lo que se requiera según sea el caso así se obtendrían los dos segundos.

- **PREESCALER**: Indica el número de ciclos máquina que deben transcurrir para incrementar en uno al registro contador del **Timer** de 16 bits (**T1CNTH:T1CNTL**).
- **Modulo counter**: registro de 16 bits que se usa como comparación, indica hasta que número podrá incrementarse el registro contador del **Timer**.

- **Period**: indica el período del Timer.
- **Overflow interrupt**: Habilita la interrupción por desbordamiento es decir cuando el **modulo counter** llega a su valor máximo vuelve a cero lo que activa la interrupción por Timer.
- **ISR name**: no debe modificarse este parámetro ya que es la etiqueta del vector que llama a la interrupción.
- **Start counter**: si esta habilitado permite que el modulo counter incremente su valor cada que se completa el número de ciclos establecidos por el preescaler; es decir, si esta deshabilitado es como pausar el módulo counter; cambiando el estado del bit 5 del registro T1SC se puede detener o continuar la cuenta

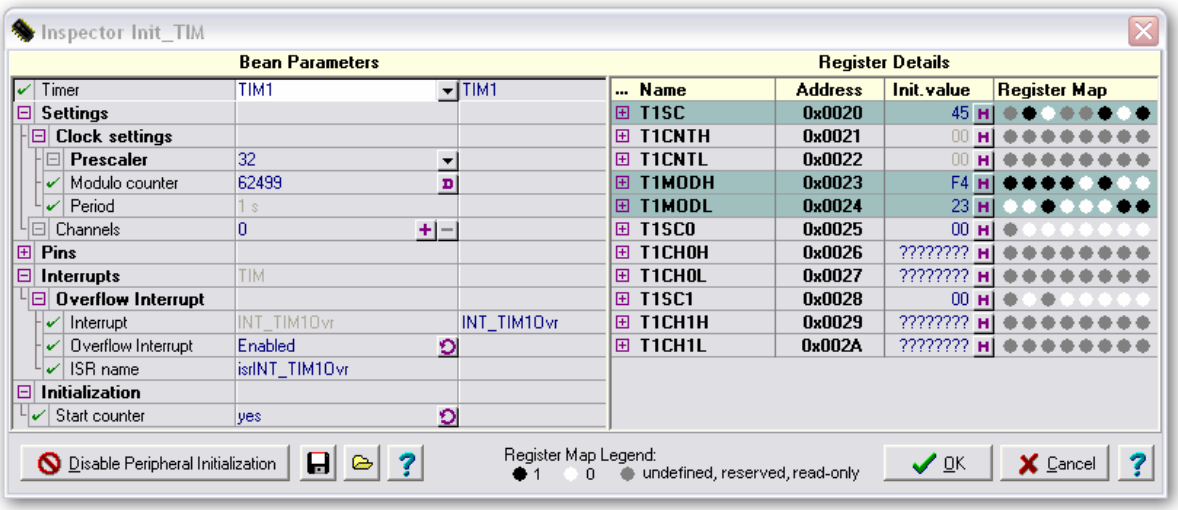


Figura 2.18: Módulo Timer 1 ejemplo de interrupción.

Nota: Con un cristal de 8MHz el mayor periodo de interrupción por Timer que puede obtenerse es de 2.097152s, esto se logra con el preescaler y el modulo counter al máximo, 64 y 65535 respectivamente.

Observaciones: Siendo que en los dos casos es correcto ¿Por que en la fórmula número 2 aparece 62500 dando como resultado un periodo de 1s, mientras que en la figura 5.17 aparece 62499 entregando también 1s? Esto se debe a que en el diseño de sistemas digitales el estado cero es un estado con valor es decir 62499 estados mas el estado cero son 62500 estados; al momento de realizar operaciones se usa el sistema decimal en el cual el cero no vale así que no debe olvidarse restar uno al resultado antes de colocarlo en el **"modulo counter"**.

2.2.5 MÓDULO KBI

En la figura 2.19 se muestra el módulo "teclado de interrupciones" con solo el pin 0 habilitado, este módulo cuenta con muy pocas opciones entre las cuales son:

- **Triggering Sensitiviti**: esta opción puede ser modificada entre edge o edge and level, que quiere decir: "borde" y "nivel y borde".

- **Pin0:Pin7**: Habilita como interrupción alguno o todos los pines del puerto A.
- **Keyboard request interrupt**: simplemente habilita o no las interrupciones por teclado.
- **ISR name**: no modifique este parámetro ya que es la etiqueta del vector que llama a la interrupción

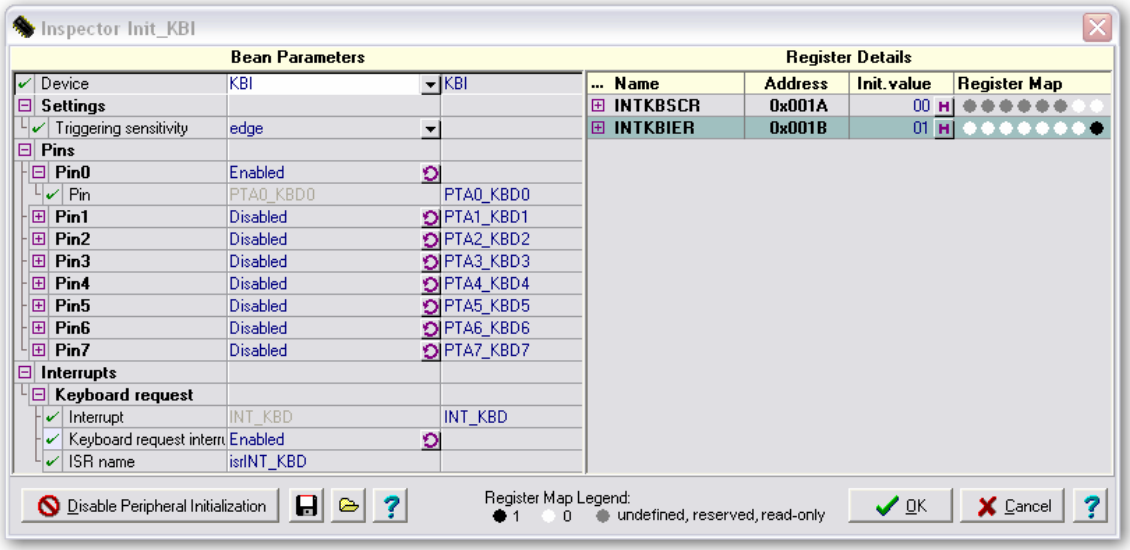


Figura 2.19: Módulo Key Borrad Interrupt

2.2.6 MÓDULO ADC

En la figura 2.20 se muestra el módulo correspondiente al convertidor analógico digital el cual es muy sencillo y con pocas opciones que manipular.
La configuración del módulo ADC en la figura es: 2 pines de entrada para el convertidor, el PTB0 y el PTB1 seleccionados, con la conversión continua activada, con el canal 0 de entrada para realizar conversiones y la interrupción por conversión completa desactivada.
Lan funciones más importantes de este módulo son:

- **Clock Source**: selecciona la fuente de reloj para el convertidor, de entre el reloj interno de Bus y CGMXCLK.
- **Preescaler**: La frecuencia del convertidor debe ser de entre 500KHz y 1048KHz por lo cual la velocidad del cristal debe ser dividida para el buen funcionamiento, los valores presentes en la figura dependen del cristal que halla seleccionado en el módulo CPU. Para seleccionar el valor del preescaler correcto solo varíe el valor de esta opción asta que el mensaje de advertencia desaparezca.
- **Conversión mode**: Esta opción es útil para decidir si se realizará una sola conversión o se realizaran conversiones una al terminar la otra y así sucesivamente, durante la ejecución del programa esta opción nos sirve para detener o echar a andar al convertidor.
- **ADC Input Pin**: esta opción indica cuantas entradas analógicas estarán habilitadas.
- **Input Pin0:7**: Esta opción indica que pines se utilizaran.

- **Conversión complete interrupt**: Habilita o no la opción de ininterrupción por conversión completa.
- **ISR name**: no modifique este parámetro ya que es la etiqueta del vector que llama a la interrupción.
- **Inicial Channel select**: esta opción determina sobre que canal empezará a realizar conversiones el convertidor, las opciones disponibles son: del canal 0 al canal 7, voltaje de referencia bajo, voltaje de referencia alto o simplemente el convertidor analógico digital apagado.

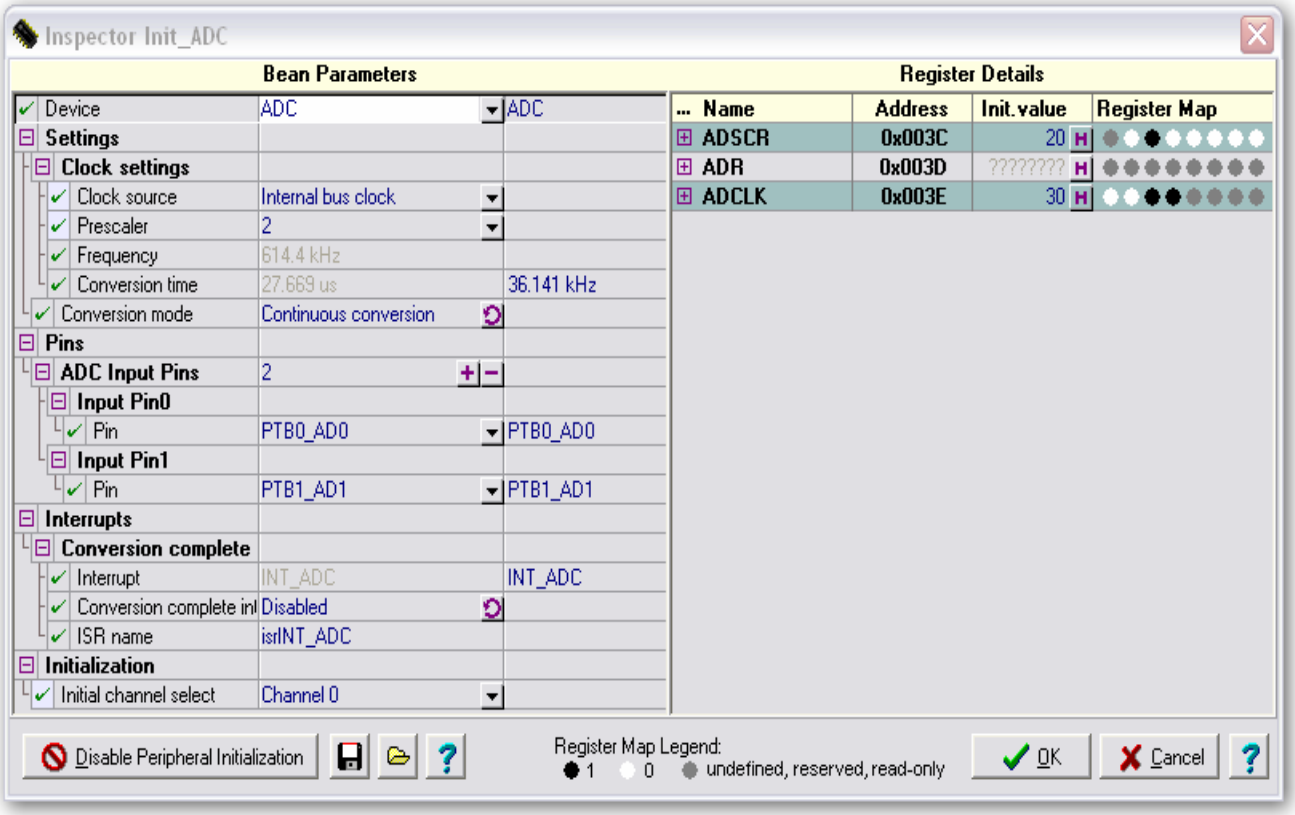


Figura 2.20: Módulo ADC

2.2.7 MÓDULO SPI

La figura 2.21 muestra el módulo SPI (**Serial Peripherial interface**), configurado en modo Maestro, con un peescaler de 2, con fase de reloj a primer flanco, la polaridad del reloj es positiva es decir activa en alto, las interrupciones están todas deshabilitadas, el modo mal funcionamiento esta desactivado, con las resistencias de pull up habilitadas, y por ultimo el sistema SPI se encuentra habilitado.

¿Qué se debe hacer para conectar un MCU maestro con uno esclavo? Para esto la configuración de un MCU en modo esclavo es similar a lo visto en el modo maestro solo que ahora se coloca **slave** en la opción **mode select**, además se deshabilita el sistema SPI con la última opción mostrada en la figura 2.21 esto se debe a que es necesario activar el SPI maestro antes que el SPI esclavo el problema es que al energizar los 2 MCU´s sus programas

se ejecutan al mismo tiempo, como ya mencionamos el SPI esclavo esta desactivado mientras que el maestro esta habilitado en los módulos de inicialización esto permanecerá así durante la ejecución del programa asta que se habilite el SPI esclavo mediante la instrucción **“BSET 1,SPCR”** después de un retardo (si fuera necesario) lo suficientemente grande en el programa principal del microcontrolador esclavo, esto activara el sistema SPI del MCU esclavo después del MCU maestro.
Alguna de las opciones que podrá modificar a través del iniciador de componente en el SPI son:

- **Mode Select**: elige entre modo Esclavo y modo Maestro.
- **Clock Polarity**: elige la polaridad del reloj, activo en alto y activo en bajo.
- **Clock Phase**: cambia entre primer flanco y segundo flanco.
- **Receive and Fault interrupt**: se activa la interrupción por fallo o la interrupción por recepción completa o ambas, nótese que si se activan las dos interrupciones se ejecutara la misma subrutina de interrupción ya que las dos interrupciones están direccionadas con el mismo vector.
- **Wire-OR-mode**: activa o desactiva las resistencias de pull up
- **Transmit interrupt**: habilita o deshabilita la Interrupción por transmisión completa.
- **ISR name**: no modifique este parámetro ya que es la etiqueta del vector que llama a la interrupción.

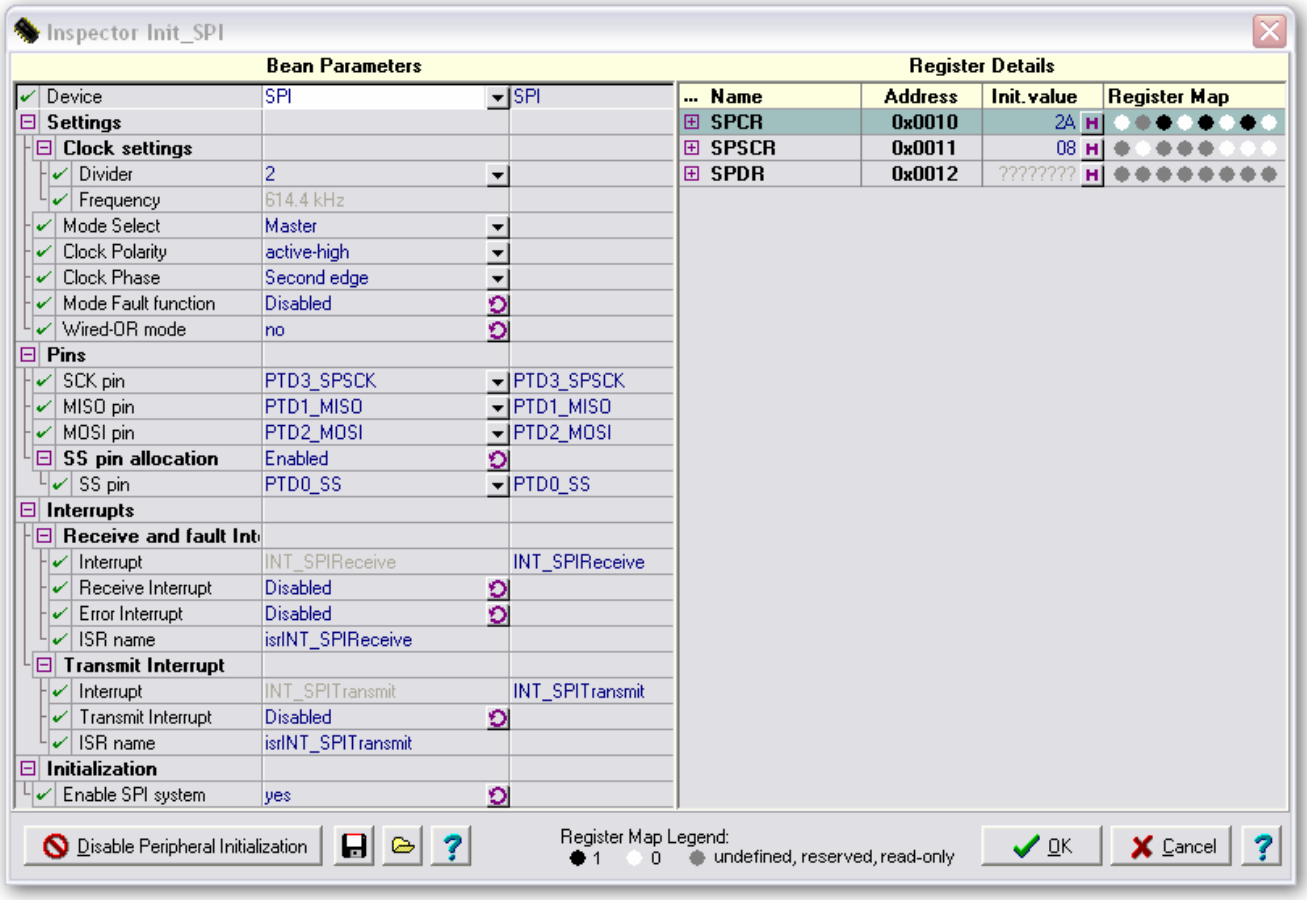


Figura 2.21: Módulo SPI

2.2.8 MÓDULO SCI

La figura 2.22 muestra el módulo SCI, este módulo cuenta con una gran cantidad de opciones a ser modificadas por lo cual se recomienda estudiar profundamente la sección correspondiente al SCI, además de profundizar conocimientos en el tema de comunicación asíncrona full-duplex.
En al figura se puede observar como esta habilitado el SCI, casi en su totalidad son las opciones que entrega por default el codewarrior ya que solo se ajustó la velocidad de comunicación con el preescalador y el divisor de baudios para obtener una velocidad estándar de 9600 baudios con la ayuda de un cristal de 4.9152MHz, el otro cambio realizado se encuentra en la sección Initialization al final de la figura que aunque no esta desplegado el contenido de esta opción el SPI se encuentra activado junto con el transmisor no obstante el receptor esta desactivado. Las opciones más importantes son:

- **Baud ClocK input**: selecciona el tipo de reloj a utilizar.
- **Baud rate preescaler**: divide la frecuencia entre números impares.
- **Baud rate divisor**: divide la frecuencia entre números pares.
- **Loop mode**: habilita o deshabilita el modo Loop
- **Data format**: define el tamaño de la palabra a transmitir de entre 8 y 9 bits.
- **Wake up**: modifica a despertar por línea ocupada o despertar por marca de dirección.
- **Idle carácter counting**: el SCI empieza a contar desde el bit de inicio o desde el bit de paro.
- **Parity**: inserta un bit de paridad y elige si será par o impar.
- **Send Break**: envía un carácter break seguido por un bit lógico 1 o envía caracteres break sin bit lógico 1 entre ellos.
- **Receiver Wake upmode**: pone al receptor en modo stand by.
- **Transmitter output**: invierte la polaridad de los datos a transmitir.
- **Interrupciones**: son 2 por transmisión, 2 por recepción y 4 debido a diferentes casos de error
- **ISR name**: no modifique este parámetro ya que es la etiqueta del vector que llama a la interrupción.

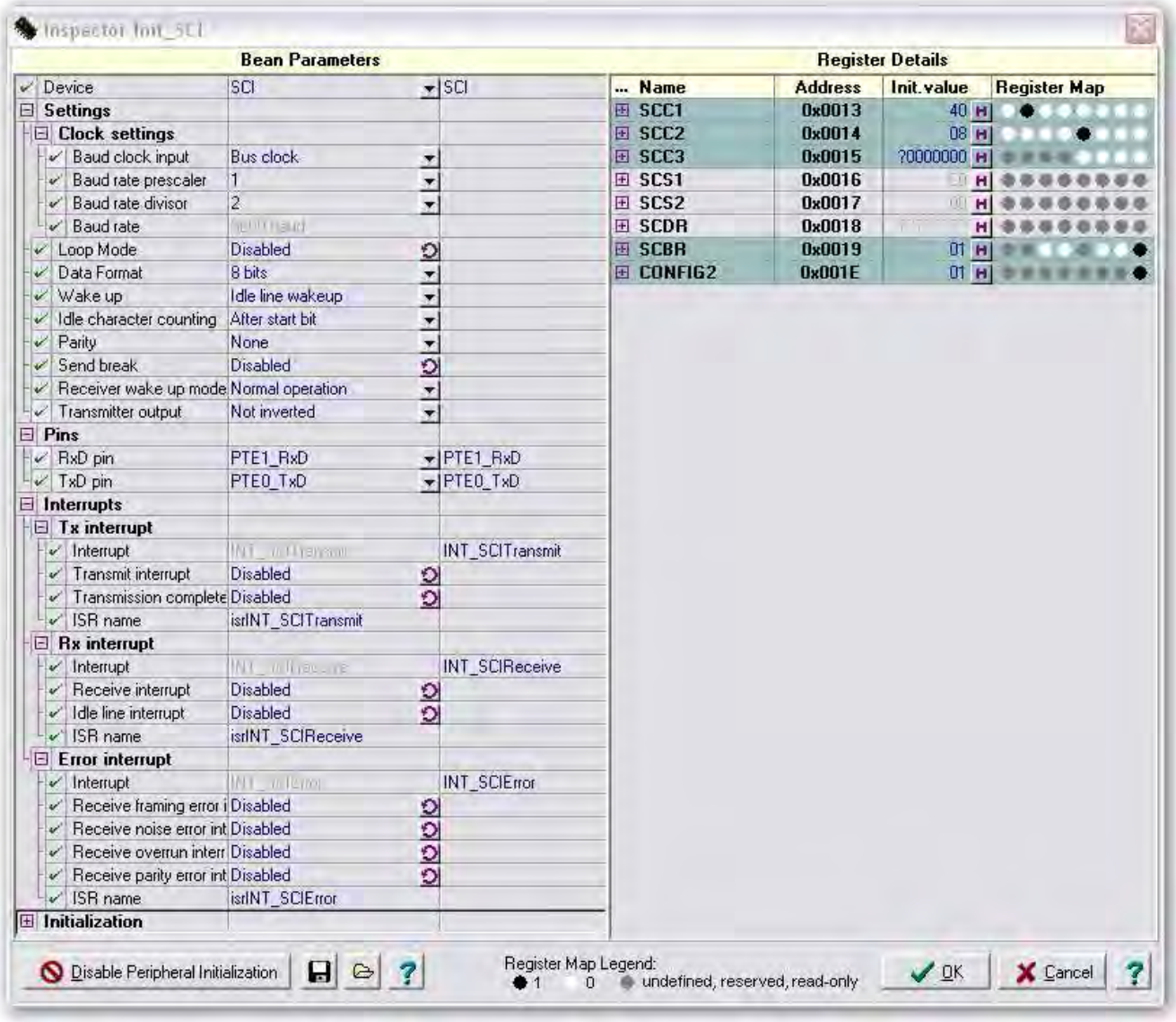


Figura 2.22: Módulo SCI

2.2.9 MÓDULO TBM

En la figura 2.23 puede verse el módulo TBM, con pocas funciones y fácil de usar, este módulo se encuentra en muy pocos microcontroladores HC08.
El módulo se encuentra configurado con su máximo preescalador, la interrupción se encuentra habilitada y por último el módulo TBM esta habilitado.
Sus tres únicas opciones son:

- **Preescaler**: divide la frecuencia del cristal.
- **Timebase Interrupt**: Habilita la interrupción por TBM.
- **Enable TBMmodule**: Habilita el módulo TBM.
- **ISR name**: no modifique este parámetro ya que es la etiqueta del vector que llama a la interrupción.

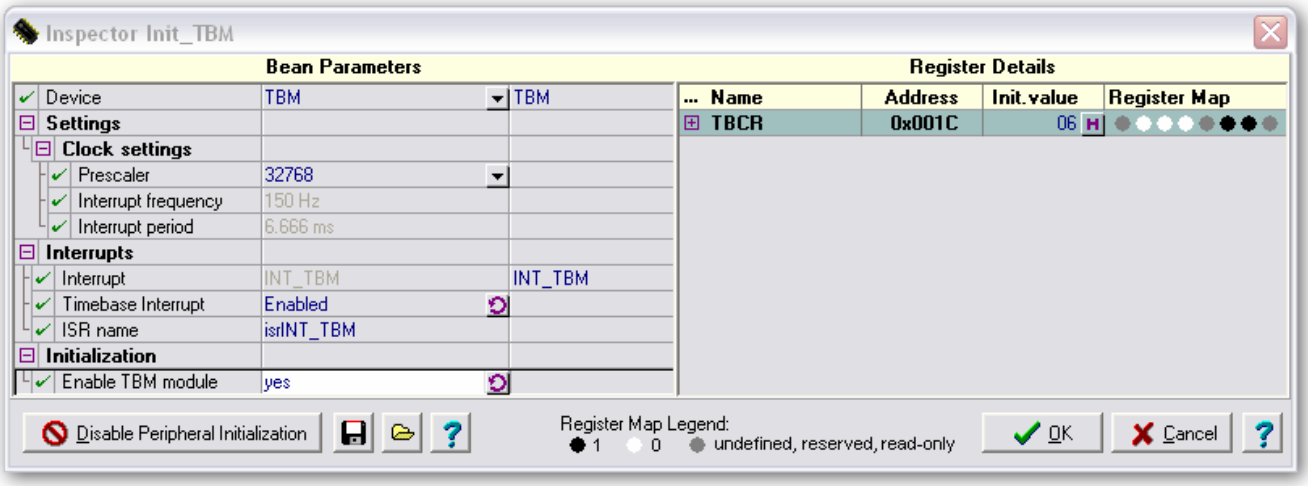


Figura 2.23: Módulo TBM

2.2.10 MÓDULO PTA, PTB, PTC, PTD, PTE

En la figura 2.24 puede verse el módulo correspondiente al puerto PTB, el cual se encuentra configurado todo como salida además de tener un valor inicial de $FF es decir todos los pines con un valor lógico alto no obstante si usted presiona el botón encerrado en el circulo rojo podrá establecer la configuración que desee pin por pin en cualquier combinación posible.
Es también posible configurar puertos y compartirlos con otros módulos es decir usted puede configurar la mitad del puerto A como puerto de propósito general de entrada o salida de datos y además puede utilizar la otra mitad del puerto como teclado de interrupciones pero tenga cuidado de no repetir pines en un módulo y en otro es decir si está ocupando la comunicación serial la cual se implementa en el puerto E no podrá utilizar estos pines como entradas o salidas de propósito general

Las opciones que podrá manipular en este módulo son:

- **Direction**: Cambia el puerto elegido entre entrada y salida

- **Output value**: Agrega un valor de inicio siempre y cuando este como salida ya que si se encuentra como entrada no importa el valor que agregues el puerto tomará el valor que la circuitería exterior le proporcione.

- **Pull resistor**: agrega una resistencia de pull up siempre y cuando el pin este configurado como entrada ya que si esta como salida la resistencia no sirve de nada puesto que el registro del puerto define el valor de salida y en ningún momento la resistencia de pull up lo altera.

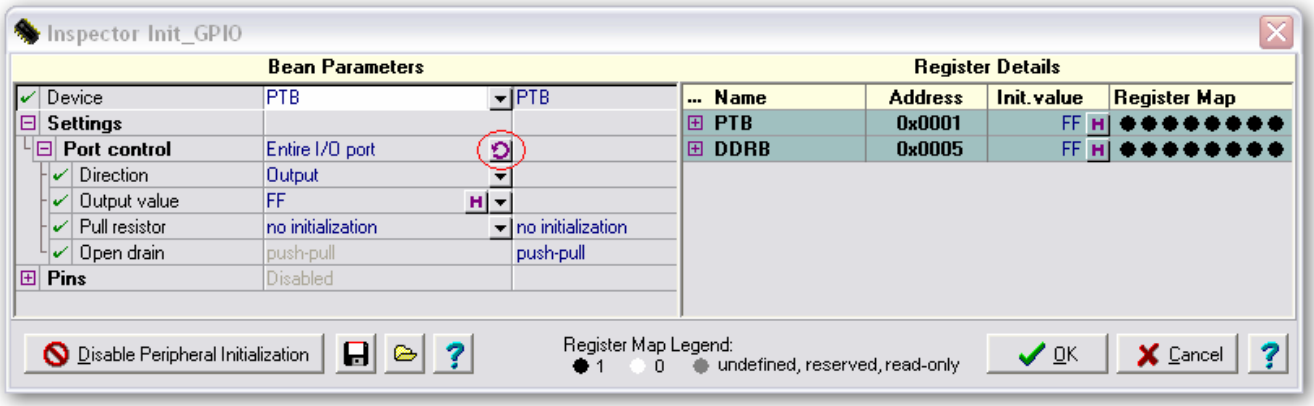


Figura 2.24: Módulo PTB

2.3 GRABADOR “MON08” ----------

El programa Codewarrior a diferencia de otros cuenta con un subprograma que se encarga de grabar el MCU así que ya no es necesario trasladar el archivo del código objeto del programa que se a diseñado al programa que se encarga de grabar al MCU, el archivo que utiliza el programa grabador y la herramienta MON08 para grabar el MCU se puede identificar por la extensión .s19 también conocido como s-record.

2.3.1 CIRCUITO MON08

El circuito de la figura 2.25 es un grabador Mon08 de fácil armado en una tabla de prototipos, en el capitulo tres se mostrará como fabricar una interfase Mon08 en circuito impreso.
Cada una de las partes del circuito de la figura 2.25 desempeñan las siguientes funciones:

DSUB9M: Un conector DB9 hembra conectado al puerto serial del PC, de este conector surgen tres hilos los cuales son Recepción del PC (pin 2), Transmisión del PC (pin 3), y GND (pin 5). La función de esta componente es conectar al ordenador y a la interfase MON08 entre si.

MAX232: Este componente es un transeiver y se encarga de traducir los niveles lógicos del puerto serial a niveles lógicos TTL y viceversa.

74LS125: Este circuito integrado cuenta con 4 buffer´s de 3 estados de los cuales solo se utilizan 2 y se encargan de convertir una comunicación serial de dos hilos a un solo hilo bidireccional evitando que los datos transmitidos por el MCU lleguen al pin del max232 que transmite datos en dirección al MCU y viceversa, en pocas palabras se evitan colisiones.

D1: El circuito max232 genera un voltaje de 10v y para que el MCU pueda ser grabado necesita un voltaje superior al de polarización, este voltaje se llama VTST. Según los datos del fabricante VTST es:

$$VTST = VCC + VHI$$

$$VHI = VCC\ (0.7)$$

Sustituyendo:

$$VTST = VCC + VCC\ (0.7)$$

Como VCC es igual a 5v entonces:

$$VTST = 5v + 5(0.7)$$

$$VTST = 8.5v$$

Por lo tanto podemos recortar el voltaje de 10v generado por el max232 con un simple diodo zener de 8.5v, puede utilizarse un diodo zener con mayor voltaje de avalancha, es mas el circuito MON08 funciona correctamente sin el diodo zener pero la resistencia R3 de 1KΩ no debe faltar ya que esta evita que se dañe el microcontrolador y el diodo zener.

C1, C2, R2, X1: Componentes que en conjunto forman el Oscilador. El cristal de 4.9152MHz puede ser cambiado por uno de 9.8304MHz, para este cambio observe las notas del circuito 2.26. Estos cristales son difíciles de conseguir, el de 4.9152MHz esta disponible en AG Electrónica. Los componentes C1 y C2 pueden estar en el rango de 6pF a 30pF, pero ambos deben ser iguales.

El circuito se alimenta de una sola fuente de 5V.
Al final obtenemos un único conector de 6 pines que convierten a este grabador en universal para todos los MCU HC08.

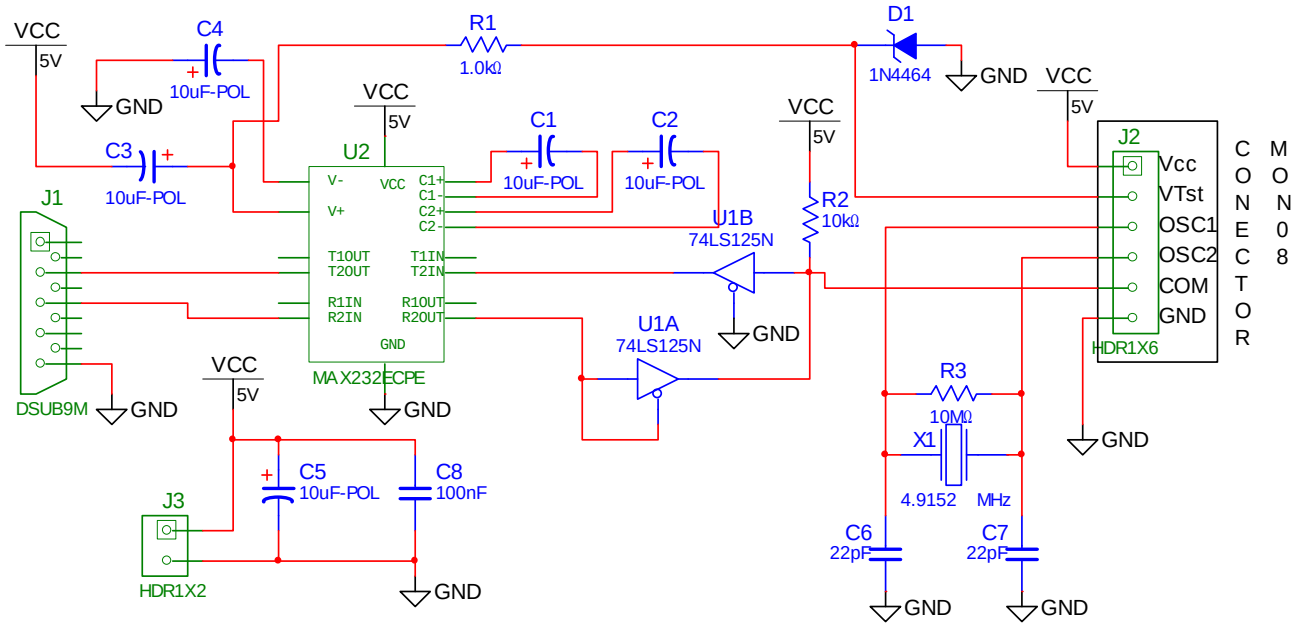


Figura 2.25: MON08

2.3.2 CONEXION ENTRE EL MCU Y EL MON08

En el circuito de la figura 2.26 puede observarse como conectar el grabador MON08 al MCU.

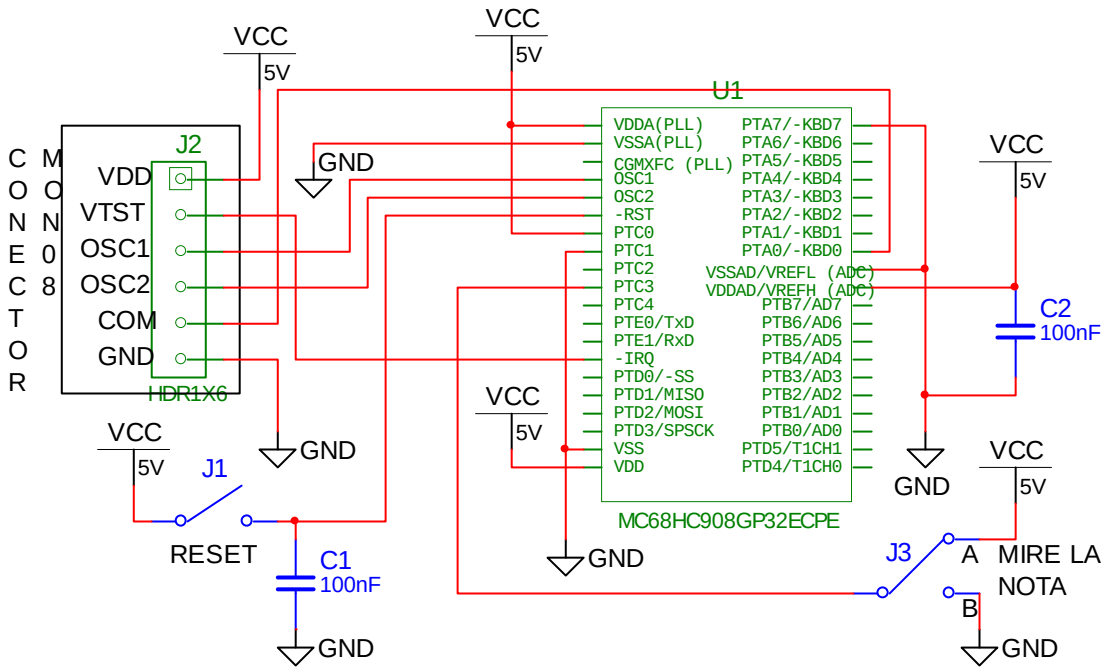


Figura 2.26: Conexión entre MON08 y MCU

> **NOTA:** Si usted tiene un MON08 con un cristal de 4.9152MHz deberá colocar el swich en la posición B.
> Si usted tiene un MON08 con un cristal de 9.8304MHz deberá colocar el swich en la posición A.

Si desea conectar su grabador MON08 a algún otro MCU consulte el manual correspondiente.

2.3.3 CONEXIÓN ENTRE EL GRABADOR Y EL PC

Para ilustrar este tema nos valemos de un sencillo ejemplo.

Ejemplo:
Se debe elaborar un programa que controle el encendido y apagado de un LED rojo a una frecuencia de 1Hz con un ciclo de trabajo del 50%, el LED debe estar conectado al PTA 0, y el MCU debe trabajar con un cristal de 4MHz es decir la velocidad de BUS es de 1MHz. Muestre paso a paso el proceso de diseño, la grabación y el circuito final. Tome en cuenta que el LED debe trabajar a su corriente nominal, y una fuente de polarización de 5v. Como un extra el diseño debe basarse en lógica positiva.

Solución:

1.- El primer paso es decidir que MCU es apropiado para el diseño, y el MC68CH908GP32 sobrepasa con creses las necesidades de nuestro circuito por lo que no es apropiado pero como hemos ejemplificado toda clase de problemas con este MCU es apropiado utilizarlo para este ejemplo de tipo didáctico.

2.- En este momento seria bueno realizar el circuito para estar totalmente seguro que nuestro MCU puede cumplir con las exigencias de nuestro diseño además de los cálculos pertinentes para cada uno de los componentes a utilizar.

Antes que nada veamos los requisitos del circuito, el problema menciona que el LED debe funcionar a su corriente nominal, la corriente de trabajo nominal de un LED promedio es de 25mA por lo que el MCU no podrá cumplir con este requisito ya que cada puerto del MCU puede suministrar tan solo 10mA por pin y con los puertos PTC0 a PTC4 hasta 15mA por lo que recurrimos ala siguiente solución.

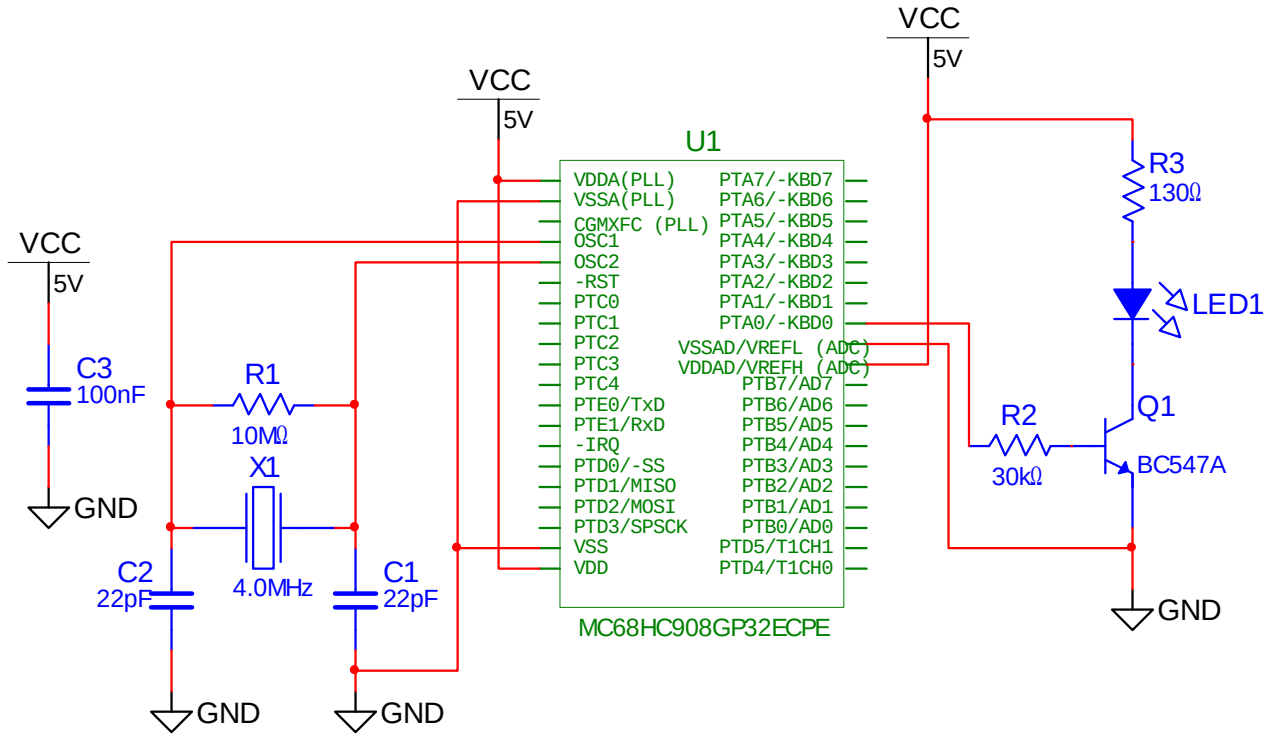


Figura 2.27: Solución al circuito del problema

Para solucionar el problema se recurre a una pequeña etapa de potencia dada por un transistor TBJ del tipo NPN el cual tiene una corriente máxima colector - emisor de 500mA lo cual supera nuestros requisitos. Con un transistor de estas características podemos notar que cuando el puerto del MCU esté puesto en alto el LED prenderá y cuando este puesto a un nivel lógico bajo se mantendrá apagado, a esto se le llama funcionamiento en lógica positiva. En el caso de utilizar un transistor PNP el funcionamiento será inverso esto es lógica negativa.

Para calcular las resistencias de base y de colector hay que tener en cuenta todos lo valores de la malla de salida y muy importante la ganancia mínima del transistor.
La ganancia del transistor es de alrededor de 180.

Analizando la malla de salida por la ley de kirchoff de voltajes y suponiendo que el transistor este saturado tenemos que:

$$V_{CC} - V_{R3} - V_{LED} - V_{CE} = 0v$$

Donde:

- V_{CC} = al voltaje de polarización = 5v
- V_{R3} = al voltaje en R3 =Ic(R3), Ic es la corriente de colector que debe ser igual a la corriente nominal del LED, R3 es la incógnita a encontrar.
- V_{LED} = al voltaje del LED rojo que es de 1.7v
- V_{CE} = al voltaje de saturación colector – emisor del transistor = 0.2v

Teniendo no más que una incógnita en la ecuación se procede a sustituir valores y a despejar R3, por lo que se obtiene:

$$R3 = \frac{5v - 1.7v - 0.2v}{0.025A}$$

$$R3 = 124\Omega$$

No existen resistencias comerciales de 124Ω por lo que se debe escoger la más próxima por arriba del valor obtenido con el objeto de no dañar al LED, en este caso 130Ω, si no tiene una resistencia de este valor a la mano puede usarse una de 120Ω ya que no se encuentra muy lejos del valor obtenido.

El siguiente paso es calcular la corriente de colector real que circularía si el transistor estuviera saturado dada por la resistencia comercial de 130Ω, para esto utilizamos la misma ley de kirchoff de voltajes y la misma malla de salida del transistor solo que en este caso cambiamos la incógnita Ic por Ic real y se despeja.

$$Ic\ real = \frac{5v - 1.7v - 0.2v}{130\Omega}$$

$$Ic\ real = 23.8mA$$

Se procede a calcular la corriente de base **“necesaria”** para que el transistor se sature esto con la formula de la corriente de base del transistor TBJ.

$$Ib = Ic / ß$$

Donde:

- ß: es la ganancia = 180
- Ic = a la corriente en el colector, en este caso se utiliza la corriente real
- Ib = a la corriente de base necesaria para la saturación del transistor

$$Ib = 23.8mA / 180$$
$$Ib = 132uA$$

Si la corriente de base no es de más de 2mA puede asumirse que el voltaje en PTA 0 es de VCC - 0.8v es decir 4.2v, este es un dato de fabricante presente en el manual del MCU que hace referencia al valor mínimo posible dado por el puerto, utilizamos el valor mínimo para asegurar la absoluta saturación del transistor.
Para el cálculo de la resistencia de base se usa la ley de kirchoff de voltajes en la malla de entrada tomando en cuenta que en este caso VCC será de 4.2v por lo antes mencionado.

Observando la malla de entrada y realizando la sumatoria tenemos que:

$$V_{CC} - V_{R2} - V_{BE} = 0v$$

Donde:

- V_{CC} = es el voltaje dado por el PTA 0 en estado alto = 4.2v
- V_{R2} = el voltaje en la resistencia = Ib (R2)
- V_{BE} = al voltaje del diodo base - emisor del transistor = BC547A transistor de silicio = 0.7v

Sustituyendo valores y despejando R2 tenemos que:

$$R2 = \frac{4.8v - 0.7v}{132uA}$$

$$R2 = 31.06\ K\Omega$$

No existen resistencias comerciales de 31.06KΩ por lo que se debe escoger la más próxima por debajo del valor obtenido con el objeto de asegurar la saturación del transistor, esta resistencia comercial es de 30 KΩ.

3.- El siguiente paso es iniciar un nuevo proyecto Codewarrior, esto ya se ha visto previamente, debe ver lo que aparece en la figura 2.6, ya hecho esto se procede a modificar el tipo de encapsulado observe la figura 2.14.

MODULO CPU

A continuación es necesario modificar los módulos pertinentes para nuestro diseño, por lo que se empieza por seleccionar el módulo CPU y modificar los puntos tal y como se muestra en la figura.

- Oscilator frequency: poner a 4MHz ya que este es el cristal que pide el problema.
- CPU mode selection: colocar en modo usuario el cual hace que el MCU divida el cristal entre cuatro para obtener el BUS

Observe la velocidad del BUS que ya ha sido calculada: 1.0MHz.

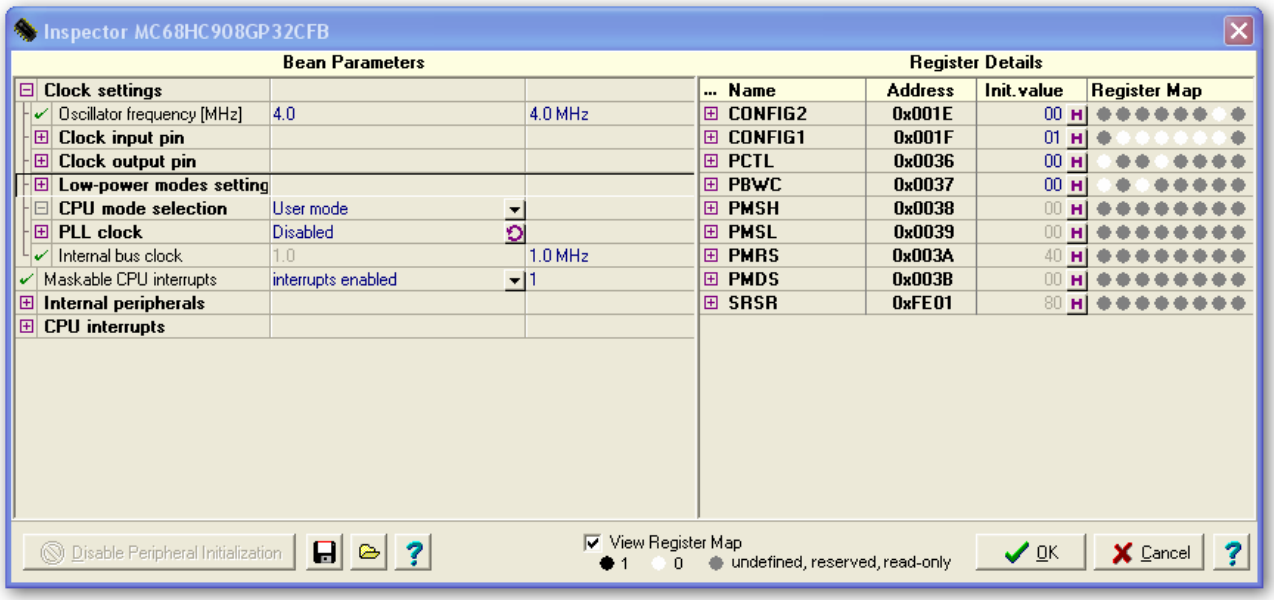


Figura 2.28: Iniciación del módulo CPU

MODULO COP

Se desactiva el COP.

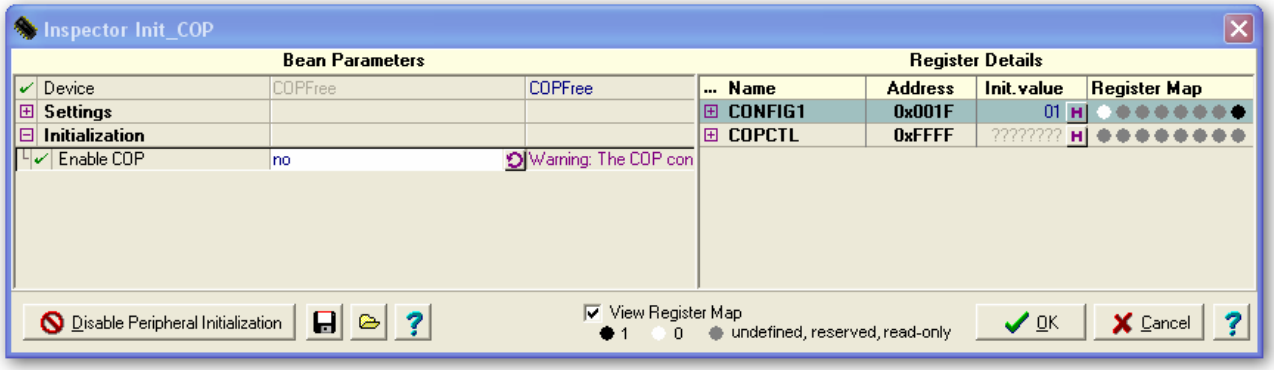


Figura 2.29: Iniciación del módulo COP

MODULO PTA

Se activa el puerto PTA 0 como salida y le damos un valor inicial de 0 es decir cuando se energice el MCU este puerto estará puesto en un nivel lógico bajo o 0v. Observe la figura.
Si no esta seguro de los cambios que ha hecho puede deshacerlos presionando el botón **Disable Peripherial Initialization** en la parte inferior izquierda de cada módulo

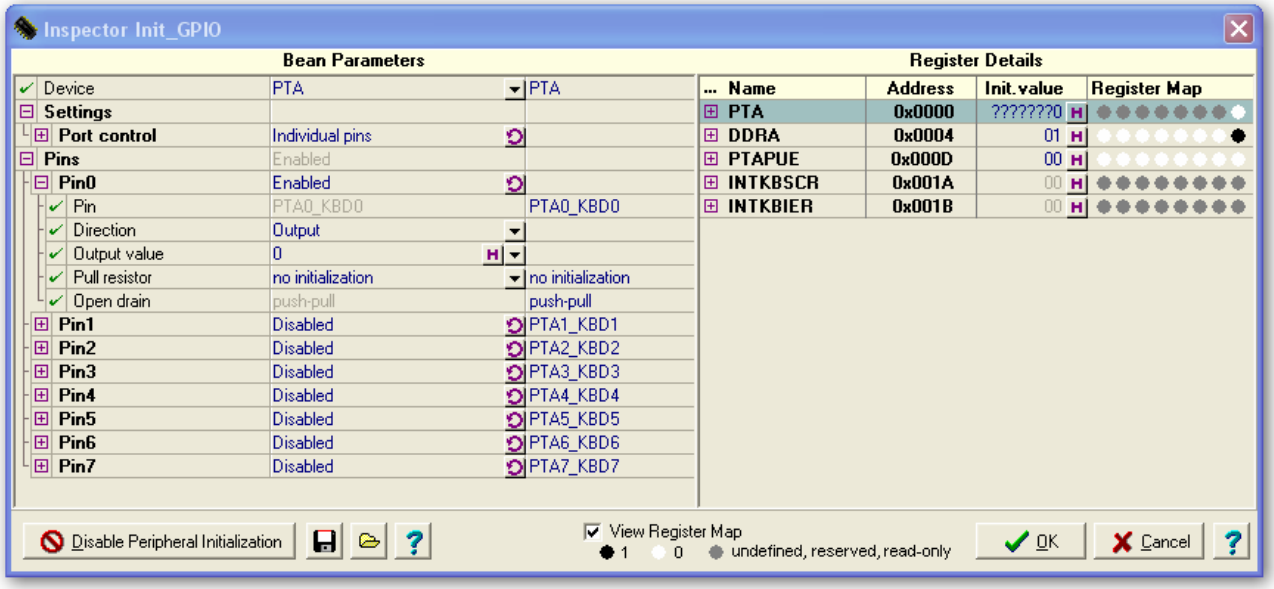


Figura 2.30: Iniciación del módulo PTA

Ya que ha terminado de utilizar el iniciador de componente y muy importante presione el botón **Generate Code** presente en el iniciador de componente ya que este anota las modificaciones hechas en los diferentes módulos en el archivo MCUinit.inc, si no hace esto su programa no funcionará correctamente por lo que al final su diseño embebido no funcionará.
Si ya a utilizado **Generate code** y después modifica algún módulo por algún factor olvidado o erroneo tendrá que volver a presionar **Generate Code** para que la modificación sea apuntada en el archivo MCUinit.inc.

Figura 2.31: Botón Generar Código

Después de presionar el botón generar código presione **Generate** en la ventana que aparecerá mostrada en la figura 2.32.

Figura 2.32: Ventana para generar el código iniciador del MCU.

Después de presionar **Generate** deberá aparecer el archivo MCUinit.inc con las modificaciones que ha hecho en el iniciador de componente tal y como se ve en la figura.

