ESCUELA POLITÉCNICA NACIONAL

ESCUELA DE INGENIERÍA

"DISEÑO Y CONSTRUCCIÓN DE UN MÓDULO DIDÁCTICO UTILIZANDO EL PSOC (PROGRAMABLE SYSTEM ON CHIP) CY8C27643"

PROYECTO PREVIO A LA OBTENCIÓN DEL TÍTULO
DE INGENIERO EN ELECTRÓNICA Y CONTROL

GALO ROBERTO CÁCERES LÓPEZ
JOSÉ LUIS LÓPEZ ALBÁN

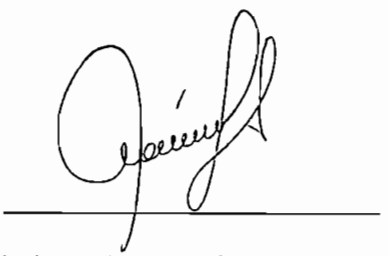
DIRECTOR: DR. LUIS CORRALES

Quito, Junio 2005

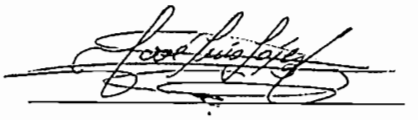
DECLARACIÓN

Nosotros, Galo Roberto Cáceres López y José Luis López Albán, declaramos bajo juramento que el trabajo aquí descrito es de nuestra autoría; que no ha sido previamente presentada para ningún grado o calificación profesional; y, que hemos consultado las referencias bibliográficas que se incluyen en este documento.

A través de la presente declaración cedemos nuestros derechos de propiedad intelectual correspondientes a este trabajo, a la Escuela Politécnica Nacional, según lo establecido por la Ley de Propiedad Intelectual, por su reglamento y por la normatividad institucional vigente.



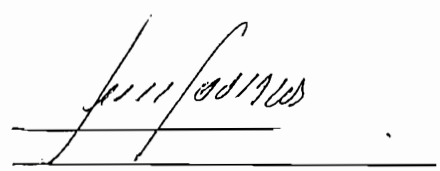
Galo Roberto Cáceres López



José Luis López Albán

CERTIFICACIÓN

Certifico que el presente trabajo fue desarrollado por Galo Roberto Cáceres López y José Luis López Albán, bajo mi supervisión.



Dr. Luis Corrales
DIRECTOR DEL PROYECTO

AGRADECIMIENTOS

Agradezco al Dr. Luis Corrales por su valiosa ayuda y acertada dirección en el desarrollo de este proyecto.

Un agradecimiento especial al Ing. Luis Oñate quien sin ningún egoísmo, aportó desinteresadamente con sus conocimientos, haciendo que muchas dificultades encontradas durante la realización de este proyecto, puedan resolverse.

A mi compañero y amigo José Luis López, por su fuerza, voluntad y alegría, virtudes que hicieron que este sueño se cumpla, a pesar de los múltiples problemas que se nos presentaron.

A mis amigos y compañeros de carrera "Los Carnales": Javier, Jorge, July, Kike, Diego, Gimmy, quienes siempre estuvieron preocupados y prestos a brindar su colaboración.

A mi tía Rosita, quien siempre me ayudó y me dio ánimos para seguir adelante, superando así muchas barreras que en mi vida como estudiante viví.

A mi tía y primas: Fanny, Paola y Karla, por toda su colaboración y esfuerzo.

Un gran agradecimiento especial para Mauro, persona que durante toda mi carrera me brindó su apoyo incondicional.

A un gran amigo Sr. Vicente Fierro, por su paciencia y comprensión.

Roberto

DEDICATORIA

El presente trabajo está dedicado a:

Dios por darme fe y voluntad, virtudes con las cuales he podido crecer como persona y así conseguir una de las metas más importantes en mi vida.

Mi madre Zoila López, por brindarme todo su amor, ternura y comprensión, por estar junto a mí en los momentos tristes y alegres, por hacer que con una palabra suya, todo sea más fácil.

Mi padre Galo Cáceres, porque a pesar de los problemas que hemos tenido, ha sabido transmitirme su amor y apoyo.

Mi hermano Santiago, por ser mi confidente y amigo durante toda mi vida, por su apoyo y aliento constante para la culminación de este trabajo.

Mi tío Víctor por el ejemplo de vida que me ha dado, por su lucha y honestidad, por su cariño y apoyo.

Mi hermana Marisol, por estar siempre junto a mí, compartiendo mis triunfos y fracasos.

Mi abuelita Liceña Vega, por su amor, preocupación y sabios consejos.

Mis tías Mercedes, Patricia, Blanca por todo su cariño y preocupación.

Sarita, quién ha sido un soporte importante en este proceso, por su amor, ternura y comprensión.

Todos mis primos Aldo, Ma. Mercedes, Karlita, Mateo, Gabrielito, Pablito, por ser la alegría que llena mi corazón.

Andrés y Andrea, dos hermanos más, para quienes siempre he querido ser un ejemplo, y porque a pesar de las pruebas difíciles que la vida nos ha puesto, hemos podido mantenernos unidos.

Roberto

AGRADECIMIENTOS

Agradezco al Phd. Luis Corrales por su dedicación y entrega para hacer posible la culminación de este proyecto.

También quiero agradecer a un gran amigo Ing. Luis Oñate quien con su ayuda incondicional supo brindar sus conocimientos y preparación, permitiéndome salir de muchas dudas durante la creación del proyecto.

A mi amigo y compañero Roberto Cáceres quien aporto con todos sus conocimientos académicos, ganas y esfuerzo para salir adelante en la realización de este proyecto.

A mis amigos y compañeros con quienes compartí alegrías y tristezas en el transcurso de mi carrera, primer eslabón de mi profesionalismo que hoy llega a su fin.

A mis profesores quienes con su pedagogía y conocimientos hicieron posible engrandecer el nivel académico alcanzado.

José Luis

DEDICATORIA

El presente trabajo está dedicado a:

Dios por permitirme culminar mis estudios haciendo de mi un profesional lleno de sabiduría e inteligencia, factores necesarios para emprender este nuevo camino.

Mi padre Guillermo López Echeverría quien con sus sabios consejos y ejemplo de profesional exitoso me fortaleció y guió para hacer posible esta meta que hoy la estoy haciendo realidad.

Mi madre Gladys Albán Ordóñez que con su ayuda y comprensión abnegada estuvo junto a mi en momentos buenos y malos demostrándome todo su amor, cariño y apoyo.

Mis hermanos Omar y Guillermo con quienes compartí mi vida estudiantil derrotando los obstáculos que desde niños les hicimos frente sin temor a tropezar.

Mi hermana Amparito quien con su amor y ternura fue el pilar de apoyo en momentos difíciles en el trascurso de mi vida.

Mi enamorada María José quien me brindó todo su apoyo durante la ejecución de este trabajo y que junto a su amor y cariño he podido salir adelante en momentos de soledad y desesperación.

José Luis

RESUMEN

En el presente trabajo se busca aprovechar al máximo las características básicas de los microprocesadores PSOC de la Cypress Microsystems, en particular, el PSOC CY8C27643.

Estos dispositivos tienen como característica principal la facilidad que presentan para ser programados y así satisfacer las facilidades del usuario.

Para demostrar la versatilidad de este dispositivo se procedió a diseñar e implementar una serie de ejemplos que además, corroboran su facilidad de configuración y operación, lo que permitirá a profesores y estudiantes familiarizarse rápidamente con esta tecnología.

Una vez terminado el trabajo, se puede afirmar que los ejercicios desarrollados efectivamente demuestran la facilidad y versatilidad de los PSOCs; en especial el tiempo reducido que se debe emplear para desarrollar una aplicación.

PRESENTACIÓN

El continuo avance tecnológico de los dispositivos electrónicos, principalmente el de los circuitos integrados, ha permitido aumentar la cantidad de elementos internos en su arquitectura, y a la vez reducir el tamaño físico de su encapsulado, sin que esto afecte de manera alguna su rendimiento.

La realización de este trabajo tiene como finalidad presentar un modulo de tipo didáctico, a través del cual se pueda conocer, aprender y manejar de una manera rápida y eficaz al microcontrolador PSOC (Programmable System on Chip). El PSOC específico utilizado para el desarrollo de este proyecto, es el CY8C27643.

Para una comprensión adecuada del usuario, el presente trabajo se ha dividido en diferentes capítulos, los cuales están distribuidos de la siguiente manera:

En el capitulo uno se realiza una descripción completa sobre el PSOC CY8C27643, además un estudio de los bloques y elementos que constituyen su arquitectura, incluyendo cada una de sus características y modos de funcionamiento.

Con estas bases, en el capitulo dos se presenta una descripción sobre los lenguajes de programación con los cuales se puede realizar un determinado firmware. Estos lenguajes se desarrollan a través del programa computacional PSOC Designer.

En e capitulo tres se presenta el diseño y construcción de las etapas y elementos que conformaran el Módulo Didáctico desarrollado, los cuales servirán para realizar las diferentes aplicaciones propuestas en este proyecto.

Determinada la estructura del hardware del módulo, en el capitulo cuatro se procede a realizar el diseño de las aplicaciones, utilizando las etapas que lo conforman.

Finalmente se presentan las conclusiones y recomendaciones obtenidas en base a las experiencias adquiridas a o largo del desarrollo del presente trabajo. Además, se ponen de manifiesto los anexos necesarios para la complementación de cada uno de los temas tratados.

CONTENIDO

CAPÍTULO 1

ESTUDIO DEL MICROCONTROLADOR PSOC CY8C27643

1.1 DESCRIPCIÓN DEL PSOC....2
1.1.1 ARQUITECTURA DEL MICROCONTROLADOR PSOC CY8C27643....3
1.2 INFORMACIÓN DE PINES....7
1.3 EL NÚCLEO DEL CPU (M8C)....9
1.3.1 ARQUITECTURA DEL NÚCLEO....9
1.4 ROM DE SUPERVISIÓN (SROM)....10
1.4.1 DESCRIPCIÓN DE LA ARQUITECTURA....10
1.4.2 DESCRIPCIÓN DE LAS FUNCIONES DE LA SROM....12
1.4.2.1 Función SWBootReset....12
1.4.2.2 Función lectura de bloque....12
1.4.2.3 Función escritura de bloque....13
1.4.2.4 Función borrado de bloque....15
1.4.2.5 Función checksum....15
1.4.2.6 Función lectura de tabla....16
1.4.2.7 Función de calibración 0....17
1.4.2.8 Función de Calibración 1....17
1.5 CONTROLADOR DE INTERRUPCIÓN....18
1.6 ENTRADAS / SALIDAS DE PROPOSITO GENERAL (GPIO)....19
1.6.1 DESCRIPCIÓN DE LA ARQUITECTURA....22
1.7 MANEJADORES DE SALIDAS ANÁLOGAS (DRIVERS)....24
1.8 OSCILADOR PRINCIPAL INTERNO (IMO)....26
1.9 OSCILADOR DE BAJA VELOCIDAD INTERIOR (ILO)....26
1.10 OSCILADOR DE CRISTAL EXTERNO DE 32 KHZ (ECO)....27

1.11 PHASE LOCKED LOOP (PLL).....28
1.12 EL SLEEP TIMER Y EL WATCHDOG.....29
1.13 SISTEMA DIGITAL.....30
1.13.1 ARQUITECTURA.....30
1.14 INTERCONEXIÓN DIGITAL GLOBAL (GDI).....31
1.14.1 DESCRIPCIÓN DE LA ARQUITECTURA.....33
1.15 INTERCONEXIÓN DIGITAL DEL ARREGLO (ADI).....34
1.15.1 DESCRIPCIÓN DE LA ARQUITECTURA.....34
1.16 INTERCONEXION DIGITAL DE FILAS (RDI).....35
1.16.1 DESCRIPCIÓN DE LA ARQUITECTURA.....35
1.17 BLOQUES DIGITALES.....36
1.17.1 DESCRIPCIÓN DE LA ARQUITECTURA.....37
1.17.1.1 Multiplexores de Entrada.....37
1.17.1.2 De-Multiplexores de Salida.....38
1.17.1.3 Bloque de Señales de Enlace.....38
1.18 SISTEMA ANÁLOGO.....39
1.18.1 ARQUITECTURA ANÁLOGA DEL PSOC.....39
1.19 INTERFAZ ANÁLOGA.....41
1.19.1 DESCRIPCIÓN DE LA ARQUITECTURA.....41
1.19.1.1 Interfaz del Bus de Datos Análogos.....41
1.19.1.2 Interfaz del Bus Comparador Análogo.....41
1.20 ARREGLO ANÁLOGO.....43
1.21 CONFIGURACIÓN DE LAS ENTRADAS ANÁLOGAS.....44
1.22 REFERENCIA ANÁLOGA.....44
1.22.1 DESCRIPCIÓN DE LA ARQUITECTURA.....44

CAPÍTULO 2

LENGUAJES DE PROGRAMACIÓN PARA EL PSOC CY8C27643

2.1 LENGUAJE ENSAMBLADOR DEL PSOC DESIGNER....46
2.2 EL MICROPROCESADOR M8C....47
2.2.1 INTRODUCCIÓN....47
2.2.2 REGISTROS INTERNOS....47
2.2.3 ESPACIOS DE DIRECCIÓN....49
2.2.4 FORMATO DE LAS INSTRUCCIONES....51
2.2.4.1 Instrucciones de 1 byte....51
2.2.4.2 Instrucciones de 2 bytes....52
2.2.4.3 Instrucciones de 3 bytes....53
2.2.5 MODOS DE DIRECCIONAMIENTO....55
2.2.5.1 Fuente inmediata....55
2.2.5.2 Fuente directa....55
2.2.5.3 Fuente indexada....56
2.2.5.4 Destino directo....56
2.2.5.5 Destino Indexado....57
2.2.5.6 Destino directo desde una fuente inmediata....57
2.2.5.7 Destino indexado desde una fuente inmediata....58
2.2.5.8 Destino directo desde una fuente directa....58
2.2.5.9 Incremento de la posición de la fuente indirecta....58
2.2.5.10 Incremento de la posición del destino indirecto....59
2.3 ASSEMBLER DEL PSOC DESIGNER....60
2.3.1 FORMATO DEL ARCHIVO FUENTE....60
2.3.1.1 Etiquetas....61
2.3.1.1.1 Etiquetas locales....61
2.3.1.1.2 Etiquetas globales....61
2.3.1.1.3 Etiquetas locales reutilizables....61
2.3.1.2 Mnemónicos....61

2.3.1.3 Operandos....62
2.3.1.3.1 Constantes....62
2.3.1.3.2 Etiquetas....62
2.3.1.3.3 Operador Punto (.)....63
2.3.1.3.4 Registros....63
2.3.1.3.5 RAM....63
2.3.1.3.6 Expresiones....63
2.3.1.4 Comentarios....64
2.3.1.5 Directores....64
2.4 SET DE INSTRUCCIONES....65
2.5 COMPILADOR DE LENGUAJE "C" DEL PSOC DESIGNER....65
2.6 ARCHIVOS DEL COMPILADOR....65
2.6.1 ARCHIVO INICIAL....65
2.6.2 DESCRIPCIÓN DE LAS LIBRERÍAS....66
2.7 FUNDAMENTOS DEL COMPILADOR....67
2.7.1 TIPOS....67
2.7.2 OPERADORES....68
2.7.3 EXPRESIONES....68
2.7.4 DECLARACIONES....68
2.7.5 PUNTEROS....69
2.7.6 DIRECTORES DE PROCESO (#'S)....70
2.8 FUNCIONES....70
2.8.1 FUNCIONES DE LIBRERÍA....71
2.8.1.1 Funciones String....71
2.8.1.2 Funciones matemáticas....71
2.8.1.3 Funciones de la librería Software API (Application Programming Interface)....72
2.8.2 INTERFAZ ENTRE C Y ASSEMBLER....72
2.9 CONSIDERACIONES ADICIONALES....73
2.9.1 INTERRUPCIONES....73
2.9.2 REGISTROS DE I / O....74

2.9.3 ÁREAS DE MEMORIA.....75
2.9.4 ÁREAS DE LA MEMORIA FLASH.....75
2.9.5 MEMORIA DE DATOS.....75
2.9.6 STRINGS.....76
2.9.7 REGISTROS VIRTUALES.....76
2.10 PROGRAMACIÓN GRÁFICA EN EL PSOC DESIGNER.....76

CAPÍTULO 3

DISEÑO Y CONSTRUCCIÓN DEL MÓDULO DIDÁCTICO

3.1 DESCRIPCIÓN GENERAL.....81
3.2 DIAGRAMA DE BLOQUES DEL MÓDULO.....81
3.3 DISEÑO DE LA FUENTE DE ALIMENTACIÓN (Etapa 1).....82
3.4 ETAPA DE CONTROL.....84
3.5 ETAPA DE CONEXIONES PARA EL PROGRAMADOR.....85
3.6 ETAPA DE DESARROLLO.....86
3.6.1 PULSANTES Y DIPSWITCH.....86
3.6.2 TECLADO MATRICIAL.....87
3.6.3 LEDS.....87
3.6.4 DISPLAY DE 7 SEGMENTOS.....88
3.6.5 DISPLAY DE CRISTAL LÍQUIDO LCD.....89
3.6.6 CONECTORES BNC Y POTENCIÓMETRO.....90
3.6.7 SENSOR DE TEMPERATURA.....90
3.6.8 COMUNICACIÓN SERIAL RS232.....91
3.6.9 ZUMBADOR.....92

CAPÍTULO 4

DISEÑO DE APLICACIONES USANDO LAS ETAPAS Y ELEMENTOS QUE CONSTITUYEN EL MÓDULO DIDÁCTICO

4.1 CONFIGURACIÓN DE I/O PARA LECTURA Y ESCRITURA EN LOS PUERTOS....94
4.1.1 DIAGRAMA CIRCUITAL....95
4.1.2 DIAGRAMA DE FLUJO....95
4.2 MANEJO DE UN TECLADO MATRICIAL Y VISUALIZACIÓN EN UN DISPLAY....96
4.2.1 DIAGRAMA CIRCUITAL....97
4.2.2 DIAGRAMA DE FLUJO....97
4.3 MANEJO BÁSICO DEL LCD....99
4.3.1 DIAGRAMA CIRCUITAL....100
4.3.2 DIAGRAMA DE FLUJO....101
4.4 USO DEL CONVERSOR ANÁLOGO DIGITAL DE 12 BITS....110
4.4.1 DIAGRAMA CIRCUITAL....111
4.4.2 DIAGRAMA DE FLUJO....112
4.5 DISEÑO DE UN AMPLIFICADOR DIFERENCIAL....113
4.5.1 DIAGRAMA CIRCUITAL....114
4.5.2 DIAGRAMA DE FLUJO....115
4.6 IMPLEMENTACIÓN DE FILTROS....116
4.6.1 DIAGRAMA CIRCUITAL....117
4.6.2 DIAGRAMA DE FLUJO....118
4.7 GENERADOR DE SONIDO....119
4.7.1 DIAGRAMA CIRCUITAL....119
4.7.2 DIAGRAMA DE FLUJO....120

4.8 MEDICIÓN DE TEMPERATURA USANDO EL SENSOR DALLAS DS1821....121
4.8.1 DIAGRAMA CIRCUITAL....122
4.8.2 DIAGRAMA DE FLUJO....123
4.9 CONVERSOR VOLTAJE / FRECUENCIA PARA UNA SEÑAL ANÁLOGA....124
4.9.1 DIAGRAMA CIRCUITAL....125
4.9.2 DIAGRAMA DE FLUJO....126
4.10 MULTIPLICADOR ANÁLOGO DE DOS SEÑALES....127
4.10.1 DIAGRAMA CIRCUITAL....129
4.10.2 DIAGRAMA DE FLUJO....129
4.11 COMUNICACIÓN SERIAL CON UN PC....131
4.11.1 DIAGRAMA CIRCUITAL....132
4.11.2 DIAGRAMA DE FLUJO....133

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

5.1 CONCLUSIONES....139
5.2 RECOMENDACIONES....141

REFERENCIAS BLIBLIOGRÁFICAS....143

ANEXOS

ANEXO A

TABLAS DE REGISTROS DE LOS BLOQUES DEL PSOC CY8C27643.

ANEXO B

DIRECTORES DEL ASSEMBLER.

SET DE INSTRUCCIONES.

OPERADORES DEL COMPILADOR.

DIRECTORES PREPROCESADORES DEL COMPILADOR.

DIRECTORES PRAGMA DEL COMPILADOR.

FUNCIONES STRING DEL COMPILADOR

FUNCIONES MATEMÁTICAS DEL COMPILADOR

FUNCIONES API DEL COMPILADOR

ANEXO C

DATASHEET DEL DALLAS DS1821

COMUNICACIÓN ONE-WIRE

ANEXO D

TOPOLOGÍA Y PARÁMETROS PARA DISEÑO DE UN FILTRO.

FRECUENCIAS PARA UNA OCTAVA DE TONOS MUSICALES.

ANEXO E

CÓDIGOS FUENTE

ÍNDICE DE FIGURAS

Figura 1.1 Descripción de los parámetros para la selección de un PSOC.

Figura 1.2 Diagrama de bloques de la arquitectura del PSOC CY8C27643.

Figura 1.3 Empaquetado del PSOC CY8C27643 de 48 pines.

Figura 1.4 Diagrama de bloques del núcleo.

Figura 1.5 Diagrama de bloques del GPIO.

Figura 1.6 Diagrama de bloques del modo de interrupción del GPIO.

Figura 1.7 Diagrama de bloques de una salida análoga para un PSOC de 48 pines.

Figura 1.8 Conexiones del oscilador de cristal externo.

Figura 1.9 Diagrama de bloques del sistema digital para el PSOC CY8C27643.

Figura 1.10 Diagrama de bloques de la interconexión global.

Figura 1.11 Estructura del arreglo del bloque digital.

Figura 1.12 Vista detallada de la agrupación de los cuatro bloques.

Figura 1.13 Diagrama de bloques de los bloques digitales.

Figura 1.14 Diagrama de bloques del sistema análogo.

Figura 1.15 Una parte del bus comparador análogo.

Figura 1.16 Arreglo de los bloques análogos del PSOC CY8C27XXX

Figura 1.17 Esquemático del control de referencia análogo.

Figura 1.18 Estructura de referencia.

Figura 2.1 Espacios de dirección del microcontrolador M8C.

Figura 2.2 Selección de los Módulos de Usuario dentro del PSOC Designer.

Figura 2.3 Acceso al datasheet de un Módulo de Usuario.

Figura 2.4 Acceso al PDF completo del datasheet de un Módulo de Usuario.

Figura 2.5 Traspaso de un Módulo de Usuario a la vista de interconexión.

Figura 2.6 Procedimiento para enrolado del Módulo de Usuario.

Figura 2.7 Enrolado del Módulo de Usuario.

Figura 2.8 Parámetros del Módulos de Usuario.

Figura 2.9 Conexiones del Módulos de Usuario.

Figura 3.1 Diagrama de Bloques del Módulo Didáctico.

Figura 3.2 Fuente de Alimentación Regulada de 5 Vdc.

Figura 3.3 Etapa de Control.

Figura 3.4 Etapa de Conexiones para el Programador.
Figura 3.5 Diagrama de Pulsantes y Dipswitch.
Figura 3.6 Diagrama del Teclado Matricial.
Figura 3.7 Diagrama de Leds.
Figura 3.8 Diagrama del Display de 7 Segmentos.
Figura 3.9 Diagrama del LCD.
Figura 3.10 Diagrama de Conectores BNC y Potenciómetro.
Figura 3.11 Diagrama del Sensor de Temperatura.
Figura 3.12 Diagrama de conexiones del MAX232.
Figura 3.13 Diagrama del Zumbador.
Figura 4.1 Diagrama de Conexiones implementado para lectura y escritura de puertos.
Figura 4.2 Diagrama de Flujo para lectura y escritura de puertos.
Figura 4.3 Diagrama de conexiones implementado para manejo de teclado y visualización en un display.
Figura 4.4 Diagrama de Flujo para manejo de teclado y visualización en un display.
Figura 4.5 Diagrama de Flujo para manejo de teclado y visualización en un display.
Figura 4.6 Diagrama de conexiones implementado para manejo básico del LCD.
Figura 4.7 Diagrama de Flujo para manejo básico del LCD.
Figura 4.8 Diagrama de conexiones implementado para uso del conversor A-D de 12 bits.
Figura 4.9 Diagrama de Flujo para uso del conversor A-D de 12 bits.
Figura4.10 Diagrama de conexiones implementado para el amplificador diferencial
Figura 4.11 Diagrama de Flujo para el Amplificador Diferencial.
Figura 4.12 Diagrama de conexiones implementado para manejo de filtros.
Figura 4.13 Diagrama de Flujo para manejo de filtros.
Figura 4.14 Diagrama de conexiones implementado para el generador de sonidos.
Figura 4.15 Diagrama de Flujo para el generador de sonidos.
Figura 4.16 Diagrama de conexiones implementado para medición de temperatura con el DS1821.
Figura 4.17 Diagrama de Flujo para medición de temperatura con el DALLAS DS1821.
Figura 4.18 Diagrama de conexiones implementado para el conversor V /f de una señal.
Figura 4.19 Diagrama de Flujo para el conversor V /f de una señal.

Figura 4.20 Diagrama de conexiones implementado para el multiplicador análogo de dos señales.
Figura 4.21 Diagrama de Flujo para el multiplicador análogo de dos señales.
Figura 4.22 Señal resultante de la multiplicación de dos ondas
Figura 4.23 Señal resultante ampliada de la multiplicación de dos ondas.
Figura 4.24 Diagrama de conexiones implementado para comunicación serial con un PC
Figura 4.25 Diagrama de Flujo para comunicación serial con un PC
Figura 4.26 Diagrama de Flujo del programa para comunicación serial con un PC en Visual Basic
Figura 4.27 Adquisición de una onda senoidal

ÍNDICE DE TABLAS

Tabla 1.1 Descripción de pines para dispositivos de la familia CY8C27XXX.
Tabla 1.2 Descripción y distribución de pines para un PSOC CY8C27643-24PVI.
Tabla 1.3 Lista de las funciones de la SROM.
Tabla 1.4 Variables de la función SROM.
Tabla 1.5 Significado de los códigos de retorno en la SROM.
Tabla 1.6 Parámetros de la función lectura de bloque (01h) .
Tabla 1.7 Parámetros de la función escritura de bloque (02h).
Tabla 1.8 Parámetros de la función borrado de bloque (03h).
Tabla 1.9 Parámetros de la función checksum (07h).
Tabla 1.10 Parámetros de la función tabla de lectura.
Tabla 1.11 Parámetros de la función de calibración 0 (08h).
Tabla 1.12 Parámetros de la función de calibración 1 (09h).
Tabla 1.13 Lista de interrupciones.
Tabla 1.14 Puertos y buses globales.
Tabla 2.1 Registros internos.
Tabla 2.2 Registro de bandera (F).
Tabla 2.3 Formato de las instrucciones de 1 byte.
Tabla 2.4 Formatos de las instrucciones de 2 bytes.

Tabla 2.5 Formatos de las instrucciones de 3 bytes.

Tabla 2.6 Fuente inmediata.

Tabla 2.7 Fuente directa.

Tabla 2.8 Fuente indexada.

Tabla 2.9 Destino directo.

Tabla 2.10 Destino indexado.

Tabla 2.11 Destino directo de la fuente inmediata.

Tabla 2.12 Destino indexado de la fuente inmediata.

Tabla 2.13 Destino directo de la fuente directa.

Tabla 2.14 Incremento del lugar de la fuente indirecta.

Tabla 2.15 Incremento del lugar del destino indirecto.

Tabla 2.16 Cinco componentes básicos de un archivo de fuente ensamblador.

Tabla 2.17 Formatos de las constantes.

Tabla 2.18 Formatos de los registros.

Tabla 2.19 Formato de la RAM.

Tabla 2.20 Expresiones.

Tabla 2.21 Tipos de datos.

Tabla 2.22 Convenciones # pragma fastcall para paso del argumento.

Tabla 2.23 Convenciones # pragma fastcall para valores de retorno.

CAPÍTULO 1

ESTUDIO DEL MICROCONTROLADOR PSOC CY8C27643

CAPÍTULO 1

ESTUDIO DEL MICROCONTROLADOR PSOC CY8C27643

La realización de este proyecto se centra en el funcionamiento y manejo del microcontrolador PSOC (Programmable System on Chip). Este dispositivo ha sido diseñado para reemplazar los múltiples componentes de un sistema basado en un MCU tradicional.

El avance tecnológico de estos dispositivos ha permitido aumentar la cantidad de elementos internos en su arquitectura, y a la vez reducir el tamaño físico del encapsulado, sin que esto afecte de manera alguna, su rendimiento.

Los microcontroladores PSOC de la Cypress Microsystems se dividen en varias familias, las cuales tienen a su haber diferentes modelos. Estos modelos se seleccionan en base a la cantidad de periféricos y aplicaciones a manejarse; es decir, de acuerdo a las necesidades de cada diseñador electrónico. Para este caso se ha escogido el PSOC CY8C27643-24PVI de 48 pines.

El objetivo de este proyecto es diseñar y construir un módulo didáctico con el microcontrolador PSOC CY8C27643 que abarque y ejecute sus funciones básicas, analizando de esta manera las ventajas que presenta, en comparación con otros microcontroladores.

En el presente capítulo se realiza el estudio de los bloques y elementos que constituyen la arquitectura del microcontrolador PSOC CY8C27643, así como cada una de sus características.

1.1 DESCRIPCIÓN DEL PSOC

La numeración que presenta el PSOC escogido para este proyecto (CY8C27643), tiene un significado, el mismo que se traduce en base a los parámetros que se usan para su selección. En la Figura 1.1 se ilustran estos parámetros.

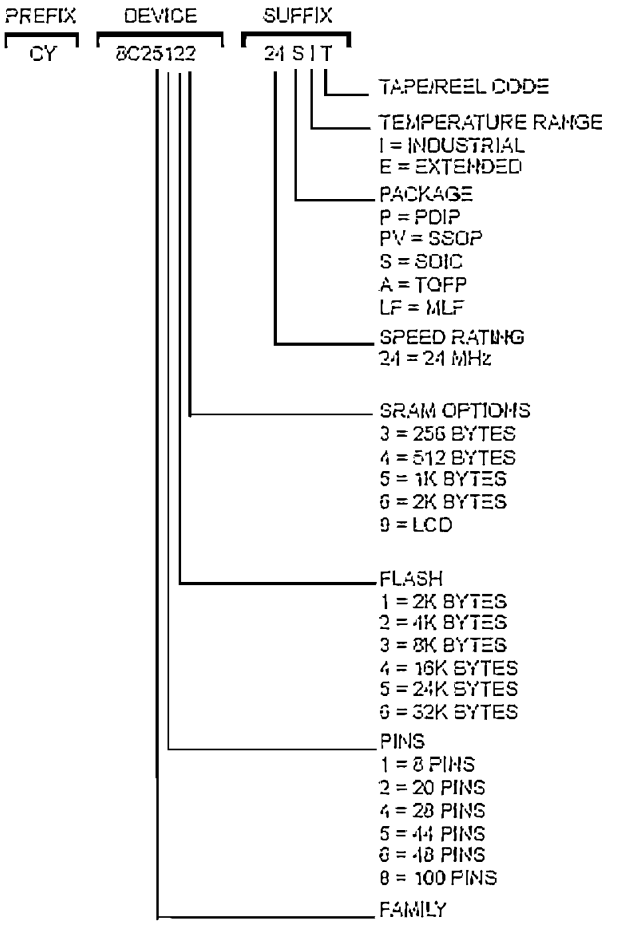


Figura 1.1 Descripción de los parámetros para la selección de un PSOC

El microcontrolador PSOC CY8C27643 es un dispositivo de alta tecnología que realiza múltiples funciones de control, aplicables a diferentes ámbitos de diseño, los cuales pueden ser desarrollados en base a los bloques programables que contiene en su estructura.

1.1.1 ARQUITECTURA DEL MICROCONTROLADOR PSOC CY8C27643

En la Figura 1.2 se ilustra la arquitectura de este PSOC

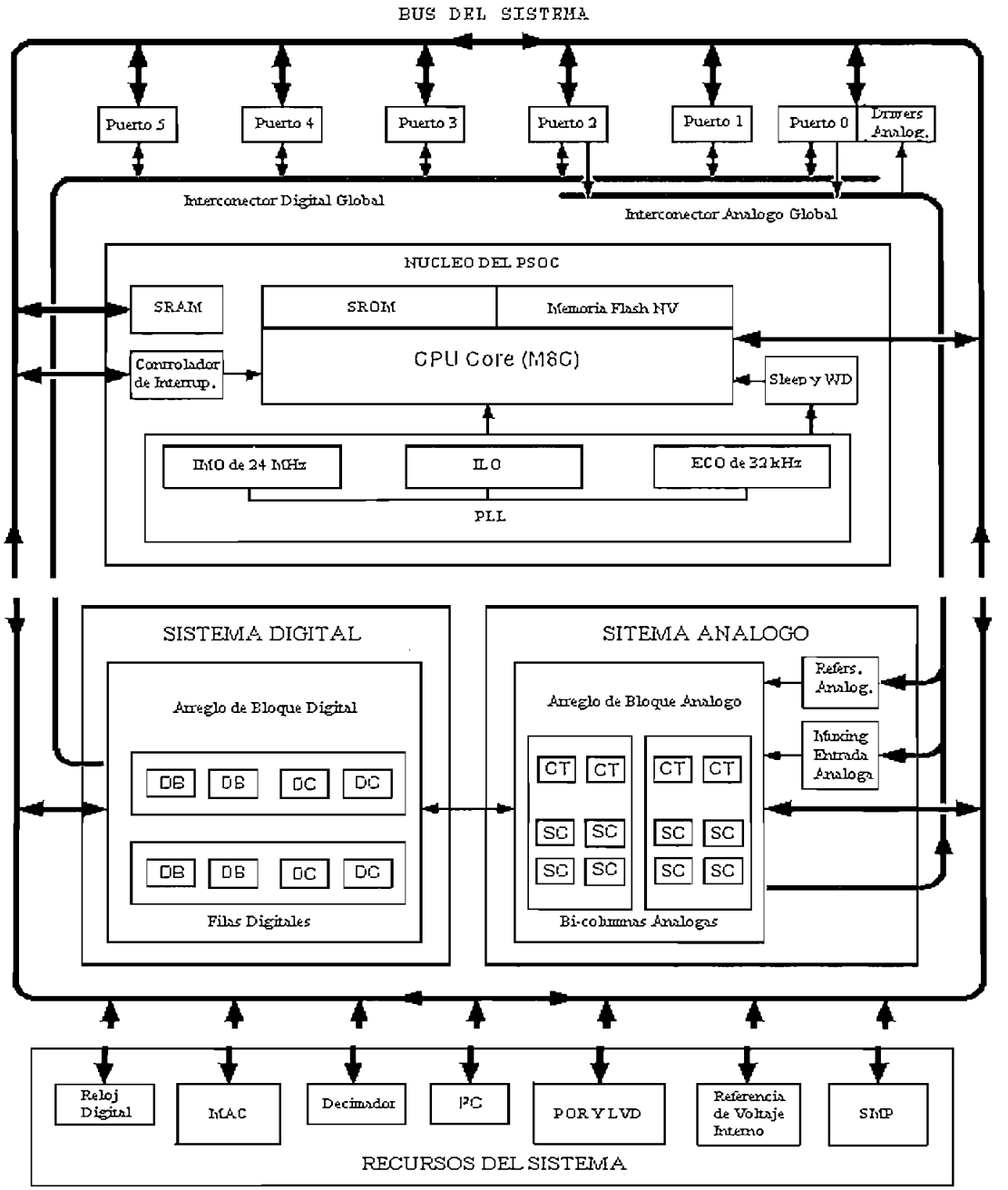


Figura 1.2 Diagrama de bloques de la arquitectura del PSOC CY8C27643

En referencia a la Figura 1.2, se observa que el microcontrolador PSOC CY8C27643 cuenta con tres bloques claramente definidos, estos son:

- Núcleo del PSOC (Core).
- Sistema Digital.
- Sistema Análogo.

Se tienen también los llamados **recursos del sistema**; estos son:

* Reloj Digital.
* Acumulador-Multiplicador (MAC).
* Decimator.
* I^2C.
* POR y LVD (Power on Reset y Low Voltaje Detect).
* Resets del Sistema.
* Referencia de Voltaje Interno.
* Switch Mode Pump (SMP).

Finalmente se cuenta con seis puertos, los cuales están comunicados continuamente con el bus general del sistema y con los interconectores para los sistemas análogo y digital. Los puertos son utilizados principalmente para manejo de periféricos, todos servirán para realizar interconexiones digitales, mientras que, solo los Puertos 0 y 2 se podrán usar para interconexiones análogas.

El PSOC CY8C27643 pertenece a la familia CY8C27XXX. A continuación se destacan las principales características de este microcontrolador:

PROCESADOR DE ARQUITECTURA HARVARD

- Procesador M8C a 24 MHz de velocidad.

- Multiplicador 8x8, acumulador de 32 bits.
- Baja potencia a velocidades altas.
- Voltaje de operación de 3.0 a 5.25 V.
- Operación a voltajes bajo 1V, usando el SMP (Switch Mode Pump).
- Rango de temperatura industrial: de –40°C a +85°C.

PERIFÉRICOS AVANZADOS (BLOQUES DEL PSOC)

- 12 bloques análogos provistos de:
  ADCs de hasta14 bits, DACs de hasta 9 bits .
  Amplificadores de ganancia programable.
  Filtros y comparadores programables.
- 8 bloques digitales provistos de:
  Timers, contadores y PWMs de 8 a 32 bits.
  Módulos CRC (chequeo de redundancia cíclica) y PRS (secuencia de la pseudo random)
  Hasta 2 UARTs Full-Duplex.
  SPI múltiple. Masters o esclavos.
  Conexión a todos los pines del microcontrolador, que tienen el bloque GPIO (I/O de propósito general).
- Periféricos complejos por combinación de bloques.

PRECISIÓN, RELOJ PROGRAMABLE

- Oscilador interno +/- 2.5% 24/48 MHz.
- 24/48 MHz con cristal opcional de 32 kHz.
- Oscilador externo opcional de hasta 24 MHz.
- Oscilador interno para el Watchdog y el Sleep.

MEMORIA ON-CHIP FLEXIBLE

- ξ 16 Kbytes de almacenamiento de programa flash.
- ξ 50.000 ciclos para borrar / escribir.
- ξ 256 bytes de almacenamiento de datos en la SRAM.
- ξ Sistema de programación serial ISSP (In-System Serial Programing).
- ξ Actualización parcial de la Flash.
- ξ Modos de protección flexibles.
- ξ Emulación de la EEPROM en la Flash.

CONFIGURACIÓN DE LOS PINES PROGRAMABLES

- ξ Manejo de 25mA en todo el GPIO.
- ξ Pull Up, Pull Down, alta impedancia (Z), strong o drenaje abierto. Modos de manejo en todo el GPIO (Strong Modes).
- ξ Hasta 12 entradas análogas en el GPIO.
- ξ 4 salidas análogas de 40mA en el GPIO.
- ξ Interrupción configurable en todo el GPIO.

RECURSOS ADICIONALES DEL SISTEMA

- ξ Maestro, Esclavo I^2C y Multi-Maestro a 400 KHz.
- ξ Watchdog y Sleep Timers.
- ξ Detector de bajo voltaje configurable para el usuario.
- ξ Circuito supervisor integrado.
- ξ Referencia del voltaje de precisión On-Chip.

HERRAMIENTAS DE DESARROLLO COMPLETO

- ξ Software libre de desarrollo (PSOC Designer).

- Emulación de velocidad completa.
- 128 Kbytes de memoria de rastreo.

1.2 INFORMACIÓN DE PINES

En la Tabla 1.1 se presenta un resumen de los pines de los dispositivos pertenecientes a la familia CY8C27XXX.

Nombre del pin	Descripción	I/O
SMP	Switch Mode Pump	Power
VCC	Voltaje de suministro	Power
VSS	Tierra	Power
XRES	Reset externo(activación en alto)	I
P0(0)-P0(1)	Puerto 0(0), 0(1) Entradas análogas	I/O
P0(2)-P0(5)	Puerto 0(2), 0(3), 0(4), 0(5) Entradas/Salidas análogas	I/O
P0(6)-P(0)7	Puerto 0(6), 0(7) Entradas análogas	I/O
P1(0)	Puerto 1(0), XTALOut/SDATA/I^2C SDA opcional	I/O
P1(1)	Puerto 1(1), XTALIn/SCLK/I^2C SCL opcional	I/O
P1(2)-P1(3)	Puerto 1(2), 1(3)	I/O
P1(4)	Puerto 1(4) EXT CLK	I/O
P1(5)	Puerto 1(5) I^2C SDA	I/O
P1(6)	Puerto 1(6)	I/O
P1(7)	Puerto 1(7) I^2C SCL	I/O
P2(0)-P2(3)	Puerto 2(0), 2(1), 2(2), 2(3) Entradas análogas no multiplexadas	I/O
P2(4)	Puerto 2(4) AGND externa	I/O
P2(5)	Puerto 2(5)	I/O
P2(6)	Puerto 2(6) VREF externo	I/O
P2(7)	Puerto 2(7)	I/O
P3(0)-P3(7)	Puerto 3(0), 3(1), 3(2), 3(3), 3(4), 3(5), 3(6), 3(7)	I/O
P4(0)-P4(7)	Puerto 4(0), 4(1), 4(2), 4(3), 4(4), 4(5), 4(6), 4(7)	I/O
P5(0)-P5(3)	Puerto 5(0), 5(1), 5(2), 5(3),	I/O

Tabla 1.1 Descripción de pines para dispositivos de la familia CY8C27XXX

A continuación la Tabla 1.2 detalla la distribución de pines para el PSOC CY8C27643-24PVI.

N° pin	Descripción	N° pin	Descripción	N° pin	Descripción
1	P0(7) A, I	17	P3(1)	33	P3(4)
2	P0(5) A, I/O	18	P5(3)	34	P3(6)
3	P0(3) A, I/O	19	P5(1)	35	XRES
4	P0(1) A, I	20	P1(7) I^2C SCL	36	P4(0)
5	P2(7)	21	P1(5) I^2C SDA	37	P4(2)
6	P2(5)	22	P1(3)	38	P4(4)
7	P2(3) A, I (ASC10)	23	P1(1) XTALIn, I^2C SCL, TC SCLK	39	P4(6)
8	P2(1) A, I (ASD20)	24	VSS	40	P2(0) A, I (ASC23)
9	P4(7)	25	P1(0) XTALOut, I^2C SDA, TC SDATA	41	P2(2) A, I (ASD23)
10	P4(5)	26	P1(2)	42	P2(4) A, I (AGND)
11	P4(3)	27	P1(4) EXT CLK	43	P2(6) A, I
12	P4(1)	28	P1(6)	44	P0(0) A, I
13	SMP	29	P5(0)	45	P0(2) A, I/O
14	P3(7)	30	P5(2)	46	P0(4) A, I/O
15	P3(5)	31	P3(0)	47	P0(6) A, I
16	P3(3)	32	P3(2)	48	VCC

A: Análogo D: Digital I/O: Entrada o Salida

Tabla 1.2 Descripción y distribución de pines para un PSOC CY8C27643-24PVI

En la Figura 1.3 se muestra una gráfica del empaquetado de este dispositivo

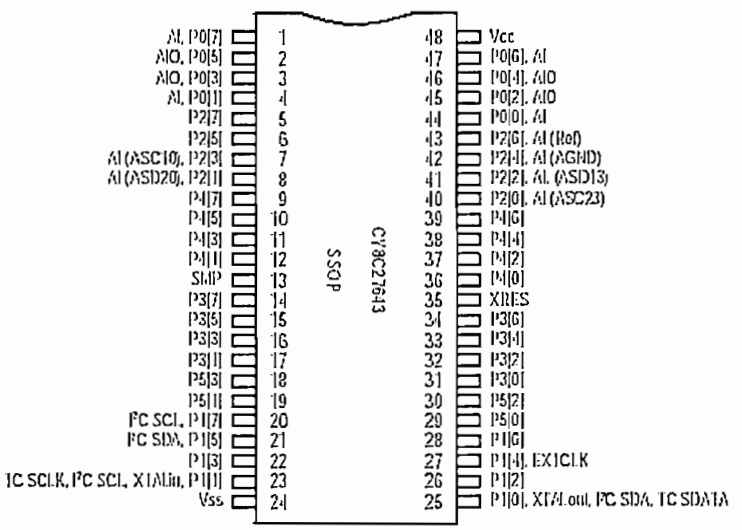


Figura 1.3 Empaquetado del PSOC CY8C27643 de 48 pines

1.3 EL NÚCLEO DEL CPU (M8C)

El núcleo o "core" en estos dispositivos es llamado M8C, en conjunto con los registros que están asociados a él.

1.3.1 ARQUITECTURA DEL NÚCLEO

En la Figura 1.4 se observa el diagrama de bloques del núcleo de un microcontrolador PSOC CY8C27643 :

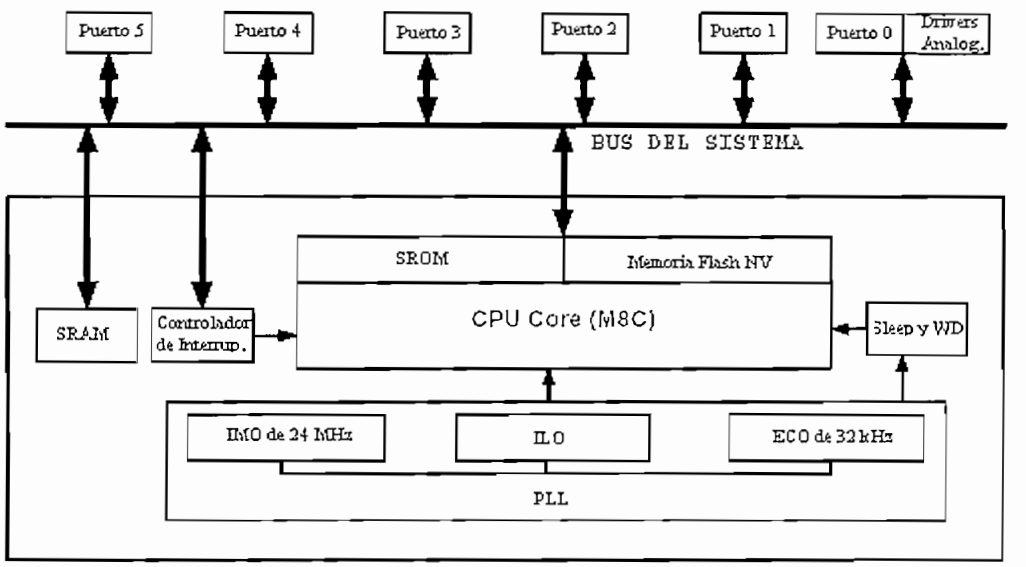


Figura 1.4 Diagrama de bloques del núcleo

El núcleo será el encargado controlar todo el sistema, está constituido por el CPU M8C, los espacios de memoria, el controlador de interrupciones, el sleep y watch dog timer, y finalmente por un bloque para manejo de la señal de reloj (cristales).

El M8C es un microprocesador de arquitectura Harvard 4 MIPS, 8 bits. El código seleccionable de velocidad del reloj del procesador, que va de 93.7 kHz a 24 MHz, permite al M8C ponerse a punto en concordancia con los requerimientos de aplicaciones particulares y energía. El M8C sustenta un "rico" set de instrucciones el

cual permite un soporte eficiente para un lenguaje de bajo nivel. En el M8C existen cinco registros internos que serán analizados en el capítulo siguiente, cuando se trate lo referente al lenguaje ensamblador del PSOC. Los espacios de dirección del M8C, también serán tratados detalladamente en el capítulo siguiente, sin embargo, a continuación se analizará lo referente a la ROM de Supervisión (SROM).

1.4 ROM DE SUPERVISIÓN (SROM)

La SROM ejecuta, un código que se usa para calibrar circuitería y para ejecutar operaciones de la Flash. Dentro de esta se deben tomar en cuenta dos aspectos: su bloque físico, y el código guardado en ella.

1.4.1 DESCRIPCIÓN DE LA ARQUITECTURA

A las funciones de la SROM se puede acceder mediante la ejecución de la instrucción: Supervisory System Call (SSC), la cual tiene un código inicial de 0x00. Antes de la ejecución del SSC el acumulador del M8C necesita ser cargado con el código de la función SROM deseada, tal como se observa en la Tabla 1.3.

Código de la función	Nombre de la función	Espacio que usa el stack
0X00	SWBootReset	0
0X01	Bloque de lectura	7
0X02	Bloque de escritura	9
0X03	Bloque de borrado	9
0X06	Lectura de tabla	3
0X07	CheckSum	3
0X08	Calibración 0	4
0X09	Calibración 1	3

Tabla 1.3 Lista de las funciones de la SROM

Funciones indefinidas causarán un HALT (parada), si se hizo la llamada desde un código de usuario.

Las funciones de la SROM ejecutan o procesan un código con "llamadas", por lo tanto, requieren espacio del stack. Con excepción del Reset, todas estas funciones tienen un parámetro en la SRAM, el cual debería estar configurado antes de la ejecución del SSC.

En la Tabla 1.4 se lista todas las posibles variables del parámetro antes mencionado.

Nombre de la variable	Dirección SRAM
KEY 1 / CONTADOR / RETURN CODE	0,F8h
KEY 2 / TMP	0,F9h
BLOCKID	0,FAh
POINTER	0,FBh
CLOCK	0,FCh
MODE	0,FDh
DELAY	0,FEh

Tabla 1.4 Variables de la función SROM

Existen dos variables importantes que son usadas para todas las funciones, estas son: KEY 1 y KEY 2, las cuales sirven para distinguir entre SSCs válidos y SSCs inadvertidos. El valor que debe tener inicialmente KEY 1 es de 0x3A; mientras que KEY 2 debería tener el mismo valor que el stack pointer cuando la función SROM empieza a ejecutarse.

La SROM tiene además la siguiente característica adicional:

CÓDIGOS DE RETORNO:

Estos determinan el éxito o la falla de una función particular.

Este código de retorno está almacenado en la posición KEY 1. Las funciones CheckSum y Lectura de Tabla no tienen código de retorno por que la posición KEY 1 es usada para el retorno de otros datos.

Valor del código de retorno	Descripción
0X00	Exitoso
0X01	Función no permitida debido al nivel de protección
0X02	Reset del software sin reset del hardware
0X03	Error fatal, SROM detenida

Tabla 1.5 Significado de los códigos de retorno en la SROM

1.4.2 DESCRIPCIÓN DE LAS FUNCIONES DE LA SROM

1.4.2.1 Función SWBootReset

Esta función es la responsable de la transición del dispositivo, de un estado de reset a un estado de funcionamiento para el usuario.

La función SWBootReset es ejecutada siempre que la SROM es ingresada con un valor en el acumulador de 0x00: el parámetro en la SRAM no se usa como una entrada para la función; esto ocurrirá, por diseño, después de un reset de hardware (porque el acumulador del M8C se resetea a 0x00), o cuando el código de usuario ejecuta la instrucción SSC con un valor en el acumulador de 0x00.

1.4.2.2 Función lectura de bloque

Esta función es usada para leer un bloque de 64 bytes de la memoria Flash y almacenarlo en la SRAM. La Flash contiene 256 bloques de 64 bytes.
La principal labor es chequear los bits de protección y determinar si se puede leer el bloque designado (BLOCKID). Si la protección está encendida, la función lectura de

bloque terminará seteando el acumulador y volviendo la KEY2 al valor 00h, mientras la KEY1 tendrá un valor de 01h, indicando de esta manera que existe una falla real. Si la protección de lectura no está habilitada, la función podrá leer el contenido del bloque de la Flash usando una instrucción ROMX, luego guardará los resultados en la SRAM, empezando por la dirección POINTER, usando una instrucción MVI.

Cuando la función lectura de bloque completa de manera satisfactoria al acumulador, KEY1 y KEY2 deberán tener un valor de 00h.

Debido a que la línea de código: *MVI [expr], A* se usa para guardar el contenido del bloque de la Flash en la SRAM, el registro DPW_DR se puede setear para indicar a cuál de las páginas de la SRAM los datos deben escribirse.

Nombre	Dirección	Descripción
KEY 1	0, F8h	3Ah
KEY 2	0, F9h	Valor del stack pointer cuando la SSC es ejecutada
BLOCKID	0, Fah	Número del bloque Flash (00h-FFh)
POINTER	0, FBh	Primera de las 64 direcciones en la SRAM donde los datos retornados deben guardarse

Tabla 1.6 Parámetros de la función lectura de bloque (01h)

1.4.2.3 Función escritura de bloque

Esta función es usada para almacenar los datos en la Flash; dichos datos vienen de la SRAM (64 bytes en una vez).

La configuración de la función escritura de bloque se envía directamente. El BLOCKID de la Flash, donde los datos se guardan, debe ser determinado y almacenado en la dirección de la SRAM (FAh), previo al llamado de la función,

usando la instrucción SSC. Los valores válidos para el BLOCKID están entre 00h y FFh.

Una instrucción: MVI A, [expr] se usa para mover datos de la SRAM a la Flash.

Por consiguiente, el puntero de lectura MVI (DPR_DR) puede ser útil para especificar la página de la SRAM desde donde se extraen los datos.

Usando el MVI y el valor de la variable POINTER, se pueden mover los datos de cualquier página a algún bloque de la Flash en cualquier macro de la misma.

La dirección del primero de los 64 bytes a ser guardado en la Flash debe ser indicada usando la variable POINTER en el bloque del parámetro (SRAM, dirección FBh).

Finalmente, el valor del reloj debe ser ajustado correctamente, ya que determina la longitud del pulso de escritura que será usado para almacenar los datos en la Flash.

Cabe recalcar que el valor del reloj depende de la velocidad del CPU.

Nombre	Dirección	Descripción
KEY 1	0,F8h	3Ah
KEY 2	0,F9h	Valor del stack pointer cuando la SSC es ejecutada
BLOCKID	0,FAh	Número del bloque Flash (00h-FFh)
POINTER	0,FBh	La primera de las 64 direcciones en la SRAM donde los datos a ser guardados en la Flash están localizados antes de llamar a la escritura de bloque
CLOCK	0,FCh	Divisor del reloj usado para setear el ancho del pulso de escritura
DELAY	0,FEh	Para una velocidad del CPU de 12MHz se setea a 0x56.

Tabla 1.7 Parámetros de la función escritura de bloque (02h)

1.4.2.4 Función borrado de bloque

Está función es utilizada para programar ceros, unos, y las secuencias de borrado.

Esta función se usa únicamente en el inicio de la programación. Borrar un bloque no causará que los datos sean cien por ciento ilegibles. Si el objetivo es eliminar datos en un bloque, el mejor método es ejecutando un borrado de bloque, seguido de una escritura del mismo; llenándolo de ceros.

Para establecer el parámetro para la función borrado de bloque, deben ser almacenados valores correctos en la KEY1 y KEY2. El número de bloque a ser borrado se debe guardar en la variable BLOCKID, y el valor del reloj se fija en base a la velocidad presente en el CPU.

Nombre	Dirección	Descripción
KEY 1	0,F8h	3Ah
KEY 2	0,F9h	Valor del stack pointer cuando la SSC es ejecutada
BLOCKID	0,Fah	Número del bloque Flash (00h-FFh)
CLOCK	0,FCh	Divisor del reloj usado para setear el ancho del pulso de borrado
DELAY	0,Feh	Para una velocidad del CPU de 12MHz se setea a 0x56.

Tabla 1.8 Parámetros de la función borrado de bloque (03h)

1.4.2.5 Función checksum

Esta función calcula un checksum de 16 bits, sobre un número especifico de bloques, dentro de un macro simple de la Flash (banco), empezando desde el bloque cero.

El parámetro BLOCKID se usa para poner en el número de bloques a calcular, el checksum.

Si el valor del parámetro BLOCKID es uno, podría calcularse el checksum solo del bloque cero, y si el valor BLOCKID es cero, se calcularía el checksum de todos los 256 bloques a usar.

El checksum de 16 bits retorna en el KEY1 y KEY2. En el parámetro KEY1 van los 8 bits menos significativos, mientras que en el KEY 2 los 8 bits más significativos.

El algoritmo del checksum ejecuta la siguiente secuencia de tres instrucciones sobre el número de bloques que estarán en este proceso:

```
romx
add  [KEY 1] , A
adc  [KEY 2] , 0
```

Nombre	Dirección	Descripción
KEY 1	0,F8h	3Ah
KEY 2	0,F9h	Valor del stack pointer cuando la SSC es ejecutada
BLOCKID	0,FAh	Número de los bloques de la Flash para calcular el checksum

Tabla 1.9 Parámetros de la función checksum (07h)

1.4.2.6 Función lectura de tabla

Esta función permite al usuario el acceso a partes específicas de los datos almacenados en la Flash.

Nombre	Dirección	Descripción
KEY 1	0,F8h	3Ah
KEY 2	0,F9h	Valor del stack pointer cuando la SSC es ejecutada
BLOCKID	0,FAh	Número de fila de la tabla a leer (0--3).

Tabla 1.10 Parámetros de la función tabla de lectura

1.4.2.7 Función de calibración 0

La función de calibración 0 se usa para leer desde una parte especial de la Flash, que es restringida para el uso normal.

Nombre	Dirección	Descripción
KEY 1	0,F8h	3Ah
KEY 2	0,F9h	Valor del stack pointer cuando la SSC es ejecutada

Tabla 1.11 Parámetros de la función de calibración 0 (08h)

1.4.2.8 Función de Calibración 1

La función de calibración 1 es independiente de la función de calibración 0, pero realizan la misma función, que es la de transferir los valores almacenados de calibración en un área especial de la Flash, hacia sus registros apropiados. La diferencia de esta función, es que calcula un checksum de los datos de calibración.

La función de calibración 1 causará un reset de hardware, por seteo del bit IRES del registro CPU_SCR1.