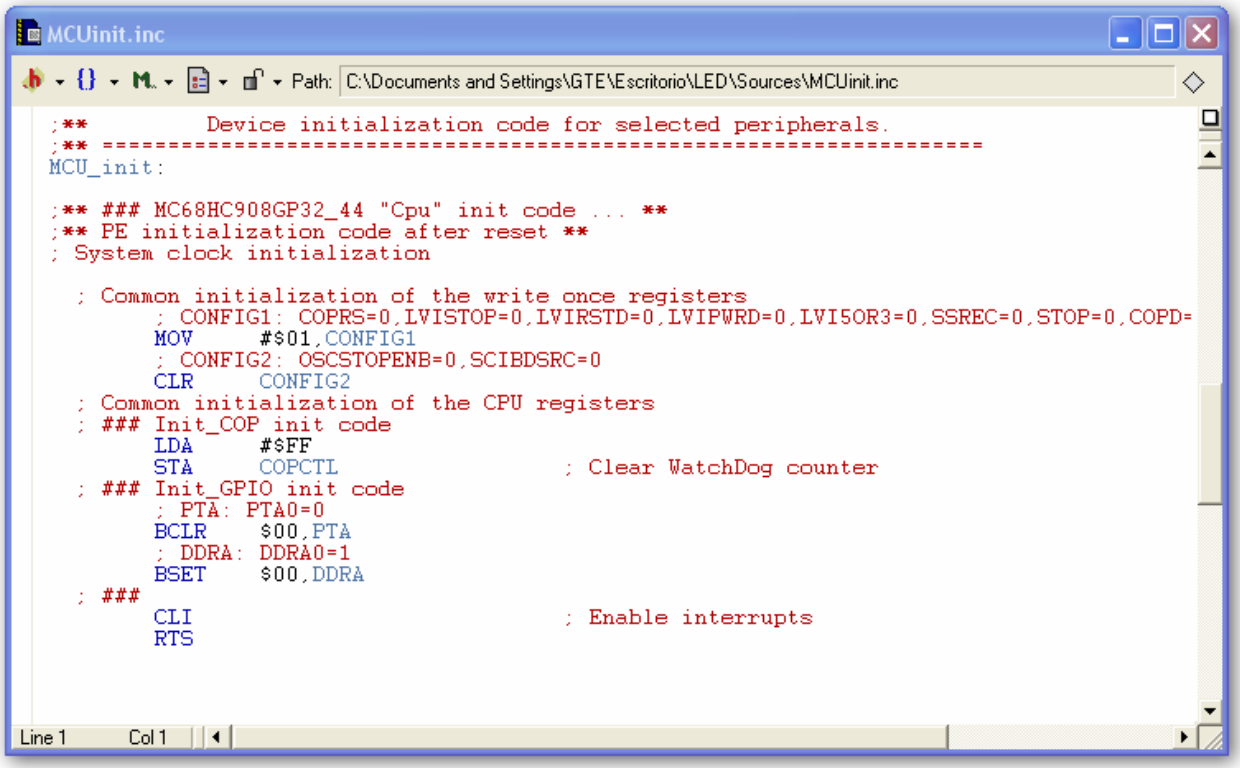


Figura 2.33: Archivo MCUinit.inc modificado.

Recuerde que si regresa a modificar algo olvidado o erróneo en el iniciador de componente tendrá que repetir la generación de código. Para que tenga efecto en el MCU.

Ahora se procede a escribir el programa principal
El problema nos plantea que el LED debe prender y apagar con un periodo de 1Hz con un ciclo de trabajo del 50% esto se traduce simplemente como decir que el LED debe estar prendido durante medio segundo para después apagarse durante medio segundo y así sucesivamente. Por lo tanto el programa se puede escribir de la siguiente manera

```
mainLoop:

        BSET 0,PTA
        BSR  RETARDO

        BCLR 0,PTA
        BSR  RETARDO

        BRA    mainLoop
```

- La primera instrucción pone a uno el puerto PTA 0 lo que provoca que el LED se prenda.
- La siguiente instrucción llama a una rutina de retardo la cual debe durar alrededor de medio segundo.
- Al terminar el retardo el LED se apaga por la siguiente instrucción que pone a 0 el puerto PTA 0.
- La siguiente instrucción llama de nuevo a la rutina de retardo.
- Al terminar el retardo la siguiente instrucción se encarga de redirigir la ejecución del programa en dirección de la etiqueta mainnLoop lo cual provoca que la siguiente instrucción a ejecutarse sea de nuevo la primera cerrando así un lazo infinito.

Para generar una subrutina apropiada es necesario tener en cuenta el tiempo de ejecución del ciclo máquina, al utilizar un BUS de 1MHz el ciclo maquina tiene un periodo de 1us, por lo tanto se necesita de una subrutina que dure 500000 ciclos máquina para generar un retardo de medio segundo

La siguiente subrutina dura 500937 ciclos lo que es muy aproximado a lo requerido, si se desea tener algo mucho mas preciso utilice las interrupciones del módulo timer.

```
RETARDO:   CLRX
LOOP2:     CLRA
LOOP:      CBEQA #$FF,ETIQUETA
           INCA
           BRA   LOOP

ETIQUETA:  CBEQX #$F3,FIN
           INCX
           BRA   LOOP2
FIN:       RTS
```

En la figura se muestra como debe quedar escrito el programa

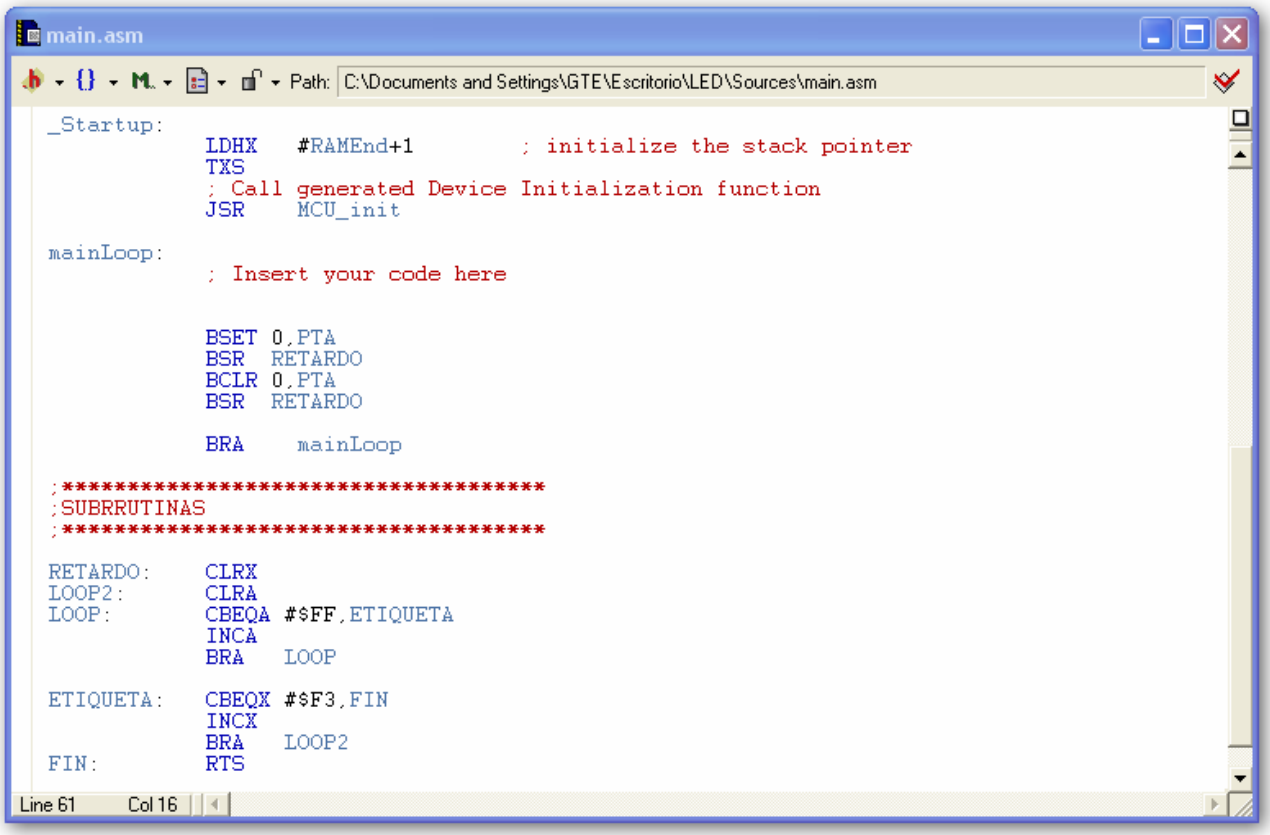


Figura 2.34: Programa principal y subrutina.

A continuación se procede a simular el programa por lo que debe seleccionar la conexión **Full chip simulation** y presionar el botón **debug** (Figura 2.35)**,** si el programa no tiene errores aparecerá la ventana de la figura 2.35 este es un programa de simulación y de depuración en circuito llamado **True - Time Simulator & Real – Time Debugger.**

En esta ventana puede verse la ventana del código que usted ha escrito y lo que el iniciador de componente ha hecho por usted **(Source)**, la ventana del leguaje ensamblador sin etiquetas **(Assembly)**, la ventana que indica el estado de los registros **(Register)** y la ventana que muestra el estado de toda la memoria **(Memory)**.

Presione el botón  o F11 para avanzar paso a paso.

Presione el botón  o F5 para correr el programa.

Presione el botón  o F6 para pausar el programa.

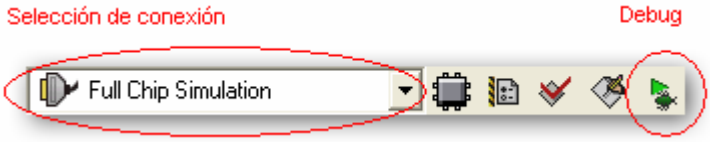
Presione el botón  o ctrl+R para reiniciar el programa.

NOTA: Si desea ejecutar el programa hasta un punto específico, presione el botón secundario del ratón dentro de la ventana **source** exactamente sobre la instrucción en la que se desea que la ejecución del programa se detenga y seleccione la opción **Run To Cursor.**

Manipule el programa utilizando el botón paso a paso para ir lentamente y la instrucción Run To Cusrsor para saltarse la subrutina de retardo, ahora puede observar como la localidad de memoria $0000 la cual corresponde al puerto A cambia de valor específicamente el bit menos significativo correspondiente al puerto PTA 0.

Cuando se ejecuta la instrucción BSET, PTA es igual a $01
Cuando se ejecuta la instrucción BCLR, PTA es igual a $00

Estos datos se observan en la ventana correspondiente a la memoria.



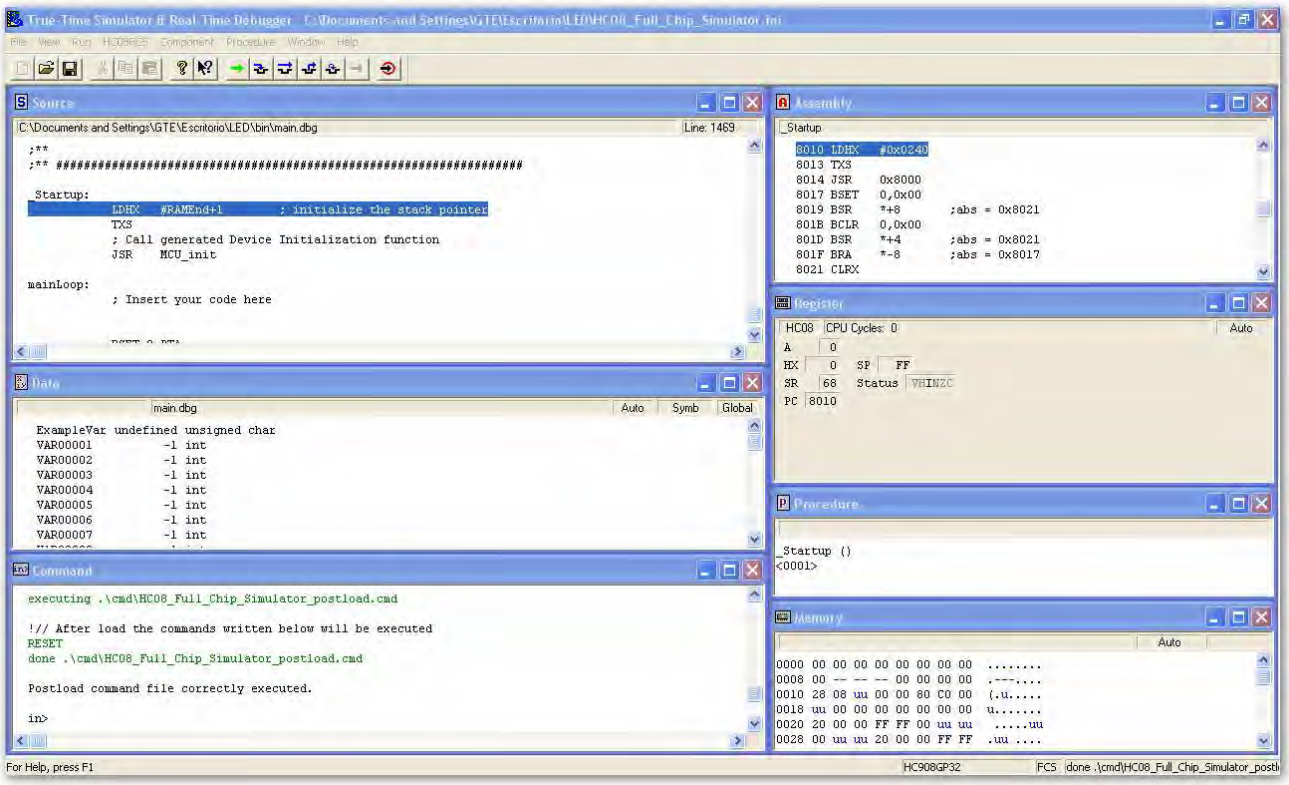


Figura 2.35: Simulación.

Si el programa se ejecuta correctamente se procede a grabar al MCU. Conecte su grabador Mon08 ya conectado al MCU al puerto serial del PC y después energice el circuito con 5v, cierre la ventana de la figura 2.35 y seleccione la conexión **Mon08 Interface** y presione **Debug,** con esto aparecerá la ventana mostrada en la figura 2.36, este es el asistente de conexión.

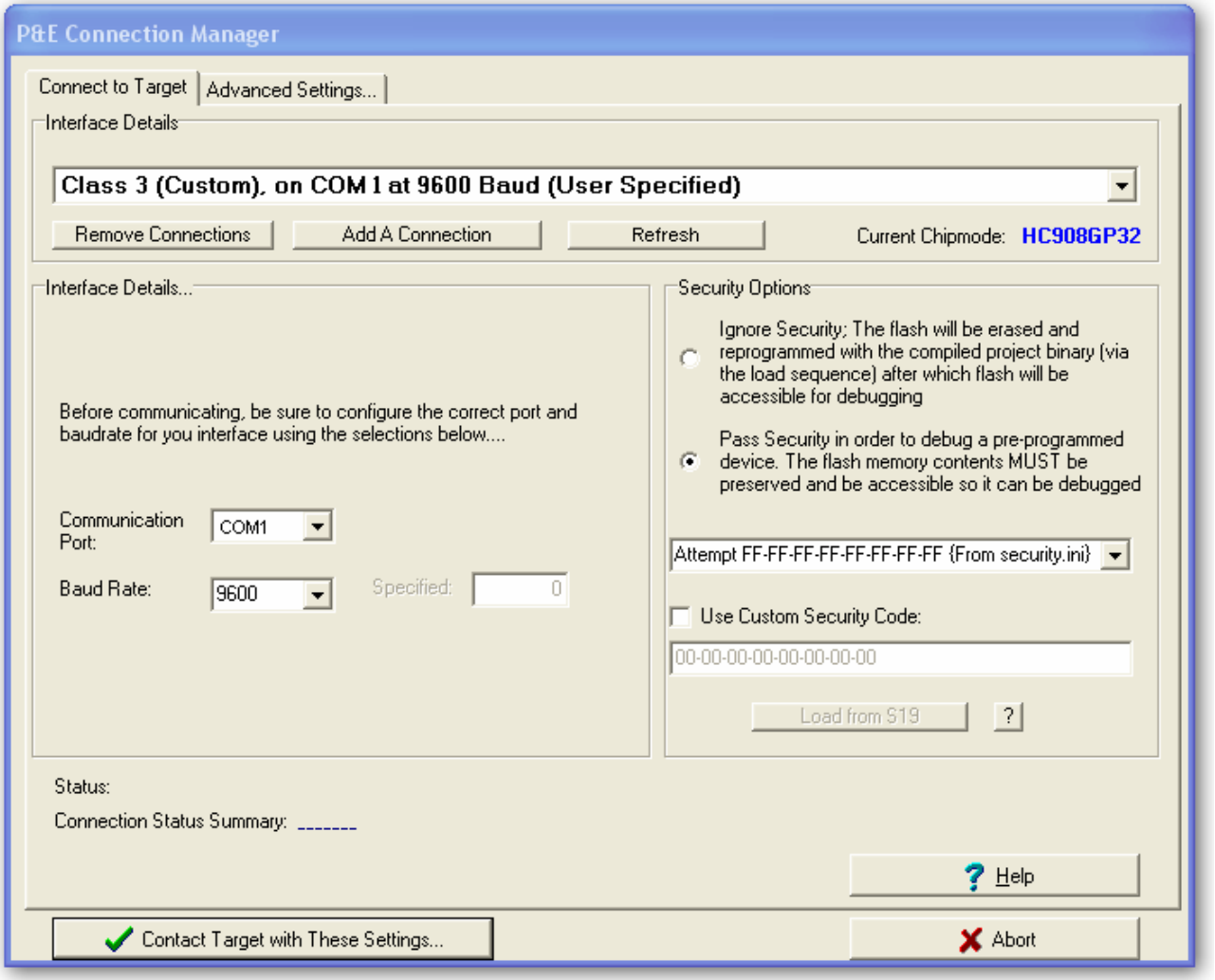


Figura 2.36: Asistente de conexión.

El primer paso es presionar el botón **Add a Connection** y seleccionar la internase clase 3.

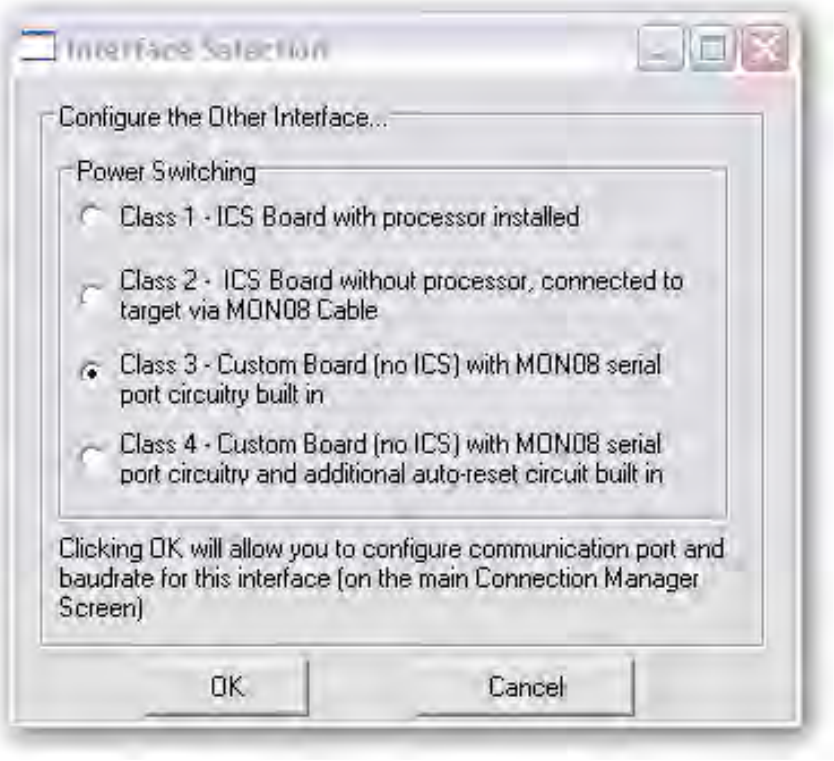


Figura 2.37: Selección de interface.

El segundo paso es seleccionar el puerto seria al que esta conectado el circuito MON08 el cual suele ser el COM 1 ya que las computadoras de escritorio actuales solo traen un puerto serial físico disponible, si aparecen mas puertos no los tome en cuenta ya que siempre el COM 1 esta ligado al conector del puerto serial disponible.

El tercer paso es seleccionar la velocidad de comunicación del puerto serial en este caso 9600 Baudios.

Por ultimo en las **opciones de seguridad** escoja la segunda opción. El asistente de conexión debe lucir tal cual aparece en la figura 2.36. Ahora solo presione el botón **Contact Target with These Settings…** con esto aparecerá la ventana mostrada en la figura 2.38 que pide que se disminuya de golpe el voltaje de polarización a menos de 0.1v para después volverlo a suministrar, esto quiere decir que se debe apagar y prenda la fuente o desconectar y conectar el borne positivo de la fuente al circuito aunque la figura menciona que se debe reducir el voltaje solo en el MCU aunque en realidad no importa este punto ya que se puede desconectar todo el circuito junto con el grabador Mon08 además de que esto es mas práctico en caso de no contar con una fuente con botón de encendido y apagado, ya hecho esto presione el botón OK.

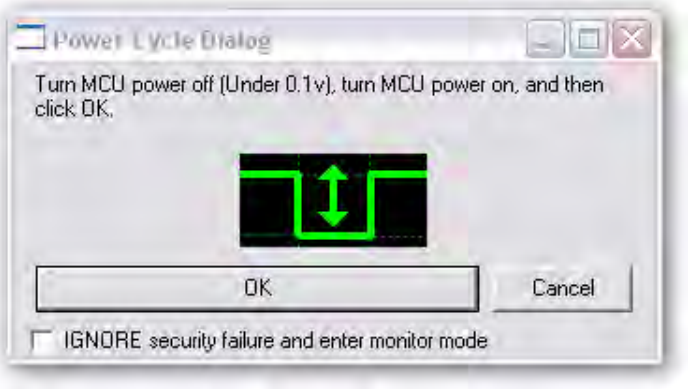


Figura 2.38: Selección de interface.

Si el circuito no tiene errores de armado, ni existen errores en la configuración de la conexión aparecerá la ventana mostrada en la figura 2.39, para grabar el MCU presione el boton **YES.**

Figura 2.39: Permiso para grabar el MCU.

En este momento el MCU se borra y reprograma, si descubre que el circuito no funciona por algún error en el programa puede grabar y borrar el MCU asta 10 000 veces, siguiendo los mismos pasos ya mencionados.
El tiempo de retención de datos de la memoria Flash es de alrededor de 10 años.

A continuación aparece la ventana mostrada en la figura 2.35 solo que en este caso la simulación será visible en el MCU, por lo que si pretende simular estando conectado el MCU a la PC a través del Mon08 deberá tener en cuenta que su programa no podrá utilizar los puertos PTA7, PTA0, PTC0, PTC1 y PTC3 ya que estos son parte de la configuración para la conexión con el PC a través del Mon08 llamado modo monitor como ya se a mencionado con anterioridad.
Los puertos para la configuración del modo monitor o conexión con el MON08 varían en cada MCU.
Recuerde que la velocidad del BUS para la ejecución del programa en modo monitor esta dada por la velocidad del cristal entre 2 es decir 4.9152MHz / 2 = 2.4576MHz, por este motivo se recomienda mucho hacer las pruebas del programa en modo monitor ya que por lo regular en el diseño de sistemas embebidos se utilizan retardos muy precisos para los procesos de comunicación calculados con determinadas velocidades de cristal.

Si desea que su programa se ejecute en modo usuario a la misma velocidad que en modo monitor utilice un cristal para el modo usuario de 9.8304MHz con esto el BUS del modo usuario será igual al BUS del modo monitor, este cristal es difícil de conseguir.

La figura 2.40 muestra al MCU conectado a un pequeño prototipo de Mon08 universal, en este caso el prototipo no cuenta con diodo zener ni con la resistencia de 1KΩ, se a agregado en la protoboard la resistencia de 1KΩ sin el diodo zener y no se a colocado el botón de reset puesto que este es solo útil cuando se reinicia el sistema en modo monitor es decir, si se esta simulando el sistema con el MCU conectado al PC y se necesita reiniciar además de presionar el botón de reinicio en la ventana del simulador el programa le pedirá que presione el botón de reinicio conectado en el MCU.

En la figura 2.40 puede apreciarse el conector del prototipo Mon08 el cual no tiene la misma configuración que el conector del circuito de la figura 2.25. Esta fotografía es el circuito armado entre las figuras 2.25 y 2.26.

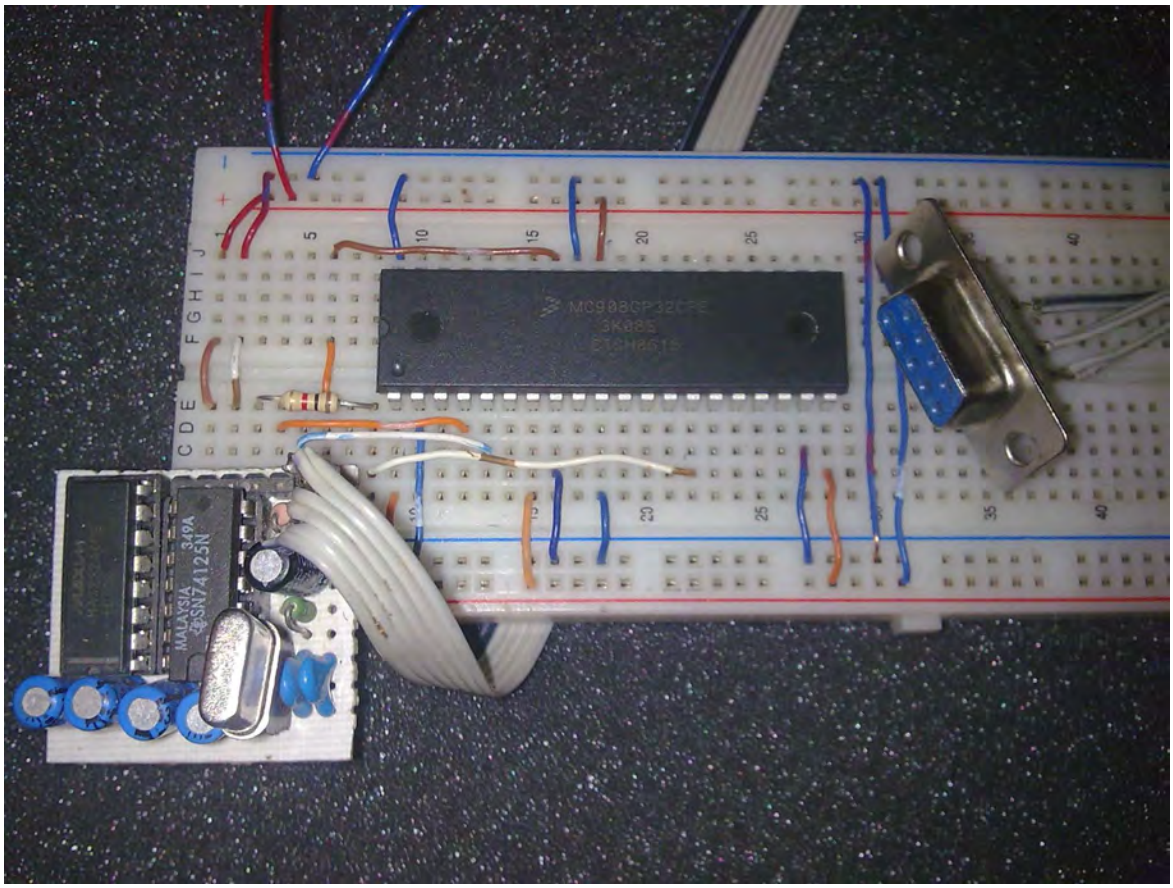


Figura 2.40: Prototipo de Mon08 conectado al MCU.

En la figura 2.41 puede apreciarse al MCU en modo usuario conectado tal como lo pide el circuito de la figura 2.27 que es la aplicación que se desarrolló en este ejemplo.

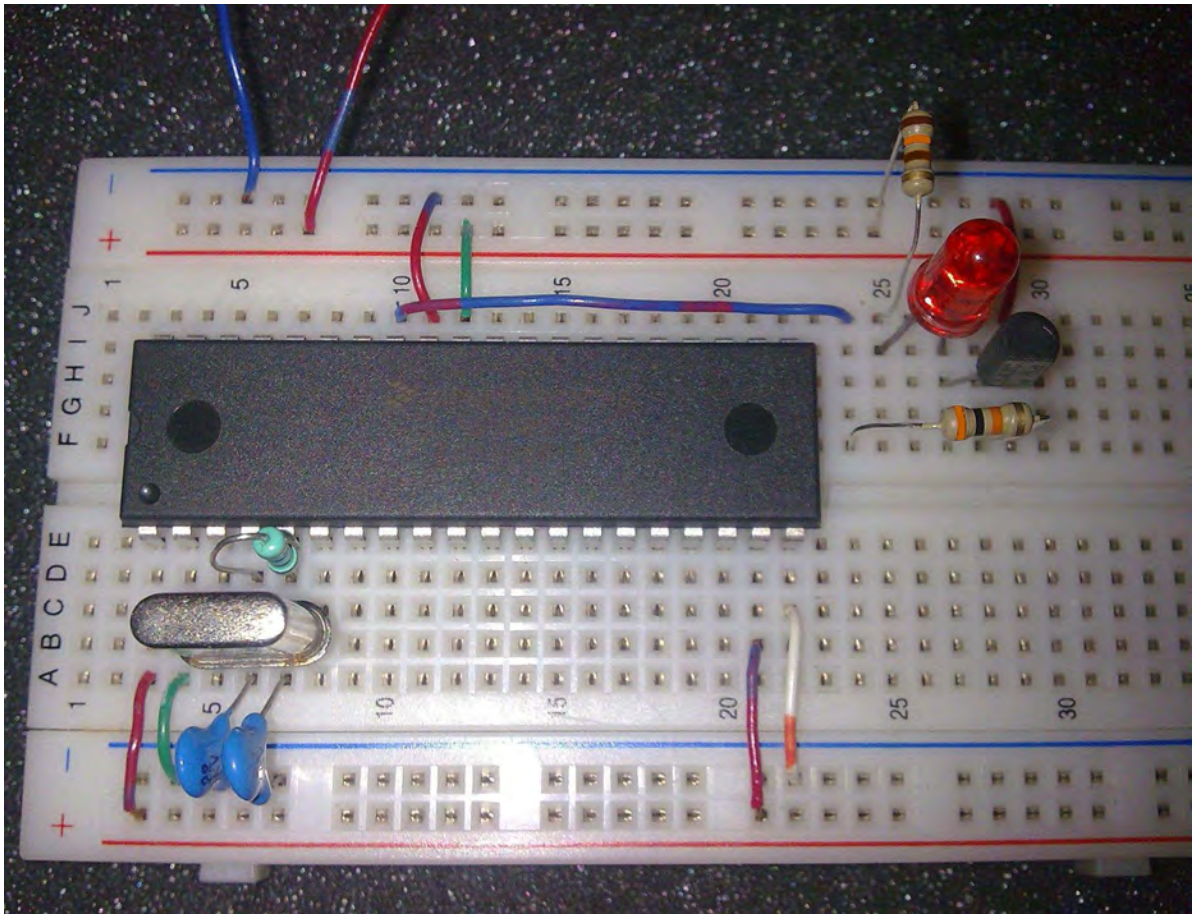


Figura 2.41: Ejemplo montado en una tabla de prototipos.

2.3.4 PCB DEL GRABADOR MON08

En la siguiente figura puede verse una ventana del programa Ultiboard 9 con el PCB finalizado de un grabador universal MON08 el cual esta diseñado en base al circuito de la figura 2.25.

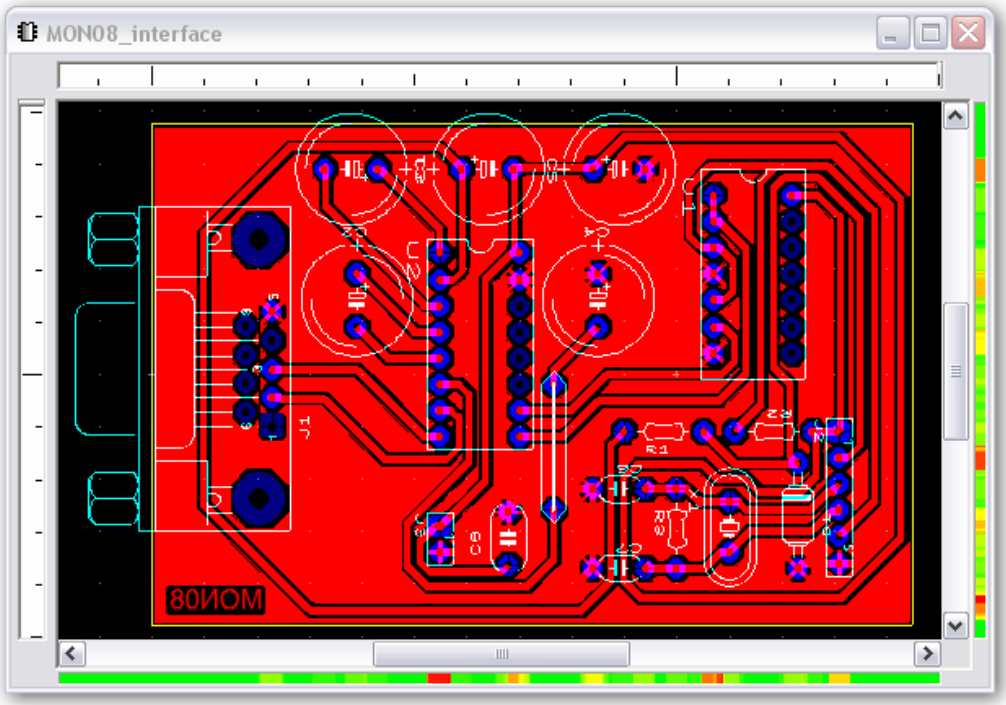


Figura 2.42: Mon08 terminado en Multisim 9.

En la siguiente imagen se puede apreciar una vista superior en 3D del diseño del grabador Mon08.

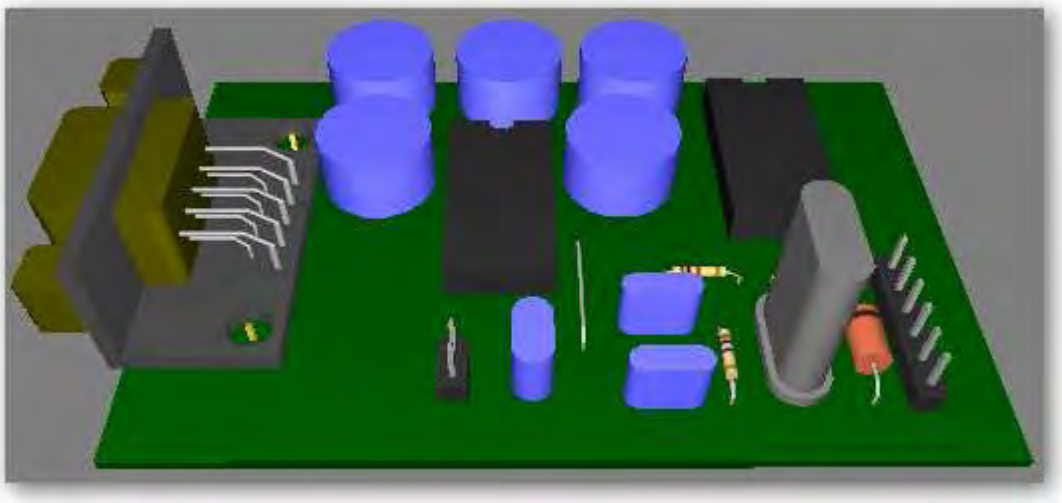


Figura 2.43: Vista superior del Mon08 en 3D.

En la siguiente imagen puede apreciarse una vista inferior en 3D del diseño del grabador Mon08.

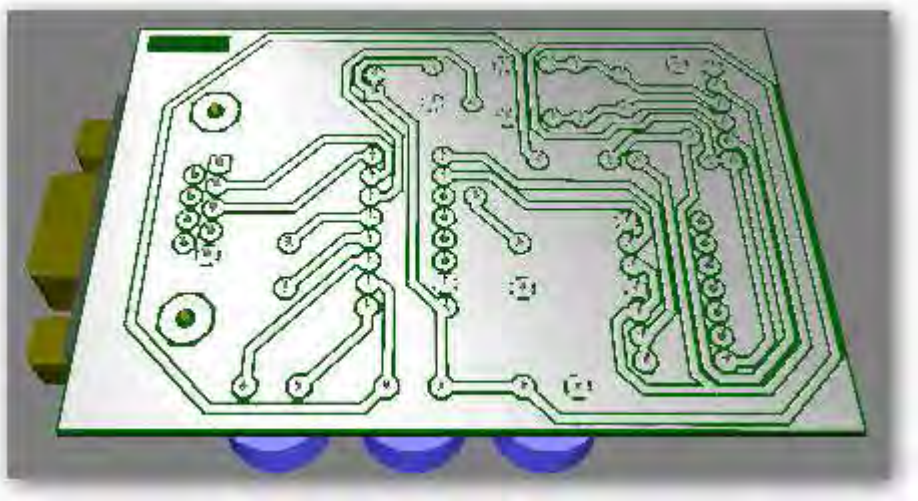


Figura 2.44: Vista inferior del Mon08 en 3D.

En esta imagen puede verse el espejo del PCB para traspasar el circuito del grabador a la tabla fenólica virgen, para obtener este impreso al tamaño correcto debe imprimir desde el programa Ultiboard 9, en el siguiente capitulo se mostrará este proceso.
Los programas Multisim 9 y Ultiboard 9, instructivo de instalación de los mismos y el archivo de este circuito se encuentran en el DVD anexo al final de este trabajo.

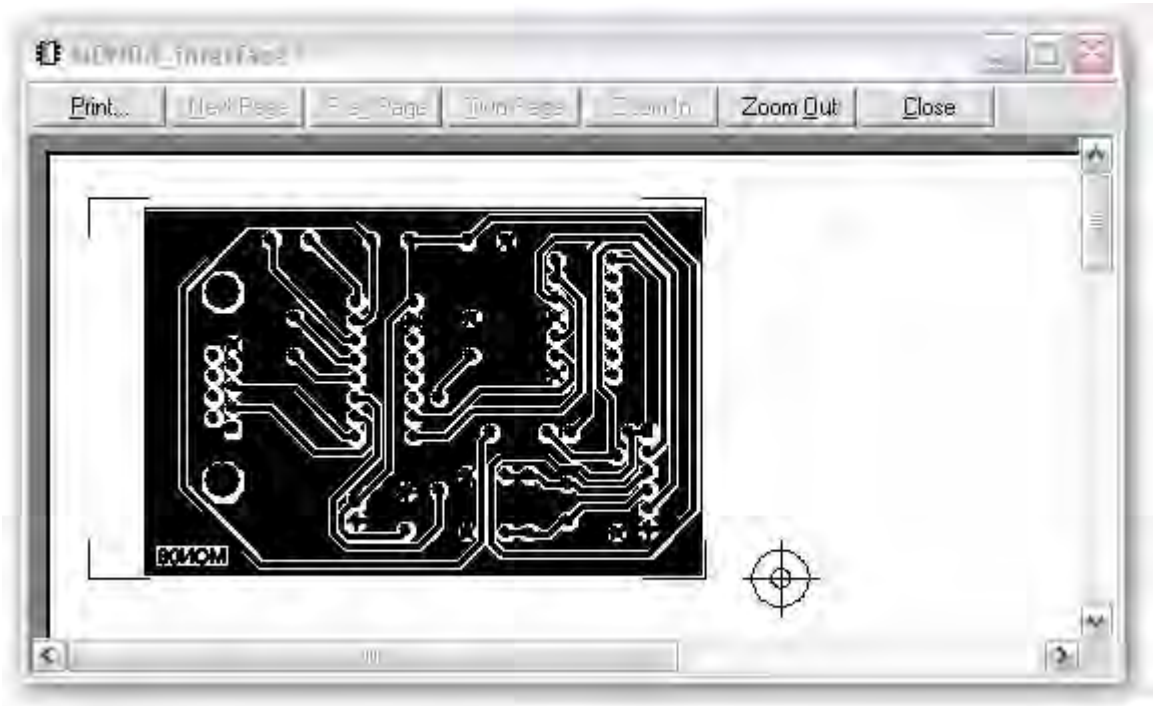


Figura 2.45: Espejo del PCB ventana de impresion.

2.4 RESUMEN ----------

Uno de los recursos mas poderosos con los que cuenta el codewarrior es el subprograma Processor Expert y su iniciador de componente, con este puede ahorrarse en mucho la estructura de inicio del programa para el microcontrolador seleccionado y además puede ayudar en gran parte a comprender como deben manipularse los registros y bits de propósito especial de un estado a otro o que código se debe utilizar para reiniciar un módulo en especial tal y como lo hubiera hecho el iniciador de componente, esto se debe en gran medida a la sección **detalles de los registros** la cual se encuentra en la parte derecha de cada uno de los módulo vistos en este capitulo, para aclarar a que me refiero si usted inicializa la interrupción por Timer con el iniciador de componente al ejecutarse esta en el plazo establecido no lo volverá a hacer por lo tanto usted tendrá que reiniciar la cuenta al final de su rutina de interrupción, puede evitarse meterse en detalles con el manual viendo tan solo que códigos utilizó el iniciador de componente para echar a funcionar la interrupción por Timer sin embargo para explotar al máximo todas las posibilidades de un microcontrolador debe entenderse que es lo que el iniciador de componente hace por usted ya que este no podrá hacer nada en el transcurso de su código es decir si usted necesita modificar los registros que el iniciador de componente ya a arreglado por usted será necesario que aplique sus conocimientos acerca de los registros del microcontrolador utilizado. Existen nuevas opciones disponibles en los microcontroladores HCS08. Existen mas módulos de los vistos en este capitulo en los microcontroladores HC08 como lo son:

Int_EEPROM: Internal EEPROM
Inr_EEPROM 2: Internal EEPROM 2
IRSCI: Infra-red serial communication interface module
MMIIC: Multi_Master IIC interface
PIT: Programmable Interrupt Timer
BDLC: Byte Data Link communication (BDLC) subsystem
CAM: MSCAN controller
USB: Universal Serial Bus Module
USBhub: USB hub
DDC12AB: DDC12AB Interface
SyncProc: Sync Processor
PWM: Pulse width Modulator
ESCI: Enhanced Serial Communication Interface Module
MSCAN08: MSCAN08 Controller
HRP: Lamp Control Module
OpAmp: Op Amp / Analog Comparator
OSD: On Scree Display Module
RTC: Real Time Clock
LCD: LCD Controller
PPI1: Programmable Periodic Interrupt
PWMMC: PWMMC
PWU: Periodic Wakeup Module
SLIC: Slave LIN Interface Controller
Analog Module: Analog Module

CAPÍTULO

3

DISEÑO DE CIRCUITOS IMPRESOS. "MULTISIM" Y "ULTIBOARD"

OBJETIVOS

Después de estudiar este capítulo usted podrá ser capas de:

- Iniciar un nuevo proyecto con el software Multisim 9.
- Diseñar un circuito para su traslado al Ultiboard 9.
- Crear nuevas componentes para diseño de circuitos no para simulación.
- Conocer el funcionamiento de Ultiboard 9 y el enrutado del circuito.
- Imprimir y traspasar el circuito a la tabla fenólica virgen.

Uno de los softwares mas sencillos y amigables para el diseño de circuitos eléctricos y electrónicos es el programa Multisim pero no solo sirve para el diseño si no para la simulación, cabe mencionar que hay componentes que no se pueden simular. La simulación se puede llevar acabo en corriente alterna y en circuitos lógicos secuénciales como lo son compuertas o contadores, la lógica programable y dispositivos con los MCU´s no se pueden simular. Además se puede transportar un circuito en Multisim en un solo paso al programa Ultiboard para el diseño de PCB´s. Las versiones de estos programas que utilizaremos son Multisim 9 y Ultiboard 9, no se utilizará la nueva versión 10.1 debido a que e detectado algunos errores y utilidades desactivadas que asen un diseño más lento y tedioso.

3.0 ENTORNO MULTISIM 9

Antes que nada se debe mencionar que este programa al igual que el Ultiboard en su novena versión están diseñados para un entorno Windows XP aunque funcionan sin contratiempos en versiones de Windows Vista. Ejecute el programa Multisim 9 y aparecerá de inmediato un nuevo proyecto siempre llamado **"Circuito 1"**, observe la figura 3.1.

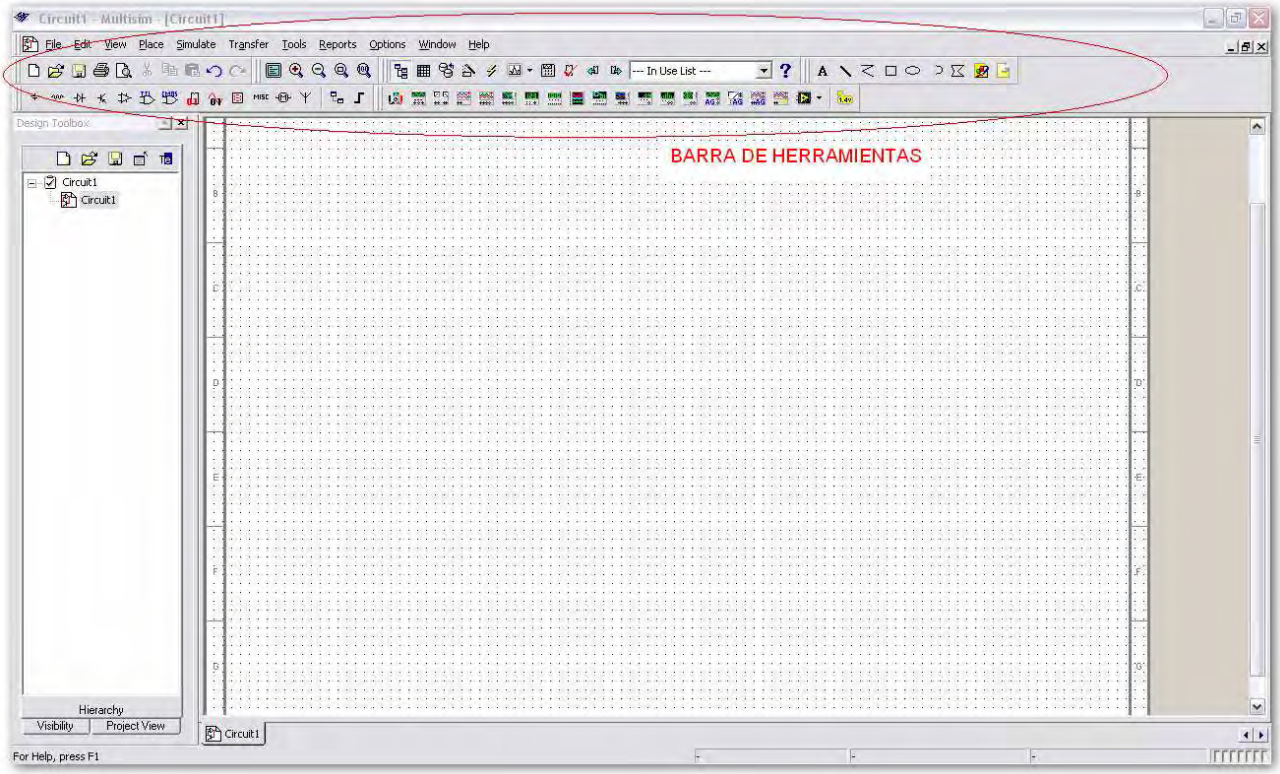


Figura 3.1: Entorno Multisim 9.

Para modificar la barra de herramientas solo tiene que presionar el botón secundario del mause sobre la barra y activar o desactivar las diferentes herramientas, aquí se muestran algunas de las herramientas que más se utilizan en el diseño.

3.0.1 ESPACIO DE TRABAJO

Puede cambiar las características del espacio de trabajo presionando el botón secundario del mause sobre el espacio de trabajo y seleccionar la opción **Properties**, entonces aparecerá la siguiente ventana mostrada en la figura 3.2, aquí se podrá modificar una gran cantidad de variables del espacio de trabajo desde el tamaño asta la aparición o no de las etiquetas con las que cuenta cada componente para mostrar sus características esto ultimo con el recuadro **show**.

Lo que regularmente cambia un diseñador en su espacio de trabajo es el color con el que mas se acomode. Para largas horas de trabajo se recomienda un fondo negro, pero si se desea incluir imágenes del diseño en informes escritos es preferible usar un fondo blanco.

De la figura 3.3 a la 3.6 se muestran los diferentes entornos de trabajo disponibles, aunque existe el modo **custom** con el cual podrá modificar el ambiente de trabajo a su gusto, este esta visible en la figura 3.7.

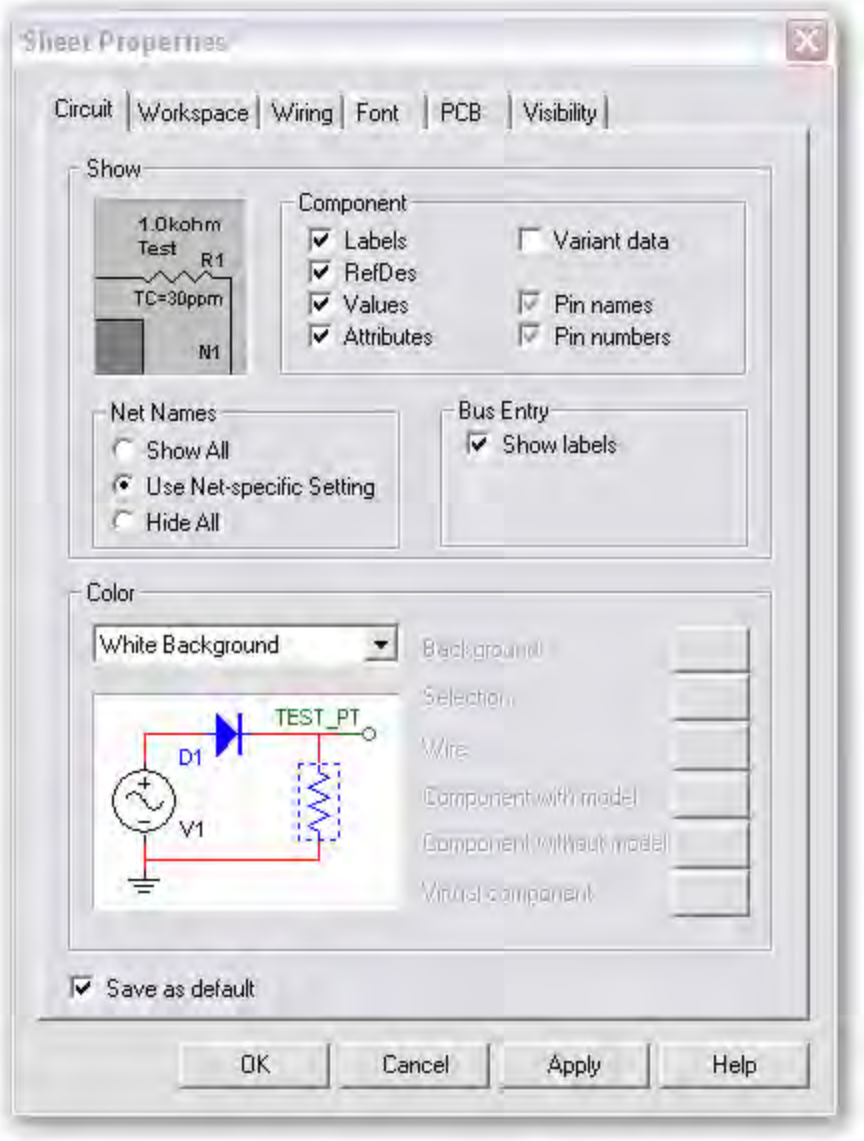


Figura 3.2: Hoja de propiedades.

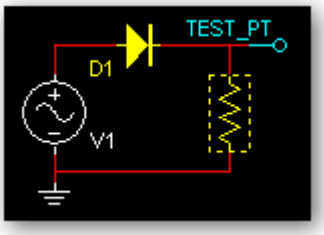


Figura 3.3: Black background.

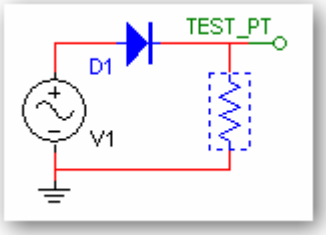


Figura 3.4: White background.

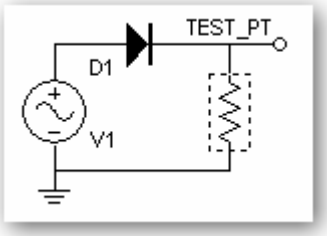


Figura 3.5: White & black.

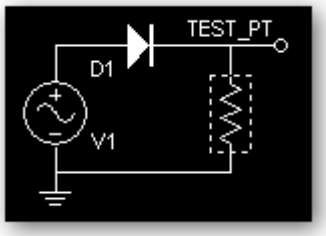


Figura 3.6: Black & white.

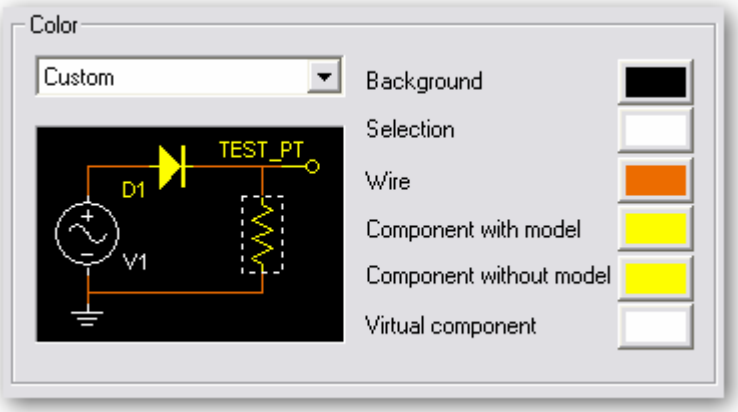


Figura 3.7: Custom.

3.0.2 BARRA DE HERRAMIENTAS

INSTRUMENTS

Barra de herramientas destinada para la selección de diferentes instrumentos muy útiles en la simulación como lo son: multímetro, osciloscopio, vatímetro, generador de funciones, analizador de estados lógicos, generador de palabras etc.

Figura 3.8: Instrumentos.

GRAPHIC ANNOTATION

Barra de herramientas para incluir texto con fines de comentario o figuras para delimitar espacios.

Figura 3.9: Gráficos y Anotaciones.

COMPONENTS
Barra de herramientas que tiene el objetivo de acceder a los componentes para pegarlos en el espacio de trabajo, también pueden generarse Buses.

Figura 3.10: Componentes.

3.1 LOS COMPONENTES ----------

3.1.1 SELECCIÓN DE LOS COMPONENTES

Cada vez que seleccione un grupo a acceder en la barra de componentes aparecerá la ventana de selección de componente, el recuadro **group** dependerá del grupo que halla seleccionado. Puede cambiar de entre los diferentes grupos mostrados en la barra de componentes a través del recuadro **group.**

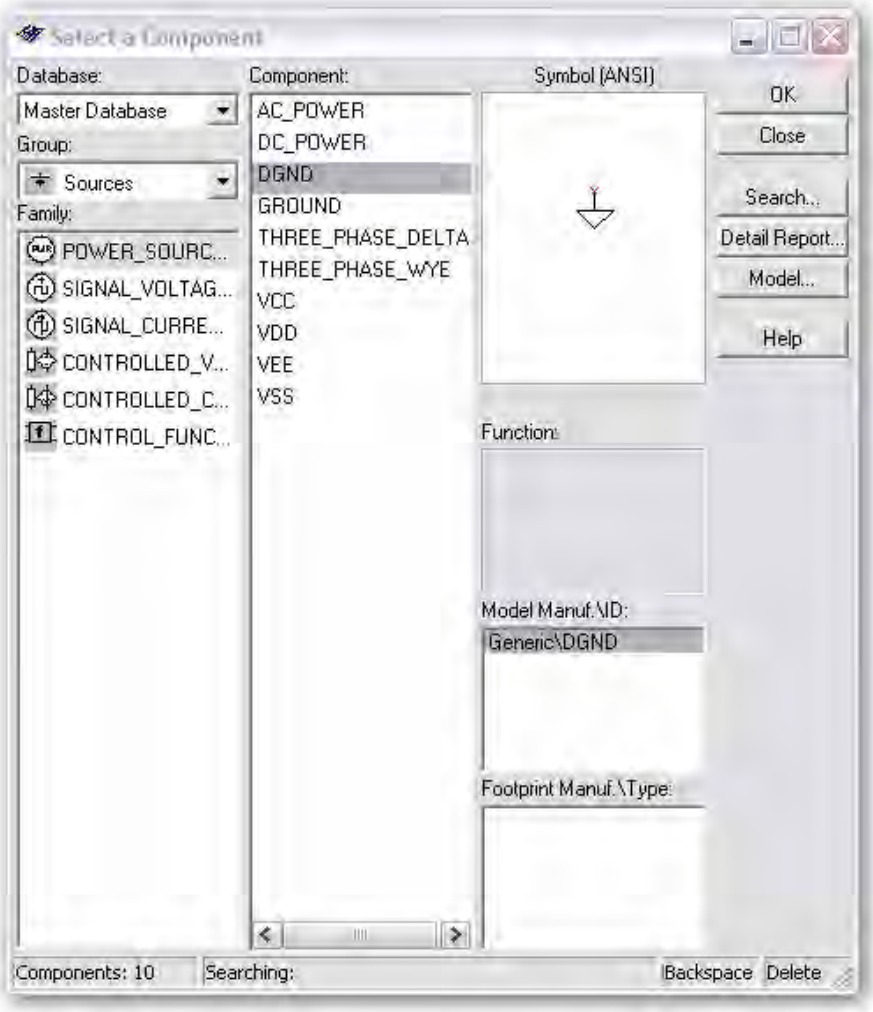


Figura 3.11: Selección de componentes.

3.1.2 ORDEN DE LOS COMPONENTES

POR BASE DE DATOS. Los componentes se encuentran divididos en tres bases de datos de las cuales utilizará generalmente dos.

- **Master Database:** Aquí se encuentran todos los componentes que posee Multisim 9, existen tres versiones de Multisim 9 la estudiantil, la profesional y la Power Pro, la versión Power Pro posee una base de datos más grande mientras que la estudiantil es la de la base más pequeña.

- **User Database:** Aquí se almacenan las componentes que se crean, cada ves que cree una componente quedará almacenada para poder ser usada en proyectos futuros.

POR GRUPO. La Master Database se encuentra dividida en 13 grupos los cuales son fuentes, básicos, diodos, transistores, análogos, TTL, CMOS, misceláneos digital, indicadores, misceláneos, radiofrecuencia, electromecánicos y serñales.

POR FAMILIA. Cada grupo se encuentra dividido en varias familias por ejemplo el grupo de básicos cuenta con 21 familias entre las cuales están las resistencias, potenciómetros, capacitares, inductores, conectores etc.

COMPONENTE. Cada familia esta integrada por varios componentes, algunos componentes tienen variables a escoger como las resistencias las cuales se pueden escoger de entre 0.25 watts y 0.5 watts.

3.1.3 COLOCAR UN COMPONENTE

Para colocar un componente seleccionamos el grupo al cual corresponde desde la barra de herramientas de componentes, si ya esta abierta la ventana de selección de componente puede cambiar el grupo desde el recuadro **group** en la ventana. En este caso se muestra una imagen donde se a seleccionado una resistencia d 10KΩ a 0.25 watts.

Nótese la dirección del componente:

Master Database → Grupo de Básicos → Familia de Resistencias → Componente resistencia de 10KΩ a 0.25 watts

Observe el filtro con el que se pueden delimitar a las resistencias a aparecer en el recuadro, pueden ser Ω, KΩ, MΩ o simplemente todos los valores.

Después de seleccionar la componente presione OK y seleccione el lugar del plano a colocarla, una ves puesta puede cambiar la locación de la componente arrastrándola con el mouse.

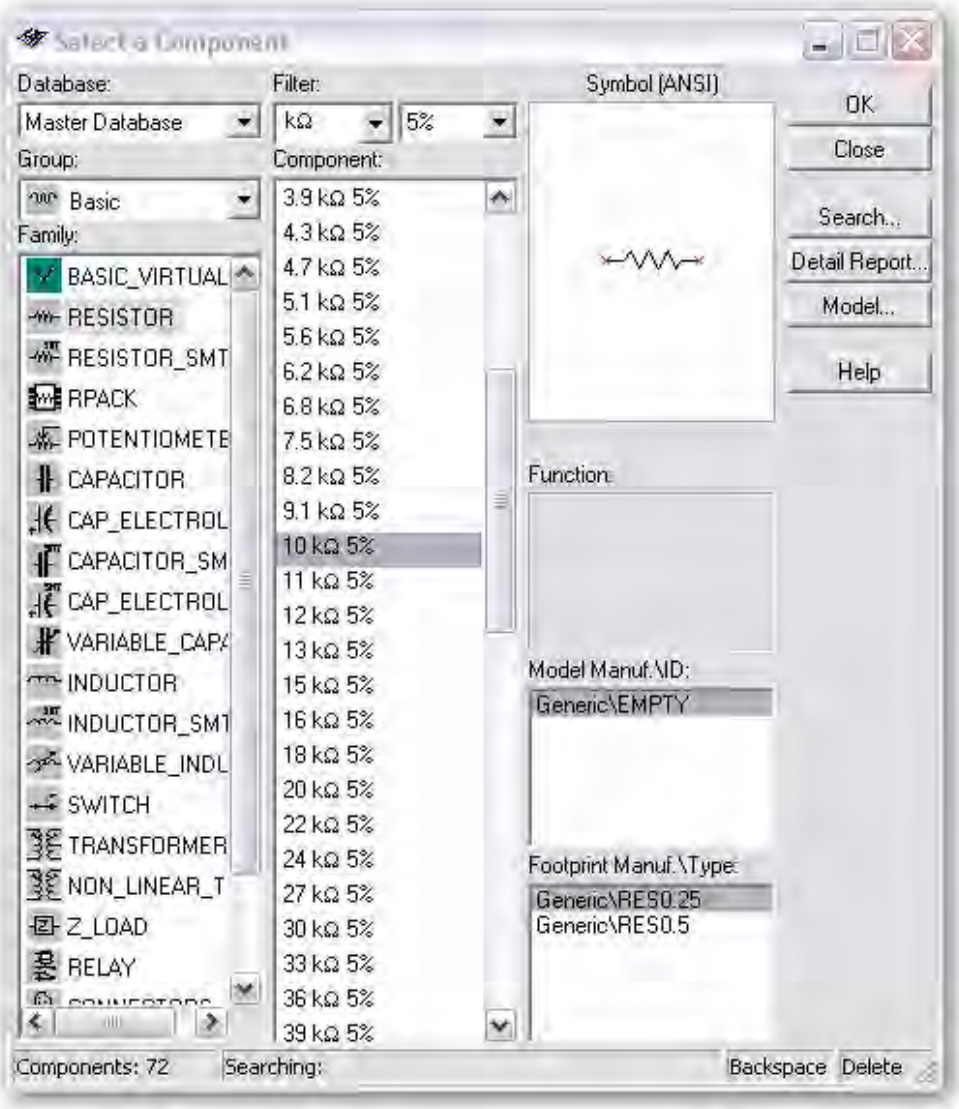


Figura 3.12: Selección de una resistencia.

3.1.4 CONECTAR COMPONENTES ENTRE SI

Para conectar dos componentes entre si solo tiene que seleccionar con el botón primario del mouse alguno de los pines de la componente y guiar la línea que aparece generando ángulos de 90° con cada presión del botón primario del mouse, para finalizar presione de nuevo el botón primario del mouse sobre el pin en donde se debe conectar.

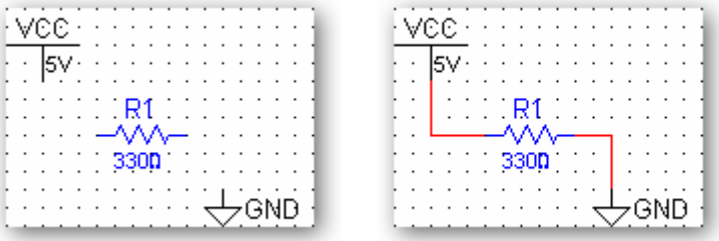


Figura 3.13: Conexión entre componentes.

3.1.5 CREACION DE UNA COMPONENTE

En este ejemplo se creara el microcontrolador que se ha venido trabajando puesto que no existe en la base de datos del Multisim 9.
Presione de entre la barra de herramientas el menú **Tool** y dentro de este la opción **Component Wizard**, con esto aparecerá la ventana de la figura 3.14 correspondiente al paso uno de la elaboración de un componente. Coloque el nombre de la componente, el nombre de autor si se desea, una breve descripción del funcionamiento y muy importante la opción **Layout only** ya que solo se desea a la componente para el diseño y no para la simulación.

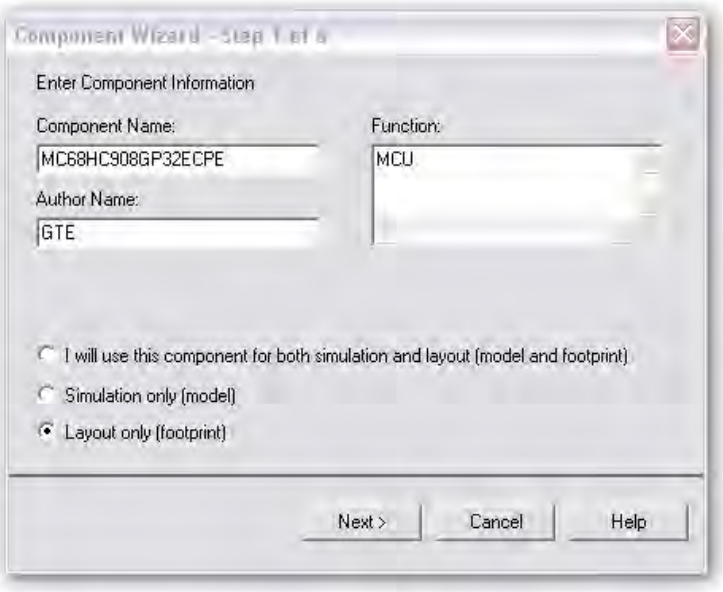


Figura 3.14: Creación de la componente paso 1.

Presione Next y deberá aparecer la siguiente ventana correspondiente al paso dos en donde colocará el número de pines de la componente.

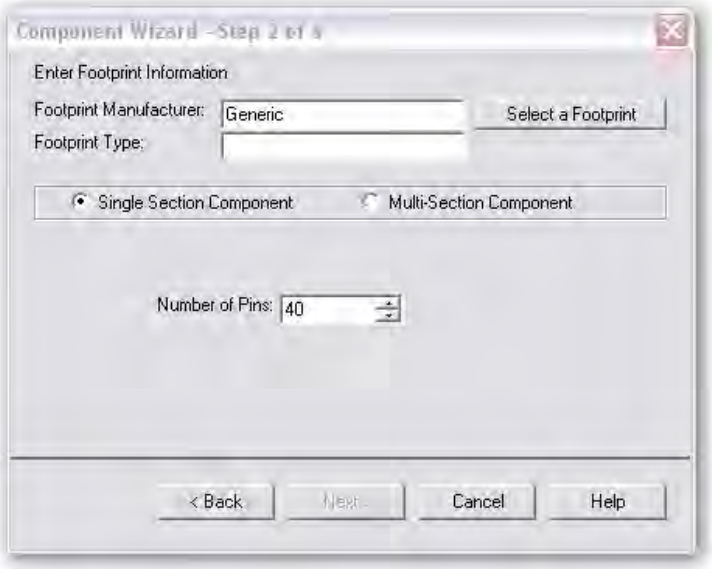


Figura 3.15: Creación de la componente paso 2.

Presione **select a Footprint** para escoger un tipo de encapsulado, para la localización mas rápida del componente presione el pequeño recuadro **PIN** para ordenar los encapsulados por número de pines.

En este caso es fácil la selección del encapsulado pero en otro tipo de piezas se debe tener mas cuidado ya que por ejemplo hay encapsulados tipo soic con el mismo número de pines pero que tienen diferente tamaño por eso consulte el manual de la componente que este diseñando para conocer exactamente el tipo de encapsulado ya que si comete un error en este punto podría darse el caso de que usted no se de cuenta asta que vaya a soldar el componente en el circuito impreso al percatarse que la huella del componente no corresponde con el que usted tiene, para evitar esto compare la huella de la componente de la cual tiene duda con la del PCB impreso en papel antes de traspasarlo a la tabla fenólica virgen.

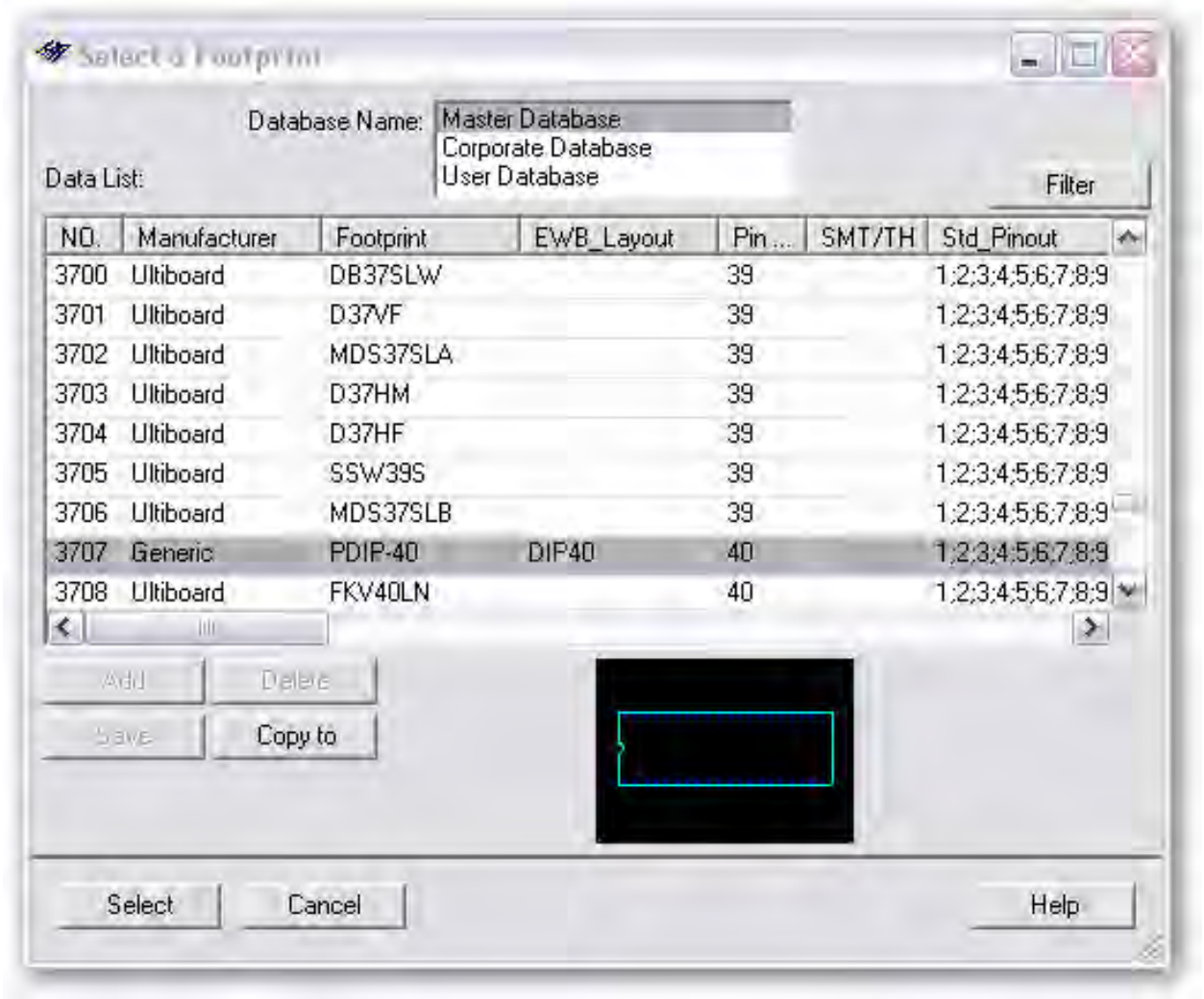


Figura 3.16: Selección de encapsulado.

Presione **select** y regresará a la pantalla anterior en donde se mostrará el encapsulado que seleccionó.

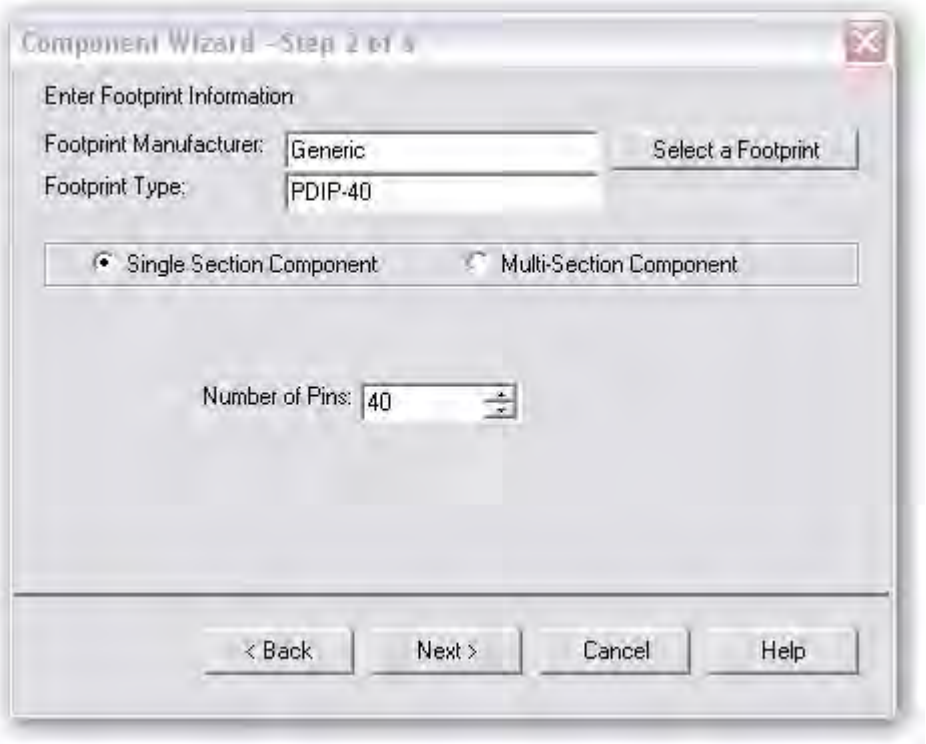


Figura 3.17: Fin del paso 2.

Presione Next y aparecerá la pantalla siguiente correspondiente al paso tres.

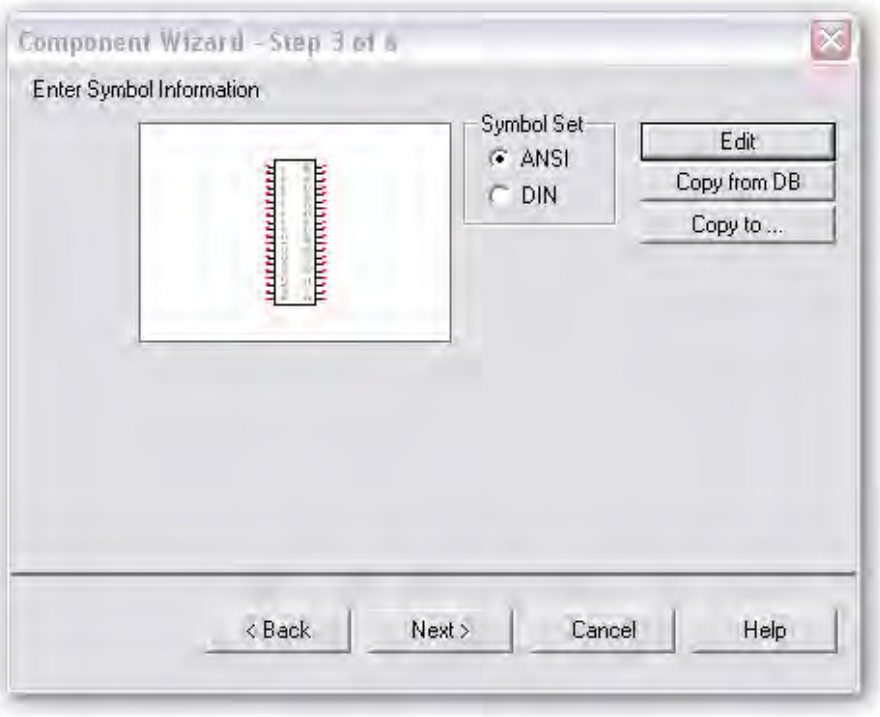


Figura 3.18: Creación del componente paso 3.

Presione **Edit** y aparecerá la ventana de edición de símbolo mostrada en la figura 3.19, aquí se debe agregar el nombre a cada pin aunque no es absolutamente necesario ya que con los números basta pero requerirá tener el manual del MCU abierto para consultar los pines lo que provocará que tarde mas en realizar su diseño, si el componente será muy utilizado por usted en diferentes proyecto tómese el tiempo de colocar el nombre de cada pin.
En el pequeño circulo rojo que esta en la parte superior de la figura 3.19 se marca la opción para redimensionar el símbolo de la componente ya que hay casos en los que los nombres de los pines son muy grande como para quedar dentro del contorno del símbolo, nótese que son dos cuadros los que hay que redimensionar, uno es el recuadro de los pines y otro es el contorno del símbolo, los dos deben quedar del mismo tamaño observe las figuras 3.20 y 3.21.

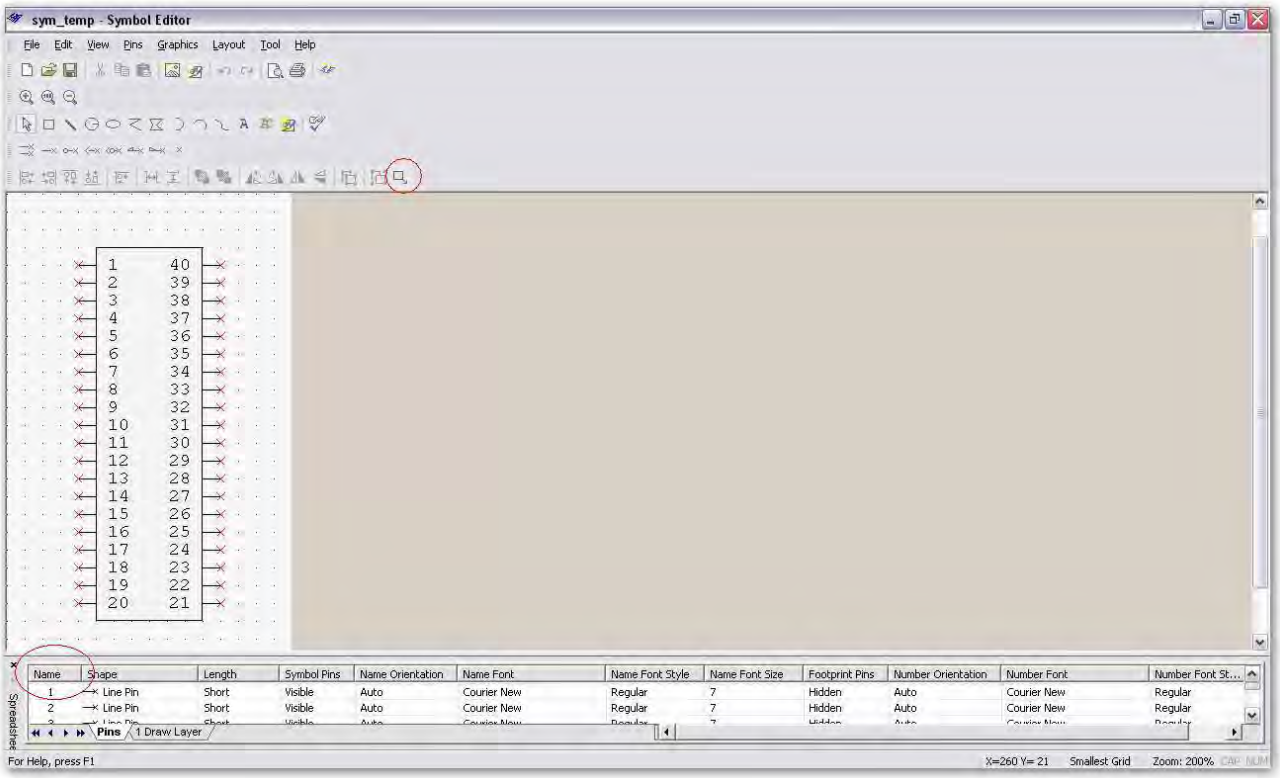


Figura 3.19: Editar símbolo.

La elipse que se encuentra en la parte inferior de la figura 3.19 marca la columna donde se cambian los nombres de cada pin. Cuando termine de nombrar los pines presione el menú **File** y la opción **save** esto con el fin de guardar los cambios realizados, a continuación presione de nuevo **File** y después la opción **exit** para regresar a la ventana del paso 3.

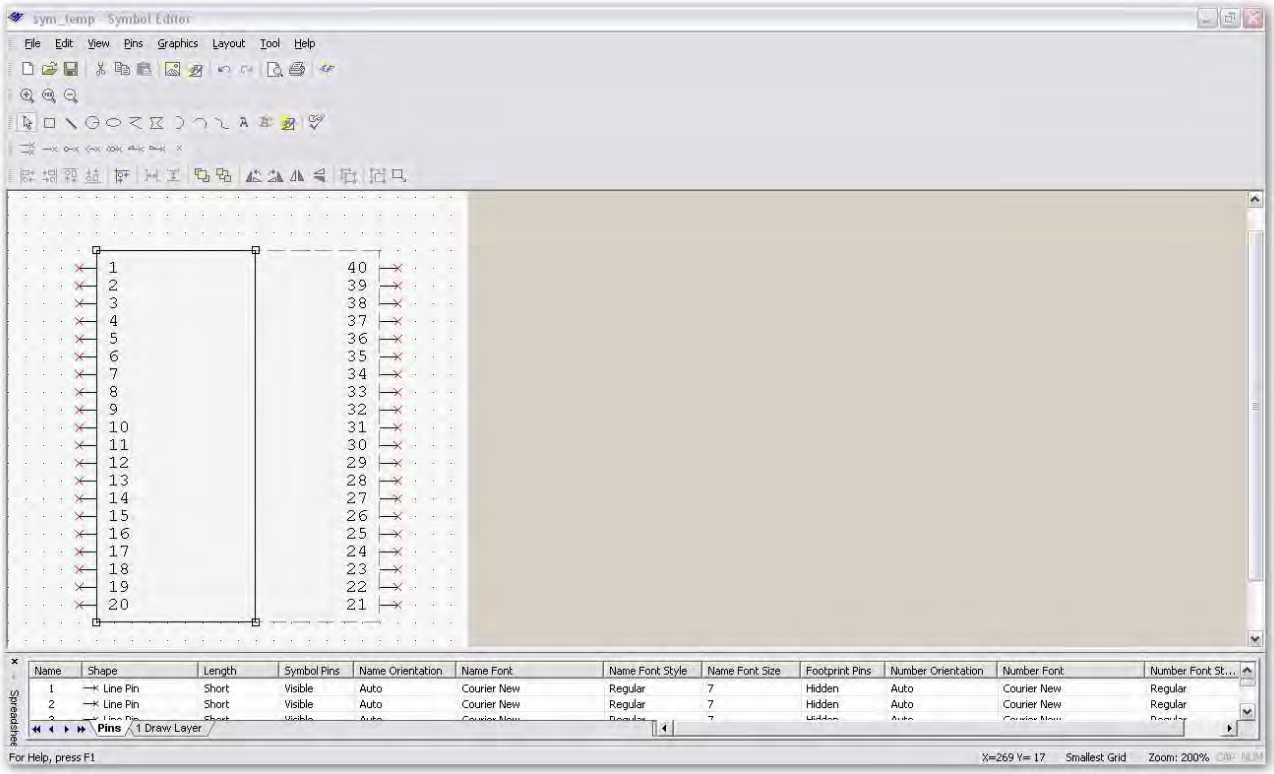


Figura 3.20: Redimensionar el recuadro de los pines.

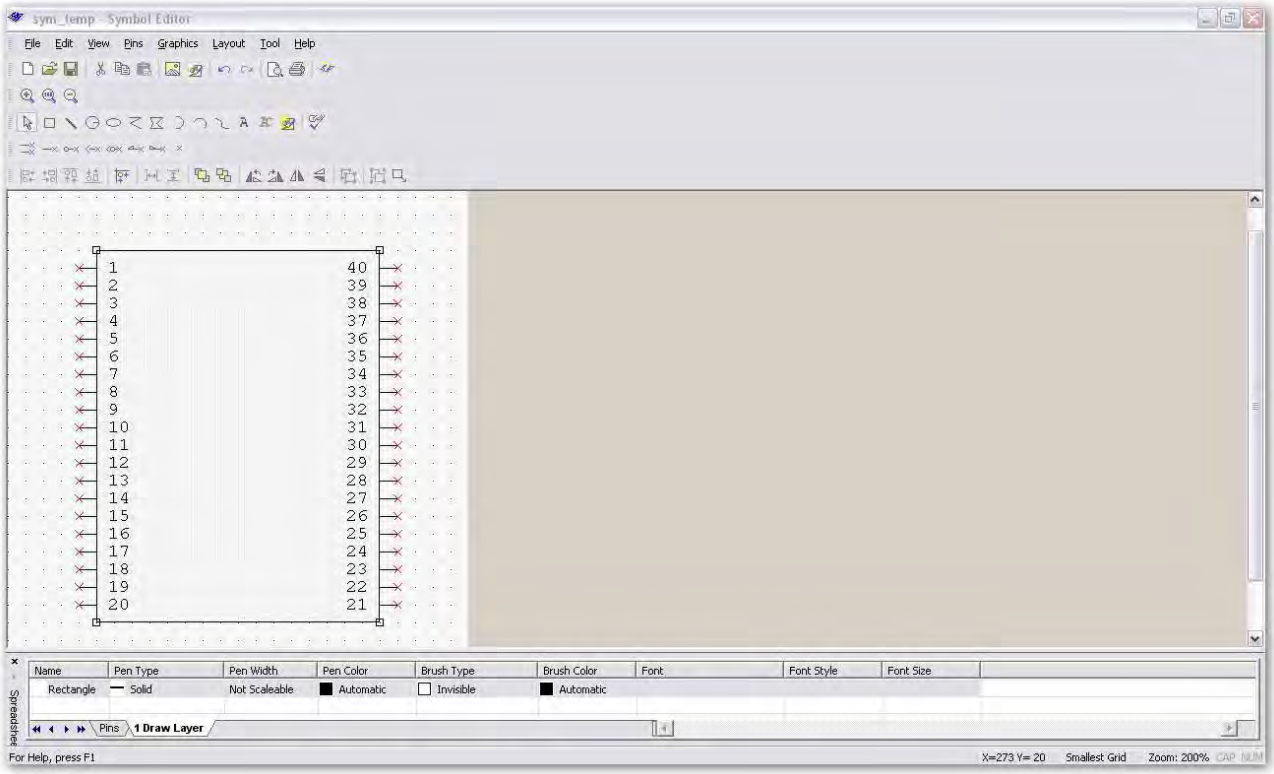


Figura 3.21: Redimensionar el contorno.

Debido al propósito de nuestra componente no es necesario hacer ninguna modificación en los pasos 4 y 5 por lo que solo se presiona el botón **Next**.

En el siguiente y último paso se define en donde se almacenará la componente por lo que se procede a abrir la carpeta **User Database.**

Figura 3.22: Almacenamiento del nuevo componente paso 6.

A continuación se selecciona el grupo al que pertenecerá la componente en este caso el de los microcontroladores, ya seleccionado se debe agregar una familia presionando el botón **Add Family,** a esta nueva familia la nombramos HC08, si se llegaran a crear mas MCU´s pertenecientes a esta familia ya no es necesario crear la familia si no solo seleccionarla.

Figura 3.23: Agregar una nueva familia.

NOTA: Se recomienda no mezclar componentes creadas con las componentes de la base de datos maestra, procure almacenar sus componentes en la USER DATABASE.

En la siguiente figura puede verse el resultado final ya plasmado en el espacio de trabajo, para encontrar la componente creada abra la ventana de selección de componente mostrada en la figura 3.11 y siga la ruta adecuada para la componente en cuestión.

- Base de datos: User Database.
- Grupo: Multi MCU.
- Familia HC08.

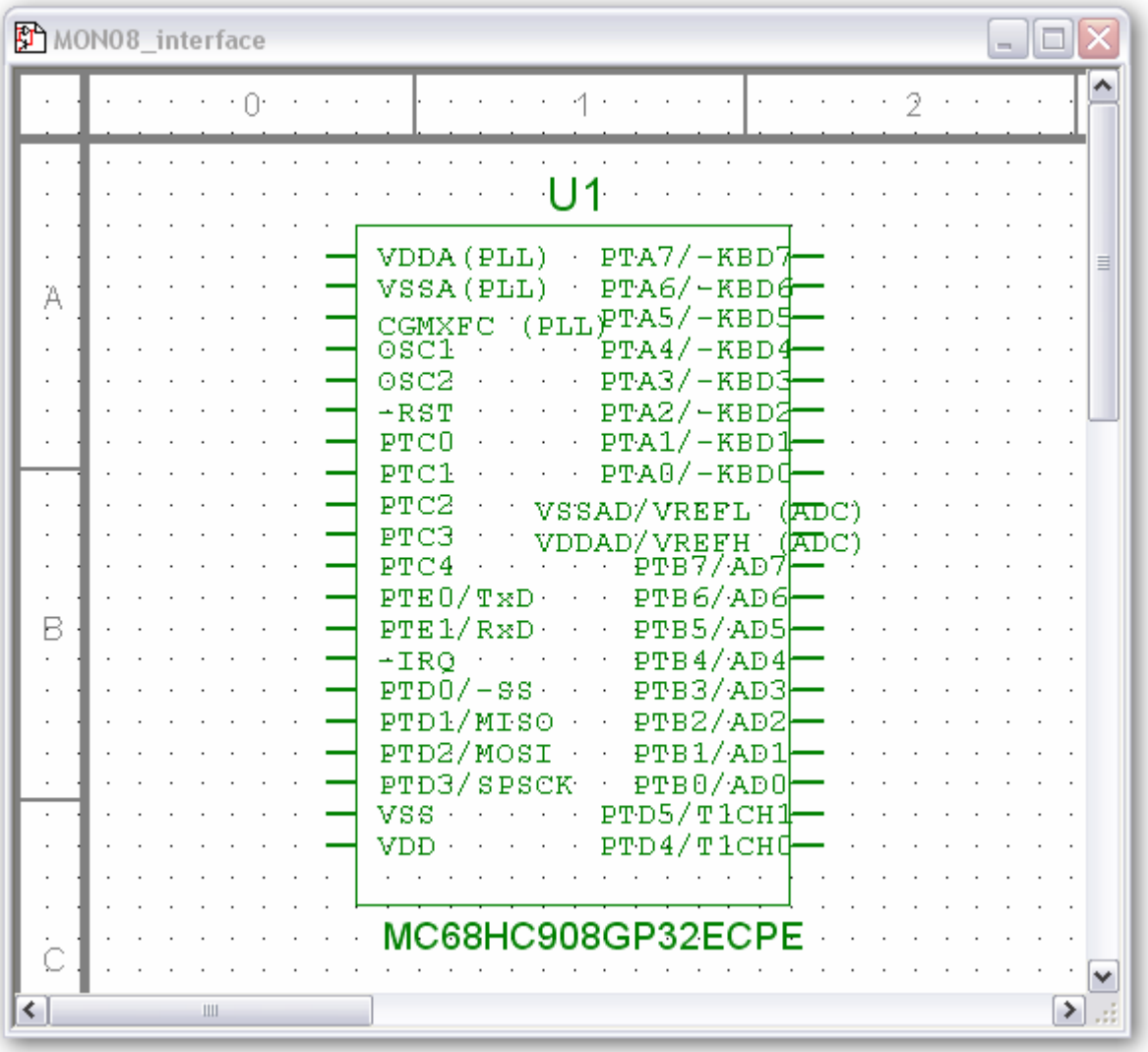


Figura 3.24: Componente finalizado.

3.2 DISEÑO DEL CIRCUITO DE UNA INTERFACE MON08----------

Antes que nada se debe hacer mención que cada componente se puede girar o invertir para esto debe presionar el botón secundario del mouse y seleccionar alguna de las opciones como rotar a la derecha o izquierda a 90° o voltear horizontal o verticalmente la componente.

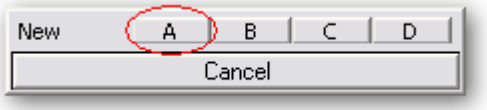
3.2.1 COLOCAR Y MODIFICAR TODOS LOS COMPONENTES

El primer paso es colocar en el plano todos los componentes a utilizar.

- **MAX232CPE (1)**: Se encuentra en el grupo Misceláneo digital en la familia transeivers lineales.

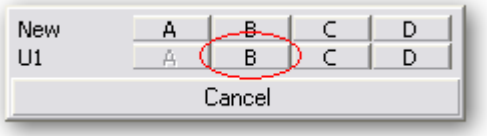
- **74LS125N (1):** Tenga la precaución de seleccionar dos compuertas pertenecientes al mismo integrado por que si selecciona dos compuertas pertenecientes a diferentes integrados el PCB se generará con dos integrados 74LS125N. La ubicación de esta compuerta es en el grupo TTL y dentro de la familia 74LS.

Cuando seleccione la primera compuerta aparecerá la siguiente ventana, presione el botón marcado.



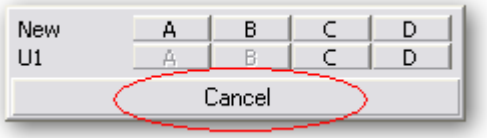
PASO 1

Coloque la compuerta en el área de trabajo y automáticamente aparecerá la siguiente ventana, seleccione el botón marcado.



PASO 2

En la siguiente ventana presione **cancel** puesto que ya no se necesitan más compuertas.



PASO 3

Si presiona alguna letra de la fila **New** estará llamando a alguna de las cuatro compuertas de un segundo circuito integrado, cada integrado cuenta con cuatro compuertas.

- **Cristal de cuarzo (1):** La base de datos del Multisim 9 no cuenta con un cristal de cuarzo de 4.9152 MHZ por lo que puede crear uno con el asistente de creación de componente o simplemente colocar el primero disponible en la base de datos y cambiarle las etiquetas a través de las propiedades. Los cristales se encuentran en el grupo Misceláneos dentro de la familia cristales. Una vez seleccionado el cristal valla a sus propiedades y modifique las opciones tal y como se muestran en las figuas.

- **Conector DB9 Hembra en 90°:** La base de datos del Multisim 9 cuenta con dos diferentes conectores DB9 en 90° uno es el DSUB9F y el otro es el DSUB9M las letra finales hacen referencia a **Male** y **Female** palabras en ingles de macho y hembra respectivamente.

NOTA: Se ha detectado un error serio en este aspecto en el que si se escoge un conector hembra en el momento de generar el PCB aparecerá la huella de un conector macho y viceversa, por esta causa si se quiere un PCB con un conector hembra como se muestra en la lista de componentes se deberá colocar en el esquemático un conector macho es decir un DSUB9M.

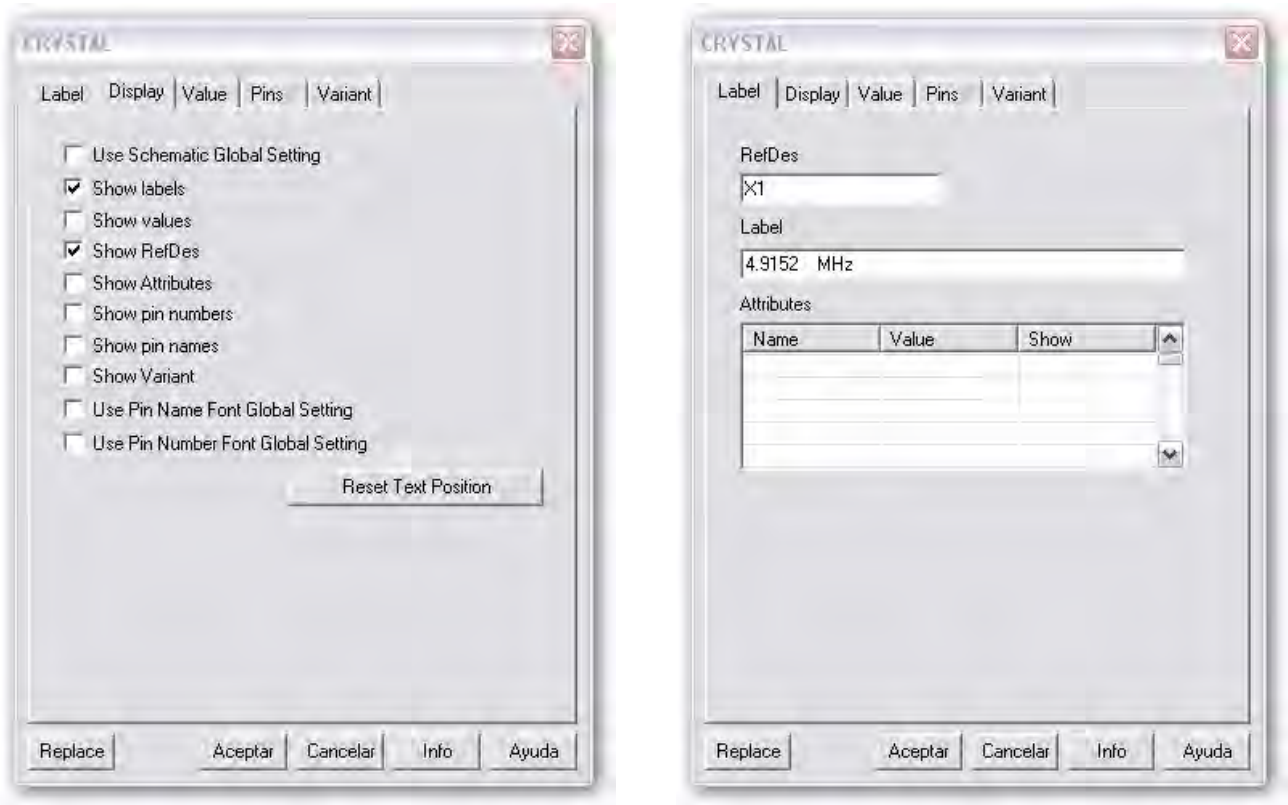


Figura 3.25: Cambio de etiqueta.

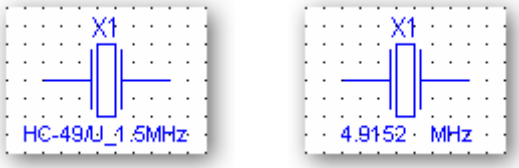


Figura 3.26: Antes y después.

- **Diodo zener de 8.5v (1). Puede usar el siguiente diodo zener con voltaje por arriba de 8.5v.**
- **Capacitor electrolítico de 10uF (5).**
- **Capacitor de 22pF (2).**
- **Capacitor de 0.1uF igual a 100nF (1).**
- **Resistencia de 10MΩ (1).**
- **Resistencia de 10KΩ (1).**
- **Resistencia de 1KΩ (1).**
- **Header de 6 pines (1).**
- **Header de 2 pines (1).**

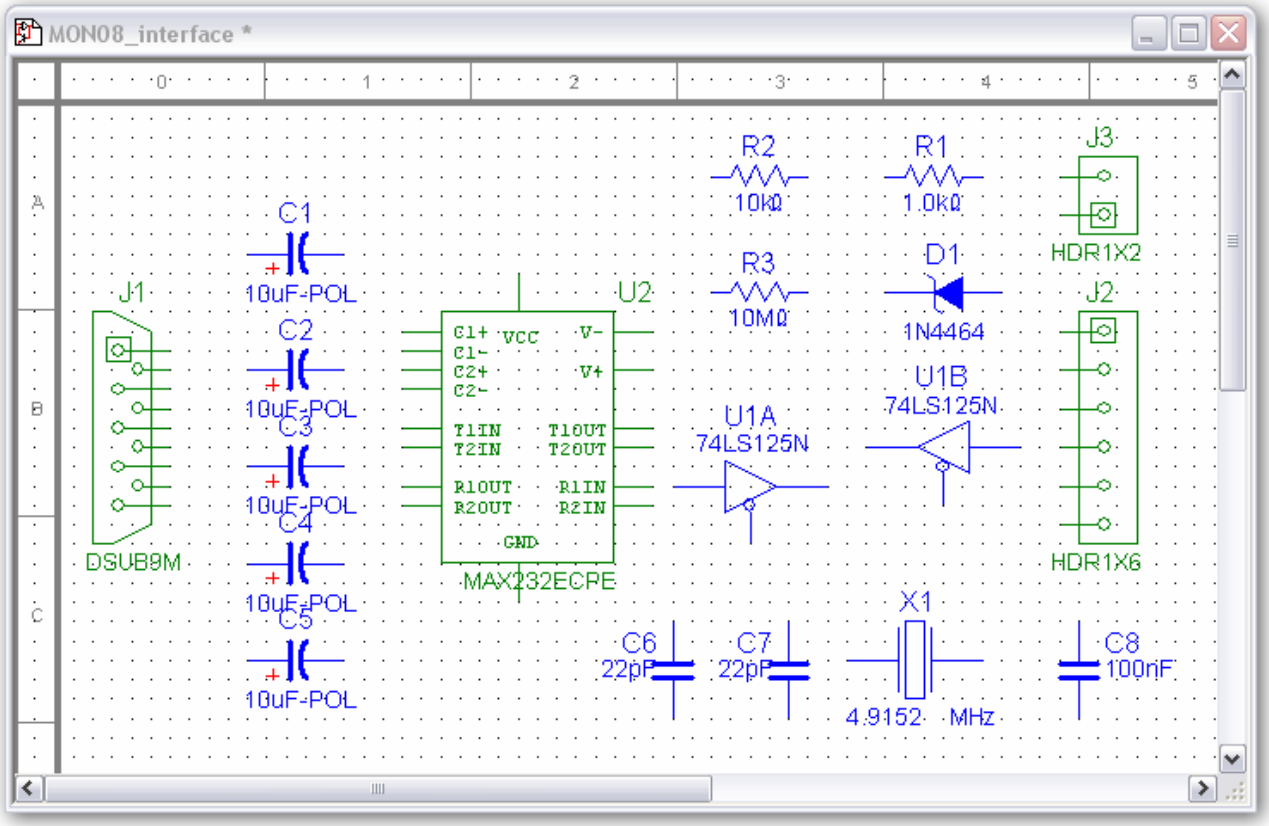
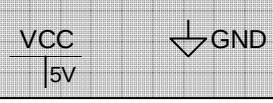


Figura 3.27: Componentes del Mon08.

NOTA: Como dispositivo de polarización debe utilizarse la fuente VCC de 5v ya que esta es la fuente necesaria para los circuitos TTL, nótese que las compuertas 74LS125N no tienen conexiones de polarización (VCC y GND) ya que estas están implícitas en el chip, el programa Ultiboard enrrutará automáticamente el pin VCC del integrado 74LS125N con la fuente VCC y su pin GND con la parte negativa de la fuente también llamado GND. Si usted coloca alguna fuente como VDD o VEE el programa no enrrutara el pin VCC del chip lo mismo pasara con el pin GND si usted utiliza GROUND por con siguiente debe de utilizar GND. Enseguida se muestra la polarización que debe utilizar:



3.2.2 UNIR TODOS LOS COMPONENTES

Una ves que todas las piezas a utilizar están en el plano se procede a conectarlas entre si, el manual del MCU muestra como conectar al circuito MAX232CPE mostrando los números de pines del integrado cosa que no se muestra en el Multisim 9, para entender esto compare el MAX232CPE de la figura 3.27 con el de la figura 3.28.

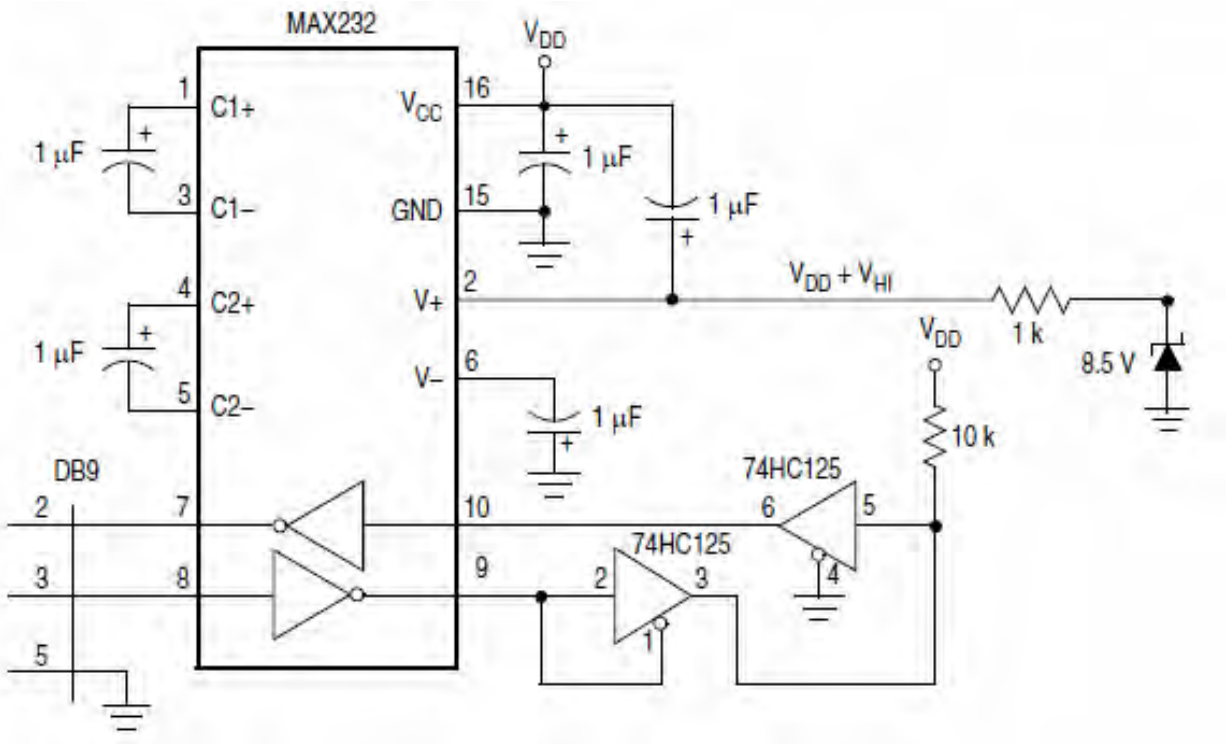


Figura 3.28: Conexión entre pines MAX232CPE.

Si aun no esta seguro de la configuración de pines y no cuenta con el manual de la componente puede entrar a las propiedades del chip en cuestión presionando el botón derecho del mouse sobre ella y después seleccionando la opción **propiedades**, en la figura 3.29 se muestra la ventana de propiedades de la componente.

Figura 3.29: Propiedades del MAX232CPE.

Presione el botón **Edit Footprint** con lo que aparecerá la siguiente ventana en donde podrá verificar el No de pin del encapsulado que corresponde a cada pin del símbolo.

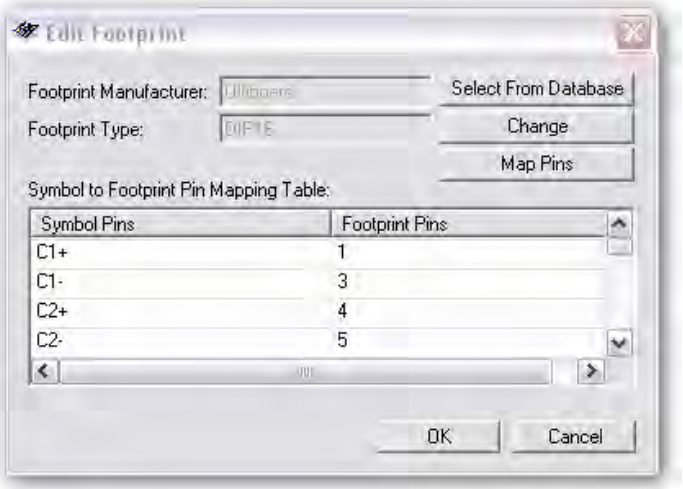


Figura 3.30: Correspondencia de pines entre símbolo y encapsulado.

NOTA: Procure no modificar estas opciones ya que si lo hace se modificara la relación de pines del encapsulado con pines del símbolo lo que generara un error en la componente por lo que seguramente el dispositivo no funcionará.

Ya conectados todos los componentes debería obtener algo parecido a lo mostrado en la figura 3.31.

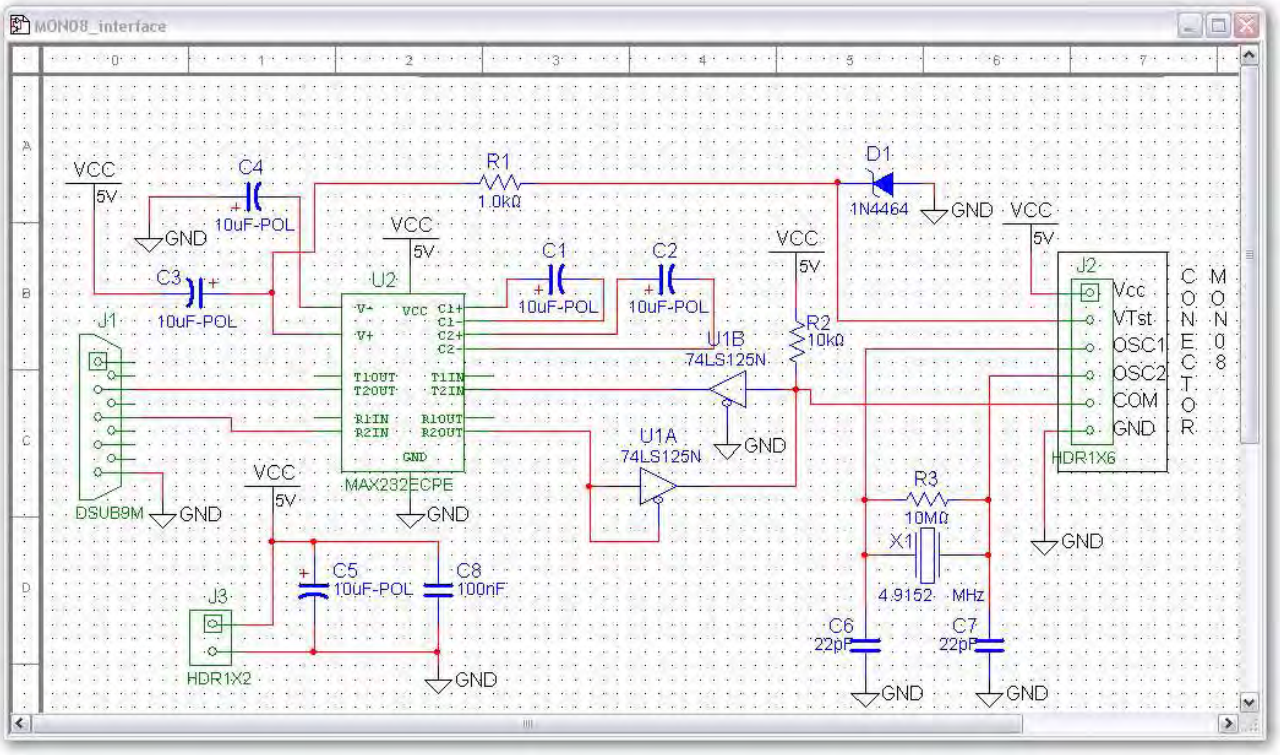


Figura 3.31: Conexiones completas.