Esta función además usa la SRAM para calcular el checksum de los datos de calibración. El valor del puntero es utilizado para indicar la dirección de un buffer de 30 bytes usado para esta función. Cuando la función se completa, los 30 bytes deberán ser seteados a 0x00.

La función de calibración1 fue creada como subfunción del SWBootReset. Sin embargo, se añade el código de calibración 1 para proveer un acceso directo.

Nombre	Dirección	Descripción
KEY 1	0,F8h	3Ah
KEY 2	0,F9h	Valor del stack pointer cuando la SSC es ejecutada
POINTER	0,FBh	El primero de los 32 direccionamientos usados para esta función

Tabla 1.12 Parámetros de la función de calibración 1 (09h)

1.5 CONTROLADOR DE INTERRUPCIÓN

El controlador de interrupción provee un mecanismo para los recursos de hardware en los dispositivos PSOC, que sirve para cambiar el programa de ejecución a una nueva dirección.

El controlador de interrupción y sus registros asociados permiten al código de usuario responder a una interrupción de casi todos los bloques funcionales del PSOC. Están disponibles interrupciones para todos los bloques digitales y para cada una de las columnas análogas, así como las interrupciones por voltaje de alimentación, sleep, relojes variables e interrupciones generales del GPIO.

Los registros asociados con el control de interrupción permiten que la interrupción sea desactivada de manera global o individual. Los registros también proveen un mecanismo por el cual el usuario puede borrar todas, o de forma individual, las interrupciones pendientes.

Mediante software se pueden también setear las interrupciones, pero en este caso solo de forma individual. Setear una interrupción mediante software es mas útil durante un desarrollo de código, más aún cuando no se cuenta con un sistema de hardware completo necesario para generar una interrupción real.

La Tabla 1.13 lista todas las interrupciones y prioridades disponibles para los dispositivos PSOC.

Prioridad de la Interrupción	Dirección de la interrupción	Nombre de la interrupción
0	0000h	Reset
1	0004h	Supervisor del voltaje de suministro
2	0008h	Columna Análoga 0 (28h)
3	000Ch	Columna Análoga 1 (2Ch)
4	0010h	Columna Análoga 2 (30h)
5	0014h	Columna Análoga 3 (34h)
6	0018h	VC3
7	001Ch	GPIO (38h)
8	0020h	Bloque del PSOC DBB00 (08h)
9	0024h	Bloque del PSOC DBB01 (0Ch)
10	0028h	Bloque del PSOC DCB02 (10h)
11	002Ch	Bloque del PSOC DCB03 (14h)
12	0030h	Bloque del PSOC DBB10 (18h)
13	0034h	Bloque del PSOC DBB11 (1Ch)
14	0038h	Bloque del PSOC DCB12 (20h)
15	003Ch	Bloque del PSOC DCB13 (24h)
24	0060h	I2C
25	0064h	Sleep Timer (3Ch)

Tabla 1.13 Lista de interrupciones

1.6 ENTRADAS / SALIDAS DE PROPÓSITO GENERAL (GPIO)

Los bloques GPIO proporcionan una interfaz entre el núcleo del M8C y el mundo exterior; estos ofrecen un gran número de configuraciones para dar soporte a varios tipos de operaciones de entrada / salida en ambos sistemas: el digital y el análogo.

El GPIO contiene buffers de entrada, drivers de salida, almacenamiento del bit de un registro y una configuración lógica para conectar el "bond pad" (bloc de enlace), hacia y desde el núcleo M8C.

Los puertos de I/O tienen un arreglo de hasta 8 bits por puerto. Cada puerto lleno contiene ocho bloques GPIO idénticos, con conexiones para identificar una dirección única y el número de bit del registro para cada bloque.

Cada bloque GPIO se puede usar por los tipos de I/O siguientes:

- I/O digitales (controladas por software).
- I/O globales.
- I/O análogas.

Cada pin de I/O tiene también varios modos de manejo (drive modes), así como capacidades de interrupción.

ξ ***I / O DIGITALES***

Una de las operaciones básicas de los puertos GPIO es permitir al M8C enviar información fuera del chip y también obtenerla desde afuera; esto es logrado a través del registro de datos del puerto PRTxDR. Se escribe desde el M8C al PRTxDR el estado de los datos, un bit por GPIO.

En el modo estándar de non-bypass, los manejadores de pines del micro (pin drives) manejan a estos en respuesta a los datos del bit. El voltaje actual en cada pin depende del modo de manejo y además de la carga externa.

El M8C puede leer el valor del puerto leyendo el registro PRTxDR. Cuando el M8C lee el PRTxDR, el valor actual de voltaje del pin es interpretado como un valor lógico

y luego retorna al M8C. Estas operaciones leen el voltaje del pin, no el estado del manejador de datos almacenado en el registro PRTxDR.

- ***I / O GLOBALES***

Los puertos GPIO se usan también para interconexión de señales para y desde los bloques digitales del PSOC, como entradas o salidas globales. La característica de las I/O globales de cada GPIO (pin del puerto) es que, por defecto, están apagadas.

Para acceder a dicha característica, se deben cambiar dos parámetros.

Para configurar un GPIO como una entrada global, se debe setear el bit seleccionado del puerto global para el GPIO deseado, usando el registro PRTxGS.

Además, el modo de manejo para el GPIO debe ser puesto al estado digital de alta impedancia (Hi-Z).

Ahora, para configurar un GPIO como salida global, se debe antes setear el bit seleccionado del puerto global. Pero en este caso, el modo de manejo se puede poner en cualquier estado, menos en el de Hi-Z.

- ***I / O ANÁLOGAS***

La señales análogas pueden transferirse desde el núcleo del chip hasta sus pines, a través del pin AOUT de bloques, el cual proporciona aproximadamente una resistencia de 300 ohms.

Para los modos análogos, el bloque GPIO típicamente se configura como modo de Hi-Z.

ξ ***INTERRUPCIONES DEL BLOQUE GPIO***

Cada bloque GPIO se puede configurar individualmente por una interrupción capacitada. Los bloques se configuran habilitando la interrupción del pin y también seleccionando el estado de la interrupción. Los bloques pueden ser seteados para interrupción cuando el pin esta en alto, bajo, o cuando este cambia desde la última vez que fue leído.

El bloque proporciona una salida de interrupción INTO (open-drain) que es conectada a otro bloque GPIO en una forma wire-OR.

Todas las interrupciones del pin que son wire-ORed son unidas para el sistema de interrupción GPIO. Sin embargo, si las interrupciones son habilitadas en múltiples pines, los usuarios de la rutina del servicio de interrupción deberán diseñar varios mecanismos para determinar cual pin fue el que generó la interrupción.

Usar una interrupción GPIO requiere de los siguientes pasos:

* Setear el modo de interrupción en el bloque del pin GPIO.
* Habilitar el bit de interrupción en el bloque GPIO.
* Setear el bit de la mascara para la interrupción global GPIO.
* Confirmar la habilitación completa de la interrupción global.

Estos dos últimos pasos son comunes para todas las interrupciones.

1.6.1 DESCRIPCIÓN DE LA ARQUITECTURA

En la Figura 1.5 se muestra el diagrama de bloques del GPIO.

Se debe notar que los modos de manejo para el GPIO se dan de acuerdo al seteo de los parámetros DM0, DM1 y DM2, los mismos que definirán como están configuradas las entradas y salidas de un determinado pin del Puerto del microcontrolador.

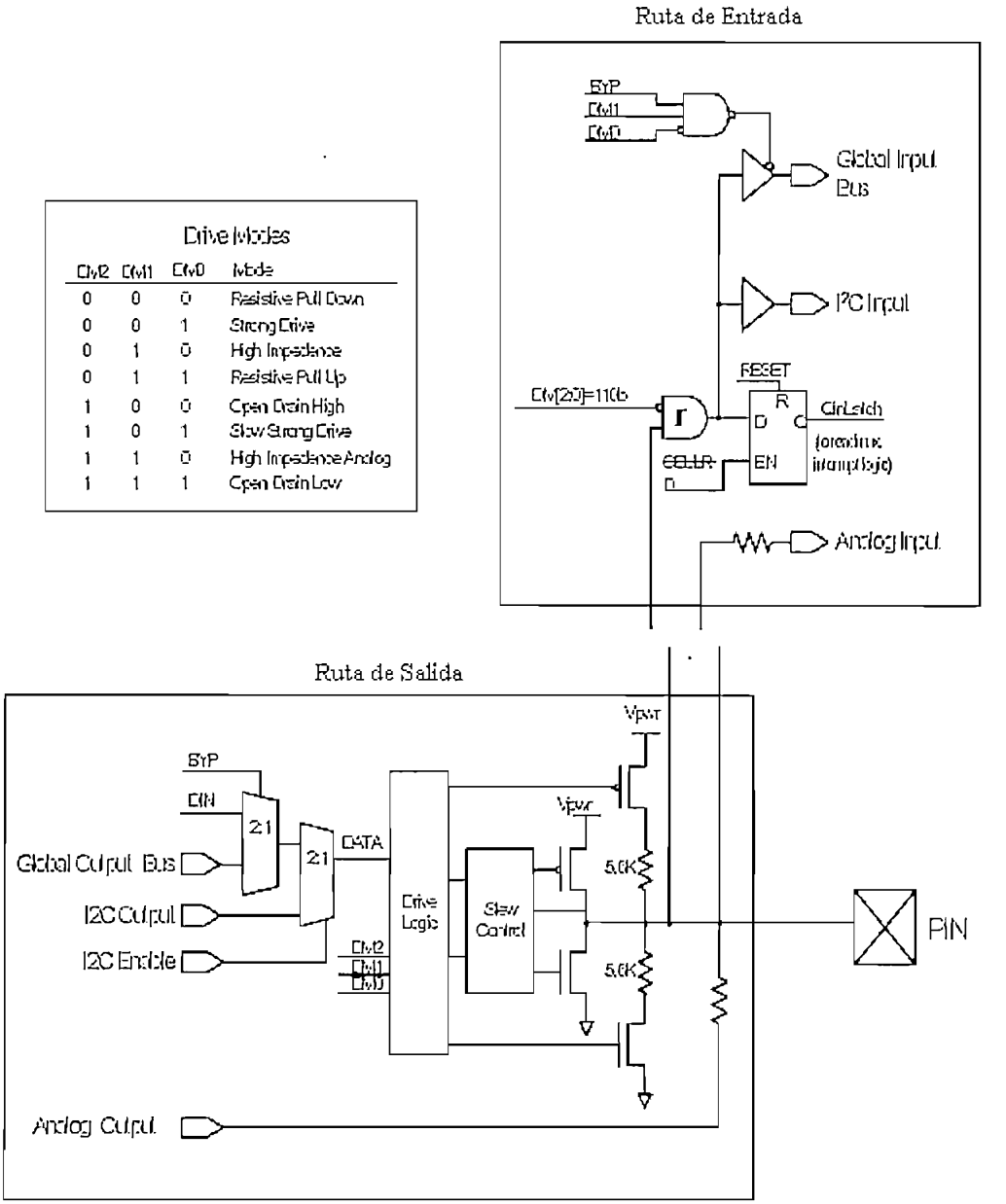


Figura 1.5 Diagrama de bloques del GPIO

A continuación, la Figura 1.6 ilustra el diagrama de bloques del modo de interrupción del GPIO.

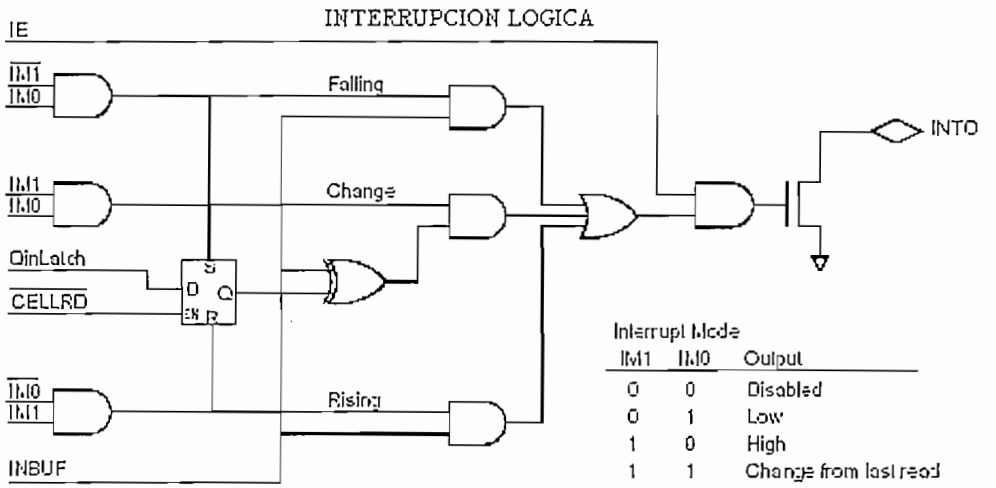


Figura 1.6 Diagrama de bloques del modo de interrupción del GPIO

En este caso los parámetros a setear para seleccionar el modo de interrupción del GPIO son IM0 e IM1.

1.7 MANEJADORES DE SALIDAS ANÁLOGAS (DRIVERS)

Los manejadores (drivers) de salidas análogas proporcionan un recurso para manejar señales análogas off-chip.

El CY8C27XXX tiene cuatro drivers análogos usados para valores de salida de este tipo en los pines del puerto. Cada uno de estos drivers es un recurso disponible para todos los bloques análogos dentro de una columna análoga particular.

El usuario deberá seleccionar un bloque análogo por columna para manejar una señal en su bus de salida análoga (ABUS); así servirá como una entrada del driver análogo para esa columna. La salida del driver de salida análoga para cada columna puede ser habilitada y deshabilitada usando el registro: manejador de salida análoga (Analog Output Driver) ABF_CR0.

La Figura 1.7 indica el diagrama de bloques de una salida análoga.

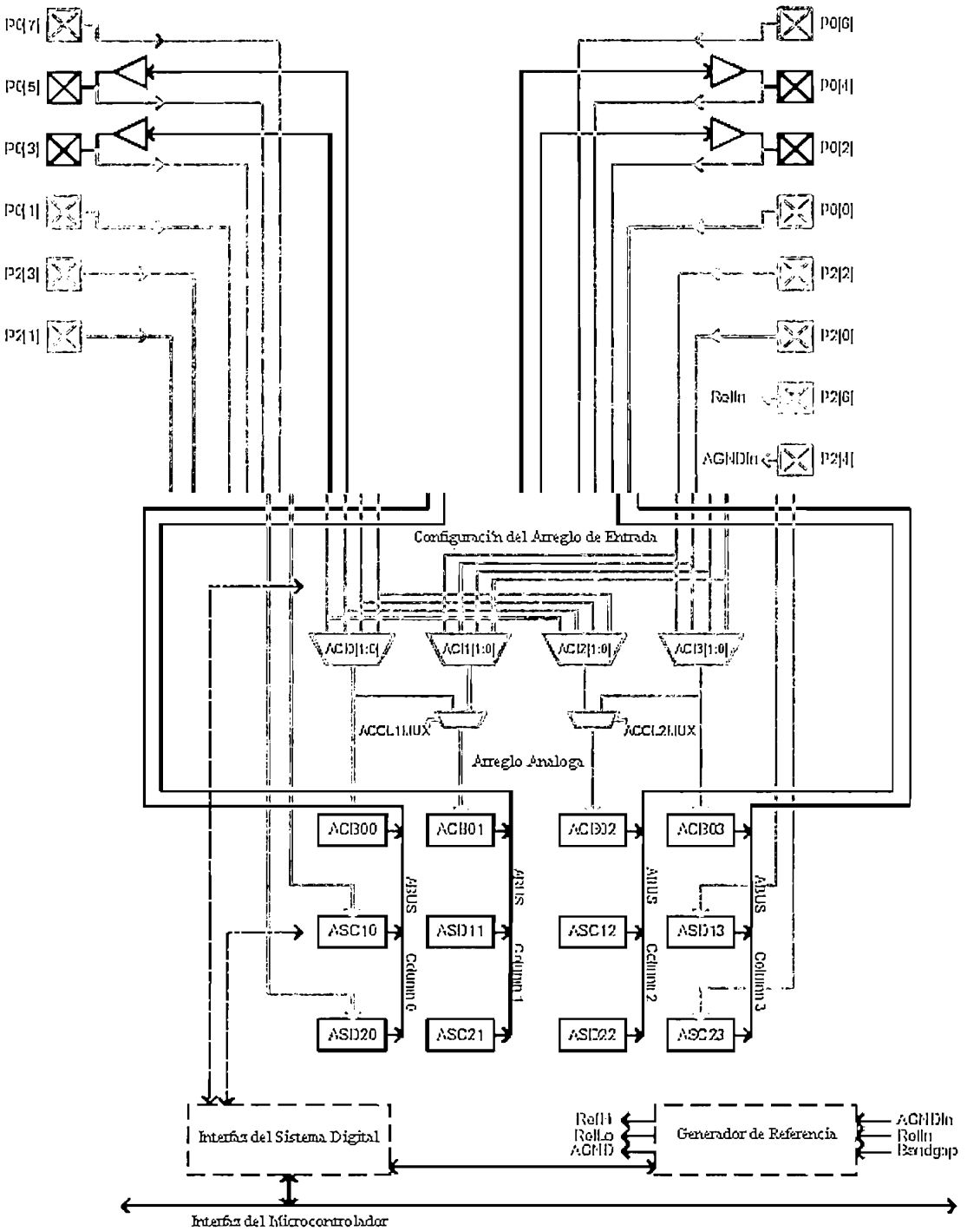


Figura 1.7 Diagrama de bloques de una salida análoga para un PSOC de 48 pines

1.8 OSCILADOR PRINCIPAL INTERNO (IMO)

El oscilador principal interno produce señales de reloj de 24 MHz y 48 MHz.

Se tiene en el IMO dos relojes de salida: un SYSCLK, el cual puede ser el reloj de 24 MHz interno o un reloj externo, y un SYSCLK2X que trabaja siempre al doble de la frecuencia del SYSCLK. En ausencia de una fuente de entrada de alta-precisión del oscilador de cristal de 32 kHz, la exactitud de los relojes internos de 24 MHz y 48 MHz podría ser de +/-2.5% sobre la variación de la temperatura y dos rangos de voltaje (3.3V +/-5% y 5.0V +/-5%).

Se requieren componentes no externos para lograr este nivel de exactitud.

La elección del cristal y su exactitud inherente determinará la exactitud global del oscilador. El oscilador de cristal externo deberá ser estable previo al aseguramiento de la frecuencia del IMO para esta fuente de referencia.

El IMO puede ser inhabilitado cuando se usa una fuente de reloj externo. Además, el circuito de doble frecuencia que produce el SYSCLK2X puede ser inhabilitado para ahorrar energía.

1.9 OSCILADOR DE BAJA VELOCIDAD INTERIOR (ILO)

Este es un oscilador de baja velocidad de 32 kHz nominal, disponible para generar interrupciones Sleep wake-up y resets del Watchdog. Este oscilador también se puede usar como una fuente de reloj para los bloques digitales del PSOC.

El oscilador opera en tres modos: energía normal, baja energía, y apagado. El de energía normal consume más corriente para producir una frecuencia más exacta, el de baja energía es siempre usado cuando se trabaja en un estado de energía inferior

(sleep) y puede seleccionarse durante el non-sleep, pero, como es de esperarse, proporciona menos exactitud de frecuencia.

1.10 OSCILADOR DE CRISTAL EXTERNO DE 32 KHZ (ECO)

El circuito del oscilador de cristal de 32 kHz le permite al usuario reemplazar el oscilador de baja velocidad interior con una fuente de tiempo más precisa a bajo costo y con consumo bajo de energía.

El circuito de este oscilador utiliza un reloj de cristal barato y dos capacitores de carga de valores pequeños como componentes externos.

El oscilador de cristal puede ser configurado para proporcionar una referencia al oscilador principal interno en modo PLL para generar un sistema de reloj más exacto de 24 MHz.

A continuación en la Figura 1.8 se observa la conexión de los componentes del oscilador de cristal externo (ECO)

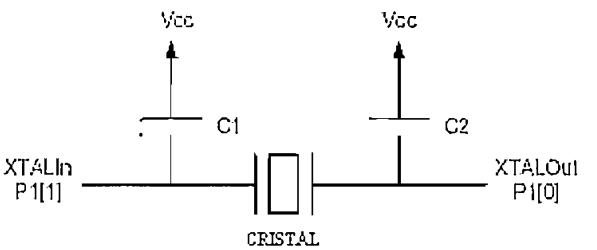


Figura 1.8 Conexiones del oscilador de cristal externo

Los pines XTALIn y XTALOut soportan la conexión de un reloj de cristal exterior de 32.768 kHz. Para correr este cristal, el bit 7 del registro 0 del control del oscilador (OSC_CR0) debe ser seteado (por defecto esta apagado). Cabe anotar que el oscilador de baja velocidad interno (ILO) continúa corriendo cuando se selecciona

esta función externa, hasta que el oscilador es automáticamente conmutado por la interrupción del sleep timer.
Los únicos componentes externos son el cristal y los dos capacitores conectados a VCC.

1.11 PHASE LOCKED LOOP (PLL)

La función PLL (lazo de bloqueo de fase) genera el reloj del sistema con exactitud del cristal. Se diseña para proporcionar un oscilador de 23.986 MHz cuando se utiliza un cristal externo de 32.768 kMz.

Aunque el PLL rastrea la exactitud del cristal, este requiere tiempo para bloquear la frecuencia de referencia. La longitud del tiempo depende del PLLGAIN controlado por el bit 7 del registro OSC_CR2. Si este bit es mantenido en bajo, el tiempo de bloqueo es menor a 10 ms; pero si este bit se lo mantiene en alto, el tiempo de bloqueo está en el orden de los 50 ms.

Después de lograr el bloqueo, se recomienda que este bit este forzado en alto para disminuir la fluctuación en la salida. Si este tiempo de bloqueo más largo es tolerable, el bit del PLLGAIN puede mantenerse en alto todo el tiempo.

Después de que el oscilador de cristal externo se ha seleccionado y habilitado, el siguiente procedimiento debería ser habilitar el PLL y permitir un bloqueo de frecuencia apropiada.

- Seleccionar una frecuencia del CPU de 3 MHz o menos.
- Habilitar el PLL.
- Esperar entre 10 y 50 ms, dependiendo del bit 7 del registro OSC_CR2.

- ξ Setear el CPU a una frecuencia más rápida, si se deseará. Para hacer esto, se escribe los bits CPU[20] en el registro USC_CPU. La frecuencia del CPU cambiará inmediatamente cuando estos bits son seteados.

1.12 EL SLEEP TIMER Y EL WATCHDOG

La meta del funcionamiento del sleep es reducir a la mitad el consumo de energía, tanto como sea posible. El sistema tiene un estado de sleep que puede ser iniciado bajo un control del firmware.

En este estado, el CPU se detiene en el límite de una instrucción, y el oscilador de 24/48 MHz, el módulo de la memoria Flash, y la referencia de voltaje band-gap son energizados de manera mínima.

Los únicos bloques que permanecen en funcionamiento son los osciladores de 32 Khz (cristal externo o interno) junto con sus respectivos bloques, y el circuito supervisor del voltaje de suministro.

Los bloques análogos del PSOC tienen seteos bajos de energía individual, los cuales son controlados por firmware, independientemente del estado del sleep.

Los bloques análogos de tiempo continuo pueden permanecer en funcionamiento, puesto que estos no requieren una fuente de reloj.

El sistema puede solo despertar del sleep como resultado de una interrupción o un evento de reset. El Sleep Timer puede proporcionar periódicas interrupciones para permitir que el sistema despierte: poll (sondeo) de los periféricos, o hacer funciones en tiempo real para luego volver al estado de sleep nuevamente.

Las interrupciones del GPIO, interrupción del supervisor de suministro de energía, interrupciones de la columna análoga, timers del reloj externo o del reloj de 32 kHz, son ejemplos de interrupciones asíncronas que también pueden usarse para despertar al sistema.

El circuito del Watch Dog Timer (WDT) está diseñado para asegurar un reset de hardware para el dispositivo después de un intervalo pre-programado, a menos que este sea abastecido periódicamente en el firmware. Este funcionamiento sirve para el reboot del sistema en caso de una caída del CPU; además puede reiniciar el sistema desde el estado de inactividad del CPU.

Una vez que el WDT se habilita, puede deshabilitarse sólo mediante un reset externo (XRES) o un power on reset (POR). Un reset WDT dejará el WDT habilitado. Sin embargo, si el WDT es usado en una aplicación, todo código (incluido el código de inicialización) debe escribirse como si el WDT estuviese habilitado.

1.13 SISTEMA DIGITAL

1.13.1 ARQUITECTURA

En la Figura 1.9 se muestra el diagrama de bloques del sistema digital.

Este sistema cuenta con dos filas digitales, que conforman el arreglo del bloque digital, Dentro de este arreglo, para cada fila, se tiene dos bloques digitales básicos seguidos de dos bloques para comunicación, que se enlazan con los puertos a través del interconector digital global.

El sistema digital, adicionalmente puede relacionarse con el sistema análogo que será estudiado posteriormente.

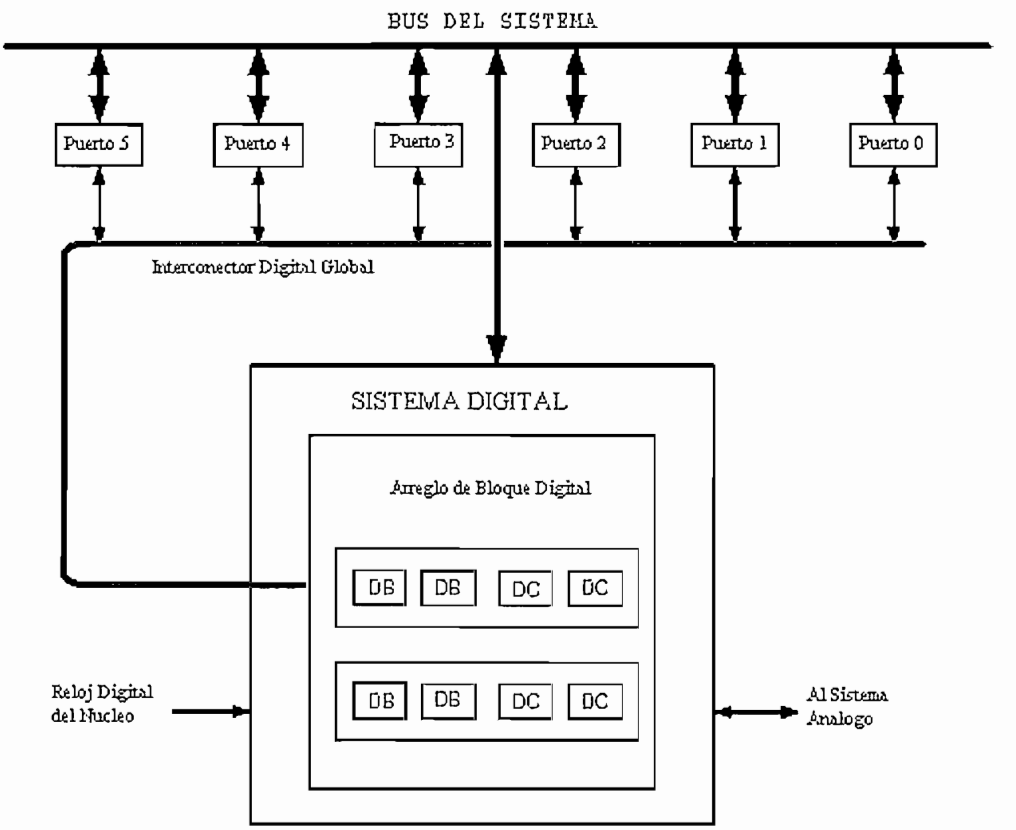


Figura 1.9 Diagrama de bloques del sistema digital para el PSOC CY8C27643.

1.14 INTERCONEXIÓN DIGITAL GLOBAL (GDI)

La Interconexión Digital Global (GDI) es el nivel más general de configuración de interconexión disponible en el PSOC.

La (GDI) consta de cuatro buses de 8 bits, dos de estos buses son de entrada, los cuales permiten que las señales pasen desde el exterior del dispositivo hacia su núcleo.

Estos buses se denominan Global Input Odd (entradas globales impares), (GIO[7:0]) y Global Input Even (entradas globales pares), (GIE[7:0]).

Los otros dos buses son de salida y permiten que las señales pasen desde el núcleo del chip hacia los pines del dispositivo. Estos buses se llaman Global Output Odd (salidas globales impares), (GOO[7:0]) y Global Output Even (salidas globales pares), (GOE[7:0]).

Las palabras "odd" (impar) o "even" (par) en el nombre del bus indican a cual de los puertos del dispositivo el bus se conecta.

Los buses con la palabra odd se conectan a todos los puertos odd y los buses con la palabra even a los puertos even. Cabe mencionar que la palabra odd o even en el nombre del bus se refiere a los puertos y no a los pines.

Hay dos extremos para la interconexión digital global: las señales del núcleo y los pines del puerto. Un extremo puede configurarse como un origen o un destino; por ejemplo: un pin del GPIO puede configurarse para manejar una entrada global o para recibir su salida desde una salida global.

Generalmente hay dos tipos de señales del núcleo conectadas a los buses globales. Los bloques digitales que pueden ser un origen o un destino de una red global, y los relojes del sistema que pueden solo manejar redes globales.

Hay hasta tres puertos conectados a los buses globales even (pares) y hasta tres puertos conectados a los buses globales odd (impares). En la Tabla 1.14, se lista el mapa entre los buses globales y los puertos.

Bus Global	Puertos
GIO(7:0), GOO(7:0)	P1, P3, P5
GIE(7:0), GOE(7:0)	P0, P2, P4

Tabla 1.14 Puertos y buses globales

Varios puertos son conectados a un solo bus global: por ejemplo, si GIO[1] se usa para traer una señal de entrada dentro de un bloque digital del PSOC, se pueden usar los pines P1[1], P3[1], o P5[1], lo mismo se realiza para las salidas:

Por ejemplo GOE[3] se usa para llevar una señal de un bloque digital a un determinado pin del puerto del PSOC, se puede usar alguno de estos pines: P0[3], P2[3], o P4[3].

1.14.1 DESCRIPCIÓN DE LA ARQUITECTURA

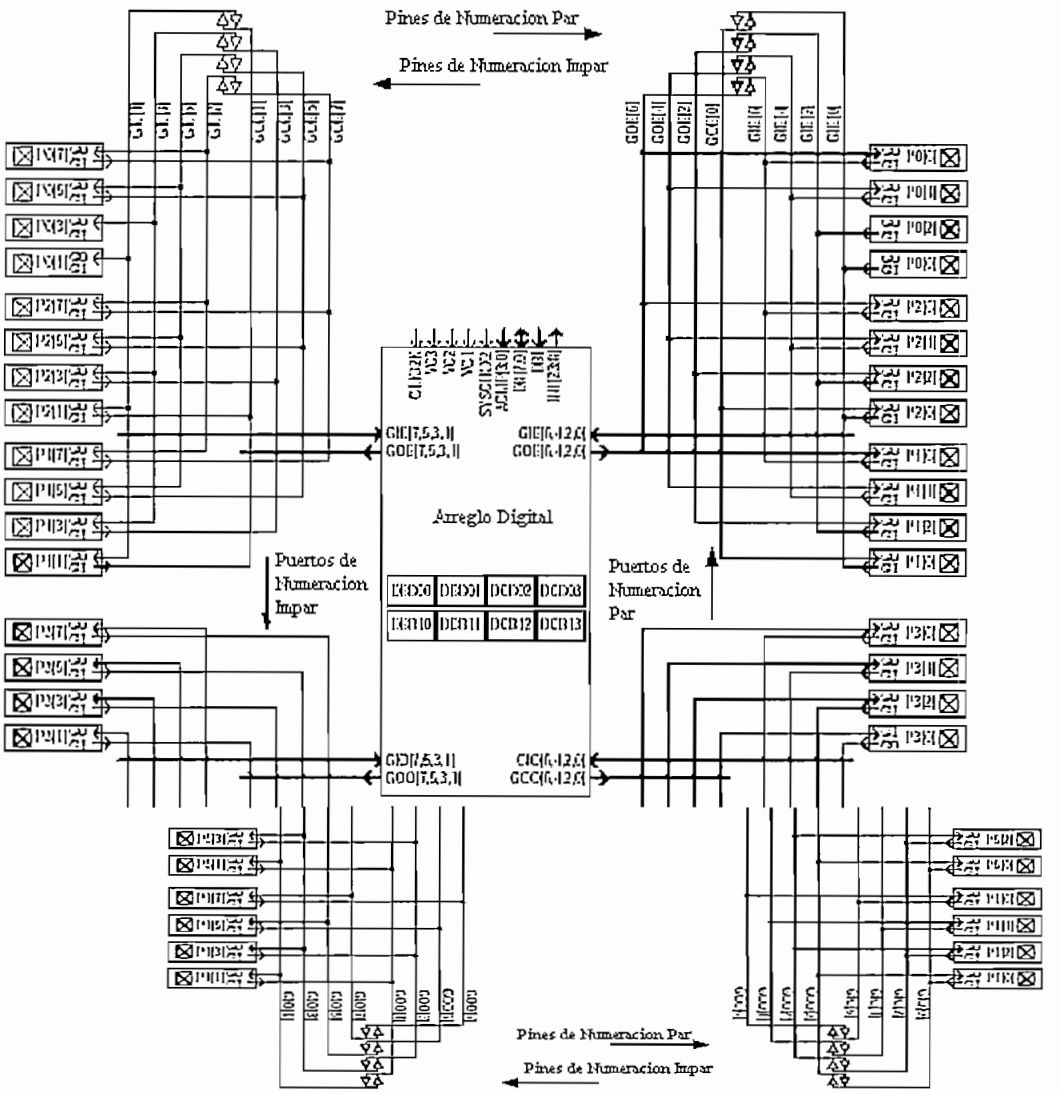


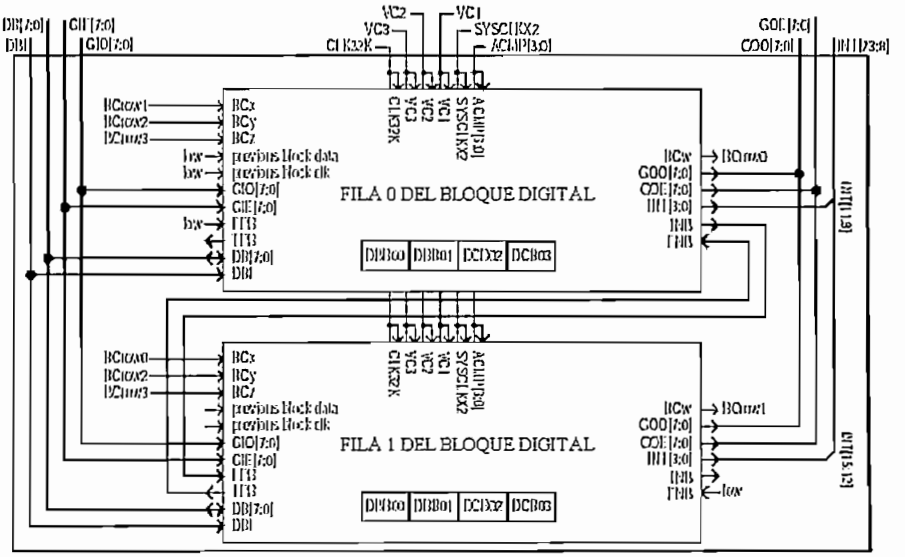
Figura1.10 Diagrama de bloques de la interconexión global

El primer objetivo que lleva a cabo la arquitectura del diagrama de bloques, es la comunicación, la relación entre los buses globales (GOE, GOO, GIE, GIO) y los pines que conforman cada puerto. Varias entradas globales pueden ser conectadas a su correspondiente salida global, usando los buffers de tres estados localizados en las esquinas de la Figura 1.10. También, las salidas globales pueden ser reducidas a entradas globales utilizando estos buffers. El rectángulo en el centro de la Figura 1.10 representa el arreglo de los bloques digitales del PSOC.

1.15 INTERCONEXIÓN DIGITAL DEL ARREGLO (ADI)

El arreglo digital del PSOC usa una arquitectura en escalera que es designada para realizar el soporte de una a cuatro filas digitales, tal como se define en la interconexión digital de filas (RDI). El arreglo no tiene ninguna interconexión configurable, por lo tanto, no tiene registros asociados.

1.15.1 DESCRIPCIÓN DE LA ARQUITECTURA



FPB= Del bloque previo TPB= Al boque anterior TNB= Al siguiente bloque FNB= Del próximo bloque

Figura 1.11 Estructura del arreglo del bloque digital

En la interconexión digital de arreglos (ADI) mostrada en la Figura 1.11, los diferentes miembros de la familia PSOC tienen varios números de bloques digitales en el arreglo digital. Estos bloques se colocan dentro de las filas y la ADI proporciona una arquitectura regular de interconexión entre el GDI y el RDI, sin tener en cuenta el número de filas disponibles en un dispositivo particular.

El aspecto más importante de la ADI y de las filas digitales es que estas últimas tienen la misma conexión a las entradas y salidas globales.

1.16 INTERCONEXIÓN DIGITAL DE FILAS (RDI)

Muchas señales pasan a través del bloque digital de filas, en su camino para o desde los bloques digitales individuales. Sin embargo, sólo un número pequeño de señales pasa a través de los circuitos configurables, en su camino para y desde los bloques digitales. Los circuitos configurables permiten una mayor flexibilidad en las conexiones entre los bloques digitales y los buses globales.

1.16.1 DESCRIPCIÓN DE LA ARQUITECTURA

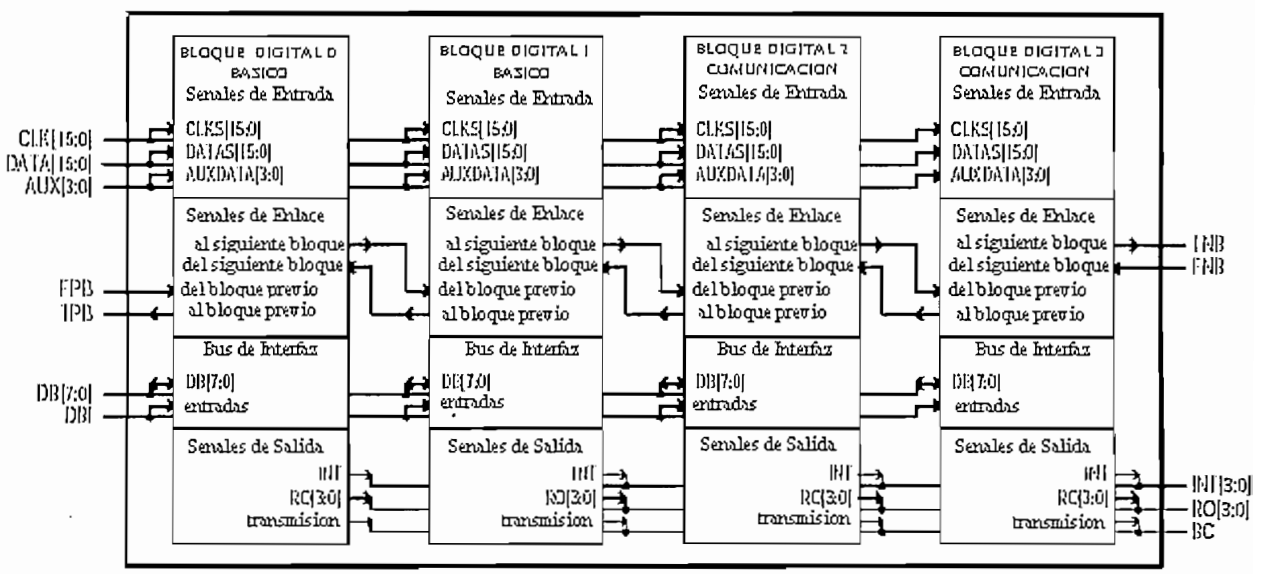


Figura 1.12 Vista detallada de la agrupación de los cuatro bloques

Existen cuatro bloques digitales. Los dos primeros bloques son de tipo básico (DBB). Los otros dos son de comunicación (DCB). La Figura 1.12 presentada anteriormente muestra las conexiones entre los bloques digitales dentro de una fila.

1.17 BLOQUES DIGITALES

Todos los bloques digitales del PSOC pueden ser configurados para realizar cualquiera de las cinco funciones básicas: timers, contadores, (PWM), secuencia de la pseudo random (PRS), o chequeo de redundancia cíclica (CRC).

Estas funciones pueden usarse para configurar un bloque individual del PSOC o para cambiar varios bloques del PSOC, también para formar funciones que son mayores a 8 bits.

El bloque de comunicaciones digitales tiene dos funciones adicionales: SPI master o esclavo y UART full duplex.

Cada función de los bloques digitales es independiente de todos los otros bloques del PSOC. Hasta siete registros son usados para determinar la función y el estado de un bloque digital.

Cada bloque digital tiene tres registros de datos (DR0, DR1, y DR2) y un registro de control (CR0). Los significados del bit para estos registros son muy dependientes de la función y se discuten con cada descripción de las mismas.

Además de los siete registros que controlan la función y el estado del bloque digital, un bit de máscara de interrupción distinto, está disponible para cada bloque del digital. Cada uno de estos bloques tienen un único vector de interrupción y por consiguiente pueden tener su propia rutina de servicio de interrupción.

1.17.1 DESCRIPCIÓN DE LA ARQUITECTURA

Los principales componentes del bloque del PSoC digital son el path de datos, multiplexores de entrada, demultiplexores de salida, buses de tres estados CRCPRS, interfaz del bus del sistema, registros de configuración y señales de enlace.

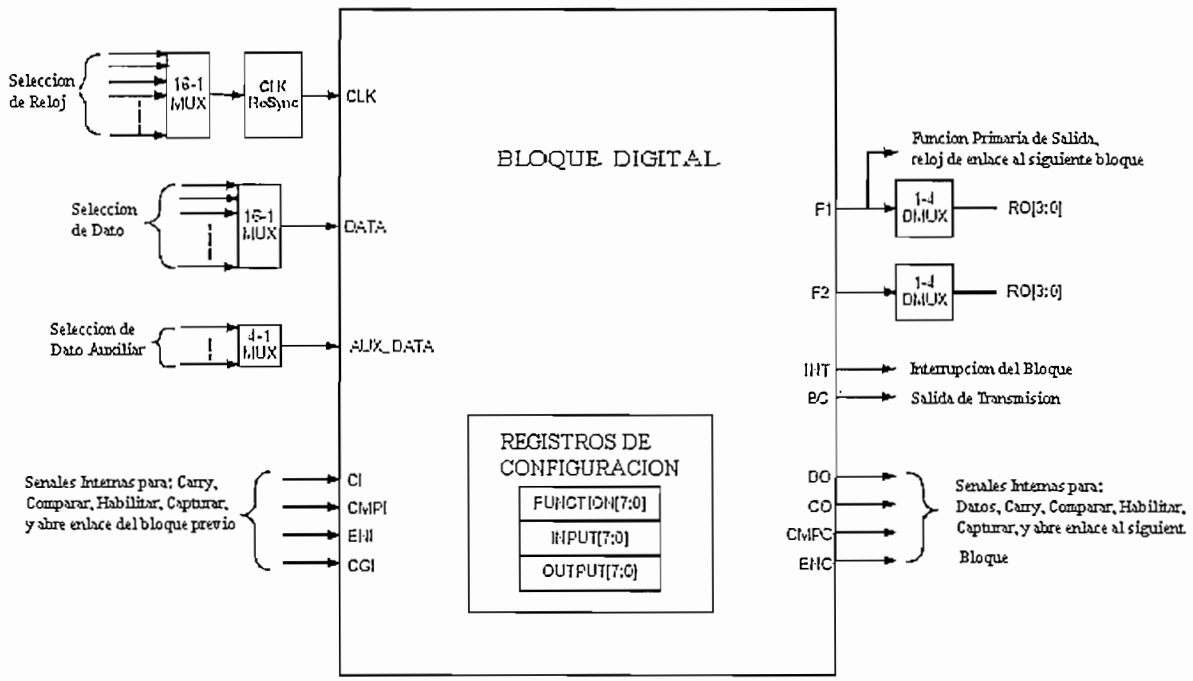


Figura1.13 Diagrama de bloques de los bloques digitales

1.17.1.1 Multiplexores de Entrada

Normalmente, cada función tiene un reloj y una entrada de datos que puede seleccionarse desde una variedad de fuentes. Cada entrada es seleccionada con un multiplexor de entrada de 16 a 1. Sin embargo, es un caso especial que existe para el CLK y para el DATA.

Adicionalmente, hay un multiplexor de 4 a 1 que proporciona una entrada auxiliar para la función esclavo del SPI que requiere tres entradas: la de reloj, la de datos y la SS_ .

La entradas para estos multiplexores se proyecta para ser una selección de las entradas del GPIO (entradas de fila).

1.17.1.2 De-Multiplexores de Salida

La mayoría de las funciones tienen dos salidas, una principal y una auxiliar, estas salidas pueden ser manejadas hacia el bus de salida de fila. Cada demultiplexor es implementado con cuatro manejadores de tres estados.

Hay dos bits para seleccionar uno de los cuatro manejadores, y un bit adicional para habilitar el que se haya seleccionado.

1.17.1.3 Bloque de Señales de Enlace

Cada bloque digital tiene la capacidad de ser enlazado, para crear funciones con una anchura de bit mayor a ocho. Existen señales para propagar la información, como: Compare, Carry, Enable, Capture y Gate, de un bloque al siguiente, para así implementar funciones de alta precisión.

La selección hecha en el registro de la función determina qué señales son apropiadas para la función deseada.

Los módulos del usuario que se han diseñado para implementar funciones digitales, con mayor anchura a ocho bits, automáticamente hacen la apropiada selección de las señales de enlace, para asegurar el flujo de información correcta entre los bloques.

1.18 SISTEMA ANÁLOGO

El PSOC esta conformado por algunos componentes análogos que se especifican a continuación.

1.18.1 ARQUITECTURA ANÁLOGA DEL PSOC

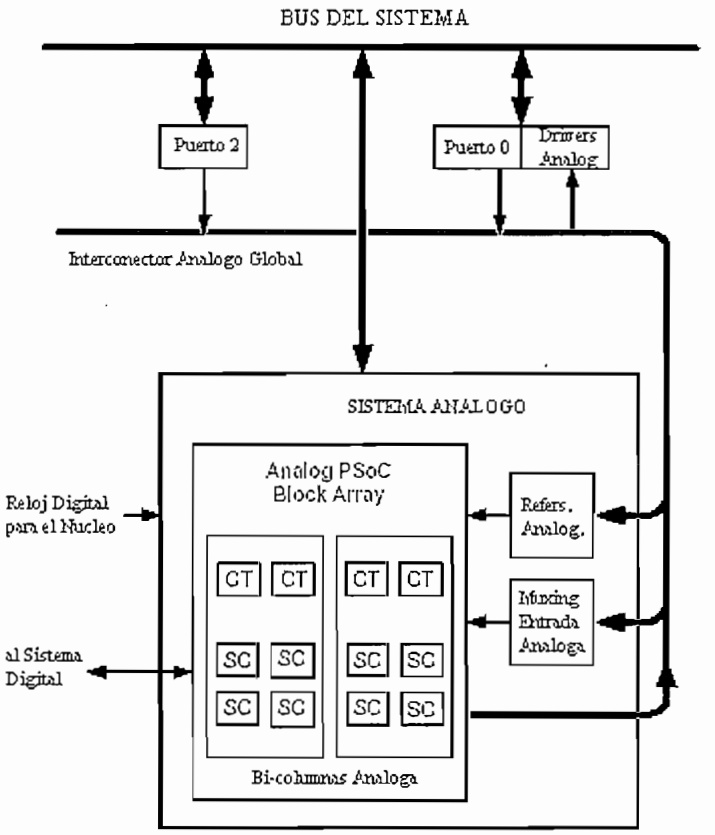


Figura 1.14 Diagrama de bloques del sistema análogo

Los bloques análogos del PSOC reducen la necesidad de muchos tipos de partes del MCU, y de componentes periféricos externos. Los bloques análogos del PSOC son configurados para proporcionar una variedad amplia de funciones periféricas.

Cada uno de los bloques análogos tienen muchas entradas posibles y varias salidas. Las entradas para estos bloques incluyen señales análogas de fuentes externas,

señales análogas intrínsecas manejadas por bloques análogos cercanos, o varias fuentes de voltaje de referencia.

Hay tres tipos de bloques análogos: bloques de tiempo continuo (CT) y bloques de capacitor conmutado (switched capacitor SC) tipo C y D. Los bloques CT proporcionan funciones análogas de tiempo continuo; mientras que los SC proporcionan funciones análogas de switched capacitor.

Los bloques análogos son organizados dentro de columnas. Hay cuatro columnas análogas en el CY8C27643, cada una de estas contiene un bloque de tiempo continuo (CT) y un Switch Capacitor (SC) tipo C y D.

Hay tres salidas desde cada bloque análogo, estas son:

- El bus de salida análoga (ABUS) es un recurso del bus análogo que es compartido por todos los bloques análogos en una columna. Solo un bloque en una columna puede manejar activamente este bus en cualquier momento y el usuario controla estas salidas a través del seteo de registros; esta es la única salida análoga que puede ser manejada por un pin externo del PSOC.

- El bus del comparador (CBUS) es un recurso del bus digital que es compartido por todos los bloques análogos en una columna. Solo un bloque en una columna puede manejar activamente este bus en cualquier momento y el usuario controla estas salidas a través del seteo de registros.

- Las salidas locales (OUT, plus GOUT, y LOUT en los bloques de tiempo continuo) son dirigidas por bloques cercanos. Las varias conexiones del multiplexor de entrada (NMux, PMux, RBotMux, AMux, BMux, and CMux), usan el bus de salida de un bloque como su entrada.

Doce bloques análogos del PSOC están disponibles, separadamente, o combinados con los bloques digitales del dispositivo. Un voltaje de referencia interno de precisión proporciona comparaciones análogas exactas.

1.19 INTERFAZ ANÁLOGA

La interfaz del sistema análogo es una colección de interfaces de nivel del sistema, que se usa para el arreglo análogo y el bloque de referencia análogo.

1.19.1 DESCRIPCIÓN DE LA ARQUITECTURA

La arquitectura de esta interfaz es la misma mostrada en la Figura 1.7.

1.19.1.1 Interfaz del Bus de Datos Análogos

Esta interfaz aísla el arreglo análogo y los registros de interfaz del sistema análogo del bus de datos del sistema CPU para reducir la carga del mismo.

1.19.1.2 Interfaz del Bus Comparador Análogo

Cada columna análoga tiene un bus comparador asociado con esta. Todo bloque análogo del PSOC tiene una salida de comparación que puede manejar este bus.

Sin embargo, solo un bloque análogo en una columna puede activamente manejar el bus comparador para una columna en cualquier momento. La salida en el bus comparador puede manejarse dentro de los bloques digitales como una entrada de datos.

También sirve como una entrada de interrupción y está disponible como datos de solo lectura en el registro de control del comparador análogo (CMP_CR0, Address=Bank 0,64H).

La Figura 1.15 ilustra una columna del bus comparador. En el Continuous Time (bloque de tiempo contínuo) (CT) los bits CPhase y CLatch del registro 2 del control de bloque CT, determinan si la señal de salida en el bus comparador es almacenada temporalmente (latchada) dentro del bloque, y si es así, qué fase del reloj es latchada. En el Switched Capacitor (SC) la salida en el bus comparador es siempre latchada.

El bit ClockPhase en el registro 0 del control de bloque SC determina la fase en la que este dato es latchado y está disponible.

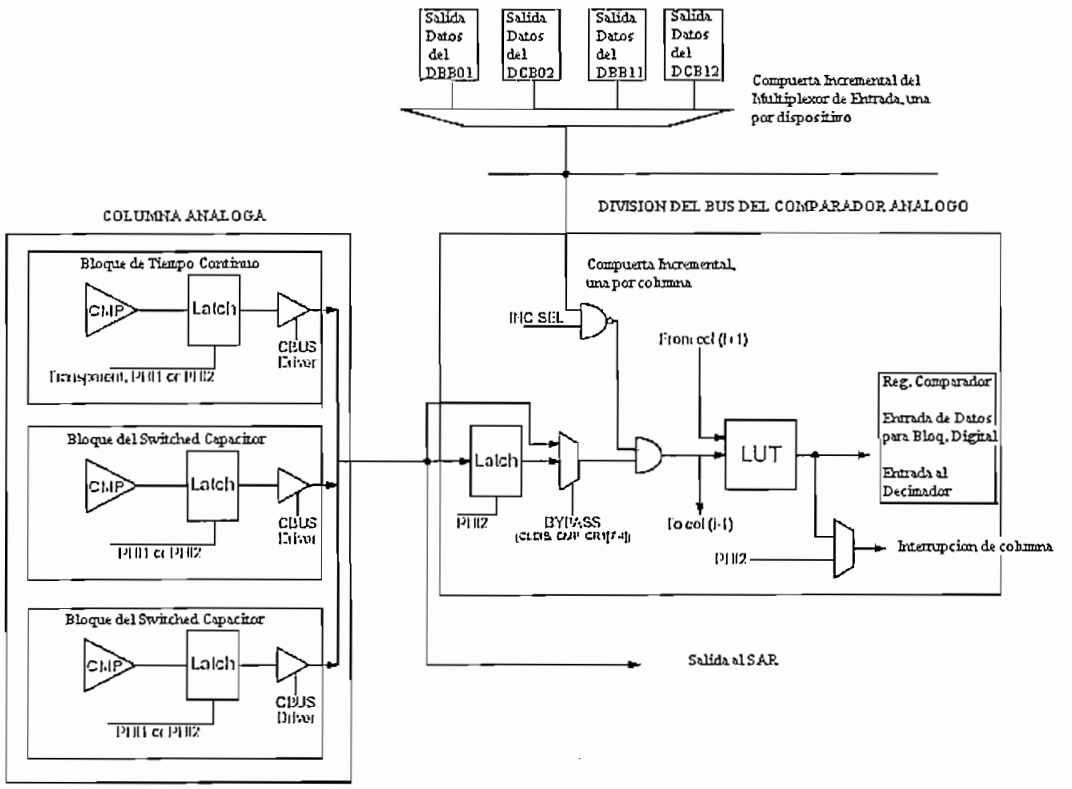


Figura 1.15 Una parte del bus comparador análogo

El bus comparador es latchado antes de que esté disponible, o maneje los bloques digitales, interrupción, etc; es decir, leído en el registro CMP_CR0.

El latch para cada bus comparador es transparente durante el periodo en alto del PHI2. Durante el período en bajo del PHI2, el latch retiene el valor en el bus comparador durante la transición de alto a bajo del PHI2.

1.20 ARREGLO ANÁLOGO

Los bloques análogos pueden usarse para implementar un amplio rango de funciones, sólo limitado por la imaginación del diseñador. Las siguientes funciones operan dentro de la capacidad de los bloques análogos del PSOC, usando uno o múltiples bloques análogos, una combinación de más de un tipo de bloque análogo, o una combinación de bloques análogos y digitales.

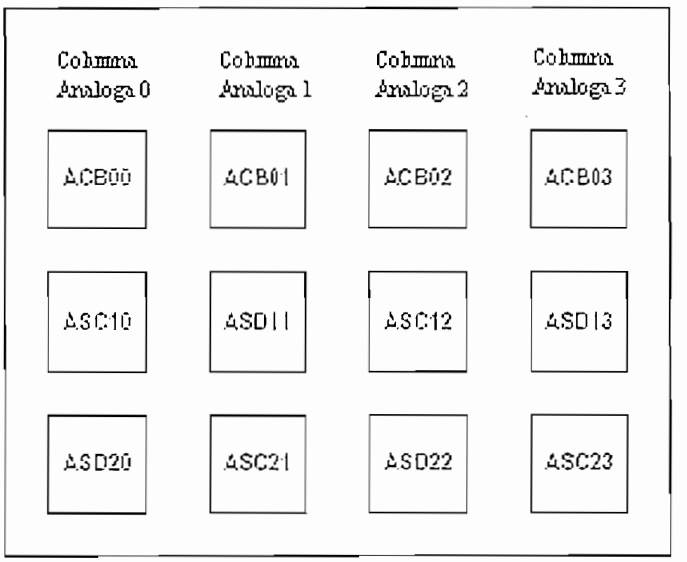


Figura 1.16 Arreglo de los bloques análogos del PSOC CY8C27XXX

1.21 CONFIGURACIÓN DE LAS ENTRADAS ANÁLOGAS

El multiplexor de entrada une las entradas del dispositivo a las columnas del arreglo análogo, basado en los valores del bit en los registros AMX_IN y ABF_CR0.

Las columnas del borde son alimentadas a través de un mux 4:1; las columnas internas son alimentadas por 1 de 2 muxes 4:1. Los muxes son switches CMOS con resistencias dentro del rango de 2Kohms.

1.22 REFERENCIA ANÁLOGA

El generador de referencia, establece un seteo de tres voltajes de referencia preparados internamente para AGND, RefHi, and RefLo.

1.22.1 DESCRIPCIÓN DE LA ARQUITECTURA

El dispositivo PSOC es una sola parte del abastecedor con voltaje no negativo disponible o aplicable. La tierra análoga (AGND) se construye cerca de la mitad del abastecedor. Esta tierra se dirige a todos los bloques analógicos y separadamente es almacenada dentro de cada bloque.

Es importante notar que puede haber un pequeño voltaje offset entre las tierras análogas almacenadas. Las señales RefHi y RefLo son generadas, almacenadas y dirigidas hacia los bloques analógicos. RefHi y RefLo se usan para setear los rangos de conversión del ADC y del DAC. RefHi y RefLo pueden también ser usados para setear los thresholds (umbrales) en los comparadores.