Solo queda guardar el circuito, la primera ves que haga esto presione en la barra de herramientas el botón **File** y después la opción **Save As** para después seleccionar una ubicación apropiada para su almacenamiento, también se debe escoger un nombre de proyecto apropiad en este caso podría ser Mon08_ Interfase. Los guardados posteriores puede hacerlos presionando el botón **save**: 

3.2.3 ARCHIVO .ms9

Este es el archivo de un proyecto Multisim 9 al dar doble clic sobre el se abrirá el programa Multisim 9 con el circuito tal y cual lo dejó justo después de ser guardado. Recomiendo instalar estos programas sobre plataforma Windows XP ya que sobre Windows vista los archivos Multisim y Ultiboard no se ligan automáticamente a sus respectivos programas debido a conflictos en los registros

Figura 3.32: Archivo .ms9.

3.2.4 TRANSPORTAR DE MULTISIM A ULTIBOARD

Ya terminado el circuito se procede a transportarlo al programa ultiboard 9, para esto es necesario presionar en la barra de herramientas el botón **Transfer** y dentro del menú emergente que aparece seleccionar **Transfer to Ultiboard**, por lo que deberá aparecer una ventana de guardado en donde se selecciona la ubicación donde se almacenará el archivo .NET, una ves seleccionada la ubicación presione guardar y aparecerá la ventana de las reglas básicas del diseño, que en lo particular yo ajusto tal y como se ve en la figura 3.33 líneas de 30 milésimas de pulgada de grosor y espaciado de 10 milésimas de pulgada. Presione OK para continuar, en la siguiente ventana no modifique nada ya que podría eliminar conexiones por lo que solo presione OK.

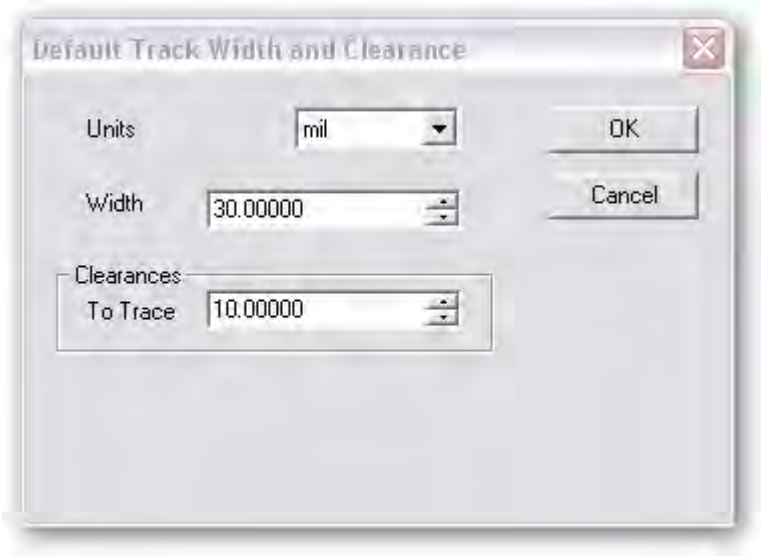


Figura 3.33: Reglas de diseño.

3.2.5 ARCHIVO .EWNET

El archivo .EWNET se podría decir que es un archivo virgen que le indica al programa Ultiboard 9 los componentes y conexiones presentes en el circuito original diseñado en Multisim por lo que cualquier cambio que haga al realizar el PCB no se vera reflejado en este archivo si no en el que se menciona mas adelante.
La función de este archivo es básica cuando se requiere realizar un diseño que sea de lo más compacto y eficiente, ya que si no le agrada su diseño final puede volver a empezarlo utilizando este archivo dando doble clic sobre el y modificando así las reglas de diseño básicas nuevamente.

Figura 3.34: Archivo .EWNET.

3.2.6 ARCHIVO .EWPrj

En este archivo se guardan todas las modificaciones hechas al archivo .EWNET es decir es el diseño del PCB.

Figura 3.35: Archivo .EWPrj.

3.3 ENTORNO ULTIBOARD 9 ----------

En la figura 3.36 puede verse el entorno de trabajo del programa Ultiboard 9, este cuenta con una gran cantidad de herramientas para la creación del PCB, estudiarlas todas seria muy extenso por lo cual se delimitará el estudio a las mas importantes. La figura muestra el circuito MON08 que se diseño en Multisim 9 transportado al Ultiboard 9.

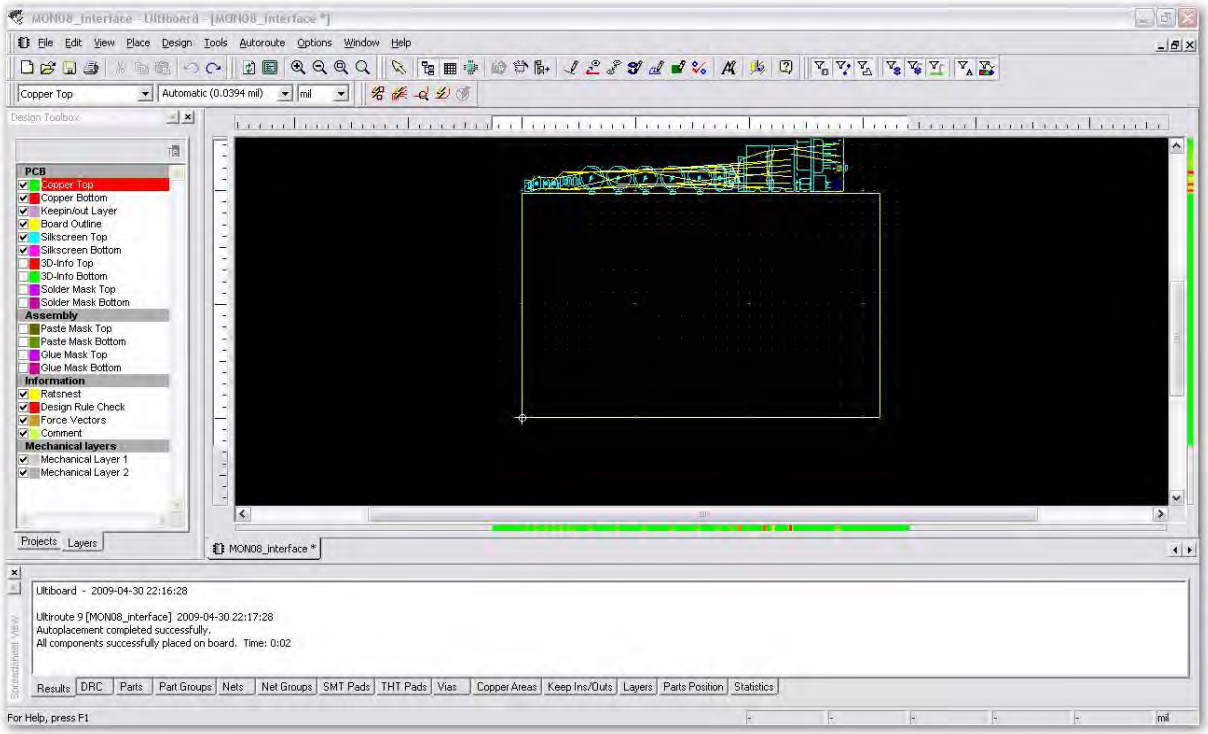


Figura 3.36: Entorno Ultiboard 9.

3.3.1 SPREADSHEET VIEW

Si esta ventana no es visible presione el botón secundario del mouse sobre la barra de herramientas del Ultiboard 9 y selecciones **Spreadsheet View** ahora solo desplace la ventana a la posición que más le agrade.
Esta ventana tiene una gran cantidad de pestañas en las cuales pueden mirarse conexiones, grupos de conexiones, errores, resultados de análisis, partes etc.

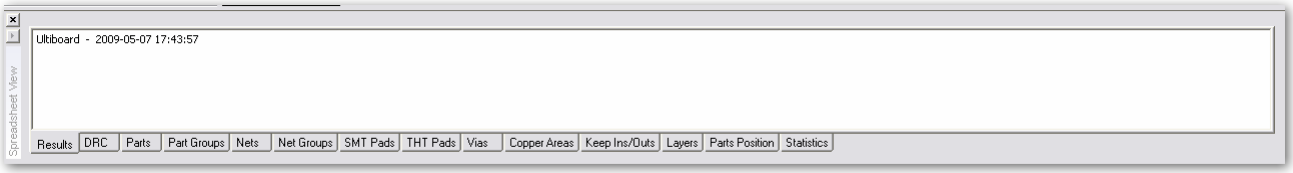


Figura 3.37: Spreadsheet View.

En esta ocasión centraremos nuestra atención en la pestaña llamada **Layers**, esta pestaña se encarga de determinar cuantas capas estarán implementadas en nuestro PCB, pueden estar presentes dos o cuatro capas, en la imagen solo se muestran dos capas.

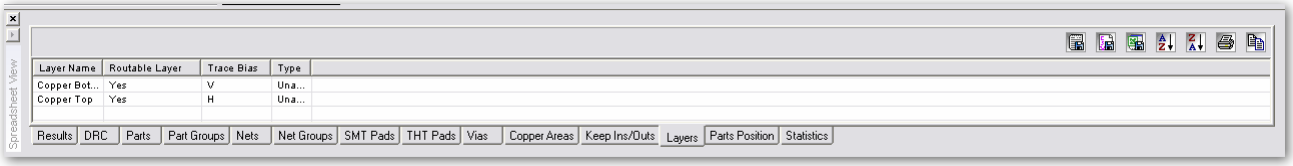


Figura 3.38: Pestaña Layers.

Hacer un diseño de dos, tres o cuatro capas es muy complicado por lo que se limitara el diseño a solo una capa, por lo que es necesario cada que realice un proyecto habilitar solo la capa inferior y deshabilitar las demás. Siempre que inicie un proyecto todas las capas estarán habilitadas.

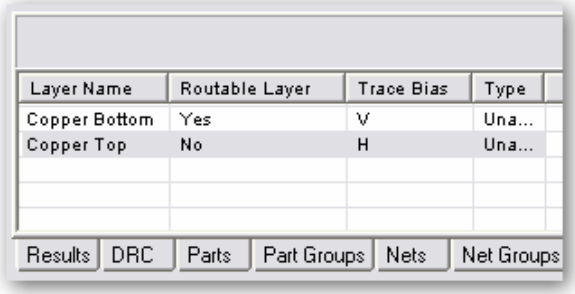


Figura 3.39: Capa inferior habilitada.

3.3.2 DESIGN TOOLBOX

La ventana llamada **Design Toolbox** en su pestaña **Layer** y los pequeños recuadros del lado izquierdo nos ayudan a hacer traslucidas o invisibles partes del diseño con el fin de simplificar la imagen del circuito cuando este se encuentre muy saturado. Pueden

modificarse los vectores de posición de las piezas (café), las líneas de conexiones faltantes (amarillo) el contorno de los componentes (azul), las pistas de las capa superior (verde), las pistas de la capa inferior (rojo) etc. Yo suelo únicamente desaparecer los **Force Vectors** (cafe) ya que pueden llegar a ser incómodos.

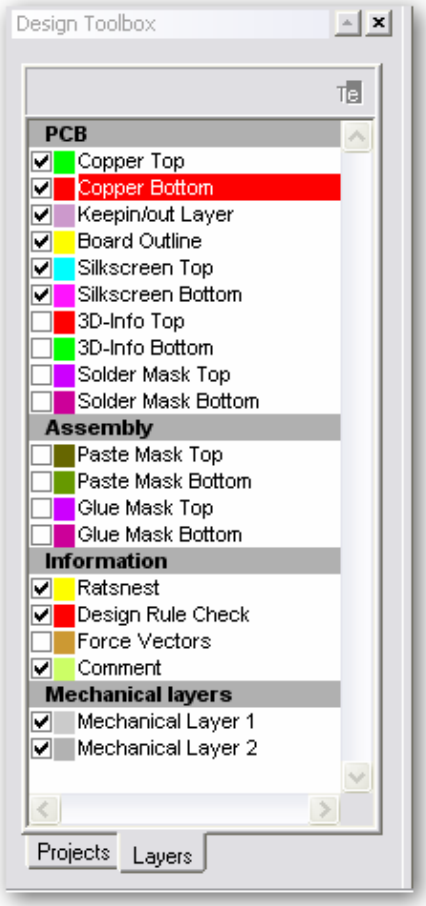


Figura 3.40: Capa inferior habilitada.

3.3.3 BARRA DE HERRAMIENTAS AUTOROUTE

Esta barra se encarga de los procesos automáticos del enrutamiento de las pistas.

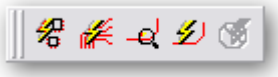


Figura 3.41: Herramientas Autoroute.

- Botón para el colocado automático de las componentes. 

Se recomiendo acomodar las componentes manualmente. Puede manipularse el espaciado automático de las componentes con la ventana **Routing Options,** la cual puede visualizarse presionando el botón **Routing** dentro de la barra de herramientas principal y dentro de este presione **Ultiroute Options.**

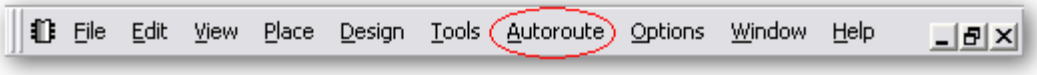


Figura 3.42: Barra de herramientas.

Seleccione la pestaña **Autoplace,** modifique la opción marcada en la figura para ajustar el espaciamiento automático, se recomienda un espaciado de entre 100 a 150 milésimas de pulgada.

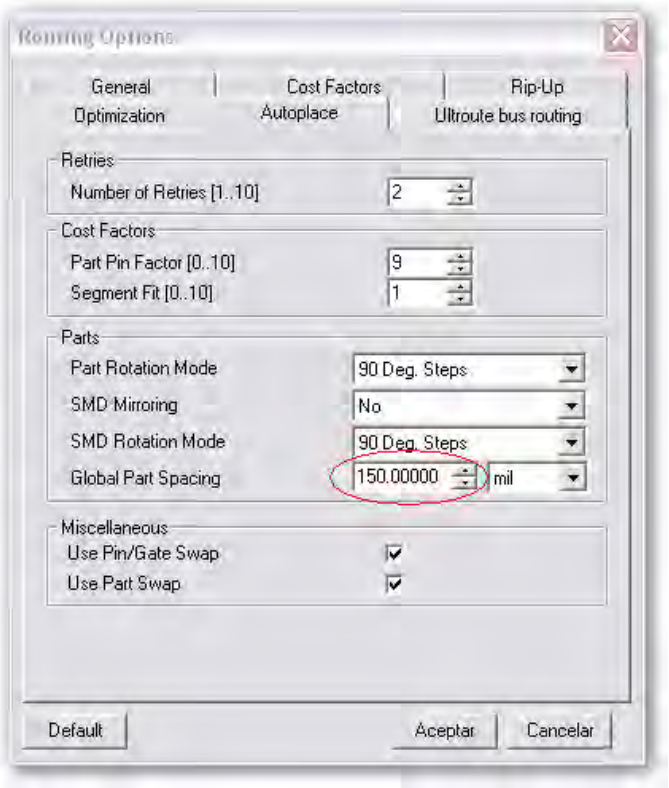


Figura 3.43: Routing Options.

- Botón para el enrrutado automático de las pistas 

Si el programa no encuentra una solución al enrrutado de las pistas serás necesario detener el proceso manualmente, las conexiones que no sean completadas tendrá que hacerlas manualmente.

- Botón para detener el enrrutado 

3.3.4 BARRA DE HERRAMIENTAS MAIN

Esta barra posee botones que se utilizaran todo el tiempo, algunos modifican la función del mouse y otros el plano de trabajo

Figura 3.44: Barra de herramientas Main.

- Al presionar este botón  se podrán arrastrar las componentes o simplemente seleccionar los pads para ser modificados, modificar pistas o seleccionarlas para poder eliminarlas si son incorrectas, etc.

- Al presionar este botón  se podrán dibujar pistas manualmente ya sea en la capa superior o en la capa inferior dependiendo de la que se tenga seleccionada en la ventana **Design Tollbox**. En la figura siguiente puede observarse que la capa inferior está habilitada, es decir que el tipo de pista que se dibuje dependerá enteramente de lo seleccionado en la ventana de la figura.

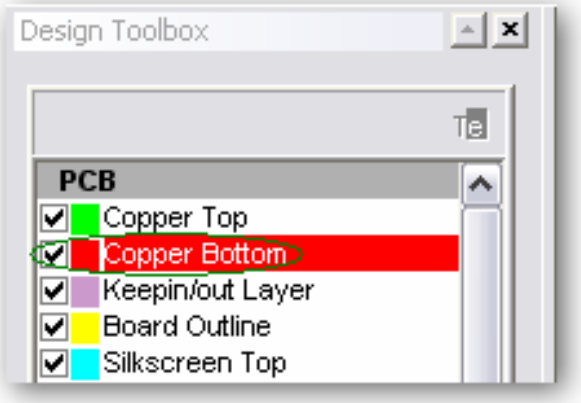


Figura 3.45: Selección de pista a dibujar manualmente.

- Con este botón  podrá generarse el **Power Plane** que no es mas que el relleno de espacio entre pistas, el power plane puede estar ligado a cualquier elemento por lo regular este elemento es GND.

- Con este botón  se podrá introducir texto en cualquiera de las capas inferior o superior del circuito dependiendo de la capa seleccionada en la figura 3.45. Al presionar este botón aparecerá la siguiente ventana en la cual se podrá escribir el texto y el tamaño del mismo.

Value: Texto a imprimir en el circuito
Windows Font: Para elegir el tipo de letra
Height: Establece el tamaño de la letra
Mirror: Si activa esta casilla el texto se verá invertido pero es necesario de esta manera para que se imprima correctamente en la tarjeta fenólica virgen.
Layer: Selecciona la capa en la que se quiere poner el texto por lo regular será la inferior o Copper Bottom.

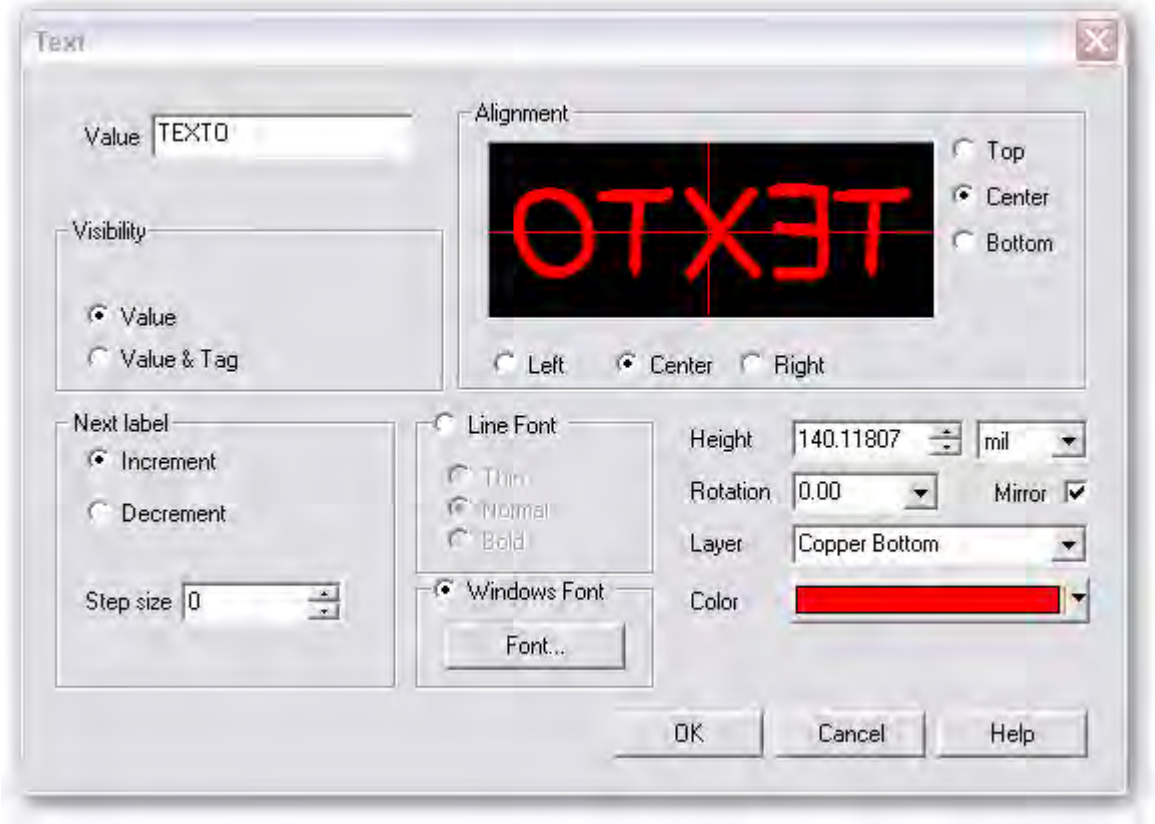


Figura 3.46: Introducción del texto.

- Con este botón  podrá visualizar el diseño final del PCB en una imagen tridimensional, esta imagen será una aproximación muy acertada del aspecto final del circuito impreso.

NOTA: Cuanto transporte un circuito de Multisim a Ultiboard todas las componentes estarán dibujadas con un contorno azul ya sean de montaje por pines o de montaje superficial lo que quiere decir que se encuentran colocadas en la parte superior de la tarjeta, para las componentes de montaje por pines esto esta bien pero para las componentes de montaje superficial no lo es así ya que deben encontrarse en la cara inferior de la tarjeta para ser soldadas, para cambiar a las componentes de montaje superficial a la cara inferior de la tarjeta debe presionar sobre ellas el botón secundario del mouse e ir al menú **Orientation** y seleccionar la opción **Swap Layer,** el indicativo de que la componente se encuentra en la cara inferior de la tarjeta es su contorno el cual se torna morado.

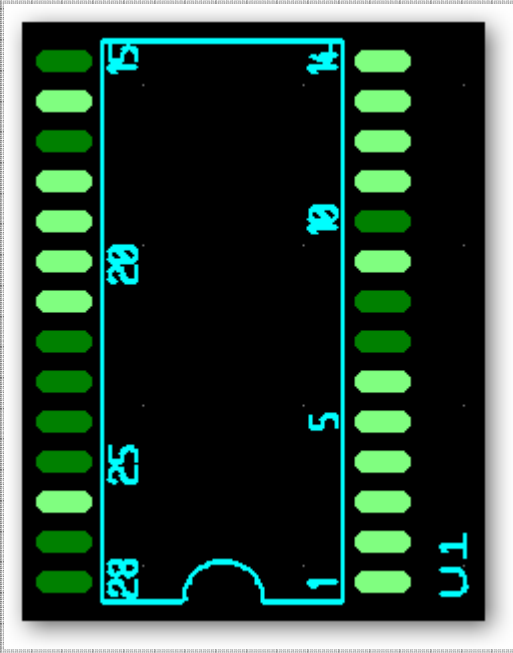


Figura 3.47: Componente de montaje superficial en la cara superior INCORRECTO.

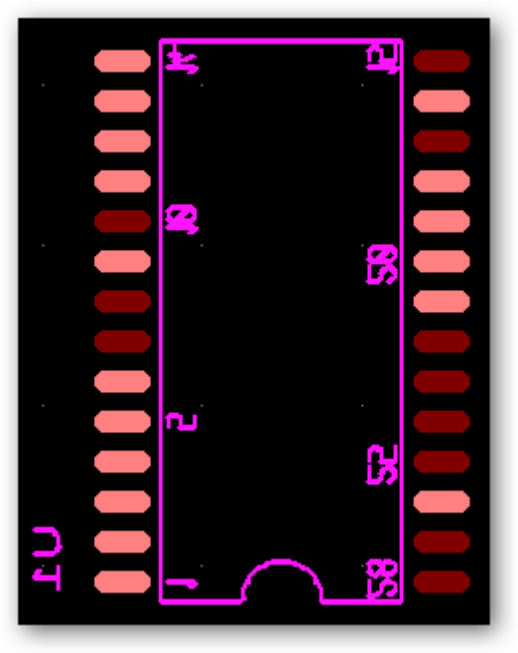


Figura 3.48: Componente de montaje superficial en la cara inferior CORRECTO.

3.4 DISEÑO DEL PCB DE UNA INTERFACE MON08----------

En esta ocasión se proseguirá con el diseño de la interfase Mon08 el cual se estudio en el tema 3.2, al final de este se explica como transportar el circuito diseñado de Multisim a Ultiboard con lo que al final del traslado y la estipulación de reglas se tendrá la siguiente ventana.

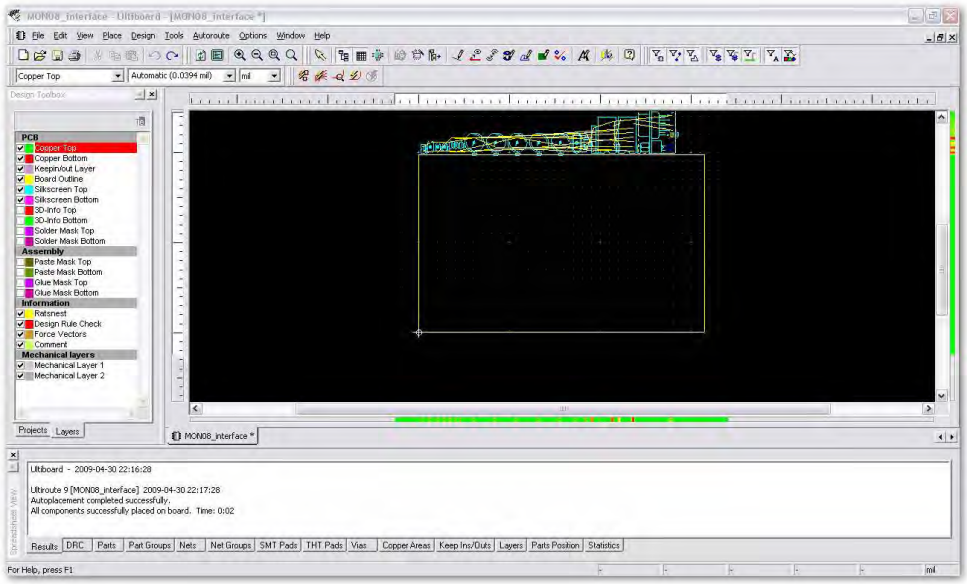


Figura 3.49: Diseño del PCB Mon08.

3.4.1 AJUSTES INICIALES

PRIMER PASO: El primer paso es activar y desactivar las capas pertinentes para nuestro diseño por lo que recurrimos a la ventana **Spreadsheet View** en su pestaña **Layers** en donde habilitamos la capa inferior (Coper Bottom) y deshabilitamos la capa superior (Top Bottom). Debe verse tal y como se muestra en la figura, si al mirar esta pestaña ve alguna otra capa (capas interiores) deshabilítelas, en resumen solo la capa inferior debe estar habilitada.

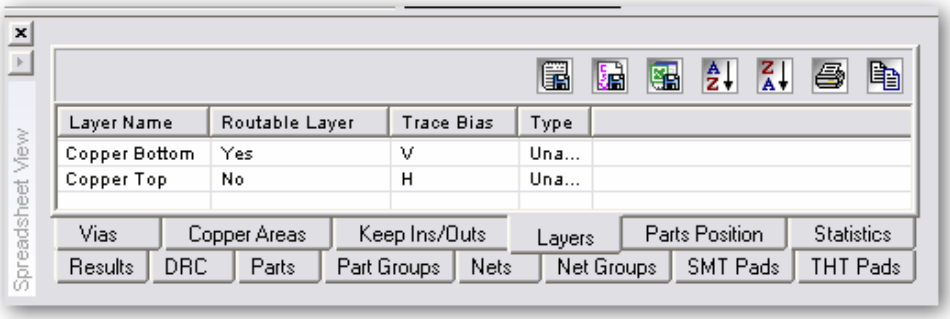


Figura 3.50: Activar solo capa inferior.

SEGUNDO PASO: Se recomienda apagar los vectores de posicionamiento de las componentes ya que pueden llegar a ser molestos en diseños muy saturados, la imagen muestra la casilla desactivada.

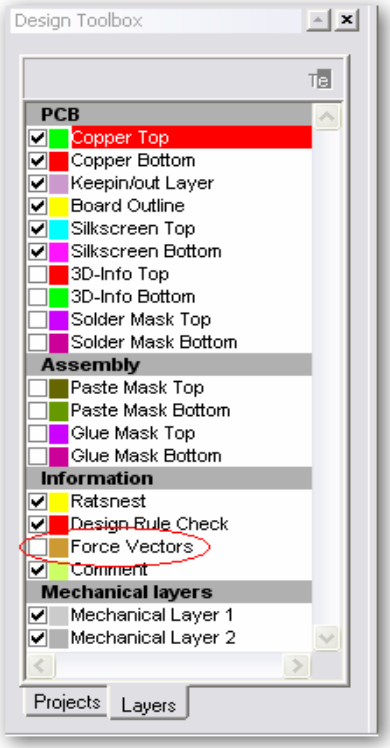


Figura 3.51: Deshabilitar Force Vectors.

TERCER PASO: En Ultiboard el formato inicial de los pads de las componentes es decir donde se tiene que perforar la tarjeta para soldar las partes esta diseñado para que una maquina muy precisa haga el trabajo pero en nuestro caso el trabajo será hecho a mano por lo que recomiendo agrandar o reducir (dependiendo cual sea el caso) cada uno de los pads de todas las componentes a un estándar que he definido para obtener una mayor facilidad al momento de fabricar la tarjeta, el cual es hacer que todos los pads tengan como medida de diámetro interior 35 milésimas de pulgada y como diámetro exterior 80 milésimas de pulgada. La imagen muestra una resistencia con los pads estándar y una con los pads modificados respectivamente.

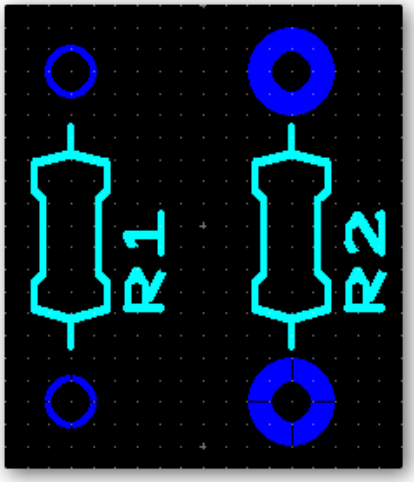


Figura 3.52: Pads originales y pads modificados respectivamente.

Para cambiar el tamaño de los pads pulse doble clic sobre el pad cuidando que la componente no esté seleccionada (observe la figura 3.53) ya que si es así entrara a las opciones de la componente y no podrá modificar el tamaño del pad, deseleccione la componente dando clic en cualquier parte del plano.

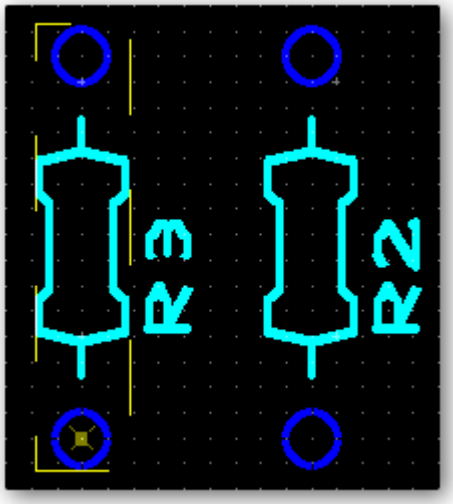


Figura 3.53: Componente seleccionada y no seleccionada respectivamente.

Ya estando la componente deseleccionada pulse doble clic sobre el pad y aparecerá la siguiente ventana, modifique tal y como se muestra lo marcado en la imagen. Repita este procedimiento en cada uno de los pads de cada una de las componentes.

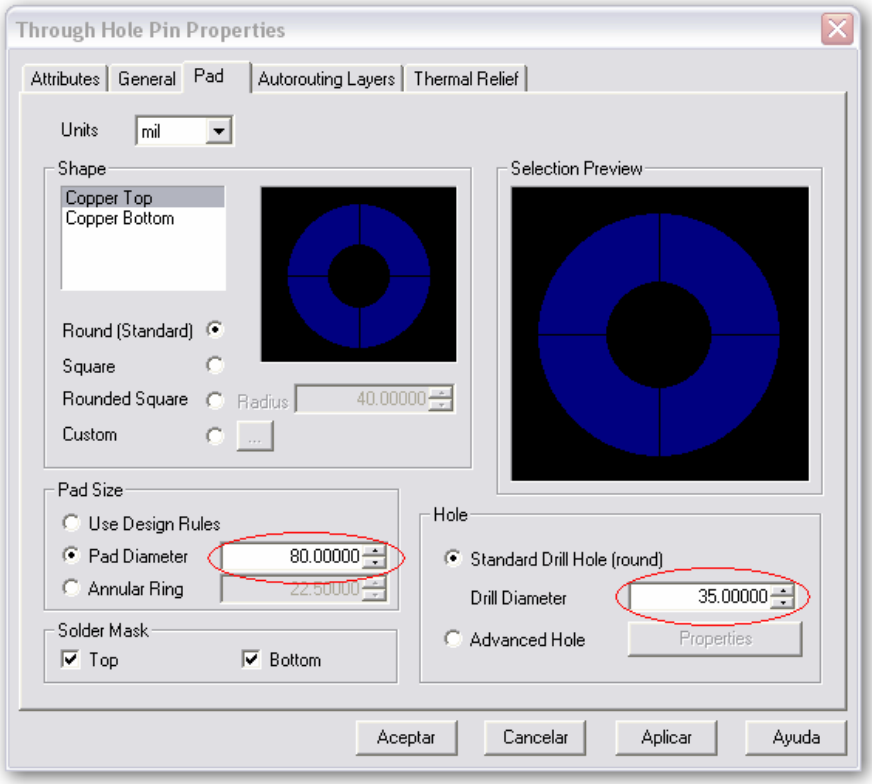


Figura 3.54: Modificaciones del pad.

3.4.2 EMPLAZAMIENTO DE LAS COMPONENTES

La siguiente etapa del diseño es acomodar las componentes en el plano y distribuirlas de tal manera que componentes que tengan muchas conexiones entre si estén dispuestas cerca una de otra, también es necesario cambiar de lado de la tabla a las componentes de montaje superficial. Las componentes deben permanecer dentro del cuadro amarillo, este representa el perímetro de la tarjeta.

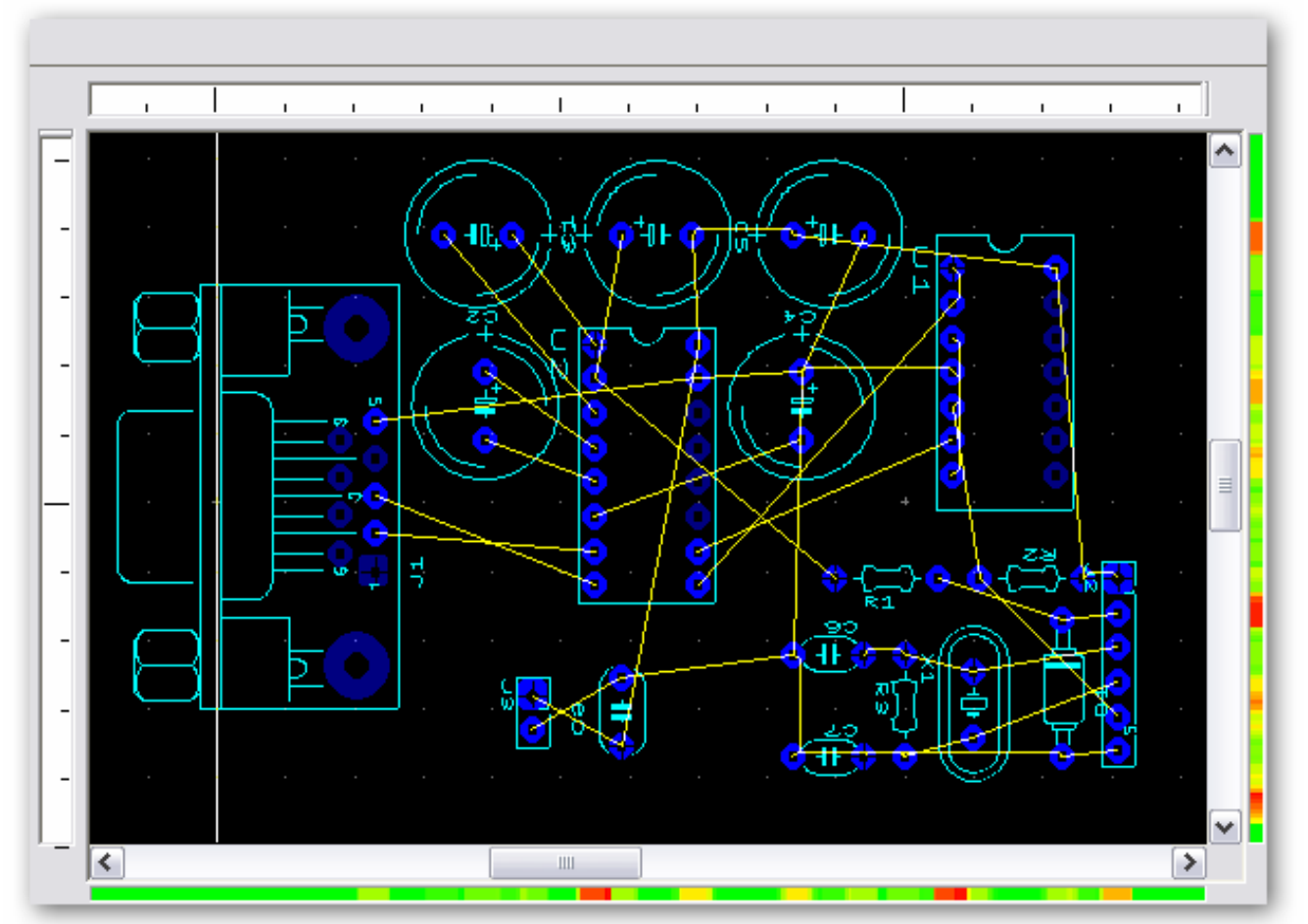


Figura 3.55: Distribución de las componentes.

3.4.3 ENRRUTADO

Para empezar a generar las pistas es necesario presionar el botón , el enrutado automático se detendrá si el programa puede enrrutar por si solo todo el circuito, si esto no ocurre (por lo regular) tendrá que detenerlo manualmente con el botón **stop**.
En la parte inferior de la imagen pueden verse las pistas enrrutadas y las totales, si aun son muchas las pistas no enrrutadas trate modificando la ubicación de las componentes para esto presione el botón **stop** y después el botón deshacer en ingles **Undo.**
En este caso dos pistas no enrrutadas es algo bueno ya que solo tendrá que enrrutar manualmente dos conexiones por lo que no habrá mucho trabajo para completar el enrrutado.
El que solo dos pistas no se hayan enrutado depende enteramente de cómo se colocaron las componentes, una mala ubicación de componentes generará un gran numero de pistas no enrrutadas.

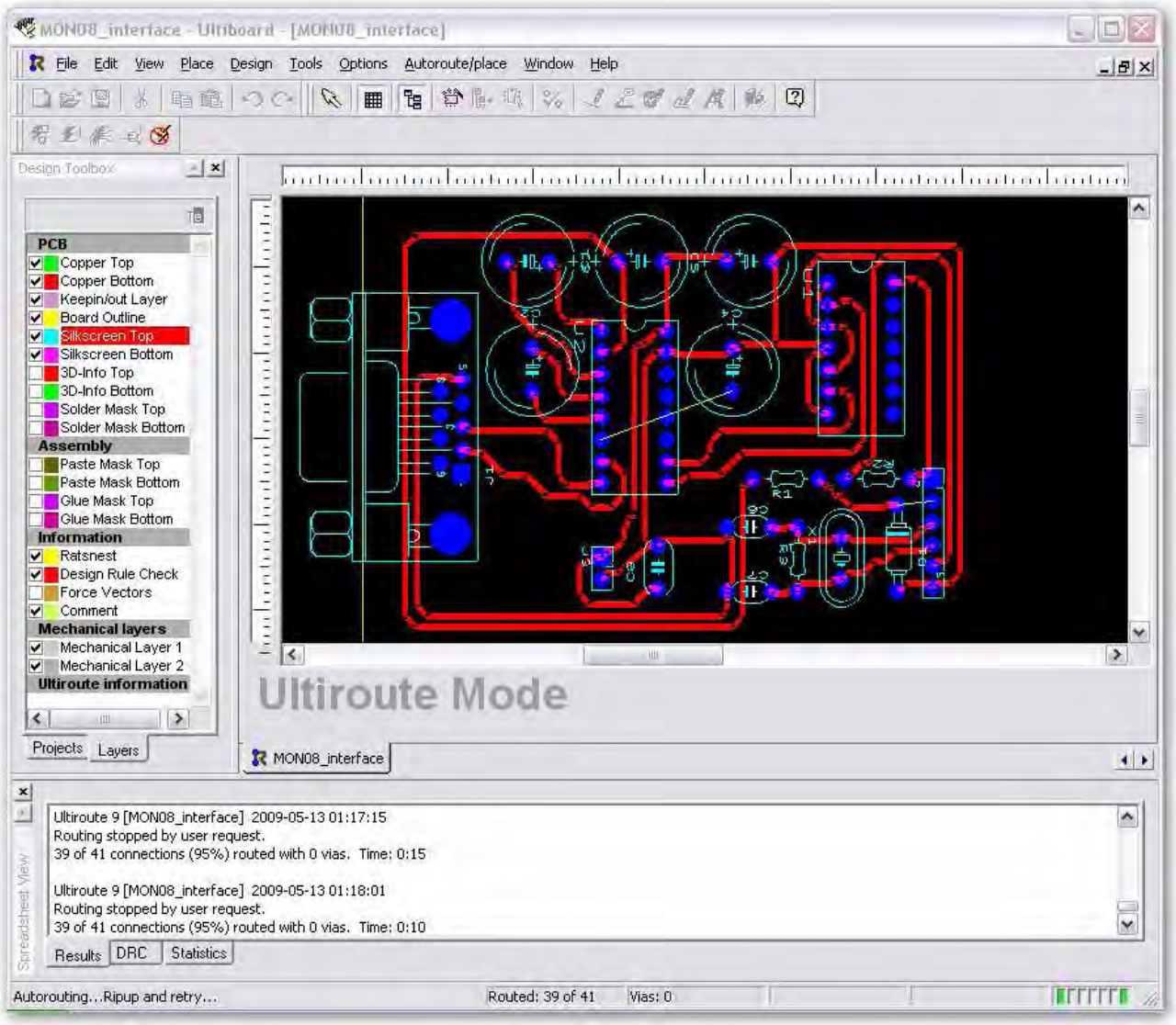


Figura 3.56: Enrrutado automático.

Trate de modificar las pistas para darle un mejor atractivo visual al circuito, evite los ángulos rectos en las pistas puesto que esto provoca que se levanten mas fácilmente con el calor, procure que estén en 45°, observe el circuito modificado (figura3.57) y compárelo con la figura anterior (3.56).

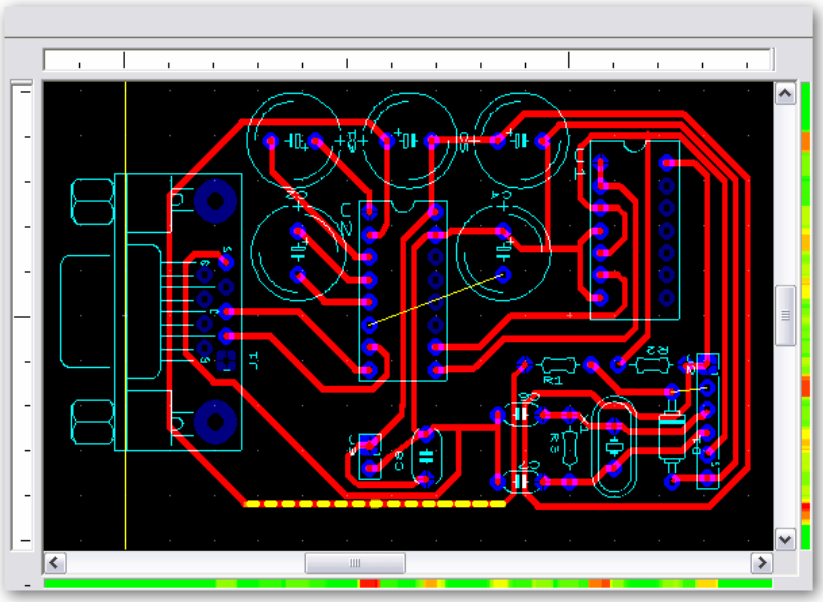


Figura 3.57: Retoque de las pistas.

A continuación coloque manualmente la mayor cantidad de pistas buscando posibles trayectos entre las conexiones faltantes. Observe la figura siguiente y compare con la anterior, nótese que se pudo realizar una conexión de las dos faltantes. Para esto puede valerse de mover componentes de dos pines, desplazar pistas y asta eliminarlas para después colocarlas en el mismo lugar o en algún otro punto que no altere el circuito.

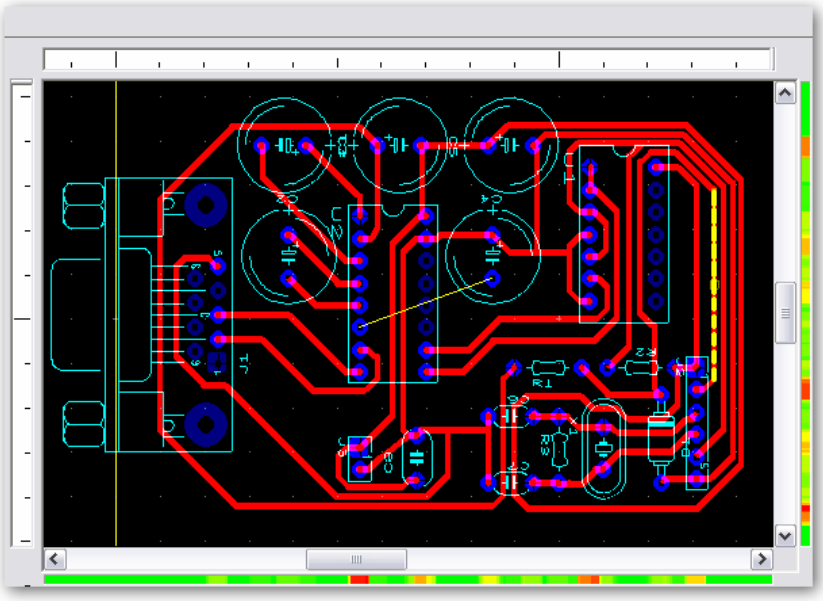


Figura 3.58: Enrrutado manual.

No se encontró ruta posible para la segunda conexión faltante por lo que se emplea un **Jumper** (elemento para saltar pistas) el cual se coloca en un lugar pertinente para acercar a las conexiones faltantes. Para colocar un **Jumper** presione en la barra de herramientas el botón **Place** y busque **Jumper**, después presione con el botón primario del mouse en el lugar donde quiere colocar uno de los pads del **Jumper**, a continuación presione de nuevo el botón primario del mouse para colocar el segundo pad. Presione el botón secundario del mouse si ya no desea colocar más **Jumpers**. Puede modificar el largo y la posición del **Jumper** después de haberlo colocado, no olvide modificar el tamaño de los pads del **Jumper** al recomendado con anterioridad.

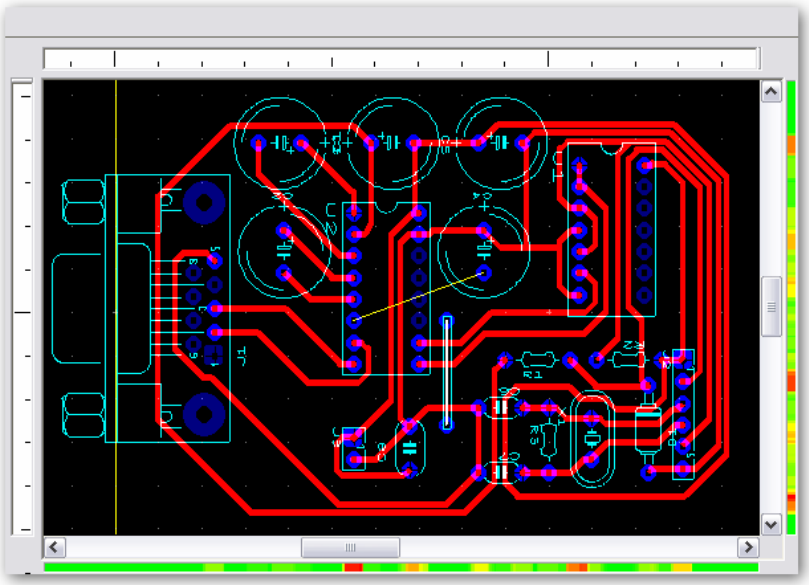


Figura 3.59: Colocado de un Jumper.

Ahora solo una la conexión restante a través del **Jumper.**

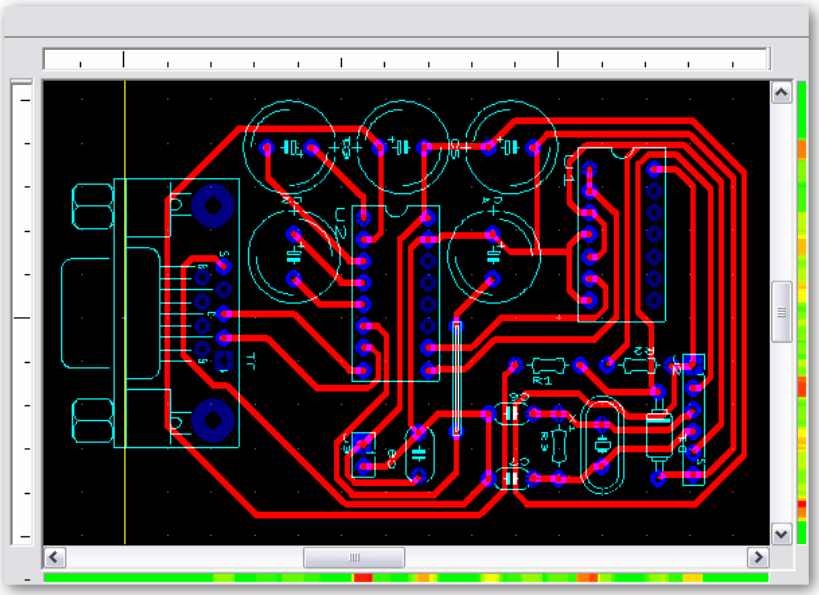


Figura 3.60: Enrrutado finalizado.

3.4.4 REDIMENSIONAMIENTO DE LA TARJETA

Modifique el contorno de la tarjeta al tamaño correspondiente, antes que nada seleccione la opción tal y como se muestra en la figura para poder modificar la tarjeta.

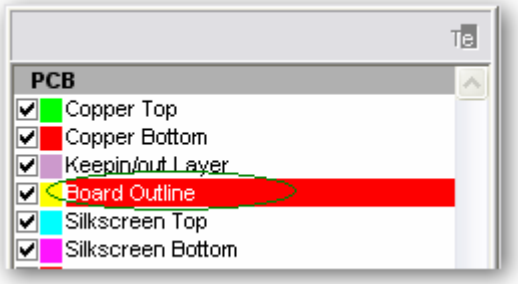


Figura 3.61: Perímetro de la tabla.

Ahora seleccione el contorno de la tarjeta presionando una sola vez con el botón primario del mouse, para modificar el perímetro presione y mantenga presionado con el botón primario sobre los pequeños cuadros amarillos (marcado en verde en la figura) que están al en medio de cada uno de los lados de la tarjeta y arrastre la línea asta la posición deseada.

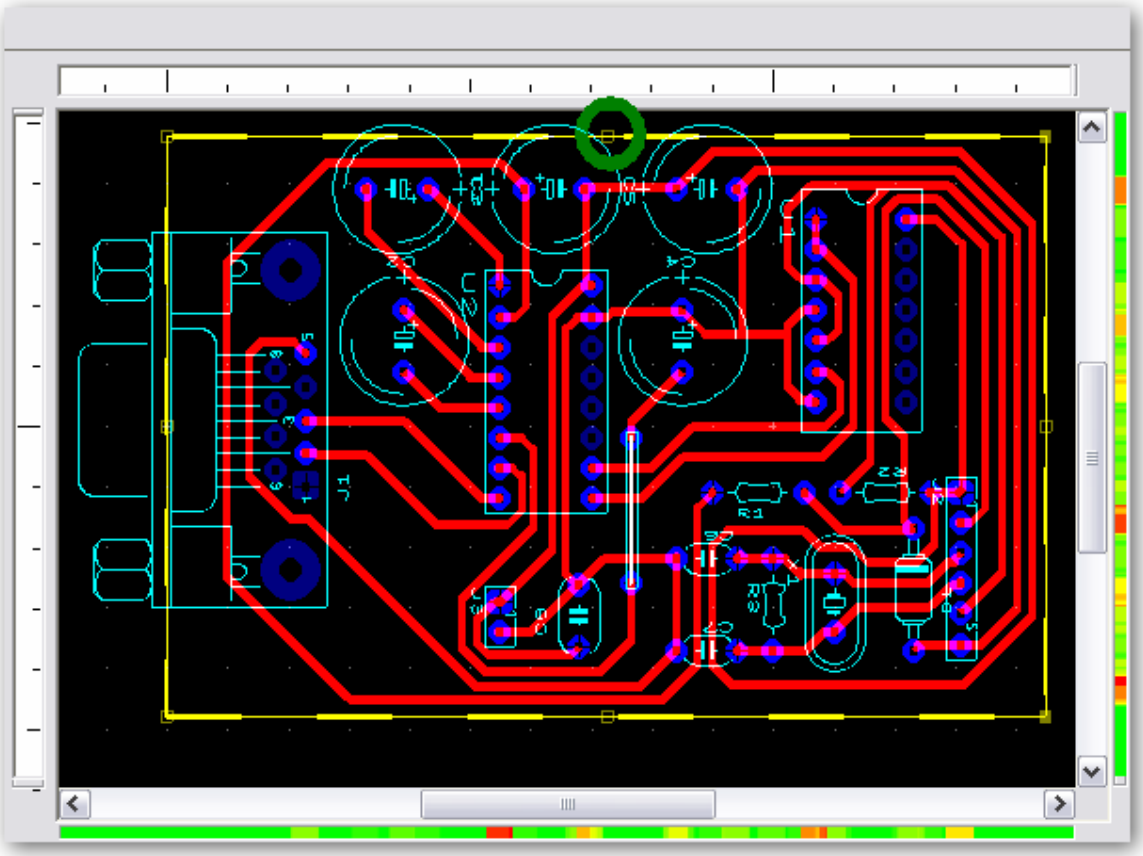


Figura 3.62: Perímetro de la tabla definido.

3.4.5 CREACIÓN DEL POWER PLANE

Generar el **power plane** no es más que rellenar los espacios vacíos del circuito impreso, este relleno debe estar ligado a alguna conexión regularmente se escoge GND pero puede ser cualquiera presente en el circuito. Este paso no es indispensable por lo que puede excluirlo.

Presione el botón  y aparecerá la ventana mostrada en la figura 3.63, ajústela tal y como se muestra.

Figura 3.63: Establecer el Power Plane.

Presione OK y tendrá el siguiente resultado

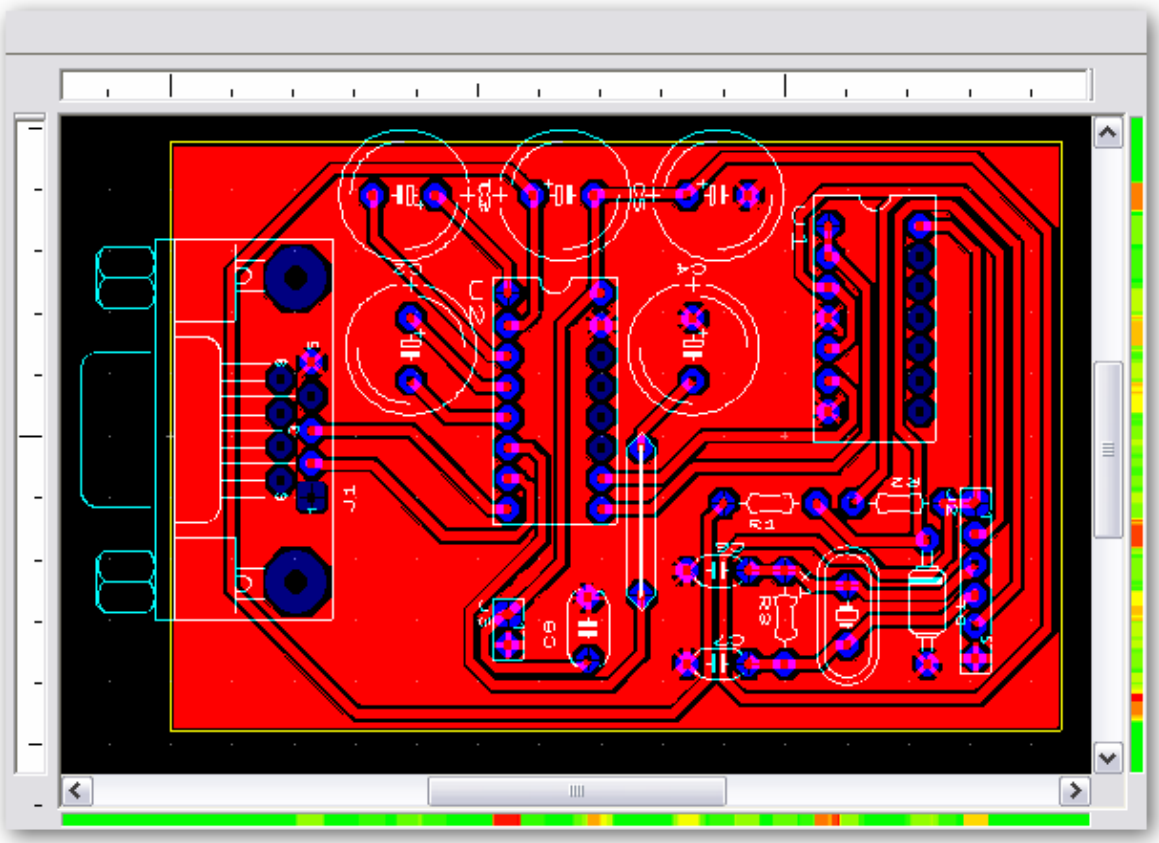


Figura 3.64: Establecer el Power Plane.

Puede incluir alguna clase de texto en el circuito con o sin **Power Plane** y así queda finalizado el diseño

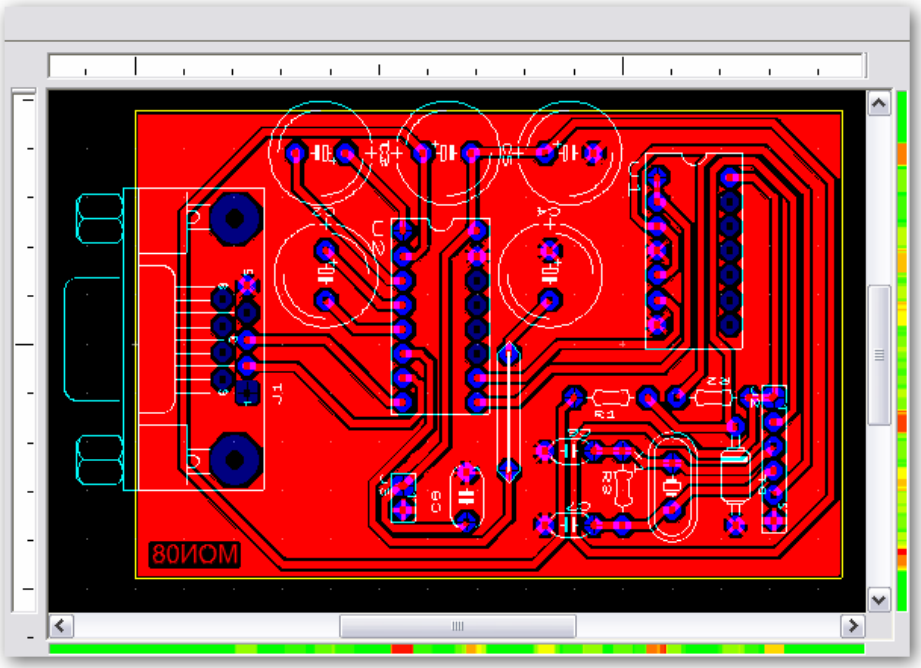


Figura 3.65: Circuito finalizado.

3.4.6 PASOS PARA LA IMPRESIÓN DEL DISEÑO

PRIMER PASO: Presione el botón  para acceder a la ventana de impresión.

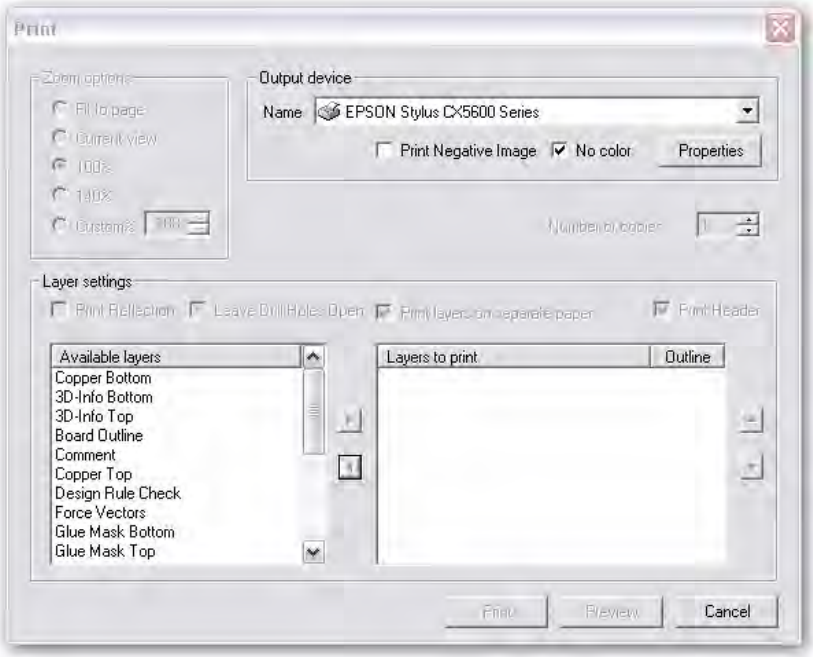


Figura 3.66: Ventana de impresión.

SEGUNDO PASO: Para imprimir el PCB simplemente seleccione la capa deseada en el recuadro inferior izquierdo y presione el pequeño botón marcado en la siguiente figura, con esto la capa a imprimir en nuestro caso la Copper Bottom será trasladada al recuadro **Capas a imprimir** (Layers to print).

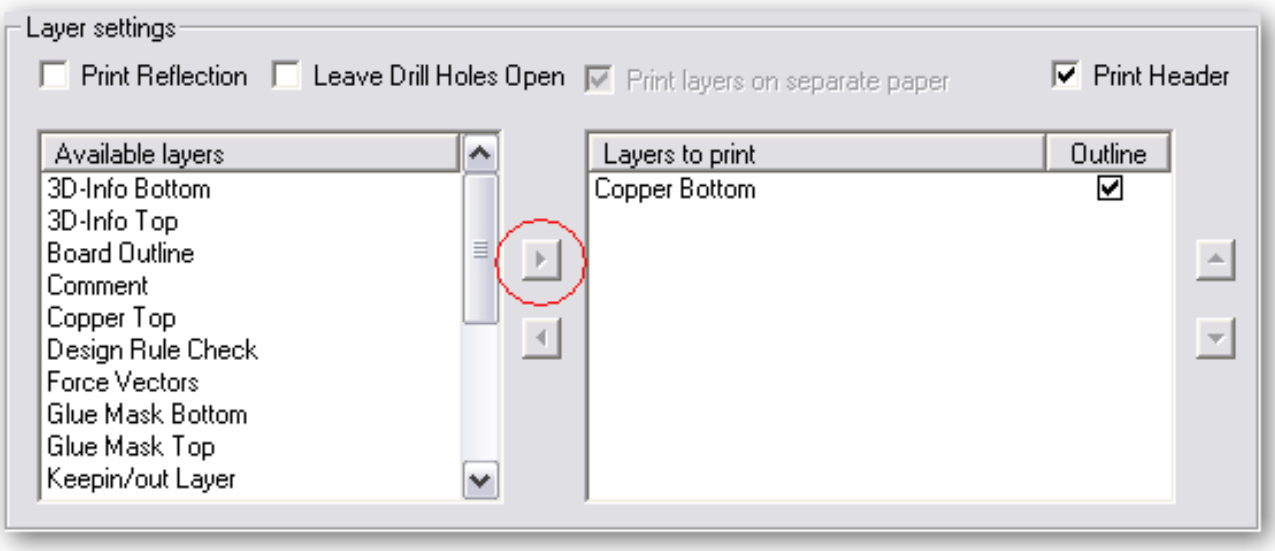
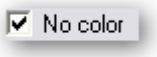


Figura 3.67 A): Capas disponibles y capas a imprimir.

Seleccione la casilla **Sin color,** esto es para que la impresión sea en color negro en caso de tener una impresora a colores. 

En las **opciones de zum** active la opción de imagen al 100%, esto es para obtener una impresión a escala real del PCB.

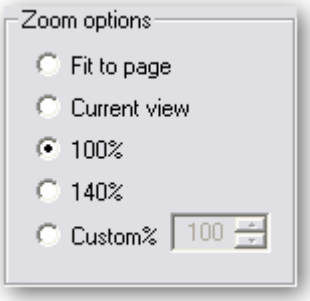


Figura 3.67 B): Impresión a escala real.

En resumen la ventana de impresión debe lucir tal y como se muestra en la figura 3.68, existen algunas otras opciones de impresión pero se recomienda establecerlas tal y como se muestra.

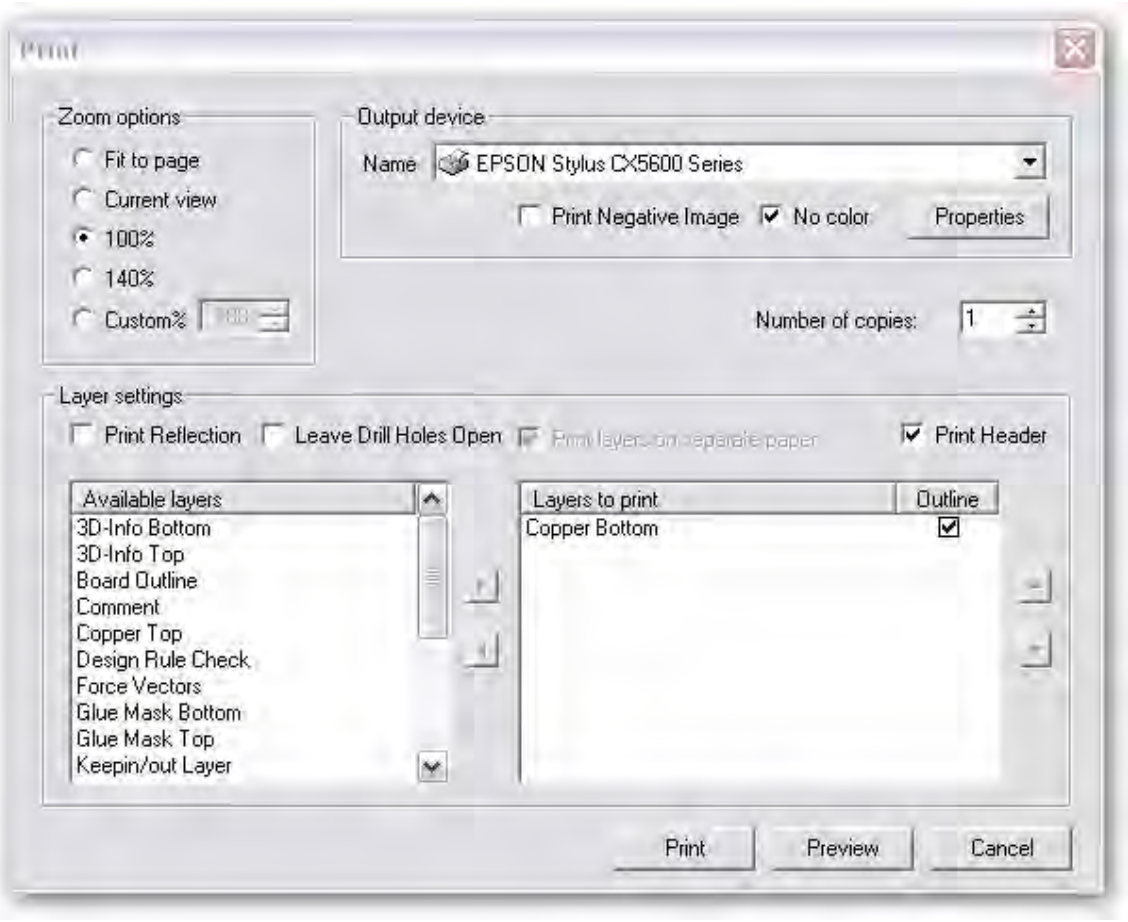


Figura 3.68: Ventana de impresión configurada.

Presione **Preview** para ver una imagen previa de la impresión o **Print** para imprimir el circuito. Es necesario imprimir con el programa Ultiboard por que el hacerlo desde otro programa como un procesador de textos como Word o un programa de edición de imágenes como **Paint** provocará la distorsión de la imagen del PCB.

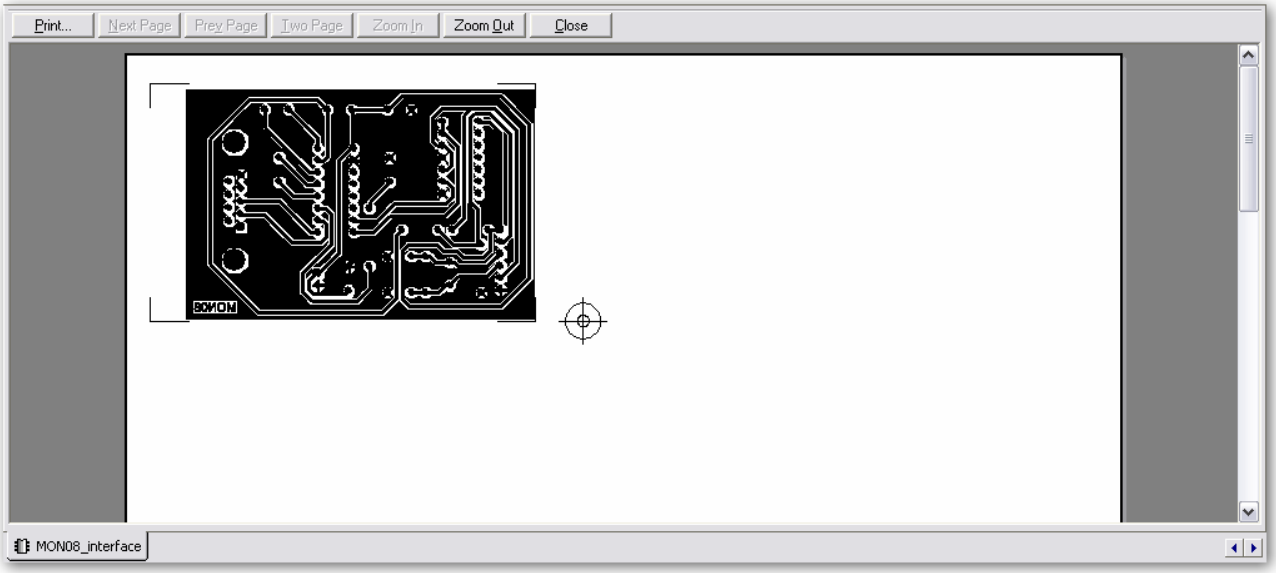


Figura 3.69: Preview.

3.5 FABRICACIÓN DEL CIRCUITO----------

Existe diferentes métodos para traspasar el diseño del PCB a la tarjeta, en mi opinión uno de los mejores es la serigrafía pero resulta ser mas costoso y complejo.
El método que recomiendo para el estudiante es el del planchado el cual consiste en traspasar el diseño del PCB a la tarjeta virgen a través de papel transfer.

3.5.1 MATERIALES

TABLA FENÓLICA VIRGEN DE UNA CARA
Es el consumible más costoso empleado en la fabricación de la tarjeta, recomiendo comprarlo en su presentación grande o mediana en AG Electrónica ya que a la larga resulta más económico que comprar este material por pequeños cuadros. Este material debe estar libre de la humedad ya que tiende a oxidarse con rapidez por lo que recomiendo guardarlo en bolsas de cerrado hermético.

Figura 3.70: Tabla fenólica virgen una cara.

CLORURO FERRICO
Este material de venta en tiendas de electrónica como Sterem o AG suele usarse sin modificarse aunque es un poco lento al momento de revelar el circuito, la ventaja es que puede reutilizarse varias veces asta que empieza a perder su efectividad pero puede renovarse agregando ácido muriático en pequeñas partes asta que el deposito fangoso generado por el uso desaparezca, ya hecho esto se procede a agregar agua oxigenada, la cantidad apropiada debe ser equivalente a la mitad de lo que agrego de ácido muriático es decir por cada parte de ácido que se agrego debe colocarse media parte de agua oxigenada. Después de varias regeneraciones con ácido muriático y agua oxigenada se encontrará una cierta cantidad de agua que puede ser eliminada haciendo que se evapore.

La formula liquida vendida por Sterem como ya se menciono es lenta pero al agregar un poco de agua a la formula esta se activa y el circuito puede estar listo en menos de 20 minutos, también ayuda que el cloruro férrico se encuentre alrededor de los 30°C, además de que mientras la placa este sumergida en el cloruro férrico la solución debe estar en movimiento, en este caso la revelación del circuito es muy rápida pero las instrucciones del producto indican desecharlo una ves usado y no mezclarlo con cloruro férrico no activado

PELIGRO: SI SE DECIDE A RENOVAR LA EFECTIVIDAD DEL CLORURO FÉRRICO DEBE TENER ESPECIAL CUIDADO CON EL ÁCIDO MURIÁTICO EN NO TOCARLO Y EN NO RESPIRAR LOS GASES DESPEDIDOS DURANTE LA MEZCLA.

Figura 3.71: Cloruro Férrico solución.

PAPEL TRANSFER

El papel transfer para impresoras de inyección de tinta puede llegar a ser muy costoso dependiendo del lugar de compra y marca del papel, definitivamente si se trata de economía STEREM no es el lugar indicado para comprarlo, si opta por este papel recomiendo acuda a alguna papelería Lumen.

Por este motivo recomiendo papel transfer para impresoras Láser o fotocopiado a tóner, el cual puede comprarse a un económico precio por pliego, de venta en cualquier papelería Lumen. Al final de este trabajo de tesis puede observar una muestra del papel transfer que recomiendo.

HERRAMIENTA ROTATORIA O TALADRO DE ALTAS REVOLUCIONES

Se requiere de un taladro de altas revoluciones debido a que el uso de uno convencional provoca la fácil ruptura de brocas tan pequeñas. Un taladro que suele usarse mucho en este ramo es el mostrado en la figura con un precio moderado, aunque no recomendado por su elevado precio en relación a su calidad pero aun así es una buena opción para realizar el trabajo

Figura 3.72: Taladro de joyero.

Lo ideal es tener una herramienta rotatoria con base pero esto suele ser muy costoso, observe la figura en la cual se muestra una base para taladro marca Dremel.

Figura 3.73: Base para taladro Dremel.

La herramienta rotatoria marca Dremel es muy costosa pueden encontrarse equipos muy completos y costosos o austeros de menor precio.

Figura 3.74: Taladro o herramienta rotatoria Dremel.

Se recomienda el uso de una herramienta rotatoria Truper por su potencia y buen precio, más económica que el taladro de joyero y tan potente como el taladro Dremel. Este taladro cuenta con una cómoda extensión para un trabajo menos agotador además de tener una velocidad máxima de 35000 RPM.

Figura 3.75: Taladro o Herramienta rotatoria Truper.

BROCAS DE 1/32 Y 3/64

Para la selección de esta herramienta lo mejor son las brocas de carburo de tuxteno las cuales son muy afiladas y rígidas pero no las recomiendo debido a su elevado costo, un inconveniente aun mayor es que tienden a romperse si las deja caer debido a su base mucho mas gruesa que su punta esto es para hacerlas compatibles con cualquier herramienta rotatoria.

Figura 3.76: Brocas de carburo de tuxteno.

Recomiendo el uso de brocas de acero sencillas son mucho mas baratas, estas no se rompen al caer debido a su base igual de gruesa que la punta con el inconveniente de que no son compatibles directamente con la herramienta rotatoria por lo que se requiere de un mandril para herramienta rotatoria. El mandril marca Dremel mostrado en la figura es compatible con las herramientas rotatorias Truper y Dremel.

Figura 3.77: Brocas económicas.

Figura 3.78: Mandril Dremel.

LIJA MUY FINA PARA AGUA
Para conseguir este material basta con ir a la tlapalería y pedir la lija mas fina para agua que tengan, esta nos sirve para retirar la tinta del circuito ya grabado, también puede valerse de acetona.

PLUMÓN INDELEBLE PUNTO FINO Y PUNTO MEDIANO
Este es utilizado para el retoque del circuito impreso antes de llevarlo al cloruro férrico, es importante que sea indeleble.

CALADORA
Se recomienda el uso de una caladora con sierra delgada para madera aunque puede usar lo que desee para cortar la tabla fenólica

3.5.2 ELABORACIÓN

PRIMER PASO
Si su impresora es láser imprima el PCB directo en el papel transfer. Si su impresora es de inyección de tinta imprima en una hoja de papel bond y fotocopie el PCB sobre papel transfer. Recorte el circuito del papel transfer y procure no tocar la superficie del papel transfer en donde se encuentra impreso el PCB ya que podría remover el toner.
Corte la tabla fenólica uno o dos milímetros más grande que el circuito impreso en el papel transfer y límpiela muy bien asta dejarla libre de grasas con agua y jabón o algún desengrasante, observe la figura.

Figura 3.79: Tarjeta virgen y PCB a tranferir.

SEGUNDO PASO

Coloque el impreso del PCB sobre la cara de la tarjeta que tiene el cobre, asegure con cinta adhesiva y planche por alrededor de 3 o 5 minutos sobre una superficie lisa

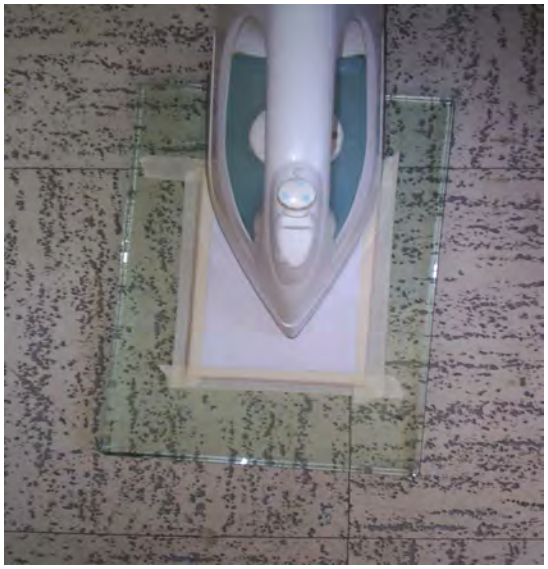


Figura 3.80: Traspaso del PCB.

Retire el papel tranfer mientras aun este caliente, deje enfriar y retoque con un plumón indeleble, el retoque es crucial para evitar pistas abiertas o pistas unidas después del grabado del circuito con el cloruro férrico, por lo que en el retoque debe añadir tinta donde sea necesario y retirar tinta raspando la tarjeta con alguna navaja en donde sea necesario,

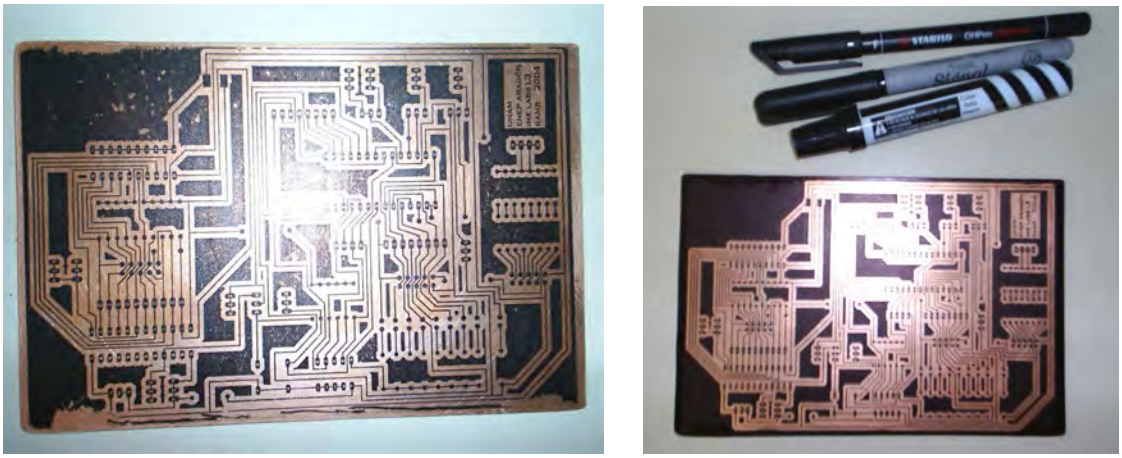


Figura 3.81: Retoque del PCB.

TERCER PASO

Sumerja la tarjeta en su totalidad en el cloruro férrico y mueva constantemente la solución hasta que el cobre que no este protegido por la tinta o el toner se disuelva en su totalidad. A continuación retire la tinta y el toner con algún solvente si esto no es suficiente puede utilizar una lija de agua muy fina, en este momento debe tener en la tarjeta solo el circuito impreso ya sin tinta por lo que se procede a perforar la tarjeta.

CUARTO PASO

Para las perforaciones de los circuitos integrado, resistencias, capacitares y demás componentes utilice la broca de 1/32.

Para las perforaciones de headers, diodos como el 1N4001, transistores de potencia, etc. utilice la broca de 3/64

Si usted no cuenta con una base para la herramienta rotatoria recomiendo que con un martillo y un punzón o clavo muy agudo marque en donde van las perforaciones para que la punta de la broca tenga una guía inicial y no tienda a deslizarse con la rotación y por consiguiente dañe el pad o alguna pista, observe la figura donde pueden apreciarse algunas marcas, ahora será mas fácil perforar cada uno de los pads de la tarjeta como corresponde.

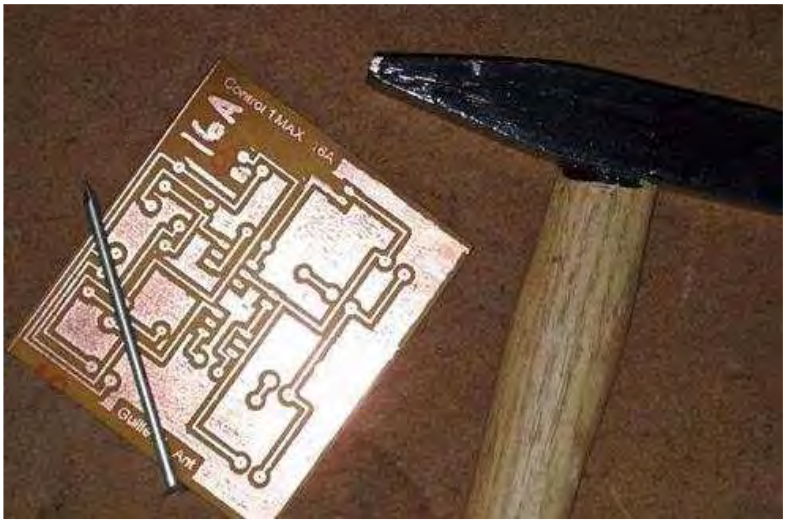


Figura 3.82: Marcado de las perforaciones.

QUINTO PASO

Ya perforada la tarjeta realice una inspección visual para confirmar la integridad de las pistas si no esta totalmente seguro utilice un multímetro y pruebe continuidad ya que pueden existir aberturas con un grosor menor al de un cabello, también debe revisar que no existan uniones entre pistas o conexiones a tierra que no deban estar presentes o cualquier error de este tipo, si es necesario repare la pista dañada o sepárelas según sea el caso.

Suelde cada una de las componentes, al finalizar esta tarea cubra el circuito impreso con una o dos capas de laca transparente en aerosol para evitar la oxidación.

3.6 RESUMEN----------

El programa Multisim es muy utilizado en la simulación de circuitos eléctricos y electrónicos, en este caso solo fue utilizado en la captura de un circuito electrónico el cual no puede simularse.

El programa Ultiboard es utilizado en el diseño de PCB´s aunque puede realizarse el diseño sin necesidad del Multisim 9 se recomienda realizar el circuito en este programa para transportarlo a Ultiboard con el fin de reducir los tiempos de diseño.

El incremento en la destreza de un diseñador para realizar un PCB depende totalmente de la práctica continua y constante, con estos elementos y un poco de tiempo el diseñador podrá desarrollar sus propias técnicas y habilidades.

Las herramientas Multisim y Ultiboard son solo eso unas herramientas que dependen totalmente del factor humano.

La técnica mostrada para elaborar circuitos impresos es recomendable en el desarrollo de prototipos, en el caso de planear realizar una producción repetitiva del mismo elemento se recomienda el uso de la serigrafía o simplemente mandar a manufacturar la tarjeta que usted ha diseñado.

CAPÍTULO

4

DISEÑO DE UN SISTEMA EMBEBIDO

OBJETIVOS

Después de estudiar el ejemplo en este capítulo usted podrá ser capas de:

- *Entender si esta en sus posibilidades resolver el problema*
- *Buscar posibles soluciones a un problema*
- *Selección de componentes adecuadas según el caso*
- *Entender un algoritmo*
- *Diseñar su propio algoritmo*
- *Convertir un algoritmo en lenguaje ensamblador*

Para el diseño de un sistema embebido es necesario reunir una gran cantidad de cocimientos en general eléctricos y electrónicos es decir deben aplicarse los conocimientos adquiridos durante la ingeniería eléctrica y electrónica, el diseño de un sistema embebido puede ir de controlar pocas acciones hasta el control de una gran cantidad de elementos los cuales pueden variar en funcionalidad como pueden ser teclados, displays y LCDs hasta poderosas etapas de potencia como el control de un motor trifásico a través de TriAC

4.0 CONOCIMIENTOS PREVIOS

Un diseñador de sistemas embebidos debe tener a su alcance una gran cantidad de conocimientos entre los principales tenemos:

- Análisis de circuitos eléctricos
- Dispositivos electrónicos
- Electrónica digital
- Electrónica analógica
- Electrónica de potencia

- Control digital
- Control Analógico
- Diseño lógico
- Diseño de sistemas digitales
- Diseño de sistemas con Microprocesadores o Microcontroladores de 8 bits en lenguaje ensamblador
- Transmisión de datos
- Procesamiento digital de señales
- Diseño de PCB (Circuitos impresos)

Algunos conocimientos extras mejoran la capacidad del diseñador

- Toda clase de protocolos y puertos de comunicación
- Programar microcontroladores en lenguajes como el C o el C++
- Programación de microcontroladores de 16 y 32 bits

Hay una gran cantidad de aplicaciones que se monitorean o controlan desde una computadora (PC) es decir en ocasiones tendrá que diseñarse hardware que puedan comunicarse con el PC en este caso debe elaborarse una interfase visual en el PC que visualice o controle el hardware, por lo regular un Ingeniero en computación se encarga de esta parte por que pueden llegarse a necesitar controladores de dispositivos que hagan compatible al Hardware con el sistema operativo del PC. Los siguientes conocimientos son necesarios:

- Uso de LabView
- Programación orientada a objetos (Visual Basic)
- Diseño de controladores que hagan compatible al PC con el Hardware

4.0.1 EL PROBLEMA Y SUS NECESIDADES, PASO UNO

A petición del Centro Tecnológico de la FES Aragón se procede a diseñar un sistema de monitoreo que transmita de alguna manera los estados físicos (temperaturas y presiones) de un sistema en este caso un refrigerador solar en fase experimental a una computadora no muy lejos del sistema.

Los encargados de este refrigerador solar nos dan los siguientes puntos a ser monitoreados

- PUNTO 1: En este puno se necesitan los valores de presión y temperatura
- PUNTO 3: En este puno se necesitan los valores de presión y temperatura
- PUNTO 4: En este puno se necesitan los valores de presión y temperatura
- PUNTO 5: Se requiere medir la presión
- PUNTO 7: Se requiere medir la presión
- PUNTO 8: Se requiere medir la presión

En realidad esto no aun no sirve de mucho, tal ves solo para establecer una nomenclatura por ejemplo la presión en el punto 7 podría llamase solo P7 (la nomenclatura es importante mientras mas clara mejor).
Lo que si es muy importante son los valores de trabajo nominales obtenidos por los encargados del refrigerador solar a través de instrumentos analógicos, los cuales son:

- P1: 006.78 mmHg
- P3: 049.10 mmHg
- P4: 006.79 mmHg
- P5: 355.26 mmHg
- P7: 006.79 mmHg
- P8: 006.79 mmHg
- T1: 30 °C
- T3: 80 °C
- T4: 37 °C

El Centro tecnológico desea esto:

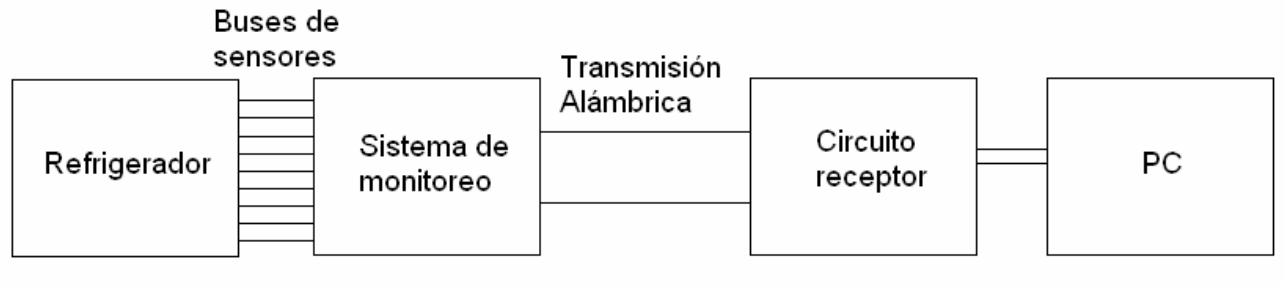


Figura 4.1: Diagrama a bloques de lo solicitado.

4.0.2 SOLUCIONES PROPUESTAS, (SELECCIÓN DE SENSORES Y ACTUADORES QUE ESTEN DENTRO DEL RANGO DE OPERACIÓN) PASO DOS

Tendiendo los requerimientos del sistema se plantea la siguiente solución, note las diferencias entre el sistema solicitado y el sistema propuesto.

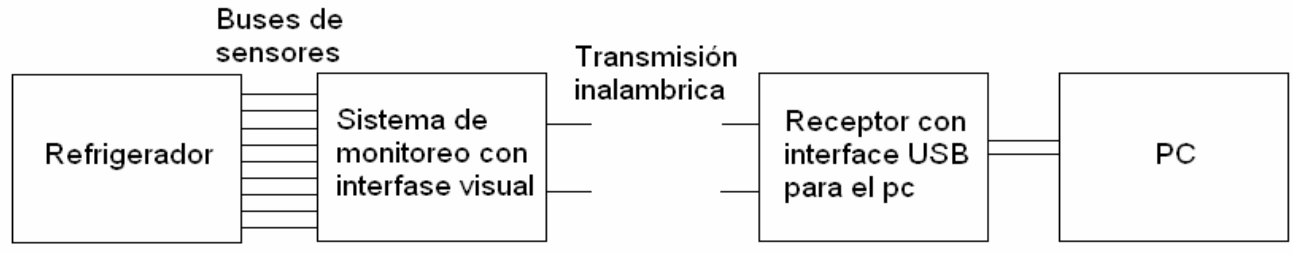


Figura 4.2: Diagrama a bloques de lo propuesto.

Ya teniendo los valores nominales se procede a seleccionar los sensores adecuados para cada uno de los puntos de acuerdo al los valores de trabajo nominal.

Se decide a utilizar una familia de sensores de presión integrados, con acondicionamiento de señal, compensados en temperatura, calibrados y con un gran rango de acción (spawn) además de poseer un funcionamiento lineal fabricada por Freescale.
Debido a estas características la dificultad en el uso de estos sensores es muy reducida además de no necesitar acondicionamiento de señal gracias a su gran spawn.

LOS SENSORES DE PRESIÓN Y TEMPERATURA A UTILIZAR SON:

Para P1, P4, P7 y P8 se a escogido el sensor MPXV5004G debido a su rango de presión el cual va de 0 mmHg a 29 mmHg y con un spawn que va de 1.0v a 4.9v

Para P3 se ha escogido el sensor MPXV5010G debido a su rango de presión el cual va de 0mmHg a 75 mmHg y con un spawn que va de 0.2v a 4.7v

Para P5 se ha escogido el sensor MPXV5100G debido a su rango de presión el cual va de 0mmHg a 750 mmHg y con un spawn que va de 0.2v a 4.7v

En cuanto a la temperatura se refiere se decide a utilizar un mismo sensor para T1, T3 y T4 el cual es un termistor lineal del fabricante Microchip MCP9700, el cual tiene un spawn que va de 0.1v a 1.75v y su rango es de -40°C a 125°C.

Entre los principales problemas a solucionar tenemos la selección adecuada del convertidor analógico a utilizar, en el caso de P1, P4 y P7 no es necesario más que un convertidor analógico digital de 8 bits debido a que no se solicito una gran resolución en el dispositivo lo mismo sucede para P3

En el caso de P5 que va de 0mmHg a 750mmHg un convertidor analógico digital de 8 bits que solo puede entregar 256 estados posibles contra 751 estados que se necesitan desplegar debido a la resolución deseada de 1mmHg resulta no conveniente por lo que se planea utilizar un convertidor analógico digital de 10 bits.

Lo mismo sucede con el sensor de temperatura sumándole además el problema del pequeño spawn que posee el sensor contra la resolución mínima de 1°C que se desea, por lo que también se debe utilizar como mínimo un convertidor analógico digital de 10 bits para evitar el uso de acondicionamiento de señal. Si este sensor tuviera un spawn parecido al que tienen los sensores de presión podría utilizarse un convertidor de 8 bits

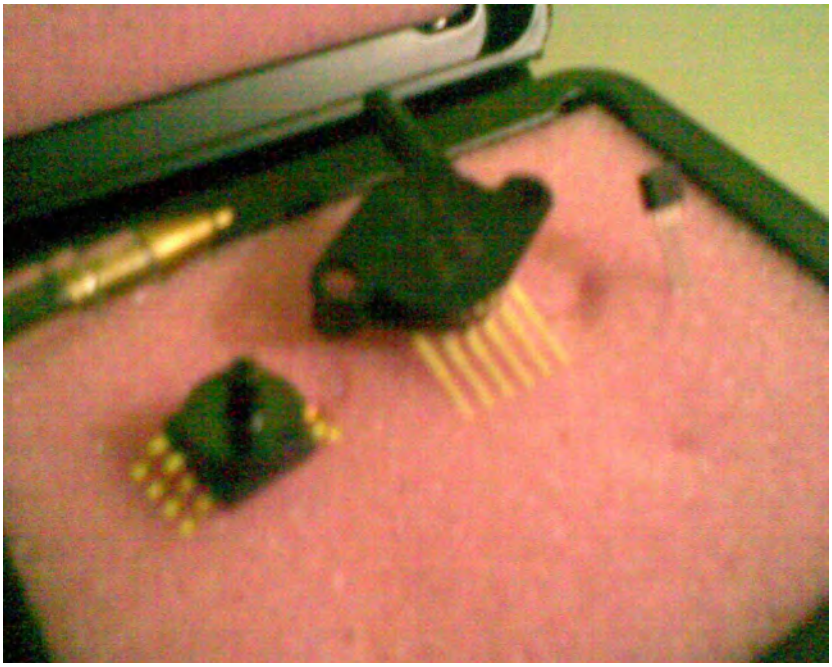


Figura 4.3: Sensores de presión y de temperatura.

VISUALIZACIÓN

Para la visualización se decide utilizar un LCD de 4 líneas por 16 segmentos, en ente punto debe tenerse en cuenta como se van a mostrar los elementos, debido a la cantidad de medidas a mostrar se decide por colocar en el sistema un swich que realice un cambio en el LCD, por ejemplo swich ON se despliegan solo los valores de presión y swich OFF se despliegan los valores de temperatura.

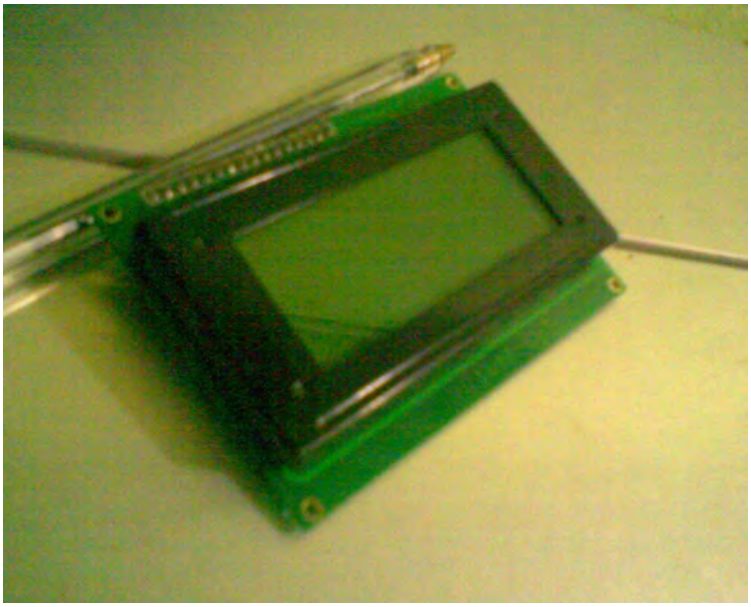


Figura 4.4: LCD.

TRANSMISION Y RECEPTOR

El receptor escogido no esta diseñado para la transmisión serial tradicional puesto que en esta los tiempos de espera que pueden llegar a ser muy largo están dados por un uno lógico y el bit de arranque por un cero lógico, el problema radica en que el circuito receptor al recibir un uno lógico muy largo lo considera como un error y después de un cierto tiempo aun a pesar de estar recibiendo un uno lógico su salida cambia de estado a cero.
La solución es muy sencilla y radica en modificar el protocolo tradicional invirtiendo el valor del tiempo de espera, el bit de arranque y el bit de paro, ahora los largos tiempos de espera estarán dados por un cero lógico por lo que el problema queda solucionado.

CIRCUITO RECEPTOR:

Para el receptor se decide utilizar un microcontrolador de 8 bits de 8 pines que reciba los datos a través del receptor de radiofrecuencia para ser introducidos en un dispositivo llamado FT232BL el cual genera un puerto virtual SERIAL siendo este conectado a través de un puerto USB. La función del MCU es recibir los datos del receptor RF con un protocolo de comunicación serial invertido y mandarlos al FT232BL con el protocolo tradicional ya que el FT232BL no es mas que un convertidor de puerto serial a USB solo que su lado serial funciona con niveles TTL y por lo tanto debe recibir los datos con el protocolo tradicional es decir el tiempo de espera debe ser un uno lógico, el bit de arranque un cero lógico y el bit de paro un uno lógico, en pocas palabras el MCU funciona de traductor.

RECEPTOR RF ⟹ MCU ⟹ FT232BL ⟹ USB ⟹ PUERTO SERIAL VIRTUAL EN EL PC

EN EL PC

Se ha comprobado que el programa Lab View es compatible con puertos virtuales por lo que se decide a utilizarlo para desplegar los datos recibidos en el PC

4.0.3 ELECCION DEL MCU Y OTRAS PIEZAS CLAVES, PASO TRES

ELECCION DEL MCU

Enumeremos todo lo que necesitaría un MCU para el sistema de monitoreo que deseamos:

1.- MCU con convertidor analógico digital de 8 y 10 bits
2.- Nueve entradas para el convertidor analógico digital
3.- Dos puertos de propósito general para el control de un LCD
4.- Ocho puertos de propósito general para el puerto de datos del LCD
5.- Un puerto de propósito general para la transmisión de los datos
6.- Un puerto de propósito general para señalizar los momentos de transmisión con un LED
7.- Un puerto de propósito general para cambiar la visualización en el LCD

Se ha encontrado un MCU con la suficiente cantidad de puertos y con convertidor analógico digital de 8 y 10 bits, es el MC68HC908JL16.

CIRCUITO TRANSMISOR

Este transmisor de 433.92MHz con una buena antena tiene un alcance de hasta 120 metros en exteriores y su voltaje de polarización puede ser de 3.0v a 12.0v. A la venta en México, visite www.robodacta.com.mx.

Figura 4.5: Transmisor.

CIRCUITO RECEPTOR

Este receptor de 433.92MHz con ancho de banda de 1MHz debe ser polarizado con 5.0v.

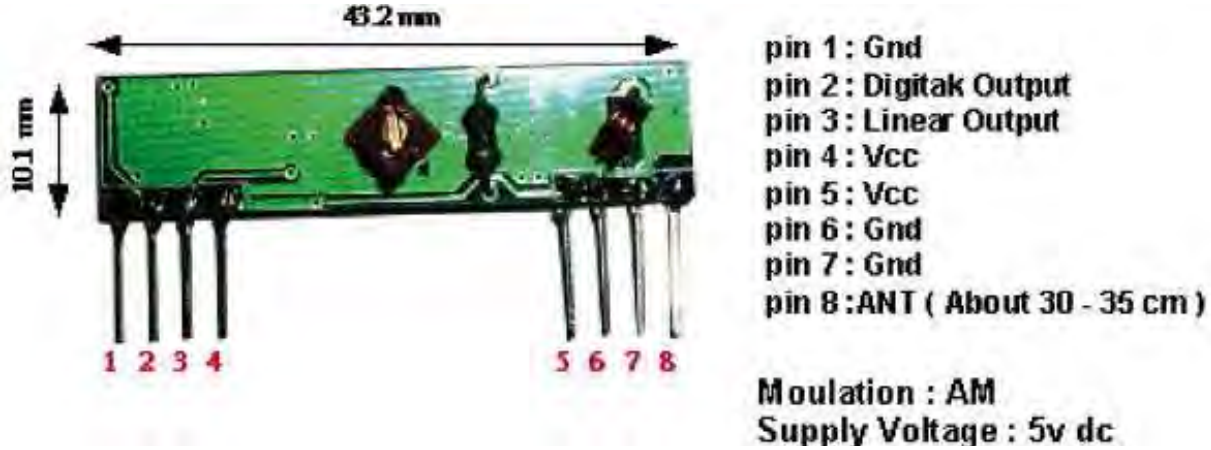


Figura 4.6: Receptor.

4.1 DISEÑO DEL SISTEMA ----------

La principal dificulta en el diseño del sistema es el programa a grabar en el microcontrolador que por la complejidad del sistema puede llegar a ser algo extenso por lo que dedique un tiempo a mirar el proyecto Codewarrior correspondiente a este prototipo que se encuentra en el anexo al final de este trabajo de tesis .

4.1.1 DISEÑO DE LOS CIRCUITOS, PASO CUATRO

En el diseño del circuito se ha decidido utilizar los elementos propuestos con anterioridad debido a que cubren en su totalidad todas nuestras necesidades.

CIRCUITO DEL SISTEMA DE MONITOREO

U1: MCU MC68HC908JL16CPE perteneciente a los HC08 grabar con MON08
U2: Regulador de voltaje LM7805
U3: Puente de diodos
P1, P4, P7, P8: Header para la conexión del sensor MPXV5004G
P3: Header para la conexión del sensor MPXV5010G
P5: Header para la conexión del sensor MPXV5100G
T1, T2, T3: Header para la conexión del sensor MCP9700
SW1: Conector header para conectar un swich con la función de cambiar el estado de LCD.

La configuración de estos 9 conectores (P1, P4, P7, P8, P3, P5, T1, T2 y T3) es:

PIN1: Vcc
PIN2: GND
PIN3: Vout = Voltaje analógico a la salida del sensor

POT: Potenciómetro o preset de 10 KΩ
HEADER 16: Conector para el LCD
AC: Conector header para el transformador de 9v a 12v
TX: Módulo transmisor.
TORNILLO: Pad de gran tamaño para colocar un tornillo con su tuerca para el cable de la antena.
X1: Cristal de cuarzo de 32 MHz.

NOTA: El circuito contiene un error en el que Vdd esta en el pin 6 y PTA1 en el pin 7 siendo esto incorrecta ya que Vdd es el pin 7 y PTA1 el pin 6.

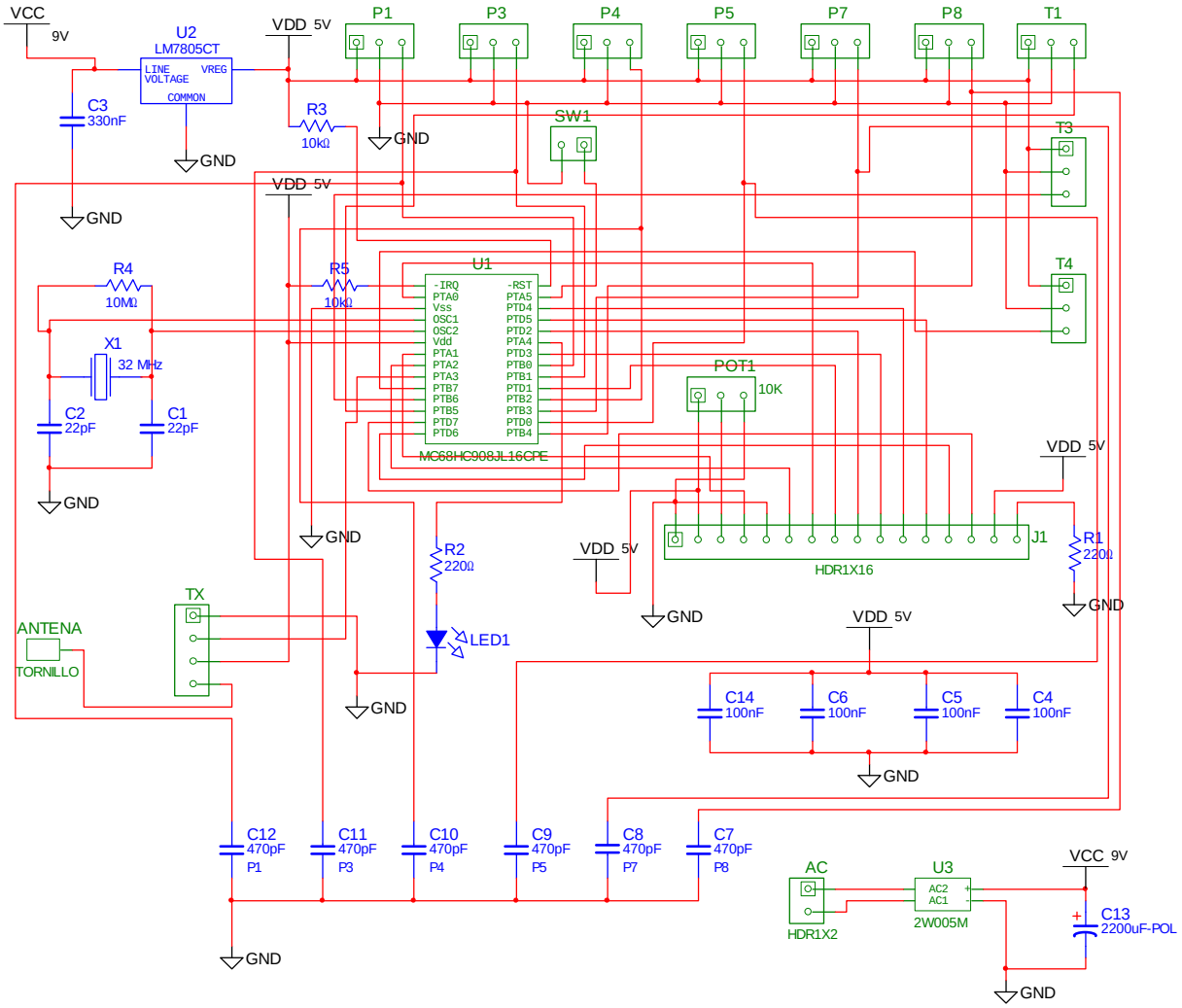


Figura 4.7: Circuito del sistema.

CIRCUITO RECEPTOR

U1: Conector USB clase B en 90°
U2: FT232BL Dispositivo USB que genera un puerto serial virtual
U3: MCU MC9S08QD4_soic perteneciente a los HCS08 grabar con las herramientas OSBDM o USBSPYDER08
J2: Conector Header BDM para la grabación del MCU
J3: Conector header con Jumper, retirar si se desea grabar
Receptor A, Receptor B: Conectores para el receptor, el pin 1 es la conexión marcada en A y el pin 5 es la conexión marcada en B
J1: Simplemente sirve de guía para colocar las partes del módulo receptor a la distancia correcta al diseñar el PCB.
ANTENA: Es un test point que se agranda en el PCB para soldar una antena.

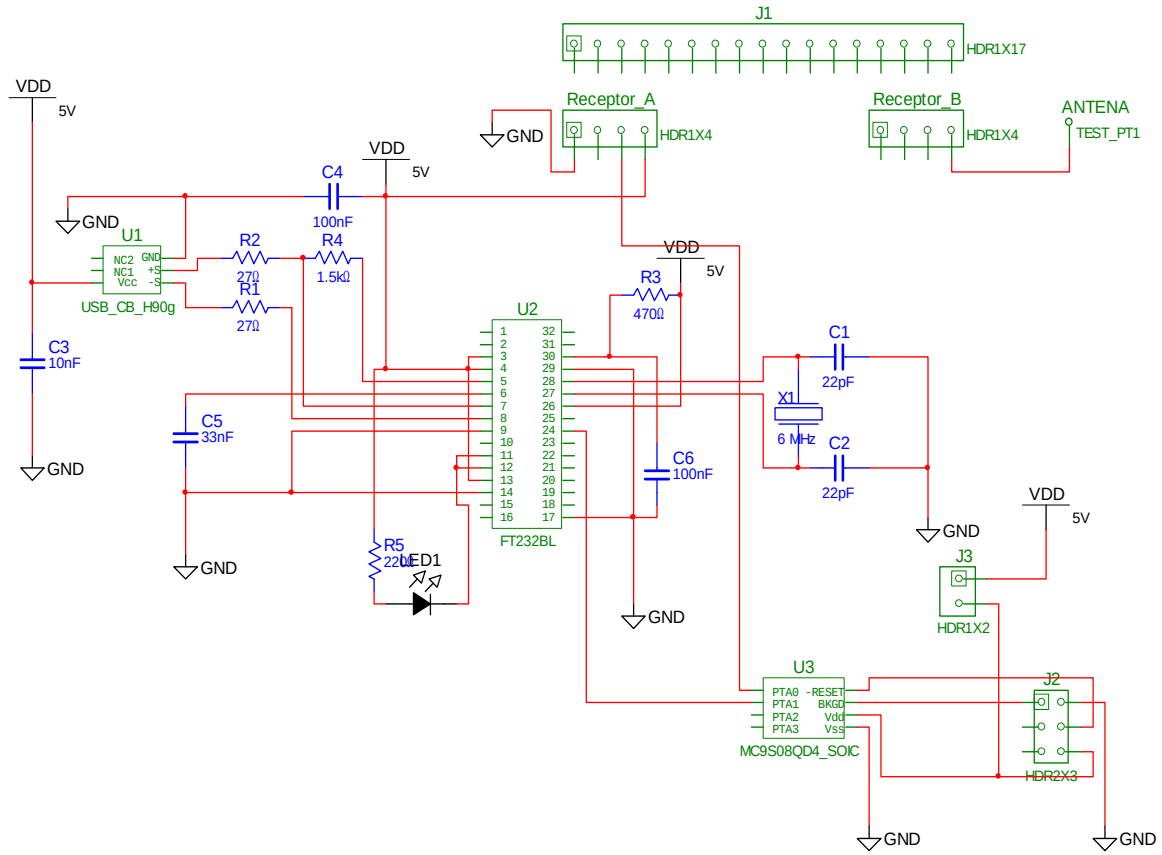


Figura 4.8: Circuito del receptor.