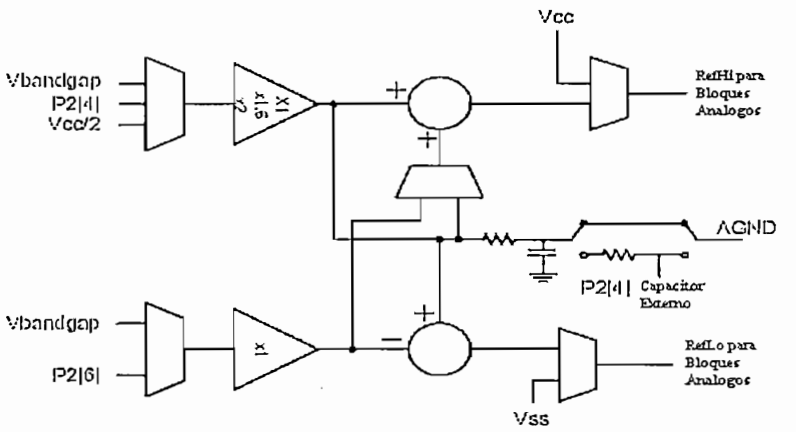


Figura 1.17 Esquemático del control de referencia análogo

La Figura 1.17 presentada anteriormente ilustra el esquemático del control de referencia análogo.

Además:

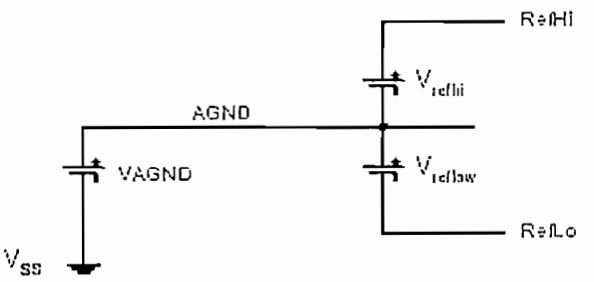


Figura 1.18 Estructura de referencia

Las tablas de registros de los temas tratados en esta sección, se encuentran en el ANEXO "A"

En el siguiente capítulo, se estudiará lo referente a los lenguajes de programación utilizados para el PSOC CY8C27643.

CAPÍTULO 2

LENGUAJES DE PROGRAMACIÓN PARA EL PSOC CY8C27643.

CAPÍTULO 2

LENGUAJES DE PROGRAMACIÓN PARA EL PSOC CY8C27643.

En el presente capítulo, se estudia los lenguajes de programación utilizados en la realización de un determinado firmware para el PSOC CY8C27643. Cabe indicar que estos lenguajes se desarrollan, a través del programa computacional PSOC Designer.

2.1 LENGUAJE ENSAMBLADOR DEL PSOC DESIGNER

Para iniciar este estudio, se debe tomar en cuenta la notación ilustrada en la Tabla 2,1, la cual es aplicable a lo largo de esta explicación para el PSOC Designer.

Notación	Descripción
A	Acumulador
CF	Bandera del Carry
expr	Expresión
F	Banderas (ZF, CF y otras)
K	Valor del operando 1
K_1	Primero de los 2 operandos
K_2	Segundo de los 2 operandos
PC	Contador de Programa
SP	Stack Pointer
X	Registro X
ZF	Bandera Cero
REG	Espacio de registro

Tabla 2.1 Registros internos

2.2 EL MICROPROCESADOR M8C

2.2.1 INTRODUCCIÓN

En el capítulo anterior se analizó lo que representa el M8C en la arquitectura del núcleo de un PSOC.

Ahora se estudiará los registros internos con los que cuenta:

2.2.2 REGISTROS INTERNOS

El M8C tienen cinco registros internos que se usan en la ejecución del programa:

Accumulator (A).
Index (X).
Program Counter (PC).
Stack Pointer (SP).
Flags (F).

Todos los registros internos del M8C son de 8 bits excepto el PC que es de 16 bits.

Cuando se realiza un reset: A, X, PC y SP se resetean a 0x00. El registro de bandera (F) es reseteado a 0x02 indicando que la bandera de Z está seteada.

Con cada operación del stack (pila) el SP es automáticamente incrementado o decrementado, para que siempre apunte al próximo byte del stack en la RAM. Si el último byte del stack está en la dirección 0xFF en la RAM, el SP apuntará a la dirección 0x00. Es responsabilidad del diseñador del firmware asegurar que el stack no se superponga con las variables definidas por el usuario en la RAM.

2.2.3 ESPACIOS DE DIRECCIÓN

El microcontrolador M8C tiene tres espacios de dirección: ROM, RAM y de registros.

Al espacio de dirección de la ROM se accede a través de su propia dirección y bus de datos; este está compuesto por la ROM de supervisión (SROM) y el almacenador del programa Flash dentro del chip. La Flash está organizada en bloques de 64-bytes.

La necesidad del usuario no debe ser comprometida con los límites de la página de almacenamiento del programa, puesto que los M8C incrementan automáticamente el PC de 16 bits en cada instrucción, haciendo los límites del bloque invisibles para el código de usuario. Instrucciones que ocurren en un límite de página Flash de 256 bytes (con la excepción de las instrucciones de salto), incurren en un ciclo extra de reloj del M8C, mientras el byte superior del PC se incrementa.

El espacio de dirección que corresponde a los registros, se usa para configurar los bloques programables del PSOC. Este consiste en dos bancos de 256 bytes cada uno.

Para cambiar entre los bancos, el bit XIO en el registro de bandera es seteado o borrado: (seteado para banco 1, borrado para banco 0). El convenio común es dejar habilitado el banco 0 (XIO borrado).

La RAM está dividida en páginas de 256-bytes. Para PSOCs con 256 bytes de RAM o menos, el stack del programa se guarda en la página 0 de la RAM. Para PSOCs con 512 bytes de RAM o más, el stack se restringe a una sola página de la RAM.

Dentro de los espacios de dirección que se ilustran en la Figura 2.1, la cual muestra el arreglo de los espacios de dirección del microcontrolador PSOC, se debe tomar en cuenta la siguiente nomenclatura:

M: Número total de bloques de la Flash en el dispositivo

XIO: Selección de banco del registro

IOR: Lectura del registro

IOW: Escritura del registro

MR: Lectura de memoria

MW: Escritura de memoria

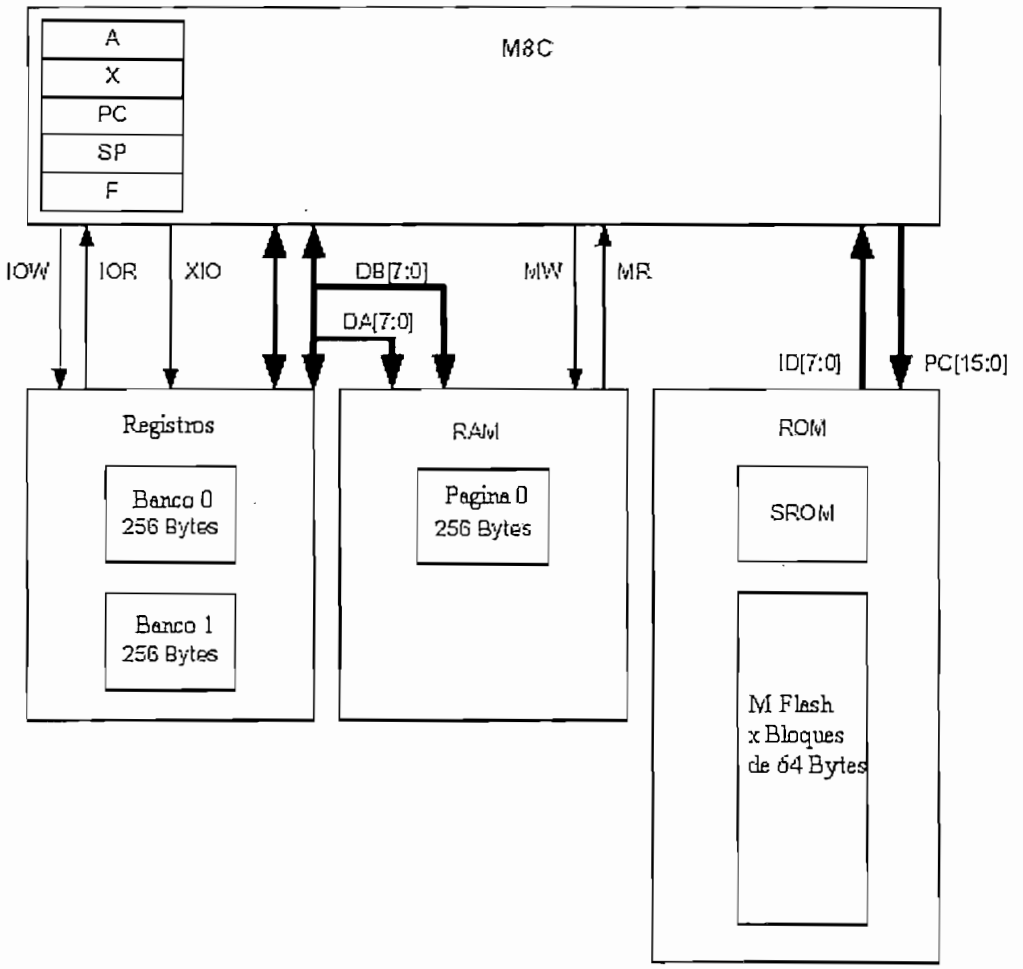


Figura 2.1 Espacios de dirección del microcontrolador M8C

2.2.4 FORMATO DE LAS INSTRUCCIONES:

El M8C tiene un total de siete formatos de instrucciones que usan longitudes de uno, dos, y tres bytes.

Todos los bytes de la instrucción se sacan de la memoria de programa (Flash), usando una dirección y el bus de datos, los cuales son independientes de la dirección y buses de datos usados por los registros y accesos de la RAM.

2.2.4.1 Instrucciones de 1 byte

Algunas de las instrucciones MOV, tienen la forma de un solo byte porque estas no usan una dirección o datos como un operando.

Como se muestra en la Tabla 2.3, las instrucciones de un byte usan un opcode de 8 bits. El set de instrucciones de un byte puede ser dividido en cuatro categorías según donde sus resultados sean guardados.

Byte 0
Opcode de 8 bits

Tabla 2.3 Formato de las instrucciones de 1 byte

La primera categoría de instrucciones de un byte, comprende aquellas que no actualizan ninguno de los registros o la RAM. Sólo las instrucciones NOP y SSC encajan en esta categoría.

Mientras el contador de programa PC (Program Counter) se incrementa cuando estas instrucciones se ejecutan, las mismas no pueden causar que ningún registro interno del M8C sea actualizado, ni que estas instrucciones afecten directamente el espacio de los registros o el espacio de dirección de la RAM.

La instrucción SSC causará la ejecución de un código SROM que modificará la RAM y los registros internos del M8C.

La segunda categoría tiene solamente las dos instrucciones PUSH y esto se da porque son las únicas instrucciones de un byte que causan que una dirección de la RAM sea modificada. Estas instrucciones incrementan automáticamente el SP.

La tercera categoría contiene sólo la instrucción HALT, esto se da porque es la única instrucción de un solo byte que causa que un registro de usuario sea modificado. La instrucción HALT modifica el registro de usuario al espacio de dirección 0xFF (CPU_SCR).

La categoría final para instrucciones de un solo byte comprende las instrucciones que causan que los registros internos del M8C sean actualizados. Esta categoría comprende el número más grande de instrucciones: ASL, ASR, CPL, DEC, INC, MOV, POP, RET, RETI, RLC, ROMX, RRC, SWAP. Estas instrucciones pueden causar que el A, el X y los registros SP o SRAM sean actualizados.

2.2.4.2 Instrucciones de 2 bytes

La mayoría de las instrucciones del M8C son de dos bytes en longitud. Mientras estas instrucciones puedan ser divididas en categorías, de manera idéntica como en las instrucciones de un byte, no proporcionan una distinción útil entre los tres formatos de instrucciones de dos bytes que el M8C usa.

Byte 0	Byte 1
Opcode de 4 bits	Dirección relativa de 12 bits
Opcode de 8 bits	Datos de 8 bits
Opcode de 8 bits	Direcciones de 8 bits

Tabla 2.4 Formatos de las instrucciones de 2 bytes

El primer formato de instrucciones de dos bytes se muestra en la Tabla 2.4, es usado por saltos cortos llamados: CALL, JMP, JACC, INDEX, JC, JNC, JNZ, JZ. Este formato de instrucción usa solo 4 bits para el opcode de la instrucción, dejando 12 bits para guardar la dirección de destinación relativa en una forma de complemento.

Estas instrucciones pueden cambiar la ejecución del programa para una dirección relativa, a la dirección actual por- 2048 ó +2047.

El segundo formato de instrucciones de dos bytes (Tabla 2.4) es usado por instrucciones que emplean el modo de direccionamiento desde una fuente inmediata.

El destino para estas instrucciones es un registro interior del M8C, mientras la fuente es un valor constante. Un ejemplo de este tipo de instrucción podría ser ADD A, 7.

El tercer formato de instrucciones de dos bytes es usado por una amplia gama de instrucciones y modos de direccionamiento. La siguiente es una lista de modos de direccionamiento que usan este tercer formato de instrucción de dos bytes:

Fuente Directa (ADD A, [7])
Fuente Indexada (ADD A, [X+7])
Destino Directo (ADD [7], A)
Destino Indexado(ADD [X+7], A)
Incremento de la posición de la fuente indirecta (MVI A, [7])
Incremento de la posición del destino indirecto (MVI [7], A)

2.2.4.3 Instrucciones de 3 bytes

Los formatos de instrucciones de tres bytes son el segundo de los formatos de instrucciones más importantes.

Estas instrucciones necesitan tres bytes porque estas: o mueven datos entre dos direcciones en el espacio de dirección accesible por el usuario (registros y RAM) o contienen direcciones absolutas de 16 bits como el destino de un salto largo o una llamada larga.

Byte 0	Byte 1	Byte 2
Opcode de 8 bits	Direcciones de 16 bits (MSB, LSB)	
Opcode de 8 bits	Direcciones de 8 bits	Datos de 8 bits
Opcode de 8 bits	Direcciones de 8 bits	Direcciones de 8 bits

Tabla 2.5 Formatos de las instrucciones de 3 bytes

El primer formato de instrucción mostrado en la Tabla 2.5 es usado por las instrucciones LJMP y LCALL. Estas instrucciones cambian incondicionalmente la ejecución del programa a una dirección absoluta. Las instrucciones usan un opcode de 8 bits que dejan espacio para una dirección de destino de 16 bits.

El segundo formato de instrucciones de tres bytes (Tabla 2.5) es usado por los dos siguientes modos de direccionamiento:

Destino directo desde una fuente inmediata (ADD [7], 5)
Destino indexado desde una fuente inmediata (ADD [X+7], 5).

El tercer formato de instrucciones de tres bytes es para el modo de direccionamiento de destino directo desde la fuente directa, que es usado por solo una instrucción.

Este formato de instrucción usa un opcode de 8 bits seguido por dos direcciones de 8 bits.

La primera dirección, es la dirección de destino en la RAM mientras la segunda, es la dirección de la fuente. El siguiente es un ejemplo de esta instrucción: MOV [7], [5].

2.2.5 MODOS DE DIRECCIONAMIENTO:

El M8C tiene diez modos de direccionamiento:

Fuente Inmediata.
Fuente Directa.
Fuente Indexada.
Destino Directo.
Destino Indexado.
Destino directo desde una fuente inmediata.
Destino indexado desde una fuente inmediata.
Destino directo desde una fuente directa.
Incremento de la posición de la fuente indirecta.
Incremento de la posición del destino indirecto.

2.2.5.1 Fuente inmediata

Para estas instrucciones el valor de la fuente se guarda en el operando 1 de la instrucción. El resultado se pone en cualquiera de los registros del M8C: A, F, o X indicado en el opcode de la instrucción. Todas las instrucciones que usan el modo de direccionamiento de fuente inmediata son de dos bytes en longitud.

Opcode	Operando 1
Instrucción	Valor inmediato

Tabla 2.6 Fuente inmediata

2.2.5.2 Fuente directa

Para estas instrucciones la dirección de la fuente se guarda en el operando 1 de la instrucción. Durante la ejecución de la instrucción, la dirección se usará para

recuperar el valor de la fuente de la RAM o del espacio de dirección del registro. El resultado de estas instrucciones se pone en cualquiera de los registros A o X, indicado por el opcode de la instrucción. Todas las instrucciones que usan el modo de direccionamiento de fuente directa son de dos bytes en longitud.

Opcode	Operando 1
Instrucción	Dirección de origen (fuente)

Tabla 2.7 Fuente directa

2.2.5.3 Fuente indexada

Para estas instrucciones, el "offset" de la fuente del registro X se guarda en el operando 1 de la instrucción. Durante la ejecución de la instrucción el valor actual del registro X es sumado al offset señalado, para determinar la dirección del valor de la fuente en la RAM o espacio de dirección del registro. El resultado de estas instrucciones se pone en cualquiera de los registros A o X del M8C indicado por el opcode de la instrucción. Todas las instrucciones que usan este modo de direccionamiento son de dos bytes en longitud.

Opcode	Operando 1
Instrucción	Fuente indexada

Tabla 2.8 Fuente indexada

2.2.5.4 Destino directo

Para estas instrucciones, la dirección del destino se guarda en el código de máquina de la instrucción. La fuente para la operación es cualquiera de los registros A o X del M8C indicado por el opcode de la instrucción. Todas las instrucciones que usan este modo de direccionamiento son de dos bytes en longitud.

Opcode	Operando 1
Instrucción	Dirección de destino

Tabla 2.9 Destino directo

2.2.5.5 Destino Indexado

Para estas instrucciones, el offset para llegar al destino desde el registro X se guarda en el código de máquina para la instrucción. La fuente para la operación es cualquiera de los registros A o X, ó un valor inmediato indicado por el opcode de la instrucción.

Todas las instrucciones que usan este modo de direccionamiento son de dos bytes en longitud.

Opcode	Operando 1
Instrucción	Destino indexado

Tabla 2.10 Destino indexado

2.2.5.6 Destino directo desde una fuente inmediata

Para estas instrucciones, la dirección del destino se guarda en el operando 1 de la instrucción. El valor de la fuente se guarda en el operando 2. Todas las instrucciones que usan este modo de direccionamiento son de tres bytes en longitud.

Opcode	Operando 1	Operando 2
Instrucción	Dirección de destino	Valor inmediato

Tabla 2.11 Destino directo desde una fuente inmediata

2.2.5.7 Destino indexado desde una fuente inmediata

Para estas instrucciones el offset para llegar al destino desde el registro X se guarda en el operando 1 de la instrucción. El valor de la fuente se guarda en el operando 2.

Todas las instrucciones que usan este direccionamiento son de tres bytes en longitud.

Opcode	Operando 1	Operando 2
Instrucción	Destino indexado	Valor inmediato

Tabla 2.12 Destino indexado desde una fuente inmediata

2.2.5.8 Destino directo desde una fuente directa

Sólo una instrucción usa este modo de direccionamiento. La dirección del destino se guarda en el operando 1 de la instrucción. La dirección de la fuente se guarda en el operando 2.

La instrucción que usa este direccionamiento es de tres bytes en longitud.

Opcode	Operando 1	Operando 2
Instrucción	Dirección de destino	Dirección de origen (fuente)

Tabla 2.13 Destino directo desde una fuente directa

2.2.5.9 Incremento de la posición de la fuente indirecta

Sólo una instrucción usa este modo de direccionamiento. La dirección de la fuente guardada en el operando 1 realmente es la dirección de un puntero (indicador).

Durante la ejecución de la instrucción, el valor actual del puntero se lee para determinar la dirección de la RAM donde el valor de la fuente estaría presente. El valor del puntero se incrementa después de que el valor de la fuente es leído.

Para microcontroladores PSOC con más de 256 bytes de RAM, el registro leer página de datos (DPR_DR) sirve para determinar qué página de la RAM se usa con la dirección de la fuente.

Por eso, valores de páginas, incluso la página actual, puede ser recuperada sin cambiar el puntero de página actual (CPP_DR). El puntero siempre se lee de la página de la RAM actual.

Opcode	Operando 1
Instrucción	Puntero de la dirección de origen (fuente)

Tabla 2.14 Incremento del lugar de la fuente indirecta

2.2.5.10 Incremento de la posición del destino indirecto

Solo una instrucción utiliza este modo de direccionamiento. La dirección del destino guardada en el operando 1 es realmente la dirección de un puntero.

Durante la ejecución de la instrucción el valor actual del puntero es leído para determinar la dirección del destino de la RAM donde se guardará el valor del acumulador. Se incrementa el valor del puntero después de que el valor se escribe en la dirección del destino.

Para PSOCs con más de 256 bytes de RAM, el registro escribir en página de datos (DPW_DR) se usa para determinar que página de la RAM usar con la dirección del destino.

Por eso, pueden guardarse valores en páginas cualquiera, incluso en la página actual sin cambiar el Page Pointer (puntero o indicador de página) actual (CPP_DR). El puntero siempre se lee de la página de la RAM actual.

Opcode	Operando 1
Instrucción	Puntero de la dirección de destino

Tabla 2.15 Incremento del lugar del destino indirecto

2.3 ASSEMBLER DEL PSOC DESIGNER

2.3.1 FORMATO DEL ARCHIVO FUENTE

Sobre los archivos fuente para el ensamblador del PSOC Designer, este tiene cinco componentes básicos, los cuales se listan en la Tabla 2.16. Cada línea del archivo fuente puede contener solamente una etiqueta, un mnemónico, y un comentario o director. Múltiples operandos o expresiones se pueden utilizar en una sola línea del archivo fuente, la máxima longitud para una línea es 2.048 caracteres (incluido espacios) y la máxima longitud para una palabra es de 256 caracteres. Una palabra es un string de caracteres rodeado por espacios.

Componente	Descripción
Etiqueta	Nombre simbólico seguido por un " : "
Mnemónico	String de caracteres representando una instrucción
Operando	Argumentos para las instrucciones
Comentario	Pueda seguir a operandos o expresiones; comienza en cualquier columna si el primer caracter que lo precede es un " // " o " ; "
Director	Un comando interpretado por el ensamblador para control de generación del código de máquina

Tabla 2.16 Cinco componentes básicos de un archivo de fuente ensamblador

2.3.1.3 Operandos

Los operandos son los argumentos para las instrucciones. El número de operandos y el formato que usan es definido por la instrucción a ser usada. Los operandos pueden tomar la forma de constantes, etiquetas, operadores de punto, registros, RAM, o expresiones.

2.3.1.3.1 Constantes: Son valores de apoyo de los operandos, explícitamente declarados en el archivo fuente. Las constantes pueden ser declaradas en el archivo fuente usando uno de los radixes listados en la Tabla 2.17.

Radix	Nombre	Formatos
127	Caracter ASCII	'J'
16	Hexadecimal	0X4A 4Ah $4ª
10	Decimal	74
8	Octal	0112
2	Binario	0b01001010 %01001010

Tabla 2.17 Formatos de las constantes

2.3.1.3.2 Etiquetas: Como se describió anteriormente pueden ser usadas como un operando para una instrucción.

Las etiquetas son muy a menudo usadas como operandos por las instrucciones de salto y de llamado, para especificar la dirección del destino. Sin embargo, pueden usarse etiquetas como un argumento para cualquier instrucción.

2.3.1.3.3 Operador Punto (.): Se usa para indicar que la dirección de la ROM del primer byte de la instrucción debe ser usada como un argumento para la instrucción.

2.3.1.3.4 Registros: Son de dos tipos: El primer tipo pertenece a aquellos que existen en los dos bancos de registros que son accesibles por el usuario. El segundo tipo pertenece a aquellos que existen en el microprocesador. La Tabla 2.18 contiene los tipos de operandos del registro.

Tipo	Formatos
Registros accesibles para el usuario	reg (expr)
Registros del M8C	A
	F
	SP
	X

Tabla 2.18 Formatos de los registros

2.3.1.3.5 RAM: Las referencias son hechas colocando la dirección o expresión en corchetes. El ensamblador evaluará la expresión para crear la dirección de la RAM actual.

Tipo	Formato
Página de la RAM en curso (actual)	[expr]

Tabla 2.19 Formato de la RAM

2.3.1.3.6 Expresiones: Pueden construirse usando cualquier combinación de etiquetas, constantes, operador punto, y las operaciones lógicas y aritméticas definidas en la Tabla 2.20.

Prioridad	Expresión	Símbolo	Forma
1	Complemento Bitwise	~	(~ a)
2	Multiplicación, División, Módulo	* / %	(a * b) (a / b) (a % b)
3	Suma, Resta	+ -	(a + b) (a - b)
4	AND Bitwise	&	(a & b)
5	XOR Bitwise	^	(a ^ b)
6	OR Bitwise	|	(a | b)
7	Byte alto de una dirección	>	(> a)
8	Byte bajo de una dirección	<	(< a)

Tabla 2.20 Expresiones

Sólo la expresión de Suma (+) puede aplicarse a un símbolo recambiable (es decir, a un símbolo externo). Todas las otras expresiones deben aplicarse a constantes o símbolos descifrables por el ensamblador (es decir, un símbolo definido en el archivo).

2.3.1.4 Comentarios

Un comentario empieza con un punto y coma (;) o con un doble slash (/ /) y va al final de una línea. Normalmente se usa para dar una explicación de lo que se va codificando y puede ponerse en cualquier parte del archivo fuente. El Assembler del PSOC Designer ignora los comentarios, sin embargo, estos se escriben en el registro de archivos para referencia.

2.3.1.5 Directores

Un director se usa para indicar al ensamblador que tome alguna acción durante el proceso de ensamblaje. Los directores no son entendidos por el microprocesador M8C.

Como tal, los directores permiten al escritor del firmware crear códigos que sean más fáciles de mantener. Los directores del Assembler, se encuentran en el ANEXO "B"

2.4 SET DE INSTRUCCIONES

El M8C sustenta un total de 256 instrucciones, las cuales están divididas en 37 tipos de instrucciones.

La tabla correspondiente al set de instrucciones se la encuentra en el ANEXO "B"

2.5 COMPILADOR DE LENGUAJE "C" DEL PSOC DESIGNER

El Compilador del PSOC Designer compila cada archivo fuente *.c* a un archivo de ensamblaje. El Ensamblador luego traduce cada archivo de ensamblaje (cualquiera que sea producido por el compilador o aquellos que hayan sido agregados), en un archivo objeto *.o*.

Después de que todos los archivos se han traducido a archivos objeto, el ***builder/linker*** los combina, con el afán de formar un archivo ejecutable. Entonces, este archivo ***.rom*** es descargado al emulador en donde se depura para obtener un funcionamiento eficaz del diseño creado.

2.6 ARCHIVOS DEL COMPILADOR

2.6.1 ARCHIVO INICIAL:

El PSOC Designer crea un archivo inicial llamado boot.asm. Dentro de sus funciones primarias se incluyen: inicialización de variables C, organización de tablas de interrupción, y llamando del ***_main***.

El guión bajo (***_main***) le permite al boot.asm llamar un "main" ya sea en C o en assembler.

Algunas funciones dentro del PSOC Designer están construidas sobre especificaciones de este archivo. Por consiguiente, es muy recomendable que este no se modifique.

El archivo inicial ***boot.asm*** también define el vector reset. No se necesita modificar este archivo para usar otras interrupciones ya que el PSOC Designer maneja interrupciones y vectores.

2.6.2 DESCRIPCIÓN DE LAS LIBRERÍAS

Hay tres librerías de código usadas por el PSOC Designer: ***libcm8c.a***, ***libpsoc.a*** y ***cms.a***.

La ***libcm8c.a*** reside en:

... \tools directory (...\Program Files\Cypress Microsystems\PSoC Designer\tools).

Esta librería contiene muchas funciones usadas típicamente en la programación ' C '.

La librería ***libpsoc.a*** reside en el *project...\lib directory,* y contiene funciones de los Módulos de Usuario. El editor del dispositivo automáticamente agrega el código fuente para los Módulos de Usuario a la librería, durante el proceso de generación-aplicación. Sin embargo, otros objetos de librería pueden ser manualmente agregados a la **libpsoc.a.**

Para agregar archivos de objeto existentes, se copia el archivo fuente al *project...\lib directory*, entonces "oficialmente" este se agrega al proyecto en el PSOC Designer.

La librería ***cms.a*** reside en ... *\tools directory.* Esta librería contiene funciones convenientes que no involucran los Módulos de Usuario.

2.7 FUNDAMENTOS DEL COMPILADOR

Dentro de estos fundamentos se encuentran:

2.7.1 TIPOS

El compilador C del PSOC Designer, sustenta los siguientes tipos de datos estándar:

Tipo	Bytes	Descripción
char	1	Un solo byte de memoria que define caracteres
Int	2	Usado para definir números enteros
short	2	Tipo estándar especificando enteros de 2 bytes
long	4	Tipo estándar especificando la entidad de un entero largo
float	4	Un solo número de precisión de punto flotante e formato IEEE
double	4	Un solo número de precisión de punto flotante e formato IEEE
enum	1 si enum < 256 2 si enum > 256	Usado para definir una lista de "seudónimos" que representan enteros

Tabla 2.21 Tipos de datos

Las siguientes definiciones de tipo, están incluidas en el ***m8c.inc***. Expresan convenciones comunes para tipos de datos adicionales.

```
typedef unsigned char BOOL;
typedef unsigned char BYTE;
typedef signed char CHAR;
typedef unsigned int WORD;
typedef signed int INT;
```

typedef unsigned long DWORD;
typedef signed long LONG;

Las siguientes operaciones de punto flotante están sustentadas en el Compilador C del PSOC Designer. Estas son funciones intrínsecas:

comparar (=)
sumar (+)
*multiplicar (*)*
restar (-)
dividir (/)
distribución (long to float), longitud del flotante.

2.7.2 OPERADORES

La lista de los operadores más comunes sustentados dentro del Compilador C, se encuentran en la tabla respectiva en el ANEXO "B".

Se debe aclarar que los operadores con una prioridad más alta son aplicados primero, mientras que operadores de la misma prioridad son aplicados de derecha a izquierda. Se usa un paréntesis donde sea apropiado para prevenir ambigüedad.

2.7.3 EXPRESIONES

El PSOC Designer sustenta expresiones estándar del lenguaje C.

2.7.4 DECLARACIONES

El compilador sustenta las siguientes declaraciones estándar:

- ***if else:*** Decide en una acción, basándose en si esta es una verdad.

- ***switch:*** Compara una sola variable hasta con algunas constantes. Si la variable iguala a una de las constantes, se realiza un salto.

- **while:** Repite (lazo reiterativo) una declaración hasta que la expresión sea falsa.

- **do:** Lo mismo que ***while***, solo que la prueba corre después de la ejecución de la declaración, no antes.

- **for:** Ejecuta un lazo controlado.

- **goto:** Transfiere la ejecución a una etiqueta.

- **continue:** Se usa en un lazo para saltar al resto de la declaración.

- **break:** Usado con un switch o en un lazo para terminar a cualquiera de los dos respectivamente.

- **return:** Termina la función en curso.

- ***struct:*** Usada para agrupar variables comunes a la vez.

- ***typedef:*** Declara un tipo.

2.7.5 PUNTEROS

Un puntero es una variable que contiene una dirección que apunta a los datos. Este puede apuntar a cualquier tipo de datos (int, flote, char, etc).

Un puntero genérico (o desconocido) es declarado como nulo o vacío, y puede ser libremente arrojado en medio de otros tipos de punteros. Debido a la naturaleza de la arquitectura Harvard de los M8C, un puntero puede apuntar a datos localizados en cualquiera de las memorias: de datos o de programa.

Para distinguir a cual de ellos se ha tenido acceso, el llamado ***const qualifier*** es usado para indicar que un item de los datos esta localizado en la memoria de programa. Los punteros requieren 2 bytes del almacenamiento de memoria.

2.7.6 DIRECTORES DE PROCESO (#'S)

Los pre-procesadores y pragmas que el PSOC Designer sustenta, se encuentran en las tablas de Directores en el ANEXO "B".

2.8 FUNCIONES

Las funciones del compilador C del PSOC Designer usan argumentos y siempre devuelven un valor.

Todos los programas en C deben tener una función ***called main ( )***.

Cada función debe ser autónoma, es decir no se puede definir una función dentro de otra función, ó extender la definición o una función en más de un archivo.

Es importante notar que el compilador genera un código en línea siempre que es posible. Sin embargo, para algunas estructuras de C, el compilador genera llamadas para rutinas de nivel bajo. A estas rutinas se anteponen dos guiones bajos y no deberían ser llamadas directamente por el usuario.

2.8.1 FUNCIONES DE LIBRERÍA

Se usa *#include <associated-header.h>* para cada función del compilador.

Notar que se proporcionan dos versiones de estas funciones. El prefijo ***'c'*** indica que el string fuente ***s2*** se localiza en la Flash, tal como fue designado por el ***const qualifier***.

El PSOC Designer sustenta las siguientes funciones de librería:

2.8.1.1 Funciones String

Estas funciones pueden ser encontradas en el directorio de instalación del PSOC Designer, en:

... :\Program Files\Cypress MicroSystems\PSoC Designer\tools\include\string.h and stdlib.h

Para este caso, dichas funciones también se ilustran en el ANEXO "B"

2.8.1.2 Funciones matemáticas

Estas funciones pueden encontrarse en el directorio de instalación del PSOC Designer en:

... :\Program Files\Cypress MicroSystems\PSoC Designer\tools\include\math.h.

Estas funciones también se presentan en el ANEXO "B"

2.8.1.3 Funciones de la librería Software API (Application Programming Interface)

De igual forma, la tabla de estas funciones se puede encontrar en el ANEXO "B"

Además, lo referente al encabezamiento y archivos *include* pueden encontrarse en: *... :\Program Files\Cypress MicroSystems\PSoC Designer\tools\include.*

2.8.2 INTERFAZ ENTRE C Y ASSEMBLER

Para optimizar el paso de un argumento y los valores de retorno, desde una función 'C' a una función assembler, se usa el *#pragma fast call* (llamado rápido). La convención del *fastcall* fue inventada para crear un mecanismo eficaz para argumento/valor de retorno, entre las funciones de los lenguajes 'C ' y assembler.

Tipo de argumento	Registro	Argumento del registro
Char	A	
char, char	A, X	Primer char en A y el segundo en X
Int	X, A	MSB en X y LSB en A
Pointer	A, X	MSB en A y LSB en X
char, ...	A, X	Primer argumento entregado en A. Sucesivamente los argumentos son apuntados a X, donde X es seteado como un puntero para los argumentos restantes. Típicamente, estos argumentos se guardan en el stack
Int, ...	X	X es seteado como un indicador que apunta al bloque inmediato de memoria que guarda los argumentos. Típicamente, estos argumentos se guardan en el stack
Todas las otras	X	Igual que la anterior

Tabla 2.22 Convenciones # pragma fastcall para paso del argumento

Los argumentos que se colocan en el stack, se ubican de derecha a izquierda.

La referencia de las estructuras de retorno se encuentra en los registros A y X. Si tiene un valor, la estructura es siempre entregada a través del stack, y no por registros.

Entregar una estructura con referencia (es decir, pasar la dirección de esa estructura) es lo mismo que entregar la dirección de cualquier ítem de datos, esto es, un puntero.

Tipo de retorno	Registro de retorno	Comentario
Char	A	
Int	X, A	
Long	*_r0* ... _r3	Entregado en los registros virtuales
Pointer	A, X	

Tabla 2-23 Convenciones # pragma fastcall para valores de retorno

2.9 CONSIDERACIONES ADICIONALES

2.9.1 INTERRUPCIONES

Interrupciones handler pueden ser escritas en C. Pero, para poder emplearlas, se debe informar primero al compilador que la función es una interrupción handler.

Se puede realizar esto, utilizando el siguiente pragma (en el archivo donde se define la función, antes de la definición de la función).

*#pragma interrupt_handler <name> **

Para una función de interrupción, el compilador genera la instrucción **reti** en lugar de la instrucción **ret**, entonces graba y restaura todos los registros usados en la función.

Los llamados registros virtuales son guardados, solo si son usados por la rutina. Si una interrupción handler llama otra función, entonces el compilador guarda y restaura todos los registros virtuales.

Por ejemplo:

```
#pragma interrupt_handler          timer_handler
...
void timer_handler ( )
{
...
}
```

Se pueden colocar múltiples nombres en un solo *pragma interrupt_handler,* separados por espacios. Por ejemplo:

```
#pragma interrupt_handler timer_ovf sci_ovf
```

2.9.2 REGISTROS DE I / O

Los registros de I/O son especificados usando el siguiente #pragma:

```
#pragma ioport          // ioport is at I/O space 0x04
LED : 0x04 ;            LED must be declared in global scope
char LED ; ....
LED = 1 ;
```

2.9.3 ÁREAS DE MEMORIA

El compilador genera código y datos en diferentes "áreas". Las áreas usadas por el compilador son:

2.9.4 ÁREAS DE LA MEMORIA FLASH

- **Vectores de interrupción:** Esta área contiene los vectores de interrupción.
- **func_lit:** Área de tabla de la función. Cada palabra en esta área contiene la dirección de una función ingresada.
- **lit:** Esta área contiene enteros y constantes de punto flotante.
- **idata:** Los valores iniciales para los datos globales son almacenados en esta área.
- **texto:** Esta área contiene código de programa.

2.9.5 MEMORIA DE DATOS

data: Esta es el área de datos que contiene variables globales y estáticas, además strings. Los valores iniciales de las variables globales se almacenan en el área idata y copiados al área de datos en un tiempo de inicio.

bss: Ésta es el área de datos que contiene variables globales en C no inicializadas. Por definición del ANSI C, estas variables pondrán a cero el tiempo de inicio.

2.9.6 STRINGS

El compilador coloca todos los strings literales en la memoria de programa.

Eficazmente, el tipo para declaración de un string literal es "const char" y el tipo para referencia de este es "const char*". Se debe asegurar que los parámetros de la función tomen el tipo de argumento apropiado.

2.9.7 REGISTROS VIRTUALES

Los registros virtuales son usados para el almacenamiento de datos temporales cuando el compilador está en proceso. Las localidades _r0, _r1, _r2, _r3, _r4, _r5, _r6, _r7, _r8, _r9, _r10, _r11, _rX, _rY, y _rZ están disponibles. Sólo aquellos que son requeridos realmente se usan.

Este espacio de registro extra es necesario porque el M8C sólo tienen un acumulador de 8 bits. Los registros virtuales se colocan en el extremo inferior de la memoria de datos.

2.10 PROGRAMACIÓN GRÁFICA EN EL PSOC DESIGNER

A parte de la programación escrita que se usa para la implementación de un determinado firmware dentro del microcontrolador PSOC, es también necesaria (según el caso) la programación gráfica; la cual se define a través de los Módulos de Usuario presentados en el PSOC Designer.

Para acceder a la lista de módulos que están disponibles se selecciona el icono correspondiente, tal como se muestra en la Figura 2.2

Cuando el módulo deseado ha sido seleccionado definitivamente, se da un doble clic al mismo para luego encontrarlo y manejarlo en la vista de interconexión. Este proceso se ilustra en la Figura 2.5.

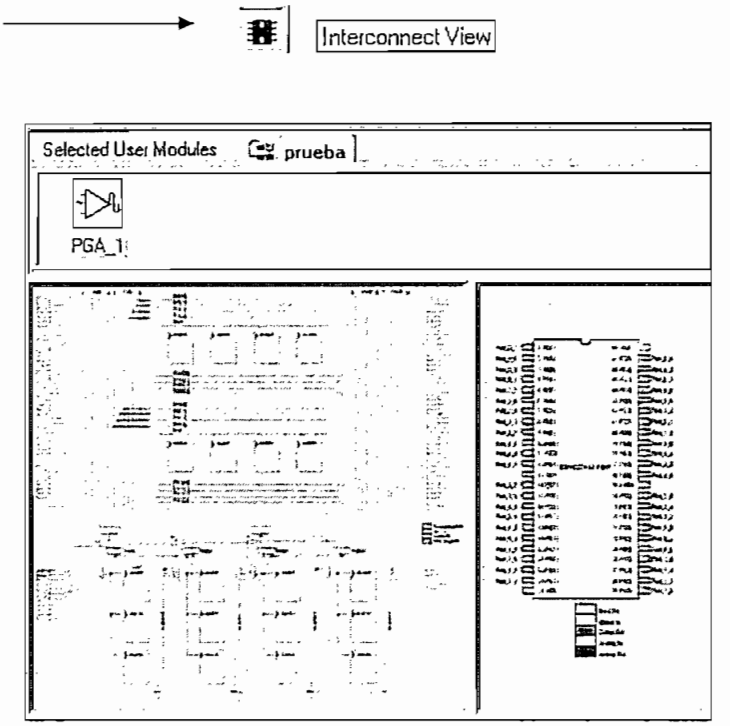


Figura 2.5 Traspaso de un Módulo de Usuario a la vista de interconexión

El siguiente paso es enrolarlo dentro del programa, procedimiento que se lleva a cabo haciendo un "place" en el módulo. Este proceso se lo indica en las Figuras 2.6 y 2.7.

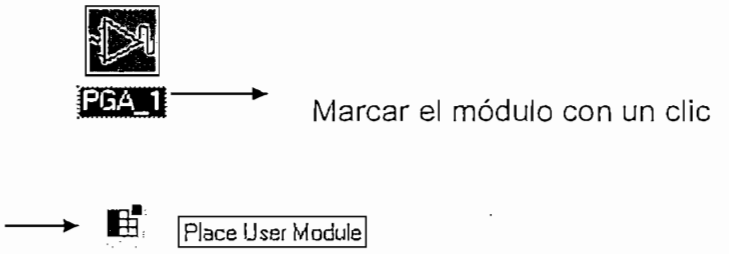


Figura 2.6 Procedimiento para enrolado del Módulo de Usuario

Finalmente el módulo queda enrolado al programa.

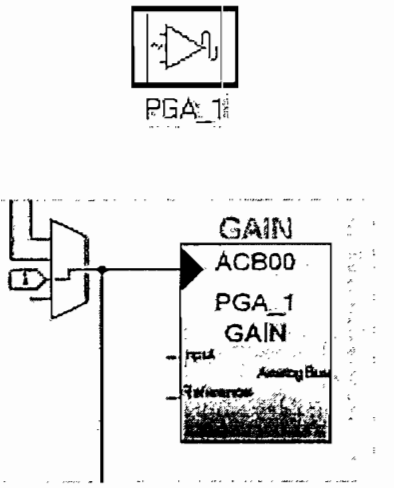


Figura 2.7 Enrolado del Módulo de Usuario

Finalmente la conexión "para" y "entre" los módulos, se realiza de acuerdo a los parámetros que cada uno de estos presente, y también de acuerdo a las necesidades que se tenga dentro de un diseño.

Las Figuras 2.8 y 2.9 ilustran este procedimiento

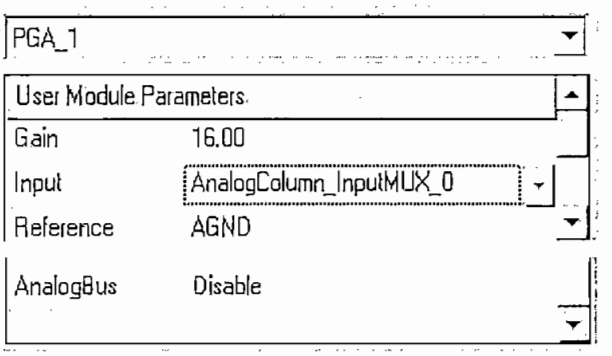


Figura 2.8 Parámetros del Módulo de Usuario

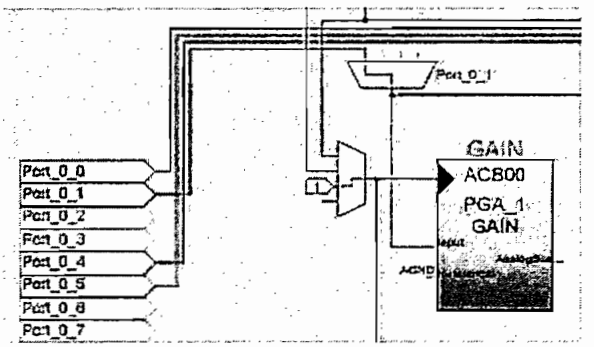


Figura 2.9 Conexiones del Módulo de Usuario

Para finalizar con la explicación de estos Módulos de Usuario, se debe acotar que cada uno de los Módulos representan la llamada a una función previamente diseñada por el fabricante del software y en la que se encuentra el código necesario para inicializarlos y ponerlos en funcionamiento. Este código se encuentra en archivos con extensión *.asm* los cuales automáticamente se generan al momento de la compilación de todo el programa.

En el capítulo siguiente se tratará lo referente al diseño y construcción del Módulo Didáctico propuesto, en el cuál se analizarán cada una de las etapas y elementos que conforman su hardware.

CAPÍTULO 3

DISEÑO Y CONSTRUCCIÓN DEL MÓDULO DIDÁCTICO

CAPÍTULO 3

DISEÑO Y CONSTRUCCIÓN DEL MÓDULO DIDÁCTICO

En el presente capítulo, se realiza una descripción de las etapas y elementos que componen el hardware diseñado para la construcción del Módulo Didáctico.

3.1 DESCRIPCIÓN GENERAL

El módulo realizado cuenta con cuatro etapas claramente definidas, las cuales son totalmente independientes una de otra.

En base a estas etapas, se desarrollarán las aplicaciones propuestas para este proyecto, además se dejará abierta la posibilidad de realizar diferentes diseños y aplicaciones futuras, ya que la estructura de este hardware no incluye conexiones fijas desde el microcontrolador PSOC CY8C27643 hacia los distintos periféricos implementados, sino que estas se las hará mediante cables externos, de acuerdo a las necesidades que se tenga.

A continuación se enumeran las etapas que conforman el módulo:

1.- Etapa correspondiente a la fuente de alimentación de 5VDC.
2.- Etapa de Control.
3.- Etapa de conexiones para el programador.
4.- Etapa de Desarrollo.

3.2 DIAGRAMA DE BLOQUES DEL MÓDULO

En la Figura 3.1 se ilustra el diagrama de bloques de la estructura del módulo desarrollado.

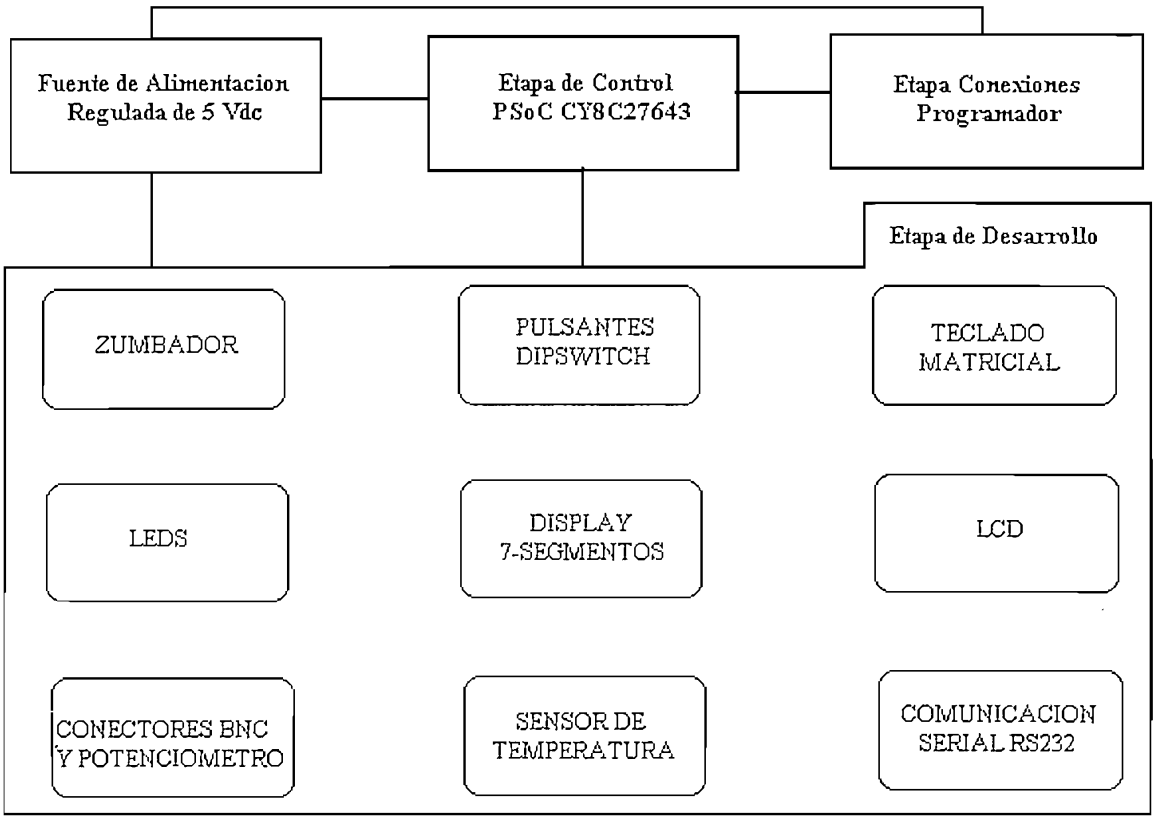


Figura 3.1 Diagrama de Bloques del Módulo Didáctico

3.3 DISEÑO DE LA FUENTE DE ALIMENTACIÓN (Etapa 1)

La fuente de alimentación, debe tener una salida de 5Vdc regulada, la misma que se encargará de energizar a las diferentes etapas numeradas anteriormente.

Los dispositivos utilizados para el diseño de esta fuente son:

- Un switch de encendido y apagado.
- Un fusible de protección.
- Un led indicador.
- Una resistencia de 330: .
- Un puente de diodos semiconductores.

Por lo tanto el valor Vrms será:

$$Vrms = \frac{\ni Vo}{3.5}$$

$$Vrms = \frac{0.68}{3.5}$$

$$Vrms = 0.194V$$

En la Figura 3.2 se muestra el esquemático de la fuente de alimentación:

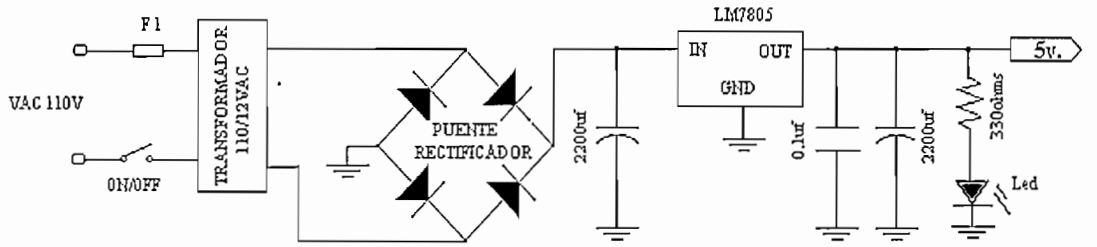


Figura 3.2 Fuente de Alimentación Regulada de 5 Vdc

3.4 ETAPA DE CONTROL

Es la etapa principal del sistema, ya que contiene al microcontrolador PSOC CY8C27643 como unidad central (incluido su circuito de reset).

También posee un switch selector, el cual seleccionará dos modos de trabajo: la carga del programa y la ejecución del mismo, evitando de esta manera la manipulación constante de la tarjeta del chip. Adicionalmente, se puede considerar un capacitor de tantalio de 0.1 μf conectado entre VCC y GND, el mismo que servirá para evitar resets involuntarios.

En la Figura 3.3 se presenta el diagrama de la etapa de control.

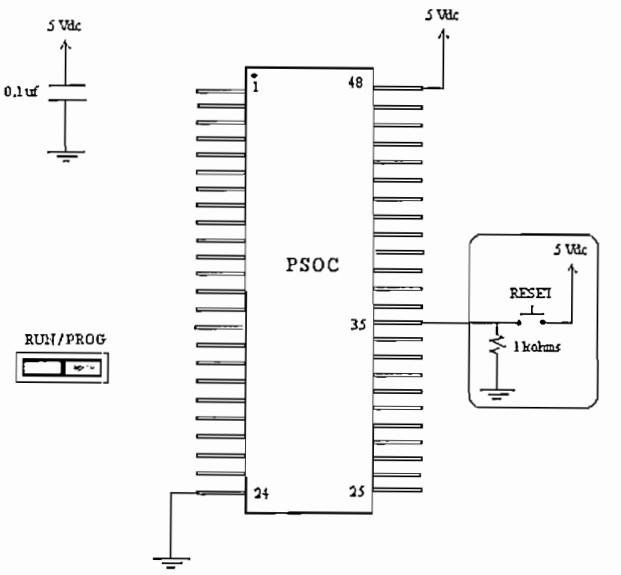


Figura 3.3 Etapa de Control

3.5 ETAPA DE CONEXIONES PARA EL PROGRAMADOR

La Figura 3.4 muestra las conexiones necesarias para hacer posible la carga de los programas (software), en el microcontrolador PSOC CY8C27643.

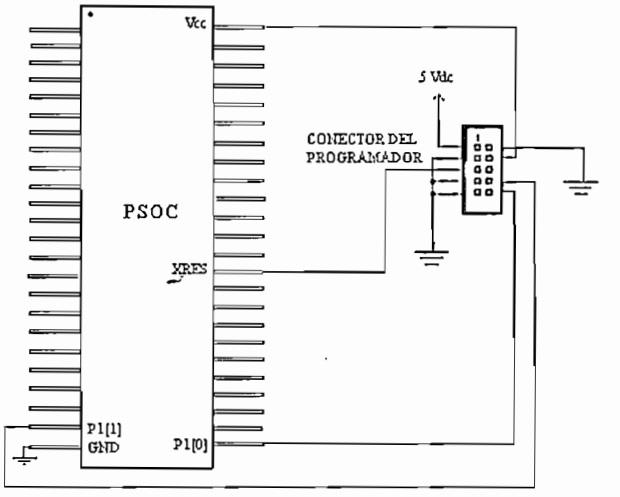


Figura 3.4 Etapa de Conexiones para el Programador

3.6 ETAPA DE DESARROLLO

Esta etapa está constituida por los siguientes periféricos, los cuales serán usados para la realización de las aplicaciones destinadas a este proyecto:

- Pulsantes y Dipswitch.
- Teclado Matricial.
- Leds.
- Display de 7segmentos.
- Display de Cristal líquido LCD.
- Conectores BNC y Potenciómetro.
- Sensor de Temperatura.
- Comunicación Serial RS232.
- Zumbador.

3.6.1 PULSANTES Y DIPSWITCH

Este periférico consta de ocho resistencias de 1k: , las mismas que se conectan en serie con ocho pulsantes on / off, que a su vez están conectados en paralelo con un dipswitch.

En la Figura 3.5 se indica el diagrama para este periférico.

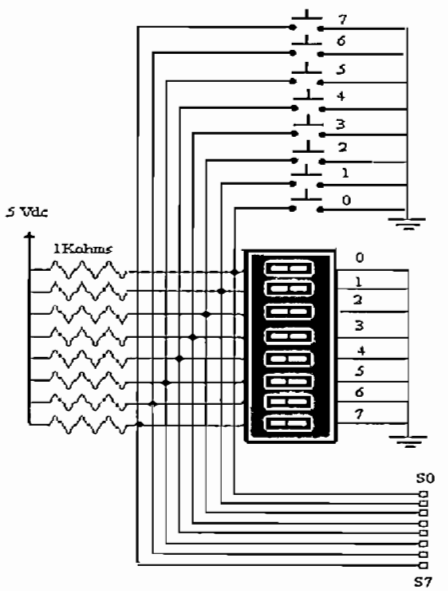


Figura 3.5 Diagrama de Pulsantes y Dipswitch

De acuerdo al esquema circuital cuando el pulsante no es presionado o el switch se encuentra en estado normalmente abierto (off), el valor lógico en los terminales S0.....S7 será de 1L (+5 Vdc); mientras que si es presionado, uno de los pulsantes o el switch se coloca en estado normalmente cerrado (on), su valor lógico será de 0L (0 Vdc).

3.6.2 TECLADO MATRICIAL

Se utilizará un teclado matricial de 4 filas por 3 columnas (4x3).

La Figura 3.6 presenta el diagrama de conexiones para el teclado.

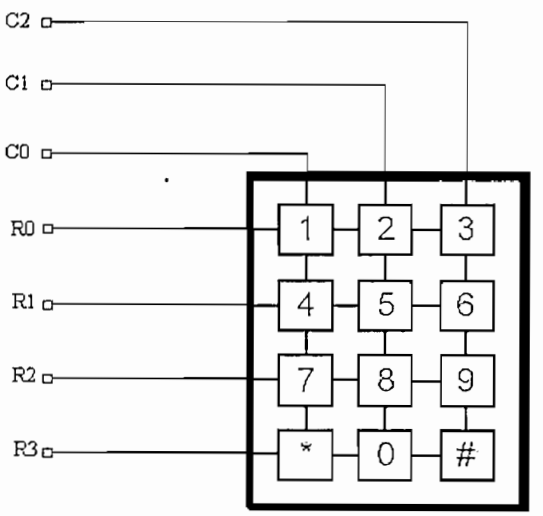


Figura 3.6 Diagrama del Teclado Matricial

3.6.3 LEDS

En este caso se utilizan doce resistencias limitadoras de corriente de 330: conectadas en serie con doce diodos leds.

Para el encendido de los leds es necesario escribir un 1L lógico (+5 Vdc) en los terminales, de tal manera que el diodo se polarice directamente y pueda encenderse.

En la Figura 3.7 se ilustra el circuito usado.

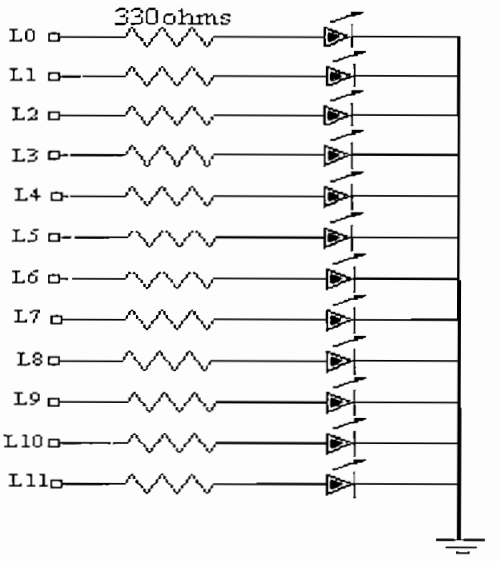


Figura 3.7 Diagrama de Leds

3.6.4 DISPLAY DE 7 SEGMENTOS

Este periférico utiliza siete resistencias limitadoras de corriente de 220: conectadas a cada segmento del display de tipo ánodo común.