4.1.2 DISEÑO DEL PCB PASO CINCO

El tamaño de las tarjetas debe ser muy específico debido a que estarán montadas en pequeños gabinetes. Mida los gabinetes y ajuste la tarjeta colocando líneas de acotación seleccionando la opción marcada en la figura para después dirigirse a la herramienta PLACE – DIMENSION LINES y en este menú puede selecciona de entre HORIZONTAL, VERTICAL o ESTÁNDAR

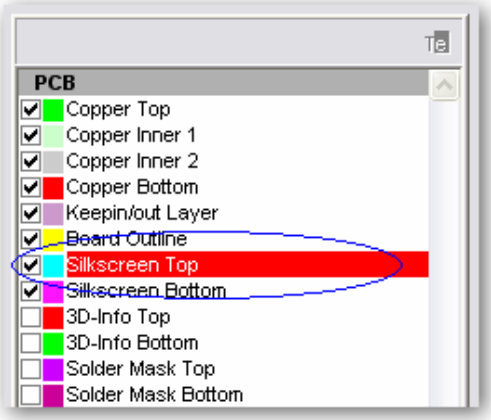


Figura 4.9: Selección de acotamiento.

PCB DEL SISTEMA DE MONITOREO

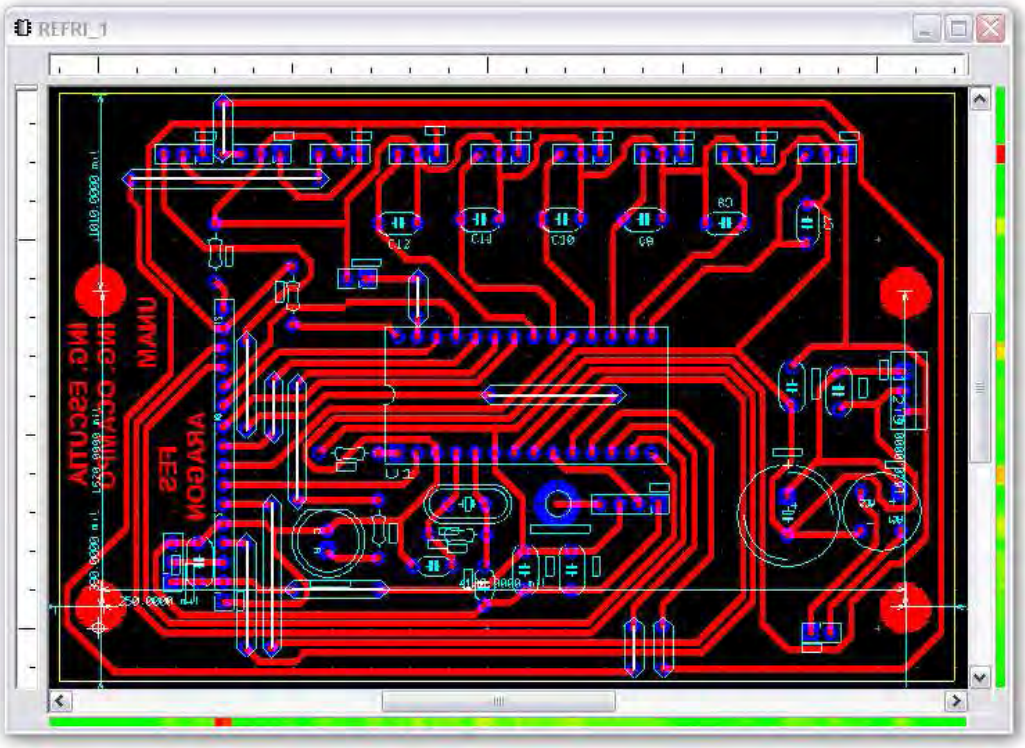


Figura 4.10: PCB del transmisor.

PCB DEL SISTEMA RECEPTOR

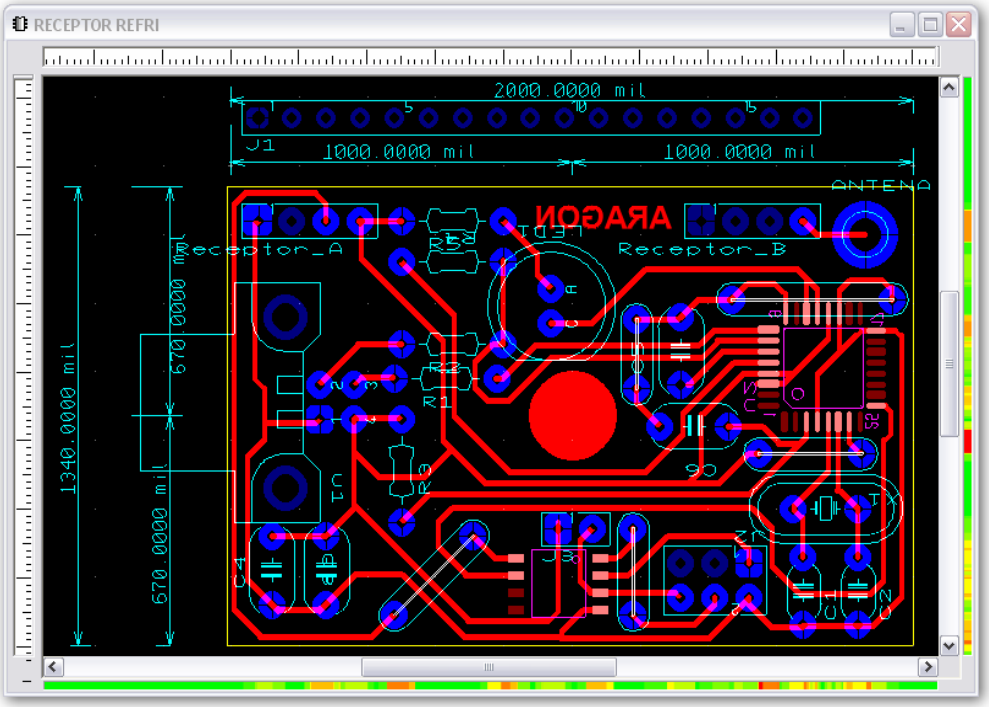


Figura 4.11: PCB del receptor.

4.2 EL ALGORITMO O PROGRAMA, PASO SEIS----------

Todo el código de este programa puede encontrarlo en el apéndice, el proyecto Codewarrior de este sistema esta incluido en el DVD presente al final de este trabajo.

4.2.1 INICIALIZACIÓN DEL MCU EN EL SISTEMA DE MONITOREO

La inicialización afecta al archivo MCUinit que ya se ha visto en capítulos anteriores.

CPU: Se informa que se usará un cristal de 32 MHz
COP: Se deshabilita el COP
ADC: Se habilita el bus interno como reloj, se ajusta la configuración a alta velocidad, se deshabilita el reloj asíncrono, el prescaler a 4, activar la conversión continua, se ajusta la conversión a 10 bits, se activan las muestras cortas, se habilitan 9 entrada del ADC (PTB0, PTB1, PTB2, PTB3, PTB4, PTB5, PTB6, PTB7 Y PYD0) y por ultimo se inicializa el sistema en el canal cero que corresponde a PTM0.
PTA: Los puertos PTA0, PTA1, PTA2, PTA3 y PTA4 se inicializan como salida y valor inicial en cero. El puerto PTA5 se inicializa como entrada y se activa su resistencia de pull up.
PTD: Los puertos PTD1, PTD2, PTD3, PTD4, PTD5, PTD6 y PTD7 se habilitan como salida y con valor inicial de cero.

4.2.2 PROGRAMA PRINCIPAL DEL SISTEMA DE MONITOREO

El programa puede verse abriendo el proyecto Codewarrior presente en el DVD anexo en este trabajo o en el apéndice al final de este.
El programa principal puede ser muy extenso y complejo como primer ejemplo a entender así que primero estudie los ejemplos de tipo didácticos mostrados más adelante.

4.2.3 INICIALIZACIÓN DEL MCU DEL RECEPTOR

CPU: Ajuste las opciones pertinentes en CPU para que el bus sea de 8 MHz
KBI: En le modulo de interrupciones habilite el pin 0 que corresponde a PTA0, la sensibilidad estará bien colocarla en edge and level o borde y nivel en español y la polaridad de interrupción será activa en nivel alto o en ingles high level.
PTA: active el pin 1 como salida y valor inicial de 1.

4.2.4 PROGRAMA PRINCIPAL DEL RECEPTOR

Para grabar el microcontrolador del circuito receptor es necesario una herramienta llamada OSDBM, su fabricación se mostrara en el siguiente capitulo. El programa puede observarse en el proyecto Codewarrior presente en el DVD o en el apéndice. Debe notar que en este

programa además del código principal hay una rutina de interrupción presente en el archivo MCUinit.

4.2.5 DEPURACION Y FINALIZACION DEL PROTOTIPO, PASO SIETE

En la depuración se corrigen los probables errores en el programa o el circuito y se realizan las pruebas necesarias hasta estar seguro que el sistema opera correctamente, aquí se ha presentado un programa y un circuito ya depurado.
Este es el prototipo finalizado montado en gabinetes de plástico.

Figura 4.12: Sistema de monitoreo.

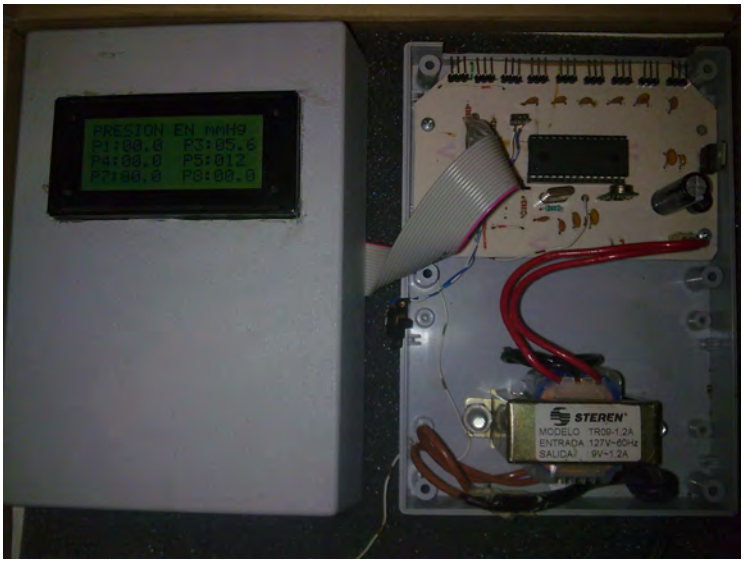


Figura 4.13: sistema de monitoreo visto por dentro.

Figura 4.14: Receptor.

Figura 4.15: Receptor visto desde dentro.

4.3 EJEMPLOS DIDACTICOS ----------

4.3.1 CONVERTIDOR ANALOGICO DIGITAL

FUNCION
Utilizando el MCU MC68HC908GP32ECPE.
Programa que despliega a través de LEDs conectados en el puerto A el valor binario del voltaje dado por el potenciómetro conectado en el canal cero (PTB 0) del convertidor analógico digital.

INICIALIZACIÓN
CPU: Coloque el valor del cristal a utilizar en este caso uno de 32 MHz y coloque a la CPU en modo usuario.
PTA: Habilite a todo el puerto A como salida y valor inicial de $00.
ADC: Ajuste el prescaler del convertidor analógico digital para que la velocidad del ADC este en su rango de operación. Para un cristal de 32 Mhz el cual entrega un bus de 8MHz el preescaler debe estar en 4. Ponga el modo de conversión en conversión continua. Active un solo pin y seleccione como canal inicial el canal cero.

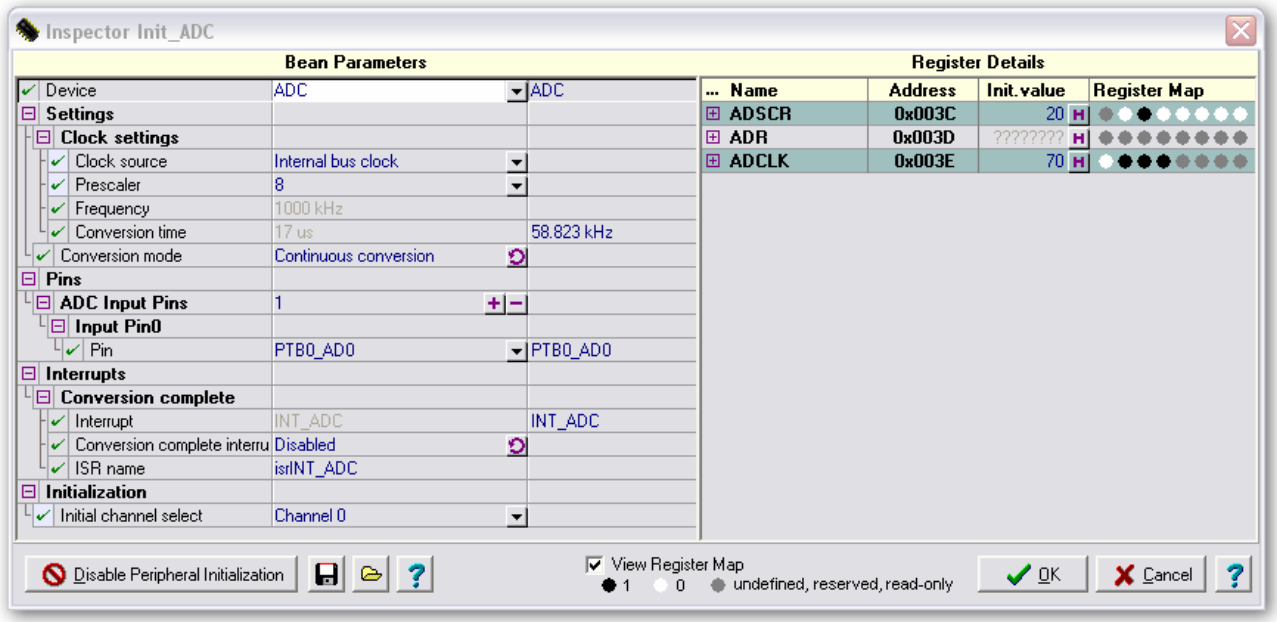


Figura 4.16: Ajuste del convertidor.

NOTA: No olvide presionar generar código después de realizar todos los ajustes.

DETALLES
Cuando el bit 7 del registro ADSCR se pone en alto significa que una conversión a sido completada, para que el convertidor continúe trabajando y realice otra conversión es necesario volver a cero el bit 7 del registro ADSCR para lograr esto debe leerse el valor del

resultado de la conversión el cual se encuentra en el registro ADR, la forma de hacer esto es con la instrucción LDA.
El símbolo asterisco (*) como etiqueta ase referencia a que si la condición se cumple el programa bifurque sobre la misma instrucción.

PROGRAMA

```
;*******************************************************************
;
        ; Insert your code here
            BRCLR   7,ADSCR,*     ;EN ESPERA POR UNA CONVERSION
            LDA     ADR           ; SE CARGA EN EL ACUMULADOR EL VALOR DE LA VONVERSION
                                  ; EL FABRICANTE RECOMIENDA QUE SE DESPRESIE LA PRIMERA
                                  ; CONVERSION
REPITE:
            BRCLR   7,ADSCR,*     ;ESPERA POR LA SEGUNDA CONVERSION
            LDA     ADR           ; SE CARGA EN EL ACUMULADOR EL VALOR DE LA CONVERSIO
            STA     PTA           ; ALMACENA EL ACUMULADOR EN EL PUERTO A

            BRA     REPITE        ; BIFURCA A LA ETIQUETA REPITE
;*******************************************************************
```

CIRCUITO

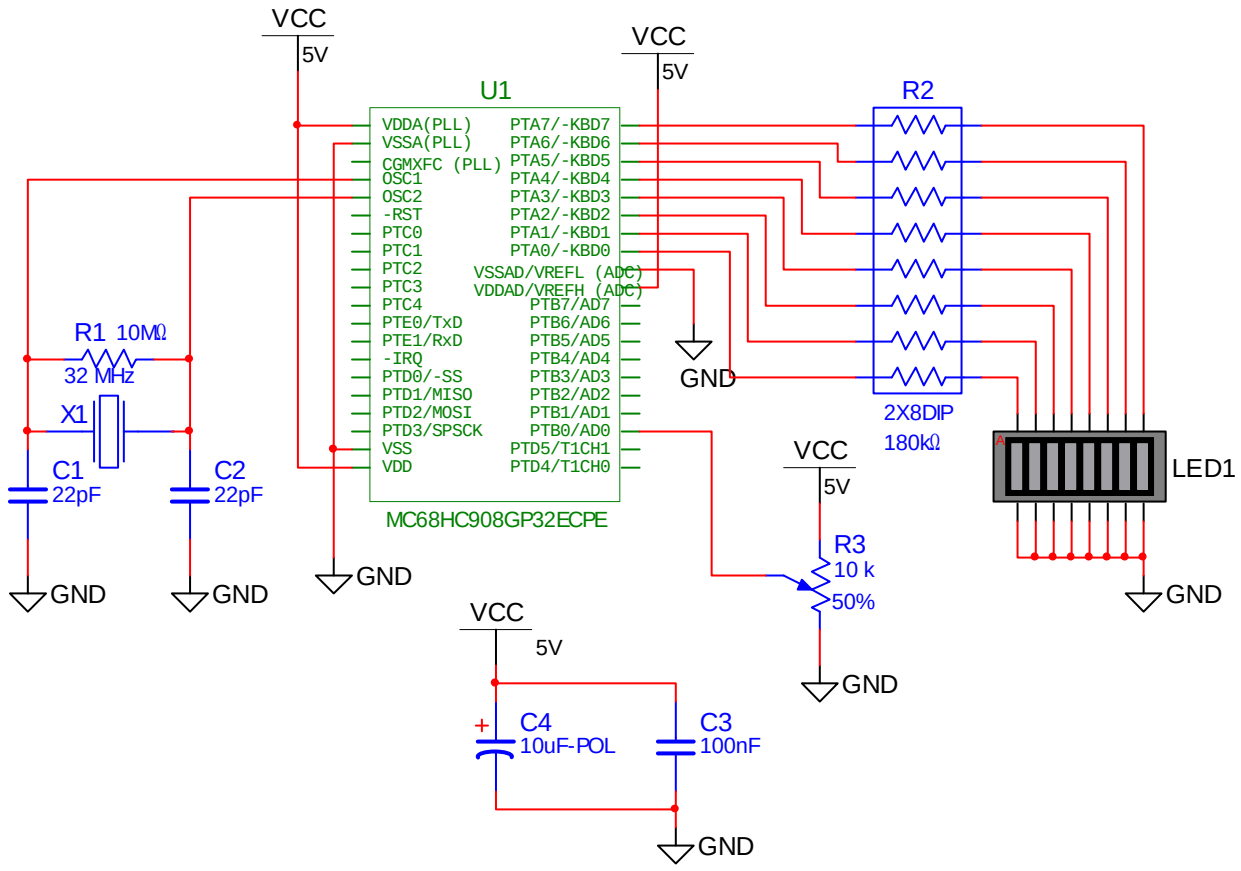


Figura 4.17: Circuito parara demostrar el funcionamiento del ADC.

4.3.2 INTERRUPCION DE UN SEGUNDO

FUNCION
Utilizando el MCU MC68HC908GP32ECPE.
Programa que prende y apaga un LED conectado en el puerto PTA 0 utilizando una interrupción por TIMER.

INICIALIZACIÓN
CPU: Coloque el valor del cristal que va a utilizar en este caso uno de 4.9152 MHz y coloque a la CPU en modo usuario
PTA: Coloque solo el puerto PTA 0 como salida y valor inicial de cero.
TIM1: Con el cristal de 4.9152 MHz se puede ajustar el Timer para obtener una interrupción de un segundo exactamente. Una manera de hacerlo es colocando el preescaler en 32 y el modulo que almacena el numero en el que la cuenta debe detenerse para ejecutar la interrupción en 38339.Habilite la interrupción por desbordamiento y por ultimo active el inicio de cuenta.

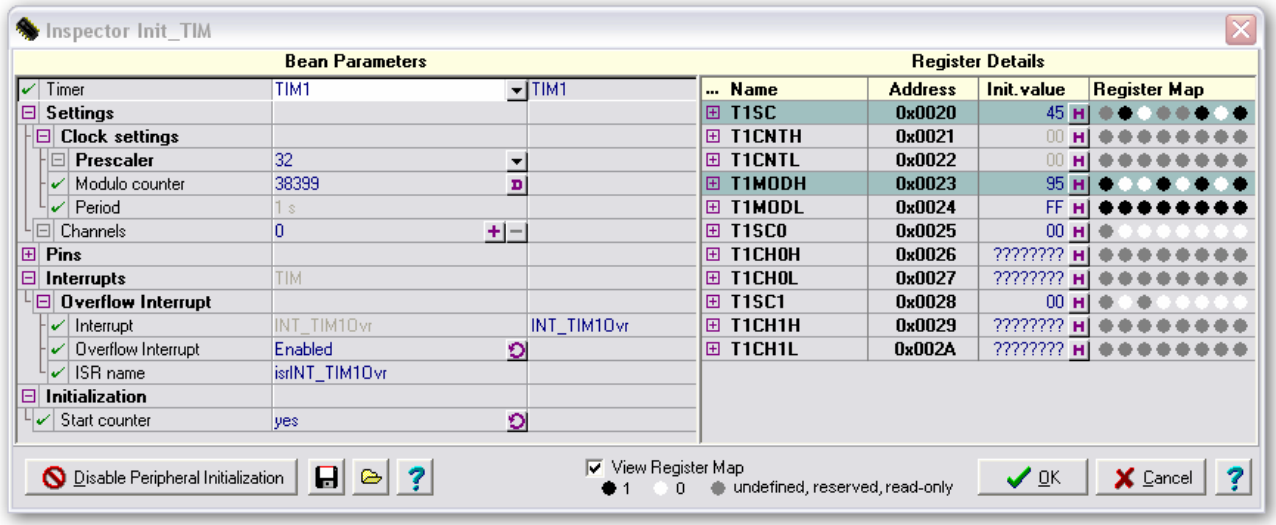


Figura 4.18: Ajuste del Timer.

DETALLES
El programa principal no es mas que una bifurcación infinita, lo importante es la rutina de interrupción presente en el archivo MCUinit.

Cuando la cuenta llega al número determinado en este caso 38339 se pone en alto el bit 7 del registro **T1SC** (bit de over flow) y si las interrupciones por teclado están activadas se ejecuta la interrupción.
Al final de la subrutina de interrupción es necesario poner a cero el bit 7 del registro T1SC debido a que si no hace esto se ejecutará inmediatamente la interrupción después de ejecutarse la instrucción RTI, podríamos llamar a esto reinicio de interrupcion.

PROGRAMA PRINCIPAL

```
;*************************************************************************
;
        ; Insert your code here

            BRA   *
;*************************************************************************
;
```

SUBRUTINA DE INTERRUPCION

```
;*************************************************************************
;
;** ======================================================================
;
;**     Interrupt handler : isrINT_TIM1Ovr
;**
;**     Description :
;**        User interrupt service routine.
;**     Parameters  : None
;**     Returns     : Nothing
;** ======================================================================
;
      XDEF    isrINT_TIM1Ovr
isrINT_TIM1Ovr:
  ; Write your interrupt code here ...
              BRSET  0,PTA,APAGA      ;PEQUEÑA SERIE DE INSTRUCCIONES QUE PONEN
              BSET   0,PTA            ; A UNO AL PUERTO SI ESTA EN CERO Y
              BRA    PRENDE           ; A CERO SI ESTA EN UNO
APAGA:        BCLR   0,PTA

PRENDE:       BCLR    7,T1SC          ;BORRA EL BIT QUE INDICA EL DESBORDAMIENTO DE LA CUENTA
              RTI                     ;RETORNA DE INTERRUPCION
; end of isrINT_TIM1Ovr
```

CIRCUITO

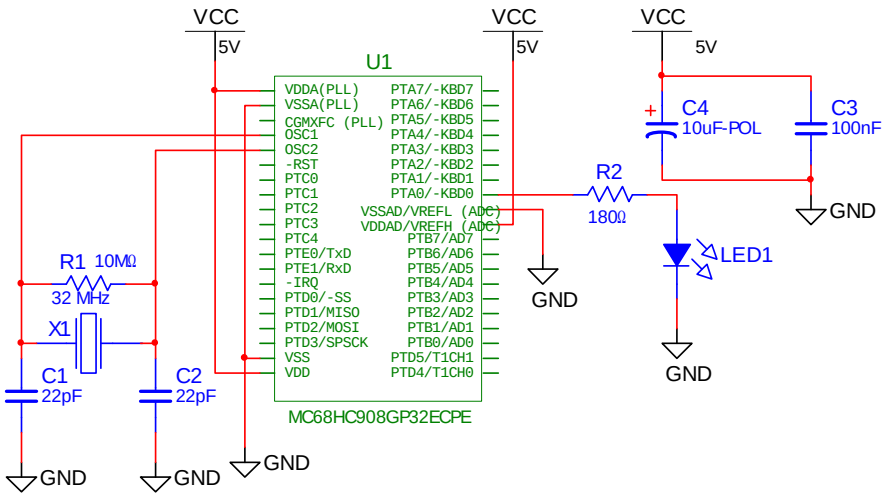


Figura 4.19: Circuito para el timer.

4.3.3 EJEMPLO DE TRANSMISION DE DATOS DE MCU A MCU CON EL MODULO SCI

FUNCION

Utilizando el MCU MC68HC908GP32ECPE.

Se muestra un ejemplo en el que se transmite de micro a micro de forma serial un dato, el dato se introduce a través de de un dip swich y se visualiza a través de LEDs.

INICIALIZACIÓN

CPU: En este caso se utiliza un cristal de 4.9152 MHz, coloque el modo de la CPU en modo usuario.

SCI: En el caso de los dos microcontroladores las opciones del modulo SCI deben estar exactamente igual. Coloque el divisor de baudios en 4 para una transmisión a 4800 baudios, habilite la interrupción por recepción, habilite el modulo SCI, la transmisión y la recepción.

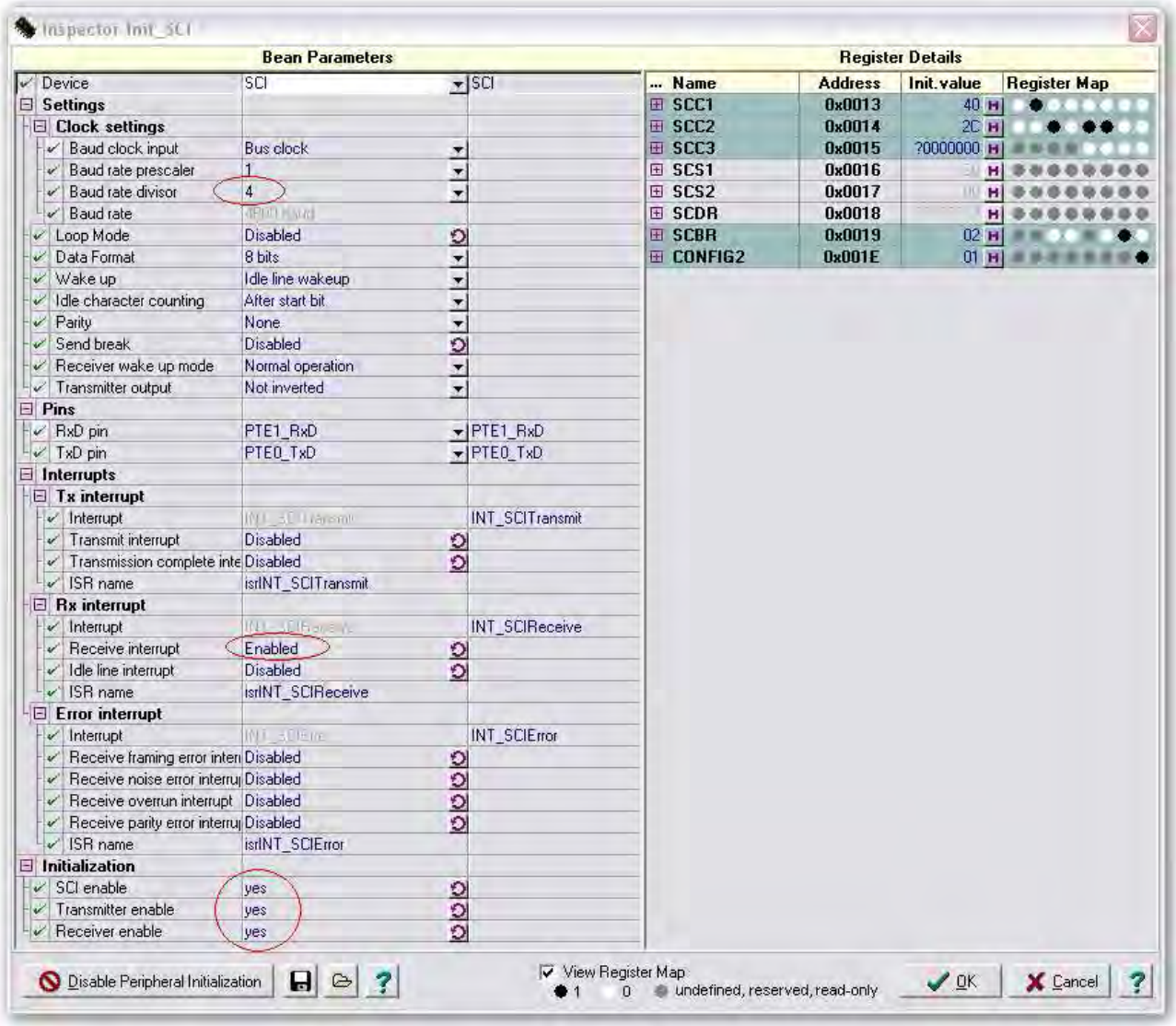


Figura 4.20: Ajustes del SCI.

DETALLES

El registro **SCDR** es un registro doble ligado directamente a la CPU. Si se escribe algún dato en el este funciona como transmisor, si se recibe un dato este se guarda en este mismo registro, es decir si se lee este registro se obtiene el dato recibido y si se escribe sobre este registro el dato escrito se transmite.

El programa principal es muy corto pero hay que tomar en cuenta que un MCU debe empezar a transmitir antes que el otro por lo que un programa debe ser ligeramente diferente del otro aunque las rutinas de interrupción son exactamente iguales.

Uno del los MCU toma la iniciativa y empieza a transmitir el dato presente en el puerto A dado por un dip swich para después irse a un bucle infinito. El segundo micro entra directamente al bucle infinito esperando a recibir un dato, cuando este llega se ejecuta la interrupción por recepción de dato completa, en esta interrupción se toma el dato recibido y se despliega en unos LEDs a través del puerto B, a continuación el programa le ordena al micro a tomar el dato presente en el puerto A dado por un dip swich y lo coloca en el registro SCDR e inmediatamente empieza la transmisión, el programa regresa al bucle infinito a través del retorno de interrupción a esperar recibir otro dato. El dato enviado por el MCU 2 llega al primer micro por lo que se activa su rutina de interrupción que realiza exactamente lo mismo que la interrupción del segundo micro por lo que este proceso se repite indefinidamente.

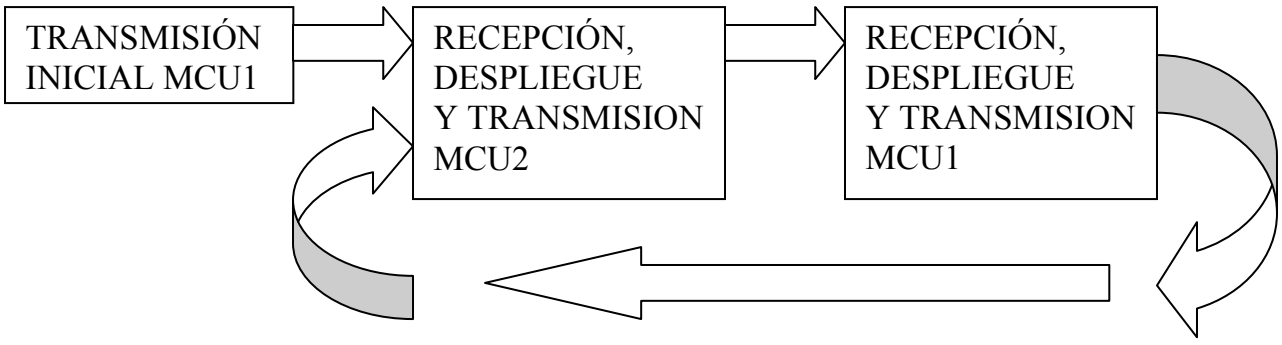


Figura 4.21: Ejecución del proceso.

NOTA: Recuerde que la inicialización y la rutina de interrupción es idéntica en los dos MCU, solo el minúsculo programa principal es ligeramente diferente.

PROGRAMA PRINCIPAL DEL MCU 1

```
;*******************************************************************
; Insert your code here
        LDA   PTA       ; CARGA EN EL ACUMULADOR EL VALOR PRESENTE EN EL
                        ; PUERTO A DADO POR EL DIP SWICH
        STA   SCDR      ; COLOCA EL VALOR DEL ACUMULADOR EN EL REGISTRO TRANSMISOR

        BRA   *         ; BIFURCA SOBRE SI MISMO INDEFINIDAMENTE
;*******************************************************************
```

PROGRAMA PRINCIPAL DEL MCU 2

```
;*************************************************************************
; Insert your code here
        BRA    *              ; BIFURCA SOBRE SI MISMO INDEFINIDAMENTE
;*************************************************************************
```

RUTINA DE INTERRUPCION PARA LOS DOS MICROCONTROLADORES

```
;** ===================================================================
;**     Interrupt handler : isrINT_SCIReceive
;**
;**     Description :
;**         User interrupt service routine.
;**     Parameters  : None
;**     Returns     : Nothing
;** ===================================================================
      XDEF    isrINT_SCIReceive
isrINT_SCIReceive:
  ; Write your interrupt code here ...
        LDA     SCDR        ; CARGA EN EL ACUMULADOR EL DATO RECIBIDO
        STA     PTB         ; COLOCA EL VALOR DEL ACUMULADOR EN EL PUERTO B
                            ; SE VISUALIZAN EN LOS LEDs EL DATO RECIBIDO
        LDA     PTA         ; CARGA EN EL ACUMULADOR EL DATO A TRANSMITIR PRESENTE
                            ; EN EL PUERTO A DADO POR EL DIP SWICH
        STA     SCDR        ; COLOCA EL DATO EN EL REGISTRO TRANSMISOR
        RTI                 ; RETORNA DE INTERRUPCION
; end of isrINT_SCIReceive
;*************************************************************************
```

CIRCUITO

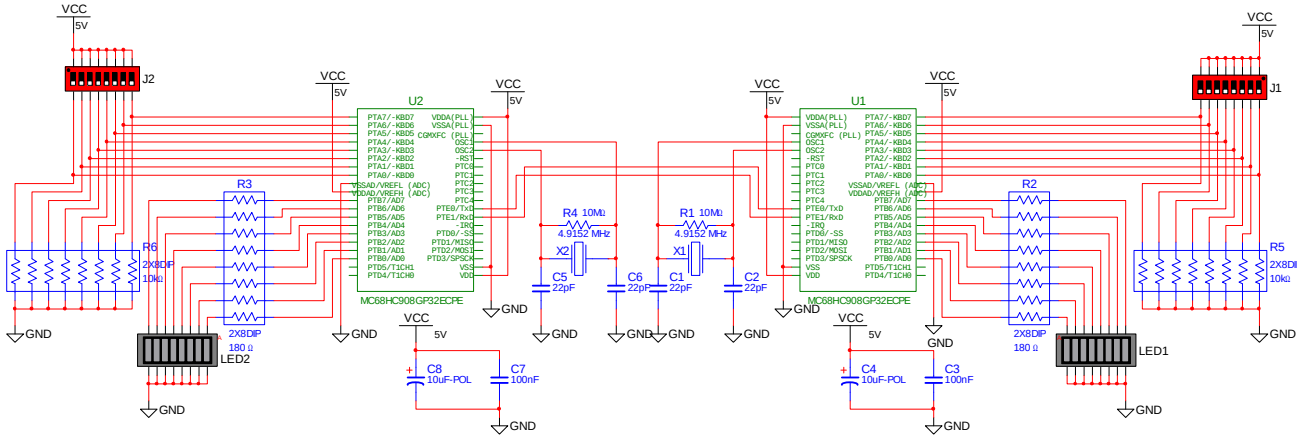


Figura 4.22: Circuito para el ejemplo del SCi.

4.3.4 ESCRIBE EN LA HYPER TERMINAL DE WINDOWS

FUNCION
Utilizando el MCU MC68HC908JL3ECPE.
Programa que escribe como mensaje inicial en la hyper terminal de windows “HOLA MUNDO”, después mandará el carácter que guste con un dip swich y un botón de **ENTER.**

INICIALIZACIÓN
CPU: Indique a este módulo que usará un cristal de 32 Mhz para obtener un bus de 8 Mhz y coloque a la CPU en modo usuario.
PTA: Coloque el puerto PTA 0 como salida y con valor inicial de uno ya que este es el valor del tiempo de espera en la comunicación serial, el PTA 1 debe ser colocado como entrada con la resistencia de pull up habilitada.
PTB: Coloque todo el puerto PTB como entrada.

DETALLES
El programa para la transmisión serial esta basado en deslizamientos que ocupan la bandera de carry para el bit que se a salido del registro por lo que el bit que se transmite es el presente en la bandera de carry, por lo que si el carry esta en cero el PTA 0 se pondrá en un estado bajo y si el carry esta en uno el puerto se pondrá en alto.

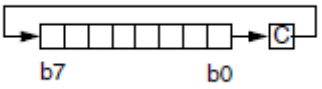


Figura 4.23: Instrucción ROR.

Para que la comunicación sea correcta debe haber retardos muy precisos entre bit y bit, observe la figura para comprender el tipo de protocolo que utiliza el puerto serial y compárelo con el programa. Primero se tiene aunque no se ve en la figura el tiempo de espera (1 lógico) el cual puede llegar a ser indefinido ya que este punto depende enteramente del sistema y sus tiempos de transmisión, después viene el bit de arranque el cual es un cero lógico, el tiempo de este será el correspondiente a un bit, después se transmite la palabra completa empezando por el bit menos significativo cada bit tiene la misma duración la cual depende de la velocidad de comunicación y por ultimo viene el bit de paro el cual es un uno lógico, después el ciclo se repite viniendo de nuevo el bit de arranque, pero dado el caso en el que no se transmites mas datos el bit de paro se convierte en tiempo de espera (1 lógico).
En este caso la comunicación se realizará a una velocidad de 2400 baudios lo que quiere decir 2400 bits por segundo por lo cual cada bit tiene una duración de 416.667us.

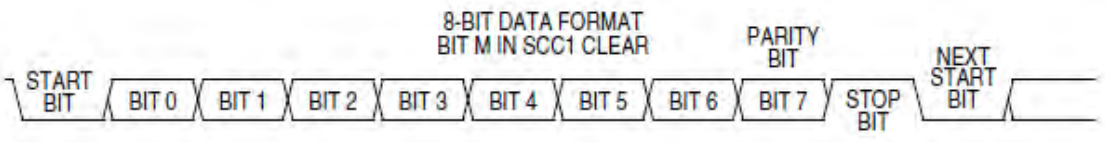


Figura 4.24: Protocolo de comunicación serial a 8 bits.

No es posible conectar directamente al MCU con el puerto serie del PC debido a que los niveles lógicos de estos son diferentes por lo que se recurre al uso de un transeiver que no es más que un traductor de niveles entre los TTL o CMOS con los del puerto serie, el dispositivo a usar es el MAX232ECPE.

PROGRAMA

```
; variable/data section
;
        ORG    Z_RAMStart
CONT_A:     DS.B   1            ; REGISTRO PARA EL ALMASENAMIENTO DE UNA CUENTA
TRANSMI:    DS.B   1            ; REGISTRO DEL DATO A TRANSMITIR
;
; code section
;
        ORG    ROMStart
; Tabla de datos para ser grabados en memoria ROM
MEN_INI:    FCB $22,$48,$4F,$4C,$41,$5F,$4D,$55,$4E,$44,$4F,$22
                        ;"   H    O    L   A    _    M   U   N   D   O   "
; Include device initialization code
        INCLUDE 'MCUInit.inc'

_Startup:
        LDHX   #RAMEnd+1       ; initialize the stack pointer
        TXS
        ; Call generated Device Initialization function
        JSR    MCU_init
;*****************************************************************
mainLoop:
        ; Insert your code here
                CLRX
                CLRH

SIG_PAL:    LDA MEN_INI,X          ;CARGA EN EL ACULULADOR EL PRIMER DATO DEL MENSAGE INICIAL
                                   ; MAS EL DESPLAZAMIENTO EN X
                STA TRANSMI        ; ALMACENA EL ACUMULADOR EN EL REGISTRO DEL DATO A RTANMITIR
                BSR TX             ; BIFURCA A LA SUBRRUTINA DE TRANSMISION
                 INCX              ; INCREMENTA EN UNO AL REGISTRO X
                 CBEQX #$0C,FIN_M  ; SI X ES IGUAL A $0C SIGNIFICA QUE TODO EL MENSAJE
                                   ; SE HA TRANSMITIDO Y BIFURCA A FIN DE MENSEJE (FIN_M)

                BRA   SIG_PAL      ; BIFURCA PARA TRANSMITIR LA SIGUIENTE LETRA

FIN_M:      BRSET 1,PTA,*            ; ESPERA A QUE EL BOTON SEA PRESIONADO
            BRCLR 1,PTA,*            ; ESPERA A QUE EL BOTON SEA SOLTADO
```

```
            MOV   PTB,TRANSMI      ; MUEVE EL CONTENIDO DEL PUERTO B AL REGISTRO TRANSMISOR
            BSR   TX               ; BIFURCA A LA SUBRRUTINA DE TRANSMISION

            BRA   FIN_M            ; BIFURCA A FIN_M PARA ESPERAR OTRO ENTER
;*****************************************************************
; SUBRRUTINA PARA LA TRANSMISION SERIAL SIN RETORNO A CERO
;*****************************************************************
TX:         MOV   #$08,CONT_A      ; NUMERO DE BITS A TRANSMITIR
            BCLR  0,PTA            ; PTA0=1 BIT DE ARRANQUE
            BSR   RET_1BIT         ; RETARDO 416.667 MICROSEGUNDOS 2400 BAUDIOS

BIT_SIG:    ROR   TRANSMI          ; ROTAR A LA DERECHA, EL BIT MENOS SIGNIFICATIVO
            BCC   ET_08            ; BIFURCA SI EL BIT CARRY ES CERO
            BSET  0,PTA            ; PTA0=1; SE TRANSMITE UN UNO
            BRA   TRANS_0

ET_08:      BCLR  0,PTA            ; PTA0=0; SE TRANSMITE UN CERO

TRANS_0:    BSR   RET_1BIT         ; RETARDO 416.667 MICROSEGUNDOS 2400 BAUDIOS
                                   ; PTA2 PERMANECE EN CERO Y SE TRANSMITE
            DEC   CONT_A
            BNE   BIT_SIG          ; BIFURCA SI AUN QUEDAN BITS POR TRANSMITIR
            BSET  0,PTA            ; BIT DE PARO
            BSR   RET_1BIT
            RTS                    ; RETORNA DE SUBRRUTINA
;*****************************************************************
; RETARDO 1BIT, 2400 BAUDIOS 416.667 MICRO SEG CON BUS DE 8 MHz
;*****************************************************************
RET_1BIT:
            LDA   #$ED
LOOP4:      BRN   *
            BRN   *
            BRN   *
            NOP
            NOP

            DBNZA LOOP4
            RTS
;*****************************************************************
```

CIRCUITO

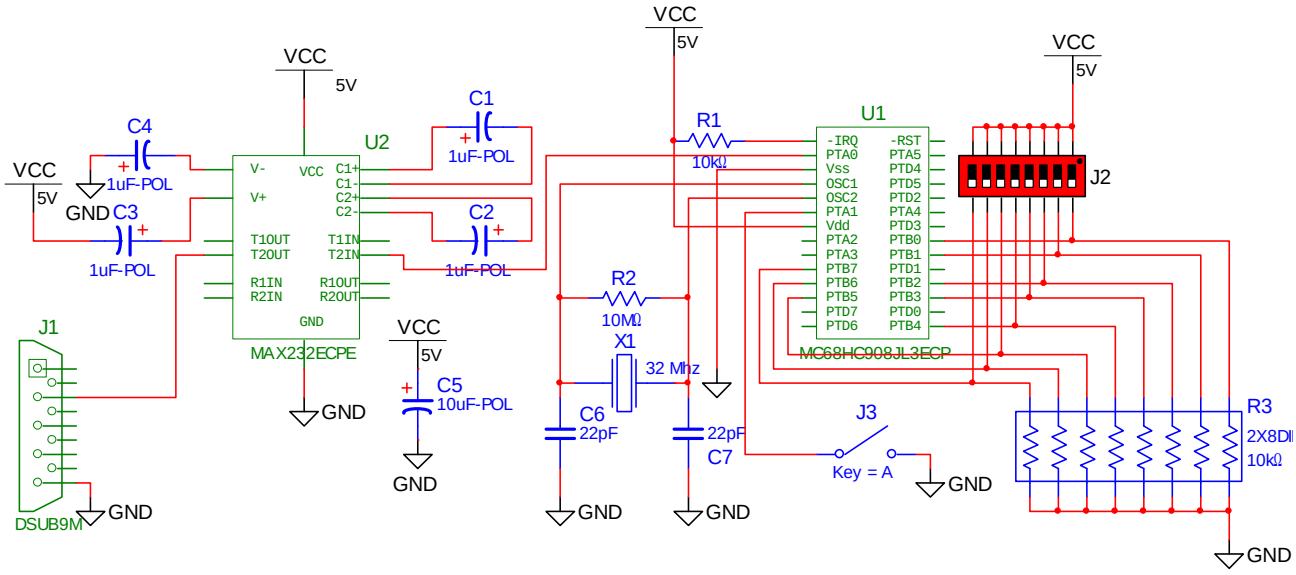


Figura 4.25: Circuito de comunicación entre PC y MCU.

NOTA: Para ajustar la hyper terminal en un entorno windows siga la siguiente ruta: inicio – todos los programas – accesorios – comunicaciones – hyper terminal, con esto debería aparecer la siguiente ventana:

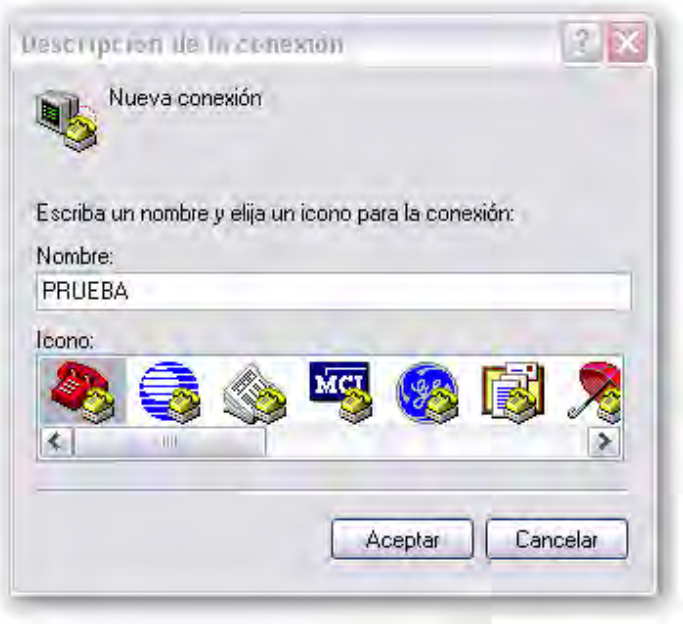


Figura 4.26: Crear conexión en la hyper Terminal.

Elija un nombre para la conexión y presione aceptar con lo que aparecerá la siguiente ventana en donde podrá elegir el puerto serie de su preferencia por lo que si se trata de una computadora de escritorio elija el COM 1 que corresponde por lo regular al primer o único puerto serial presente en la tarjeta madre de un PC de escritorio convencional.

Figura 4.27: Selección de puerto serie.

Al presionar aceptar aparecerá la siguiente ventana en donde debe elegir la velocidad de comunicación, el tamaño de la palabra, la paridad, los bits de parada y el control de flujo, establezca todas estas opciones tal y como se muestra en la ventana, se elige una velocidad de 2400 baudios debido a que el MCU fue programado a esta velocidad.

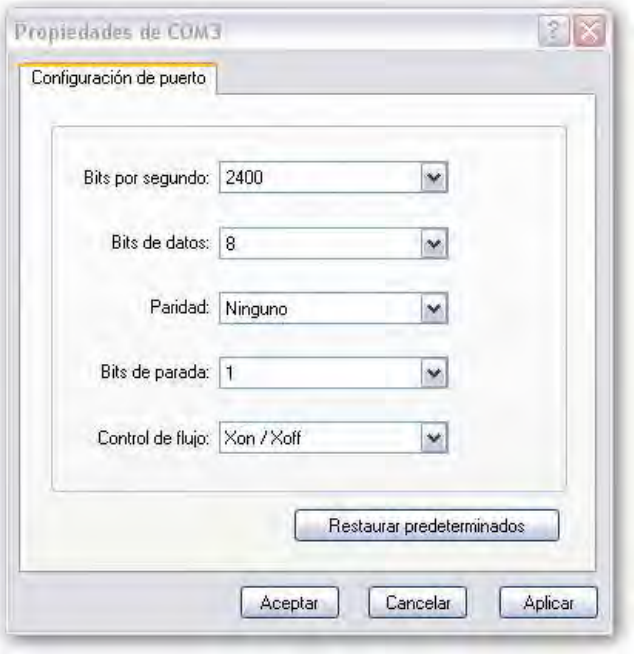


Figura 4.28: Propiedades del puerto.

Ya hecho esto aparecerá la ventana de la Hyper terminal con la conexión abierta, ahora puede conectar al MCU con el PC y polarizarlo con lo que debería aparecer el mensaje inicial.

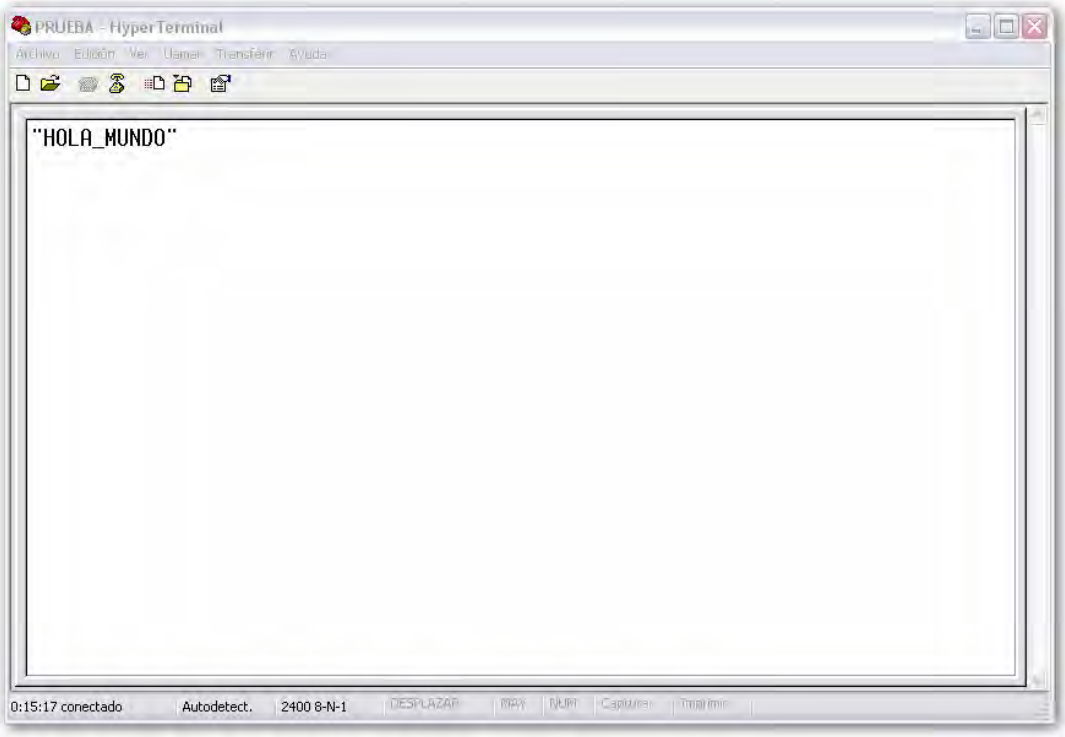


Figura 4.29: Hyper terminal con conexión abierta.

Si desea modificar las propiedades del puerto cierre la conexión presionando  y diríjase a archivo – propiedades, ya realizado los ajustes presione  para abrir de nuevo la conexión.

4.3.5 MANÓMETRO DIGITAL

FUNCION
Utilizando el MCU MC68HC908GP32ECPE.
Sistema para la medición de la presión en neumáticos pequeños con presiones no mayores a 36 Pci con resolución de un Pci. Si la medición se sale de rango es decir si la presión medida es mayor a 36 Pci el punto del display con el digito menos significativo prende indicando este hecho. El sensor a utilizar es el MPX4250GP.

INICIALIZACIÓN
CPU: Indique a este módulo que usará un cristal de 32 Mhz para obtener un bus de 8 Mhz y coloque a la CPU en modo usuario.
PTA: Coloque todo el puerto A como salida y con un valor inicial de $00
PTD: Coloque el puerto PTD 4 y PTD 5 como salida con valor inicial en ambos de 0.
ADC: Ajuste el ADC tal y como se muestra en la imagen. Utilice el reloj del bus interno, el prescaler en 8, el modo de conversión en continua, active un solo canal de entrada y como entrada inicial el canal cero.