La Figura 3.8 muestra el diagrama de conexiones para el display.

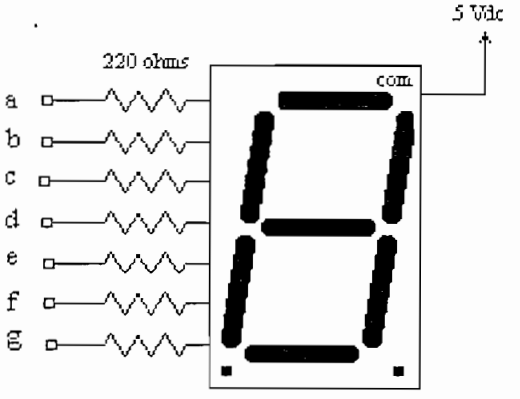


Figura 3.8 Diagrama del Display de 7 Segmentos

El estado en cada terminal debe ser 1L (+5 Vdc), entonces, los segmentos permanecerán apagados ya que no existe diferencia de potencial; por lo tanto, para encender cada uno de estos, se deberá colocar 0L (0 Vdc) en el terminal respectivo.

3.6.5 DISPLAY DE CRISTAL LÍQUIDO LCD

El dispositivo LCD usado es de 2x16 (dos filas de 16 caracteres), este cuenta con catorce pines de los cuales los pines 1 y 2 son para polarización (GND y VCC respectivamente), el pin 3 se usa para la regulación del contraste de la pantalla (potenciómetro10k:). Cuatro pines son necesarios para el bus de datos y 3 para las líneas de control.

En la Figura 3.9 se presenta el LCD con su respectivo circuito de conexión.

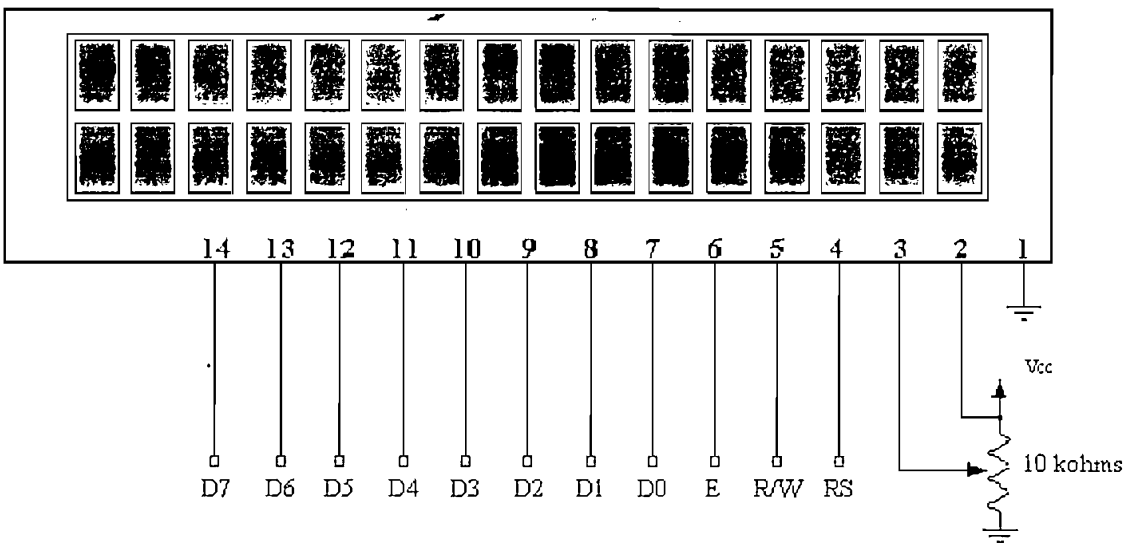


Figura 3.9 Diagrama del LCD

3.6.6 CONECTORES BNC Y POTENCIÓMETRO

Este periférico cuenta con tres conectores BNC utilizados para las entradas análogas que provienen desde un generador, y también para salidas de señal, que serán visualizadas en un osciloscopio.

También consta de un potenciómetro de 20 k: , del cual se obtendrá una señal análoga variable de 0 a 5 Vdc.

En la Figura 3.10 se ilustrará dicho periférico.

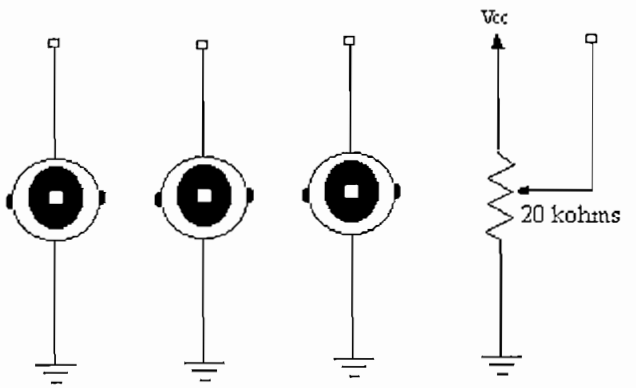


Figura 3.10 Diagrama de Conectores BNC y Potenciómetro

3.6.7 SENSOR DE TEMPERATURA

Para este caso se utilizará un sensor de temperatura digital 1-wire, *DALLAS DS1821*, y una resistencia pull-up de 4.7k: (recomendada por el fabricante). Esta resistencia irá conectada entre VCC y el terminal DQ del sensor, haciendo que el estado lógico en DQ, cuando este está inactivo, sea alto (1L).

Es posible también eliminar esta resistencia externa. Lo que se hace es configurar el pin del PSOC por el cual ingresa el DQ, como pull-up.

El Datasheet del *DALLAS DS1821* se lo encuentra en el ANEXO "C"

En la Figura 3.11 se presenta la configuración del DS1821.

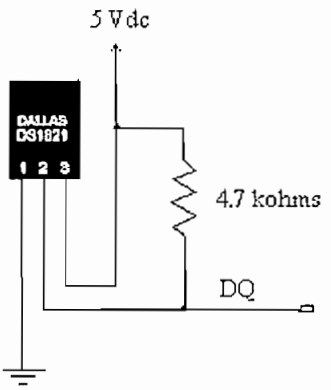


Figura 3.11 Diagrama del Sensor de Temperatura

3.6.8 COMUNICACIÓN SERIAL RS232

Se usa el MAX232, que es la interfaz de comunicación entre el PSOC CY8C27643 y el PC, un conector DB9 para conexión con el puerto serial del PC, y cinco capacitores electrolíticos de 10uf. Este esquemático se presenta en la Figura 3.12

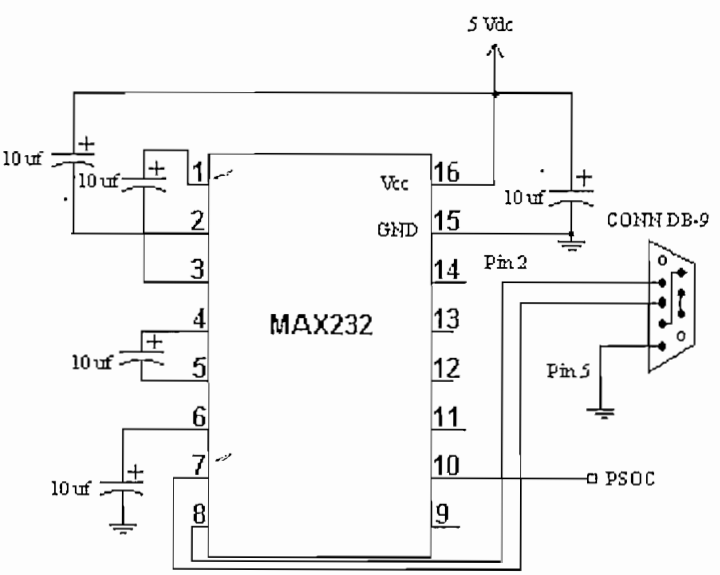


Figura 3.12 Diagrama de conexiones del MAX232

3.6.9 ZUMBADOR

El periférico que se ilustra en la Figura 3.13 consta de un zumbador de 12 voltios conectado en serie a una resistencia de 470 ohms.

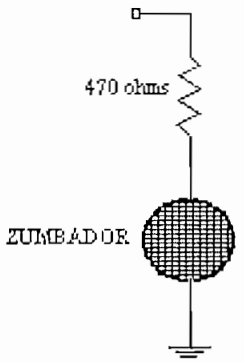


Figura 3.13 Diagrama del Zumbador

Finalmente en la Figura 3.14 que se presenta a continuación, se ilustra el esquema completo del hardware del Módulo Didáctico propuesto, en el cual constan en conjunto, todas las etapas descritas anteriormente.

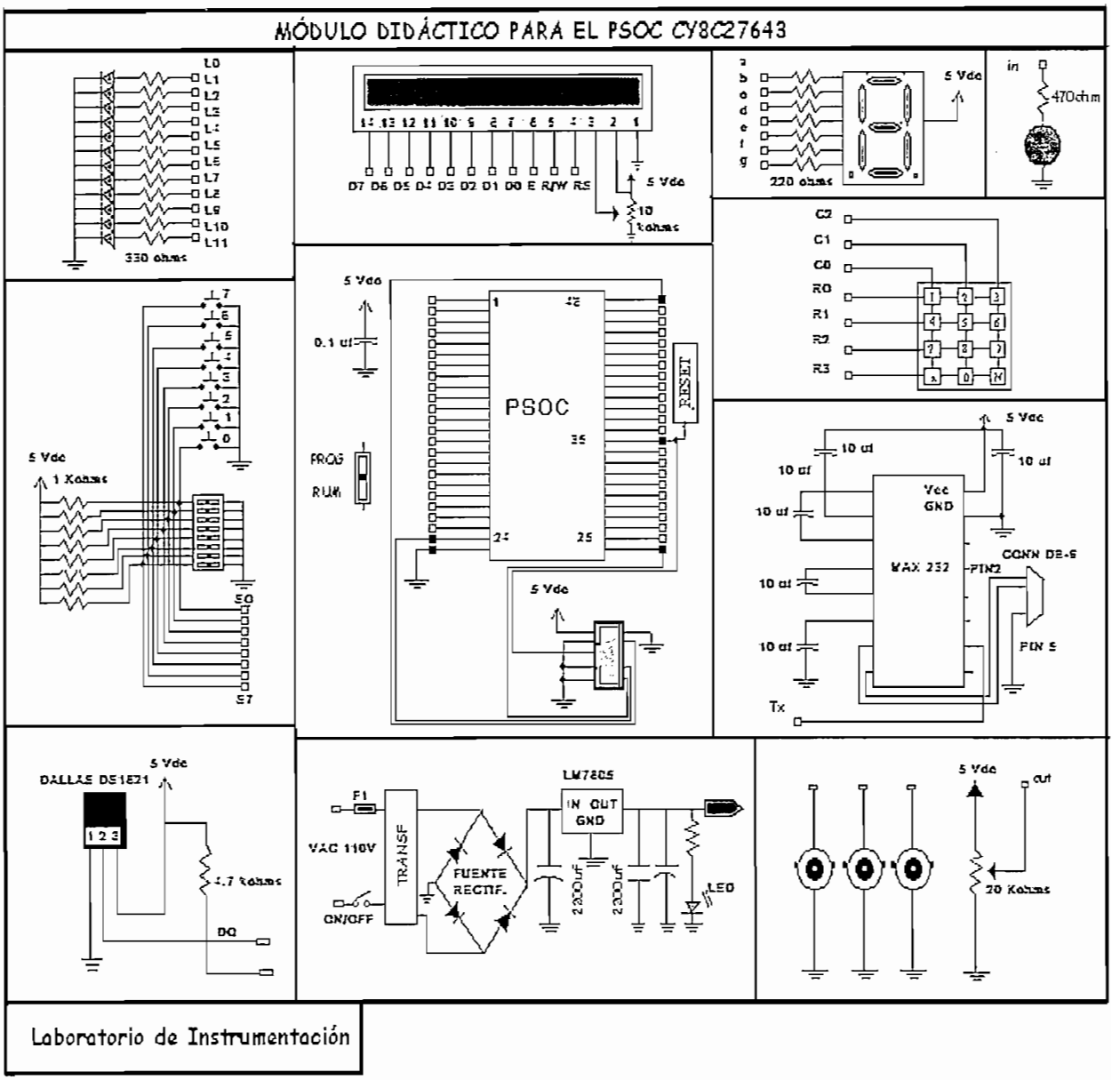


Figura 3.14 Esquema del Módulo Didáctico

En el capítulo siguiente se presenta el diseño de las prácticas a desarrollarse, utilizando las etapas del Módulo Didáctico aquí descritas.

CAPÍTULO 4

DISEÑO DE APLICACIONES USANDO LAS ETAPAS Y ELEMENTOS QUE CONSTITUYEN EL MÓDULO DIDÁCTICO

CAPÍTULO 4

DISEÑO DE APLICACIONES USANDO LAS ETAPAS Y ELEMENTOS QUE CONSTITUYEN EL MÓDULO DIDÁCTICO

En el presente capítulo, se trata lo referente a las aplicaciones prácticas que se desarrollaron y que servirán como referencia de las posibilidades del Módulo Didáctico construido.

A continuación se detalla cada una de las prácticas diseñadas y que, con el afán de ser didácticas, empiezan desde ejercicios simples hasta más complejos.

4.1 CONFIGURACIÓN DE I/O PARA LECTURA Y ESCRITURA EN LOS PUERTOS

La ejecución de este ejercicio es la apertura para familiarizar al usuario con el nuevo microcontrolador PSOC CY8C27643.

El objetivo de esta práctica es leer un Dipswitch conectado al Puerto 0, el cual debe estar configurado como entrada. El dato obtenido debe ser visualizado en ocho Leds conectados al Puerto 2, que se configura como salida.

Los periféricos utilizados son:

- Dipswitch.
- Ocho Leds.

4.1.1 DIAGRAMA CIRCUITAL

En la Figura 4.1 se ilustra el circuito utilizado para esta aplicación.

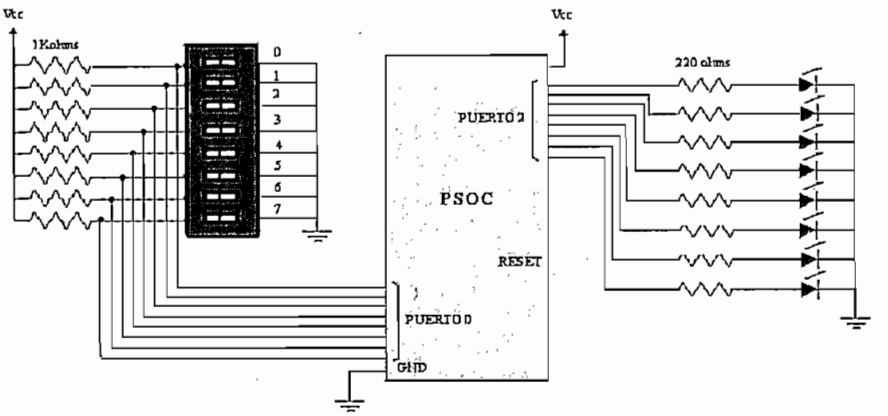


Figura 4.1 Diagrama de Conexiones implementado para lectura y escritura de puertos

4.1.2 DIAGRAMA DE FLUJO

El Diagrama de Flujo mostrado en la Figura 4.2 explica las funciones que el microcontrolador PSOC CY8C27643 realiza en esta aplicación:

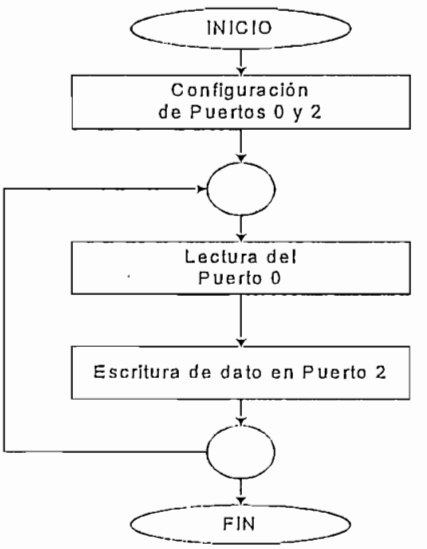


Figura 4.2 Diagrama de Flujo para lectura y escritura de puertos

A continuación se presenta en lenguaje estructurado, el desarrollo del programa.

Configuración de Puertos

- *Configurar Puerto 0 como entrada.*
- *Configurar Puerto 2 como salida.*

Fin Tarea

Lectura del Puerto 0

- *Leer puerto 0 (entradas).*
- *Guardar dato leído.*

Fin Tarea

Escritura de dato en Puerto 2

- *Enviar dato guardado, al Puerto 2.*

Fin Tarea

4.2 MANEJO DE UN TECLADO MATRICIAL Y VISUALIZACIÓN EN UN DISPLAY

Para este ejercicio se usa un Teclado Matricial de cuatro filas por tres columnas, el cual se conecta al Puerto 1 (pines P1.0 – P1.6, configurados como entradas).

El objetivo de la práctica es detectar qué tecla fue presionada por parte del usuario, para luego indicar el valor de este caracter en un Display de 7 segmentos tipo ánodo común, conectado en el Puerto 0 (pines P0.0 – P0.6, configurados como salidas).

Los periféricos utilizados son:

- Teclado Matricial 4x3
- Display de 7 segmentos tipo ánodo común.

4.2.1 DIAGRAMA CIRCUITAL

En la Figura 4.3 se ilustra el circuito utilizado para esta aplicación.

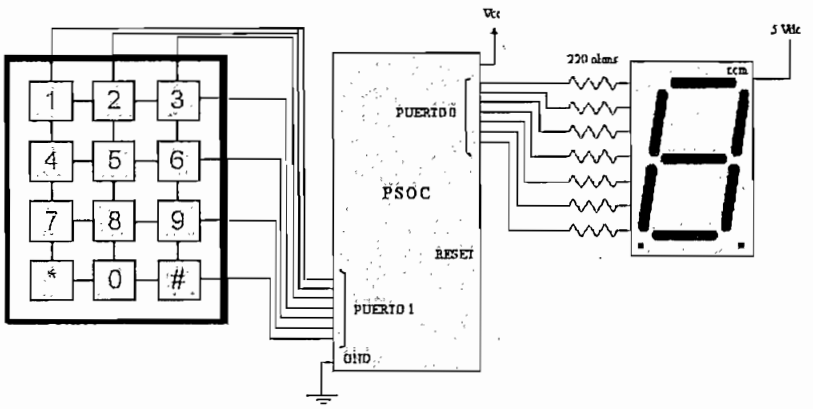


Figura 4.3 Diagrama de conexiones implementado para manejo de teclado y visualización en un display

4.2.2 DIAGRAMA DE FLUJO

El Diagrama de Flujo mostrado en la Figura 4.4 explica las funciones que el microcontrolador PSOC CY8C27643 realiza en esta aplicación:

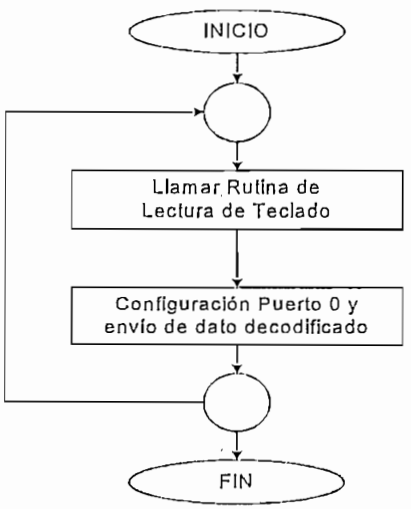


Figura 4.4 Diagrama de Flujo para manejo de teclado y visualización en un display

A continuación se presenta en lenguaje estructurado, el desarrollo del programa.

Llamar Rutina de Lectura de Teclado

- *Llamar rutina de Lectura de teclado.*

Fin Tarea

Configuración Puerto 0 y envío de dato decodificado

- *Configurar el Puerto 0 como salida.*
- *Escribir el dato decodificado en el Puerto 0.*

Fin Tarea

Rutina de Lectura de Teclado

En la Figura 4.5 se muestra el Diagrama de Flujo correspondiente a la Rutina de Lectura de Teclado, descrita en el programa principal

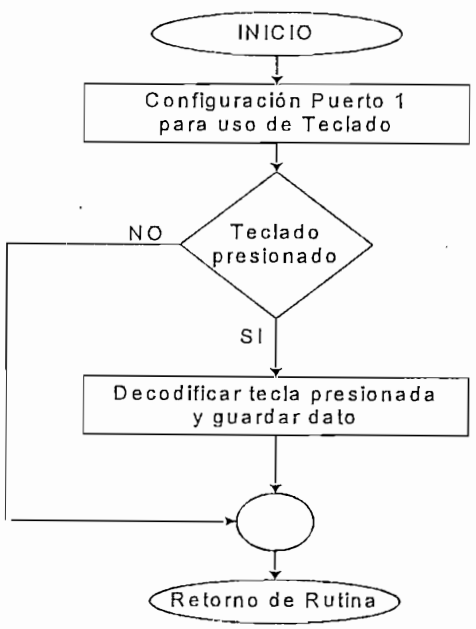


Figura 4.5 Diagrama de Flujo para manejo de teclado y visualización en un display

A continuación se detalla en lenguaje estructurado, el desarrollo de esta rutina.

Configuración Puerto 1 para uso de Teclado

- *Configurar Puerto 1 como entrada.*

Fin Tarea

Teclado Presionado

Si :

Decodificar tecla presionada y guardar dato

- *Leer dato del Puerto 1.*
- *Guardar dato leído en un registro.*

Fin Tarea

Retorno de Rutina

No :

Retorno de Rutina

4.3 MANEJO BÁSICO DEL LCD

Con este ejercicio se da inicio al uso y manejo de los Módulos de Usuario que contiene el PSOC Designer para hacer programación gráfica..

El objetivo de la práctica consiste en mostrar en el LCD, un texto que previamente es almacenado en la memoria de programa del PSOC, utilizando una interfaz de datos de 4 bits. De esta manera se reducen a cuatro, las ocho salidas del puerto de un microcontrolador que generalmente se utilizan para este propósito.

El LCD está conectado al Puerto 2 (pines P2.0 – P2.3 para datos y pines P2.4 – P2.6 para control).

La selección del puerto, al cual se conecta el LCD (periférico), se lo hace a través de los parámetros que el Módulo de Usuario LCD ofrece.

El periférico utilizado es:

- Display de Cristal Líquido (LCD).

El Módulo de Usuario que se utiliza en esta aplicación es:

- LCD.

4.3.1 DIAGRAMA CIRCUITAL

En la Figura 4.6 se ilustra el circuito utilizadó para esta aplicación.

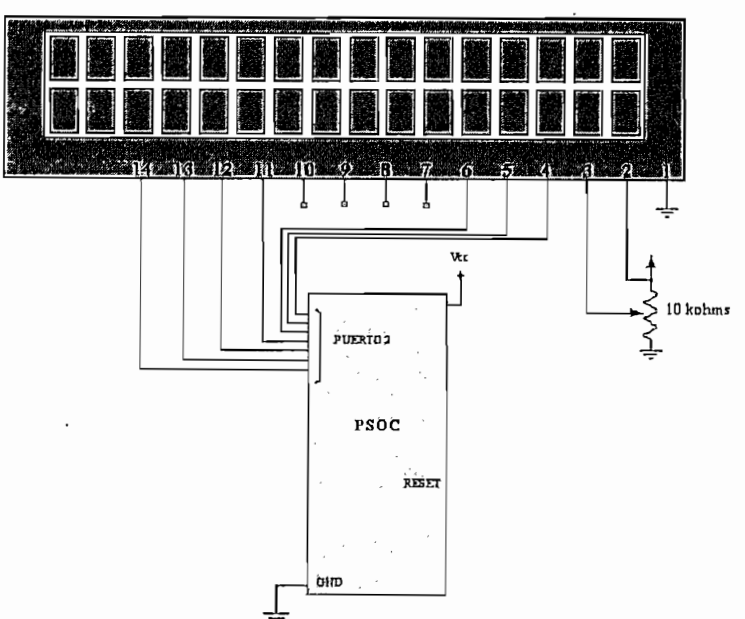


Figura 4.6 Diagrama de conexiones implementado para manejo básico del LCD

4.3.2 DIAGRAMA DE FLUJO

El Diagrama de Flujo mostrado en la Figura 4.7 explica las funciones que el microcontrolador PSOC CY8C27643 realiza en esta aplicación:

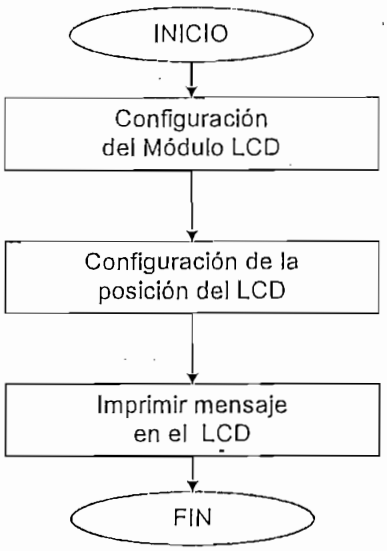


Figura 4.7 Diagrama de Flujo para manejo básico del LCD

A continuación se presenta en lenguaje estructurado, el desarrollo del programa.

Configuración del Módulo LCD

- *Inicializar LCD en modo nibble (4 bits).*
- *Inicializar LCD en modo multi-line (2 líneas).*
- *Esperar tiempo hasta inicialización correcta del LCD.*

Fin Tarea

Configuración de la posición del LCD

- *Configurar posición adecuada del LCD.*
- *Cargar comando de posicionamiento a la dirección establecida.*

Fin Tarea

Imprimir mensaje en el LCD

- *Apuntar a la ROM del microprocesador, de donde se leerá el string.*
- *Enviar comando para mostrar caracteres del string en el LCD.*

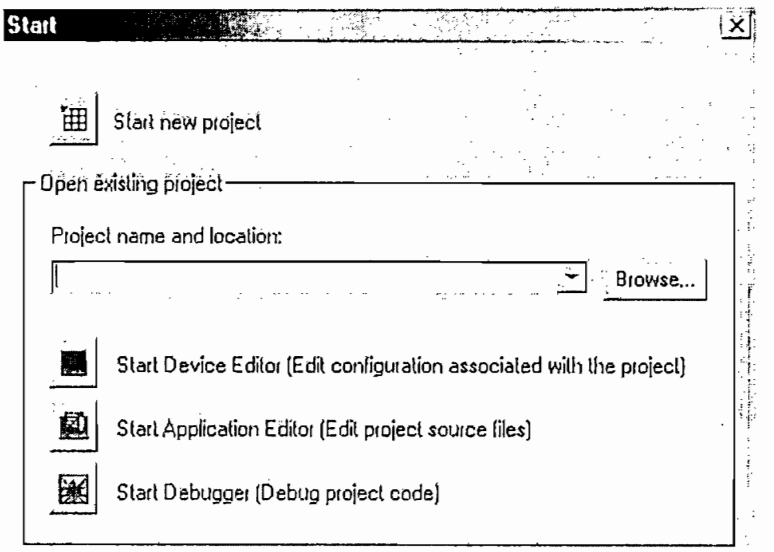
Fin Tarea

Como premisa para las aplicaciones siguientes, en esta práctica se detalla cada uno de los procedimientos necesarios para la creación del programa, lo cual implica el manejo del software computacional PSOC Designer, así como el uso correcto de los Módulos de Usuario que este posee.

A continuación se presentan estos procedimientos :

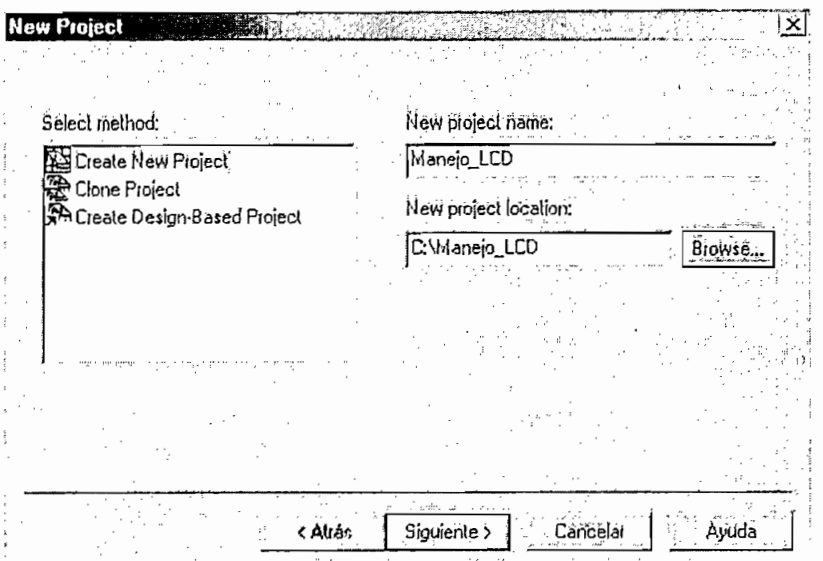
Selección del icono del PSOC Designer desde el escritorio.

Una semipantalla se abrirá:

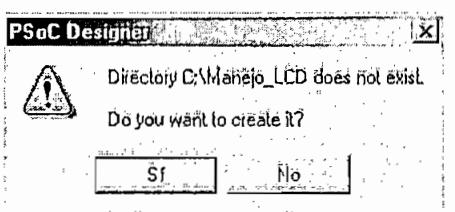


En esta se selecciona la opción que permite crear un nuevo proyecto, para esto se da un click en:

Al realizar esta acción, otra semipantalla se despliega:

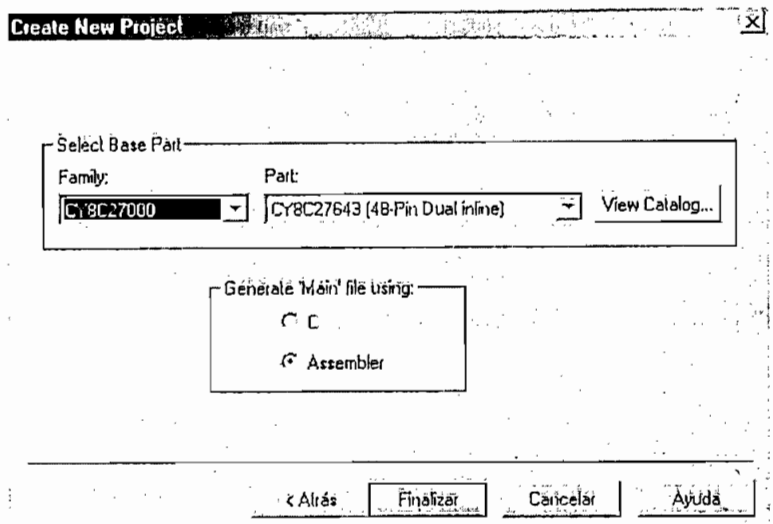


Aquí se escoge el nombre del proyecto y la ubicación en donde la carpeta de archivos, se almacenará. Luego se da un click en Siguiente.



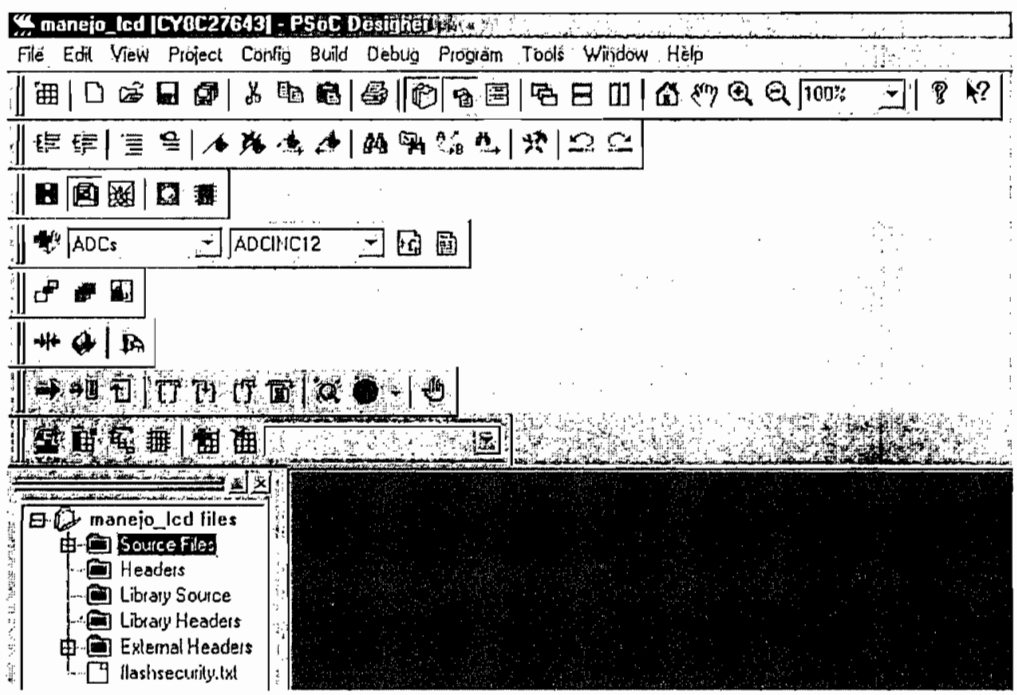
Se acepta la opción de crear el directorio correspondiente.

Al realizar esto, aparece la siguiente semipantalla:



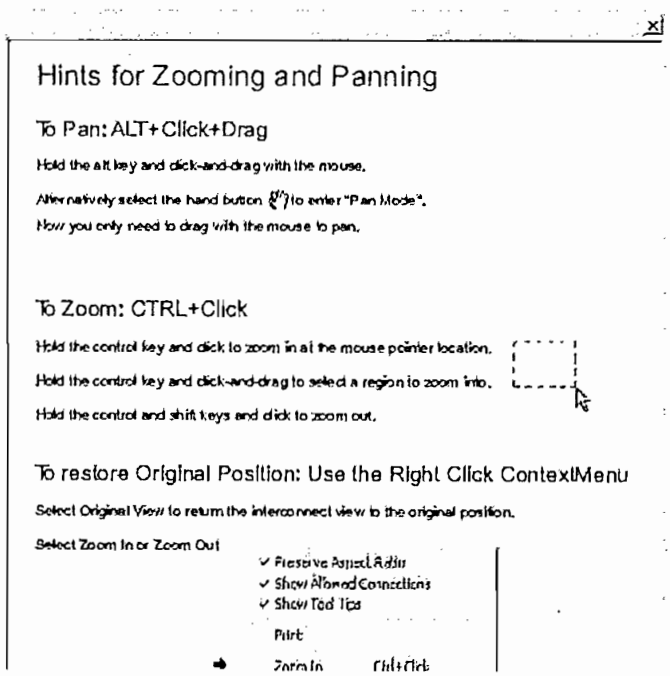
Esta permite escoger la familia a la que pertenece el dispositivo PSOC a usarse, su modelo y características, y finalmente el lenguaje de programación en el cual se va a trabajar. Al terminar la elección, se hace click en Finalizar.

Luego, una pantalla completa se despliega como sigue:



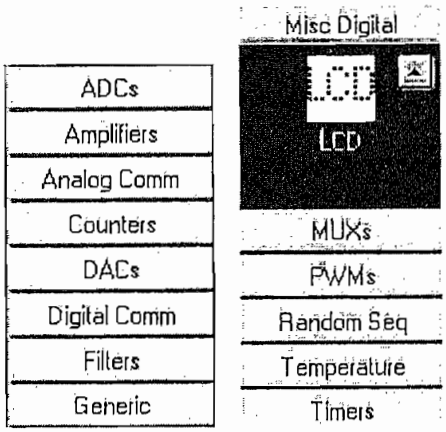
Como en esta práctica se va a introducir un Módulo de Usuario (LCD), el siguiente paso es seleccionarlo y enrolarlo al programa (procedimiento que se presentó anteriormente en el Capítulo 2), pero que para entendimiento adecuado del usuario, se lo ilustra dentro de esta práctica. Se hace entonces lo siguiente:

Aparece una semipantalla de recomendaciones, la cual después de leerla tiene que cerrarse:

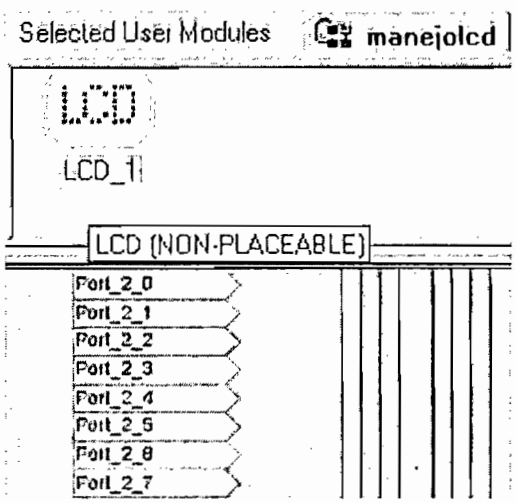


Ahora se accede a la lista de módulos que están disponibles haciendo lo siguiente:

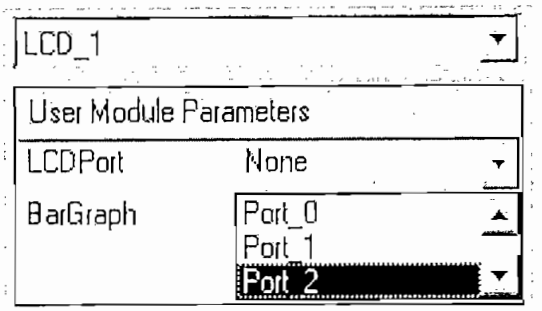
Con lo que se obtiene:



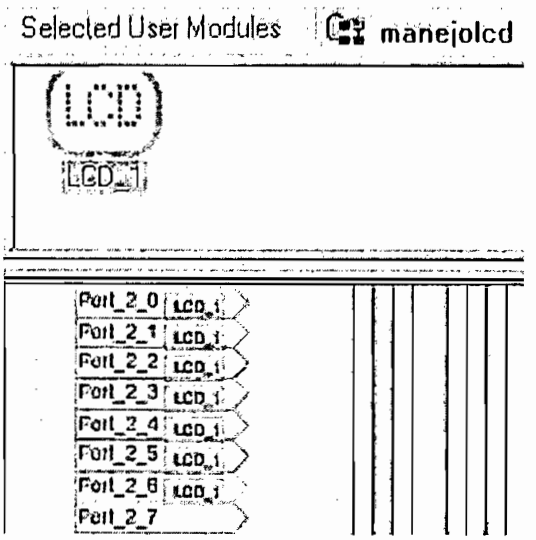
Cuando el módulo deseado ha sido seleccionado definitivamente, se da un doble click al mismo para luego encontrarlo y manejarlo en la vista de interconexión.



El siguiente paso es enrolarlo dentro del programa, acción que se realiza simplemente definiendo sus parámetros.

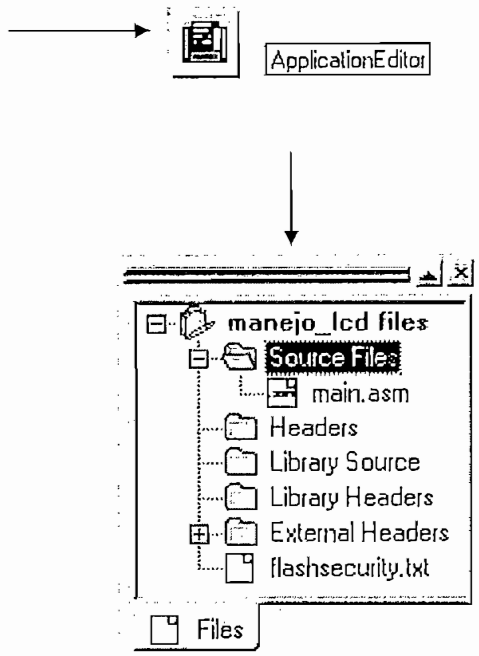


En la gráfica se puede observar que los parámetros del módulo LCD son configurados para que el LCD (periférico), pueda ser conectado en el Puerto 2, tal como se indicó anteriormente en la descripción de la práctica. El resultado finalmente se muestra a continuación:



Cabe notar que en este caso, al Módulo de Usuario LCD no es necesario realizarle el "place", puesto que este no tiene tal propiedad.

Ahora lo que resta, es desarrollar el programa principal *.asm (main)*, para lo cual se hace lo siguiente:

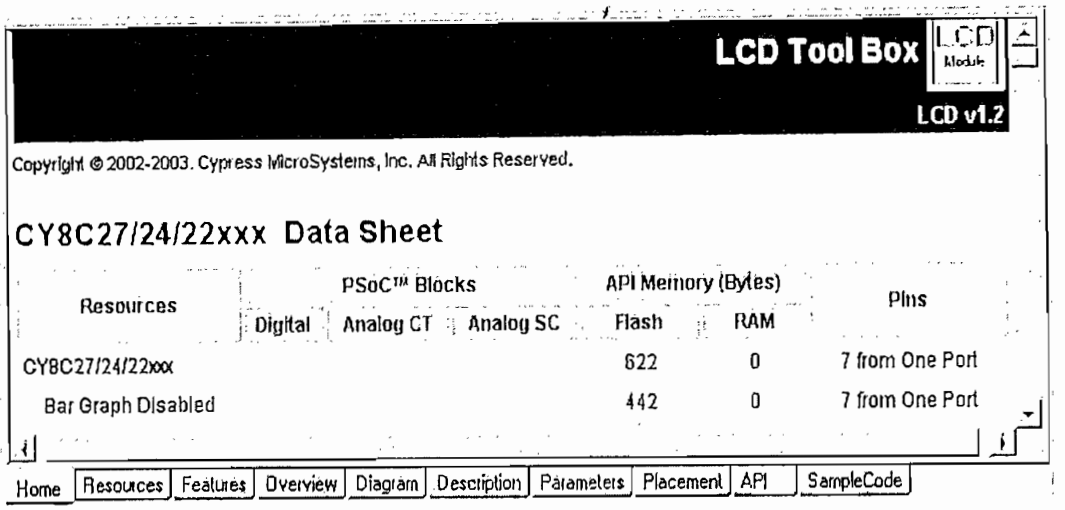


Luego se da un click en la carpeta Source Files, y doble click en el archivo *main.asm.*

Se presenta un espacio en el cual se puede ya escribir el programa principal

```
;-----------------------------------------------------------------------

include "m8c.inc"        ; part specific constants and macros
include "PSoCAPI.inc"    ; PSoC API definitions for all User Modules

export _main

_main:

    ; Insert your main assembly code here.
```

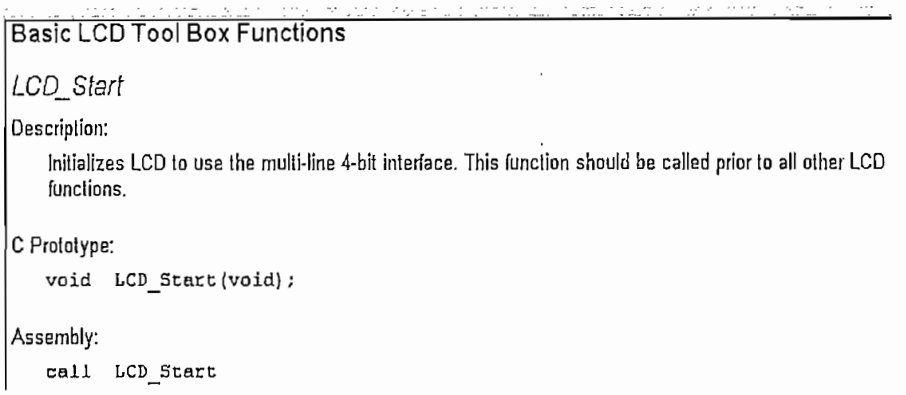
Como también se había visto en el capítulo 2, este módulo posee un Datasheet, el cual puede obtenerse abriendo el archivo PDF respectivo. Para esto, se da un click sobre el módulo escogido:



Ahora seleccionando:

→ API

Se obtienen las rutinas de programa API, provistas para el módulo LCD, tanto en assembler como en C:



Basic LCD Tool Box Functions

LCD_Start

Description:

Initializes LCD to use the multi-line 4-bit interface. This function should be called prior to all other LCD functions.

C Prototype:

```
void  LCD_Start(void);
```

Assembly:

```
call  LCD_Start
```

En esta gráfica se presenta solo una de ellas, pero el usuario es quien escoge estas rutinas de acuerdo a sus necesidades.

Para este ejercicio se utilizan las siguientes rutinas: *LCD_Start, LCD_Position y LCD_ PrCString.*

Finalmente el programa en assembler para el manejo de un LCD, queda de la siguiente manera:

```
#include <m8c.h>        // parte específica, constantes y macros
#include "PSoCAPI.h"    // definiciones PSOC API para todos los Módulos de Usuario

include "lcd_1.inc"
export _main

        area text(ROM,REL)
_main:
        call LCD_1_Start
        mov A,00h
        mov X,03h
        call LCD_1_Position
        mov A,>THE_STR      ; cargar el puntero al string de la ROM
        mov X,<THE_STR
        call LCD_1_PrCString
loop:
        jmp loop
        . LITERAL
        THE_STR:
        DS "hola mundo"
        DB 00h
        ENDLITERAL
```

Para las siguientes aplicaciones el procedimiento es el mismo, la diferencia radica en la mayor cantidad de Módulos de Usuario que se manejan.

4.4 USO DEL CONVERSOR ANÁLOGO DIGITAL DE 12 BITS

Este ejercicio se centra, en el uso de un Conversor Análogo Digital (para este caso de 12 bits), el cual se lo configura internamente por medio del Módulo de Usuario correspondiente.

El objetivo de esta práctica es lograr que una señal análoga (0 - 5 Vdc), que debe ingresar por el Puerto 2 (pin P2.1, configurado como entrada análoga), sea leída y luego convertida en un valor digital de doce bits.

Este valor se presenta en doce Leds conectados en los Puertos 0 y 1, respectivamente; los cuales se configuran como salidas (pines P0.0 - P0.7 y P1.0 - P1.3).

Los periféricos utilizados son:

- Potenciómetro.
- Doce leds.

El Módulo de Usuario que se utiliza en esta aplicación es:

- ADCINC12 (ADC Incremental de 12 bits).

ADCINC12_1

4.4.1 DIAGRAMA CIRCUITAL

En la Figura 4.8 se ilustra el circuito utilizado para esta aplicación.

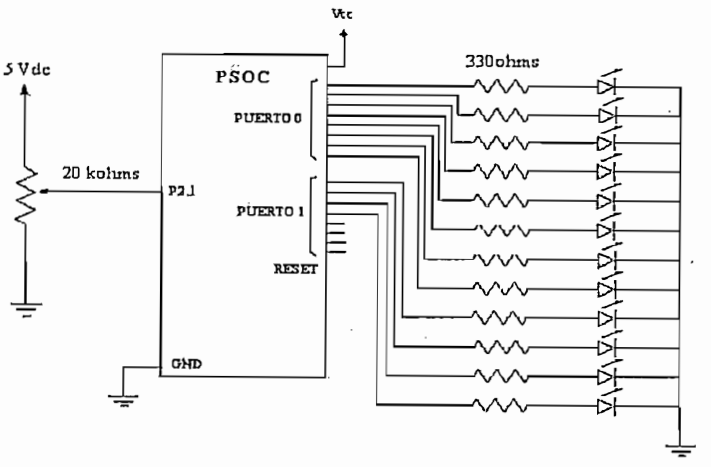


Figura 4.8 Diagrama de conexiones implementado para uso del conversor A-D de 12 bits

4.4.2 DIAGRAMA DE FLUJO

El Diagrama de Flujo mostrado en la Figura 4.9 explica las funciones que el microcontrolador PSOC CY8C27643 realiza en esta aplicación:

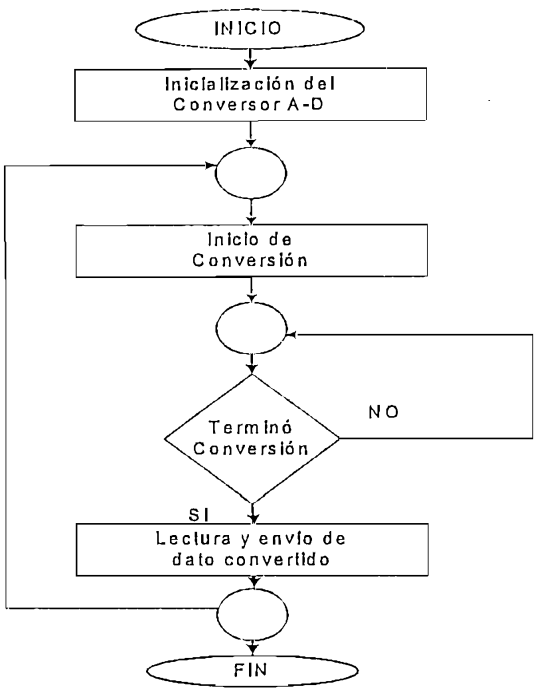


Figura 4.9 Diagrama de Flujo para uso del conversor A-D de 12 bits

A continuación se presenta en lenguaje estructurado, el desarrollo del programa.

Inicialización del Conversor A-D

- *Configurar nivel de potencia que maneja el A-D.*
- *Inicializar A-D en modo continuo.*

Fin Tarea

Inicio de Conversión

- *Arrancar el A-D para empezar la conversión del dato.*
- *Limpiar flag de fin de conversión del dato.*

Fin Tarea

Terminó Conversión

Sí:

Lectura y envío de dato convertido

- *Leer dato ya convertido.*
- *Enviar este dato a los Puertos 0 y 1, para su visualización.*

Fin Tarea

No:

- *Sigue chequeando si la conversión terminó.*

4.5 DISEÑO DE UN AMPLIFICADOR DIFERENCIAL

Este ejercicio representa una introducción al Diseño de Amplificadores a partir de los Módulos de Usuario con los que cuenta el PSOC Designer.

La simetría del amplificador diferencial permite convertir los voltajes de entrada de dos señales análogas, en voltajes de salida en modo común y modo diferencial.

Los voltajes de entrada en este caso, ingresan por el Puerto 0 (pines P0.1 y P0.4, configurados como entradas análogas).

Si se introducen señales iguales, la señal en modo común que se obtiene por el Puerto 0 (pin P0.5 configurado como salida análoga) será cero.

Si se introducen dos señales de voltajes distintos, su diferencia se amplificará de acuerdo a la ganancia con la que se configura globalmente al amplificador. La señal en modo diferencial se obtiene por el Puerto 0 (pin P0.3 configurado como salida análoga).

El periférico utilizado es:

- Conectores BNC.

Los Módulos de Usuario que se utilizan en esta aplicación son:

- PGA (Amplificadores de ganancia programable).

PGA_1

- SCBLOCK (Bloque Switched Capacitor análogo).

SCBLOCK_1

4.5.1 DIAGRAMA CIRCUITAL

En la Figura 4.10 se ilustra el circuito utilizado para esta aplicación.

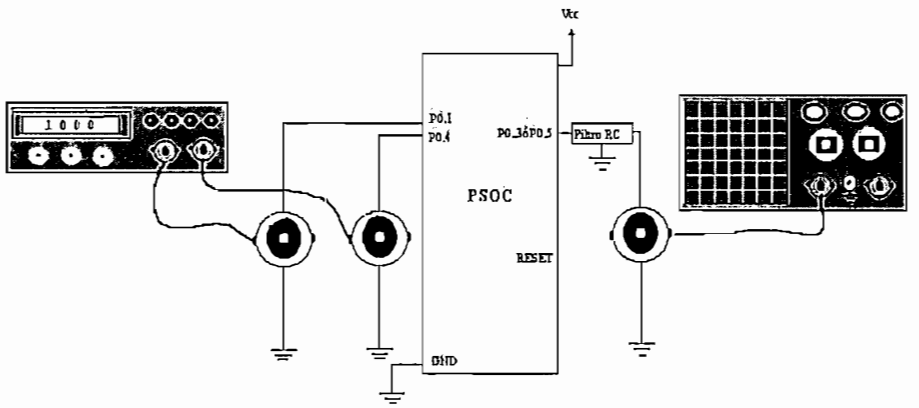


Figura 4.10 Diagrama de conexiones implementado para el Amplificador Diferencial

Configuración para Amplificador Inversor

- *Configuración del nivel de potencia del Switched Capacitor.*
- *Configuración de los parámetros del Switched Capacitor, para que trabaje como amplificador inversor.*
- *Habilitar e iniciar su funcionamiento.*

Fin Tarea

Configuración para Amplificador Diferencial

- *Configuración del nivel de potencia del Switched Capacitor, y de sus parámetros, para que trabaje como un amplificador diferencial.*
- *Habilitar e iniciar su funcionamiento.*

Fin Tarea

Configuración para Amplificador en Modo Común

- *Configuración del nivel de potencia del Switched Capacitor, y de sus parámetros, para que trabaje como un amplificador en modo común.*
- *Habilitar e iniciar su funcionamiento.*

Fin Tarea

4.6 IMPLEMENTACIÓN DE FILTROS

En este ejercicio se emplean las herramientas internas que posee el PSOC Designer para la implementación y diseño de filtros; las características de estos se dan de acuerdo a las necesidades de cada diseñador.

Primeramente se define la topología que el filtro usará, lo cual involucra las conexiones disponibles para las columnas análogas.

El siguiente paso es escoger los parámetros necesarios para el diseño; es decir, tipo de filtro, valor de frecuencia de corte, valor de ganancia, valores de los capacitores requeridos, y finalmente la localización de los dos polos con los cuales se trabaja.

Para la práctica propuesta, una señal análoga ingresa por el Puerto 2 (pin P2.1 configurado como entrada análoga) la cual internamente es filtrada a la frecuencia de corte deseada, obteniéndose su respuesta por el Puerto 0 (pin P0.3 configurado como salida análoga).

La descripción de la topología y parámetros para el diseño de un filtro, se la encuentra en el ANEXO "D"

El periférico utilizado es:

- Conectores BNC.

Los Módulos de Usuario que se utilizan en esta aplicación son:

- LPF2 (Filtro Pasa-bajos de dos polos).

LPF2_1

- BPF2 (Filtro Pasa-banda de dos polos).

BPF2_1

4.6.1 DIAGRAMA CIRCUITAL

En la Figura 4.12 se ilustra el circuito utilizado para esta aplicación.

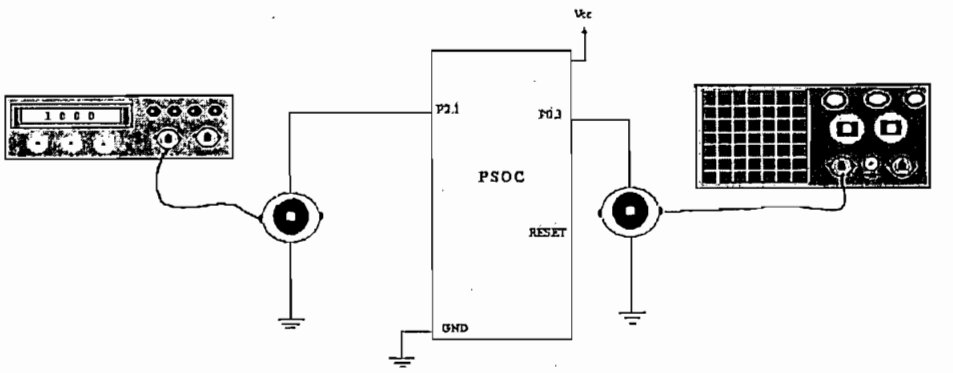


Figura 4.12 Diagrama de conexiones implementado para manejo de filtros

4.6.2 DIAGRAMA DE FLUJO

El Diagrama de Flujo mostrado en la Figura 4.13 explica las funciones que el microcontrolador PSOC CY8C27643 realiza en esta aplicación:

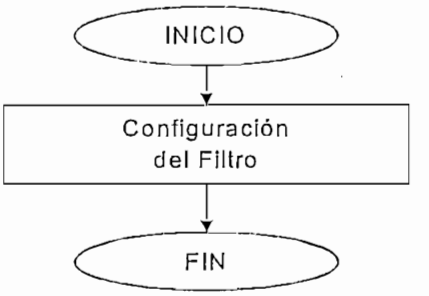


Figura 4.13 Diagrama de Flujo para manejo de filtros

A continuación se presenta en lenguaje estructurado, el desarrollo del programa.

Configuración del Filtro

- *Configurar nivel de potencia para el filtro.*
- *Configurar tipo de filtro a usarse.*
- *Configurar características del filtro.*
- *Habilitar e iniciar el funcionamiento del filtro.*

Fin Tarea

4.7 GENERADOR DE SONIDO

El objetivo de esta práctica es desarrollar un generador de sonido capaz de producir ocho tonos diferentes, los cuales se emitirán a través de un zumbador de 12V conectado al Puerto 0 (pin P0.1 configurado como salida). Los diferentes tonos se producen luego de presionar cualquiera de los ocho pulsantes incorporados para este propósito, los cuales están conectados al Puerto 2 (pines configurados por defecto como entradas High-Z).

El microcontrolador debe detectar que pulsante fue presionado, para así proceder a cargar los valores del periodo y ancho de pulso del PWM, de acuerdo a la frecuencia que cada tono genera. La tabla de frecuencias para una octava de tonos musicales, se la puede encontrar en el ANEXO "D".

Los periféricos utilizados son:

- Pulsantes
- Zumbador

El Módulo de Usuario que se utiliza en esta aplicación es:

- PWM16 (PWM de 16 bits).

PWM16_1

4.7.1 DIAGRAMA CIRCUITAL

En la Figura 4.14 se ilustra el circuito utilizado para esta aplicación.

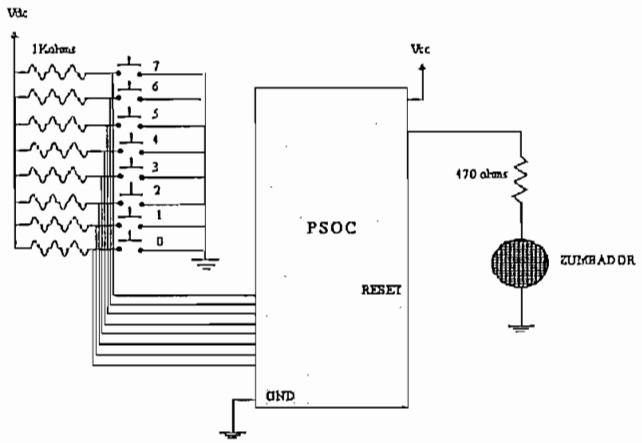


Figura 4.14 Diagrama de conexiones implementado para el generador de sonidos

4.7.2 DIAGRAMA DE FLUJO

El Diagrama de Flujo mostrado en la Figura 4.15 explica las funciones que el microcontrolador PSOC CY8C27643 realiza en esta aplicación:

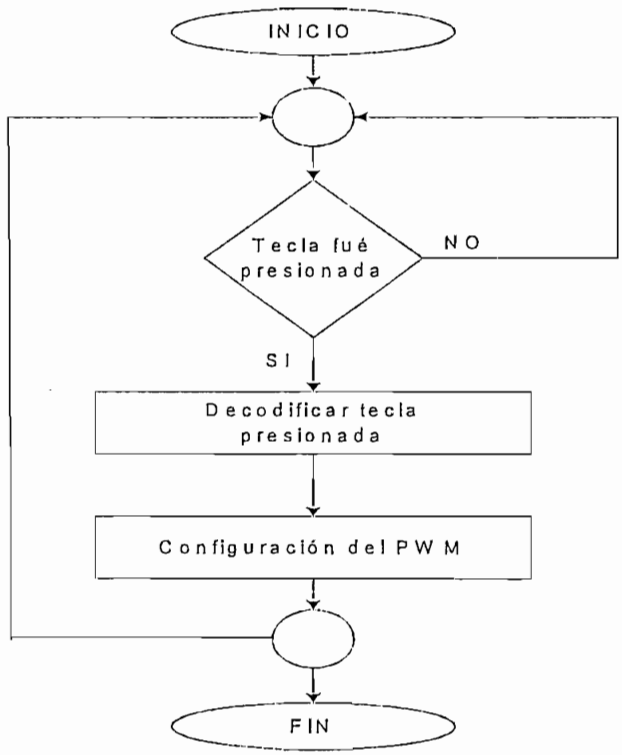


Figura 4.15 Diagrama de Flujo para el generador de sonidos

A continuación se detallará en lenguaje estructurado el desarrollo de esta rutina.

Tecla fue presionada

Si :

Decodificar tecla presionada

- *Lectura del Puerto 2.*
- *Decodificar que tecla fue presionada y saltar a su respectiva configuración.*

Fin Tarea

Configuración del PWM

- *Configurar el Periodo del PWM (frecuencia), de acuerdo a la tecla presionada.*
- *Configurar el ancho de pulso del PWM, de acuerdo a la tecla presionada.*
- *Iniciar el PWM.*

Fin Tarea

No :

- *Sigue chequeando estado de las teclas.*

4.8 MEDICIÓN DE TEMPERATURA USANDO EL SENSOR DALLAS DS1821

El objetivo de esta práctica es medir la temperatura en un ambiente, usando un sensor digital DALLAS DS1821.

Para este propósito se usa una interfaz tipo one-wire, la misma que hace posible la comunicación entre el terminal DQ (I/O de Datos) del dispositivo y el Puerto 2 del PSOC (pin P2.3 configurado como entrada).

La resolución de medición será de un grado centígrado y el valor de temperatura se desplegará en un LCD, el cual se conectará al Puerto 0 (pines P0.0 - P0.6, configurados por el Módulo de Usuario LCD).

La descripción referente a la interfaz one-wire, se la encuentra en el ANEXO "C", en conjunto con el Datasheet del DALLAS DS1821, descrito en el Capítulo 3.

Los periféricos utilizados son:

- Sensor de Temperatura.
- LCD.

El Módulo de Usuario que se utiliza en esta aplicación es:

- LCD.

4.8.1 DIAGRAMA CIRCUITAL

En la Figura 4.16 se ilustra el circuito utilizado para esta aplicación.

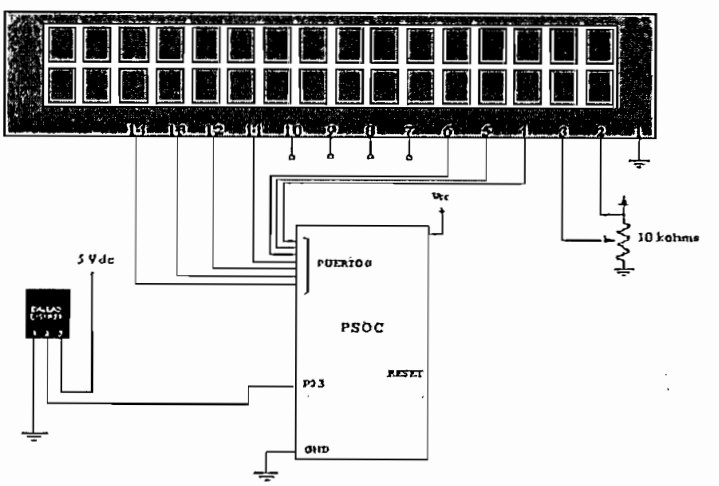


Figura 4.16 Diagrama de conexiones implementado para medición de temperatura con el DS1821

4.8.2 DIAGRAMA DE FLUJO

El Diagrama de Flujo mostrado en la Figura 4.17 explica las funciones que el microcontrolador PSOC CY8C27643 realiza en esta aplicación:

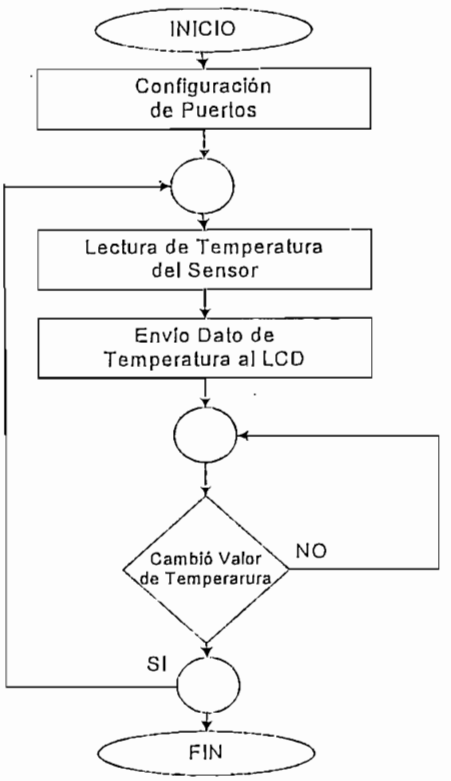


Figura 4.17 Diagrama de Flujo para medición de temperatura con el DALLAS DS1821

A continuación se presenta en lenguaje estructurado, el desarrollo del programa.

Configuración de Puertos

- *Configurar Pull-Up interna para el pin del micro que recibe al terminal DQ del sensor.*
- *Habilitar Pull-Up para comunicación one-wire.*

Fin Tarea

Lectura de Temperatura del Sensor

- *Resetear sensor de temperatura desde el PSOC para iniciar lectura de bits.*
- *Configurar registro de estado del sensor para iniciar conversión.*
- *Iniciar conversión.*
- *Leer valor de temperatura sensado y guardarlo en un registro.*

Fin Tarea

Envío Dato de Temperatura al LCD

- *Convertir dato hexadecimal leído a BCD.*
- *Configurar la posición correcta de cada carácter en el LCD y enviarlo para su visualización.*

Fin Tarea

Cambió Valor de Temperatura

Si :

- *Vuelve a lectura de temperatura del sensor.*

No :

- *Sigue chequeando el cambio de temperatura.*

Fin Tarea

4.9 CONVERSOR VOLTAJE – FRECUENCIA PARA UNA SEÑAL ANÁLOGA

El objetivo de esta práctica es demostrar el principio de funcionamiento de un conversor voltaje-frecuencia.

Se introduce una señal análoga por el Puerto 2 (pin P2.1 configurado como entrada análoga), la cual es procesada por un conversor V/f que está diseñado internamente.

Este proceso da como resultado una señal de onda cuadrada, la misma que se obtiene por el Puerto 2 (pin P2.5 configurado como salida análoga), cuya frecuencia varía en función de la amplitud del voltaje de la señal de entrada.

Por supuesto, la aplicación tiene además como fin demostrar las limitaciones de esta conversión, puesto que los resultados son correctos únicamente hasta ciertos valores de frecuencia.

El periférico utilizado es:

- Conectores BNC.

Los Módulos de Usuario que se utilizan en esta aplicación son:

- CMPPRG (Comparador Threshold programable).

CMPPRG_1

- SCBLOCK (Bloque Switched Capacitor análogo).

SCBLOCK_1

- Timer 8 (Timer de 8 bits).

Timer8_1

4.9.1 DIAGRAMA CIRCUITAL

En la Figura 4.18 se ilustra el circuito utilizado para esta aplicación.

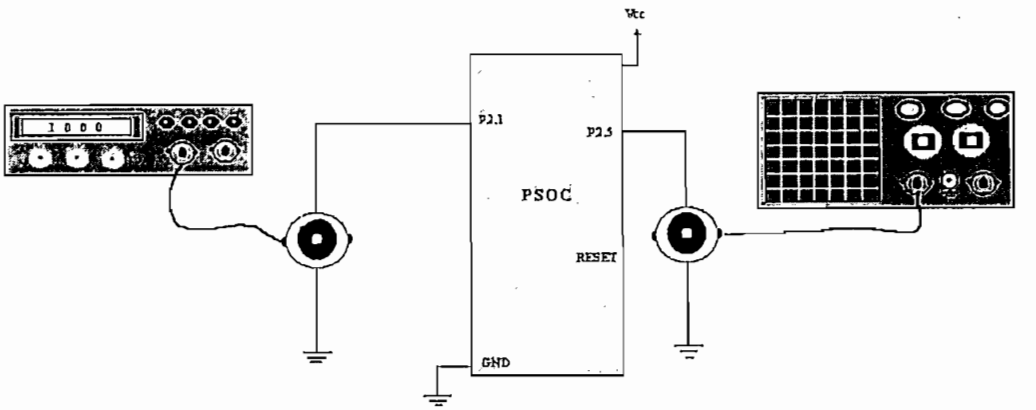


Figura 4.18 Diagrama de conexiones implementado para el conversor V /f de una señal

4.9.2 DIAGRAMA DE FLUJO

El Diagrama de Flujo mostrado en la Figura 4.19 explica las funciones que el microcontrolador PSOC CY8C27643 realiza en esta aplicación:

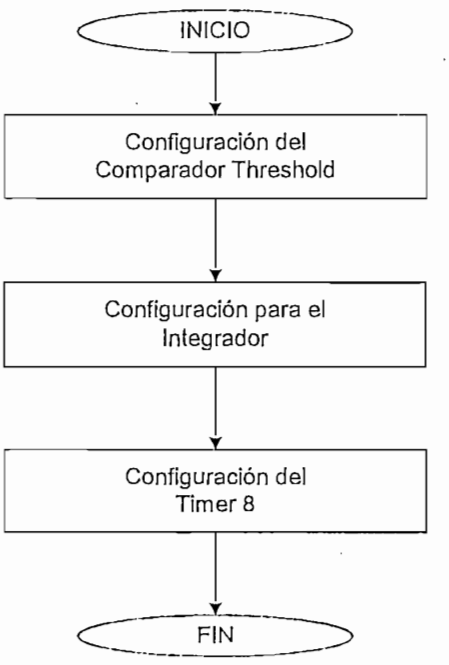


Figura 4.19 Diagrama de Flujo para el conversor V /f de una señal

A continuación se presenta en lenguaje estructurado, el desarrollo del programa.

Configuración Comparador Threshold

- *Configuración del nivel de potencia.*
- *Configuración para manejo del bus análogo.*
- *Habilitación del comparador para formación del Schmitt trigger .*

Fin Tarea

Configuración para el Integrador

- *Configuración del nivel de potencia del Switched Capacitor, y de sus parámetros, para que trabaje como integrador.*
- *Habilitar e iniciar su funcionamiento.*

Fin Tarea

Configuración del Timer 8

- *Configuración de los registros del Timer 8.*
- *Habilitar e iniciar su funcionamiento, para entregar la señal del bus comparador a un pin externo.*

Fin Tarea

4.10 MULTIPLICADOR ANÁLOGO DE DOS SEÑALES

El objetivo de esta práctica es implementar un multiplicador análogo de dos señales.

Para la demostración del funcionamiento del mismo, se hará uso del principio de modulación en amplitud. En el principio de modulación es necesaria la entrada de dos señales análogas, una denominada modulante, la cual es de baja frecuencia e ingresa por el Puerto 0 (pin P0.7 configurado como entrada análoga), y otra llamada portadora que debe ser de una frecuencia mucho mayor a la anterior y que ingresa también por el Puerto 0 (pin P0.1 configurado como entrada análoga).

El resultado de la multiplicación de estas dos señales será una onda modulada en amplitud la cual se obtiene por Puerto el 0 (pin P0.4 configurado como salida análoga).

El periférico utilizado es:

- Conectores BNC.

Los Módulos de Usuario que se utilizan en esta aplicación son:

- CMPPRG (Comparador Threshold programable).

- SCBLOCK (Bloque Switched Capacitor análogo).

- PGA (Amplificadores de ganancia programable).

- LPF2 (Filtro Pasa-bajos de dos polos).

- RefMux (Multiplexor de referencia).

- Timer 8 (Timer de 8 bits).

4.10.1 DIAGRAMA CIRCUITAL

En la Figura 4.20 se ilustra el circuito utilizado para esta aplicación.

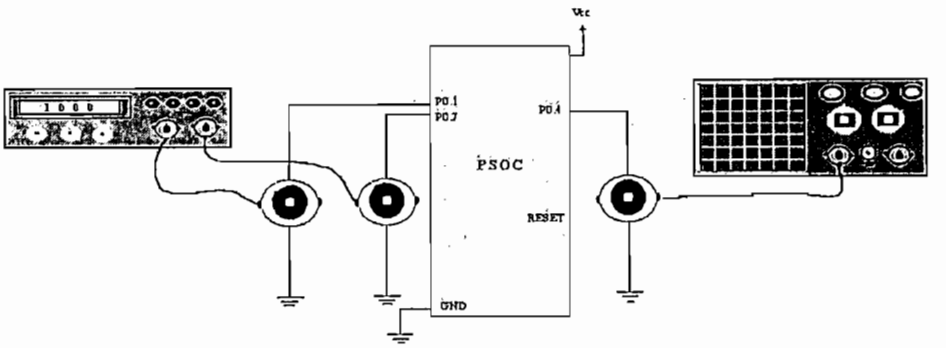


Figura 4.20 Diagrama de conexiones implementado para el multiplicador análogo de dos señales

4.10.2 DIAGRAMA DE FLUJO

El Diagrama de Flujo mostrado en la Figura 4.21 explica las funciones que el microcontrolador PSOC CY8C27643 realiza en esta aplicación:

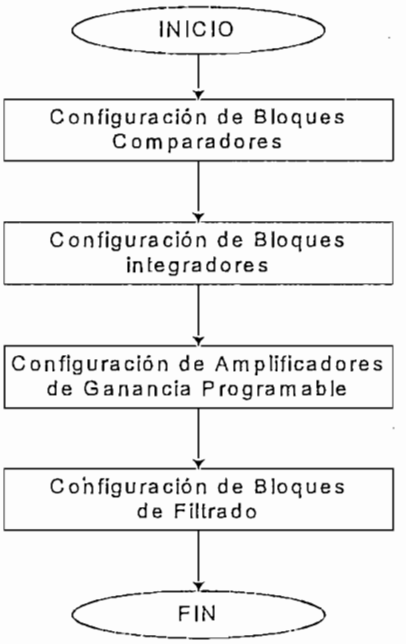


Figura 4.21 Diagrama de Flujo para el multiplicador análogo de dos señales

A continuación se explicará en lenguaje estructurado el desarrollo del programa.

Configuración de Bloques Comparadores

- *Seteo del registro comparador.*
- *Habilitación de comparadores.*

Fin Tarea

Configuración de Bloques Integradores

- *Configuración del integrador y del capacitor conmutado.*
- *Habilitación de estos dos periféricos.*

Fin Tarea

Configuración de Amplificadores de Ganancia Programable

- *Configuración de potencia y ganancia de los PGA.*
- *Habilitación de los PGA.*
- *Configuración del voltaje de referencia.*

Fin Tarea

Configuración de Bloques de Filtrado

- *Configuración de parámetros de los filtros.*
- *Habilitación de filtros.*

Fin Tarea

Los resultados ilustrados en la Figura 4.22 se obtienen a partir de la multiplicación de dos señales sinusoidales:

- Modulante de 103Hz.
- Portadora de 10.6Khz.

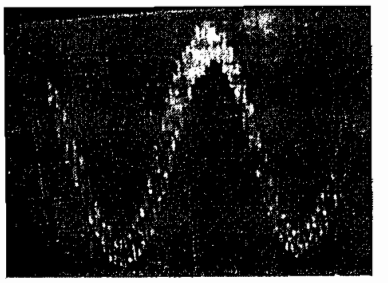


Figura 4.22 Señal resultante de la multiplicación de dos ondas

En la Figura 4.23 se observa esta onda resultante en una diferente escala de tiempo.

Figura 4.23 Señal resultante ampliada de la multiplicación de dos ondas

4.11 COMUNICACIÓN SERIAL CON UN PC

El objetivo de este ejercicio es realizar la transmisión de datos a una velocidad de 115200 bps, desde el PSOC CY8C27643 hacia un computador personal. Para la transmisión de estos datos, se utiliza el protocolo de comunicación RS-232.

Una señal análoga ingresa por el Puerto 2 (pin P2.1 configurado como entrada análoga), la cual es internamente digitalizada a través de un conversor A-D.

Esta señal de nivel TTL que se obtiene por el Puerto 0 (pin P0.0 configurado como salida) es receptada por el MAX232, el cual la convierte en voltajes lógicos para el computador, realizando de esta manera la transmisión de datos.

Los periféricos utilizados son:

- Comunicación Serial RS-232.
- Conectores BNC.

Los Módulos de Usuario que se utilizan en esta aplicación son:

- ADCINC12 (ADC Incremental de 12 bits).

ADCINC12_1

- TX8 (Transmisor serial de 8 bits).

TX8_3

4.11.1 DIAGRAMA CIRCUITAL

En la Figura 4.24 se ilustra el circuito utilizado para esta aplicación.

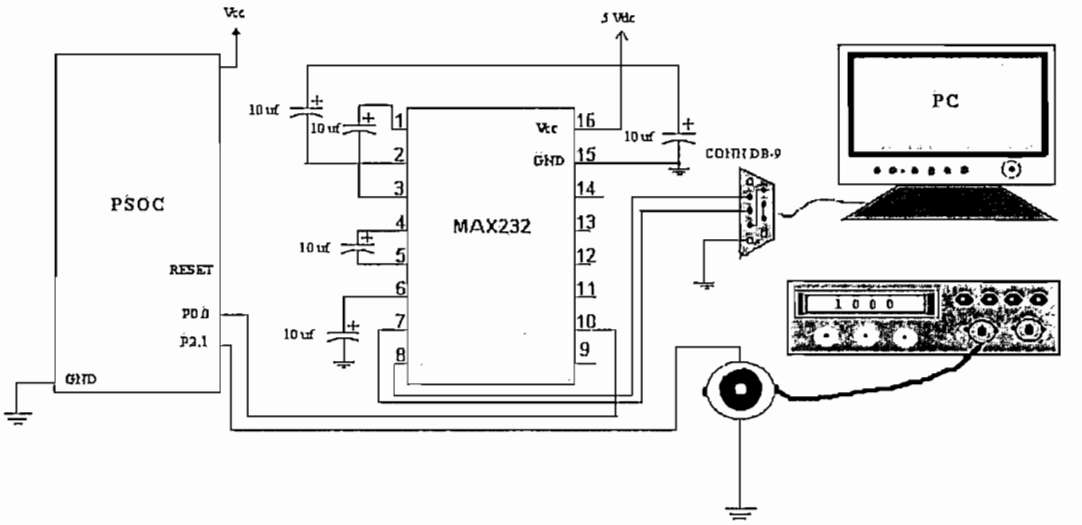


Figura 4.24 Diagrama de conexiones implementado para comunicación serial con un PC

4.11.2 DIAGRAMA DE FLUJO

El Diagrama de Flujo mostrado la Figura 4.25 explica las funciones que el microcontrolador PSOC CY8C27643 realiza en esta aplicación:

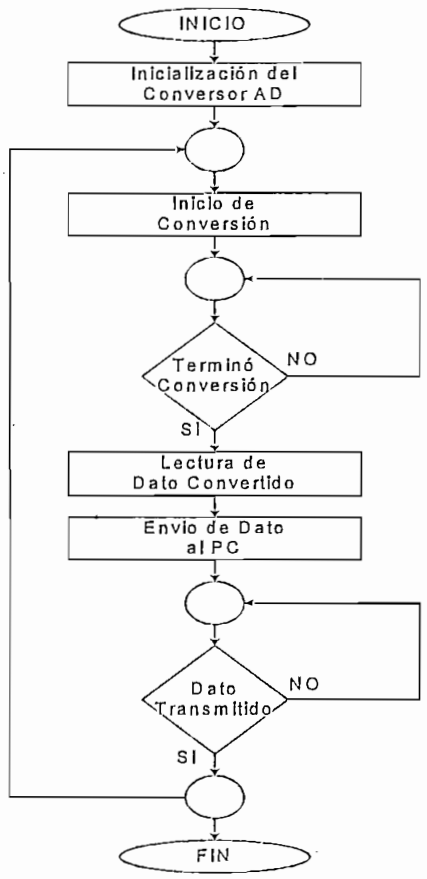


Figura 4.25 Diagrama de Flujo para comunicación serial con un PC

A continuación en lenguaje estructurado se detallará el desarrollo del programa.

Inicialización del Conversor A-D

- *Configurar nivel de potencia que maneja el ADC.*
- *Inicializar el A-D en modo contínuo.*

Fin Tarea

Inicio de Conversión

- *Arrancar el A-C para empezar la conversión de datos.*
- *Limpiar flag de fin de conversión de dato.*

Fin Tarea

Terminó Conversión

Si :

- *Continua con la siguiente tarea.*

No :

- *Sigue chequeando si terminó conversión.*

Fin Tarea

Lectura de Dato convertido

- *Lectura de dato ya convertido.*
- *Guardar dato convertido para posterior adquisición del PC.*

Fin tarea

Envío de Dato al PC

- *Recuperar dato convertido.*
- *Envío de dato al PC por el UART.*

Fin tarea

Dato Transmitido

Si :

- *Vuelve a inicio de conversión.*

No :

- *Sigue chequeando si el dato se transmitió.*

Fin tarea

Para la visualización de los datos adquiridos por el PC, se realizó un programa en Visual Basic, cuyo Diagrama de flujo, es ilustrado en la Figura 4.26.

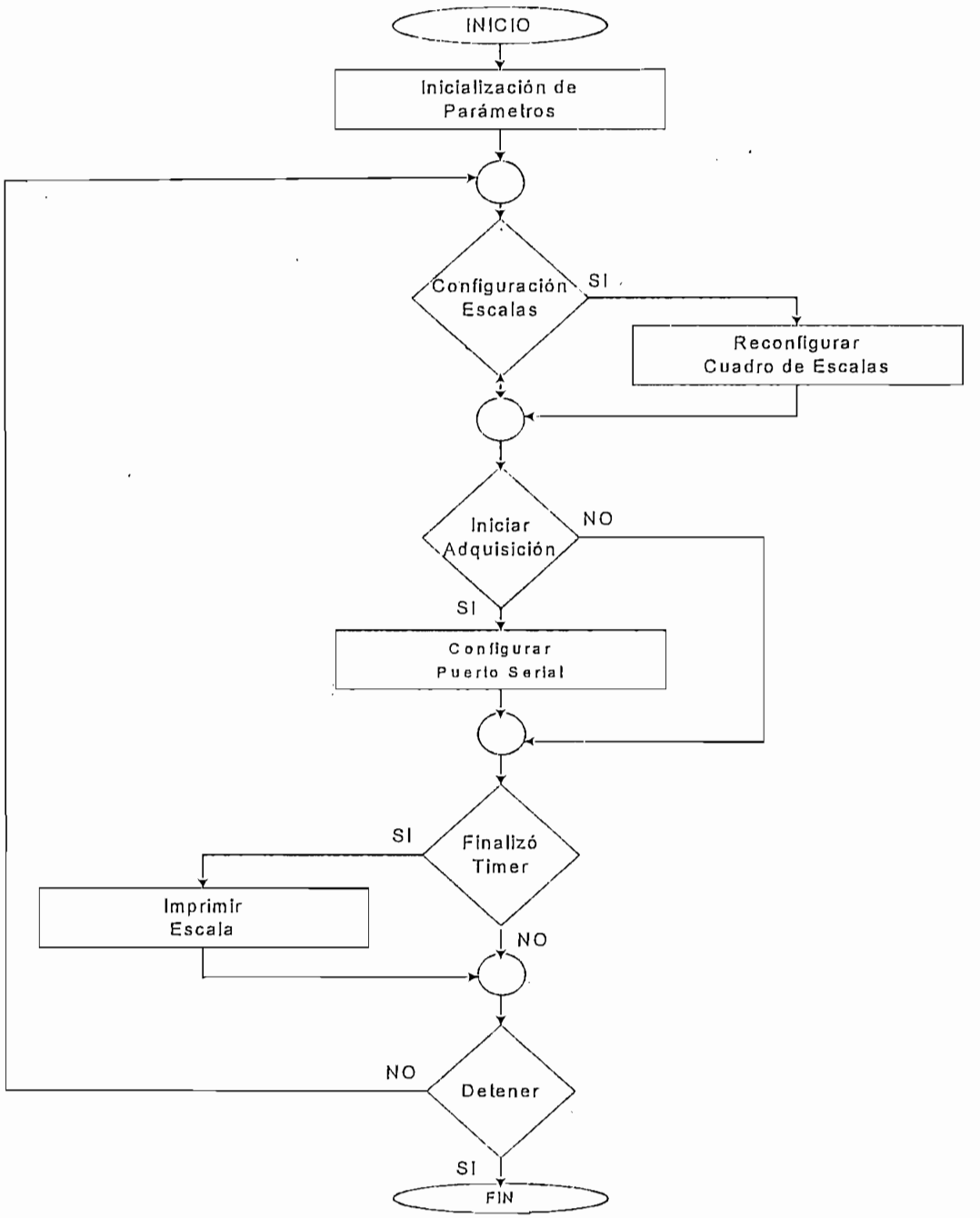


Figura 4.26 Diagrama de Flujo del programa para comunicación serial con un PC en Visual Basic

A continuación en lenguaje estructurado se detallará el desarrollo del programa.

Inicialización de Parámetros

- *Deshabilitar Timer de escalas.*
- *Dibujar marco, ejes y grillas del cuadro de dibujo.*
- *Limpiar cuadros de texto donde se indican tiempos y amplitud de la onda a dibujar.*

Fin Tarea

Configurar Escala

Si :

Reconfigurar Cuadro de Escalas

- *Re dibujar los ejes a la nueva escala.*
- *Re dibujar la grilla a la nueva escala.*
- *Re dibujar marcos en la nueva escala.*

Fin Tarea

No :

Continua

Iniciar Adquisición

Si :

Configurar Puerto Serial

- *Configurar número de Puerto Serial a usar.*
- *Configurar parámetros de transmisión serial.*
- *Abrir puerto serial a usar.*
- *Habilitar timer de escalas.*

Fin Tarea

No :

Continua

Finalizó Timer

Si :

Imprimir Escalas

- *Calcula las escalas usadas para mostrar tiempo y amplitud de la señal a dibujar.*
- *Imprimir escalas en el cuadro de dibujo.*

Fin Tarea

No :

Continua

Detener

No :

Reinicia

Si :

Fin

Los resultados mostrados en la Figura 4.27 corresponden a la adquisición de una onda senoidal, de 1.2Khz.

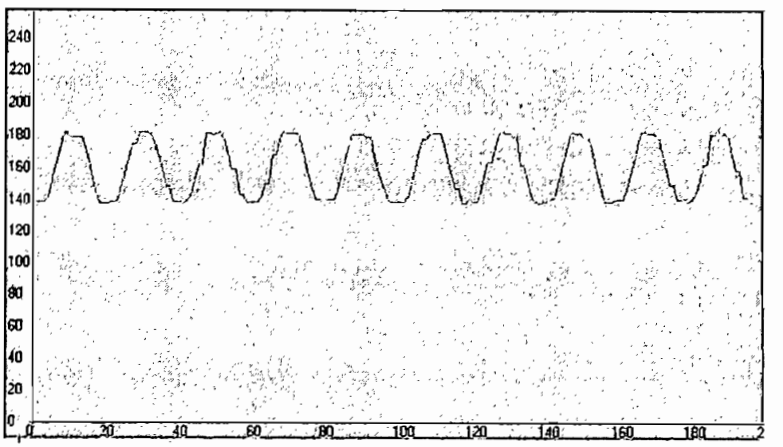


Figura 4.27 Adquisición de una onda senoidal

El archivo fuente de todas las prácticas, incluido el de Visual Basic, se los encuentra en el ANEXO "D".

Después de haber culminado con el desarrollo de las prácticas propuestas para el uso del Módulo Didáctico, en el siguiente capítulo se extraen las conclusiones y recomendaciones, que en base a las experiencias adquiridas durante la realización de este proyecto, se han logrado definir.

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

CAPÍTULO 5

CONCLUSIONES Y RECOMENDACIONES

En este capítulo se presentan las conclusiones y recomendaciones generadas a partir de las experiencias obtenidas en el Diseño y Construcción del Módulo Didáctico con el microcontrolador PSOC CY8C27643.

5.1 CONCLUSIONES

- En este proyecto se pone de manifiesto la introducción a una nueva tecnología en cuanto a microprocesadores, ya que el PSOC presenta diferencias muy marcadas con respecto a micros tradicionales que actualmente están en uso. Estas diferencias se convierten en ventajas competitivas en el medio (velocidad, eficiencia, desempeño, facilidad de desarrollo, reducción de espacio físico y sobre todo un reducido costo económico), las mismas que a lo largo de la implementación de este trabajo fueron comprobadas y cristalizadas, principalmente al corroborar que el manejo de componentes discretos externos es mínimo.

- La estructura del Módulo Didáctico es muy versátil para su manejo; incluye los periféricos estrictamente necesarios, con los cuales se logra desarrollar aplicaciones básicas, que permiten un fácil y eficaz aprendizaje. Además su diseño en etapas independientes, deja abierta la posibilidad de implementar futuros proyectos de acuerdo a las necesidades de cada diseñador.

- El diseño del hardware evita la manipulación constante del dispositivo al momento de la carga y ejecución del software correspondiente a cada aplicación, protegiendo así al PSOC de posibles daños.

- Para el funcionamiento del PSOC CY8C27643, no necesariamente se debe utilizar un cristal externo. Se comprobó en este proyecto que es suficiente configurar el cristal internamente.

- En las prácticas análogas propuestas, se debe indicar que los resultados son satisfactorios hasta un valor de frecuencia determinado. En el conversor voltaje-frecuencia, valores hasta 10Khz aproximadamente, en el multiplicador análogo valores alrededor de 100Hz para la modulante y 10KHz para la portadora. Esto se da debido a las limitaciones de las características físicas de los componentes internos del microcontrolador.

- En la práctica referente a la adquisición de datos, la señal análoga sinusoidal enviada, es visible para valores de frecuencia desde 950Hz hasta 1.5Khz, aproximadamente. Esto se da debido a la velocidad de muestreo que proporciona el conversor y además porque el PSOC es un dispositivo que realiza una tarea a la vez.

- Es importante tomar en cuenta las altas velocidades a las que puede transmitir el dispositivo PSOC.

- Los Módulos de Usuario que se encuentran dentro de la programación gráfica en el PSOC Designer, resultaron efectivamente ser muy útiles por su velocidad de configuración, modificación y puesta a prueba.

- Una de las limitaciones del PSOC es que su voltaje de operación es sólo positivo (hasta 5.25 Vdc), lo cual influye en el desempeño de sus módulos.

5.2 RECOMENDACIONES

- El microcontrolador PSOC CY8C27643 con el que se trabaja, es de tecnología superficial, característica que dificultó en un principio la realización de las diferentes pruebas, debido al reducido espacio existente entre los pines.

  La solución para eliminar este problema fue incorporar el chip a una tarjeta adicional, la cual facilita el manejo de tipo didáctico que se quiere dar a este módulo. Se recomienda seguir este ejemplo en caso se necesite reproducir más módulos.

- Para el correcto funcionamiento de los Módulos de Usuario, se recomienda enrolarlos adecuadamente en el programa y además configurar los parámetros que sean necesarios para una determinada tarea. Para esto se recomienda también, al menos en un principio, consultar el Data Sheet correspondiente.

- Se debe tener precaución al momento de realizar las conexiones externas desde los diferentes periféricos hacia la etapa de control, ya que un incorrecto cableado puede causar fallos en la operación de los dispositivos durante el desarrollo de la práctica y/o dañar el PSOC.

- Siempre será recomendable verificar el nivel de voltaje que proporciona la fuente de alimentación, antes de someter al PSOC a las diferentes pruebas que demanda el módulo.

- En caso de resets involuntarios, se recomienda colocar un capacitor de tantalio de 0.1uf entre VCC y GND, con el fin de estabilizar el sistema.

- En base al estudio y desarrollo de este proyecto que dio prioridad al sistema análogo, se recomienda continuar con el mismo, pero esta vez centrado a un proyecto puntual donde se explore y exija al máximo la versatilidad del PSOC.

  A continuación se listan futuros proyectos que podrían ser implementados.

  - Diseño y construcción de un sistema de control de temperatura usando un RTD (PT-100).

  - Control de un motor a pasos de potencia desde un PC.

  - Diseño de una aplicación basada en redes neuronales.

  - Control inteligente de luces.

- Es válido recomendar al usuario de este sistema, un previo estudio del uso y manejo del módulo, para evitar así contratiempos durante la ejecución de las prácticas propuestas en el proyecto.

- Se recomienda a la E.P.N. adquirir elementos de esta clase, ya que así se facilitaría el desarrollo de nuevas prácticas. Esto debido a las dificultades existentes en el mercado local para la adquisición de este tipo de dispositivos.

- Existen también en el mercado, microcontroladores PSOC de tecnología PDIP, por lo tanto es recomendable usarlos durante la ejecución de nuevos proyectos para así, facilitar su manejo.

REFERENCIAS BIBLIOGRÁFICAS

REFERENCIAS BIBLIOGRÁFICAS

- **PSOC™ Mixed Signal Array:** Preliminary Data Sheet, Document #:38-12012. Cypress MicroSystems.

- **PSOC™ Designer:** Assembly Language User Guide. Cypress MicroSystems.

- **PSOC™ Designer:** C Languaje Compiler User Guide. Cypress MicroSystems.

- **PSOC Prog-Manual:** Programmer User Guide. Mikroelektronika Development Systems.

- **Data Sheet for DS1821:** Programmable Digital Thermostat and Thermometer. Dallas Semiconductor.

- **COUGHLIN, Robert; DRISCOLL, Frederick**. Amplificadores Operacionales y Circuitos Integrados. Quinta Edición. Editorial Prentice Hall. México 1999.

- **PERRY,Gray**. Aprendiendo Visual Basic 6.0. Primera Edición. Editorial Prentice Hall. México. 1999

REFERENCIAS ADICIONALES

- http://www.cypress.com
- http://www.mikroelektronica.co.yu
- http://www.maxim-ic.com
- http://www.digikey.com
- http://www.arrow.com

ANEXOS

ANEXO A

TABLAS DE REGISTROS DE LOS BLOQUES DEL PSOC CY8C27643.

Tabla de los Registros del Núcleo

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
M8C REGISTERS										
M8C Register										
x,F7h	CPU_F				IOX		Carry	Zero	GIE	RL : 00
Related Registers										
1,ECh	OSC_CR0	32k Select	PLL Mode	No Buzz	Sleep[1:0]		CPU Speed[2:0]			RW : 00
x,FFh	CPU_SCR0	GIES		WDRS	PORS	Sleep			STOP	W : 17
SUPERVISORY ROM (SROM) REGISTER										
x,FEh	CPU_SCR1								IRAMDIS	RW : 00
INTERRUPT CONTROLLER REGISTERS										
0,DAh	INT_CLR0	VC3	Sleep	GPIO	Analog 3	Analog 2	Analog 1	Analog 0	V Monitor	RW : 00
0,DBh	INT_CLR1	DCB13	DCB12	DBB11	DBB10	DCB03	DCB02	DBB01	DBB00	RW : 00
0,DDh	INT_CLR3								I2C	RW : 00
0,DEh	INT_MSK3	ENSWINT							I2C	RW : 00
0,E0h	INT_MSK0	VC3	Sleep	GPIO	Analog 3	Analog 2	Analog 1	Analog 0	V Monitor	RW : 00
0,E1h	INT_MSK1	DCB13	DCB12	DBB11	DBB10	DCB03	DCB02	DBB01	DBB00	RW : 00
0,E2h	INT_VC	Pending Interrupt[7:0]								RC : 00
x,F7h	CPU_F				IOX		Carry	Zero	GIE	RL : 00
GENERAL PURPOSE IO (GPIO) REGISTERS										
0,00h	PRT0DR	Data Input[7:0]								RW : 00
0,01h	PRT0IE	Interrupt Enables[7:0]								RW : 00
0,02h	PRT0GS	Global Select[7:0]								RW : 00
0,03h	PRT0DM2	Drive Mode 2[7:0]								RW : FF
1,00h	PRT0DM0	Drive Mode 0[7:0]								RW : 00
1,01h	PRT0DM1	Drive Mode 1[7:0]								RW : FF
1,02h	PRT0IC0	Interrupt Control 0[7:0]								RW : 00
1,03h	PRT0IC1	Interrupt Control 1[7:0]								RW : 00
0,04h	PRT1DR	Data Input[7:0]								RW : 00
0,05h	PRT1IE	Interrupt Enables[7:0]								RW : 00
0,06h	PRT1GS	Global Select[7:0]								RW : 00
0,07h	PRT1DM2	Drive Mode 2[7:0]								RW : FF
1,04h	PRT1DM0	Drive Mode 0[7:0]								RW : 00
1,05h	PRT1DM1	Drive Mode 1[7:0]								RW : FF
1,06h	PRT1IC0	Interrupt Control 0[7:0]								RW : 00
1,07h	PRT1IC1	Interrupt Control 1[7:0]								RW : 00
0,08h	PRT2DR	Data Input[7:0]								RW : 00
0,09h	PRT2IE	Interrupt Enables[7:0]								RW : 00
0,0Ah	PRT2GS	Global Select[7:0]								RW : 00
0,0Bh	PRT2DM2	Drive Mode 2[7:0]								RW : FF
1,08h	PRT2DM0	Drive Mode 0[7:0]								RW : 00
1,09h	PRT2DM1	Drive Mode 1[7:0]								RW : FF
1,0Ah	PRT2IC0	Interrupt Control 0[7:0]								RW : 00
1,0Bh	PRT2IC1	Interrupt Control 1[7:0]								RW : 00
0,0Ch	PRT3DR	Data Input[7:0]								RW : 00
0,0Dh	PRT3IE	Interrupt Enables[7:0]								RW : 00
0,0Eh	PRT3GS	Global Select[7:0]								RW : 00
0,0Fh	PRT3DM2	Drive Mode 2[7:0]								RW : FF
1,0Ch	PRT3DM0	Drive Mode 0[7:0]								RW : 00
1,0Dh	PRT3DM1	Drive Mode 1[7:0]								RW : FF
1,0Eh	PRT3IC0	Interrupt Control 0[7:0]								RW : 00

Tabla de los Registros del Núcleo (continuación)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
1,0Fh	PRT3IC1	Interrupt Control 1[7:0]								RW : 00
0,10h	PRT4DR	Data Input[7:0]								RW : 00
0,11h	PRT4IE	Interrupt Enables[7:0]								RW : 00
0,12h	PRT4GS	Global Select[7:0]								RW : 00
0,13h	PRT4DM2	Drive Mode 2[7:0]								RW : FF
1,10h	PRT4DM0	Drive Mode 0[7:0]								RW : 00
1,11h	PRT4DM1	Drive Mode 1[7:0]								RW : FF
1,12h	PRT4IC0	Interrupt Control 0[7:0]								RW : 00
1,13h	PRT4IC1	Interrupt Control 1[7:0]								RW : 00
0,14h	PRT5DR	Data Input[7:0]								RW : 00
0,15h	PRT5IE	Interrupt Enables[7:0]								RW : 00
0,16h	PRT5GS	Global Select[7:0]								RW : 00
0,17h	PRT5DM2	Drive Mode 2[7:0]								RW : FF
1,14h	PRT5DM0	Drive Mode 0[7:0]								RW : 00
1,15h	PRT5DM1	Drive Mode 1[7:0]								RW : FF
1,16h	PRT5IC0	Interrupt Control 0[7:0]								RW : 00
1,17h	PRT5IC1	Interrupt Control 1[7:0]								RW : 00
ANALOG OUTPUT DRIVER REGISTER										
1,62h	ABF_CR0	ACol1Mux	Acol2Mux	ABUF1EN0	ABUF2EN0	ABUF0EN0	ABUF3EN0	Bypass	PWR	RW : 00
INTERNAL MAIN OSCILLATOR (IMO) REGISTER										
1,E8h	IMO_TR	Trim[7:0]								W : 00
INTERNAL LOW SPEED OSCILLATOR (ILO) REGISTER										
1,E9h	ILO_TR		Disable	Bias Trim[1:0]		Freq Trim[3:0]				W : 00
32 kHz CRYSTAL OSCILLATOR (ECO) REGISTER										
1,EBh	ECO_TR	PSSDC[1:0]								W : 00
PHASE LOCKED LOOP (PLL) REGISTERS										
1,E0h	OSC_CR0	32k Select	PLL Mode	No Buzz	Sleep[1:0]		CPU Speed[2:0]			RW : 00
1,E2h	OSC_CR2	PLLGAIN					EXTCLKEN	IMODIS	SYSCLKX2 DIS	RW : 00
SLEEP AND WATCHDOG REGISTERS										
0,E0h	INT_MSK0	VC3	Sleep	GPIO	Analog 3	Analog 2	Analog 1	Analog 0	V Monitor	RW : 00
0,E3h	RES_WDT	WDSL_Clear[7:0]								W : 00
1,E0h	OSC_CR0	32k Select	PLL Mode	No Buzz	Sleep[1:0]		CPU Speed[2:0]			RW : 00
1,E9h	ILO_TR		Disable	Bias Trim[1:0]		Freq Trim[3:0]				W : 00
1,EBh	ECO_TR	PSSDC[1:0]								W : 00
x,FFh	CPU_SCR0	GIES		WDRS	PORS	Sleep			STOP	W : XX

LEGEND
L: The AND and OR flag instructions can be used to modify this register.
#: Access is bit specific. Refer to register detail for additional information.
X: The value for power on reset is unknown.
x: An "x" before the comma in the address field indicates that this register can be accessed or written to no matter what bank is used.

Registros del M8C

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
M8C Register										
x,F7h	CPU_F				IOX		Carry	Zero	GIE	RL : 00
Related Registers										
1,E0h	OSC_CR0	32k Select	PLL Mode	No Buzz	Sleep[1:0]		CPU Speed[2:0]			RW : 00
x,FF	CPU_SCR0	GIES		WDRS	PORS	Sleep			STOP	W : 17

Registro de la SROM

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
x,FEh	CPU_SCR1								IRAMDIS	RW:00

Registros del Controlador de Interrupción

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
0,DAh	INT_CLR0	VC3	Sleep	GPIO	Analog 3	Analog 2	Analog 1	Analog 0	V Monitor	RW : 00
0,DBh	INT_CLR1	DCB13	DCB12	DBB11	DBB10	DCB03	DCB02	DBB01	DBB00	RW : 00
0,DDh	INT_CLR3								I2C	RW : 00
0,DEh	INT_MSK3	ENSWINT							I2C	RW : 00
0,E0h	INT_MSK0	VC3	Sleep	GPIO	Analog 3	Analog 2	Analog 1	Analog 0	V Monitor	RW : 00
0,E1h	INT_MSK1	DCB13	DCB12	DBB11	DBB10	DCB03	DCB02	DBB01	DBB00	RW : 00
0,E2h	INT_VC	Pending Interrupt[7:0]								RC : 00
x,F7h	CPU_F				IOX		Carry	Zero	GIE	RL : 00

Registros del GPIO

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
0,xxh	PRTxDR	Data Register								RW : 00
0,xxh	PRTxIE	Bit Interrupt Enables								RW : 00
0,xxh	PRTxGS	Global Select								RW : 00
0,xxh	PRTxDM2	Drive Mode 2								RW : FF
1,xxh	PRTxDM0	Drive Mode 0								RW : 00
1,xxh	PRTxDM1	Drive Mode 1								RW : FF
1,xxh	PRTxIC0	Interrupt Control 0								RW : 00
1,xxh	PRTxIC1	Interrupt Control 1								RW : 00

Direccionamiento de bits

IOX	DA[1:0]	Register	Resets to:	(Name)	Function
0	00	PRTxDR	0	DIN	Data
0	01	PRTxIE	0	IE	Interrupt Enable
0	10	PRTxGS	0	BYP	Global Select
0	11	PRTxDM2	1	DM2	Drive Mode, Bit 2
1	00	PRTxDM0	0	DM0	Drive Mode, Bit 0
1	01	PRTxDM1	1	DM1	Drive Mode, Bit 1
1	10	PRTxIC0	0	IM0	Intrpt. Mask, Bit 0
1	11	PRTxIC1	0	IM1	Intrpt. Mask, Bit 1

Pin Drive Modes

Drive Mode DM[2:0]	Pin State	Description
000b	Resistive pull down	Strong high, resistive low
001b	Strong drive	Strong high, strong low
010b	High impedance	Hi-Z high and low, digital input enabled
011b	Resistive pull up	Resistive high, strong low
100b	Open drain high	Slow strong high, Hi-Z low
101b	Slow strong drive	Slow strong high, slow strong low
110b	High impedance, analog (reset state)	Hi-Z high and low, digital input disabled (for zero power) (reset state)
111b	Open drain low	Slow strong low, Hi-Z high

Modos de Interrupción del GPIO

Interrupt Mode IM[1:0]	Description
00b	Bit interrupt disabled, INT0 de-asserted
01b	Assert INT0 when PIN = low
10b	Assert INT0 when PIN = high
11b	Assert INT0 when PIN = change from last read

Registro de los Drivers (manejadores) de Salida Análoga

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
1,62h	ABF_CR0	ACol1Mux	Acol2Mux	ABUF1EN0	ABUF2EN0	ABUF0EN0	ABUF3EN0	Bypass	PWR	RW : 00

Registro del IMO

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
1,E8h	IMO_TR	Trim[7:0]								W : 00

Registro del ILO

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
1,E9h	ILO_TR		Disable	Bias Trim[1:0]		Freq Trim[3:0]				W : 00

Registro del ECO

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
1,EBh	ECO_TR	PSSDC[1:0]								W : 00

Valores de capacitancia para el oscilador (típicos)

Package	Package Capacitance
8 PDIP	0.9 pF
20 PDIP	2.0 pF
20 SOIC	1.0 pF
20 SSOP	0.5 pF
28 PDIP	2.0 pF
28 SOIC	1.0 pF
28 SSOP	0.5 pF
44 TQFP	0.5 pF
48 SSOP	0.6 pF

Registros del PLL

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
1,E0h	OSC_CR0	32k Select	PLL Mode	No Buzz	Sleep[1:0]		CPU Speed[2:0]			RW : 00
1,E2h	OSC_CR2	PLLGAIN					EXTCLKEN	IMODIS	SYSCLKX2 DIS	RW : 00

OSC_CR0[2:0] Bits: CPU Speed

Bits	Internal Main Oscillator	External Clock
000	3 MHz	EXTCLK/ 8
001	6 MHz	EXTCLK/ 4
010	12 MHz	EXTCLK/ 2
011	24 MHz	EXTCLK/ 1
100	1.5 MHz	EXTCLK/ 16
101	750 kHz	EXTCLK/ 32
110	187.5 kHz	EXTCLK/ 128
111	93.7 kHz	EXTCLK/ 256

Registros del Sleep y Watchdog

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
0,E0h	INT_MSK0	VC3	Sleep	GPIO	Analog 3	Analog 2	Analog 1	Analog 0	V Monitor	RW : 00
0,E3h	RES_WDT	WDSL_Clear								W : 00
1,E0h	OSC_CR0	32k Select	PLL Mode	No Buzz	Sleep[1:0]		CPU Speed[2:0]			RW : 00
1,E9h	ILO_TR		Disable	Bias Trim[1:0]		Freq Trim[3:0]				W : 00
1,EBh	ECO_TR	PSSDC[1:0]								W : 00
x,FFh	CPU_SCR0	GIES		WDRS	PORS	Sleep			STOP	W : XX

Selección del Power System Sleep Duty Cycle

PSSDC	Sleep Timer Counts	Period (Nominal)
00 (default)	128	4 ms
01	512	15.6 ms
10	32	1 ms
11	8	250 us

Convenciones de los Registros

Convention	Description
Empty, grayed-out table cell	Illustrates a reserved bit or group of bits.
'x' before the comma in an address	Indicates the register exists in register bank 1 and register bank 2.
'x' in a register name	Indicates that there are multiple instances/address ranges of the same register.
RW	Read and write register or bit(s)
R	Read register or bit(s)
W	Write register or bit(s)
L	Logical register or bit(s)
C	Clearable register or bit(s)
#	Access is bit specific

Registros para el espacio del usuario, MAPA 0

Name	Addr (0,Hex)	Access	Page
PRT0DR	00	RW	88
PRT0IE	01	RW	89
PRT0GS	02	RW	90
PRT0DM2	03	RW	91
PRT1DR	04	RW	88
PRT1IE	05	RW	89
PRT1GS	06	RW	90
PRT1DM2	07	RW	91
PRT2DR	08	RW	88
PRT2IE	09	RW	89
PRT2GS	0A	RW	90
PRT2DM2	0B	RW	91
PRT3DR	0C	RW	88
PRT3IE	0D	RW	89
PRT3GS	0E	RW	90
PRT3DM2	0F	RW	91
PRT4DR	10	RW	88
PRT4IE	11	RW	89
PRT4GS	12	RW	90
PRT4DM2	13	RW	91
PRT5DR	14	RW	88
PRT5IE	15	RW	89
PRT5GS	16	RW	90
PRT5DM2	17	RW	91
	18		
	19		
	1A		
	1B		
	1C		
	1D		
	1E		
	1F		
DBB00DR0	20	#	92
DBB00DR1	21	W	93
DBB00DR2	22	RW	94
DBB00CR0	23	#	95
DBB01DR0	24	#	92
DBB01DR1	25	W	93
DBB01DR2	26	RW	94
DBB01CR0	27	#	95
DCB02DR0	28	#	92
DCB02DR1	29	W	93
DCB02DR2	2A	RW	94
DCB02CR0	2B	#	95
DCB03DR0	2C	#	92
DCB03DR1	2D	W	93
DCB03DR2	2E	RW	94
DCB03CR0	2F	#	95
DBB10DR0	30	#	92
DBB10DR1	31	W	93
DBB10DR2	32	RW	94
DBB10CR0	33	#	95
DBB11DR0	34	#	92
DBB11DR1	35	W	93
DBB11DR2	36	RW	94
DBB11CR0	37	#	95
DCB12DR0	38	#	92
DCB12DR1	39	W	93
DCB12DR2	3A	RW	94
DCB12CR0	3B	#	95
DCB13DR0	3C	#	92
DCB13DR1	3D	W	93
DCB13DR2	3E	RW	94
DCB13CR0	3F	#	95

Name	Addr (0,Hex)	Access	Page
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		
	48		
	49		
	4A		
	4B		
	4C		
	4D		
	4E		
	4F		
	50		
	51		
	52		
	53		
	54		
	55		
	56		
	57		
	58		
	59		
	5A		
	5B		
	5C		
	5D		
	5E		
	5F		
AMX_IN	60	RW	103
	61		
	62		
ARF_CR	63	RW	104
CMP_CR0	64	#	105
ASY_CR	65	#	106
CMP_CR1	66	RW	107
	67		
	68		
	69		
	6A		
	6B		
	6C		
	6D		
	6E		
	6F		
ACB00CR3	70	RW	108
ACB00CR0	71	RW	109
ACB00CR1	72	RW	110
ACB00CR2	73	RW	111
ACB01CR3	74	RW	108
ACB01CR0	75	RW	109
ACB01CR1	76	RW	110
ACB01CR2	77	RW	111
ACB02CR3	78	RW	108
ACB02CR0	79	RW	109
ACB02CR1	7A	RW	110
ACB02CR2	7B	RW	111
ACB03CR3	7C	RW	108
ACB03CR0	7D	RW	109
ACB03CR1	7E	RW	110
ACB03CR2	7F	RW	111

Name	Addr (0,Hex)	Access	Page
ASC10CR0	80	RW	112
ASC10CR1	81	RW	113
ASC10CR2	82	RW	114
ASC10CR3	83	RW	115
ASD11CR0	84	RW	116
ASD11CR1	85	RW	117
ASD11CR2	86	RW	118
ASD11CR3	87	RW	119
ASC12CR0	88	RW	112
ASC12CR1	89	RW	113
ASC12CR2	8A	RW	114
ASC12CR3	8B	RW	115
ASD13CR0	8C	RW	116
ASD13CR1	8D	RW	117
ASD13CR2	8E	RW	118
ASD13CR3	8F	RW	119
ASD20CR0	90	RW	116
ASD20CR1	91	RW	117
ASD20CR2	92	RW	118
ASD20CR3	93	RW	119
ASC21CR0	94	RW	112
ASC21CR1	95	RW	113
ASC21CR2	96	RW	114
ASC21CR3	97	RW	115
ASD22CR0	98	RW	116
ASD22CR1	99	RW	117
ASD22CR2	9A	RW	118
ASD22CR3	9B	RW	119
ASC23CR0	9C	RW	112
ASC23CR1	9D	RW	113
ASC23CR2	9E	RW	114
ASC23CR3	9F	RW	115
	A0		
	A1		
	A2		
	A3		
	A4		
	A5		
	A6		
	A7		
	A8		
	A9		
	AA		
	AB		
	AC		
	AD		
	AE		
	AF		
RDI0RI	B0	RW	120
RDI0SYN	B1	RW	121
RDI0IS	B2	RW	122
RDI0LT0	B3	RW	123
RDI0LT1	B4	RW	124
RDI0RO0	B5	RW	125
RDI0RO1	B6	RW	126
	B7		
RDI1RI	B8	RW	120
RDI1SYN	B9	RW	121
RDI1IS	BA	RW	122
RDI1LT0	BB	RW	123
RDI1LT1	BC	RW	124
RDI1RO0	BD	RW	125
RDI1RO1	BE	RW	126
	BF		

Name	Addr (0,Hex)	Access	Page
	C0		
	C1		
	C2		
	C3		
	C4		
	C5		
	C6		
	C7		
	C8		
	C9		
	CA		
	CB		
	CC		
	CD		
	CE		
	CF		
	D0		
	D1		
	D2		
	D3		
	D4		
	D5		
I2C_CFG	D6	RW	127
I2C_SCR	D7	#	128
I2C_DR	D8	RW	129
I2C_MSCR	D9	#	130
INT_CLR0	DA	RW	131
INT_CLR1	DB	RW	132
	DC		
INT_CLR3	DD	RW	134
INT_MSK3	DE	RW	135
	DF		
INT_MSK0	E0	RW	136
INT_MSK1	E1	RW	137
INT_VC	E2	RC	138
RES_WDT	E3	W	139
DEC_DH	E4	RC	140
DEC_DL	E5	RC	141
DEC_CR0	E6	RW	142
DEC_CR1	E7	RW	143
MUL_X	E8	W	144
MUL_Y	E9	W	145
MUL_DH	EA	R	146
MUL_DL	EB	R	147
ACC_DR1	EC	RW	148
ACC_DR0	ED	RW	149
ACC_DR3	EE	RW	150
ACC_DR2	EF	RW	151
	F0		
	F1		
	F2		
	F3		
	F4		
	F5		
	F6		
CPU_F	F7	RL	152
	F8		
	F9		
	FA		
	FB		
	FC		
	FD		
CPU_SCR1	FE	#	153
CPU_SCR0	FF	#	154

Registros para el espacio de configuración MAPA 1

Name	Addr (1,Hex)	Access	Page
PRT0DM0	00	RW	155
PRT0DM1	01	RW	156
PRT0IC0	02	RW	157
PRT0IC1	03	RW	158
PRT1DM0	04	RW	155
PRT1DM1	05	RW	156
PRT1IC0	06	RW	157
PRT1IC1	07	RW	158
PRT2DM0	08	RW	155
PRT2DM1	09	RW	156
PRT2IC0	0A	RW	157
PRT2IC1	0B	RW	158
PRT3DM0	0C	RW	155
PRT3DM1	0D	RW	156
PRT3IC0	0E	RW	157
PRT3IC1	0F	RW	158
PRT4DM0	10	RW	155
PRT4DM1	11	RW	156
PRT4IC0	12	RW	157
PRT4IC1	13	RW	158
PRT5DM0	14	RW	155
PRT5DM1	15	RW	156
PRT5IC0	16	RW	157
PRT5IC1	17	RW	158
	18		
	19		
	1A		
	1B		
	1C		
	1D		
	1E		
	1F		
DBB00FN	20	RW	159
DBB00IN	21	RW	161
DBB00OU	22	RW	162
	23		
DBB01FN	24	RW	159
DBB01IN	25	RW	161
DBB01OU	26	RW	162
	27		
DCB02FN	28	RW	159
DCB02IN	29	RW	161
DCB02OU	2A	RW	162
	2B		
DCB03FN	2C	RW	159
DCB03IN	2D	RW	161
DCB03OU	2E	RW	162
	2F		
DBB10FN	30	RW	159
DBB10IN	31	RW	161
DBB10OU	32	RW	162
	33		
DBB11FN	34	RW	159
DBB11IN	35	RW	161
DBB11OU	36	RW	162
	37		
DCB12FN	38	RW	159
DCB12IN	39	RW	161
DCB12OU	3A	RW	162
	3B		
DCB13FN	3C	RW	159
DCB13IN	3D	RW	161
DCB13OU	3E	RW	162
	3F		