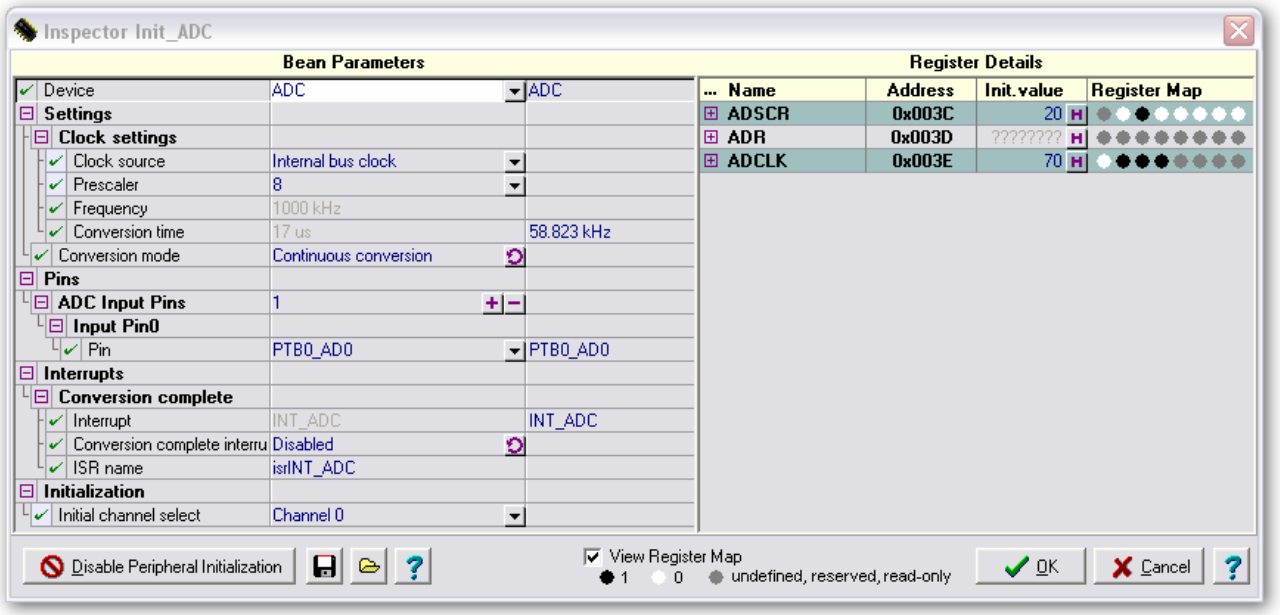


Figura 4.30: Ajustes de ADC.

DETALLES

Antes que nada debe tomarse en cuenta la forma de programar displays de 7 segmentos que de hecho la mayoría son de 8 debido al punto decimal que poseen, en este caso se muestra la tabla de códigos para displays de 7 segmentos cátodo común por lo que se requiere de lógica positiva para su programación.

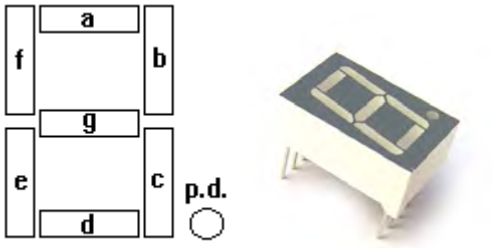


Figura 4.31: Display de 7 segmentos.

La tabla muestra en las filas 1 y 2 como se conectan los segmentos del display con los puertos del MCU. La columna 1 muestra los caracteres a los que se les esta obteniendo el código. La última columna muestra los códigos en lógica positiva de cada carácter presente en la primera columna.

Nótese que todos los caracteres tienen el segmento H en cero esto quiere decir que el segmento del punto decimal siempre estará apagado pero ya no es necesario hacer otra tabla para caracteres con punto ya que en todos los casos el segmento H vale lo mismo, desde el punto de vista del puerto H vale $80 por lo que por ejemplo si al carácter 3 el cual su código es $4F se le desea agregar el punto solo hay que sumarle el valor del punto funciona igual que una suma:

CARÁCTER 3 + CARÁCTER PUNTO = $4F + $80 = $CF = CARÁCTER 3 CON PUNTO

	H	G	F	E	D	C	B	A	
No Dec	PTA 7	PTA 6	PTA 5	PTA 4	PTA 3	PTA 2	PTA 1	PTA 0	HEXA
0	0	0	1	1	1	1	1	1	3F
1	0	0	0	0	0	1	1	0	06
2	0	1	0	1	1	0	1	1	5B
3	0	1	0	0	1	1	1	1	4F
4	0	1	1	0	0	1	1	0	66
5	0	1	1	0	1	1	0	1	6D
6	0	1	1	1	1	1	0	1	7D
7	0	0	0	0	0	1	1	1	07
8	0	1	1	1	1	1	1	1	7F
9	0	1	1	0	0	1	1	1	67

Figura 4.32: Tabla de códigos para displays cátodo común (lógica positiva).

Otros caracteres que pueden escribirse en un display de 7 segmentos son A, b, C, c, d, E, y F.

PROGRAMA

```
; variable/data section
        ORG    Z_RAMStart
TEMP:      DS.B   2
DIG_MEN:   DS.B   1
DIG_MAS:   DS.B   1

        ORG    RAMStart        ; Insert your data definition here
;
; code section
;
        ORG    ROMStart
NUM_7SEG:  FCB $3F,$06,$5B,$4F,$66,$6D,$7D,$07,$7F,$67,$FD
                        ;0  1    2    3    4    5    6     7    8    9  (6+.=6.=7D+80=FD)
; Include device initialization code
        INCLUDE 'MCUInit.inc'

_Startup:
        LDHX   #RAMEnd+1        ; initialize the stack pointer
        TXS
        ; Call generated Device Initialization function
        JSR    MCU_init

mainLoop:
        ; Insert your code here
              BSR    RETARDO
              LDHX   #$0000
              STHX   TEMP
```

```
;*******************************************************************
            BRCLR   7,ADSCR,*   ;EN ESPERA POR UNA CONVERSION
            LDA     ADR         ;SE CARGA EN EL ACUMULADOR EL VALOR DE LA CONVERSION
                                ;EL FABRICANTE RECOMIENDA QUE SE DESPRESIE LA PRIMERA
                                ;CONVERSION

REPITE:     BRCLR   7,ADSCR,*   ;ESPERA POR LA SEGUNDA CONVERSION
            LDA     ADR


            CMP     #$0A        ;COMPARA LA CONVERSION CON #$0A
            BLO     IMP_00      ;BIFURCA SI EL ACUMULADOR ES MENOR

            CMP     #$FA        ;COMPARA LA CONVERSION CON #$FA
            BHI     IMP_36      ;BIFURCA SI EL ACUMULADOR ES MAYOR
            BRA     ALGO        ;BIFURCA AL INICIO DEL ALGORITMO

IMP_00:     CLR     DIG_MAS
            CLR     DIG_MEN
            BRA     IMPRIME     ;IMPRIME DIRECTO 00 PCI

IMP_36:     MOV     #$03,DIG_MAS
            MOV     #$0A,DIG_MEN
            BRA     IMPRIME     ;IMPRIME DIRECTO 36. CON PUNTO AL FINAL
                                ;INDICATIVO QUE LA PRESION ESTA FUERA DE RANGO
;*******************************************************************
;ALGORITMO DE CONVERSION
;*******************************************************************
ALGO        SUB     #$0A        ;RESTALE AL ACUMULADOR #$0A
            LDX     #$0F        ;CARGA EN EL REGISTRO EL VALOR DE #$0F
            MUL                 ;MULTIPLICA (X)(A)=(XA)

            STX     TEMP        ;MUEVE EL CONTENIDO DE "X" A "H"
            LDHX    TEMP        ;APOYANDOSE EN DOS LOCALIDADES DE
MEMORIA


            LDX     #$64
            DIV                 ;DIVIDE (HA)/(X)=(A)...RESIDUO=(H)

            CLRH                ;LIMPIA EL REGISTRO H
            LDX     #$0A

            DIV                 ;DIVIDE (HA)/(X)=(A)...RESIDUO=(H)
            STHX    DIG_MEN     ;ALMACENA EL NUMERO DIGITO DICIMAL DECENA
```

```
            STA    DIG_MAS      ;ALMACENA EL DIGITO DECIMAL UNIDAD
;*****************************************************************
;VISUALIZA EN DISPLAYS DE 7 SEGMENTOS
;PTD4 CONTROLA EL DIGITO MENOS SIGNIFICATIVO
;PTD5 CONTROLA EL DIGITO MAS SIGNIFICATIVO
;*****************************************************************
IMPRIME:        CLRH

            LDX    DIG_MEN      ;CARGA EN X EL VALOR DEL DIGITO MENOS SIGNIFICATIVO
            LDA    NUM_7SEG,X   ;CARGA EN A EL CODIGO PARA VISUALIZAR EL DIGITO MENOS
                                ;SIGNIFICATIVO
            STA    PTA          ;MANDA EL COGIDO DEL DIGITO MENOS SIGNIFICATIVO AL DISPLAY
            BSET   4,PTD        ;ACTIVA EL DISPLAY DE 7 SEGMENTOS MENOS SIGNIFICATIVO
            BSR    RETARDO      ;MANTENLO PRENDIDO UN MOMENTO.
            BCLR   4,PTD        ;APAGA EL DISPLAY DE 7 SEGMENTOS MENOS SIGNIFICATIVO

            LDX    DIG_MAS      ;CARGA EN X EL VALOR DEL DIGITO MAS SIGNIFICATIVO
            LDA    NUM_7SEG,X   ;CARGA EN A EL CODIGO PARA VISUALIZAR EL DIGITO MAS
                                ;SIGNIFICATIVO
            STA    PTA          ;MANDA EL COGIDO DEL DIGITO MAS SIGNIFICATIVO AL DISPLAY
            BSET   5,PTD        ;ACTIVA EL DISPLAY DE 7 SEGMENTOS MAS SIGNIFICATIVO
            BSR    RETARDO      ;MANTENLO PRENDIDO UN MOMENTO.
            BCLR   5,PTD        ;APAGA EL DISPLAY DE 7 SEGMENTOS MAS SIGNIFICATIVO

            BRA    REPITE
;*****************************************************************
;RETARDO DE 8.2 MILISEGUNDOS, LO QUE PROBOCA UNA FRECUENCIA DE OPERACION
;EN LOS DISPLAYS DE 120 Hz CON UN CICLO DE TRBAJO AL 50%
;*****************************************************************
RETARDO:  CLRX
LOOP2:    CLRA
LOOP:     CBEQA #$FF,FIN
          INCA
          BRA   LOOP

FIN:      CBEQX #$1F,FIN2
          INCX
          BRA   LOOP2

FIN2:     RTS
;*****************************************************************
```

ANÁLISIS Y DISEÑO DEL ALGORITMO

Tenemos que tener en cuenta varios conceptos:

Funcionamiento del sensor MPX4259GP: LINEAL
SPAWN: 0.2v – 4.9v
RANGO: 0KPa – 250 KPa
RANGO equivalente: 0Pci – 36 Pci

Funcionamiento del ADC: LINEAL
SPAWN: 0.0v – 5.0v
RANGO: estado 0 – estado 255, 8 bits
Valor en volts por estado: 19.6 volts/estado

ALINEASION DE RANGOS POR SOFTWARE: consiste en alinear el rango de funcionamiento del convertidor analógico digital con el del sensor, para lograr esto devemos saber que valor en Hexadecimal nos arrojará el ADC con los voltajes de 0.2v y de 4.9v que no son mas que el spawn del sensor, esto lo logramos con la siguiente formula:

$$\frac{\text{Vspawn (255 estados del ADC)}}{5\text{v}}$$

Sustituyendo con la parte baja del spawn del sensor:

$$\frac{0.2\ (255)}{5\text{v}} = 10.2$$

Por lo que el ADC arrojará el valor 10 en decimal o $0A en hexadecimal correspondiente al valor mínimo entregado por el sensor.

Haciendo lo mismo con la pare alta del spawn del sensor:

$$\frac{4.9\ (255)}{5\text{v}} = 249.9$$

Por lo que el ADC arrojará el valor de 250 decimal o $FA en hexadecimal.

CANTIDAD DE STADOS UTILES: El siguiente paso es obtener el número de estados del ADC utilizados por el sensor, esto se obtiene restando los valores del spawn entregados por el ADC.

$$\$FA - \$0A = \$F0 = \text{a } 240 \text{ decimal}$$

FACTOR DE CONVERSIO: Para obtener el factor de conversión es necesario dividir el valor máximo que puede medir el sensor ya sea en Pci o en Kpa entre la cantidad de estados

utilizados por el sensor, como en la medición de presión en neumático es un estándar los Pci utilizaremos estos.

$$\frac{36\ Pci}{240} = 0.15$$

Puesto que el MCU no puede operar números decimales debemos multiplicar a este factor de conversión por algún número que lo vuelva un entero, he seleccionado para esto el número 100 decimal que en hexadecimal resulta ser $64.

$$(0.15)\ (100) = 15 = \text{a } \$0F \text{ hexadecimal}$$

ALGORITMO FINAL PARA EL MCU: Primero es necesario restarle al valor arrojado por el ADC la parte inicial del SPAWN.

ADR - $0A = Dato alineado

El siguiente paso es multiplicar este dato ya alineado por el factor de conversión vuelto entero.

Dato alineado ($0F) = dato convertido 100 veces mayor al real

A continuación es necesario eliminar el aumento en 100 que le fue otorgado al factor de conversión para volverse un entero para esto solo dividimos entre 100 decimal o $64 hexadecimal.

$$\frac{\text{Dato convertido 100 veses mayor al real}}{\$64} = \text{valor en hexadecimal de la presión que hay en el sensor}$$

Ahora simplemente hay que dividir el valor de la presión obtenida entre 10 decimal o $0A hexadecimal, esto se hace para convertir el valor de la presión que se encuentra en hexadecimal a decimal, el resultado de esta división nos da el dígito de las centenas el residuo de la división resulta ser el digito de las unidades.

$$\frac{\text{Presión en hexadecimal}}{\$0A} = \text{centenas de la presión, residuo = unidades de la presión}$$

Este algoritmo es para que el MCU convierta el valor del voltaje en presión pero también nos sirve para comprobar su funcionamiento, observe el ejemplo.

EJEMPLO: Si en la salida del sensor se mide un voltaje de 2.55v ¿Cuál será el valor de la presión?

$$\frac{2.55v\ (255)}{5v} = 130.05d = \$82$$

$$(\$82 - \$0A)\ (\$0F) = \$0708$$

$$\frac{\$0708}{\$64} = \$12$$

$$\frac{\$12}{\$0A} = 1 \quad \text{residuo} = 8$$

Por lo que la presión que se encuentra excitando al sensor es de 18 Pci.

NOTA: Note que el programa excluye los valores del ADC menores a $0A (0.2v) para imprimir directamente 00 Pci y los valores mayores a $FA (4.9v) para imprimir directamente 36 Pci con un punto indicativo de que la presión esta fuera del rango del sensor.

CIRCUITO

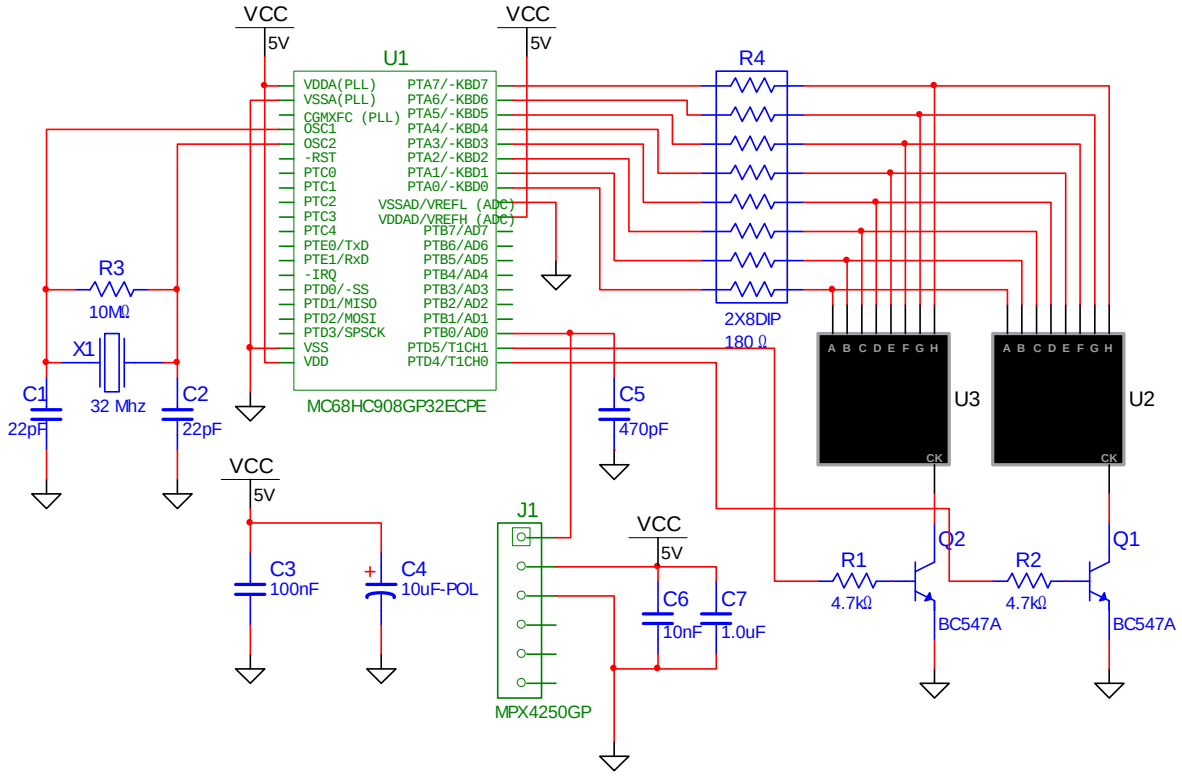


Figura 4.33: Circuito del sensor de presión.

NOTA DEL CIRCUITO: Si estudia el programa y el circuito con detenimiento notará que en ningún momento los dos displays están prendidos al mismo tiempo ya que esto podría sobrecargar a los puertos o hasta dañarlos, el resultadotas más probable de esto son

caracteres con segmentos disparejos en intensidad, en realidad los displays encienden periódicamente a una frecuencia de 120 Hz con un ciclo de trabajo al 50 % este ciclo de trabajo esta dado por el número de display, para saber el ciclo de trabajo en otros casos solo divida 100% entre el número de displays.
La corriente en cada segmento del display esta dada por el siguiente circuito equivalente siempre y cuando de suponga que el transistor este saturado, que el display sea de color rojo y recordando que al cargar 10mA en el puerto se cae a 3.5v.

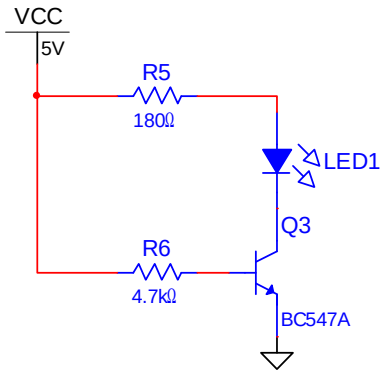


Figura 4.33: Circuito equivalente para cada segmento de los displays.

El análisis de este circuito se ha visto en capítulos anteriores por lo que sabemos que la corriente en el LED es de 10mA.

El par de transistores BC547A que aparecen en el circuito tienen una resistencia de base de 4.7KΩ tomando en cuenta que este transistor en el caso de ser fabricado por motorota el datasheet indica que posee una beta de 90 esta resistencia es necesaria para lograr una saturación lo suficientemente grande como para que la corriente colector – emisor pueda llegar a ser de 80mA es decir que el transistor permita el paso de corriente equivalente a 8 segmentos ya que cada segmento equivale a una carga de 10mA.

4.4 RESUMEN

La importancia de un algoritmo radica en mostrar la manera de llevar a cabo procesos y resolver mecánicamente problemas matemáticos o de otro tipo. Al igual que las funciones matemáticas, los algoritmos reciben una entrada y la transforman en una salida, comportándose como una caja negra. Sin embargo, no toda caja negra que convierta una entrada en una salida se puede considerar un algoritmo: para que un algoritmo pueda ser considerado como tal, debe ser una secuencia ordenada, finita y definida (formalización de su comportamiento) de instrucciones. De este modo se puede seguir y predecir el comportamiento del algoritmo para cualquier entrada posible, a partir del seguimiento de esa secuencia de instrucciones, que como es ordenada y definida, no da lugar a ambigüedades y puede seguirse sus respuestas.

La parte mas complicada en el diseño de un sistema con MCU es la implementación del algoritmo que puede ser tan básico como un vólmetro hasta llegar a la complejidad de un filtro digital, o le procesamiento digital de señales.

CAPÍTULO

5

MIGRACIÓN A NUEVAS TECNOLOGÍAS

OBJETIVOS

Después de estudiar el ejemplo en este capítulo usted podrá ser capas de:

- *Fabricar el circuito impreso una herramienta de desarrollo OSBDM.*
- *Programar el MCU bajo el cual esta diseñado esta herramienta a través del puerto USB*
- *Instalar este dispositivo en un entorno windows*

La tecnología crece constantemente y a veces estar a la vanguardia puede llegar a ser un tanto costoso debido a los elevados precios de los sistemas de desarrollo profesionales, este es el caso de los MCU HCS08 que actualmente están en auge y cuentan con una gran variedad de microcontroladores además de ser mas rápidos, económicos y tener la ventaja de poder selecciona la velocidad de su bus a través de software evitando así el uso de un cristal.

5.0 UTILIDAD----------

El puerto serial a desaparecido casi en su totalidad en computadores portátiles por lo que la aparición de nuevos dispositivos con un puerto USB incluido es lo mas común en la actualidad, aquí es donde esta herramienta nos es útil, una forma mucho mas fácil de depurar y grabar nuestros programas con nuevos y mejores microcontroladores.

La herramienta OSBDM no necesita de una fuente externa ya que su alimentación proviene del mismo puerto USB por lo que se puede depurar el programa simulándolo en el mismo MCU reduciendo así los tiempos de diseño.

5.0.1 CIRCUITO

COMPONENTES: La lista de componentes puede obtenerla del circuito, los dos primeros componentes no son sencillos de conseguir:

SN74LVC1T45DBVR (2): Transeiver con nivel seleccionable
MC68HC908JB16DW (1): Microcontrolador de la familia HC08
U4 (1): conector USB clase B en 90 grados
78MR33 (1): Regulador de voltaje 3.3v

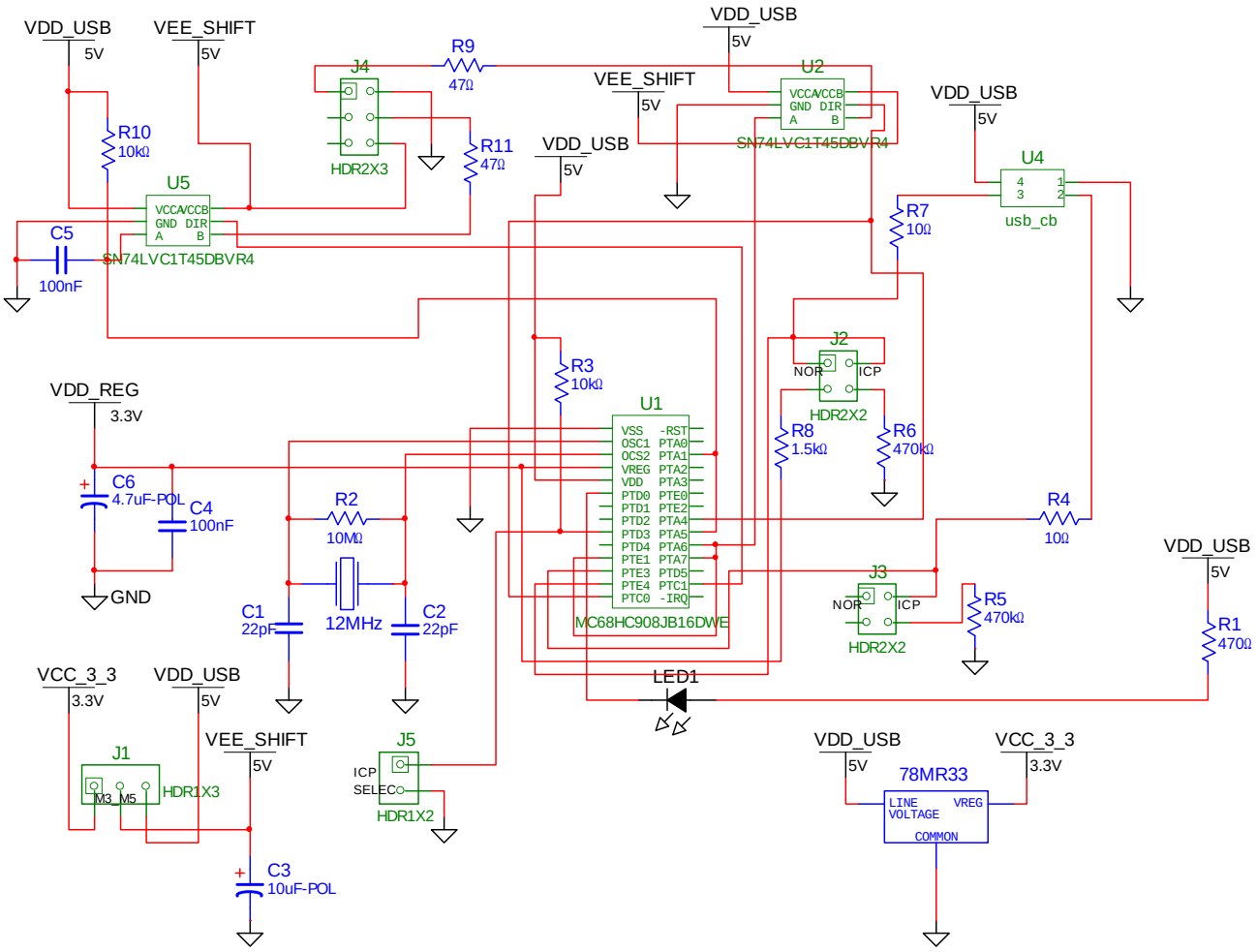


Figura 5.1: Circuito OSBDM.

5.0.2 PCB

El proyecto multisim y ultiboard puede encontrarlo en el DVD anexo al final de este trabajo

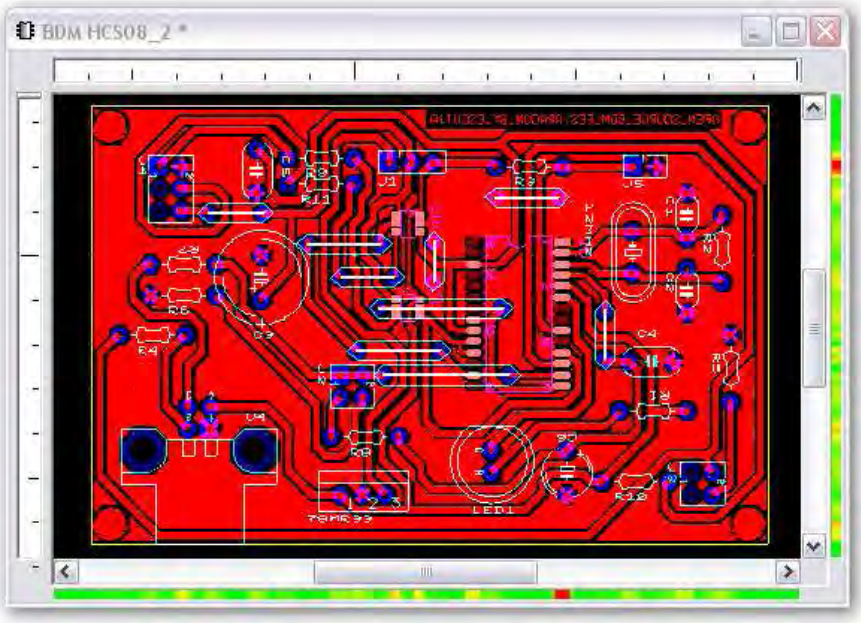


Figura 5.2: PCB del OSBDM.

5.0.3 ESPEJO DEL CIRCIUITO

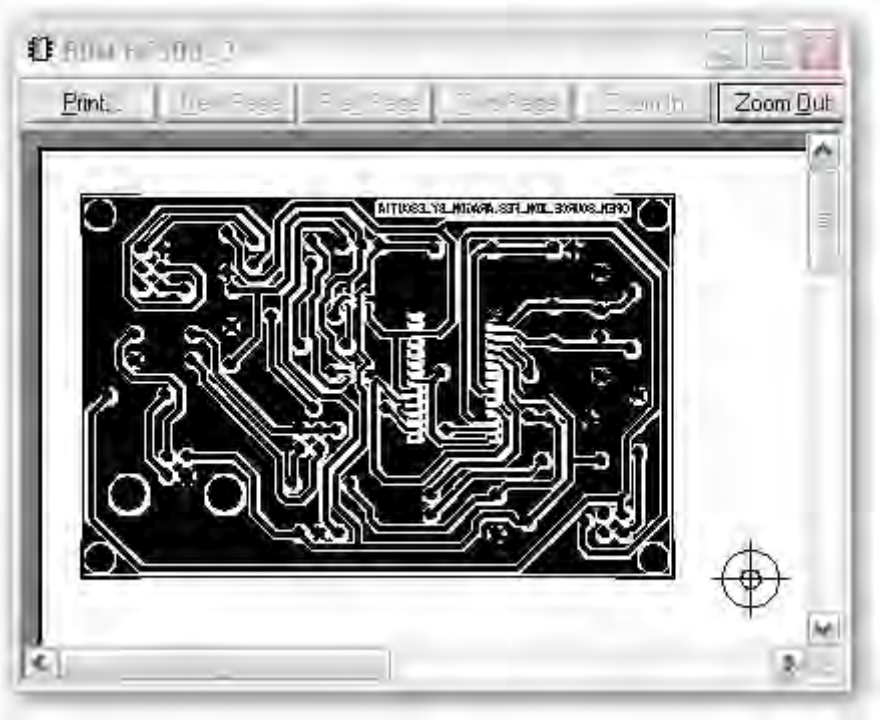


Figura 5.3: Espejo del circuito a transferir del OSBDM.

5.0.4 RESULTADO FINAL

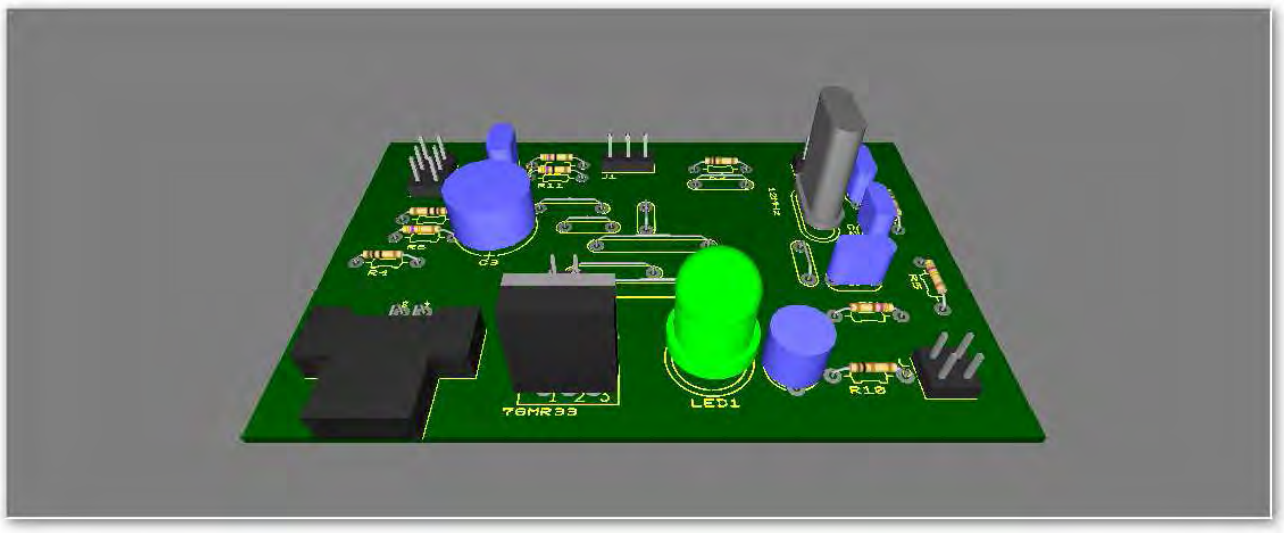


Figura 5.4: Aproximación del resultado final.

5.0.5 GRABADO DEL MICROCONTROLADOR

Coloque los jumpers en los headers tal y cual se muestra en la imagen

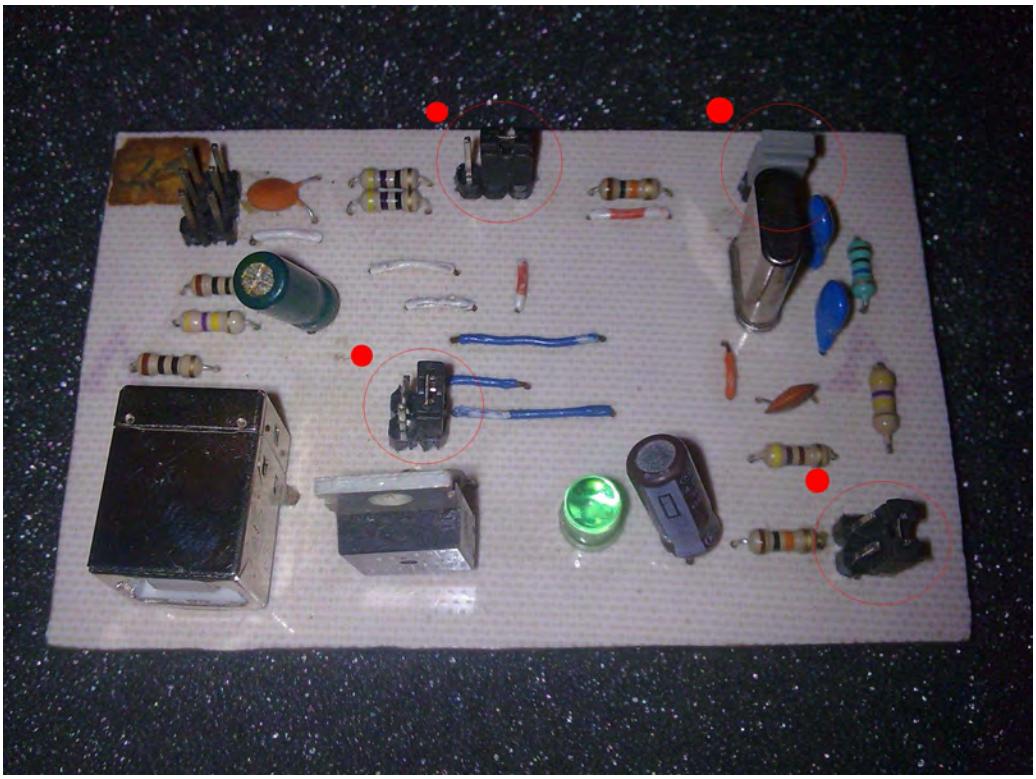


Figura 5.5: Colocación de jumpers para la grabación.

Conecte el cable USB al OSBDM y al PC con lo que debería aparecer la siguiente ventana:

Figura 5.6: Instalación manual.

Escoja la opción mostrada en la en la figura y presione next, a continuación seleccione la ubicación del controlador el cual se encuentra en el DVD anexo al final de este trabajo. La ubicación es: DVD – OSBDM - ICP_Application - **PC Software**, ya seleccionada esta carpeta presione aceptar.

Ya que la tarjeta sea instalada y reconocida ejecute el programa USBICP el cual se encuentra dentro de la misma carpeta **PC Software** con lo que aparecerá la siguiente ventana, si no aparece la dirección mostrada en la ventana solo tiene que colocar la dirección mostrada en el párrafo anterior.

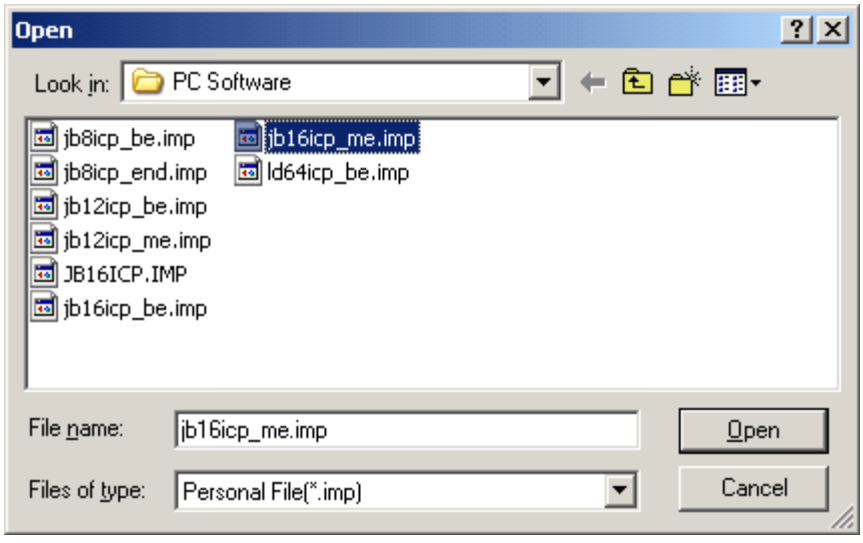


Figura 5.7: Selección del micro a grabar.

Seleccione la opción marcada y presione abrir con lo que aparecerá la siguiente ventana

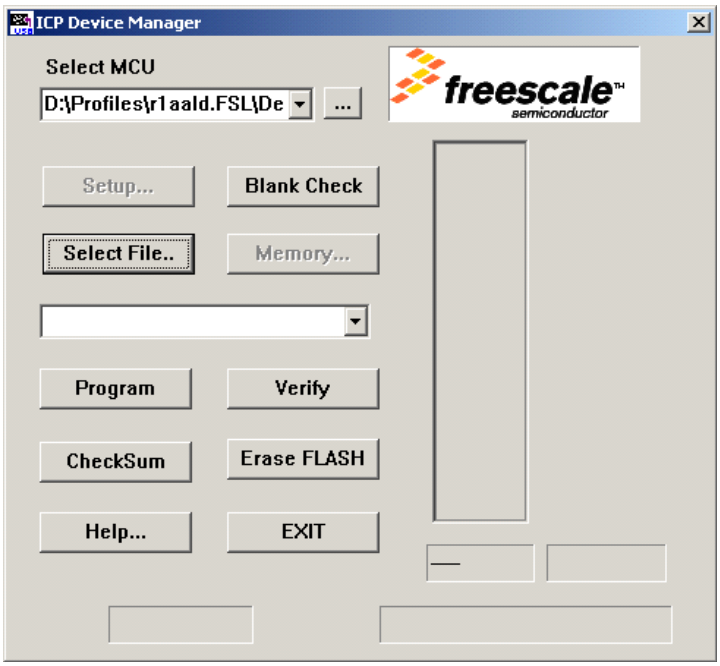


Figura 5.8: Programa para la grabación vía USB.

Aquí debe presionar el botón SELECT FILE y colocar la dirección del archivo .s19 del grabador, el cual no es mas que el programa a grabar en el MCU en código hexadecimal, este archivo se encuentra en el DVD en la dirección DVD – OSBDM – OpenSource BDMFirmware, dentro de esta carpeta se encuentra el archivo **opensourcebdm.abs.s19,** selecciónelo y presione abrir, por ultimo seleccione el botón PROGRAM para que empiece la grabación del MCU.
Ya grabado el MCU desconecte la tarjeta para retirar y acomodar los jumpers tal y como se muestra en la imagen.

Figura 5.9: Ajuste de jumpers.

Ahora conecte de nuevo la tarjeta y repita el proceso de instalación manual solo que en este caso indique la siguiente dirección: DVD – OSBDM - OpenSourceBDM_S08_WinDriver con lo que se instalara el controlador LibUsb-win32.

Figura 5.10: Instalación de controlador.

Su tarjeta esta lista para funcionar, para grabar MCUs a 3 voltios coloque el jumper tal y como se muestra en la siguiente figura, si desea grabar MCUs a 5 voltios coloque el jumper tal cual se ve en la imagen anterior, para saber con que voltaje grabar el MCU consulte el manual del MCU que este empleando.
El pin 1 del conector BDM es el que se puede ver marcado con una etiqueta en la trarjeta en la figura.

Figura 5.11: Colocación de jumpers para la grabación a 3 voltios.

EJEMPLO DE CONEXIÓN ENTRE UN MICRO Y OSBDM

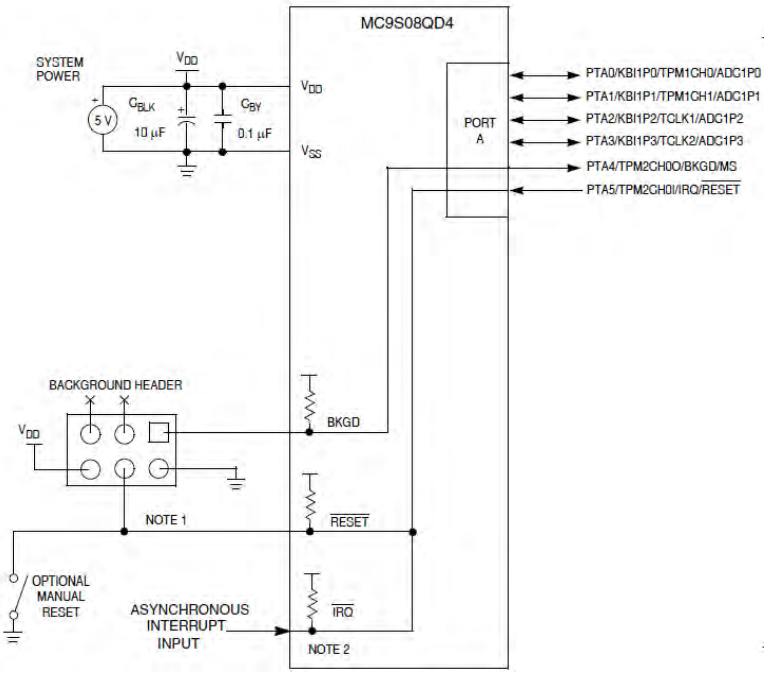


Figura 5.12: Conexión entre un MCU y el OSBDM.

El pin 1 del conector BDM está marcado en la siguiente figura y su configuración es idéntica a la de la figura 5.12

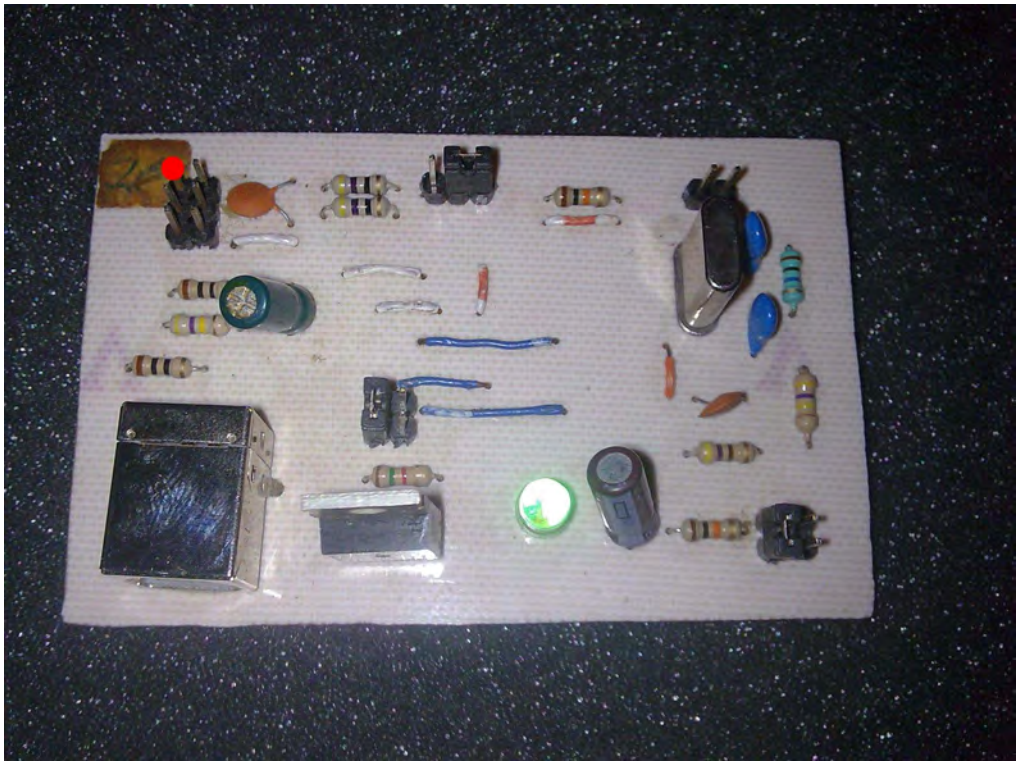


Figura 5.13: Pin 1 del conector BDM.

5.1 RESUMEN----------

La fabricación de este dispositivo no es muy compleja pero debe tener especial cuidado en revisar cada una de las pistas buscando posibles cortos o pitas abiertas especialmente en la zona donde va soldado el MCU ya que una ves soldado no podrá arreglar nada que este debajo de el.
El movimiento de los jumpers tiene un solo objetivo el cual es que el PC reconozca a la tarjeta como un dispositivo diferente. De un modo el PC reconoce a la tarjeta como un dispositivo USB el cual necesita su propio controlador, y de esta forma se graba el MCU, colocando los jumpers del otro modo y ya grabado el MCU el PC lo reconoce como otro dispositivo que también requiere de su propio controlador.

El diseño mostrado del grabador BDM es una versión reducida y mejorada, la documentación original está presente en un archivo en el disco incluido en esta tesis.

CONCLUSIONES

En este trabajo de tesis se a logrado comprender y exponer detalladamente y a conciencia el set de instrucciones que poseen los microcontroladores HC08, el énfasis que se le dio a este capítulo se debió a que es de suma importancia de hecho creo que la habilidad mas importante de un diseñador de sistemas embebidos es la destreza que el posea al programar el microcontrolador, la forma y el momento en que decide utilizar un tipo de direccionamiento en ves de otro, o una instrucción en ves de otra, esto se adquiere con practica y en especial conociendo bien el potencial de cada instrucción y direccionamiento.

Se exponen algunos programas y ejemplos didácticos con una dificultad baja o intermedia, esto es para que el alumno tenga una sencilla introducción a la programación, aunque también se explica un diseño embebido de dificultad intermedia, el cual resulta ser una aplicación practica que puede ser de ayuda al lector para estimular su imaginación y a entender el rol de las instrucciones aritméticas en el diseño de un sistema embebido, y por ultimo se muestra un sistema embebido con un programa de nivel experto para que el alumno sepa lo que puede llegar a hacer si se esfuerza y practica mucho.

No se muestran muchos ejemplos en el presente trabajo, pero esto tiene una explicación ya que se desea que el alumno estimule su creatividad y no opte solo por copiar pedazos de lo que ya existe para generar su trabajo.

Queda abierto el presente trabajo de tesis para realizar una continuación evocada tal ves ha microcontroladores de 8 o 16 bits pero esta ves la programación mostrada sea en lenguaje C, ya que en el software Codewarrior pueden escribirse los programas no solo en lenguaje ensamblador sino también en C o C++.

Si este trabajo escrito no le resuelve alguna duda sobre los microcontroladores se le recuerda al lector que todo esta disponible en la hoja de datos del microcontrolador usado, y solo es cuestión de poner un poco de atención y paciencia para resolver el problema.

Suerte en sus diseños.

APENDICE

PROGRAMA PRINCIPAL DEL SISTEMA DE MONITOREO

El programa se muestra desde la inicialización de registros

```
;
; Variable/data section
;
            ORG    Z_RAMStart        ; Insert your data definition here
;*******************************************************************
PRES_ADC_PP1:  DS.B   1       ;Declaración de registros en memoria RAM
PRES_ADC_PP3:  DS.B   1       ;Necesarios para el almacenamiento
PRES_ADC_PP4:  DS.B   1       ;de datos extraídos del convertidor
PRES_ADC_PP7:  DS.B   1       ;analógico digital para cada uno de los sensores
PRES_ADC_PP8:  DS.B   1
PRES_ADC_PP5H: DS.B   1
PRES_ADC_PP5L: DS.B   1
PRES_ADC_TP1H: DS.B   1
PRES_ADC_TP1L: DS.B   1
PRES_ADC_TP3H: DS.B   1
PRES_ADC_TP3L: DS.B   1
PRES_ADC_TP4H: DS.B   1
PRES_ADC_TP4L: DS.B   1
;*******************************************************************
TEMP_H:      DS.B   1        ; Declaración de registros en memoria RAM
TEMP_L:      DS.B   1        ; Para la transferencia de datos entre el registro
TEMP_T:      DS.B   1        ; X y el registro H.
;*******************************************************************
DIG3_PP1:    DS.B   1        ; Declaración de registros en memoria RAM
DIG2_PP1:    DS.B   1        ; para el almacenamiento de los valores de las
DIG1_PP1:    DS.B   1        ; mediciones ya convertidas en decimal
DIG3_PP3:    DS.B   1        ; DIG_3 es el digito mas significativo
DIG2_PP3:    DS.B   1
DIG1_PP3:    DS.B   1
DIG3_PP4:    DS.B   1
DIG2_PP4:    DS.B   1
DIG1_PP4:    DS.B   1
DIG3_PP5:    DS.B   1
DIG2_PP5:    DS.B   1
DIG1_PP5:    DS.B   1
DIG3_PP7:    DS.B   1
DIG2_PP7:    DS.B   1
DIG1_PP7:    DS.B   1
DIG3_PP8:    DS.B   1
DIG2_PP8:    DS.B   1
DIG1_PP8:    DS.B   1
DIG3_TP1:    DS.B   1
DIG2_TP1:    DS.B   1
DIG1_TP1:    DS.B   1
DIG3_TP3:    DS.B   1
DIG2_TP3:    DS.B   1
DIG1_TP3:    DS.B   1
DIG3_TP4:    DS.B   1
DIG2_TP4:    DS.B   1
DIG1_TP4:    DS.B   1
;*******************************************************************
DAT_DIS:     DS.B   1        ; Declaración de registros para almacenamiento de
CONT_A:      DS.B   1        ; un identificador o bandera y una cuenta
;*******************************************************************
; code section
;
            ORG    ROMStart
; Creación de tablas con valores representativos del codigo ASCII para mensajes a
; desplegar en el LCD
MEN_PRE:    FCB $50,$52,$45,$53,$49,$4F,$4E,$10,$45,$4E,$10,$6D,$6D,$48,$67,$10
                        ;P  R  E  S  I  O  N  _  E  N  _  m  m  H  g  _

MEN_TEM:    FCB $10,$54,$45,$4D,$50,$45,$52,$41,$54,$55,$52,$41,$10,$DF,$43,$10
                        ;_  T  E  M  P  E  R  A  T  U  R  A  _  o  C  _
```

```
; Include device initialization code
        INCLUDE 'MCUInit.inc'
_Startup:
        LDHX   #RAMEnd+1       ; initialize the stack pointer
        TXS
        ; Call generated Device Initialization function
        JSR    MCU_init
;*******************************************************************
mainLoop:   JSR RET_1BIT
        ; Insert your code here
;*******************************************************************
;
            NOP

            CLR     TEMP_H
                CLR     TEMP_L
            CLR     TEMP_T

             BRCLR   4,PTA,ET_SET      ;secuencia de comandos que prenden
             BCLR    4,PTA             ;y apagan el LED conectado en PTA 4
             BRA     ET_SE             ; por cada transmisión
ET_SET:      BSET    4,PTD
;*******************************************************************
;
; Direccionamiento de subrutinas
;*******************************************************************
;
ET_SE:      JSR     RET_INI             ; retardo inicial para la inicialización del LCD
            JSR     INI_DISP            ; inicialización del LCD
            JSR     TOMA_LEC            ; Toma de lecturas del ADC
            JSR     CONV_P1             ; Algoritmos para las conversiones
                JSR     CONV_P3
            JSR     CONV_P4
            JSR     CONV_P7
            JSR     CONV_P8
            JSR     CONV_P5
            JSR     CONV_T1
            JSR     CONV_T3
            JSR     CONV_T4

            BRSET   5,PTA,IM_P        ;En este puerto hay un swich conectado por lo
            JSR     IMP_TEM           ; según su estado en el LCD se mostraran
            BRA     ET_CON            ; temperaturas o presiones
IM_P:       JSR     IMP_PRE
ET_CON:
             JSR     TRANSMISION       ; subrutina para la transmisión serial

             JMP     mainLoop          ; brinca al inicio
```