Name	Addr (1,Hex)	Access	Page
	40		
	41		
	42		
	43		
	44		
	45		
	46		
	47		
	48		
	49		
	4A		
	4B		
	4C		
	4D		
	4E		
	4F		
	50		
	51		
	52		
	53		
	54		
	55		
	56		
	57		
	58		
	59		
	5A		
	5B		
	5C		
	5D		
	5E		
	5F		
CLK_CR0	60	RW	164
CLK_CR1	61	RW	165
ABF_CR0	62	RW	166
AMD_CR0	63	RW	167
	64		
	65		
AMD_CR1	66	RW	168
ALT_CR0	67	RW	169
ALT_CR1	68	RW	170
CLK_CR2	69	RW	171
	6A		
	6B		
	6C		
	6D		
	6E		
	6F		
ACB00CR3	70	RW	108
ACB00CR0	71	RW	109
ACB00CR1	72	RW	110
ACB00CR2	73	RW	111
ACB01CR3	74	RW	108
ACB01CR0	75	RW	109
ACB01CR1	76	RW	110
ACB01CR2	77	RW	111
ACB01CR3	78	RW	108
ACB02CR0	79	RW	109
ACB02CR1	7A	RW	110
ACB02CR2	7B	RW	111
ACB03CR3	7C	RW	108
ACB03CR0	7D	RW	109
ACB03CR1	7E	RW	110
ACB03CR2	7F	RW	111

Name	Addr (1,Hex)	Access	Page
ASC10CR0	80	RW	112
ASC10CR1	81	RW	113
ASC10CR2	82	RW	114
ASC10CR3	83	RW	115
ASD11CR0	84	RW	116
ASD11CR1	85	RW	117
ASD11CR2	86	RW	118
ASD11CR3	87	RW	119
ASC12CR0	88	RW	112
ASC12CR1	89	RW	113
ASC12CR2	8A	RW	114
ASC12CR3	8B	RW	115
ASD13CR0	8C	RW	116
ASD13CR1	8D	RW	117
ASD13CR2	8E	RW	118
ASD13CR3	8F	RW	119
ASD20CR0	90	RW	116
ASD20CR1	91	RW	117
ASD20CR2	92	RW	118
ASD20CR3	93	RW	119
ASC21CR0	94	RW	112
ASC21CR1	95	RW	113
ASC21CR2	96	RW	114
ASC21CR3	97	RW	115
ASD22CR0	98	RW	116
ASD22CR1	99	RW	117
ASD22CR2	9A	RW	118
ASD22CR3	9B	RW	119
ASC23CR0	9C	RW	112
ASC23CR1	9D	RW	113
ASC23CR2	9E	RW	114
ASC23CR3	9F	RW	115
	A0		
	A1		
	A2		
	A3		
	A4		
	A5		
	A6		
	A7		
	A8		
	A9		
	AA		
	AB		
	AC		
	AD		
	AE		
	AF		
RDI0RI	B0	RW	120
RDI0SYN	B1	RW	121
RDI0IS	B2	RW	122
RDI0LT0	B3	RW	123
RDI0LT1	B4	RW	124
RDI0RO0	B5	RW	125
RDI0RO1	B6	RW	126
	B7		
RDI1RI	B8	RW	120
RDI1SYN	B9	RW	121
RDI1IS	BA	RW	122
RDI1LT0	BB	RW	123
RDI1LT1	BC	RW	124
RDI1RO0	BD	RW	125
RDI1RO1	BE	RW	126
	BF		

Name	Addr (1,Hex)	Access	Page
	C0		
	C1		
	C2		
	C3		
	C4		
	C5		
	C6		
	C7		
	C8		
	C9		
	CA		
	CB		
	CC		
	CD		
	CE		
	CF		
GDI_O_IN	D0	RW	172
GDI_E_IN	D1	RW	173
GDI_O_OU	D2	RW	174
GDI_E_OU	D3	RW	175
	D4		
	D5		
	D6		
	D7		
	D8		
	D9		
	DA		
	DB		
	DC		
	DD		
OSC_CR4	DE	RW	176
OSC_CR3	DF	RW	177
OSC_CR0	E0	RW	178
OSC_CR1	E1	RW	179
OSC_CR2	E2	RW	180
VLT_CR	E3	RW	181
VLT_CMP	E4	R	182
	E5		
	E6		
	E7		
IMO_TR	E8	W	183
ILO_TR	E9	W	184
BDG_TR	EA	W	185
ECO_TR	EB	W	186
	EC		
	ED		
	EE		
	EF		
	F0		
	F1		
	F2		
	F3		
	F4		
	F5		
	F6		
CPU_F	F7	RL	152
	F8		
	F9		
	FA		
	FB		
	FC		
	FD		
CPU_SCR1	FE	#	153
CPU_SCR0	FF	#	154

Registros Digitales

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
GLOBAL DIGITAL INTERCONNECT (GDI) REGISTERS										
1,D0h	GDI_O_IN	GOINOUT7	GOINOUT6	GOINOUT5	GOINOUT4	GOINOUT3	GOINOUT2	GOINOUT1	GOINOUT0	RW : 00
1,D1h	GDI_E_IN	GEINOUT7	GEINOUT6	GEINOUT5	GEINOUT4	GEINOUT3	GEINOUT2	GEINOUT1	GEINOUT0	RW : 00
1,D2h	GDI_O_OU	GOOUTIN7	GOOUTIN6	GOOUTIN5	GOOUTIN4	GOOUTIN3	GOOUTIN2	GOOUTIN1	GOOUTIN0	RW : 00
1,D3h	GDI_E_OU	GEOUTIN7	GEOUTIN6	GEOUTIN5	GEOUTIN4	GEOUTIN3	GEOUTIN2	GEOUTIN1	GEOUTIN0	RW : 00
DIGITAL ROW REGISTERS										
x,B0h	RDI0RI	RI3[1:0]		RI2[1:0]		RI1[1:0]		RI0[1:0]		RW : 00
x,B1h	RDI0SYN					RI3SYN	RI2SYN	RI1SYN	RI0SYN	RW : 00
x,B2h	RDI0IS			BCSEL[1:0]		IS3	IS2	IS1	IS0	RW : 00
x,B3h	RDI0LT0	LUT1[3:0]				LUT0[3:0]				RW : 00
x,B4h	RDI0LT1	LUT3[3:0]				LUT2[3:0]				RW : 00
x,B5h	RDI0RO0	GOO5EN	GOO1EN	GOE5EN	GOE1EN	GOO4EN	GOO0EN	GOE4EN	GOE0EN	RW : 00
x,B6h	RDI0RO1	GOO7EN	GOO3EN	GOE7EN	GOE3EN	GOO6EN	GOO2EN	GOE6EN	GOE2EN	RW : 00
x,B8h	RDI1RI	RI3[1:0]		RI2[1:0]		RI1[1:0]		RI0[1:0]		RW : 00
x,B9h	RDI1SYN					RI3SYN	RI2SYN	RI1SYN	RI0SYN	RW : 00
x,BAh	RDI1IS			BCSEL[1:0]		IS3	IS2	IS1	IS0	RW : 00
x,BBh	RDI1LT0	LUT1[3:0]				LUT0[3:0]				RW : 00
x,BCh	RDI1LT1	LUT3[3:0]				LUT2[3:0]				RW : 00
x,BDh	RDI1RO0	GOO5EN	GOO1EN	GOE5EN	GOE1EN	GOO4EN	GOO0EN	GOE4EN	GOE0EN	RW : 00
x,BEh	RDI1RO1	GOO7EN	GOO3EN	GOE7EN	GOE3EN	GOO6EN	GOO2EN	GOE6EN	GOE2EN	RW : 00
DIGITAL BLOCK REGISTERS										
Data and Control Registers										
0,20h	DBB00DR0	Data[7:0]								# : 00
0,21h	DBB00DR1	Data[7:0]								W : 00
0,22h	DBB00DR2	Data[7:0]								# : 00
0,23h	DBB00CR0	Function control/status bits for selected function(7:1)							Enable	# : 00
1,20h	DBB00FN	Data Invert	BCEN	End/Single		Mode[1:0]		Function[2:0]		RW : 00
1,21h	DBB00IN	Data Input[3:0]				Clock Input[3:0]				RW : 00
1,22h	DBB00OU	AUXCLK		AUXEN	AUX IO Select[1:0]		OUTEN	Output Select[1:0]		RW : 00
0,24h	DBB01DR0	Data[7:0]								# : 00
0,25h	DBB01DR1	Data[7:0]								W : 00
0,26h	DBB01DR2	Data[7:0]								# : 00
0,27h	DBB01CR0	Function control/status bits for selected function(7:1)							Enable	# : 00
1,24h	DBB01FN	Data Invert	BCEN	End/Single		Mode[1:0]		Function[2:0]		RW : 00
1,25h	DBB01IN	Data Input[3:0]				Clock Input[3:0]				RW : 00
1,26h	DBB01OU	AUXCLK		AUXEN	AUX IO Select[1:0]		OUTEN	Output Select[1:0]		RW : 00
0,28h	DCB02DR0	Data[7:0]								# : 00
0,29h	DCB02DR1	Data[7:0]								W : 00
0,2Ah	DCB02DR2	Data[7:0]								# : 00
0,2Bh	DCB02CR0	Function control/status bits for selected function(7:1)							Enable	# : 00
1,28h	DCB02FN	Data Invert	BCEN	End/Single		Mode[1:0]		Function[2:0]		RW : 00
1,29h	DCB02IN	Data Input[3:0]				Clock Input[3:0]				RW : 00
1,2Ah	DCB02OU	AUXCLK		AUXEN	AUX IO Select[1:0]		OUTEN	Output Select[1:0]		RW : 00
0,2Ch	DCB03DR0	Data[7:0]								# : 00
0,2Dh	DCB03DR1	Data[7:0]								W : 00
0,2Eh	DCB03DR2	Data[7:0]								# : 00
0,2Fh	DCB03CR0	Function control/status bits for selected function(7:1)							Enable	# : 00
1,2Ch	DCB03FN	Data Invert	BCEN	End/Single		Mode[1:0]		Function[2:0]		RW : 00

Registros Digitales (continuación)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
1,20h	DCB03IN	Data Input[3:0]				Clock Input[3:0]				RW : 00
1,2Eh	DCB03OU	AUXCLK		AUXEN	AUX IO Select[1:0]		OUTEN	Output Select[1:0]		RW : 00
0,30h	DBB10DR0	Data[7:0]								# : 00
0,31h	DBB10DR1	Data[7:0]								W : 00
0,32h	DBB10DR2	Data[7:0]								# : 00
0,33h	DBB10CR0	Function control/status bits for selected function[7:1]							Enable	# : 00
1,30h	DBB10FN	Data Invert	BCEN	End/Single	Mode[1:0]		Function[2:0]			RW : 00
1,31h	DBB10IN	Data Input[3:0]				Clock Input[3:0]				RW : 00
1,32h	DBB10OU	AUXCLK		AUXEN	AUX IO Select[1:0]		OUTEN	Output Select[1:0]		RW : 00
0,34h	DBB11DR0	Data[7:0]								# : 00
0,35h	DBB11DR1	Data[7:0]								W : 00
0,36h	DBB11DR2	Data[7:0]								# : 00
0,37h	DBB11CR0	Function control/status bits for selected function[7:1]							Enable	# : 00
1,34h	DBB11FN	Data Invert	BCEN	End/Single	Mode[1:0]		Function[2:0]			RW : 00
1,35h	DBB11IN	Data Input[3:0]				Clock Input[3:0]				RW : 00
1,36h	DBB11OU	AUXCLK		AUXEN	AUX IO Select[1:0]		OUTEN	Output Select[1:0]		RW : 00
0,38h	DCB12DR0	Data[7:0]								# : 00
0,39h	DCB12DR1	Data[7:0]								W : 00
0,3Ah	DCB12DR2	Data[7:0]								# : 00
0,3Bh	DCB12CR0	Function control/status bits for selected function[7:1]							Enable	# : 00
1,38h	DCB12FN	Data Invert	BCEN	End/Single	Mode[1:0]		Function[2:0]			RW : 00
1,39h	DCB12IN	Data Input[3:0]				Clock Input[3:0]				RW : 00
1,3Ah	DCB12OU	AUXCLK		AUXEN	AUX IO Select[1:0]		OUTEN	Output Select[1:0]		RW : 00
0,3Ch	DCB13DR0	Data[7:0]								# : 00
0,3Dh	DCB13DR1	Data[7:0]								W : 00
0,3Eh	DCB13DR2	Data[7:0]								# : 00
0,3Fh	DCB13CR0	Function control/status bits for selected function[7:1]							Enable	# : 00
1,3Ch	DCB13FN	Data Invert	BCEN	End/Single	Mode[1:0]		Function[2:0]			RW : 00
1,3Dh	DCB13IN	Data Input[3:0]				Clock Input[3:0]				RW : 00
1,3Eh	DCB13OU	AUXCLK		AUXEN	AUX IO Select[1:0]		OUTEN	Output Select[1:0]		RW : 00
Interrupt Mask Register										
0,E1h	INT_MSK1	DCB13	DCB12	DBB11	DBB10	DCB03	DCB02	DBB01	DBB00	RW : 00

LEGEND
#: Access is bit specific. Refer to register detail for additional information.
x: An "X" before the comma in the address field indicates that this register can be accessed or written to no matter what bank is used.

Registros del GDI

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
1,D0h	GDI_O_IN	GOINOUT7	GOINOUT6	GOINOUT5	GOINOUT4	GOINOUT3	GOINOUT2	GOINOUT1	GOINOUT0	RW : 00
1,D1h	GDI_E_IN	GEINOUT7	GEINOUT6	GEINOUT5	GEINOUT4	GEINOUT3	GEINOUT2	GEINOUT1	GEINOUT0	RW : 00
1,D2h	GDI_O_OU	GOOUTIN7	GOOUTIN6	GOOUTIN5	GOOUTIN4	GOOUTIN3	GOOUTIN2	GOOUTIN1	GOOUTIN0	RW : 00
1,D3h	GDI_E_OU	GEOUTIN7	GEOUTIN6	GEOUTIN5	GEOUTIN4	GEOUTIN3	GEOUTIN2	GEOUTIN1	GEOUTIN0	RW : 00

Registros del RDI

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
x,xxh	RDIxRI	RI3[1:0]		RI2[1:0]		RI1[1:0]		RI0[1:0]		RW : 00
x,xxh	RDIxSYN					RI3SYN	RI2SYN	RI1SYN	RI0SYN	RW : 00
x,xxh	RDIxIS			BCSEL[1:0]		IS3	IS2	IS1	IS0	RW : 00
x,xxh	RDIxLT0	LUT1[3:0]				LUT0[3:0]				RW : 00
x,xxh	RDIxLT1	LUT3[3:0]				LUT2[3:0]				RW : 00
x,xxh	RDIxRO0	GOO5EN	GOO1EN	GOE5EN	GOE1EN	GOO4EN	GOO0EN	GOE4EN	GOE0EN	RW : 00
x,xxh	RDIxRO1	GOO7EN	GOO3EN	GOE7EN	GOE3EN	GOO6EN	GOO2EN	GOE6EN	GOE2EN	RW : 00

Registros del Bloque Digital

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
Data and Control Registers										
0,xxh	DxBxxDR0	Data[7:0]								# : 00
0,xxh	DxBxxDR1	Data[7:0]								W : 00
0,xxh	DxBxxDR2	Data[7:0]								# : 00
0,xxh	DxBxxCR0	Function control/status bits for selected function[7:0]							Enable	# : 00
Interrupt Mask Register										
0,E1h	INT_MSK1	DCB13	DCB12	DBB11	DBB10	DCB03	DCB02	DBB01	DBB00	RW : 00
Configuration Registers										
1,xxh	DxBxxFN	Data Invert	BCEN	End/Single	Mode[1:0]		Function[2:0]			RW : 00
1,xxh	DxBxxIN	Data Input[3:0]				Clock Input[3:0]				RW : 00
1,xxh	DxBxxOU	AUXCLK		AUXEN	AUX IO Select[1:0]		OUTEN	Output Select[1:0]		RW : 00

Registros Análogos

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
ANALOG INTERFACE REGISTERS										
0,64h	CMP_CR0	COMP[3]	COMP[2]	COMP1[]	COMP[0]	AINT[3]	AINT[2]	AINT[1]	AINT[0]	RW : 00
0,65h	ASY_CR		SARCNT[2:0]			SARSIGN	SARCOL[1:0]		SYNCEN	RW : 00
0,66h	CMP_CR1	CLDIS[3]	CLDIS[2]	CLDIS[1]	CLDIS[0]					RW : 00
0,E6h	DEC_CR0	IGEN[3:0]				ICLKS0	DCOL[1:0]		DCLKS0	RW : 00
0,E7h	DEC_CR1	ECNT	IDEC			ICLKS1			DCLKS1	RW : 00
1,61h	CLK_CR0	Acolumn3[1:0]		Acolumn2[1:0]		Acolumn1[1:0]		Acolumn0[1:0]		RW : 00
1,62h	CLK_CR1		SHDIS	ACLK1[2:0]			ACLK0[2:0]			RW : 00
1,63h	AMD_CR0		AMOD2[2:0]				AMOD0[2:0]			RW : 00
1,64h	AMD_CR1		AMOD2[2:0]				AMOD1[2:0]			RW : 00
1,67h	ALT_CR0	LUT1[3:0]				LUT0[3:0]				RW : 00
1,68h	ALT_CR1	LUT3[3:0]				LUT2[3:0]				RW : 00
1,69h	CLK_CR2					ACLK1R			ACLK0R	RW : 00
ANALOG INPUT CONFIGURATION REGISTERS										
0,60h	AMX_IN	ACI3[1:0]		ACI2[1:0]		ACI1[1:0]		ACI0[1:0]		RW : 00
1,62h	ABF_CR0	ACol1Mux	ACol2Mux	ABUF1EN	ABUF2EN	ABUF0EN	ABUF3EN	Bypass	PWR	RW : 00
ANALOG REFERENCE REGISTER										
0,62h	ARF_CR		HBE	REF[2:0]			PWR[2:0]			RW : 00
SWITCHED CAPACITOR BLOCK REGISTERS										
Switched Capacitor Block Registers, Type C										
x,80h	ASC10CR0	FCap	ClockPhase	ASign	ACap[4:0]					RW : 00
x,81h	ASC10CR1	ACMux[2:0]			BCap[4:0]					RW : 00
x,82h	ASC10CR2	AnalogBus	CompBus	AutoZero	CCap[4:0]					RW : 00
x,83h	ASC10CR3	ARefMux[1:0]		FSW1	FSW0	BMuxSC[1:0]		PWR[1:0]		RW : 00
x,88h	ASC12CR0	FCap	ClockPhase	ASign	ACap[4:0]					RW : 00
x,89h	ASC12CR1	ACMux[2:0]			BCap[4:0]					RW : 00
x,8Ah	ASC12CR2	AnalogBus	CompBus	AutoZero	CCap[4:0]					RW : 00
x,8Bh	ASC12CR3	ARefMux[1:0]		FSW1	FSW0	BMuxSC[1:0]		PWR[1:0]		RW : 00
x,94h	ASC21CR0	FCap	ClockPhase	ASign	ACap[4:0]					RW : 00
x,95h	ASC21CR1	ACMux[2:0]			BCap[4:0]					RW : 00
x,96h	ASC21CR2	AnalogBus	CompBus	AutoZero	CCap[4:0]					RW : 00
x,97h	ASC21CR3	ARefMux[1:0]		FSW1	FSW0	BMuxSC[1:0]		PWR[1:0]		RW : 00
x,9Ch	ASC23CR0	FCap	ClockPhase	ASign	ACap[4:0]					RW : 00
x,9Dh	ASC23CR1	ACMux[2:0]			BCap[4:0]					RW : 00
x,9Eh	ASC23CR2	AnalogBus	CompBus	AutoZero	CCap[4:0]					RW : 00
x,9Fh	ASC23CR3	ARefMux[1:0]		FSW1	FSW0	BMuxSC[1:0]		PWR[1:0]		RW : 00
Switched Capacitor Block Registers, Type D										
x,84h	ASD11CR0	FCap	ClockPhase	ASign	ACap[4:0]					RW : 00
x,85h	ASD11CR1	AMux[2:0]			BCap[4:0]					RW : 00
x,86h	ASD11CR2	AnalogBus	CompBus	AutoZero	CCap[4:0]					RW : 00
x,87h	ASD11CR3	ARefMux[1:0]		FSW1	FSW0	BSW	BMuxSD	PWR[1:0]		RW : 00
x,8Ch	ASD13CR0	FCap	ClockPhase	ASign	ACap[4:0]					RW : 00
x,8Dh	ASD13CR1	AMux[2:0]			BCap[4:0]					RW : 00
x,8Eh	ASD13CR2	AnalogBus	CompBus	AutoZero	CCap[4:0]					RW : 00
x,8Fh	ASD13CR3	ARefMux[1:0]		FSW1	FSW0	BSW	BMuxSD	PWR[1:0]		RW : 00
x,90h	ASD20CR0	FCap	ClockPhase	ASign	ACap[4:0]					RW : 00
x,91h	ASD20CR1	AMux[2:0]			BCap[4:0]					RW : 00
x,92h	ASD20CR2	AnalogBus	CompBus	AutoZero	CCap[4:0]					RW : 00

Registros Análogos (continuación)

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
x,93h	ASD20CR3	ARefMux[1:0]		FSW1	FSW0	BSW	BMuxSD	PWR[1:0]		RW : 00
x,98h	ASD22CR0	FCap	ClockPhase	ASign	ACap[4:0]					RW : 00
x,99h	ASD22CR1	AMux[2:0]			BCap[4:0]					RW : 00
x,9Ah	ASD22CR2	AnalogBus	CompBus	AutoZero	CCap[4:0]					RW : 00
x,9Bh	ASD22CR3	ARefMux[1:0]		FSW1	FSW0	BSW	BMuxSD	PWR[1:0]		RW : 00
CONTINUOUS TIME BLOCK REGISTERS										
x,70h	ACB00CR3					LPCMPEN	CMOUT	INSAMP	EXGAIN	RW : 00
x,71h	ACB00CR0	RTapMux[3:0]				Gain	RTopMux	RBotMux[1:0]		RW : 00
x,72h	ACB00CR1	AnalogBus	CompBus	NMux[2:0]		PMux[2:0]				RW : 00
x,73h	ACB00CR2	CPhase	CLatch	CompCap	TMUXEN	TestMux[1:0]		PWR[1:0]		RW : 00
x,74h	ACB01CR3					LPCMPEN	CMOUT	INSAMP	EXGAIN	RW : 00
x,75h	ACB01CR0	RTapMux[3:0]				Gain	RTopMux	RBotMux[1:0]		RW : 00
x,76h	ACB01CR1	AnalogBus	CompBus	NMux[2:0]		PMux[2:0]				RW : 00
x,77h	ACB01CR2	CPhase	CLatch	CompCap	TMUXEN	TestMux[1:0]		PWR[1:0]		RW : 00
x,78h	ACB02CR3					LPCMPEN	CMOUT	INSAMP	EXGAIN	RW : 00
x,79h	ACB02CR0	RTapMux[3:0]				Gain	RTopMux	RBotMux[1:0]		RW : 00
x,7Ah	ACB02CR1	AnalogBus	CompBus	NMux[2:0]		PMux[2:0]				RW : 00
x,7Bh	ACB02CR2	CPhase	CLatch	CompCap	TMUXEN	TestMux[1:0]		PWR[1:0]		RW : 00
x,7Ch	ACB03CR3					LPCMPEN	CMOUT	INSAMP	EXGAIN	RW : 00
x,7Dh	ACB03CR0	RTapMux[3:0]				Gain	RTopMux	RBotMux[1:0]		RW : 00
x,7Eh	ACB03CR1	AnalogBus	CompBus	NMux[2:0]		PMux[2:0]				RW : 00
x,7Fh	ACB03CR2	CPhase	CLatch	CompCap	TMUXEN	TestMux[1:0]		PWR[1:0]		RW : 00

LEGEND
x: An "x" before the comma in the address field indicates that this register can be accessed or written to no matter what bank is used.

Registros de la Interfaz Análoga

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
0,64h	CMP_CR0	COMP[3]	COMP[2]	COMP1[]	COMP[0]	AINT[3]	AINT[2]	AINT[1]	AINT[0]	RW : 00
0,65h	ASY_CR		SARCNT[2:0]			SARSIGN	SARCOL[1:0]		SYNCEN	RW : 00
0,66h	CMP_CR1	CLDIS[3]	CLDIS[2]	CLDIS[1]	CLDIS[0]					RW : 00
0,E6h	DEC_CR0	IGEN[3:0]				ICLKS0	DCOL[1:0]		DCLKS0	RW : 00
0,E7h	DEC_CR1	ECNT	IDEC			ICLKS1			DCLKS1	RW : 00
1,60h	CLK_CR0	Acolumn3[1:0]		Acolumn2[1:0]		Acolumn1[1:0]		Acolumn0[1:0]		RW : 00
1,61h	CLK_CR1		SHDIS	ACLK1[2:0]			ACLK0[2:0]			RW : 00
1,62h	AMD_CR0		AMOD2[2:0]				AMOD0[2:0]			RW : 00
1,66h	AMD_CR1		AMOD3[2:0]				AMOD1[2:0]			RW : 00
1,67h	ALT_CR0	LUT1[3:0]				LUT0[3:0]				RW : 00
1,68h	ALT_CR1	LUT3[3:0]				LUT2[3:0]				RW : 00
1,69h	CLK_CR2					ACLK1R			ACLK0R	RW : 00

Registros de la Configuración de Entradas Análogas

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
0,60h	AMX_IN	ACI3[1:0]		ACI2[1:0]		ACI1[1:0]		ACI0[1:0]		RW : 00
1,62h	ABF_CR0	ACd1Mux	ACd2Mux	ABUF1EN	ABUF2EN	ABUF0EN	ABUF3EN	Bypass	PWR	RW : 00

Registro de Referencia Análoga

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
0,62h	ARF_CR		HBE	REF[2:0]			PWR[2:0]			RW : 00

Parámetros de operación para el AGND, RefHi y RefLO

	AGND		RefHI		RefLO		Notes
	Source	Voltage	Source	Voltage	Source	Voltage	
000	Vcc/2	2.5 V 1.65 V	Vcc/2+Vbg	3.8 V 2.95 V	Vcc/2-Vbg	1.2 V 0.35 V	5.0 V System 3.3 V System
001	P2[4]	2.2 V	P2[4]+P2[6]	3.2 V	P2[4]-P2[6]	1.2 V	User Adjustable Example: P2[4]=2.2V and P2[6]=1.0V
010	Vcc/2	2.5 V 1.65 V	Vcc	5.0 V 3.3 V	Vss	0.0 V 0.0 V	5.0 V System 3.3 V System
011	2*Vbg	2.6 V	3*Vbg	3.9 V	1*Vbg	1.3 V	Not for 3.3 V Systems
100	2*Vbg	2.6 V	2*Vbg+P2[6]	3.6 V	2*Vbg-P2[6]	1.6 V	P26 <= Vcc - 2.6 Example: P2[6]=1.0V
101	P2[4]	2.2 V	P2[4]+Vbg	3.5 V	P2[4]-Vbg	0.9 V	User Adjustable. Example: P2[4]=2.2V 1.3 <= P2[4] <= Vcc -1.3
110	Vbg	1.30 V	2*Vbg	2.6 V	Vss	0	5.0 V System 3.3 V System
111	1.6*Vbg	2.08 V	3.2*Vbg	4.16 V	Vss	0	Not for 3.3 V Systems

Registros del Switched Capacitor Análogo

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
Analog Switch Cap Type C PSoC Block Control Registers										
x,xxh	ASCxxCR0	FCap	ClockPhase	ASign	ACap[4:0]					RW : 00
x,xxh	ASCxxCR1	ACMux[2:0]			BCap[4:0]					RW : 00
x,xxh	ASCxxCR2	AnalogBus	CompBus	AutoZero	CCap[4:0]					RW : 00
x,xxh	ASCxxCR3	ARefMux[1:0]		FSW1	FSW0	BMuxSC[1:0]		PWR[1:0]		RW : 00
Analog Switch Cap Type D PSoC Block Control Registers										
x,xxh	ASDxxCR0	FCap	ClockPhase	ASign	ACap[4:0]					RW : 00
x,xxh	ASDxxCR1	AMux[2:0]			BCap[4:0]					RW : 00
x,xxh	ASDxxCR2	AnalogBus	CompBus	AutoZero	CCap[4:0]					RW : 00
x,xxh	ASDxxCR3	ARefMux[1:0]		FSW1	FSW0	BSW	BMuxSD	PWR[1:0]		RW : 00

LEGEND
x: An "x" before the comma in the address field indicates that the register exists in both register banks.
xx: An "x" after the comma in the address field indicates that there are multiple instances of the register.

Registros del Continuous Time Análogo

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Access
x,7xh	ACBxxCR3					LPCMPEN	CMOUT	INSAMP	EXGAIN	RW : 00
x,7xh	ACBxxCR0	RTapMux[3:0]				Gain	RTapMux	RBotMux[1:0]		RW : 00
x,7xh	ACBxxCR1	AnalogBus	CompBus	NMux[2:0]			PMux[2:0]			RW : 00
x,7xh	ACBxxCR2	CPhase	CLatch	CompCap	TMUXEN	TestMux[1:0]		PWR[1:0]		RW : 00

LEGEND
x: An "x" before the comma in the address field indicates that the register exists in both register banks.
x: An "x" after the comma in the address field indicates that there are multiple instances of the register. Fi

ANEXO B

DIRECTORES DEL ASSEMBLER.

SET DE INSTRUCCIONES.

OPERADORES DEL COMPILADOR.

DIRECTORES PREPROCESADORES DEL COMPILADOR.

DIRECTORES PRAGMA DEL COMPILADOR.

FUNCIONES STRING DEL COMPILADOR

FUNCIONES MATEMÁTICAS DEL COMPILADOR

FUNCIONES API DEL COMPILADOR

Resumen del Set de Instrucciones

Opcode Hex	Cycles	Bytes	Instruction Format	Flags	Opcode Hex	Cycles	Bytes	Instruction Format	Flags	Opcode Hex	Cycles	Bytes	Instruction Format	Flags
00	15	1	SSC		2D	8	2	OR [X+expr], A	Z	5A	5	2	MOV [expr], X	
01	4	2	ADD A, expr	C, Z	2E	9	3	OR [expr], expr	Z	5B	4	1	MOV A, X	Z
02	6	2	ADD A, [expr]	C, Z	2F	10	3	OR [X+expr], expr	Z	5C	4	1	MOV X, A	
03	7	2	ADD A, [X+expr]	C, Z	30	9	1	HALT		5D	6	2	MOV A, reg[expr]	Z
04	7	2	ADD [expr], A	C, Z	31	4	2	XOR A, expr	Z	5E	7	2	MOV A, reg[X+expr]	Z
05	8	2	ADD [X+expr], A	C, Z	32	6	2	XOR A, [expr]	Z	5F	10	3	MOV [expr], [expr]	
06	9	3	ADD [expr], expr	C, Z	33	7	2	XOR A, [X+expr]	Z	60	5	2	MOV reg[expr], A	
07	10	3	ADD [X+expr], expr	C, Z	34	7	2	XOR [expr], A	Z	61	6	2	MOV reg[X+expr], A	
08	4	1	PUSH A		35	8	2	XOR [X+expr], A	Z	62	8	3	MOV reg[expr], expr	
09	4	2	ADC A, expr	C, Z	36	9	3	XOR [expr], expr	Z	63	9	3	MOV reg[X+expr], expr	
0A	6	2	ADC A, [expr]	C, Z	37	10	3	XOR [X+expr], expr	Z	64	4	1	ASL A	C, Z
0B	7	2	ADC A, [X+expr]	C, Z	38	5	2	ADD SP, expr		65	7	2	ASL [expr]	C, Z
0C	7	2	ADC [expr], A	C, Z	39	5	2	CMP A, expr	if (A=B) Z=1	66	8	2	ASL [X+expr]	C, Z
0D	8	2	ADC [X+expr], A	C, Z	3A	7	2	CMP A, [expr]	if (A<B) C=1	67	4	1	ASR A	C, Z
0E	9	3	ADC [expr], expr	C, Z	3B	8	2	CMP A, [X+expr]		68	7	2	ASR [expr]	C, Z
0F	10	3	ADC [X+expr], expr	C, Z	3C	8	3	CMP [expr], expr		69	8	2	ASR [X+expr]	C, Z
10	4	1	PUSH X		3D	9	3	CMP [X+expr], expr		6A	4	1	RLC A	C, Z
11	4	2	SUB A, expr	C, Z	3E	10	2	MVI A, [[expr]++]	Z	6B	7	2	RLC [expr]	C, Z
12	6	2	SUB A, [expr]	C, Z	3F	10	2	MVI [[expr]++], A		6C	8	2	RLC [X+expr]	C, Z
13	7	2	SUB A, [X+expr]	C, Z	40	4	1	NOP		6D	4	1	RRC A	C, Z
14	7	2	SUB [expr], A	C, Z	41	9	3	AND reg[expr], expr	Z	6E	7	2	RRC [expr]	C, Z
15	8	2	SUB [X+expr], A	C, Z	42	10	3	AND reg[X+expr], expr	Z	6F	8	2	RRC [X+expr]	C, Z
16	9	3	SUB [expr], expr	C, Z	43	9	3	OR reg[expr], expr	Z	70	4	2	AND F, expr	C, Z
17	10	3	SUB [X+expr], expr	C, Z	44	10	3	OR reg[X+expr], expr	Z	71	4	2	OR F, expr	
18	5	1	POP A	Z	45	9	3	XOR reg[expr], expr	Z	72	4	2	XOR F, expr	
19	4	2	SBB A, expr	C, Z	46	10	3	XOR reg[X+expr], expr	Z	73	4	1	CPL A	Z
1A	6	2	SBB A, [expr]	C, Z	47	8	3	TST [expr], expr	Z	74	4	1	INC A	C, Z
1B	7	2	SBB A, [X+expr]	C, Z	48	9	3	TST [X+expr], expr	Z	75	4	1	INC X	C, Z
1C	7	2	SBB [expr], A	C, Z	49	9	3	TST reg[expr], expr	Z	76	7	2	INC [expr]	C, Z
1D	8	2	SBB [X+expr], A	C, Z	4A	10	3	TST reg[X+expr], expr	Z	77	8	2	INC [X+expr]	C, Z
1E	9	3	SBB [expr], expr	C, Z	4B	5	1	SWAP A, X	Z	78	4	1	DEC A	C, Z
1F	10	3	SBB [X+expr], expr	C, Z	4C	7	2	SWAP A, [expr]	Z	79	4	1	DEC X	C, Z
20	5	1	POP X		4D	7	2	SWAP X, [expr]		7A	7	2	DEC [expr]	C, Z
21	4	2	AND A, expr	Z	4E	5	1	SWAP A, SP	Z	7B	8	2	DEC [X+expr]	C, Z
22	6	2	AND A, [expr]	Z	4F	4	1	MOV X, SP		7C	13	3	LCALL	
23	7	2	AND A, [X+expr]	Z	50	4	2	MOV A, expr	Z	7D	7	3	LJMP	
24	7	2	AND [expr], A	Z	51	5	2	MOV A, [expr]	Z	7E	10	1	RETI	C, Z
25	8	2	AND [X+expr], A	Z	52	6	2	MOV A, [X+expr]	Z	7F	8	1	RET	
26	9	3	AND [expr], expr	Z	53	5	2	MOV [expr], A		8x	5	2	JMP	
27	10	3	AND [X+expr], expr	Z	54	6	2	MOV [X+expr], A		9x	11	2	CALL	
28	11	1	ROMX	Z	55	8	3	MOV [expr], expr		Ax	5	2	JZ	
29	4	2	OR A, expr	Z	56	9	3	MOV [X+expr], expr		Bx	5	2	JNZ	
2A	6	2	OR A, [expr]	Z	57	4	2	MOV X, expr		Cx	5	2	JC	
2B	7	2	OR A, [X+expr]	Z	58	6	2	MOV X, [expr]		Dx	5	2	JNC	
2C	7	2	OR [expr], A	Z	59	7	2	MOV X, [X+expr]		Ex	7	2	JACC	
										Fx	13	2	INDEX	Z

Note: Interrupt routines take 13 cycles before execution resumes at Interrupt Vector table.

Operadores

Pre.	Op.	Function	Group	Form	Description
1	++	Postincrement		a ++	
1	--	Postdecrement		a --	
1	[ ]	Subscript		a[b]	
1	( )	Function Call		a(b)	
1	.	Select Member		a.b	
1	->	Point at Member		a->b	
2	sizeof	Sizeof		sizeof a	
2	++	Preincrement		++ a	
2	--	Predecrement		-- a	
2	&	Address of		&a	
2	*	Indirection		*a	
2	+	Plus		+a	
2	-	Minus		-a	
2	~	Bitwise NOT	Unary	~ a	1's complement of a
2	!	Logical NOT		!a	
2	(decla-ration)	Type Cast			(declaration)a
3	*	Multiplication	Binary	a * b	a multiplied by b
3	/	Division	Binary	a / b	a divided by b
3	%	Modulus	Binary	a % b	Remainder of a divided by b
4	+	Addition	Binary	a + b	a plus b
4	-	Subtraction	Binary	a - b	a minus b
5	<<	Left Shift	Binary	a << b	Value of a shifted b bits left
5	>>	Right Shift	Binary	a >> b	Value of a shifted b bits right
6	<	Less		a < b	a less than b
6	<=	Less or Equal		a <= b	a less than or equal to b
6	>	Greater		a > b	a greater than b
6	>=	Greater or Equal		a >= b	a greater than or equal to b
7	==	Equals		a == b	

Operadores (continuación)

Pre.	Op.	Function	Group	Form	Description
7	!=	Not Equals		a != b	
8	&	Bitwise AND	Bitwise	a & b	Bitwise AND of a and b
9	^	Bitwise Exclusive OR	Bitwise	a ^ b	Bitwise Exclusive OR of a and b
10	|	Bitwise Inclusive OR	Bitwise	a | b	Bitwise OR of a and b
11	&&	Logical AND		a && b	
12	||	Logical OR		a || b	
13	? :	Conditional		c?a:b	
14	=	Assignment		a = b	
14	*=	Multiply Assign		a *= b	
14	/=	Divide Assign		a /= b	
14	%=	Remainder Assign		a %= b	
14	+=	Add Assign		a += b	
14	-=	Subtract Assign		a -= b	
14	<<=	Left Shift Assign		a <<= b	
14	>>=	Right Shift Assign		a >>= b	
14	&=	Bitwise AND Assign		a &= b	
14	^=	Bitwise Exclusive OR Assign		a ^= b	
14	|=	Bitwise Inclusive OR Assign		a |= b	
15	,	Comma		a , b	

Directores preprocesadores

Preprocessor	Description
#define	Define a preprocessor constant or macro
#else	Executed if #if, #ifdef, or #ifndef fails
#endif	Close #if, #ifdef, or #ifndef
#if	Branch based on an expression
#ifdef	Branch if preprocessor constant has been defined
#ifndef	Branch if a preprocessor constant has *not* been defined
#include	Insert a source file
#line	Specify the number of the next source line
#undef	Remove a preprocessor constant

Directores pragma

#pragma	Description
#pragma ioport LED:0x04; char LED;	Defines a variable that occupies a region in I/O space. This variable can then be used in I/O reads and writes. The #pragma ioport must precede a variable declaration defining the variable type used in the pragma.
#pragma fastcall GetChar	Provides an optimized mechanism for argument passing. This #pragma is used only for assembly functions called from "C."
#pragma abs_address:<address>	Allows you to locate 'C' code/Flash data at a specific address such as #pragma abs_address:0x500. The #pragma end_abs_address (described below) should be used to terminate the block of code/Flash data. Note that "data" includes both ROM and RAM.

Directores pragma (continuación)

#pragma	Description
#pragma end_abs_address	Terminates a block of code/Flash data that was located with the abs_address pragma. This allows the code that follows the end_abs_address pragma to be located from the last relocatable point. Note that "data" includes both ROM and RAM.
#pragma text:<name>	Change the name of the "text" area. Make modifications to "Custom.LKP" in the project directory to place the new area in the code space.
#pragma interrupt_handler <func1> [,<func2>]*	For interrupt handlers. Virtual registers are saved only if they are used, unless the handler calls another function. In that case, all Virtual registers are saved.
#pragma nomac #pragma usemac	These two pragmas override the command line -nomac argument, or Project >> Settings, Compiler tab, Enable MAC option. See Section 3, Compiler Project Settings in the *PSoC Designer: Integrated Development Environment User Guide*. The pragmas should be specified outside of a function definition.

Funciones string

Function	Prototype	Description	Header
abs	int abs(int);		stdlib.h
atof	double atof(CONST char *);		stdlib.h
atoi	int atoi(CONST char *);		stdlib.h
atol	long atol(CONST char *);		stdlib.h
itoa	void itoa(char *string, unsigned int value, int base);		stdlib.h
ltoa	void ltoa(char *string, unsigned long value, int base);		stdlib.h
ftoa	char *ftoa(float f, int *status); /* ftoa function */ #define _FTOA_TOO_LARGE -2 /* |input| > 2147483520 */ #define _FTOA_TOO_SMALL -1 /* |input| < 0.0000001 */ /* ftoa returns static buffer of ~15 chars. If the input is out of * range, *status is set to either of the above #define, and 0 is * returned. Otherwise, *status is set to 0 and the char buffer is * returned.	* This version of the ftoa is fast but cannot handle values outside * of the range listed. Please contact us if you need a (much) larger * version that handles greater ranges. * Note that the prototype differs from the earlier version of this * function. */	stdlib.h
rand	int rand(void);		stdlib.h
srand	void srand(unsigned);		stdlib.h
strtol	long strtol(CONST char *, char **, int);		stdlib.h
strtoul	unsigned long strtoul(CONST char *, char **, int);		stdlib.h
	char *cstrncpy(char *, const char *cs, size_t);		string.h
cstrcat	char *cstrcat(char *, const char *);		string.h
cstrcmp	int cstrcmp(const char *cs, char *);		string.h
cstrcpy	char *cstrcpy(char *, const char *cs);		string.h
cstrlen	size_t cstrlen(const char *cs);		string.h
memchr	void *memchr(void *, int, size_t);		string.h
memcmp	int memcmp(void *, void *, size_t);		string.h

Funciones string (continuación)

Function	Prototype	Description	Header
memcpy	void *memcpy(void *, void *, size_t);		string.h
memmove	void *memmove(void *, void *, size_t);		string.h
memset	void *memset(void *, int, size_t);		string.h
strcat	char *strcat(char *, CONST char *);		string.h
strcmp	int strcmp(CONST char *, CONST char *);		string.h
strcoll	int strcoll(CONST char *, CONST char *);		string.h
strcpy	char *strcpy(char *, CONST char *);		string.h
strcspn	size_t strcspn(CONST char *, CONST char *);		string.h
strlen	size_t strlen(CONST char *);		string.h
strncat	char *strncat(char *, CONST char *, size_t);		string.h
strncmp	int strncmp(CONST char *, CONST char *, size_t);		string.h
strncpy	char *strncpy(char *, CONST char *, size_t);		string.h
strpbrk	char *strpbrk(CONST char *, CONST char *);		string.h
strrchr	char *strrchr(CONST char *, int);		string.h
strspn	size_t strspn(CONST char *, CONST char *);		string.h
strstr	char *strstr(CONST char *, CONST char *);		string.h

Funciones matemáticas

Function	Description
float fabs(float x);	fabs calculates the absolute value (magnitude) of the argument x, by direct manipulation of the bit representation of x. Return the absolute value of the floating point number x.
float frexp(float x, int *eptr);	All non zero, normal numbers can be described as m * 2**p. frexp represents the double val as a mantissa m and a power of two p. The resulting mantissa will always be greater than or equal to 0.5, and less than 1.0 (as long as val is nonzero). The power of two will be stored in *exp. Return the mantissa and exponent of x as the pair (m, e). m is a float and e is an integer such that x == m * 2**e. If x is zero, returns (0.0, 0), otherwise 0.5 <= abs(m) < 1.
float tanh(float x);	The function returns the hyperbolic tangent of x.
float sin(float x);	Return the sine of x.
float atan(float x);	The function returns the angle whose tangent is x, in the range [-pi/2, +pi/2] radians.
float atan2(float y, float x);	The function returns the angle whose tangent is y/x, in the full angular range [-pi, +pi] radians.
float asin(float x);	The function returns the angle whose sine is x, in the range [-pi/2, +pi/2] radians.
float exp10(float x);	Returns 10 raised to the specified real number.
float log10(float x);	log10 returns the base 10 logarithm of x. It is implemented as log(x) / log(10).
float fmod(float y, float z);	The fmod function computes the floating-point remainder of x/y (x modulo y). The fmod function returns the value for the largest integer i such that, if y is nonzero, the result has the same sign as x and magnitude less than the magnitude of y.
float sqrt(float x);	The function returns the square root of x, x^(1/2).
float cos(float x);	The function returns the cosine of x for x in radians. If x is large the value returned might not be meaningful, but the function reports no error.
float ldexp(float d, int n);	ldexp calculates the value that it takes and returns float rather than double values. ldexp returns the calculated value.

Funciones matemáticas (continuación)

Function	Description
float modf(float y, float *i);	modf splits the double val apart into an integer part and a fractional part, returning the fractional part and storing the integer. The fractional part is returned. Each result has the same sign as the supplied argument val.
float floor(float y);	floor finds the nearest integer less than or equal to x. floor returns the integer result as a double.
float ceil(float y);	ceil finds the nearest integer greater than or equal to x. ceil returns the integer result as a double.
float fround(float d);	Produce a quotient that has been rounded to the nearest mathematical integer; if the mathematical quotient is exactly halfway between two integers, (that is, it has the form integer+1/2) then the quotient has been rounded to the even (divisible by two) integer.
float tan(float x);	The function returns the tangent of x for x in radians. If x is large the value returned might not be meaningful, but the function reports no error.
float acos(float x);	acos computes the inverse cosine (arc cosine) of the input value. Arguments to acos must be in the range -1 to 1. The function returns the angle whose cosine is x, in the range [0, pi] radians.
float exp(float x);	exp calculates the exponential of x, that is, the base of the natural system of logarithms, approximately 2.71828). The function returns the exponential of x, e^x.
float log(float x);	Return the natural logarithm of x, that is, its logarithm base e (where e is the base of the natural system of logarithms, 2.71828...). The function returns the natural logarithm of x.
float pow(float x,float y);	pow calculates x raised to the exp1.0nt y. On success, pow returns the value calculated.
float sinh(float x);	sinh computes the hyperbolic sine of the argument x. The function returns the hyperbolic sine of x.
float cosh(float x);	cosh computes the hyperbolic cosine of the argument x. The function returns the hyperbolic cosine of x.

Funciones API

Function	Prototype	Description	Header
bFlashWriteBlock	BYTE bFlashWriteBlock (FLASH_WRITE_STRUCT *) See *flashblock* header file for definition of structure.	Writes data to the Flash Program Memory.	*flashblock.h*, *flashblock.inc* (for assembly language)
FlashReadBlock	void FlashReadBlock (FLASH_READ_STRUCT *) See *flashblock* header file for definition of structure.	Reads data from the Flash Program Memory into RAM.	*flashblock.h*, *flashblock.inc* (for assembly language)

5.1 Area AREA

Description: Defines where code or data is located in Flash by the Linker. The Linker gathers all areas with the same name together from the source files, and either concatenates or overlays them, depending on the attributes specified. All areas with the same name must have the same attributes, even if they are used in different modules.

The following is a complete list of valid key words that can be used with the AREA directive:

- **RAM:** Specifies that data is stored in RAM. Only used for variable storage. Commonly used with BLK directive.
- **ROM:** Specifies that code or data is stored in Flash.
- **ABS:** Absolute, i.e., non-relocatable, location for code or data specified by the ORG directive. Default value if ABS or REL is not specified.
- **REL:** Allows the linker to relocate the code or data.
- **CON:** Specifies that sequential AREAs follow each other in memory. Each AREA is allocated its own memory. The total size of the AREA is the sum of all AREA sizes. Default value if CON or OVR is not specified.
- **OVR:** Specifies that sequential AREAs start at the same address. This is a union of the AREAs. The total size of the AREA is the size of the largest area.

PSoC Designer requires that the bss area be used for RAM variables.

Directive	Arguments
AREA	<name> (< RAM | ROM >, [ABS | REL], [CON | OVR])

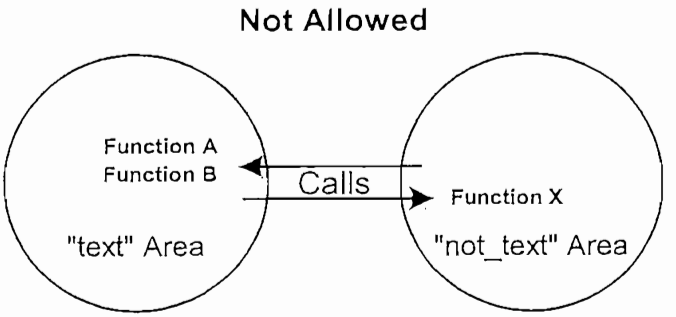
5.1.1 Example

A code area is defined at address 2000.

```
AREA MyArea(ROM,ABS,CON)
_MyArea_start:
        ORG 2000h
```

5.1.2 Code Compressor and the AREA Directive

The Code Compressor "looks" for duplicate code within the 'text" Area. The "text" Area is the default area in which all 'C' code is placed.



The above diagram shows a scenario that is not allowed or potentially problematic. Code areas created with the AREA directive, using a name other than "text," are not compressed or "fixed up" (following compression). Therefore, if Function A in the "text" Area calls Function X in the "non_text" Area, then Function X calls Function B where there would be "thepotential" that the location of Function B changed. The call or jump generated in the code for Function X would go to the wrong location.

It is allowable for Function A to call a function in a "non_text" Area and simply return.

For example, if Function A in the "text" Area calls Function X in the "non_text" Area, then Function X calls to Function B could be invalid. The location for Function B can change because it is in the "text" Area. Calls and jumps are fixed up in the "text" Area only. Following code compression, the call location to Function B from Function X in the "non_text" Area will not be fixed up.

All normal user code that is to be compressed must be in the default "text" Area. If you create code in other area, for example, in a bootloader, then it must not call any functions in the "text" Area. However, it is acceptable for a function in the "text" Area to call functions in other areas. The exception is the TOP area where the interrupt vectors and the startup code can call functions in the "text" Area. Addresses within the "text" Area must be not used directly otherwise.

If you reference any text area function by address, then it must be done indirectly. Its address must be put in a word in the area "func_lit." At runtime, you must de-reference the content of this word to get the correct address of the function. Note that if you are using C to call a function indirectly, the compiler will take care of all these details for you. The information is useful if you are writing assembly code.

For further details on enabling and using code compression, see:

- *PSoC Designer: C Language Compiler User Guide* (*Code Compression*)
- PSoC Designer: Integrated Development Environment User Guide (Project Settings)

5.2 NULL Terminated ASCII String — ASCIZ

Description: Stores a string of characters as ASCII values and appends a terminating NULL (00h) character. The string must start and end with quotation marks "".

The string is stored character by character in ASCII hex format. The back-slash character \ is used in the string as an escape character. Non-printing characters, such as \n and \r, can be used. A quotation mark (") can be entered into a string using the backslash (\"), a single quote (') as (\'), and a backslash (\) as (\\).

Directive	Arguments
ASCIZ	< "character string" >

5.2.1 Example

My"String\ is defined with a terminating NULL character.

```
MyString:
        ASCIZ "My\"String\\"
```

5.3 RAM Block in Bytes — BLK

Description: Reserves blocks of RAM in bytes. The argument is an expression, specifying the size of the block, in bytes, to reserve. The AREA directive must be used to ensure the block of bytes will reside in the correct memory location.

PSoC Designer requires that the bss area be used for RAM variables.

Directive	Arguments
BLK	< size >

5.3.1 Example

A 4-byte variable called MyVariable is allocated.

```
        AREA bss
```

```
MyVariable:
        BLK 4
```

5.4 RAM Block in Words — BLKW

Description: Reserves a block of RAM. The amount of RAM reserved is determined by the size argument to the directive. The units for the size argument is words (16 bits).

PSoC Designer requires that the `AREA bss` be used for RAM variables.

Directive	Arguments
BLKW	< size >

5.4.1 Example

A 4-byte variable called MyVariable is allocated.

```
        AREA bss
MyVariable:
        BLKW 2
```

5.5 Define Byte — DB

Description: Reserves bytes of ROM and assigns the specified values to the reserved bytes. This directive is useful for creating data tables in ROM.

Arguments may be constants or labels. The length of the source line limits the number of arguments in a `DB` statement.

Directive	Arguments
DB	< value1 > [, value2, ..., valuen]

5.5.1 Example

3 bytes are defined starting at address 3000.

```
MyNum:  EQU 77h
        ORG 3000h
MyTable:
        DB 55h, 66h, MyNum
```

5.6 Define ASCII String DS

Description: Stores a string of characters as ASCII values. The string must start and end with quotation marks "".

The string is stored character by character in ASCII hex format. The backslash character \ is used in the string as an escape character. Non-printing characters, such as \n and \r, can be used. A quotation mark (") can be entered into a string using the backslash (\"), a single quote (') as (\'), and a backslash (\) as (\\).

The string is not null terminated. To create a null terminated string; follow the DS with a `DB 00h` or use `ASCIZ`.

Directive	Arguments
DS	< "character string" >

5.6.1 Example

My"String\ is defined:

```
MyString:
        DS "My\"String\\"
```

5.7 Define UNICODE String DSU

Description: Stores a string of characters as UNICODE values with little ENDIAN byte order. The string must start and end with quotation marks "".

The string is stored character by character in UNICODE format. Each character in the string is stored with the low byte followed by the high byte.

The backslash character \ is used in the string as an escape character. Non-printing characters, such as \n and \r, can be used. A quotation mark (") can be entered into a string using the backslash (\"), a single quote (') as (\'), and a backslash (\) as (\\).

Directive	Arguments
DSU	< "character string" >

5.7.1 Example

My"String\ is defined with little endian byte order.

```
MyString:
        DSU "My\"String\\"
```

5.8 Define Word DW

Description: Reserves two-byte pairs of ROM and assigns the specified words to each reserved byte. This directive is useful for creating tables in ROM.

The arguments may be constants or labels. Only the length of the source line limits the number of arguments in a DW statement.

Directive	Arguments
DW	< value1 > [, value2, ..., valuen]

5.8.1 Example

6 bytes are defined starting at address 2000.

```
MyNum:  EQU 3333h
        ORG 2000h
MyTable:
        DW 1111h, 2222h, MyNum
```

5.9 Define Word, Little Endian Ordering DWL

Description: Reserves two-byte pairs of ROM and assigns the specified words to each reserved byte, swapping the order of the upper and lower bytes.

The arguments may be constants or labels. The length of the source line limits the number of arguments in a DWL statement.

Directive	Arguments
DWL	< value1 > [, value2, ..., valuen]

5.9.1 Example

6 bytes are defined starting at address 2000.

```
MyNum:  EQU 6655h
        ORG 2000h
MyTable:
        DWL 2211h, 4433h, MyNum
```

5.10 Equate Label EQU

Description: Assign an integer value to a label. The label and operand are required for an EQU directive. The argument must be a constant or label or "." (the current PC). Each EQU directive may have only one argument and if a label is defined more than once, an assembly error will occur.

To use the same equate in more than one assembly source file, place the equate in an *.inc* file and include *that* file in the referencing source files. Do not export equates from assembly source files, or the PSoC Designer Linker will resolve the directive in unpredictable ways.

Directive	Syntax
EQU	< label> EQU < value | address >

5.10.1 Example

BITMASK is equated to 1Fh.

```
BITMASK: EQU 1Fh
```

5.11 Export EXPORT

Description: Designate that a label is global, and can be referenced in another file. Otherwise, the label is not visible to another file. Another way to export a label is to end the label definition with two colons instead of one.

Directive	Syntax
EXPORT	EXPORT < label >

5.11.1 Example

MyVariable is exported.

```
Export MyVariable
        AREA bss
MyVariable:
        BLK 1
```

5.12 Conditional Source IF, ELSE, ENDIF

Description: All source lines between the `IF` and `ENDIF` (or `IF` and `ELSE`) directives are assembled if the condition is true. These statements can be nested.

`Else` delineates a "not true" action for a previous `IF` directive.

`ENDIF` finishes a section of conditional assembly that began with an `IF` directive.

Directive	Arguments
IF	value
[ELSE]	
ENDIF	

5.12.1 Example

Sections of the source code are conditional.

```
Cond1:  EQU 1
Cond2:  EQU 0
        ORG 1000h
        IF (Cond1)
        ADD A, 33h
        IF (Cond2)
        ADD A, FFh
        ENDIF
        NOP
        ELSE
        MOV A, FFh
        ENDIF
// The example creates the following code
        ADD A, 33h
        NOP
```

5.13 Include Source File INCLUDE

Description: Used to add additional source files to the file being assembled. When an INCLUDE directive is encountered, the assembler reads in the specified source file until either another INCLUDE is encountered or the end of file is reached. If additional INCLUDES are encountered, additional source files are read in. When an end of file is encountered, the assembler resumes reading the previous file.

Specify the full (or relative) path to the file if the source file does not reside in the current directory.

Directive	Arguments
INCLUDE	< file name >

5.13.1 Example

Three files are included into the source code.

```
INCLUDE "MyIncludel.inc"
INCLUDE "MyIncludeFiles\MyInclude2.inc"
INCLUDE "C:\MyGlobalIncludeFiles\MyInclude3.inc"
```

5.14 Prevent Code Compression of Data .LITERAL, .ENDLITERAL

Description: Used to avoid code compression of the data defined between the .LITERAL and .ENDLITERAL directives. For the code compressor to function, all data defined in ROM with ASCIZ, DB, DS, DSU, DW, or DWL must use this directive. The .LITERAL directive must be followed by an exported global label. The .ENDLITERAL directive resumes code compression.

Directive	Syntax
.LITERAL .ENDLIT- ERAL	< none >

5.14.1 Example

Code compression is suspended for the data table.

```
Export DataTable
.LITERAL
DataTable:
DB 01h, 02h, 03h
.ENDLITERAL
```

5.15 Macro Definition MACRO, ENDM

Description: Used to specify the start and end of a macro definition. The lines of code defined between a MACRO statement and an ENDM statement are not directly assembled into the program. Instead, it forms a macro that can later be substituted into the code by a macro call. Following the MACRO directive is used to call the macro as well as a list of parameters. Each time a parameter is used in the macro body of a macro call, it will be replaced by the corresponding value from the macro call.

Any assembly statement is allowed in a macro body except for another macro statement. Within a macro body, the expression @digit, where digit is between 0 and 9, is replaced by the corresponding macro argument when the macro is invoked. You cannot define a macro name that conflicts with an instruction mnemonic or an assembly directive.

Directive	Arguments
MACRO	< name >< arguments >
ENDM	

5.15.1 Example

A MACRO is defined and used in the source code.

```
        MACRO MyMacro
        ADD A, 42h
        MOV X, 33h
        ENDM
// The Macro instructions are expanded at address 2400
        ORG 2400h
        MyMacro
```

5.16 Area Origin ORG

Description: Allows the programmer to set the value of the Program/Data Counter during assembly. This is most often used to set the start of a table in conjunction with the define directives DB, DS, and DW. The ORG directive can only be used in areas with the ABS mode.

An operand is required for an ORG directive and may be an integer constant, a label, or "." (the current PC). The assembler does not keep track of areas previously defined and will not flag overlapping areas in a single source file.

Directive	Arguments
ORG	< address >

5.16.1 Example

The bytes defined after the ORG statement are at address 1000.

```
ORG 1000h
DB 55h, 66h, 77h
```

5.17 Section for Dead-Code Elimination .SECTION, .ENDSECTION

Description: Allows the removal of code specified between the .SECTION and .ENDSECTION directives. The .SECTION directive must be followed by an exported global label. If there is no call to the global label, the code will be eliminated and call offsets will be adjusted appropriately. The .ENDSECTION directive ends the dead-code section.

Note that use of this directive is not limited to removing dead code. PSoC Designer takes care of dead code if you check the "Enable Elimination of un-used User Modules (area) APIs" field. This feature can be accessed under Project >> Settings, Compiler tab. If you check this field, upon a build the system will go in and remove all "dead code" from the APIs in effort to free up space.

Directive	Arguments
.SECTION	< none >
.ENDSECTION	

5.17.1 Example

The section of code is designated as possible dead code.

```
Export Counter8_1_WriteCompareValue
.SECTION
Counter8_1_WriteCompareValue:
```

```
    MOV   reg[Counter8_1_COMPARE_REG], A
    RET
.ENDSECTION
```

5.18 Suspend and Resume Code Compressor Suspend - OR F,0 Resume - ADD SP,0

Description: Used to prevent code compression of the code between the `OR F,0` and `ADD SP,0` instructions. The code compressor may need to be suspended for timing loops and jump tables. If the `JACC` instruction is used to access fixed offset boundaries in a jump table, any `LJMP` and/or `LCALL` entries in the table may be optimized to relative jumps or calls, changing the proper offset value for the `JACC`. An `RET` or `RETI` instruction will resume code compression if it is encountered before an `ADD SP,0` instruction. These instructions are defined as the macros `Suspend_CodeCompressor` and `Resume_CodeCompressor` in *m8c.inc*.

Directive	Arguments
OR F,0	<none>
ADD SP,0	

5.18.1 Example

Code compression is suspended for the jump table.

```
OR F,0
MOV A, [State]
JACC StateTable
StateTable:
LJMP State1
LJMP State2
LJMP State3
ADD SP,0
```

ANEXO C

DATASHEET DEL DALLAS DS1821

COMUNICACIÓN ONE-WIRE

DALLAS SEMICONDUCTOR

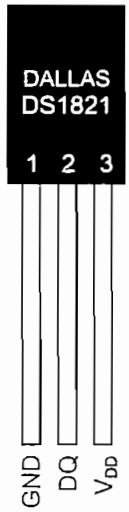
DS1821
Programmable Digital Thermostat and Thermometer

www.maxim-ic.com

FEATURES

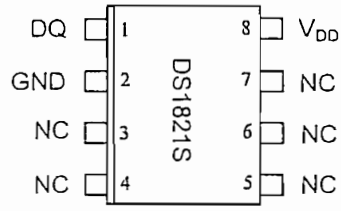
- Requires no external components
- Unique 1-Wire® interface requires only one port pin for communication
- Operates over a -55°C to +125°C (-67°F to +257°F) temperature range
- Functions as a standalone thermostat with user-definable trip-points
- Provides 8-bit (1°C resolution) centigrade temperature measurements
- Accuracy is ±1°C over 0°C to +85°C range
- Converts temperature to a digital word in 1 second (max)
- Available in 3-pin PR35 and 8-pin SOIC packages
- Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system

PIN ASSIGNMENT



(BOTTOM VIEW)

PR35
(DS1821)



8-pin 208-mil SOIC
(DS1821S)

PIN DESCRIPTION

GND	- Ground
DQ	- Data In/Out and Thermostat Output
V_{DD}	- Power Supply Voltage
NC	- No Connect

DESCRIPTION

The DS1821 can function as a standalone thermostat with user-programmable trip-points or as 8-bit temperature sensor with a 1-wire digital interface. The thermostat trip-points are stored in nonvolatile memory, so DS1821 units can be programmed prior to system insertion for true standalone operation. The DS1821 has an operating temperature range of –55°C to +125°C and is accurate to ±1°C over a range of 0°C to +85°C. Communication with the DS1821 is accomplished through the open-drain DQ pin; this pin also serves as the thermostat output.

DETAILED PIN DESCRIPTIONS Table 1

PR35	8-PIN SOIC*	SYMBOL	DESCRIPTION
1	2	GND	**Ground pin.**
2	1	DQ	**Open drain data input/output pin** – 1-wire operation; **Open drain thermostat output pin** –thermostat operation.
3	8	V_{DD}	**Power supply pin.**

*All pins not specified in this table are "No Connect" pins.

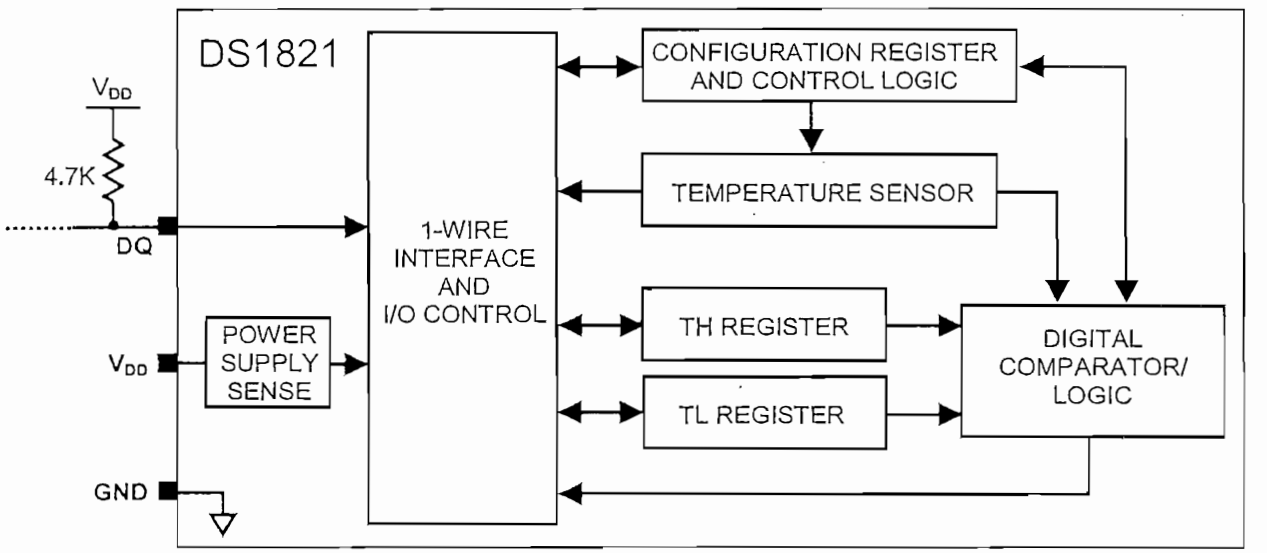
OVERVIEW

Figure 1 shows a block diagram of the DS1821 and pin descriptions are given in Table 1. The DS1821 can operate as a standalone thermostat with user-programmable trip-points or as 8-bit temperature sensor with a 1-wire digital interface. The open-drain DQ pin functions as the thermostat output for thermostat operation and as the data I/O pin for 1-wire communications. The 1-wire interface provides user access to the nonvolatile (EEPROM) thermostat trip-point registers (T_H and T_L), the status/configuration register, and the temperature register.

When configured as standalone thermostat, temperature conversions start immediately at power-up. In this mode, the DQ pin becomes active when the temperature of the DS1821 exceeds the limit programmed into the T_H register, and remains active until the temperature drops below the limit programmed into the T_L register.

The DS1821 uses Dallas' exclusive 1-wire bus protocol that implements bus communication with one control signal. This system is explained in detail in the 1-WIRE BUS SYSTEM section of this datasheet.

DS1821 BLOCK DIAGRAM Figure 1



TEMPERATURE SENSOR FUNCTIONALITY

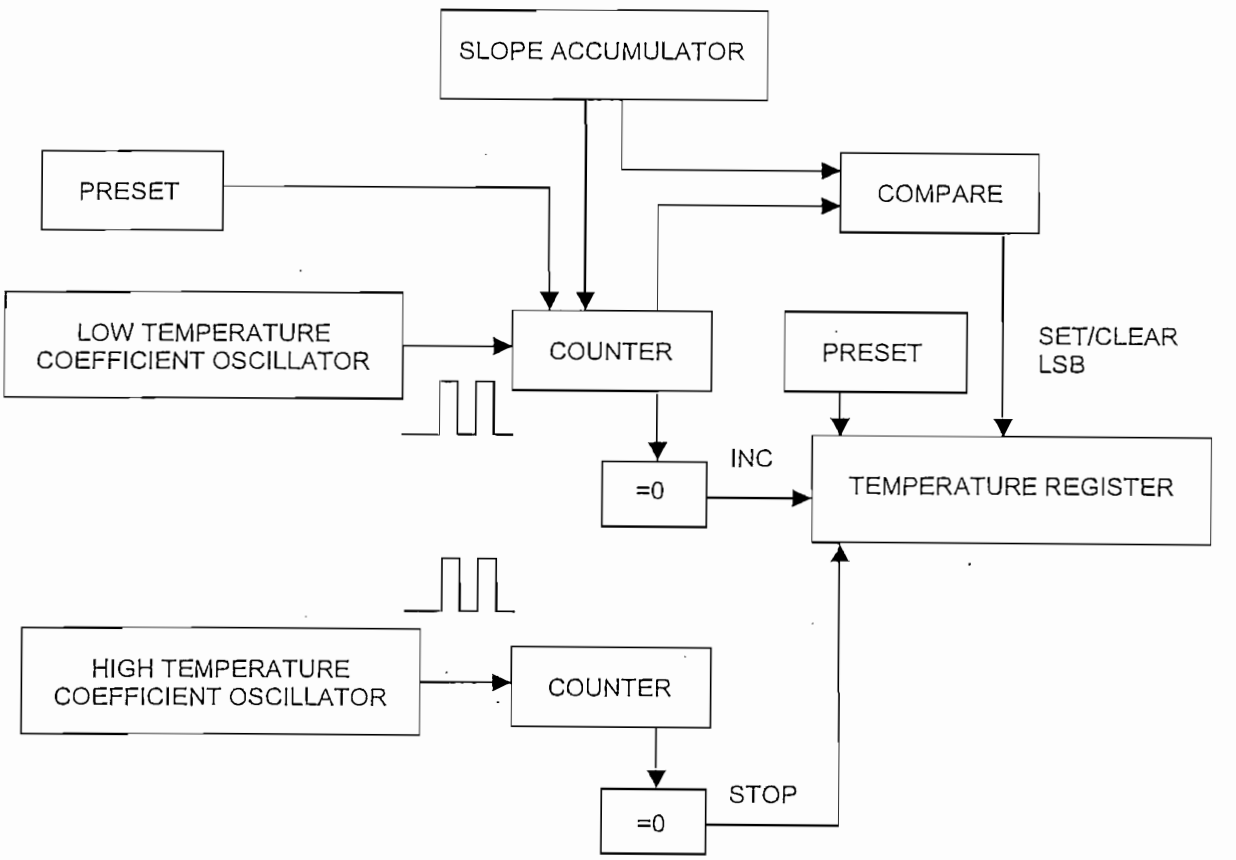
The core functionality of the DS1821 is its proprietary direct-to-digital temperature sensor, which provides 8-bit (1°C increment) centigrade temperature readings over the range of -55°C to +125°C.

A block diagram of the temperature measurement circuitry is shown in Figure 2. This circuit measures the temperature by counting the number of clock cycles generated by an oscillator with a low temperature coefficient (temp-co) during a gate period determined by a high temp-co oscillator. The low temp-co

counter is preset with a base count that corresponds to –55°C. If the counter reaches 0 before the gate period is over, the temperature register, which is preset to –55°C, is incremented by one degree, and the counter is again preset with a starting value determined by the slope accumulator circuitry. The preset counter value is unique for every temperature increment and compensates for the parabolic behavior of the oscillators over temperature.

At this time, the counter is clocked again until it reaches 0. If the gate period is not over when the counter reaches 0, the temperature register is incremented again. This process of presetting the counter, counting down to zero, and incrementing the temperature register is repeated until the counter takes less time to reach zero than the duration of the gate period of the high temp-co oscillator. When this iterative process is complete, the value in the temperature register will indicate the centigrade temperature of the device.

TEMPERATURE MEASURING CIRCUITRY Figure 2



OPERATING MODES

The DS1821 has two operating modes: 1-wire mode and thermostat mode. The power-up operating mode is determined by the user-programmable T/$\overline{R}$ bit in the status/configuration register: if T/$\overline{R}$ = 0 the device powers-up in 1-wire mode, and if T/$\overline{R}$ = 1 the device powers-up in thermostat mode. The T/$\overline{R}$ bit is stored in nonvolatile memory (EEPROM), so it will retain its value when the device is powered down.

1-WIRE MODE

The DS1821 arrives from the factory in 1-wire mode ($T/\overline{R} = 0$). In this mode, the DQ pin of the DS1821 is configured as a 1-wire port for communication with a microprocessor using the protocols described in the 1-WIRE BUS SYSTEM section of this datasheet. These communications can include reading and writing the high and low thermostat trip-point registers (T_H and T_L) and the configuration register, and reading the temperature, counter, and slope accumulator registers. Also in this mode, the microprocessor can initiate and stop temperature measurements as described in the OPERATION – MEASURING TEMPERATURE section of this datasheet.

The T_H and T_L registers and certain bits (THF, TLF, $T/\overline{R}$, POL and 1SHOT) in the status/configuration register are stored in nonvolatile EEPROM memory, so they will retain data when the device is powered down. This allows these registers to be pre-programmed when the DS1821 is to be used as a standalone thermostat. Writes to these nonvolatile registers can take up to 10ms. To avoid data corruption, no writes to nonvolatile memory should be initiated while a write to nonvolatile memory is in progress. Nonvolatile write status can be monitored by reading the NVB bit in the status/configuration register: NVB = 0 – a write to EEPROM memory is in progress, NVB = 0 – nonvolatile memory is idle.

THERMOSTAT MODE

In thermostat mode ($T/\overline{R} = 1$), the DS1821 can operate as a standalone thermostat that triggers according to the T_H and T_L trip-points programmed while the device was in 1-wire mode. In thermostat mode the DS1821 powers-up performing continuous temperature conversions, and the DQ pin acts as the thermostat output. Detailed operation of the thermostat output is provided in the OPERATION – STANDALONE THERMOSTAT section of this datasheet.

Communications can be re-establish with the DS1821 while it is in thermostat mode by pulling V_{DD} to 0V while the DQ line is held high, and then toggling the DQ line low 16 times as shown in Figure 12. This temporarily places the DS1821 in 1-wire mode, allowing microprocessor communication with the DS1821 via the DQ pin. At this time any I/O function can be performed, such as reading/writing the T_H, T_L or configuration registers or reading the temperature register. To return to thermostat mode, the same procedure can be performed (pulling V_{DD} to 0V while the DQ line is held high, and then clocking the DQ line 16 times) or the power can be cycled. Note that temporarily putting the DS1821 into 1-wire mode does not change the power-up mode of the device; this can only be changed by rewriting the $T/\overline{R}$ bit in the status/configuration register. Also note that holding both V_{DD} and DQ low for more than approximately 10 seconds will cause the DS1821 to be powered down.

OPERATION – MEASURING TEMPERATURE

DS1821 output temperature data is calibrated in degrees centigrade and is stored in two's complement format in the 1-byte (8-bit) temperature register (see Figure 3), which the user can access when the DS1821 is in 1-wire mode ($T/\overline{R} = 0$ in the status/configuration register). The sign bit (S) indicates if the temperature is positive or negative; for positive numbers S = 0 and for negative numbers S = 1. Table 2 gives examples of digital output data and the corresponding temperature reading. For Fahrenheit measurements, a lookup table or conversion routine must be used.

The DS1821 can be configured by the user to take continuous temperature measurements (continuous conversion mode) or single measurements (one-shot mode). The desired configuration can be achieved by setting the nonvolatile1SHOT bit in the status/configuration register: 1SHOT = 0 – continuous conversion mode, 1SHOT = 1 – one-shot mode. Note that the 1SHOT setting only controls the operation of the device in 1-wire mode; in thermostat mode, continuous temperature conversions are started automatically at power-up.

In continuous conversion mode, the Start Convert T [EEh] command initiates continuous temperature conversions, which can be stopped using the Stop Convert T [22h] command. In one-shot mode the Start Convert T [EEh] command initiates a single temperature conversion after which the DS1821 returns to a low-power standby state. In this mode, the microprocessor can monitor the DONE bit in the configuration register to determine when the conversion status: DONE = 0 — conversion in progress, DONE = 1 — conversion complete. The DONE bit does not provide conversion status in continuous conversion mode since measurements are constantly in progress (i.e., DONE will always be 0).

TEMPERATURE, T_H and T_L REGISTER FORMAT Figure 3

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

TEMPERATURE/DATA RELATIONSHIP Table 2

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C*	0111 1101	7Dh
+85°C	0101 0101	55h
+25°C	0001 1001	19h
0°C	0000 0000	00h
-1°C	1111 1111	FFh
-25°C	1110 0111	E7h
-55°C	1100 1001	C9h

HIGH-RESOLUTION TEMPERATURE READINGS

The user can calculate temperature values with higher than 8-bit resolution using the data remaining in the counter and slope accumulator when the temperature conversion is complete. To do this the user must first read the temperature from the 8-bit temperature register. This value is called TEMP_READ in the high-resolution equation (see Eq. 1). The 9-bit counter value must then be obtained by issuing the Read Counter [A0h] command. This value is the count remaining in the counter at the end of the gate period and is called COUNT_REMAIN in Eq. 1. Next the Load Counter [41h] command must be issued, which loads the 9-bit slope accumulator value into the counter register. The slope accumulator value (called COUNT_PER_C in Eq. 1) can then be read from the counter by again issuing the Read Counter [A0h] command. The slope accumulator value is called "COUNT_PER_C" because it represents the number of counts needed for an accurate measurement at a given temperature (i.e., the counts per degree C). The high-resolution temperature can then be calculated using Eq. 1:

Eq. 1) $$TEMPERATURE = TEMP_READ - 0.5 + \frac{(COUNT_PER_C - COUNT_REMAIN)}{COUNT_PER_C}$$

Additional information about high-resolution temperature calculations can be found in Application Note 105: "High Resolution Temperature Measurement with Dallas Direct-to-Digital Temperature Sensors".

OPERATION – THERMOSTAT

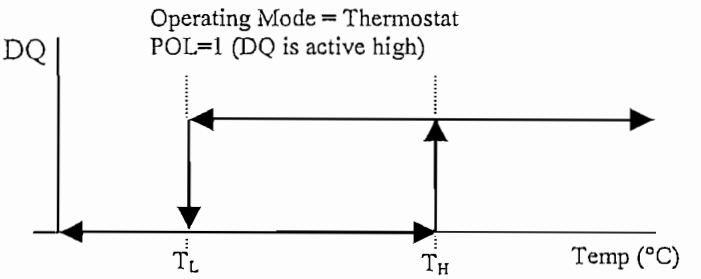
When the DS1821 is in thermostat mode ($T/\overline{R}$ = 1 in the status/configuration register), temperature conversions are performed continuously beginning at power-up (regardless of the value of the 1SHOT bit), and the DQ pin serves as the thermostat output. The DQ output will become active when the temperature of the DS1821 exceeds the user-defined limit in the T_H register, and will remain active until the temperature drops below the user-defined limit in the T_L register as illustrated in Figure 4. Thus, the user can select T_H and T_L to provide the desired amount of thermostat output hysteresis.

The user-defined 8-bit centigrade trip-point values (T_H and T_L) must be stored in two's complement format as shown in Figure 3. The sign bit (S) indicates if the temperature is positive or negative; for positive numbers S = 0 and for negative numbers S = 1. The non-volatile T_H and T_L registers must be programmed when the DS1821 is in 1-wire mode as explained in the OPERATING MODES section of this datasheet. The DS1821 can be temporarily switched from thermostat mode to 1-wire mode to change the T_H and T_L values as also explained in the OPERATING MODES section.

The polarity (i.e., the active state) of the DQ output is user-selectable with the nonvolatile POL bit in the status/configuration register. DQ is active-high when POL = 1, and DQ is active-low when POL = 0.

Two bits in the status/configuration register, THF and TLF, provide additional thermostatic information. The value of these bits is normally 0. The THF (temperature high flag) bit will be set to 1 if the measured temperature is ever greater than the value in the T_H register and will remain a 1 until the user rewrites the bit with a 0. The THL (temperature low flag) bit will be set to 1 if the temperature is ever lower than the value in the T_L register and will remain a 1 until the user rewrites the bit with a 0. These bits provide a record of the device temperature relative to the thermostat trip-points over a period of time. They are stored in nonvolatile memory, so the data stored in THF and TLF can be analyzed after any number of power cycles. The THF and THL bits function in both 1-wire and thermostat mode.

DQ OPERATION IN THERMOSTATE MODE Figure 4



STATUS/CONFIGURATION REGISTER

The status/configuration register provides information to the user about conversion status, EEPROM activity and thermostat activity. It also allows the user to program various DS1821 options such as power-up operating mode, thermostat output polarity and conversion mode. The status/configuration register is arranged as shown in Figure 5 and detailed descriptions of each bit are provided in Table 3. Note that the THF, THL $T/\overline{R}$, POL and 1SHOT bits are stored in nonvolatile memory (EEPROM).

CONFIGURATION REGISTER Figure 5

bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
DONE	1	NVB	THF*	THL*	$T/\overline{R}$*	POL*	1SHOT*

*Stored in EEPROM

CONFIGURATION REGISTER BIT DESCRIPTIONS Table 3

Bit Name (User Access)	Functional Description
DONE — Temperature Conversion Done (Read Only)	DONE = 0 — Temperature conversion is in progress. DONE = 1 — Temperature conversion is complete.
NVB — Non-volatile Memory Busy (Read Only)	NVB = 0 — Nonvolatile memory is not busy. NVB = 1 — A write to EEPROM memory is in progress
THF* — Temperature High Flag (Read/Write)	THF = 0 — The measured temperature has not exceeded the value stored in the T_H register. THF = 1 — At some point in time the measured temperature has been higher than the value stored in the T_H register. THF will remain a 1 until it is over-written with a 0 by the user.
TLF* — Temperature Low Flag (Read/Write)	TLF = 0 — The measured temperature has not been lower than the value stored in the T_L register. TLF = 1 — At some point in time the measured temperature has been lower than the value stored in the T_L register. TLF will remain a 1 until it is over-written with a 0 by the user.
$T/\overline{R}$* — Power-up Operating Mode (Read/Write)	$T/\overline{R}$ = 0 — DS1821 powers up in 1- wire mode. $T/\overline{R}$ = 1 — DS1821 powers up in thermostat mode.
POL* — Thermostat Output (DQ) Polarity (Read/Write)	POL = 0 — Thermostat output (DQ) is active low. POL = 1 — Thermostat output (DQ) is active high.
1SHOT* — Conversion Mode (Read/Write)	1SHOT = 0 — Continuous conversion mode. The Start Convert T [EEh] command initiates continuous temperature conversions, which can be stopped with the Stop Convert T [22h] command. 1SHOT = 1 — One-shot mode. The Start Convert T [EEh] command initiates a single temperature conversion after which the DS1821 returns to a low-power standby state.

*Stored in EEPROM

1-WIRE BUS SYSTEM

The 1-wire bus system uses a single bus master (i.e., a microprocessor) to control slave devices. The DS1821 functions as a slave device when it is used in 1-wire mode; however, since the DS1821 is not addressable or multi-droppable, a single 1-wire-mode DS1821 must be the only slave device on the bus. All data and commands are transmitted least significant bit first over the 1-wire bus.

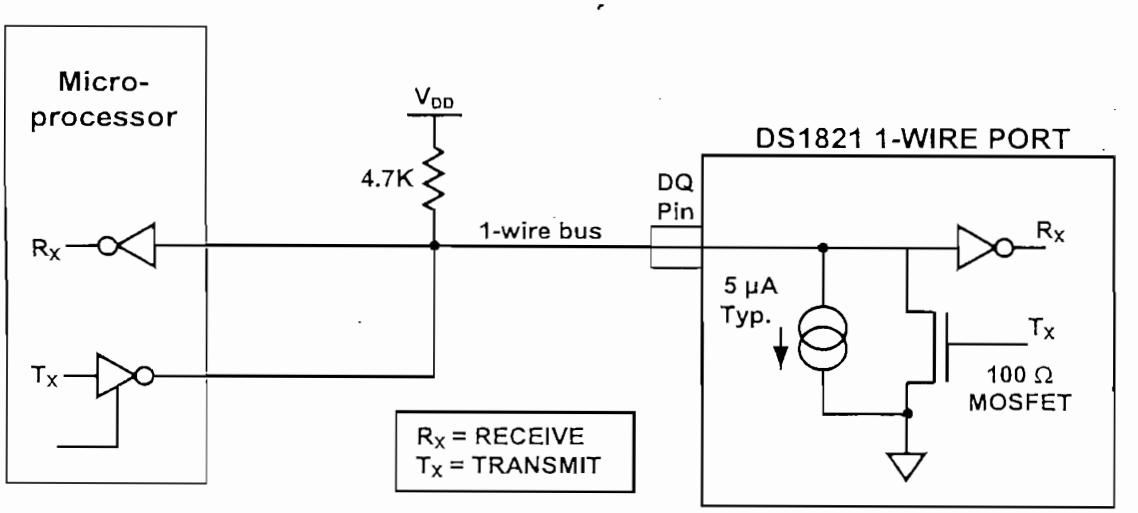
The following discussion of the 1-wire bus system is broken down into three topics: hardware configuration, transaction sequence, and 1-wire signaling (signal types and timing).

HARDWARE CONFIGURATION

The 1-wire bus has by definition only a single data line. Each device (in this case, the master and one DS1821) interfaces to the data line via an open drain or 3–state port. This allows each device to "release" the data line when the device is not transmitting data so that the bus is available for use by the other device. The 1-wire port of the DS1821 (the DQ pin) is open drain with an internal circuit equivalent to that shown in Figure 6.

The 1-wire bus requires an external pullup resistor of approximately 5 kΩ; thus, the idle state for the 1-wire bus is high. If for any reason a transaction needs to be suspended, the bus MUST be left in the idle state if the transaction is to resume. Infinite recovery time can occur between bits so long as the 1-wire bus is in the inactive (high) state during the recovery period. If the bus is held low for more than 480 µs, the DS1821 will be reset.

HARDWARE CONFIGURATION Figure 6



TRANSACTION SEQUENCE

The transaction sequence for accessing the DS1821 via the 1-wire port is as follows:

- Initialization
- DS1821 Function Command
- Data Transmitted/Received

INITIALIZATION

All transactions on the 1-wire bus begin with an initialization sequence. The initialization sequence consists of a reset pulse transmitted by the bus master followed by a presence pulse transmitted by the DS1821. The presence pulse lets the bus master know that the DS1821 is on the bus and ready to operate. Timing for the reset and presence pulses is detailed in the 1-WIRE SIGNALING section.

DS1821 FUNCTION COMMANDS

The DS1821 function commands in this section allow the master to communicate with and configure the DS1821. The DS1821 function commands are summarized in Table 4.

READ TEMPERATURE [AAh]

Provides read access to the 1-byte temperature register.

START CONVERT T [EEh]

Initiates temperature conversions. If the part is in one-shot mode (1SHOT = 1), only one conversion will be performed. If it is in continuous mode (1SHOT = 0), continuous conversions will be performed until a Stop Convert T command is received.

STOP CONVERT T [22h]

Stops temperature conversions when the device is in continuous conversion mode (1SHOT = 0). This opcode has no function if the device is in one-shot mode (1SHOT = 1).

WRITE TH [01h]
WRITE TL [02h]

Provides write access to the 8-bit T_H and T_L registers, respectively.

READ TH [A1h]
READ TL [A2h]

Provides read access to the 8-bit T_H and T_L registers, respectively.

WRITE STATUS [0Ch]

Provides write access to the 8-bit status/configuration register.

READ STATUS [ACh]

Provides read access to the 8-bit status/configuration register.

READ COUNTER [A0h]

Provides read access to data in the 9-bit counter register for use in high-resolution temperature calculations. This is explained in detail in the HIGH-RESOLUTION TEMPERATURE READINGS section.

LOAD COUNTER [41h]

Loads the 9-bit data from the slope accumulator register into the counter register so that it can be accessed using the Read Counter [A0h] command. Use of the Load Counter command is explained in detail in the HIGH-RESOLUTION TEMPERATURE READINGS section.

DS1821 FUNCTION COMMAND SET Table 4

Command	Description	Protocol	1-Wire Bus Activity After Command is Issued
TEMPERATURE CONVERSION COMMANDS			
Read Temperature	Reads last converted temperature value from temperature register.	AAh	Master receives 8-bit temperature value from DS1821.
Start Convert T	Initiates temperature conversions.	EEh	None
Stop Convert T	Halts temperature conversions.	22h	None
THERMOSTAT and STATUS/CONFIGURATION COMMANDS			
Write TH	Writes data to the T_H register.	01h	Master transmits 8-bit T_H value to DS1821.
Write TL	Writes data to the T_L register.	02h	Master transmits 8-bit T_L value to DS1821.
Read TH	Reads data from the T_H register.	A1h	Master receives 8-bit T_H value from DS1821.
Read TL	Reads data from the T_L register.	A2h	Master receives 8-bit T_L value from DS1821.
Write Status	Writes data to the status/configuration register.	0Ch	Master transmits 8-bit status/configuration value to DS1821.
Read Status	Reads data from the status/configuration register.	ACh	Master receives 8-bit status/configuration value from DS1821.
HIGH-RESOLUTION COMMANDS			
Read Counter	Reads data from the counter register	A0h	Master receives 9-bit counter value from DS1821.
Load Counter	Loads slope accumulator data into the counter register	41h	None

1-WIRE SIGNALING

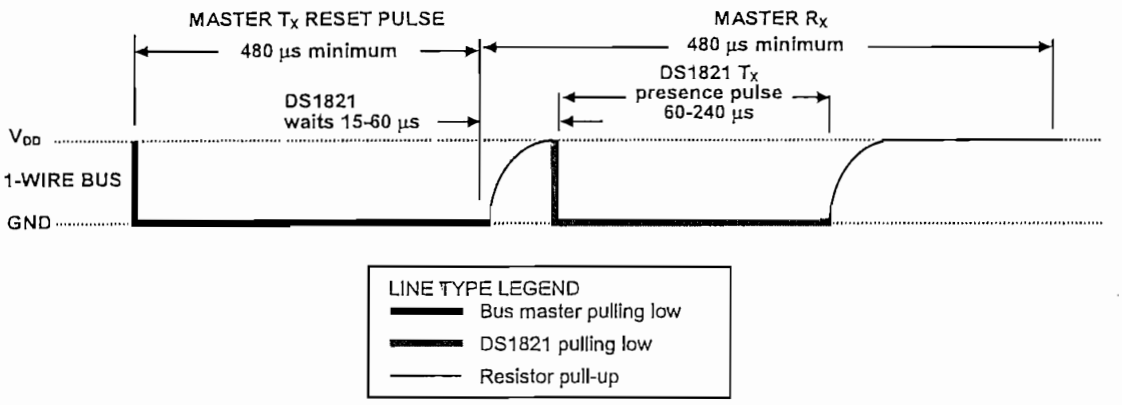
The DS1821 uses a strict 1-wire communication protocol to insure data integrity. Several signal types are defined by this protocol: reset pulse, presence pulse, write 0, write 1, read 0, and read 1. All of these signals, with the exception of the presence pulse, are initiated by the bus master.

INITIALIZATION PROCEDURE: RESET AND PRESENCE PULSES

All communication with the DS1821 begins with an initialization sequence that consists of a reset pulse from the master followed by a presence pulse from the DS1821. This is illustrated in Figure 7. When the DS1821 sends the presence pulse in response to the reset, it is indicating to the master that it is on the bus and ready to operate given an appropriate function command.

During the initialization sequence the bus master transmits (T_X) the reset pulse by pulling the 1-wire bus low for a minimum of 480 µs. The bus master then releases the bus and goes into receive mode (R_X). When the bus is released, the 5k pullup resistor pulls the 1-wire bus high. When the DS1821 detects this rising edge, it waits 15–60 µs and then transmits a presence pulse by pulling the 1-wire bus low for 60–240 µs.

NITIALIZATION TIMING Figure 7



READ/WRITE TIME SLOTS

The bus master writes data to the DS1821 during write time slots and reads data from the DS1821 during read time slots. One bit is data is transmitted over the 1-wire bus per time slot.

WRITE TIME SLOTS

There are two types of write time slots: "Write 1" time slots and "Write 0" time slots. The bus master uses a Write 1 time slot to write a logic 1 to the DS1821 and a Write 0 time slot to write a logic 0 to the DS1821. All write time slots must be a minimum of 60 µs in duration with a minimum of a 1 µs recovery time between individual write slots. Both types of write time slots are initiated by the master pulling the 1-wire bus low (see Figure 8).

To generate a Write 1 time slot, after pulling the 1-wire bus low, the bus master must release the 1-wire bus within 15 µs. When the bus is released, the 5k pullup resistor will pull the bus high. To generate a Write 0 time slot, after pulling the 1-wire bus low, the bus master must continue to hold the bus low for the duration of the time slot (at least 60 µs).

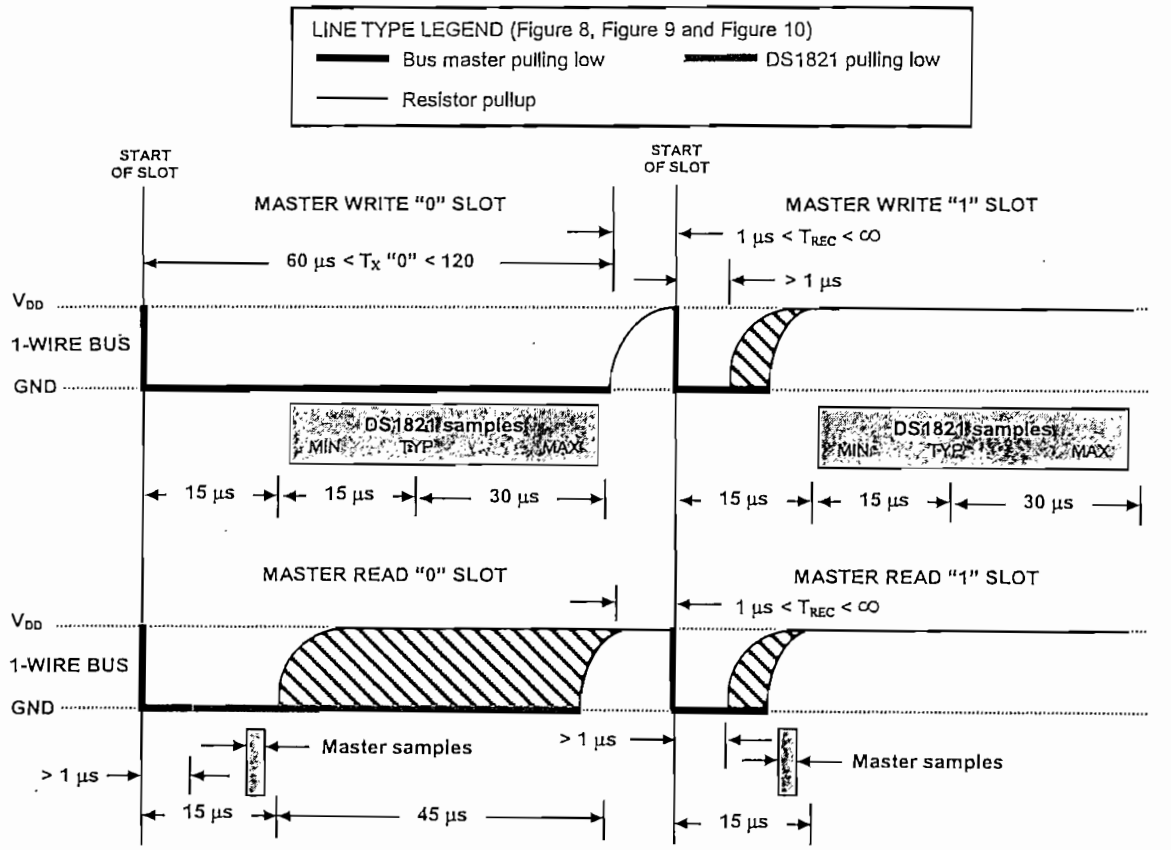
The DS1821 samples the 1-wire bus during a window that lasts from 15 µs to 60 µs after the master initiates the write time slot. If the bus is high during the sampling window, a 1 is written to the DS1821. If the line is low, a 0 is written to the DS1821.

READ TIME SLOTS

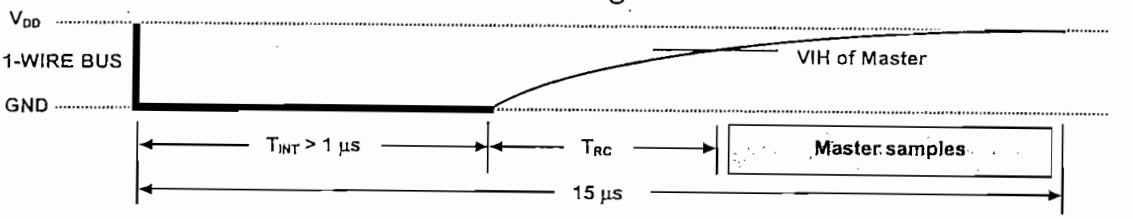
The DS1821 can only transmit data to the master when the master issues read time slots. Therefore, the master must generate read time slots immediately after issuing a read command (e.g., Read Temperature [AAh]), so that the DS1821 can provide the requested data. All read time slots must be a minimum of 60 µs in duration with a minimum of a 1 µs recovery time between slots. A read time slot is initiated by the master device pulling the 1-wire bus low for a minimum of 1 µs and then releasing the bus (see Figure 8). After the master initiates the read time slot, the DS1821 will begin transmitting a 1 or 0 on bus. The DS1821 transmits a 1 by leaving the bus high and transmits a 0 by pulling the bus low. When transmitting a 0, the DS1821 will release the bus by the end of the time slot, and the bus will be pulled back to its high idle state by the pullup resister. Output data from the DS1821 is valid for 15 µs after the falling edge that initiated the read time slot. Therefore, the master must release the bus and then sample the bus state within 15 µs from the start of the slot.

Figure 9 illustrates that the sum of T_{INIT}, T_{RC}, and T_{SAMPLE} must be less than 15 µs for a read time slot. Figure 10 shows that system timing margin is maximized by keeping T_{INIT} and T_{RC} as short as possible and by locating the master sample time during read time slots towards the end of the 15 µs period.

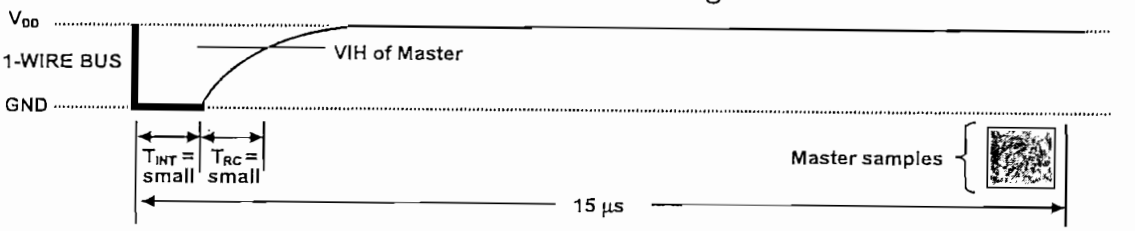
READ/WRITE TIME SLOT TIMING DIAGRAM Figure 8



DETAILED MASTER READ 1 TIMING Figure 9



RECOMMENDED MASTER READ 1 TIMING Figure 10



RELATED APPLICATION NOTES

The following Application Notes pertain to the DS1821. These notes can be obtained from the Dallas Semiconductor "Application Note Book," via the Dallas website at http://www.dalsemi.com or through our faxback service at (214) 450–0441.

Application Note 67: "Applying and Using the DS1620 in Temperature Control Applications"
Application Note 74: "Reading and Writing Touch Memories via Serial Interfaces"
Application Note 105: "High Resolution Temperature Measurement with Dallas Direct-to-Digital Temperature Sensors"

Sample 1-wire subroutines that can be used in conjunction with AN74 can be downloaded from the Dallas website or anonymous FTP Site.

DS1821 OPERATION EXAMPLE

In this example, the master device programs the DS1821 with T_L = +10°C and T_H = +40°C and verifies that the data has been saved correctly. The master then programs the status/configuration register so that the device will power-up in thermostat mode ($T/\overline{R}$ = 1) and the thermostat output will have active high polarity (POL = 1).

MASTER MODE	DATA (LSB FIRST)	COMMENTS
TX	Reset	Master issues reset pulse.
RX	Presence	DS1821 responds with presence pulse.
TX	01h	Master issues Write TH command.
TX	28h	Master sends data for T_H = +40°C.
TX	Reset	Master issues reset pulse.
RX	Presence	DS1821 responds with presence pulse.
TX	02h	Master issues Write TL command.
TX	0Ah	Master sends data for T_L = +10°C.
TX	Reset	Master issues reset pulse.
RX	Presence	DS1821 responds with presence pulse.
TX	A1h	Master issues Read TH command.
RX	28h	Master reads stored T_H value to verify data.
TX	Reset	Master issues reset pulse.
RX	Presence	DS1821 responds with presence pulse.
TX	A2h	Master issues Read TL command.
RX	0Ah	Master reads stored T_L value to verify data.
TX	Reset	Master issues reset pulse.
RX	Presence	DS1821 responds with presence pulse.
TX	0Ch	Master issues Write Status command.
TX	06h	Master sends status/configuration data to the DS1821 with $T/\overline{R}$ = 1 (thermostat mode at power-up) and POL = 1 (active high thermostat output).
—	—	Power is cycled; DS1821 powers-up in thermostat mode.

ABSOLUTE MAXIMUM RATINGS*

Voltage on any pin relative to ground	–0.5V to +7.0V
Operating temperature	–55°C to +125°C
Storage temperature	–55°C to +125°C
Soldering temperature	See-JTD-020A Specification

*These are stress ratings only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

DC ELECTRICAL CHARACTERISTICS (-55°C to +125°C; V_{DD}=2.7V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	V_{DD}		+2.7		+5.5		1
Thermometer Error	t_{ERR}	0°C to +85°C V_{DD} = 3.6V to 5.5V			±1	°C	2,3,4
		-55°C to +125°C V_{DD} = 3.6V to 5.5V		See Typical Curve (Figure 11)			
DQ Logic Low	V_{IL}		-0.3		+0.8	V	1,5
DQ Logic High	V_{IH}		+2		The lower of +5.5 or V_{DD} + 0.3	V	1,6
Sink Current	I_L	V_{DQ} = 0.4V V_{DD} = 3.6V to 5.5V	4			mA	1
Standby Current	I_Q	-55°C to +85°C		1	3	µA	7
Active Current	I_{DD}	V_{DD}= 5V		500	1000	µA	8
DQ Input Current	I_{DQ}			5		µA	9

NOTES:

1. All voltages are referenced to ground.
2. Thermometer error reflects the sensor accuracy as tested during calibration.
3. See typical performance curve in Figure 11 for specification limits outside the 0°C to +85°C range.
4. For T<0°C, accuracy degrades by 0.5°C/V for V_{DD}<4.3V.
5. Logic low voltages are specified at a sink current of 4 mA.
6. Logic high voltages are specified at a source current of 1 mA.
7. Standby current is typically 5 µA at 125°C.
8. Active current refers to supply current during active temperature conversions or EEPROM writes.
9. DQ line is high ("hi-Z" state).

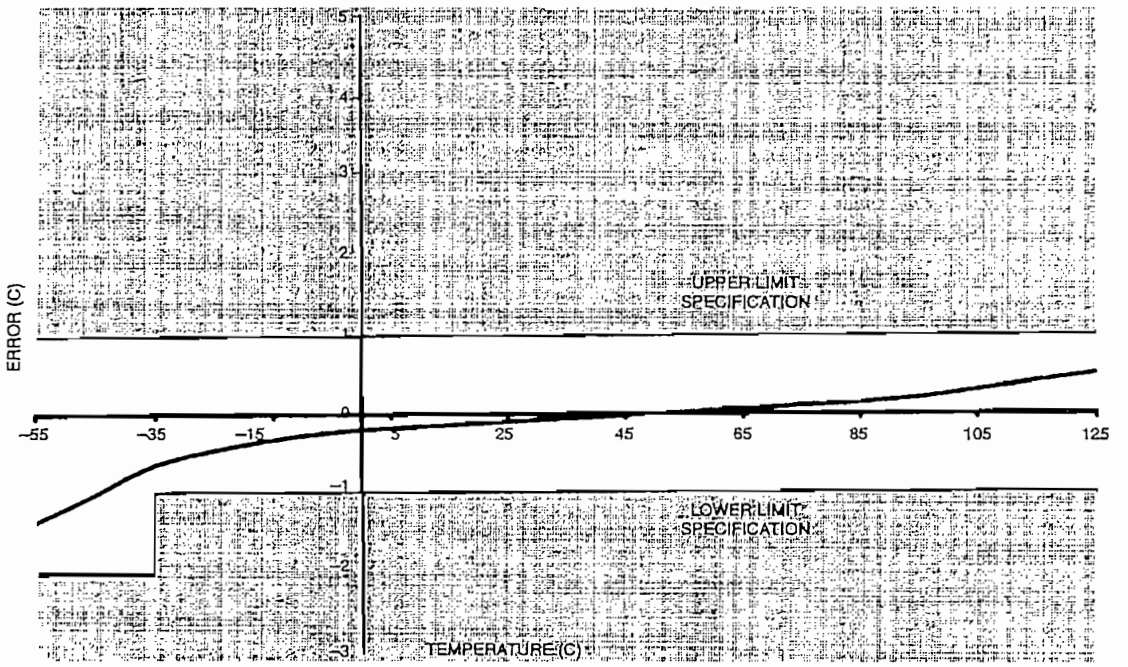
AC ELECTRICAL CHARACTERISTICS: (-55°C to +125°C; V_{DD}=3.6V to 5.5V)

PARAMETER	SYMBOL	MIN	TYP	MAX	UNITS	NOTES
Temperature Conversion Time	t_{CONV}		0.4	1.0	s	
EEPROM Write Time	t_{WR}		10	50	ms	
Time Slot	t_{SLOT}	60		120	µs	1
Recovery Time	t_{REC}	1			µs	1
Write 0 Low Time	t_{LOW0}	60		120	µs	1
Write 1 Low Time	t_{LOW1}	1		15	µs	1
Read Data Valid	t_{RDV}			15	µs	1
Reset Time High	t_{RSTH}	480			µs	1
Reset Time Low	t_{RSTL}	480			µs	1,2
Presence Detect High	t_{PDHIGH}	15		60	µs	1
Presence Detect Low	t_{PDLOW}	60		240	µs	1
V_{DD} Low to Mode Toggle Clock Low	t_{PC}	100			ns	1,3
Mode Toggle Clock 16 High to V_{DD} High	t_{CP}	100			ns	1
Mode Toggle Clock Pulse Low Time	t_{CL}	0.1		10	µs	1
Mode Toggle Clock Pulse High Time	t_{CH}	0.1			µs	1
Mode Toggle Clock High-to-Low or Low-to-High Transition Time	t_T			100	ns	1
Capacitance	$C_{IN/OUT}$			25	pF	

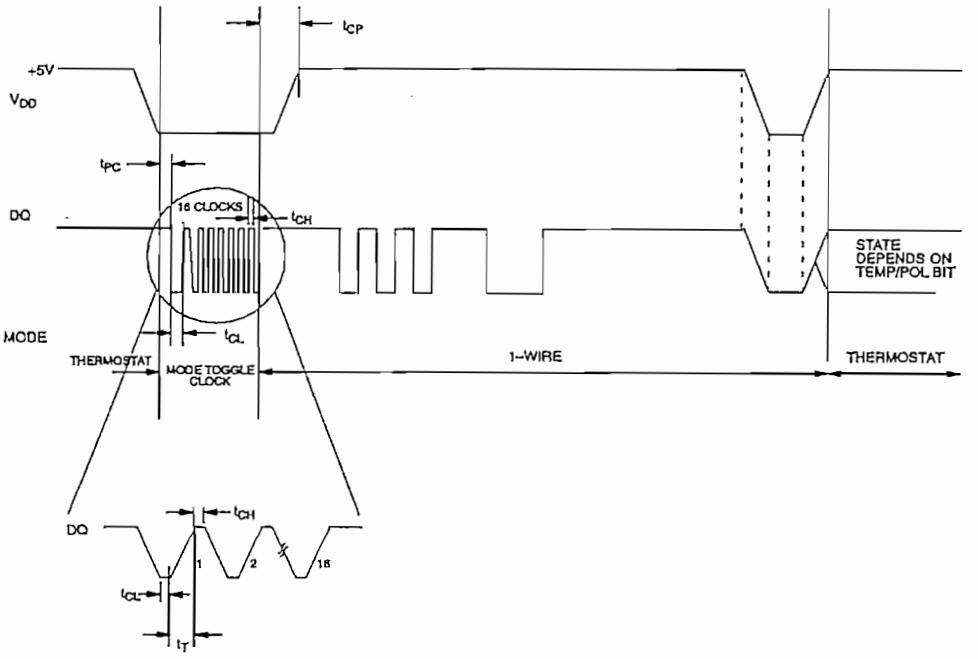
NOTES:

1. Refer to timing diagrams in Figure 13.
2. If t_{RSTL} > 960 µs, a power-on-reset may occur.
3. Time required for part to disable thermostat output.

TYPICAL PERFORMANCE CURVE Figure 11

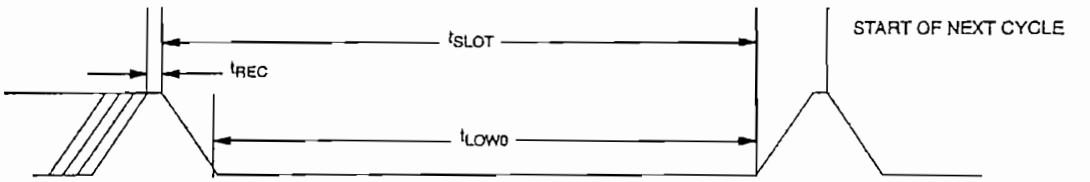


MODE TOGGLE TIMING WHEN T/$\overline{R}$ = 1 Figure 12

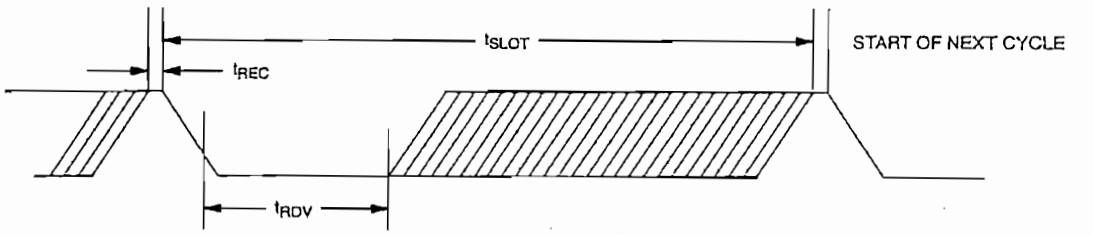


TIMING DIAGRAMS Figure 13

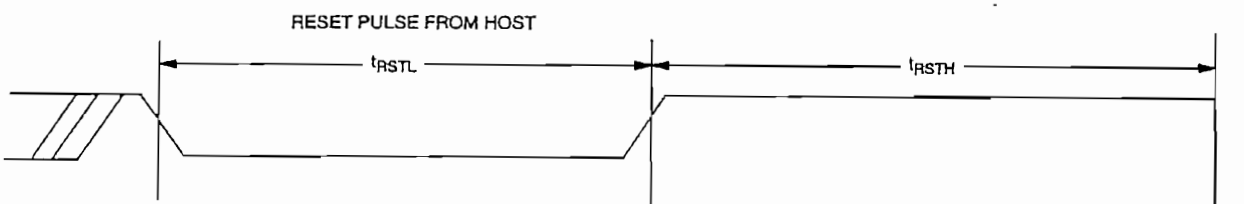
1–WIRE WRITE ZERO TIME SLOT



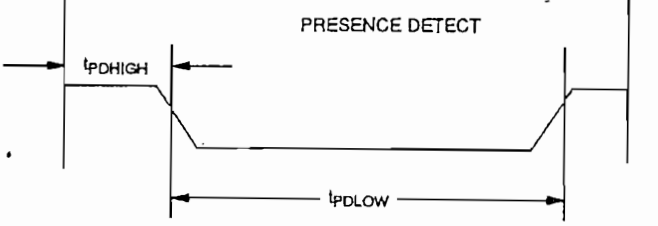
1–WIRE READ ZERO TIME SLOT



1–WIRE RESET PULSE



1–WIRE PRESENCE DETECT

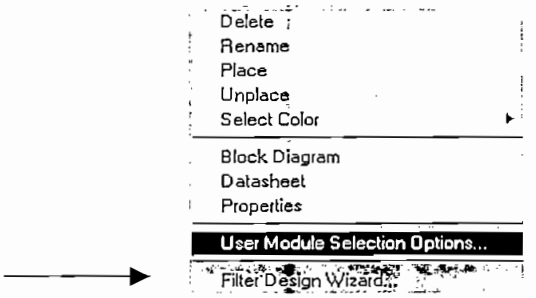


ANEXO D

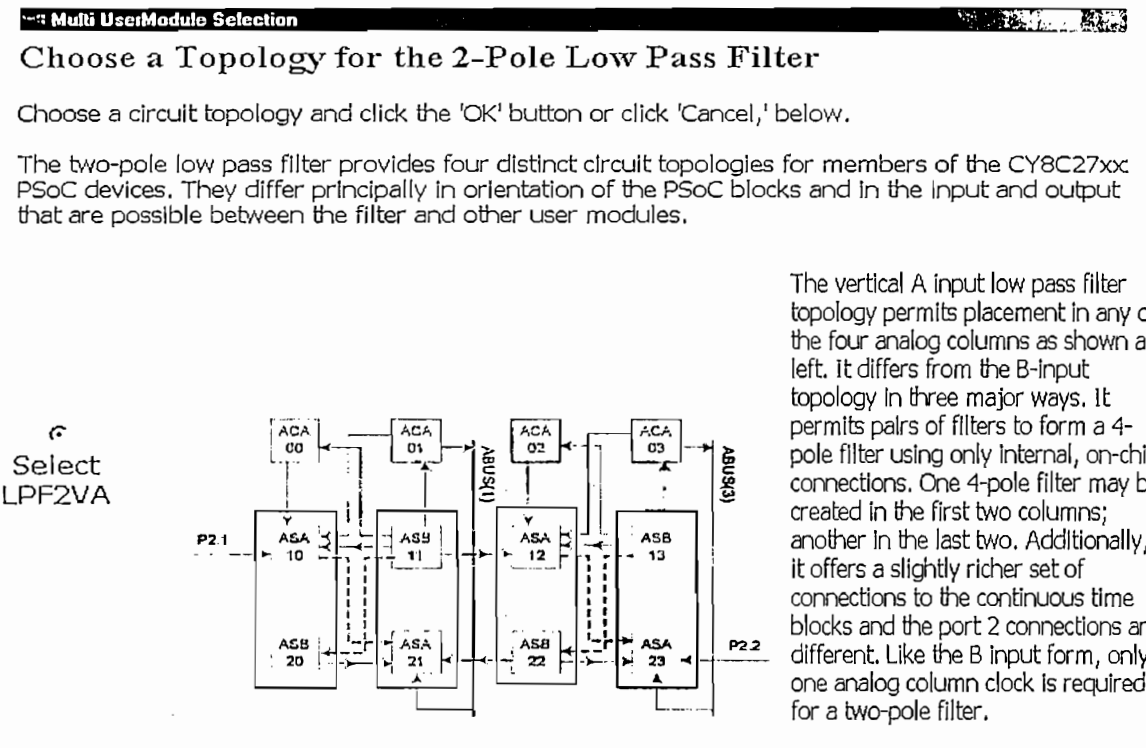
TOPOLOGÍA Y PARÁMETROS PARA DISEÑO DE UN FILTRO.
FRECUENCIAS PARA UNA OCTAVA DE TONOS MUSICALES.