SUBRUTINAS DEL SISTEMA DE MONITOREO

```
;*******************************************************************
;
; SUBRRUTINAS
;*******************************************************************
;
; TOMA DE LECTURAS
;*********************************************************
;
TOMA_LEC:
                MOV     #$50,ADCLK
                MOV     #$20,ADCSC      ;INICIALIZA EL CONVERTIDOR ANALOGICO DIGITAL CANAL 00 PP1 8BITS
                BRCLR   7,ADCSC,*
                LDA     ADRL
                BRCLR   7,ADCSC,*
                LDA     ADRL
                STA     PRES_ADC_PP1

                MOV     #$50,ADCLK
                MOV     #$21,ADCSC      ;INICIALIZA EL CONVERTIDOR ANALOGICO DIGITAL CANAL 01 PP3 8BITS
                BRCLR   7,ADCSC,*
                LDA     ADRL
                BRCLR   7,ADCSC,*
                LDA     ADRL
                STA     PRES_ADC_PP3

                MOV     #$50,ADCLK
```

```
            MOV    #$22,ADCSC      ;INICIALIZA EL CONVERTIDOR ANALOGICO DIGITAL CANAL 02 PP4 8BITS
            BRCLR  7,ADCSC,*
            LDA    ADRL
            BRCLR  7,ADCSC,*
            LDA    ADRL
            STA    PRES_ADC_PP4

            MOV    #$50,ADCLK
            MOV    #$23,ADCSC      ;INICIALIZA EL CONVERTIDOR ANALOGICO DIGITAL CANAL 03 PP7 8BITS
            BRCLR  7,ADCSC,*
            LDA    ADRL
            BRCLR  7,ADCSC,*
            LDA    ADRL
            STA    PRES_ADC_PP7

            MOV    #$50,ADCLK
            MOV    #$24,ADCSC      ;INICIALIZA EL CONVERTIDOR ANALOGICO DIGITAL CANAL 04 PP8 8BITS
            BRCLR  7,ADCSC,*
            LDA    ADRL
            BRCLR  7,ADCSC,*
            LDA    ADRL
            STA    PRES_ADC_PP8

            MOV    #$54,ADCLK
            MOV    #$2B,ADCSC      ;INICIALIZA EL CONVERTIDOR ANALOGICO DIGITAL CANAL 05 PP5 10BITS
            BRCLR  7,ADCSC,*
            LDA    ADRL
            BRCLR  7,ADCSC,*
            LDHX   ADRH
            STHX   PRES_ADC_PP5H

            MOV    #$54,ADCLK
            MOV    #$25,ADCSC      ;INICIALIZA EL CONVERTIDOR ANALOGICO DIGITAL CANAL 06 TP1 10BITS
            BRCLR  7,ADCSC,*
            LDA    ADRL
            BRCLR  7,ADCSC,*
            LDHX   ADRH
            STHX   PRES_ADC_TP1H

            MOV    #$54,ADCLK
            MOV    #$26,ADCSC      ;INICIALIZA EL CONVERTIDOR ANALOGICO DIGITAL CANAL 07 TP3 10BITS
            BRCLR  7,ADCSC,*
            LDA    ADRL
            BRCLR  7,ADCSC,*
            LDHX   ADRH
            STHX   PRES_ADC_TP3H

            MOV    #$54,ADCLK
            MOV    #$27,ADCSC      ;INICIALIZA EL CONVERTIDOR ANALOGICO DIGITAL CANAL 11 TP4 10BITS
            BRCLR  7,ADCSC,*
            LDA    ADRL
            BRCLR  7,ADCSC,*
            LDHX   ADRH
            STHX   PRES_ADC_TP4H

            RTS
;*************************************************************************
; PRESION EN P1 conversión
;*************************************************************************
CONV_P1:  LDA    PRES_ADC_PP1
          CMP    #$33
          BLS    IMP1_000
          CMP    #$FA
          BHS    IMP1_298

          SUB    #$33
          LDX    #$0F
          MUL

        STX    TEMP_H        ; TRANSFERIR X HACIA H
               LDHX   TEMP_H
               LDX    #$64
               DIV
```

```
            STA    TEMP_T
            STHX   TEMP_H

            CLRH
            LDA    TEMP_H
            LDX    #$0A
            DIV
            STA    DIG1_PP1

           CLRH
            LDA    TEMP_T
            DIV
            STA    DIG3_PP1
            STHX   TEMP_H
            MOV    TEMP_H,DIG2_PP1
            BRA    FIN_PP1

IMP1_000:   CLR    DIG3_PP1
            CLR    DIG2_PP1
            CLR    DIG1_PP1
            BRA    FIN_PP1

IMP1_298:   MOV    #$02,DIG3_PP1
            MOV    #$09,DIG2_PP1
            MOV    #$08,DIG1_PP1

FIN_PP1:    RTS
;*************************************************************
; PRECION EN P3 conversión
;*************************************************************
CONV_P3:
                       LDA    PRES_ADC_PP3
                       CMP    #$0A
                       BLS    IMP3_000
                       CMP    #$F0
                       BHS    IMP3_759

            SUB    #$0A
            LDX    #$21
            MUL

            STX    TEMP_H        ; TRANSFERIR X HACIA H
            LDHX   TEMP_H
            LDX    #$64
            DIV
            STA    TEMP_T
            STHX   TEMP_H

            CLRH
            LDA    TEMP_H
            LDX    #$0A
            DIV
            STA    DIG1_PP3

            CLRH
            LDA    TEMP_T
            DIV
            STA    DIG3_PP3
            STHX   TEMP_H
            MOV    TEMP_H,DIG2_PP3
            BRA    FIN_PP3

IMP3_000:   CLR    DIG3_PP3
            CLR    DIG2_PP3
            CLR    DIG1_PP3
            BRA    FIN_PP3

IMP3_759:   MOV    #$07,DIG3_PP3
            MOV    #$05,DIG2_PP3
            MOV    #$09,DIG1_PP3

FIN_PP3:    RTS
;*************************************************************
```

```
; PRECION EN P4 conversión
;*******************************************************
;
CONV_P4:
            LDA    PRES_ADC_PP4
            CMP    #$33
            BLS    IMP4_000
            CMP    #$FA
            BHS    IMP4_298

            SUB    #$33
            LDX    #$0F
            MUL

            STX    TEMP_H        ; TRANSFERIR X HACIA H
            LDHX   TEMP_H
            LDX    #$64
            DIV
            STA    TEMP_T
            STHX   TEMP_H

            CLRH
            LDA    TEMP_H
            LDX    #$0A
            DIV
            STA    DIG1_PP4

            CLRH
            LDA    TEMP_T
            DIV
            STA    DIG3_PP4
            STHX   TEMP_H
            MOV    TEMP_H,DIG2_PP4
                     BRA    FIN_PP4

IMP4_000:   CLR    DIG3_PP4
            CLR    DIG2_PP4
            CLR    DIG1_PP4
            BRA    FIN_PP4

IMP4_298:   MOV    #$02,DIG3_PP4
            MOV    #$09,DIG2_PP4
            MOV    #$08,DIG1_PP4

FIN_PP4     RTS
;*******************************************************
;
; PRECION EN P7 conversión
;*******************************************************
;
CONV_P7:
            LDA    PRES_ADC_PP7
            CMP    #$33
            BLS    IMP7_000
            CMP    #$FA
            BHS    IMP7_298

            SUB    #$33
            LDX    #$0F
            MUL

            STX    TEMP_H        ; TRANSFERIR X HACIA H
            LDHX   TEMP_H
            LDX    #$64
            DIV
            STA    TEMP_T
            STHX   TEMP_H

            CLRH
            LDA    TEMP_H
            LDX    #$0A
            DIV
            STA    DIG1_PP7

            CLRH
            LDA    TEMP_T
```

```
            DIV
            STA    DIG3_PP7
            STHX   TEMP_H
            MOV    TEMP_H,DIG2_PP7
            BRA    FIN_PP7

IMP7_000:   CLR    DIG3_PP7
            CLR    DIG2_PP7
            CLR    DIG1_PP7
            BRA    FIN_PP7

IMP7_298:   MOV    #$02,DIG3_PP7
            MOV    #$09,DIG2_PP7
            MOV    #$08,DIG1_PP7

FIN_PP7:    RTS
;*************************************************************
; PRECION EN P8 conversión
;*************************************************************
CONV_P8:
            LDA    PRES_ADC_PP8
            CMP    #$33
            BLS    IMP8_000
            CMP    #$FA
            BHS    IMP8_298

            SUB    #$33
            LDX    #$0F
            MUL

            STX    TEMP_H        ; TRANSFERIR X HACIA H
            LDHX   TEMP_H
            LDX    #$64
            DIV
            STA    TEMP_T
            STHX   TEMP_H

            CLRH
            LDA    TEMP_H
            LDX    #$0A
            DIV
            STA    DIG1_PP8

            CLRH
            LDA    TEMP_T
            DIV
            STA    DIG3_PP8
            STHX   TEMP_H
            MOV    TEMP_H,DIG2_PP8
            BRA    FIN_PP8

IMP8_000:   CLR    DIG3_PP8
            CLR    DIG2_PP8
            CLR    DIG1_PP8
            BRA    FIN_PP8

IMP8_298:   MOV    #$02,DIG3_PP8
            MOV    #$09,DIG2_PP8
            MOV    #$08,DIG1_PP8

FIN_PP8:    RTS
;*************************************************************************
; PRESION EN P5 conversión 10 bits
;*************************************************************************
CONV_P5:    LDA    PRES_ADC_PP5H
            BNE    EP1_01
            LDA    PRES_ADC_PP5L
            CMP    #$29
            BLS    IMP5_000
            BRA    INI_CP5

EP1_01:     LDA    PRES_ADC_PP5H
            CMP    #$03
```

```
            BHI    IMP5_750
            BEQ    EP1_02
            BRA    INI_CP5


EP1_02:      LDA    PRES_ADC_PP5L
            CMP    #$C3
            BHS    IMP5_750



INI_CP5:    LDHX   PRES_ADC_PP5H
            AIX    #$D7
            TXA
            LDX    #$7B
            DIV
            STA    DIG3_PP5

            STHX   TEMP_H
            LDA    TEMP_H
            LDX    #$64
            MUL
            STX    TEMP_H
            LDHX   TEMP_H
            LDX    #$7B
            DIV
            CLRH
            LDX    #$0A
            DIV

            STA    DIG2_PP5
            STHX   TEMP_H
            MOV    TEMP_H,DIG1_PP5
            BRA    FIN_PP5

IMP5_000:   CLR    DIG3_PP5
            CLR    DIG2_PP5
            CLR    DIG1_PP5
            BRA    FIN_PP5

IMP5_750:   MOV    #$07,DIG3_PP5
            MOV    #$05,DIG2_PP5
            MOV    #$00,DIG1_PP5

FIN_PP5:    RTS
;*************************************************************
; TEMPERATURA EN T1  conversión  10 bits
;*************************************************************
CONV_T1:    LDA    PRES_ADC_TP1H
            BNE    ET1_01
            LDA    PRES_ADC_TP1L
            CMP    #$14
            BLS    IMT1_40
            BRA    INI_CT1

ET1_01:      LDA    PRES_ADC_TP1H
            CMP    #$01
            BHI    IMT1_125

            LDA    PRES_ADC_TP1L
            CMP    #$66
            BHS    IMT1_125


INI_CT1:    LDHX   PRES_ADC_TP1H
            AIX    #$EC
            TXA
            LDX    #$02
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,TEMP_T
```

```
            LDX    #$F4
            MUL
            STX    TEMP_H
            STA    TEMP_L
            LDHX   TEMP_H
            LDA    TEMP_T

            BEQ    ET1_02
            AIX    #$7A
ET1_02:     TXA
            LDX    #$FA
            DIV
            SUB    #$28
            BMI    NEG_T1
            CLRH
            LDX    #$0A
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,DIG1_TP1

            CLRH
            LDX    #$0A
            DIV

            STA    DIG3_TP1
            STHX   TEMP_H
            MOV    TEMP_H,DIG2_TP1
            BRA    FIN_TP1

NEG_T1:     MOV    #$FD,DIG3_TP1
            NEGA
            CLRH
            LDX    #$0A
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,DIG1_TP1
            CLRH
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,DIG2_TP1
            BRA    FIN_TP1

IMT1_40:    MOV    #$FD,DIG3_TP1        ;2D=AL SIGNO MENOS EN ASCII 2D-30= FD
            MOV    #$04,DIG2_TP1
            CLR    DIG1_TP1
            BRA    FIN_TP1

IMT1_125:   MOV    #$01,DIG3_TP1
            MOV    #$02,DIG2_TP1
            MOV    #$05,DIG1_TP1

FIN_TP1:    RTS
;*********************************************************
; TEMPERATURA EN T3  10 bits
;*********************************************************
CONV_T3:    LDA    PRES_ADC_TP3H
            BNE    ET3_01
            LDA    PRES_ADC_TP3L
            CMP    #$14
            BLS    IMT3_40
            BRA    INI_CT3

ET3_01:     LDA    PRES_ADC_TP3H
            CMP    #$01
            BHI    IMT3_125

            LDA    PRES_ADC_TP3L
            CMP    #$66
            BHS    IMT3_125

INI_CT3:    LDHX   PRES_ADC_TP3H
            AIX    #$EC
            TXA
```

```
            LDX    #$02
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,TEMP_T
            LDX    #$F4
            MUL
            STX    TEMP_H
            STA    TEMP_L
            LDHX   TEMP_H
            LDA    TEMP_T

            BEQ    ET3_02
            AIX    #$7A
ET3_02:     TXA
            LDX    #$FA
            DIV
            SUB    #$28
            BMI    NEG_T3
            CLRH
            LDX    #$0A
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,DIG1_TP3

            CLRH
            LDX    #$0A
            DIV

            STA    DIG3_TP3
            STHX   TEMP_H
            MOV    TEMP_H,DIG2_TP3
            BRA    FIN_TP3

NEG_T3:     MOV    #$FD,DIG3_TP3
            NEGA
            CLRH
            LDX    #$0A
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,DIG1_TP3
            CLRH
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,DIG2_TP3
            BRA    FIN_TP3

IMT3_40:    MOV    #$FD,DIG3_TP3        ;2D=AL SIGNO MENOS EN ASCII 2D-30= FD
            MOV    #$04,DIG2_TP3
            CLR    DIG1_TP3
            BRA    FIN_TP3

IMT3_125:   MOV    #$01,DIG3_TP3
            MOV    #$02,DIG2_TP3
            MOV    #$05,DIG1_TP3

FIN_TP3:    RTS
;*********************************************************
; TEMPERATURA EN T4  10 bits
;*********************************************************
 CONV_T4:  LDA    PRES_ADC_TP4H
            BNE    ET4_01
            LDA    PRES_ADC_TP4L
            CMP    #$14
            BLS    IMT4_40
            BRA    INI_CT4

ET4_01:     LDA    PRES_ADC_TP4H
            CMP    #$01
            BHI    IMT4_125

            LDA    PRES_ADC_TP4L
            CMP    #$66
            BHS    IMT4_125
```

```
INI_CT4:    LDHX   PRES_ADC_TP4H
            AIX    #$EC
            TXA
            LDX    #$02
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,TEMP_T
            LDX    #$F4
            MUL
            STX    TEMP_H
            STA    TEMP_L
            LDHX   TEMP_H
            LDA    TEMP_T

            BEQ    ET4_02
            AIX    #$7A
ET4_02:     TXA
            LDX    #$FA
            DIV
            SUB    #$28
            BMI    NEG_T4
            CLRH
            LDX    #$0A
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,DIG1_TP4

            CLRH
            LDX    #$0A
            DIV

            STA    DIG3_TP4
            STHX   TEMP_H
            MOV    TEMP_H,DIG2_TP4
            BRA    FIN_TP4

NEG_T4:     MOV    #$FD,DIG3_TP4
            NEGA
            CLRH
            LDX    #$0A
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,DIG1_TP4
            CLRH
            DIV
            STHX   TEMP_H
            MOV    TEMP_H,DIG2_TP4
            BRA    FIN_TP4

IMT4_40:    MOV    #$FD,DIG3_TP4        ;2D=AL SIGNO MENOS EN ASCII 2D-30= FD
            MOV    #$04,DIG2_TP4
            CLR    DIG1_TP4
            BRA    FIN_TP4

IMT4_125:   MOV    #$01,DIG3_TP4
            MOV    #$02,DIG2_TP4
            MOV    #$05,DIG1_TP4

FIN_TP4:    RTS
;*************************************************************
; INICIALIZACION DEL DISPLAY
; DIRECCIONES DD RAM
; 1ra CARACTER LINEA 1  00 + 80 = 80
; 2da CARACTER LINEA 2  40 + 80 = C0
; 3ra CARACTER LINEA 3  10 + 80 = 90
; 4ta CARACTER LINEA 4  50 + 80 = D0
;*************************************************************
 INI_DISP:
            LDA #$01        ; BORRA EL LCD
            JSR instruccion1
```

```
            LDA #$38        ; CONFIGURA MODO DEL LCD(FUENTE 5X7,2 LINEAS,8BITS)
            JSR instruccion

            LDA #$0F        ;CONFIGURA LCD (LCD ON/OFF,CURSOR ON/OFF,PARPADEO ON/OF)
            JSR instruccion

            LDA #$06        ;CONFIGURA CURSOR(DEZPLAZAMIENTO TEXTO/CURSOR,
            JSR instruccion  ;DEZPLAZAMIENTO CURSOR DER/IZQZ)

            RTS
;*********************************************************************
; INSTRUCCION CON RETARDO DE 46 MICROSEGUNDOS PARA BUS DE 8 MHz
; AL LCD SE LE MANDARÁ UNA INSTRUCCION
;*********************************************************************
instruccion:   STA DAT_DIS
            BCLR 1,PTA    ; RS=0,  LE INDICA AL LCD QUE ENVIARA UNA INSTRUCCION
                          ; R/W=0, LE INDICA AL LCD QUE ESCRIBIRA
            BSET 2,PTA    ; E=1,   HABILITA AL LCD

            BRSET 0,DAT_DIS,ETD1
            BCLR  0,PTA
            BRA   CON1
ETD1:         BSET  0,PTA

CON1:         STA  PTD          ; MANDA EL CONTENIDO DE A AL PUERTO D
            BCLR 2,PTA        ; DESHABILITA AL LCD
            JSR  RETARDO
            RTS               ; RETORNA AL PROGRAMA PRINCIPAL
;*********************************************************************
; SUBRRUTINA INSTRUCCION1 CON RETARDO DE 1.64 MILISEGUNDOS PARA ; ;
; BUS DE 8 MHz
;*********************************************************************
instruccion1:  STA DAT_DIS
            BCLR 1,PTA        ;RS=0,  LE INDICA AL LCD QUE ENVIARA UNA INSTRUCCION
                              ;R/W=0, LE INDICA AL LCD QUE ESCRIBIRA(SIEMPRE=0)
            BSET 2,PTA        ;E=1,   HABILITA AL LCD

            BRSET 0,DAT_DIS,ETD2
            BCLR  0,PTA
            BRA   CON2
ETD2:               BSET  0,PTA

CON2:         STA   PTD
            BCLR 2,PTA          ;DESHABILITA AL LCD
            BSR  RETARDO1
            RTS                 ;RETORNA AL PROGRAMA PRINCIPAL
;*********************************************************************
; SUBRRUTINA QUE MANDA DATOS AL LCD
;*****************************************************************
DATO:         STA DAT_DIS
            BSET 1,PTA            ;RS=1,  LE INDICA AL LCD QUE ENVIARA UN DATO
                                        ;R/W=0, LE INDICA AL LCD QUE ESCRIBIRA UN DATO
            BSET 2,PTA            ;E=1,   HABILITA AL LCD

            BRSET 0,DAT_DIS,ETD3
            BCLR  0,PTA
            BRA   CON3
ETD3:         BSET  0,PTA

CON3:         STA  PTD
            BCLR 2,PTA            ; E=0,    DESHABILITA EL LCD
            JSR  RETARDO
            RTS
;*****************************************************************
; RETARDO INICIAL 15ms         BUS 8MHz
;*****************************************************************
RET_INI:
            CLRX
LOOOP:      CLRA
LOOP:         CBEQA #$FF,LAP
            INCA
            BRA   LOOP
LAP:          CBEQX #$3B,FLOOP
```

```
            INCX
            BRA   LOOOP
FLOOP:      RTS
;*******************************************************************
;SUBRRUTINA RETARDO  1.64 MILISEGUNDOS    BUS 8MHz
;*******************************************************************
;
RETARDO1:  CLRX
LOOOP1:     CLRA
LOOP1:      CBEQA #$FF,LAP1
            INCA
            BRA   LOOP1
LAP1:       CBEQX #$06,FLOOP1
            INCX
            BRA   LOOOP1
FLOOP1:     RTS
;*******************************************************************
;RETARDO DE 46 MICROSEGUNDOS         BUS 8MHz
;*******************************************************************
;
RETARDO:   CLRA
LOOP3:      CBEQA #$30,LAP3
            INCA
            BRA   LOOP3
LAP3:       RTS
;*******************************************************************
;RETARDO 1BIT, 2400 BAUDIOS 416.667 MICRO SEG CON BUS DE 8 MHz
;*******************************************************************
;
RET_1BIT:
            LDA   #$ED
LOOP4:      BRN   *
            BRN   *
            BRN   *
            NOP
            NOP

            DBNZA LOOP4
            RTS
;*******************************************************************
; SUBRRUTINA PARA LA TRANSMISION SERIAL CON PROTOCOLO INVERTIDO
; Y SIN RETORNO A CERO
;*******************************************************************
;
TRANSMISION:
            CLRH
            CLRX
BY_SIG:
            MOV   #$08,CONT_A   ;NUMERO DE BITS A TRANSMITIR
            BSET  3,PTA         ;PTA2=1 BIT DE INICIO (INVERTIDO SEGUN EL PROTOCOLO TRADICIONAL)
            BSR   RET_1BIT      ; RETARDO 416.667 MICROSEGUNDOS 2400 BAUDIOS

BIT_SIG:    ROR   DIG3_PP1,X    ;ROTAR A LA DERECHA, EL BIT MENOS SIGNIFICATIVO
            BCC   ET_08         ; BIFURCA SI EL BIT CARRY ES CERO
            BSET  3,PTA         ;PTA2=1; SE TRANSMITE UN UNO
            BRA   TRANS_0

ET_08:      BCLR  3,PTA         ;QUEDA EN LA BANDERA DE CARRY

TRANS_0:    BSR   RET_1BIT      ; RETARDO 416.667 MICROSEGUNDOS 2400 BAUDIOS
                                ; PTA2 PERMANECE EN CERO Y SE TRANSMITE
            DEC   CONT_A
            BNE   BIT_SIG       ; BIFURCA SI AUN QUEDAN BITS POR TRANSMITIR
            BCLR  3,PTA
            BSR   RET_1BIT

            ROR   DIG3_PP1,X
            INCX
            CBEQX #$1B,FIN_TX
            BRA   BY_SIG
FIN_TX:     RTS
;*******************************************************************
; SUBRRUTINA ENCARGADA DEL DESPLIEGUE EN EL LCD DE LAS PRESIONES
;*******************************************************************
;
IMP_PRE:    LDA #$01            ; BORRA EL LCD
            JSR instruccion1
```

```
            LDA #$02                    ; return to home instrucción
            JSR instruccion1

            CLRX
            CLRH
AT_01:        CBEQX #$10,CONT_1
            LDA   MEN_PRE,X
            JSR   DATO
            INCX
            BRA  AT_01
;*************************************
CONT_1:      LDA  #$C0
            JSR  instruccion1

            LDA  #$50   ;P
            JSR  DATO
            LDA  #$31   ;1
            JSR  DATO
            LDA  #$3A   ;:
            JSR  DATO
            LDA  DIG3_PP1
            ADD  #$30
            JSR  DATO
            LDA  DIG2_PP1
            ADD  #$30
            JSR  DATO
            LDA  #$2E   ;.
            JSR  DATO
            LDA  DIG1_PP1
            ADD  #$30
            JSR  DATO
            LDA  ' '
            JSR  DATO
            LDA  ' '
            JSR  DATO

            LDA  #$50   ;P
            JSR  DATO
            LDA  #$33   ;3
            JSR  DATO
            LDA  #$3A   ;:
            JSR  DATO
            LDA  DIG3_PP3
            ADD  #$30
            JSR  DATO
            LDA  DIG2_PP3
            ADD  #$30
            JSR  DATO
            LDA  #$2E   ;.
            JSR  DATO
            LDA  DIG1_PP3
            ADD  #$30
            JSR  DATO
;*********************************
            LDA  #$90
            JSR  instruccion1

            LDA  #$50   ;P
            JSR  DATO
            LDA  #$34   ;4
            JSR  DATO
            LDA  #$3A   ;:
            JSR  DATO
            LDA  DIG3_PP4
            ADD  #$30
            JSR  DATO
            LDA  DIG2_PP4
            ADD  #$30
            JSR  DATO
            LDA  #$2E   ;.
            JSR  DATO
            LDA  DIG1_PP4
            ADD  #$30
```

```
            JSR  DATO
            LDA ' '
            JSR  DATO
            LDA ' '
            JSR  DATO

            LDA  #$50  ;P
            JSR  DATO
            LDA  #$35   ;5
            JSR  DATO
            LDA  #$3A   ;:
            JSR  DATO
            LDA  DIG3_PP5
            ADD  #$30
            JSR  DATO
            LDA  DIG2_PP5
            ADD  #$30
            JSR  DATO
            LDA  DIG1_PP5
            ADD  #$30
            JSR  DATO
;************************************
            LDA  #$D0
            JSR  instruccion1

            LDA  #$50   ;P
            JSR  DATO
            LDA  #$37   ;7
            JSR  DATO
            LDA  #$3A   ;:
            JSR  DATO
            LDA  DIG3_PP7
            ADD  #$30
            JSR  DATO
            LDA  DIG2_PP7
            ADD  #$30
            JSR  DATO
            LDA  #$2E   ;.
            JSR  DATO
            LDA  DIG1_PP7
            ADD  #$30
            JSR  DATO
            LDA ' '
            JSR  DATO
            LDA ' '
            JSR  DATO

            LDA  #$50  ;P
            JSR  DATO
            LDA  #$38   ;8
            JSR  DATO
            LDA  #$3A   ;:
            JSR  DATO
            LDA  DIG3_PP8
            ADD  #$30
            JSR  DATO
            LDA  DIG2_PP8
            ADD  #$30
            JSR  DATO
            LDA  #$2E   ;.
            JSR  DATO
            LDA  DIG1_PP8
            ADD  #$30
            JSR  DATO

            RTS
;*****************************************************************************
; SUBRRUTINA QUE VISULIZA LAS TEMPERATURAS EN EL LCD
;*****************************************************************************
IMP_TEM:    LDA #$01              ; BORRA EL LCD
            JSR instruccion1

            LDA #$02              ;return to home instrucción
```

```
            JSR instruccion1

            CLRX
            CLRH
AT_02:        CBEQX #$10,CONT_2
            LDA  MEN_TEM,X
           JSR  DATO
            INCX
            BRA  AT_02
;**********************************
CONT_2:      LDA  #$C0
            JSR  instruccion1

            LDA  #$54  ;T
            JSR  DATO
            LDA  #$31   ;1
            JSR  DATO
            LDA  #$3A   ;:
            JSR  DATO
            LDA  DIG3_TP1
            ADD  #$30
            JSR  DATO
            LDA  DIG2_TP1
            ADD  #$30
            JSR  DATO
            LDA  DIG1_TP1
            ADD  #$30
            JSR  DATO
;**********************************
            LDA  #$90
            JSR  instruccion1

            LDA  #$54  ;T
            JSR  DATO
            LDA  #$33   ;3
            JSR  DATO
            LDA  #$3A   ;:
            JSR  DATO
            LDA  DIG3_TP3
            ADD  #$30
            JSR  DATO
            LDA  DIG2_TP3
            ADD  #$30
            JSR  DATO
            LDA  DIG1_TP3
            ADD  #$30
            JSR  DATO

            LDA  #$D0
            JSR  instruccion1
;**********************************
            LDA  #$54  ;T
            JSR  DATO
            LDA  #$34   ;4
            JSR  DATO
            LDA  #$3A   ;:
            JSR  DATO
            LDA  DIG3_TP4
            ADD  #$30
            JSR  DATO
            LDA  DIG2_TP4
            ADD  #$30
            JSR  DATO
            LDA  DIG1_TP4
            ADD  #$30
            JSR  DATO

            RTS
;**********************************
```

PROGRAMA PRINCIPAL DEL SISTEMA RECEPTOR

```
;
; variable/data section
;
        ORG    Z_RAMStart        ; Insert your data definition here
REG_CONT:  DS.B   1
DATA_TX:   DS.B   1
ZETA:      DS.B   1
DIG3_PP1:    DS.B   1
DIG2_PP1:    DS.B   1
DIG1_PP1:    DS.B   1
DIG3_PP3:    DS.B   1
DIG2_PP3:    DS.B   1
DIG1_PP3:    DS.B   1
DIG3_PP4:    DS.B   1
DIG2_PP4:    DS.B   1
DIG1_PP4:    DS.B   1
DIG3_PP5:    DS.B   1
DIG2_PP5:    DS.B   1
DIG1_PP5:    DS.B   1
DIG3_PP7:    DS.B   1
DIG2_PP7:    DS.B   1
DIG1_PP7:    DS.B   1
DIG3_PP8:    DS.B   1
DIG2_PP8:    DS.B   1
DIG1_PP8:    DS.B   1
DIG3_TP1:    DS.B   1
DIG2_TP1:    DS.B   1
DIG1_TP1:    DS.B   1
DIG3_TP3:    DS.B   1
DIG2_TP3:    DS.B   1
DIG1_TP3:    DS.B   1
DIG3_TP4:    DS.B   1
DIG2_TP4:    DS.B   1
DIG1_TP4:    DS.B   1
CONT_INT:    DS.B   1
;
; code section
;
        ORG    ROMStart
; Include device initialization code
        INCLUDE 'MCUInit.inc'

_Startup:
        LDHX   #RAMEnd+1       ; initialize the stack pointer
        TXS
        ; Call generated Device Initialization function
        JSR    MCU_init

mainLoop:
        ; Insert your code here
        CLR CONT_INT
        MOV #$2A,ZETA     ;CARGAR EL VALOR EN ASCII DE LA LETRA Z - 30H = 2AH
;*****************************************
;SERIE DE RETARDOS IGUAL A 2 SEGUNDOS
;*****************************************
            CLR   REG_CONT
ET_01:                LDA   REG_CONT
            CBEQA #$1F,ET_00
            INC   REG_CONT
            JSR   RETARDO
            BRA   ET_01
;*****************************************
ET_00:                CLRH
            CLRX
LOOP:                 CLR DIG3_PP1,X     ;RUTINA DE
            INCX                    ;INICIALIZACION A CERO DE LOS REGISTROS
            CBEQX #$1B,FIN1
            BRA LOOP
FIN1:
            CLRH
;*************************************
```

```
ESPE:     CLRX                  ;RUTINA SINPLE QUE BUSCA UN TIEMPO DE ESPERA
SI_ES:    JSR  RET_1BIT         ;ENTRE LOS BLOQUES DE TRANSMISION PARA ALINEAR EN
          BRSET 0,PTAD,ESPE     ;CUALQUIER MOMENTO LA TRANSMISION CON LA RESEPCION.
                                ;AL ENCONTRAR UN TIEMPO DE ESPERA SE HABILITAN LAS
          CBEQX #$0F,FIN        ;INTERRUPCIONES POR TECLADO.
          INCX
          BRA  SI_ES
;*************************************
FIN:      CLRX
          BSET  2,KBISC         ;SE APAGA LA BANDERA DE INTERRUPCION GENERADA POR POSIBLE
                                ;DESALINEADO CON EL RECEPTOR O TRANSMISIONES
                                ;RECIBIDAS.
          BSET 1,KBISC          ;HABILITA LAS INTERRUPCIONES POR TECLADO

ESP_INT:          BRSET 0,CONT_INT, REPIT   ;EN ESPERA POR UN BIT DE ARRANQUE CON IDENTIFICADOR
          BRA  ESP_INT          ;DE INTERRUPCION EJECUTADA

REPIT:            BCLR  0,CONT_INT        ;LIMPIA EL IDENTIFICADOR
          BRA  ESPE             ;RETORNO AL TIEMPO DE ESPERA
;*****************************************
```

SUBRUTINAS DEL SISTEMA RECEPTOR

```
;*********************************************************************
;SUBRRUTINA DE TRANSMISION A LA COMPU A 4800 BAUDIOS
;*********************************************************************
 TX:      STA  DATA_TX
          MOV  #$08,REG_CONT
          BCLR  1,PTAD          ;BIT DE INICIO
          BSR  RET_1BITd

NEXTBIT:   ROR  DATA_TX
          BCC  SALTO
          BSET  1,PTAD
          BRA  SALTO2
SALTO:      BCLR  1,PTAD

SALTO2:     BSR  RET_1BITd      ;bit de datos
          DEC  REG_CONT
          BNE  NEXTBIT

          BSET  1,PTAD          ;bit de parada
          BSR  RET_1BITd
          BSR  RET_1BITd
          RTS
;*************************************************************************
;RETARDOS DE UN BIT Y MEDIO BIT A 2400 BAUDIOS
;1BIT=416.667 MICROSEGUNDOS, MEDIO BITS= 208.333 MICROSEGUNDOS
;*************************************************************************
RET_M_BIT:  LDA   #$7F
           BRA   ET_02

RET_1BIT:          NOP
           LDA   #$FF
ET_02:             BRN   *
           BRN   *

           NOP
           NOP
           NOP
           DBNZA  ET_02
        RTS
;*************************************************************************
;RETARDO DE UN BIT A 4800 BAUDIOS      ;208.333 microsegundos  PARA BUS DE  8 MHz
;*************************************************************************
 RET_1BITd:  lda #$B8
 ET_F02d:    nop
          nop
          nop
          nop
          DBNZA  ET_F02d
```

```
            RTS
;*******************************************************************************
RETARDO:    CLRX
LLOP2:              CLRA
LLOP:               CBEQA #$FF,SIG
            INCA
            BRA LLOP

SIG:        CBEQX #$FF,FINAL
            INCX
            BRA   LLOP2
FINAL:              RTS
;*******************************************************************************
```

INTERUPCION DEL SISTEMA RECEPTOR

```
;** ===================================================================
;**     Interrupt handler : isrVkeyboard1
;**
;**     Description :
;**         User interrupt service routine.
;**     Parameters  : None
;**     Returns     : Nothing
;** ===================================================================
      XDEF    isrVkeyboard1
isrVkeyboard1:
;*************************************************
;*************************************************
  ; Write your interrupt code here ...

        BCLR    1,KBISC
;---------------------------------------------------
;RECIBIR CODIGO DE 8 BITS   27 PALABRAS  A 2400 BAUDIOS
;---------------------------------------------------
             JSR   RET_M_BIT
BY_SIG:       JSR   RET_1BIT

            MOV   #$08, REG_CONT
BIT_SIG0:   CLC
            BRCLR 0,PTAD,ET_03
            SEC
ET_03:              ROR   DIG3_PP1,X
            JSR   RET_1BIT

            DEC   REG_CONT
            BNE   BIT_SIG0

            INCX
            CBEQX #$1B,FIN_RX
            JSR   RET_1BIT
            BRA   BY_SIG
;---------------------------------------------------
;TRANSMITIR A LA COMPUTADORA A 4800 BAUDIOS  28 PALABRAS
;---------------------------------------------------
FIN_RX:       CLRH
            CLRX
BYTE_SIG:   LDA   ZETA,X
            ADD   #$30          ;CONVIERTE LOS NUMEROS DECIMALES A CODIGO ASCII
            JSR   TX
            INCX
            CBEQX #$1C,FIN_TX
            BRA   BYTE_SIG

FIN_TX:       BRA    ESPE
;---------------------------------------------------
            BSET 0,CONT_INT       ;IDENTIFICADOR DE SUBRRUTINA CUMPLIDA
               RTI
; end of isrVkeyboard1
;*******************************************************************************
```

DATASHEETS

MAX232, MAX232I
DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047I – FEBRUARY 1989 – REVISED OCTOBER 2003

- Meet or Exceed TIA/EIA-232-F and ITU Recommendation V.28
- Operate With Single 5-V Power Supply
- Operate Up to 120 kbit/s
- Two Drivers and Two Receivers
- ±30-V Input Levels
- Low Supply Current . . . 8 mA Typical
- Designed to be Interchangeable With Maxim MAX232
- ESD Protection Exceeds JESD 22 – 2000-V Human-Body Model (A114-A)
- Applications
  - TIA/EIA-232-F
  - Battery-Powered Systems
  - Terminals
  - Modems
  - Computers

MAX232 . . . D, DW, N, OR NS PACKAGE
MAX232I . . . D, DW, OR N PACKAGE
(TOP VIEW)

description/ordering information

The MAX232 is a dual driver/receiver that includes a capacitive voltage generator to supply EIA-232 voltage levels from a single 5-V supply. Each receiver converts EIA-232 inputs to 5-V TTL/CMOS levels. These receivers have a typical threshold of 1.3 V and a typical hysteresis of 0.5 V, and can accept ±30-V inputs. Each driver converts TTL/CMOS input levels into EIA-232 levels. The driver, receiver, and voltage-generator functions are available as cells in the Texas Instruments LinASIC™ library.

ORDERING INFORMATION

T_A	PACKAGE†		ORDERABLE PART NUMBER	TOP-SIDE MARKING
0°C to 70°C	PDIP (N)	Tube	MAX232N	MAX232N
	SOIC (D)	Tube	MAX232D	MAX232
		Tape and reel	MAX232DR	
	SOIC (DW)	Tube	MAX232DW	MAX232
		Tape and reel	MAX232DWR	
	SOP (NS)	Tape and reel	MAX232NSR	MAX232
–40°C to 85°C	PDIP (N)	Tube	MAX232IN	MAX232IN
	SOIC (D)	Tube	MAX232ID	MAX232I
		Tape and reel	MAX232IDR	
	SOIC (DW)	Tube	MAX232IDW	MAX232I
		Tape and reel	MAX232IDWR	

† Package drawings, standard packing quantities, thermal data, symbolization, and PCB design guidelines are available at www.ti.com/sc/package.

Please be aware that an important notice concerning availability, standard warranty, and use in critical applications of Texas Instruments semiconductor products and disclaimers thereto appears at the end of this data sheet.

LinASIC is a trademark of Texas Instruments.

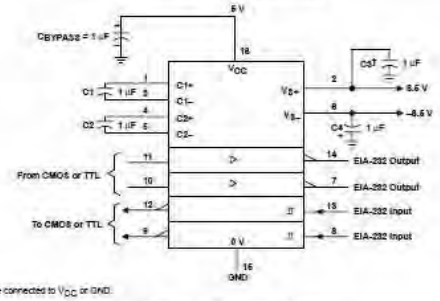
PRODUCTION DATA information is current as of publication date. Products conform to specifications per the terms of Texas Instruments standard warranty. Production processing does not necessarily include testing of all parameters.

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

MAX232, MAX232I
DUAL EIA-232 DRIVERS/RECEIVERS

SLLS047I – FEBRUARY 1989 – REVISED OCTOBER 2003

APPLICATION INFORMATION



† C3 can be connected to V_{CC} or GND.

Figure 4. Typical Operating Circuit

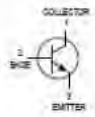
TEXAS INSTRUMENTS

MOTOROLA
SEMICONDUCTOR TECHNICAL DATA

Order this document by BC548/D

Amplifier Transistors
NPN Silicon

BC546, B
BC547, A, B, C
BC548, A, B, C



CASE 29–04, STYLE 17
TO–92 (TO–226AA)

MAXIMUM RATINGS

Rating	Symbol	BC 546	BC 547	BC 548	Unit
Collector–Emitter Voltage	V_{CEO}	65	45	30	Vdc
Collector–Base Voltage	V_{CBO}	80	50	30	Vdc
Emitter–Base Voltage	V_{EBO}		6.0		Vdc
Collector Current — Continuous	I_C		100		mAdc
Total Device Dissipation @ T_A = 25°C Derate above 25°C	P_D		625 5.0		mW mW/°C
Total Device Dissipation @ T_C = 25°C Derate above 25°C	P_D		1.5 12		Watt mW/°C
Operating and Storage Junction Temperature Range	T_J, T_{stg}		–55 to +150		°C

THERMAL CHARACTERISTICS

Characteristic	Symbol	Max	Unit
Thermal Resistance, Junction to Ambient	$R_{\theta JA}$	200	°C/W
Thermal Resistance, Junction to Case	$R_{\theta JC}$	83.3	°C/W

ELECTRICAL CHARACTERISTICS (T_A = 25°C unless otherwise noted)

Characteristic		Symbol	Min	Typ	Max	Unit
OFF CHARACTERISTICS						
Collector–Emitter Breakdown Voltage (I_C = 1.0 mA, I_B = 0)	BC546 BC547 BC548	$V_{(BR)CEO}$	65 45 30	—	—	V
Collector–Base Breakdown Voltage (I_C = 100 µAdc)	BC546 BC547 BC548	$V_{(BR)CBO}$	80 50 30	—	—	V
Emitter–Base Breakdown Voltage (I_E = 10 µA, I_C = 0)	BC546 BC547 BC548	$V_{(BR)EBO}$	6.0 6.0 6.0	—	—	V
Collector Cutoff Current (V_{CE} = 70 V, V_{BE} = 0) (V_{CE} = 50 V, V_{BE} = 0) (V_{CE} = 30 V, V_{BE} = 0) (V_{CE} = 30 V, T_A = 125°C)	BC546 BC547 BC548 BC546/547/548	I_{CES}	—	0.2 0.2 0.2	15 15 15 4.0	nA µA

REV Y

MOTOROLA

MOTOROLA
SEMICONDUCTOR TECHNICAL DATA

Order this document by BC556/D

Amplifier Transistors
PNP Silicon

BC556,B
BC557A,B,C
BC558B

CASE 29–04, STYLE 17
TO–92 (TO–226AA)

MAXIMUM RATINGS

Rating	Symbol	BC 556	BC 557	BC 558	Unit
Collector–Emitter Voltage	V_{CEO}	–65	–45	–30	Vdc
Collector–Base Voltage	V_{CBO}	–80	–50	–30	Vdc
Emitter–Base Voltage	V_{EBO}		–5.0		Vdc
Collector Current — Continuous	I_C		–100		mAdc
Total Device Dissipation @ T_A = 25°C Derate above 25°C	P_D		625 5.0		mW mW/°C
Total Device Dissipation @ T_C = 25°C Derate above 25°C	P_D		1.5 12		Watt mW/°C
Operating and Storage Junction Temperature Range	T_J, T_{stg}		–55 to +150		°C

THERMAL CHARACTERISTICS

Characteristic	Symbol	Max	Unit
Thermal Resistance, Junction to Ambient	$R_{\theta JA}$	200	°C/W
Thermal Resistance, Junction to Case	$R_{\theta JC}$	83.3	°C/W

ELECTRICAL CHARACTERISTICS (T_A = 25°C unless otherwise noted)

Characteristic		Symbol	Min	Typ	Max	Unit
OFF CHARACTERISTICS						
Collector–Emitter Breakdown Voltage (I_C = –2.0 mAdc, I_B = 0)	BC556 BC557 BC558	$V_{(BR)CEO}$	–65 –45 –30	—	—	V
Collector–Base Breakdown Voltage (I_C = –100 µAdc)	BC556 BC557 BC558	$V_{(BR)CBO}$	–80 –50 –30	—	—	V
Emitter–Base Breakdown Voltage (I_E = –100 µAdc, I_C = 0)	BC556 BC557 BC558	$V_{(BR)EBO}$	–5.0 –5.0 –5.0	—	—	V
Collector–Emitter Leakage Current (V_{CES} = –40 V) (V_{CES} = –25 V) (V_{CES} = –30 V, T_A = 125°C)	BC556 BC557 BC558 BC556 BC557 BC558	I_{CES}	—	–2.0 –2.0 –2.0	–100 –100 –100 –4.0 –4.0 –4.0	nA µA

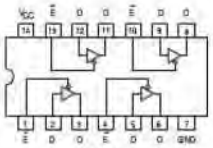
MOTOROLA

MOTOROLA

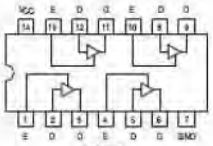
QUAD 3-STATE BUFFERS

SN54/74LS125A
SN54/74LS126A

QUAD 3-STATE BUFFERS
LOW POWER SCHOTTKY



LS125A



LS126A

J SUFFIX
CERAMIC
CASE 632-08

N SUFFIX
PLASTIC
CASE 646-06

D SUFFIX
SOIC
CASE 751A-02

ORDERING INFORMATION

SN54LSXXXJ Ceramic
SN74LSXXXN Plastic
SN74LSXXXD SOIC

TRUTH TABLES

LS125A

INPUTS		OUTPUT
E	D	
L	L	L
L	H	H
H	X	(Z)

LS126A

INPUTS		OUTPUT
E	D	
H	L	L
H	H	H
L	X	(Z)

L = LOW Voltage Level
H = HIGH Voltage Level
X = Don't Care
(Z) = High Impedance (off)

GUARANTEED OPERATING RANGES

Symbol	Parameter		Min	Typ	Max	Unit
VCC	Supply Voltage	54 74	4.5 4.75	5.0 5.0	5.5 5.25	V
TA	Operating Ambient Temperature Range	54 74	−55 0	25 25	125 70	°C
IOH	Output Current — High	54 74			−1.0 −2.6	mA
IOL	Output Current — Low	54 74			12 24	mA

FAST AND LS TTL DATA

MICROCHIP

MCP9700/9700A
MCP9701/9701A

Low-Power Linear Active Thermistor™ ICs

Features

- Tiny Analog Temperature Sensor
- Available Packages: SC-70-5, SOT-23-5, TO-92-3
- Wide Temperature Measurement Range:
  - -40°C to +125°C
- Accuracy:
  - ±2°C (max.), 0°C to +70°C (**MCP9700A/9701A**)
  - ±4°C (max.), 0°C to +70°C (**MCP9700/9701**)
- Optimized for Analog-to-Digital Converters (ADCs):
  - 10.0 mV/°C (typical) **MCP9700/9700A**
  - 19.5 mV/°C (typical) **MCP9701/9701A**
- Wide Operating Voltage Range:
  - V_{DD} = 2.3V to 5.5V **MCP9700/9700A**
  - V_{DD} = 3.1V to 5.5V **MCP9701/9701A**
- Low Operating Current: 6 µA (typical)
- Optimized to Drive Large Capacitive Loads

Typical Applications

- Hard Disk Drives and Other PC Peripherals
- Entertainment Systems
- Home Appliance
- Office Equipment
- Battery Packs and Portable Equipment
- General Purpose Temperature Monitoring

Description

The MCP9700/9700A and MCP9701/9701A family of Linear Active Thermistor™ Integrated Circuit (IC) is an analog temperature sensor that converts temperature to analog voltage. It's a low-cost, low-power sensor with an accuracy of ±2°C from 0°C to +70°C (MCP9700A/9701A) ±4°C from 0°C to +70°C (MCP9700/9701) while consuming 6 µA (typical) of operating current.

Unlike resistive sensors (such as thermistors), the Linear Active Thermistor IC does not require an additional signal-conditioning circuit. Therefore, the biasing circuit development overhead for thermistor solutions can be avoided by implementing this low-cost device. The voltage output pin (V_{OUT}) can be directly connected to the ADC input of a microcontroller. The MCP9700/9700A and MCP9701/9701A temperature coefficients are scaled to provide a 1°C/bit resolution for an 8-bit ADC with a reference voltage of 2.5V and 5V, respectively.

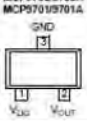
The MCP9700/9700A and MCP9701/9701A provide a low-cost solution for applications that require measurement of a relative change of temperature. When measuring relative change in temperature from +25°C, an accuracy of a 1°C (typical) can be realized from 0°C to +70°C. This accuracy can also be achieved by applying system calibration at +25°C.

In addition, this family is immune to the effects of parasitic capacitance and can drive large capacitive loads. This provides Printed Circuit Board (PCB) layout design flexibility by enabling the device to be remotely located from the microcontroller. Adding some capacitance at the output also helps the output transient response by reducing overshoots or undershoots. However, capacitive load is not required for sensor output stability.

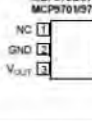
Package Type

3-Pin TO-92
MCP9700/9701
Only

3-Pin SOT-23
MCP9700/9700A
MCP9701/9701A



5-Pin SC-70
MCP9700/9700A
MCP9701/9701A



Freescale Semiconductor
Technical Data

MPX5100
Rev 11, 4/2008

Integrated Silicon Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated, and Calibrated

MPX5100/MPXV5100 SERIES

INTEGRATED PRESSURE SENSOR
0 to 100 kpa (0 to 14.5 psi)
15 to 115 kPa
(2.2 to 16.7 psi)
0.2 to 4.7 V Output

The MPX5100 series piezoresistive transducer is a state-of-the-art monolithic silicon pressure sensor designed for a wide range of applications, but particularly those employing a microcontroller or microprocessor with A/D inputs. This patented, single element transducer combines advanced micromachining techniques, thin-film metallization, and bipolar processing to provide an accurate, high level analog output signal that is proportional to the applied pressure.

Features

- 2.5% Maximum Error over 0° to 85°C
- Ideally suited for Microprocessor or Microcontroller-Based Systems
- Patented Silicon Shear Stress Strain Gauge
- Available in Absolute, Differential and Gauge Configurations
- Durable Epoxy Unibody Element
- Easy-to-Use Chip Carrier Option

Typical Applications

- Patient Monitoring
- Process Control
- Pump/Motor Control
- Pressure Switching

SMALL OUTLINE PACKAGES

ORDERING INFORMATION

Device Type	Options	Case No.	MPS Series Order Number	Device Marking
UNIBODY PACKAGE (MPX5100 SERIES)				
Basic Elements	Absolute	867	MPX5100A	MPX5100A
	Differential	867	MPX5100D	MPX5100D
Ported Elements	Differential Dual Ports	867C	MPX5100DP	MPX5100DP
	Absolute, Single Port	867B	MPX5100AP	MPX5100AP
	Gauge, Single Port	867B	MPX5100GP	MPX5100GP
	Gauge, Axial PC Mount	867F	MPX5100GSX	MPX5100D
	Gauge, Axial Port, SMT	482A	MPXV5100GC6U	MPXV5100G
	Gauge, Axial Port, DIP	482C	MPXV5100GC7U	MPXV5100G
	Gauge, Dual Port, SMT	1351	MPXV5100DP	MPXV5100
	Gauge, Side Port, SMT	1369	MPXV5100GP	MPXV5100C

PIN NUMBER(1)

1	N/C	5	N/C
2	V_S	6	N/C
3	GND	7	N/C
4	V_{OUT}	8	N/C

1. Pins 1, 5, 6, 7, and 8 are internal device connections. Do not connect to external circuitry or ground. Pin 1 is noted by the notch in the lead.

PIN NUMBER(1)

1	V_{OUT}	4	N/C
2	GND	5	N/C
3	V_S	6	N/C

1. Pins 4, 5, and 6 are internal device connections. Do not connect to external circuitry or ground. Pin 1 is noted by the notch in the lead.

UNIBODY PACKAGES

freescale

Freescale Semiconductor
Technical Data

MPX5010
Rev 11, 01/2007

Integrated Silicon Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated and Calibrated

MPX5010 MPXV5010G SERIES

INTEGRATED
PRESSURE SENSOR
0 to 10 kPa (0 to 1.45 psi)
0.2 to 4.7 V OUTPUT

The MPX5010/MPXV5010G series piezoresistive transducers are state-of-the-art monolithic silicon pressure sensors designed for a wide range of applications, but particularly those employing a microcontroller or microprocessor with A/D inputs. This transducer combines advanced micromachining techniques, thin-film metallization, and bipolar processing to provide an accurate, high level analog output signal that is proportional to the applied pressure.

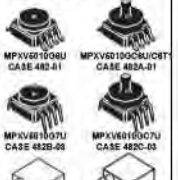
Features

- 5.0% Maximum Error over 0° to 85°C
- Ideally Suited for Microprocessor or Microcontroller-Based Systems
- Durable Epoxy Unibody and Thermoplastic (PPS) Surface Mount Package
- Temperature Compensated over −40° to +125°C
- Patented Silicon Shear Stress Strain Gauge
- Available in Differential and Gauge Configurations
- Available in Surface Mount (SMT) or Through-hole (DIP) Configurations

Application Examples

- Hospital Beds
- HVAC
- Respiratory Systems
- Process Control

SMALL OUTLINE PACKAGE



ORDERING INFORMATION

Device Type	Options	Case No.	MPX Series Order No.	Packing Options	Device Marking
SMALL OUTLINE PACKAGE (MPXV5010G SERIES)					
Basic Elements	Gauge, Element Only, SMT	482	MPXV5010G6U	Rails	MPXV5010G
	Gauge, Element Only, DIP	482B	MPXV5010G7U	Rails	MPXV5010G
Ported Elements	Gauge, Axial Port, SMT	482A	MPXV5010GC6U	Rails	MPXV5010G
	Gauge, Axial Port, DIP	482C	MPXV5010GC7U	Rails	MPXV5010G
	Gauge, Axial Port, SMT	482A	MPXV5010GC6T1	Tape & Reel	MPXV5010G
	Gauge, Side Port, SMT	1369	MPXV5010GP	Trays	MPXV5010G
	Gauge, Dual Port, SMT	1351	MPXV5010DP	Trays	MPXV5010G
UNIBODY PACKAGE (MPX2053 SERIES)					
Basic Element	Differential	867	MPX5010D	—	MPXV5010D
Ported Elements	Differential, Gauge	867C	MPX5010DP	—	MPXV5010DP
	Gauge	867B	MPX5010GP	—	MPXV5010GP
	Gauge, Axial	867E	MPX5010GS	—	MPXV5010D
	Gauge, Axial PC Mount	867F	MPX5010GSX	—	MPXV5010D

UNIBODY PACKAGE PIN NUMBERS(1)

1	N/C	5	N/C
2	GND	6	N/C
3	V_S	7	N/C
4	V_{out}	8	N/C

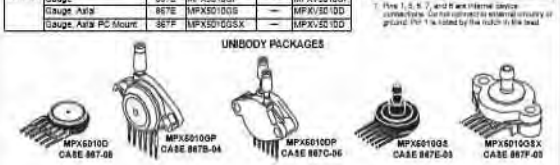
1. Pins 1, 5, 6, 7, and 8 are internal device connections. Do not connect to external circuitry or ground. Pin 1 is noted by the notch in the lead.

SMALL OUTLINE PACKAGE PIN NUMBERS(1)

1	N/C	5	N/C
2	V_S	6	N/C
3	GND	7	N/C
4	V_{out}	8	N/C

1. Pins 1, 5, 6, 7, and 8 are internal device connections. Do not connect to external circuitry or ground. Pin 1 is noted by the notch in the lead.

UNIBODY PACKAGES



freescale

Freescale Semiconductor
Technical Data

MPXV5004G
Rev 10, 01/2007

Integrated Silicon Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated and Calibrated

MPXV5004G SERIES

INTEGRATED PRESSURE SENSOR
0 TO 3.92 kPA
(0 TO 400 mm H_2O)
1.0 TO 4.8 V OUTPUT

The MPXV5004G series piezoresistive transducer is a state-of-the-art monolithic silicon pressure sensor designed for a wide range of applications, but particularly those employing a microcontroller or microprocessor with A/D inputs. This sensor combines a highly sensitive implanted strain gauge with advanced micromachining techniques, thin-film metallization, and bipolar processing to provide an accurate, high level analog output signal that is proportional to the applied pressure.

Features

- Temperature Compensated over 10° to 60°C
- Available in Gauge Surface Mount (SMT) or Through-Hole (DIP) Configurations
- Durable Thermoplastic (PPS) Package

Typical Applications

- Washing Machine Water Level
- Ideally Suited for Microprocessor or Microcontroller-Based Systems

SMALL OUTLINE PACKAGES THROUGH-HOLE

MPXV5004G7U
CASE 482B-05

MPXV5004GC7U
CASE 482C-03

ORDERING INFORMATION[1]

Device Type	Case No.	MPXV Series Order No.	Packing Options	Device Marking
Through-Hole	482B	MPXV5004G7U	Rails	MPXV5004G
	482C	MPXV5004GC7U	Rails	MPXV5004G
Surface Mount	482	MPXV5004G6U	Rails	MPXV5004G
	482	MPXV5004G6T1	Tape & Reel	MPXV5004G
	482A	MPXV5004GC6U	Rails	MPXV5004G
	482A	MPXV5004GC6T1	Tape & Reel	MPXV5004G
	1351	MPXV5004DP	Trays	MPXV5004G
	1369	MPXV5004GVP	Trays	MPXV5004G
	1369	MPXV5004GP	Trays	MPXV5004G

1. MPXV5004G series pressure sensors are available in the basic element package or with a pressure port. Two packing options are offered for the surface mount configuration.

PIN NUMBERS[1]

1	N/C	5	N/C
2	V_{S1}	6	N/C
3	GND	7	N/C
4	V_{out}	8	N/C

1. Pins 1, 5, 6, 7, and 8 are internal device connections. Do not connect to external circuitry or ground. Pin 1 is noted by the notch in the lead.

SMALL OUTLINE PACKAGES SURFACE MOUNT

MPXV5004G6U
CASE 482-01

MPXV5004GC6U
CASE 482A-01

MPXV5004DP
CASE 1351-01

MPXV5004GVP
CASE 1369-01

MPXV5004GP
CASE 1369-01

freescale semiconductor

Preliminary

Freescale Semiconductor

MPX4250
Rev 7, 1/2009

Integrated Silicon Pressure Sensor On-Chip Signal Conditioned, Temperature Compensated and Calibrated

MPX4250 Series
0 to 250 kPa (0 to 36.3 psi)
0.2 to 4.9 V Output

The MPX4250 series piezoresistive transducer is a state-of-the-art monolithic silicon pressure sensor designed for a wide range of applications, particularly those employing a microcontroller or microprocessor with A/D inputs. This transducer combines advanced micromachining techniques, thin-film metallization, and bipolar processing to provide an accurate, high-level analog output signal that is proportional to the applied pressure. The small form factor and high reliability of on-chip integration make the Freescale sensor a logical and economical choice for the automotive system engineer.

Application Examples

- Ideally Suited for Microprocessor or Microcontroller-Based Systems

Features

- Differential and Gauge Applications Available
- 1.4% Maximum Error Over 0° to 85°C
- Patented Silicon Shear Stress Strain Gauge
- Temperature Compensated Over −40° to +125°C
- Offers Reduction in Weight and Volume Compared to Existing Hybrid Modules
- Durable Epoxy Unibody Element

ORDERING INFORMATION

Device Name	Package Options	Case No.	# of Ports			Pressure Type			Device Marking
			None	Single	Dual	Gauge	Differential	Absolute	
Unibody Package (MPX4250 Series)									
MPX4250D	Tray	867	•				•		MPX4250D
MPX4250GP	Tray	867B		•		•			MPX4250GP
MPX4250DP	Tray	867C			•		•		MPX4250DP

UNIBODY PACKAGES

MPX4250D
CASE 867

MPX4250GP
CASE 867B

MPX4250DP
CASE 867C

freescale semiconductor