TOPOLOGÍA Y PARÁMETROS PARA EL DISEÑO DE UN FILTRO

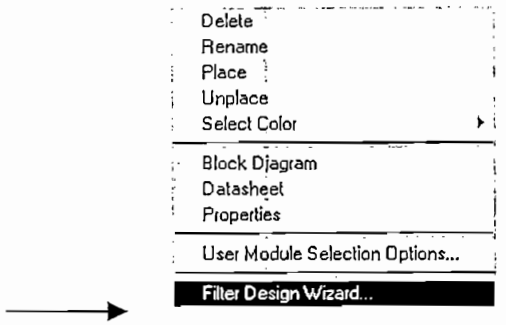
Cuando el Módulo de Usuario para un filtro ha sido enrolado en el programa, se da un click derecho en este, con lo cual la siguiente información aparece:



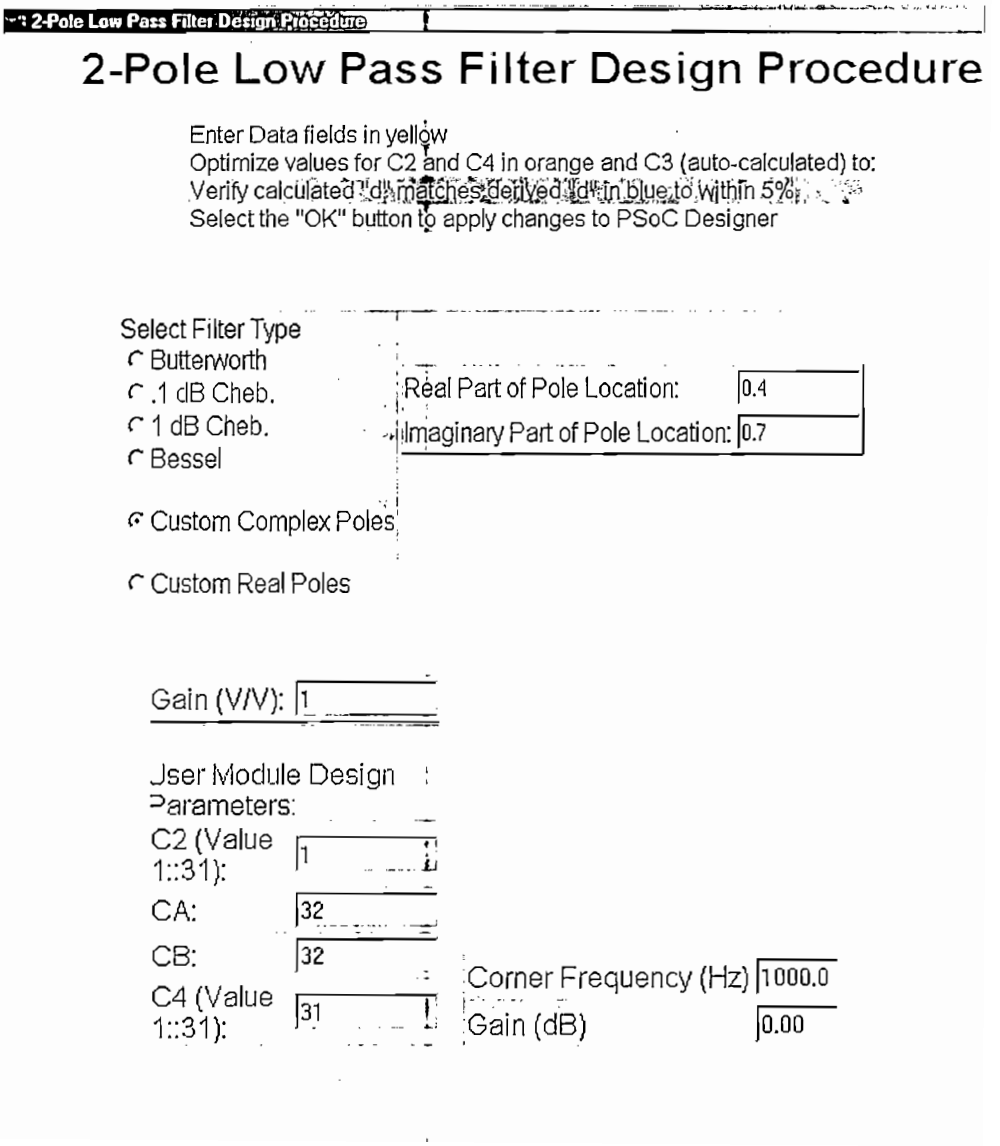
Se selecciona la opción marcada para poder observar la topología escogida:

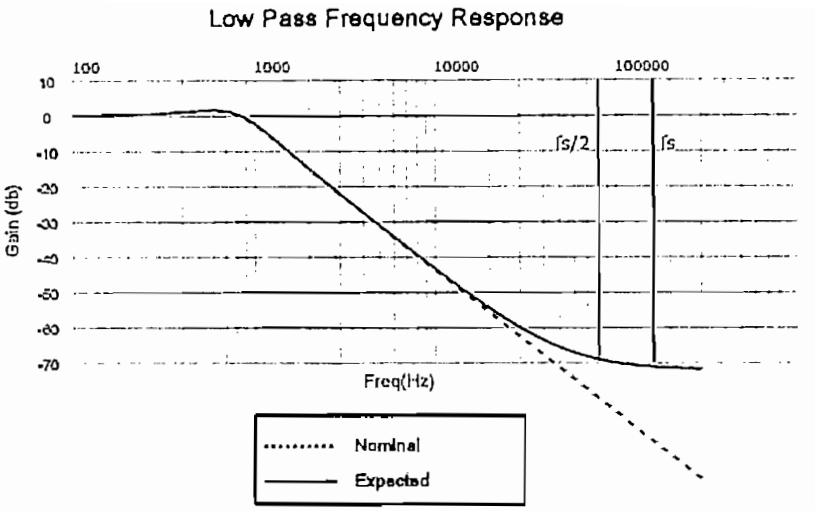


Ahora se realiza el mismo procedimiento anterior con la diferencia que se escoge la siguiente opción:



Con esta acción aparecerá:





Se pueden observar entonces, todos los procedimientos para el diseño del filtro que se quiera llevar a cabo.

FRECUENCIAS PARA UNA OCTAVA DE TONOS MUSICALES

OCT #	C	C#	D	D#	E	F	F#	G	G#	A	A#	B
1	32.703	34.648	36.708	38.891	41.203	43.654	46.249	48.999	51.913	55.000	58.270	61.735
2	65.406	69.296	73.416	77.782	82.407	87.307	92.499	97.999	103.83	110.00	116.54	123.47
3	130.81	138.59	146.83	155.56	164.81	174.61	185.00	196.00	207.65	220.00	233.08	246.94
4	261.63	277.18	293.66	311.13	329.63	349.23	369.99	392.00	415.30	440.00	466.16	493.88
5	523.25	554.37	587.33	622.25	659.26	698.46	739.99	783.99	830.61	880.00	932.33	987.77
6	1046.5	1108.7	1174.7	1244.5	1318.5	1396.9	1480.0	1568.0	1661.2	1760.0	1864.7	1975.5
7	2093.0	2217.5	2349.3	2489.0	2637.0	2793.8	2960.0	3136.0	3322.4	3520.0	3729.3	3951.1
8	4186.0	4434.9	4698.6	4978.0	5274.0	5587.7	5919.9	6271.9	6644.9	7040.0	7458.6	7902.1

ANEXO E

CÓDIGOS FUENTE.

CÓDIGO FUENTE CONFIGURACIÓN DE I/O PARA LECTURA Y ESCRITURA EN LOS PUERTOS

```
;-----------------------------------------------------------------

include "m8c.inc"        ; Parte especifica, constantes y macros.
include "PSoCAPI.h"      // definiciones PSOC API para todos
                         // los Módulos de Usuario.
export _main; Designación de una etiqueta global.

_main:

M8C_SetBank1 ;Ir a banco1, registros Drive Mode están en el banco1.
mov reg[PRT2DM0], FFh ;Configura pines del puerto2 como salidas
                      ;(STRONG DRIVE).
mov reg[PRT2DM1], 00h ;DM0=1 DM1=0 para strong drive.
mov reg[PRT0DM0], 00h ;Configura pines del puerto0 como entradas.
mov reg[PRT0DM1], FFh ;DM0=0 DM1=1 para High Z.
LAZO:
M8C_SetBank0; Regreso al banco0.
mov A,reg[PRT0DR] ;Se mueve lo que está en el puerto0 al A.
MOV reg[PRT2DR],A ;Lo del A se mueve al puerto2
nop
JMP LAZO
    ret
```

CÓDIGO FUENTE MANEJO DE UN TECLADO MATRICIAL Y VISUALIZACIÓN EN UN DISPLAY

Programa principal:

```
;----------------------------------------
; Este programa lee el teclado en el Puerto1
; y controla el display en el Puerto0
;----------------------------------------
;
include "m8c.inc"; Parte específica, constantes y macros.
export _main; Designación de una etiqueta global.

_main:

   loop:
      call lecturateclado; LLamado a una subrutina para lectura del
                         ; teclado que está en un archivo adicional.
      index tecladisp; Instrucción usada para recuperar información
                     ; desde una tabla hacia el A.
      mov reg[PRT0DR],A; Mueve lo del A al puerto0.
   jmp loop
ret
```

```
;                 a b c d e f g
;null       00h   1 1 1|1 1 1 1   7fh
;1    c0r0  11h   1 0 0|1 1 1 1   4fh
;2    c1r0  21h   0 0 1|0 0 1 0   12h
;3    c2r0  41h   0 0 0|0 1 1 0   06h
;4    c0r1  12h   1 0 0|1 1 0 0   4ch
;5    c1r1  22h   0 1 0|0 1 0 0   24h
;6    c2r1  42h   0 1 0|0 0 0 0   20h
;7    c0r2  14h   0 0 0|1 1 1 1   0fh
;8    c1r2  24h   0 0 0|0 0 0 0   00h
;9    c3r2  44h   0 0 0|0 1 0 0   04h
;*    c0r3  18h   1 1 1|1 0 0 0   78h
;0    c1r3  28h   0 0 0|0 0 0 1   01h
;#    c2r3  48h   0 0 1|1 0 0 0   18h
;indefinida       0 1 1|0 0 0 0   30h

xxh: equ 30h
tecladisp:
;       0   1   2   3   4   5   6   7   8   9   a  b   c   d   e   f
   db 7fh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh
   db xxh,4fh,4ch,xxh,0fh,xxh,xxh,xxh,78h,xxh,xxh,xxh,xxh,xxh,xxh,xxh
   db xxh,12h,24h,xxh,00h,xxh,xxh,xxh,01h,xxh,xxh,xxh,xxh,xxh,xxh,xxh
   db xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh
   db xxh,06h,20h,xxh,04h,xxh,xxh,xxh,18h,xxh,xxh,xxh,xxh,xxh,xxh,xxh
   db xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh
   db xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh
   db xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh,xxh
```

Rutina Lectura de teclado:

```
;-----------------------------------------
;  Esta rutina sirve para leer un teclado 4*3
;  en el Puerto1. Los estados de las teclas
;  presionadas se ponen en el A.
;  Esta rutina se encuentra en el archivo teclado.asm.
;  Pines del P1 para filas y columnas:
;  P1.0 R0
;  P1.1 R1
;  P1.2 R2
;  P1.3 R3
;  P1.4 C0
;  P1.5 C1
;  P1.6 C2
;-----------------------------------------
;

export    lecturateclado; Designación de etiquetas globales, que en
                        ; este caso se referencian en otro archivo.
export   _lecturateclado
;
include "m8c.inc"; Parte especifica, constantes y macros.
;
```

```
 lecturateclado:
_lecturateclado:
   mov reg[PRT1DR],F0h; El movimiento de este valor al puerto1,
                      ; se usa para el manejo de columnas.
   push X; El valor de X se transfiere al tope del stack.
         ; El SP es incrementado.
   mov X,SP; Recuperación del nuevo valor en X.
   mov A,reg[PRT1DR]; El valor del puerto1 se mueve al A,
                     ; para lectura de filas
   mov reg[PRT2DR],0Fh; El movimiento de este valor al puerto2,'
                      ; se usa para el manejo de filas.
   push A; El valor del A se transfiere al tope del stack.
         ; El SP es incrementado.
   mov A,reg[PRT1DR]; El valor del puerto1 se mueve al A,
                     ; para lectura de columnas
   and [X],A; Operación para combinación de ambas.
   pop A; Remueve el último byte ubicado en el stack para ponerlo
        ; en el A.
        ; El SP es incrementado.
   pop X; Remueve el último byte ubicado en el stack para ponerlo
        ; en X.
ret
```

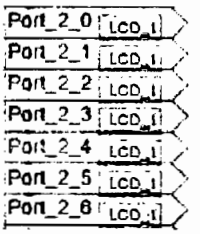
CÓDIGO FUENTE MANEJO BÁSICO DEL LCD

```
;-------------------------------------------------------------------------

include <m8c.h>         // Parte específica, constantes y macros
include "PSoCAPI.h"     // definiciones PSOC API para todos
                        // los Módulos de Usuario
include "lcd_1.inc"; Para manejo de las rutinas del Módulo de Usuario
                   ; LCD.
export _main

 area texto(ROM,REL); Especiíca que código o datos están guardados
                    ; en la Flash, permitiendo al linker relocalizarlos

_main:

    call LCD_1_Start; LLamado rutina para inicializar el Módulo LCD.
    mov A,00h; Carga de valor para posición LCD.
    mov X,03h; Carga de valor para posición LCD.
    call LCD_1_Position; LLamado rutina posicionamiento LCD.
    mov   A,>THE_STR; Cargar el puntero al string de la ROM
    mov   X,<THE_STR
    call   LCD_1_PrCString; LLamado rutina para impresión del string.
loop:
    jmp loop
    .LITERAL
    THE_STR:
    DS "hola mundo"; Texto impreso para demostración.
    DB 00h
    .ENDLITERAL
```

Visualización del módulo de usuario que interviene en esta aplicación:

LCD_1

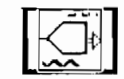
Interconexión del módulo:



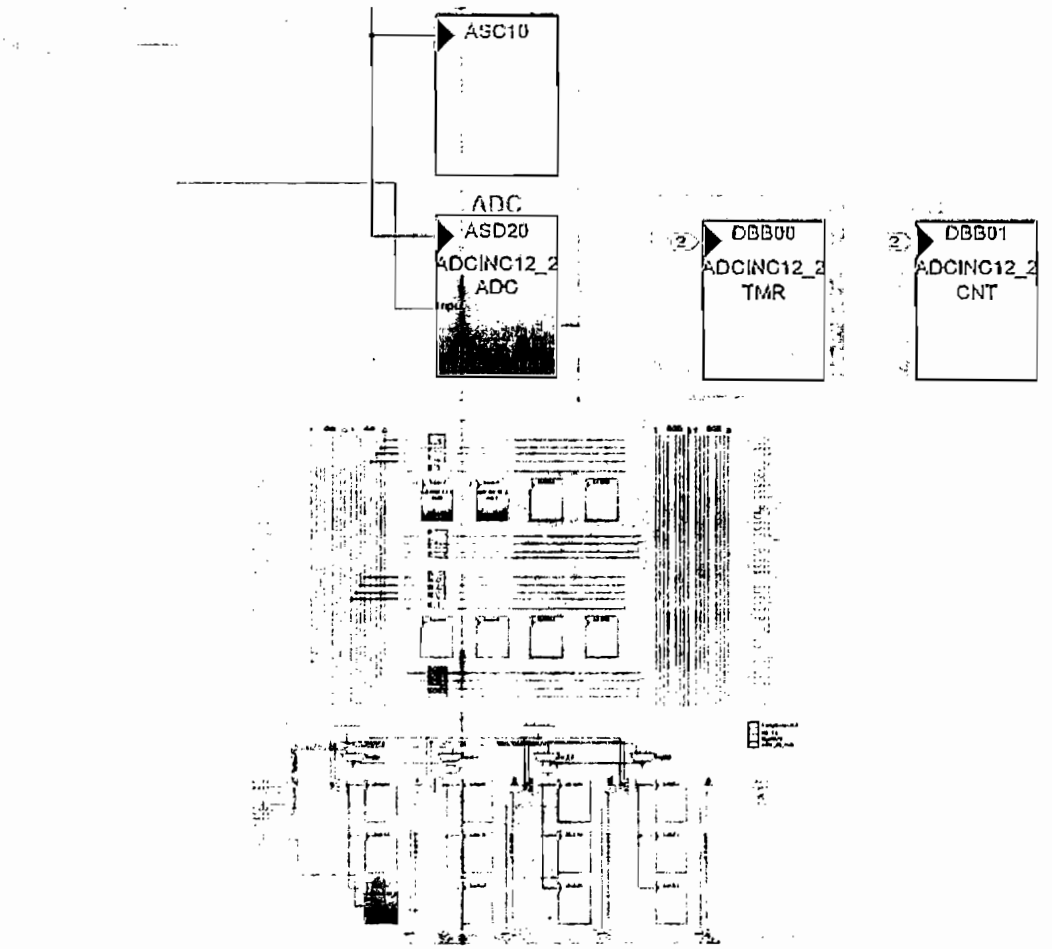
CÓDIGO FUENTE USO DEL CONVERSOR ANÁLOGO DIGITAL DE 12 BITS

```
;-----------------------------------------
include "m8c.inc"         // Parte especifica, constantes y macros.
include "PSoCAPI.h"       // definiciones PSOC API para todos
                         // los Módulos de Usuario.
include "adcinc12_1.inc" ; Para manejo de las rutinas del Módulo
                          ; de Usuario ADCINC12.
export _main
_main:
   mov  A,ADCINC12_1_HIGHPOWER; Elección nivel de potencia
                              ; para el conversor.
   call ADCINC12_1_Start; LLamado rutina para inicialización
                        ; del conversor.
   M8C_EnableGInt; Habilitación de la interrupción global.
   mov  A,0; Limpieza del A.
   call ADCINC12_1_GetSamples; LLamado rutina para muestreo.
loop:
   call ADCINC12_1_ClearFlag; Reseteo de la bandera de
                            ; los datos disponibles.
;esperar por datos válidos.
Stop_datosvalidos:
   call ADCINC12_1_fIsData; LLamado rutina para chequeo del estado
                          ; del ADC.
   jz  Stop_datosvalidos
   call ADCINC12_1_iGetData; Retorno de los datos convertidos.

   mov reg[PRT0DR],A   ;LSB al puerto0
   mov A,X; Descargo de los datos del registro X al A.
   mov reg[PRT1DR],A   ;MSB al puerto1

   jmp loop
   ret
```

Visualización del módulo de usuario que interviene en esta aplicación:



Interconexión del módulo:

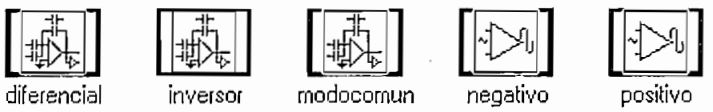


CÓDIGO FUENTE DISEÑO DE UN AMPLIFICADOR DIFERENCIAL

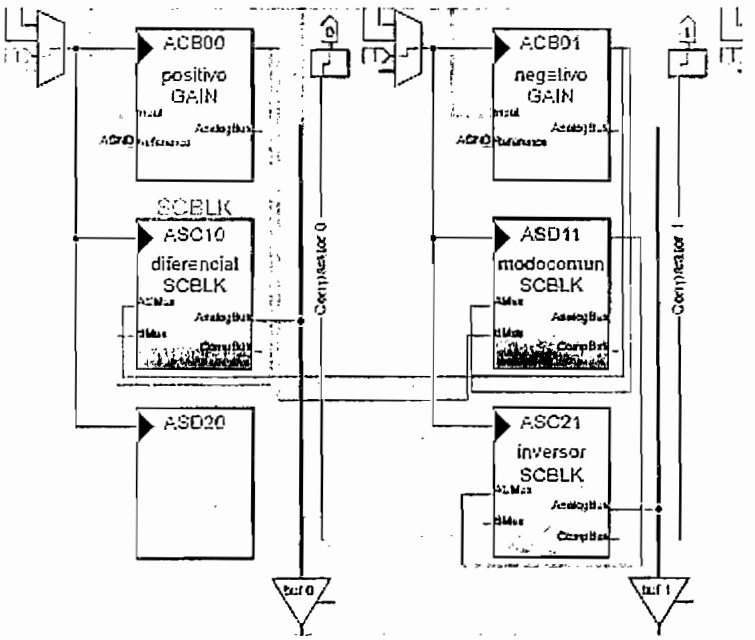
```
;-----------------------------------------------------------------------

include "m8c.inc"        ; Parte especifica, constantes y macros.
include "PSoCAPI.inc"    // Definiciones PSOC API para todos
                         // los Módulos de Usuario.

export _main
```

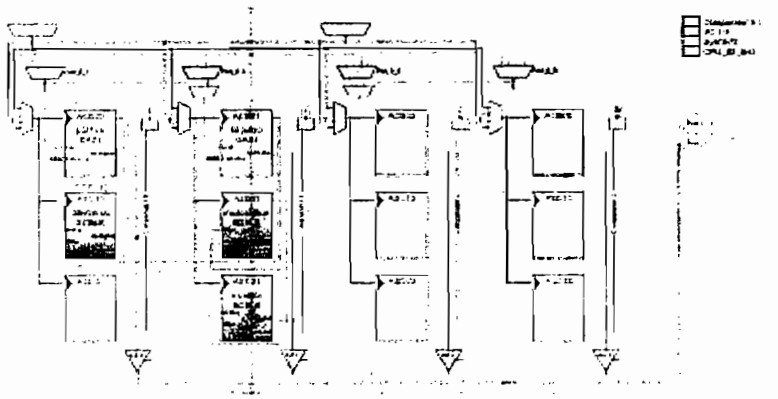
```
_main:

mov A,positivo_HIGHPOWER; Elección nivel de potencia para el bloque
                         ; PGA que actuará como entrada positiva.
call positivo_Start; LLamado rutina para inicialización PGA (+).
mov A,negativo_HIGHPOWER; Elección nivel de potencia para el bloque
                         ; PGA que actuará como entrada negativa.
call negativo_Start; LLamado rutina para inicialización PGA (-).
mov A,inversor_HIGHPOWER; Elección nivel de potencia para el bloque
                         ; SC que actuará como inversor.
call inversor_Start; LLamado rutina para inicialización SC inversor.
mov A,diferencial_HIGHPOWER; Elección nivel de potencia para el bloque
                           ; SC que actuará como diferencial.
call diferencial_Start;LLamado rutina para inicialización SC diferencial.
mov A,modocomun_HIGHPOWER;Elección nivel de potencia para el bloque
                         ;SC que actuará como modo común.

call modocomun_Start;LLamado rutina para inicialización SC modo común.

ret
```

Visualización de los módulos de usuario que intervienen en esta aplicación:



Interconexión de los módulos:



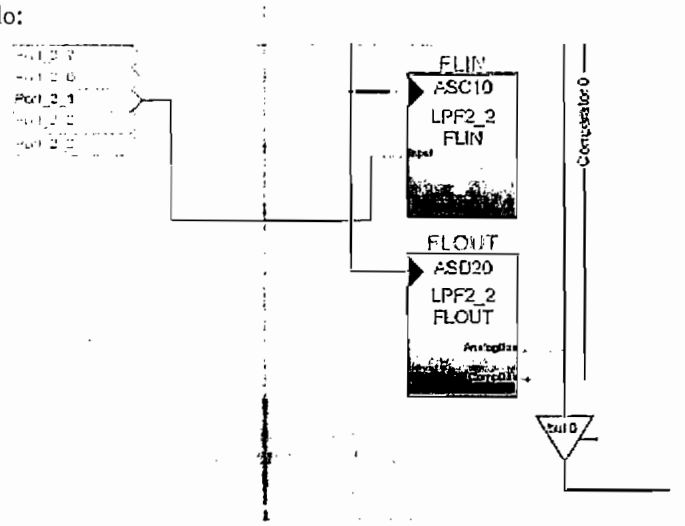


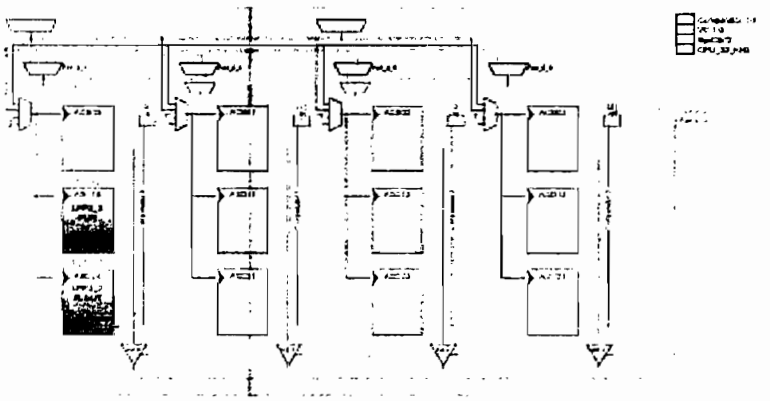
CÓDIGO FUENTE IMPLEMENTACIÓN DE FILTROS

```
;-------------------------------------------------------------------------
include "m8c.inc"        ; Parte especifica, constantes y macros.
include "PSoCAPI.inc"    ; Definiciones PSoC API para todos
                         ;los Módulos de Usuario.
export _main
_main:
    lcall  LPF2_2_Start; LLamado de la rutina para inicialización
                       ; del filtro.
    ret
```

Visualización del módulo de usuario que interviene en esta aplicación:

Interconexión del módulo:





CÓDIGO FUENTE GENERADOR DE SONIDO

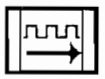
```
;-----------------------------------------------------------------------
include "m8c.inc"      ; Parte especifica, constantes y macros.
include "PWM16_1.inc" ; Para manejo de las rutinas del Módulo de Usuario
                       ; PWM16.

export _main

area bss(ram); Para especificar que datos están almacenados en la RAM.

TEMP3:       BLK 1; Para reservar bloques de la RAM en bytes. Se
                  ; especifica también el tamaño en bytes de este
                  ; bloque (1 byte).
TEMP4:       BLK 1

area text(rom); Especifica que código o datos están guardados en la
              ; Flash.

_main:

beginning1:
    mov A,reg[PRT2DR]// Valor del puerto2 se mueve al A.
    mov [TEMP4],A// Valor del A se mueve a TEMP4.
beginning:
    mov A,reg[PRT2DR]// Valor del puerto2 se mueve al A.
    mov [TEMP3],A// Valor actual del puerto2 se mueve a TEMP3.
    cmp A,[TEMP4]// Comparar los valores previo y actual del puerto2.
    jz beginning
    mov [TEMP4],[TEMP3]// Mover el último valor del puerto2 a TEMP4 para
                       // que así pueda compararse con el próximo valor
                       // del mismo.
    call PWM16_1_Stop // Si el valor anterior es diferente que el
                      // presente, el PWM16 se detiene.
```

```
mismo_pulsante:
    mov [TEMP4],[TEMP3]// Mover el último valor del puerto2 a TEMP4 para
                       // que así pueda compararse con el próximo valor
                       // del mismo.
    mov A,[TEMP3] // Chequea si el pulsante1 es presionado.
    and A,01h // Operación AND con el valor 01h.
    jz pulsante1 // Si es presionado el programa va a la etiqueta
                 // pulsante1.
    mov A,[TEMP3] // Chequea si el pulsante2 es presionado.
    and A,02h // Operación AND con el valor 02h.
    jz pulsante2 // Si es presionado el programa va a la etiqueta
                 // pulsante2.
    mov A,[TEMP3] // Chequea si el pulsante3 es presionado.
    and A,04h // Operación AND con el valor 04h.
    jz pulsante3 // Si es presionado el programa va a la etiqueta
                 // pulsante3.
    mov A,[TEMP3] // Chequea si el pulsante4 es presionado.
    and A,08h // Operación AND con el valor 08h.
    jz pulsante4 // Si es presionado el programa va a la etiqueta
                 // pulsante4.
    mov A,[TEMP3] // Chequea si el pulsante5 es presionado.
    and A,10h // Operación AND con el valor 10h.
    jz pulsante5 // Si es presionado el programa va a la etiqueta
                 // pulsante5.
    mov A,[TEMP3] // Chequea si el pulsante6 es presionado.
    and A,20h // Operación AND con el valor 20h.
    jz pulsante6 // Si es presionado el programa va a la etiqueta
                 // pulsante6.
    mov A,[TEMP3] // Chequea si el pulsante7 es presionado.
    and A,40h // Operación AND con el valor 40h.
    jz pulsante7 // Si es presionado el programa va a la etiqueta
                 // pulsante7.
    mov A,[TEMP3] // Chequea si el pulsante8 es presionado.
    and A,80h // Operación AND con el valor 80h.
    jz pulsanten8 // Si es presionado el programa va a la etiqueta
                 // pulsante8.
    jmp beginning // Si ningún pulsante se presiona, regresa a beginning

pulsante1: // Para la nota musical C.
    mov X,B3h // Se carga a X con el MSB del periodo.
    mov A,2Ah // Se carga al A con el LSB del periodo.
    call PWM16_1_WritePeriod //Llamado de rutina para escribir el
                             // periodo del PWM.
    mov X,59h // Se carga a X con el MSB del ancho de pulso.
    mov A,95h // Se carga al A con el LSB del ancho de pulso.
    call PWM16_1_WritePulseWidth //Llamado de rutina para escribir el
                             // ancho de pulso del PWM.
    call PWM16_1_Start // Después de escribir el periodo y el ancho
                       // de pulso, el PWM se inicializa.
    jmp beginning // Regresa a esta etiqueta para chequear si hay algún
                  // nuevo valor en el puerto2.
```

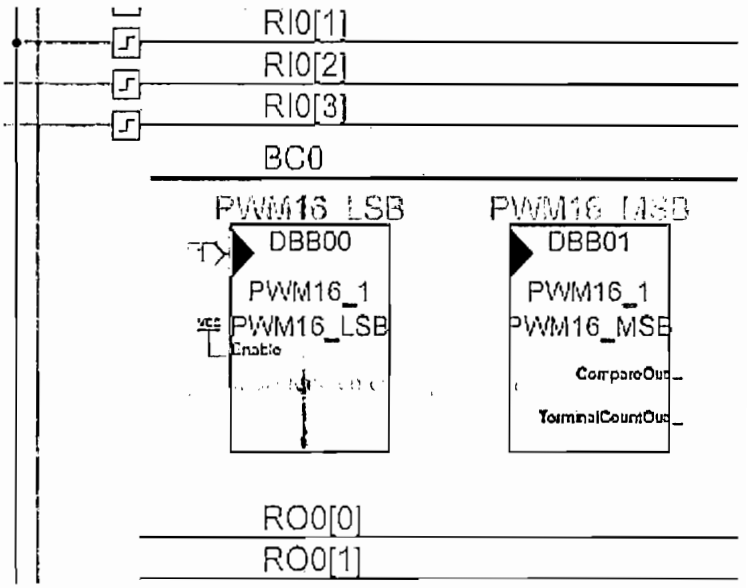
```
pulsante2: // Para la nota musical D.
    mov X,9Fh // MSB del periodo.
    mov A,9Fh // LSB del periodo.
    call PWM16_1_WritePeriod // Se escribe el periodo del PWM.
    mov X,4Fh // MSB del ancho de pulso.
    mov A,CFh // LSB del ancho de pulso.
    call PWM16_1_WritePulseWidth // Se escribe el ancho de pulso del PWM.
    call PWM16_1_Start // Después de escribir el periodo y el ancho
                       // de pulso, el PWM se inicializa.
    jmp beginning // Regresa a esta etiqueta para chequear si hay algún
                  // nuevo valor en el puerto2.
pulsante3: // Para la nota musical E.
    mov X,8Eh // MSB del periodo.
    mov A,34h // LSB del periodo.
    call PWM16_1_WritePeriod
    mov X,47h // MSB del ancho de pulso.
    mov A,1Ah // LSB del ancho de pulso.
    call PWM16_1_WritePulseWidth
    call PWM16_1_Start
    jmp beginning
pulsante4: // Para la nota musical F.
    mov X,86h // MSB del periodo.
    mov A,39h // LSB del periodo.
    call PWM16_1_WritePeriod
    mov X,43h // MSB del ancho de pulso.
    mov A,1Ch // LSB del ancho de pulso.
    call PWM16_1_WritePulseWidth
    call PWM16_1_Start
    jmp beginning
pulsante5: // Para la nota musical G.
    mov X,77h // MSB del periodo.
    mov A,94h // LSB del periodo.
    call PWM16_1_WritePeriod
    mov X,3Bh // MSB del ancho de pulso.
    mov A,CAh // LSB del ancho de pulso.
    call PWM16_1_WritePulseWidth
    call PWM16_1_Start
    jmp beginning
pulsante6: // Para la nota musical A.
    mov X,6Ah // MSB del periodo.
    mov A,88h // LSB del periodo.
    call PWM16_1_WritePeriod
    mov X,35h // MSB del ancho de pulso.
    mov A,44h // LSB del ancho de pulso.
    call PWM16_1_WritePulseWidth
    call PWM16_1_Start
    jmp beginning
pulsante7: // Para la nota musical B.
    mov X,5Eh // MSB del periodo.
    mov A,E9h // LSB del periodo.
    call PWM16_1_WritePeriod
    mov X,2Fh // MSB del ancho de pulso.
    mov A,74h // LSB del ancho de pulso.
```

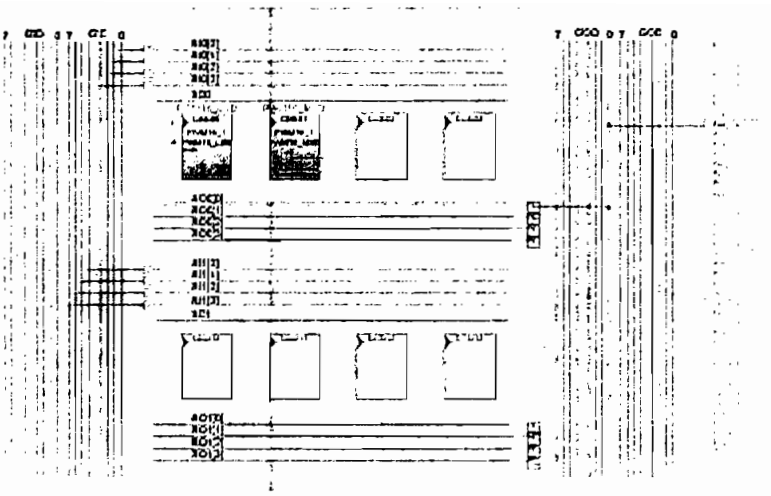
```
    call PWM16_1_WritePulseWidth
    call PWM16_1_Start
    jmp beginning
pulsante8: // Para la nota musical C de la siguiente Octava.
    mov X,59h // MSB del periodo.
    mov A,95h // LSB del periodo.
    call PWM16_1_WritePeriod
    mov X,2Ch // MSB del ancho de pulso.
    mov A,CAh // LSB del ancho de pulso.
    call PWM16_1_WritePulseWidth
    call PWM16_1_Start
    jmp beginning
```

Visualización del módulo de usuario que interviene en esta aplicación:



Interconexión del módulo:





CÓDIGO FUENTE MEDICIÓN DE TEMPERATURA USANDO EL SENSOR DALLAS DS1821

Programa principal:

```
//--------------------------------------------------------------------
// Archivos Include
//--------------------------------------------------------------------
#include <m8c.h>          // Parte específica, constantes y macros.
#include "PSoCAPI.h"      // Definiciones PSOC API para todos
                          // los Módulos de Usuario.
#include "define.h"       // Para definiciones de puerto y
                          // prototipos de función.
//--------------------------------------------------------
// Declaración de variables globales
//--------------------------------------------------------
extern char uval;
extern BYTE nuevovalor;
BYTE temp;
BYTE valor=0;
BYTE valoranterior;
BYTE chequeo;

//-----------------------------------------------
//Programa principal//
void main()
{
   //BYTE valoranterior;
   //BYTE nuevovalor;
   char theStr[] = "Temperatura:    C";  //Define string de la RAM.

   hacerpullup();
   salida_alto(DQ,pin);

   valoranterior=leer_temp();       //Lectura de temperatura.
```

```
   LCD_1_Start();                  // Inicializa el LCD.
   LCD_1_Position(0,0);            // Pone el cursor del LCD en
                                   // fila0, columna0.
   LCD_1_PrString(theStr);         // Imprime "mensaje" en el LCD

Imprimir:
   Imprimirdatos();                // Convierte e imprime los datos
   valoranterior=nuevovalor;

Nuevo:
   nuevovalor=leer_temp();
   if (nuevovalor==valoranterior) goto Nuevo;
   else                    goto Imprimir;
}

//---------------------------------------------------------------
//Presence
//---------------------------------------------------------------
BYTE ds_reset( void)

    {
        BYTE presence;

        salida_bajo(DQ,pin);
        DelayuFunction(220);   //retardo 500 us

        salida_alto(DQ,pin);
        DelayuFunction(42);    //retardo 100 us

        if(entrada(DQ,pin)) {presence=true;}
        else               {presence=false;}

        DelayuFunction(185);    //retardo 420 us

        return presence;
    }

//---------------------------------------------------------------------
// Lectura de un bit
//---------------------------------------------------------------------
BYTE  ds_r_bit(void)

    {
        salida_bajo(DQ,pin);
        asm("nop\n" "nop\n" "nop\n");  // retardo 1us
        salida_alto(DQ,pin);
        DelayuFunction(2);   // retardo 11 us ,

        if(entrada(DQ,pin)) {return(true);}
        else               {return(false);}

     }
```

```
//---------------------------------------------------------------
// Escritura de un bit
//---------------------------------------------------------------
void ds_w_bit(BYTE bitval)
    {
        salida_bajo(DQ,pin);
        DelayuFunction(3); // retardo 12 us
        if(bitval)
         {
           salida_alto(DQ,pin);
         }
        DelayuFunction(14); // retardo 36 us
        salida_alto(DQ,pin);
        return;
    }
//---------------------------------------------------------------
//Leer Byte
//---------------------------------------------------------------
BYTE ds_r_byte(void)
    {
       // valor entero=0;
        char i=0;
        char bit=0;
        valor=0;

            for(i=0;i<8;i++)
            {
               bit=ds_r_bit();
               if(bit) valor|=(1<<i);
               DelayuFunction(29);     // retardo 70 us
            }

        return(valor);

    }

//---------------------------------------------------------------
//Escribir Byte
//---------------------------------------------------------------
void ds_w_byte(BYTE val)
    {
        BYTE i,temp;
        for (i=0; i<8; i++)
          {
            DelayuFunction(29); // retardo 70 us
            temp=val>>i;
            temp&=1;
            ds_w_bit(temp);
          }

        return;
    }
```

```
//-----------------------------------------------------------------------
//Leer Temperatura
//-----------------------------------------------------------------------
BYTE leer_temp(void)
{
  static short start_conver=0;
  static BYTE  temp_copy;
  static BYTE  ds_data;
  static int   counter=0;

Nuevamente:

  if(ds_reset())
     {
      start_conver=false;
      temp=0;
      goto  salir;
     }

    else
      {
        if(!start_conver)
         {
              if(!ds_reset())
                {
                  ds_w_byte(0x0C);   //Escribir al registro Status.
                  ds_w_byte(0x00);   //Configuración registro Status.
                  LCD_1_Delay50uTimes(200); //Espera necesaria de 10ms.

                }
              else goto Nuevamente;

              if(!ds_reset())
                {
                  ds_w_byte(0xEE);    //Inicio de conversión.
                  start_conver=true;
                }
               else goto Nuevamente;

              if(!ds_reset())
                {
                  ds_w_byte(0xAA);    //Leer último valor del conversor.
                  ds_data=ds_r_byte();
                }
              else goto Nuevamente;
         }

        else
          {

              if (!ds_reset())
              {
                ds_w_byte(0xAA);
                ds_data=ds_r_byte();
```

```
                }

            }
        }

          if(ds_data<=99){ temp=ds_data; }
          else           { temp=99;      }

salir:
         if(temp_copy!=temp)
          {
            temp_copy=temp;
          }

  return temp;
}

//-----------------------------------------------------
//Función de retardo:
//Calculado para 12MHz
//retardo min. disponible=4.5us
//retardo max. disponible=578.25us
//-----------------------------------------------------
void DelayuFunction(char u)
{
              // llama a DelayuFunction [11]
   uval=u;    // mov A, u [5]
   Delayu(); // llama Delayu [11]
   return;    // ret [8]
}


//-----------------------------------------------
//Función hacerpullup:
//cambia el modo del puerto a pull up
//-----------------------------------------------
void hacerpullup (void)
{
 M8C_SetBank1; // Ir al banco1
 PRT2DM0=0x08; //00001000
 PRT2DM1=0x08; //00001000
 PRT2DM2=0x00; //00000000
 M8C_SetBank0; //Ir al banco0
 return;
}
```

Para escritura en el LCD:

```
include "m8c.inc" ; Parte especifica, constantes y macros.
include "LCD_1.inc" ; Para manejo de las rutinas del Módulo de Usuario
                    ; LCD.
export nuevovalor
```

```
area bss (RAM)

_nuevovalor::
 nuevovalor:: BLK 1

 res1:  BLK 1 ; Definición de variables.
 res2:  BLK 1
 res3:  BLK 1
 copy:  BLK 1

area text (ROM, REL) ; Especifica que código o datos están guardados
                     ; en la Flash, permitiendo al linker relocalizarlos

export  Imprimirdatos ; Designación de etiquetas globales, que en
                      ; este caso se referencian en otro archivo.
export _Imprimirdatos

 Imprimirdatos:
_Imprimirdatos:

   mov     [res1],00h          ;borrado de variables
   mov     [res2],00h
   mov     [res3],00h
   mov     A,[nuevovalor]

HEXto1:
   mov     [copy],A
   sub     A,100
   jc      HEXto2
   inc     [res3]              ;MSB del dato.
   jmp     HEXto1

HEXto2:
   mov     A,[copy]

HEXto3:
   mov     [copy],A
   sub     A,10
   jc      HEXto4
   inc     [res2]              ;SB medio del dato.
   jmp     HEXto3

HEXto4:
   mov     A,[copy]
   mov     [res1],A            ;LSB del dato.

   mov     A,00h               ; Poner posición cursor en fila = 0
   mov     X,0Ch               ; col = 12.
   call    LCD_1_Position ; Llamado de rutina para posicionamiento LCD.

   cmp [res3],00h
   jz  Pres2
```

```
   add     [res3],30h          ; Convierte a ASCII.
   mov     A,[res3]            ; Carga datos para escribirlos
                               ; al registro de control.
   call    LCD_1_Write_Data    ; Llamar a la función de escritura de datos.

Pres2:
   cmp [res2],00h
   jz  Pres1

   add     [res2],30h          ; Convierte a ASCII.
   mov     A,[res2]            ; Carga datos para escribirlos
                               ; al registro de control.
   call    LCD_1_Write_Data    ; Llamar a la función de escritura de datos.

Pres1:
   add     [res1],30h          ; Convierte a ASCII.
   mov     A,[res1]            ; Carga datos para escribirlos
                               ; al registro de control.
   call    LCD_1_Write_Data    ; Llamar a la función de escritura de datos.

   ret
```

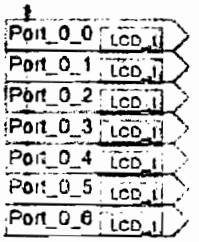
Para retardos:

```
include "m8c.inc" ; Parte específica, constantes y macros.

area bss (RAM); Para especificar que datos están almacenados en la RAM.

_uval::
 uval:: BLK 1 ; Para reservar bloques de la RAM en bytes. Se
              ; especifica también el tamaño en bytes de este
              ; bloque (1 byte).

 ucounter: BLK 1

area text (ROM, REL) ; Especifica que código o datos están guardados
                     ; en la Flash, permitiendo al linker relocalizarlos.
export  Delayu ; Designación de etiquetas globales, que en
               ; este caso se referencian en otro archivo.
export _Delayu

 Delayu:
_Delayu:
          mov A,[uval]        ;[5]   (u)
          mov [ucounter],A    ;[5]
uloop:    nop                 ;[4*u]
          nop                 ;[4*u]
          nop                 ;[4*u]
          nop                 ;[4*u]
          dec [ucounter]      ;[7*u]
          jnz uloop           ;[(4*u)+1]
          ret                 ;[8]
                              ; Total=(54+27u)
                              //Retardo=(54+27U)*1/freq
```

Visualización del módulo de usuario que interviene en esta aplicación:

LCD_1

Interconexión del módulo:

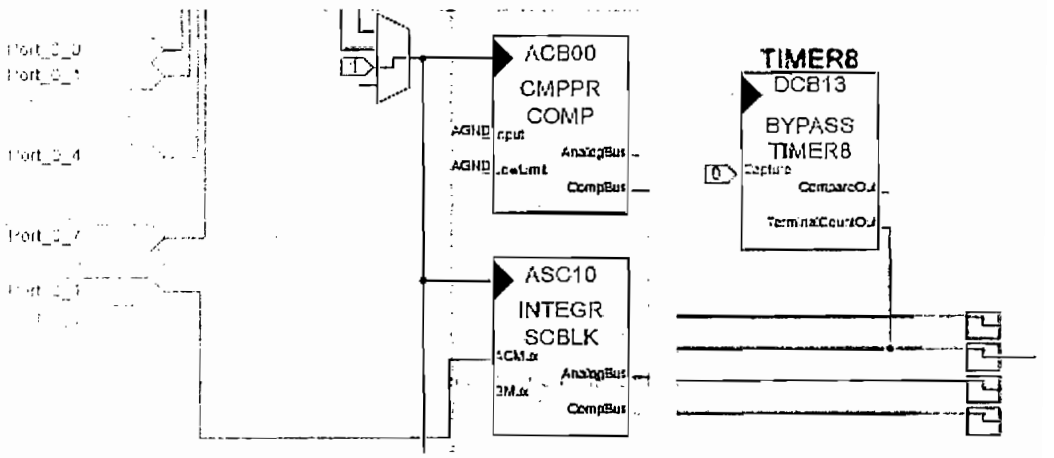


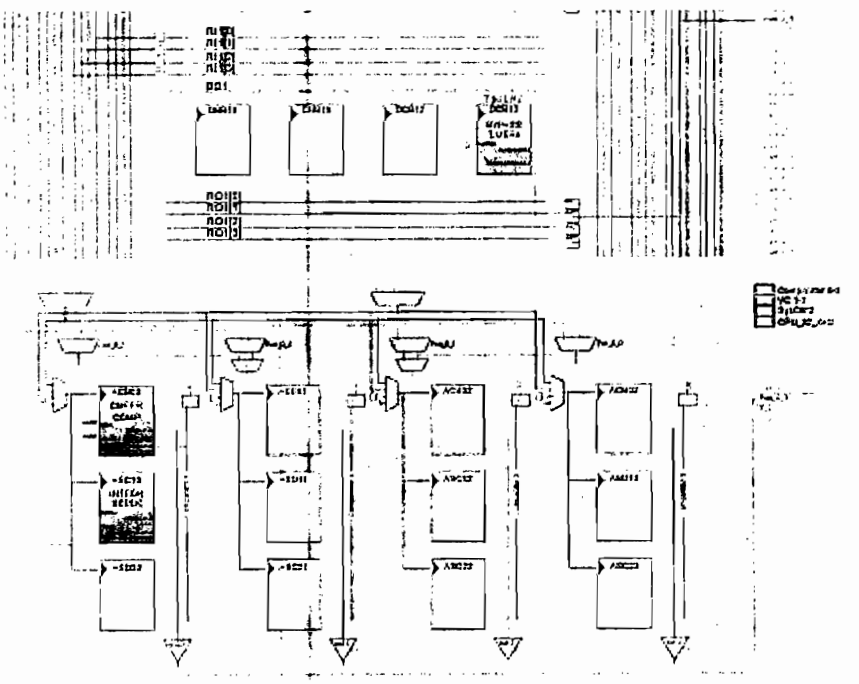
CÓDIGO FUENTE CONVERSOR VOLTAJE – FRECUENCIA PARA UNA SEÑAL ANÁLOGA

```
//-------------------------------------------------------------------------
#include <m8c.h>        // Parte específica, constantes y macros.
#include "PSoCAPI.h"    // Definiciones PSOC API para todos
                        // los Módulos de Usuario.

#include "utils.h"


void main()
{
    CMPPR_COMP_CR0 |= BIT(2); //Conecta el RTopMux a la salida opam
    CMPPR_COMP_CR1 = (CMPPR_COMP_CR1 & 0xC0) | 0x2F;//Seteo de conexiones
                                                    // NMux y PMux
    CMPPR_COMP_CR2 &= BIT(6); //El latch de salida es siempre transparente
    CMPPR_Start(CMPPR_HIGHPOWER); //Elección nivel de potencia para el
                                  //comparador
    AMD_CR0 |= 0x04;
    INTEGR_Start(INTEGR_HIGHPOWER);//Elección nivel de potencia para el
                                   //bloque SC que actuará como integrador

    BYPASS_FUNC_REG = BYPASS_FUNC_REG & 0xFC | 0x02;
    BYPASS_CONTROL_REG |= 0x03; //Para entregar la señal del bus
                                //comparador a un pin externo
    BYPASS_Start();

    while(1);
}
```

Visualización de los módulos de usuario que intervienen en esta aplicación:

Interconexión de los módulos:



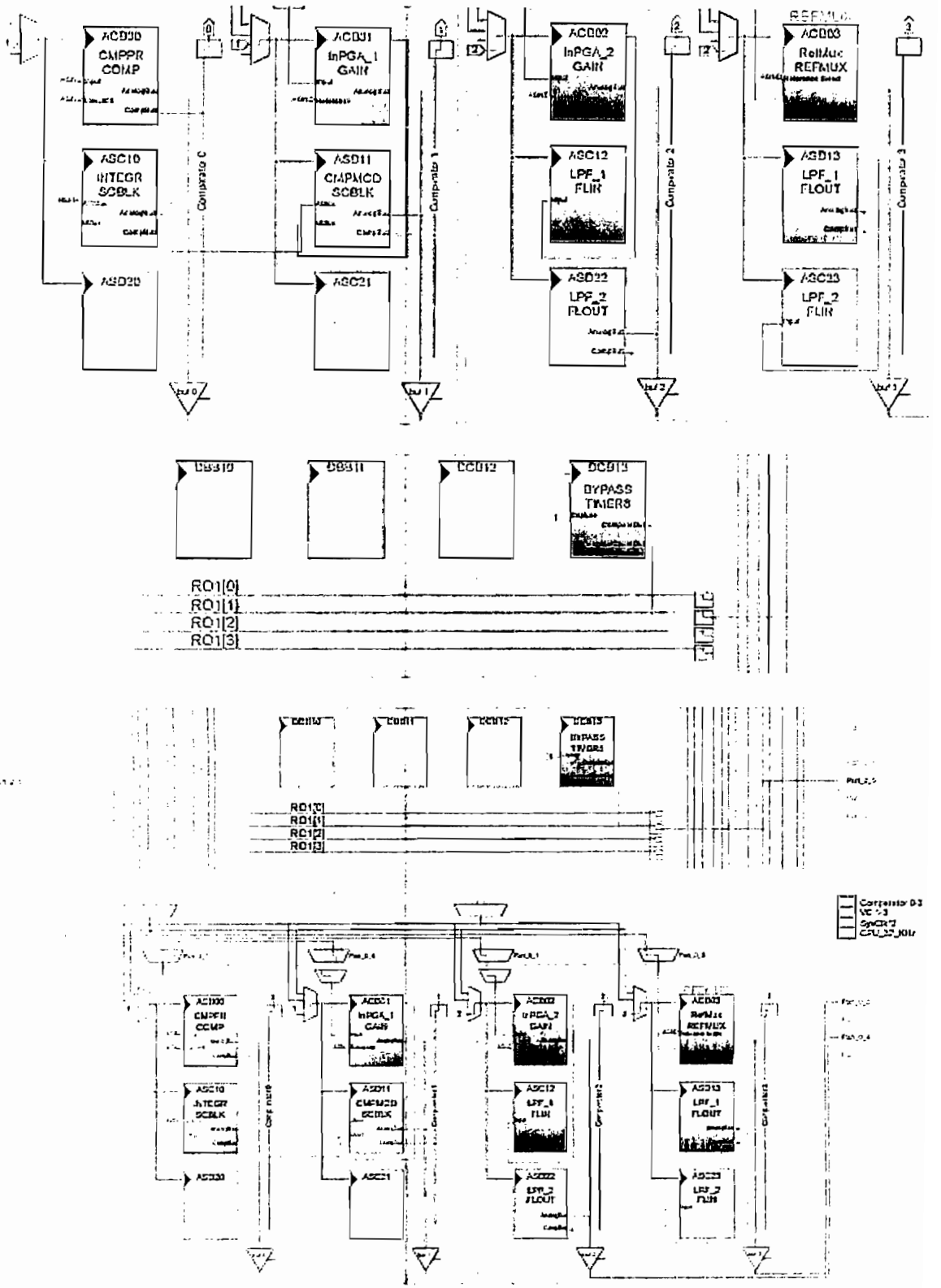


CÓDIGO FUENTE MULTIPLICADOR ANÁLOGO DE DOS SEÑALES

```
//--------------------------------------------------------------------------
#include <m8c.h>        // Parte especifica, constantes y macros.
#include "PSoCAPI.h"    // Definiciones PSOC API para todos
                        // los Módulos de Usuario.

#include "utils.h"

void main()
{
   CMPPR_COMP_CR0 |= BIT(2); //Conecta el RTopMux a la salida opam
   CMPPR_COMP_CR1 = (CMPPR_COMP_CR1 & 0xC0) | 0x2F; //Seteo de conexiones
                                                    // NMux y PMux

   CMPPR_COMP_CR2 &= BIT(6); //El latch de salida es siempre transparente

   CMPPR_Start(CMPPR_HIGHPOWER);//Elección nivel de potencia para el
                                //comparador

   AMD_CR0 |= 0x04 | 0x50; //Habilita la señal de control de modulación
                           //análoga para la columna análoga Comp.0&1
   INTEGR_Start(INTEGR_HIGHPOWER);//Elección nivel de potencia para el
                                  //bloque SC que actuará como integrador

   CMPMOD_Start(CMPMOD_HIGHPOWER);//Elección nivel de potencia para el
                                  //bloque SC que actuará como CMPMOD

   InPGA_1_Start(3);//Para inicialización PGA 1
   InPGA_2_Start(3);//Para inicialización PGA 2

   LPF_1_Start(3);//Para inicialización filtro1

   LPF_2_Start(3);//Para inicialización filtro2

   RefMux_Start(RefMux_HIGHPOWER);//Seteo del nivel de potencia para
                                  //el bloque CT

   BYPASS_FUNC_REG = BYPASS_FUNC_REG & 0xFC | 0x02;
   BYPASS_CONTROL_REG |= 0x03;
   BYPASS_Start(); //Para entregar la señal del bus
                   //comparador a un pin externo
   while(1);
}
```

Visualización de los módulos de usuario que intervienen en esta aplicación:

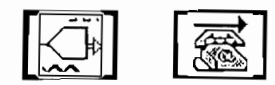
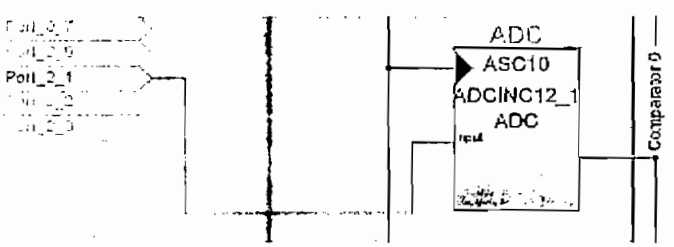
Interconexión de los módulos:



CÓDIGO FUENTE COMUNICACIÓN SERIAL CON UN PC

```
;-----------------------------------------------------------------------
include "m8c.inc"        // Parte especifica, constantes y macros.
include "PSoCAPI.h"      // definiciones PSOC API para todos
                         // los Módulos de Usuario.
export _main ;

_main:
    mov A,ADCINC12_1_HIGHPOWER ; Elección nivel de potencia
                               ; para el conversor.
    call ADCINC12_1_Start ; LLamado rutina para inicialización
                          ; del conversor.
    M8C_EnableGInt ; Habilitación de la interrupción global.
    mov A,0 ; Limpieza del A.
    call ADCINC12_1_GetSamples ; LLamado rutina para muestreo.
    call TX8_1_Start ;Habilitación del módulo TX8 para transmisión.


lazo:
    call ADCINC12_1_ClearFlag ; Reseteo de la bandera de
                              ; los datos disponibles.
    call ADCINC12_1_fIsData ; LLamado rutina para chequeo del estado
                            ; del ADC.
    call ADCINC12_1_iGetData ; Retorno de los datos convertidos.
    call TX8_1_SendData ; LLamado de rutina para inicio de transmisión
                        ; de datos.
ESPERA:
    call bTX8_1_ReadTxStatus; Llamado rutina para recuperar el contenido
                             ; del registro de control.
    and A,TX8_TX_COMPLETE; Máscara para el estado de recepción.
    jz ESPERA

    jmp lazo
```

Visualización de los módulos de usuario que intervienen en esta aplicación:



Interconexión de los módulos: