INTRODUCCIÓN AL PSoC5LP
Teoría y aplicaciones prácticas

Julián Rolando Camargo López
César Andrey Perdomo Charry

ESPACIOS

A mis hermosas hijas Valeria y Valentina, que son mi inspiración para todos mis logros.

Julian Rolando Camargo López

A mi hijo Gabriel, el motor de mi vida. A mis padres por permitirme ser quien soy.

César Andrey Perdomo Charry

Contenido

¿Qué es un PSoC®? **15**
Arquitectura general 16
Bloque de enrutamiento programable y de interconexión 16
Bloques analógicos y digitales configurables 16
Subsistema CPU 17
Familias PSoC® 17
Sistemas de desarrollo para PSoC®'s 19
CY3210-PSoCEval1 19
CY3214-PSoCEvalUSB 19
CY8CKIT-030 PSoC®3 20
CY8CKIT-001 PSoC® 21
CY8CKIT-042 PSoC4 22
CY8CKIT-040 PSoC4000 22
CY8CKIT-038 PSoC4200 23
CY8CKIT-049 PSoC4 23
CY8CKIT-044 PSoC®4 M-SERIES 23
CY8KIT-059 PSoC®5LP 23

Arquitectura de la familia PSoC®5LP **25**
Unidad central de procesamiento (CPU) 26
Registros 27
Registros especiales 28
Modos de operación 28
Nested vectored interrupt controller (NVIC) 29
Mapa de memoria 29
Set de instrucciones 29
Memoria 31

Recursos del sistema 32
Sistema digital 34
Bloques digitales universales 36
Camino de datos 37
Arreglo de bloques universales digitales 37
Sistema digital de interconexión 37
Sistema analógico 38
Nomenclatura utilizada para nombrar los PSoC®s 43
Familia PSoC® CY8C58LP 45
Características generales del CY8C5888LTI-LP097 45
Distribución y función de pines del CY8C5888LTI-LP097 48

Software de programación para el PSoC®5LP 55
Plantilla diseño esquemático 57
Plantilla recursos del sistema 58
Plantilla edición de código C 60
Debugger 61
Creación de un proyecto con PSoC® Creator 62

Kit de desarrollo CY8CKIT-059 para PSoC®5LP 65
Diagrama de bloques del CY8CKIT-059 66
Detalles del hardware del kit 67
Distribución de pines de la tarjeta con el PSoC®5LP 68
Distribución de pines de la tarjeta KitProg 70
Conexión entre la tarjeta KitProg y la tarjeta con el PSoC®5LP T2 71
Tarjeta de periféricos para el kit CY8CKIT-059 T1 72

Puertos de entrada/salida (E/S) 77
Puertos de E/S de propósito general (GPIO) 77
Puertos de E/S especiales (SIO) 79
Puerto de E/S USB (USBIO) 79
Uso de los puertos de E/S con el PSoC Creator 81
Configuración 82
Rutinas utilizadas para el manejo de las líneas de E/S 85
Ejemplos de aplicación de los puertos de E/S 89
Diseño de un decodificador de teclado matricial 4x4 89
Solución 89
Visualización de datos en display de 7 segmentos 96
Solución 96

Ejemplo 5-2: 101
Manejo de tiempo con rutinas por software 102
Solución 102
Ejemplo 5-3: 104
Solución 104
Ejemplo 5-4: 106

Bloque LCD alfanumérica **109**
Descripción general 109
Uso del bloque LCD_Char 110
Configuración del bloque LCD T2 110
Rutinas utilizadas para el manejo del bloque LCD 112
Ejemplos de aplicación del bloque LCD 117
Manejo básico de la LCD 117
Solución 117
Ejemplo 6-1: 118
Decodificador de teclado matricial con visualización en LCD 119
Solución 119
Ejemplo 6-2: 120
Manejo de caracteres personalizados 121
Solución 121
Ejemplo 6-3: 122
Animación simple con caracteres personalizados en la LCD 123
Solución 123
Ejemplo 6-4: 125

Interrupciones en el PSoC5®LP **129**
Bloque de interrupción del PSoC®5LP 129
Configuración del bloque de interrupción 130
Rutinas utilizadas para el manejo del bloque de interrupción 131
Ejemplos de uso del bloque de interrupción 132
Aplicación de una interrupción simple 132
Solución 132
Ejemplo 7-1: 134
Captura de interrupción en un bloque de E/S 135
Solución 135
Ejemplo 7-2: 136
Contador de objetos utilizando un sensor infrarrojo y visualización dinámica en dos displays de 7 segmentos 138

Solución 138
Ejemplo 7-3: 143
Implementación de un contador de objetos en una banda transportadora 144
Solución 145
Ejemplo 7-4: 146

Conversor analógico a digital del PSoC5®LP **149**
ADC por aproximaciones sucesivas 150
Señales de entrada/salida del bloque ADC por aproximaciones sucesivas 151
Configuración del bloque ADC por aproximaciones sucesivas 153
Rutinas utilizadas para el manejo del bloque ADC_SAR 156
ADC delta sigma 158
Señales de entrada/salida del ADC delta sigma T2 159
Configuración del bloque ADC delta sigma 160
Rutinas utilizadas para el manejo del bloque ADC_Del_Sig 165
Ejemplos de uso del bloque ADC 167
Voltímetro digital con visualización en LCD 167
Solución 167
Ejemplo 8-1: 169
Termómetro digital 170
Solución 170
Ejemplo 8-2: 172
Temporizador/contador del PSoC5®LP 174
Descripción general 174
Uso del bloque Timer 175
Señales de entrada/salida del Timer 175
Configuración del bloque Timer 177
Rutinas utilizadas para el manejo del bloque Timer 181
Ejemplos de uso del bloque Timer 183
Operación básica del bloque Timer 183
Solución 183
Ejemplo 9-1: 185
Reloj digital con visualización en LCD 186
Solución 186
Eejemplo 9-2: 188

PWM del PSoC5®LP **193**
Características y uso del bloque PWM 194

.....138
.....143
.....144
.....145
.....146
.....149
.....150
.....151
.....153
.....156
.....158
.....159
.....160
.....165
.....167
.....167
.....167
.....169
.....170
.....170
.....172
.....174
.....174
.....175
.....175
.....177
.....181
.....183
.....183
.....183
.....185
.....186
.....186
.....188
.....193
.....194

Señales de entrada/salida del PWM T2 195
Configuración del bloque PWM 196
Rutinas utilizadas para el manejo del bloque PWM 202
Ejemplo de uso del bloque PWM 203
Dimmer digital controlado por PWM 203
Solución 204
Ejemplo 10-1: 205

Conversor digital a analógico del PSoC5®LP 207
Características y uso del bloque DAC 207
Señales de entrada/salida del DAC de voltaje 208
Configuración del bloque DAC de voltaje 209
Rutinas utilizadas para el manejo del bloque DAC de voltaje 211
Ejemplo de uso del bloque DAC 212
Generador de señales con DAC de voltaje 212
Solución 212
Ejemplo 11-1: 213

Módulo UART del PSoC5®LP 217
Características y uso del bloque UART T1 218
Señales de entrada/salida del bloque UART 218
Configuración del bloque UART 221
Rutinas utilizadas para el manejo del bloque UART 224
Ejemplos De Uso Del Bloque UART 225
Operación básica del bloque UART 225
Solución 226
Nota: 227
Ejemplo 12-1: 227
Voltímetro digital serial con visualización en PC 229
Solución 229
Ejemplo 12-2: 230

Algunos módulos analógicos del PSoC5®LP 233
Bloque comparador analógico 233
Señales de entrada/salida del bloque Comparador 234
Configuración del bloque comparador 235
Rutinas utilizadas para el manejo del bloque comparador 236
Bloque amplificador operacional 237
Señales de entrada/salida del bloque amplificador operacional 238

Configuración del bloque amplificador operacional 239
Rutinas utilizadas para el manejo del bloque amplificador operacional 239
Bloque amplificador de ganancia programable (PGA) 240
Señales de entrada/salida del bloque PGA 241
Configuración del bloque PGA 241
Rutinas utilizadas para el manejo del bloque PGA 242
Ejemplos de uso de los bloques analógicos 243
Operación básica del bloque comparador analógico 243
Solución 244
Ejemplo 13-1: 246
Operación básica del bloque amplificador operacional T2 247
Solución 247
Ejemplo 13-2: 247
Operación básica del bloque amplificador de ganancia programable 249
Solución 249
Ejemplo 13-3: 250

Referencias **251**

¿Qué e

El PSoC® (
pacidades ir
Cypress Sen
control de s
digitales (UI
desarrollo d
asignar cual
una gran ve

Los PSoC e
do, ofrecien
integrated c
PSoC integr
crocontrola
revisión de l
solo PSoC p
tiempo de c
calidad del

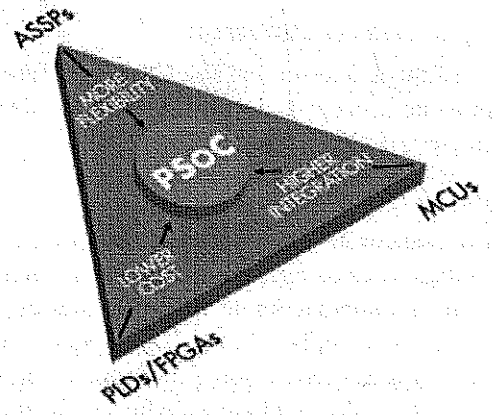
.....239
.....239
.....240
.....241
.....241
.....242
.....243
.....243
.....244
.....246
.....247
.....247
.....247
.....249
.....249
.....250
.....251

¿Qué es un PSoC®?

El PSoC® (Programmable System-on-Chip) es un sistema que ofrece novedosas capacidades integradas en un solo chip, fabricados por la compañía norteamericana Cypress Semiconductor, con un moderno método de adquisición, procesamiento y control de señales y una excelente precisión. Incluye además un arreglo de bloques digitales (UDB) y analógicos que hacen del PSoC® una muy buena propuesta para el desarrollo de proyectos de ingeniería, lo que proporciona al sistema la capacidad de asignar cualquier función, a cualquier terminal del circuito integrado, lo que confiere una gran versatilidad.

Los PSoC emplean una arquitectura configurable para el control del diseño embebido, ofreciendo un equivalente a un FP-ASIC (field programable application-specific integrated circuit) con la ventaja del tiempo de implementación. Los dispositivos PSoC integran circuitos digitales y analógicos configurables, controlados por un microcontrolador interno, de modo que proveen tanto una capacidad mejorada para la revisión de los diseños como la disminución del número de componentes usados. Un solo PSoC puede integrar hasta 100 funciones periféricas, y ahorrar de esta manera tiempo de diseño, espacio físico y consumo de energía mientras que se mejora la calidad del sistema y se reduce su costo.

Figura 1. El PSoC® integra lo mejor de varios mundos

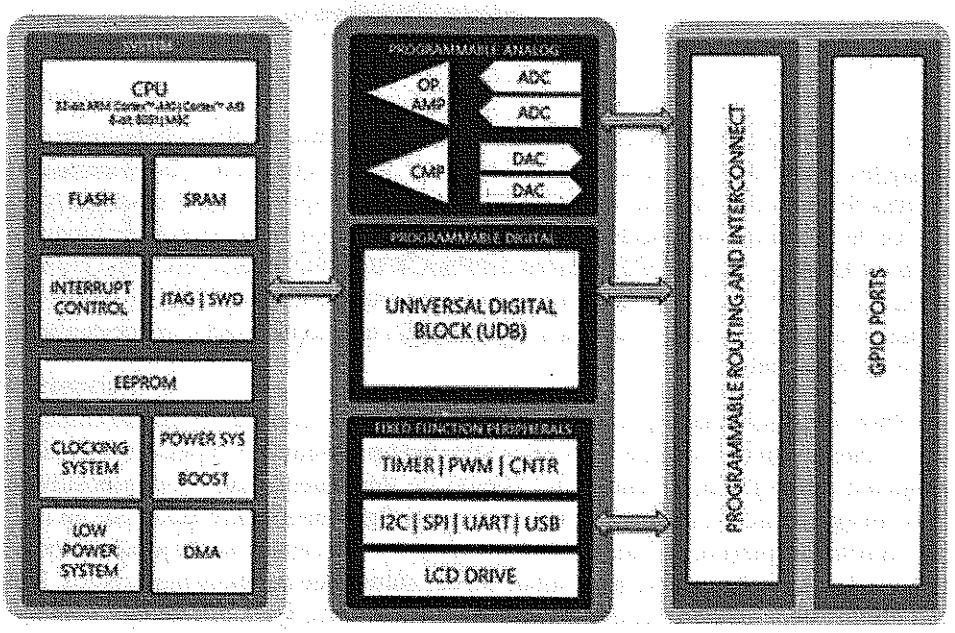


Fuente: [1]

Arquitectura general

La arquitectura interna de un PSoC® se puede dividir en tres grandes bloques, como se muestra en la figura 2: bloques analógicos y digitales configurables, una CPU y un sistema de enrutamiento programable y de interconexión. El PSoC® permite utilizar rutinas predefinidas y probadas de la biblioteca PSoC® de funciones dada por el fabricante, o permite al usuario generar su código propio.

Figura 2. Diagrama de bloques de la arquitectura típica de un PSoC®



Fuente: [1]

La descripción general de cada uno de estos bloques se presenta a continuación.

Bloque de enrutamiento programable y de interconexión

Permite conectar los periféricos internos del PSoC® con el exterior, lo que hace dinámico el uso de los pines de E/S del dispositivo. Esto quiere decir que un pin GPIO puede ser utilizado por cualquier periférico del PSoC.

Bloques analógicos y digitales configurables

La unión de circuitos analógicos y digitales configurables es la base de la plataforma PSoC®. Se pueden configurar estos bloques con funciones predefinidas o con funciones propias. Mediante la combinación de varios bloques digitales. Asimismo, se pueden crear recursos lógicos de 16,24 y 32 bits. Los bloques analógicos están compuestos por un condensador interruptor (switch capacitor), amplificador operacional (op-amp), comparador, ADC, DAC, y bloques de filtros digitales, lo que permite el uso de señales analógicas complejas.

Subsistema CPU

El PSoC® ofrece diversas opciones de CPU (M8C y 8051 de 8 bits, Cortex-M0 y Cortex-M3 de 32 bits) con SRAM, EEPROM y Memoria Flash, y una variedad de recursos del sistema esenciales, incluyendo:

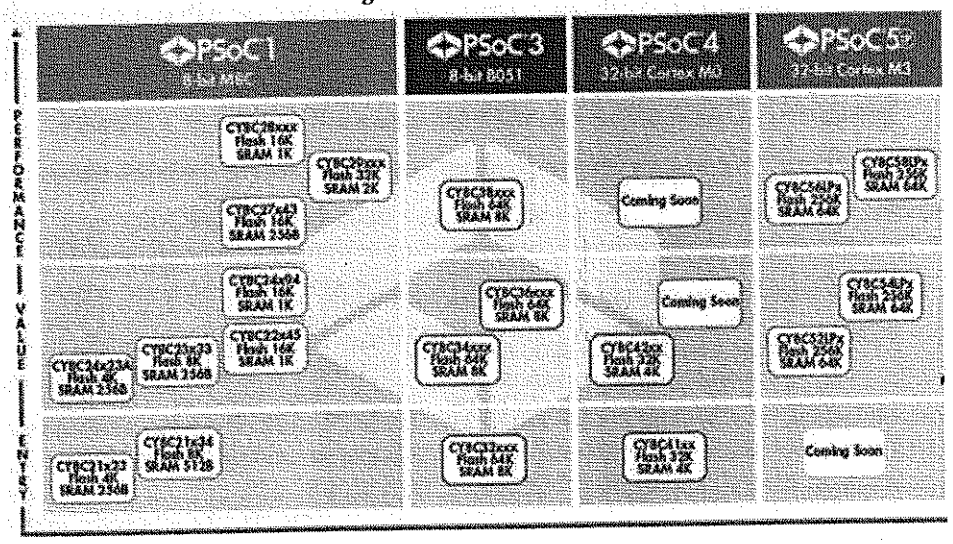
- Oscilador principal y de baja velocidad interna.
- Conectividad con oscilador externo o cristal externo de alta precisión, sincronización, programable.
- Modos de bajo consumo de energía y watchdog timer.
- Fuentes de reloj múltiples que incluyen un PLL.

Los PSoC® también incluyen las interfaces de comunicación I2C, Full-Speed USB 2.0, CAN 2.0, driver para manejo directo de LCD y capacidades de depuración en chip usando JTAG y depuración en serie.

Familias PSoC®

Entre los PSoC®´s ofrecidos por Cypress, se diferencian actualmente cuatro grandes familias: PSoC®1, PSoC®3, PSoC®4 y PSoC®5LP. Cada una de ellas incluye diversas opciones de acuerdo con los requerimientos del usuario, ya sea sencillez, costo o rendimiento. Un resumen de cada una de las familias puede verse en la figura 3.

Figura 3. Familias PSoC®



Fuente: [1]

En la tabla 1 se presenta un resumen con las características más relevantes de las diversas familias PSoC® existentes en la actualidad.

	PSoC®1	PSoC®3	PSoC®4	PSoC®5LP
CPU	M8U de Cypress optimizada de 8 bits	8051 de 8 bits de ciclo sencillo de rendimiento optimizado	ARM Cortex-M0 de 32 bits de alto rendimiento	ARM Cortex-M3 de 32 bits de alto rendimiento
Oscilador	Máximo 24MHz, 4 MIPS	Máximo 67MHz, 33 MIPS	De 24MHz a 48MHz	Máximo 67MHz, 84 MIPS
Memoria	Flash de 4KB a 32KB SRAM de 256B a 2KB	Flash de 8KB a 64KB SRAM de 2KB a 8KB	Flash de 32KB con acelerador de lectura SRAM de 4 KB a 32KB	Flash de 32KB a 256KB SRAM de 16KB a 64KB
Alimentación	Operación desde 1.7V hasta 2.25V	Operación desde 0.5V a 5.5V	Operación desde 1.71 a 5.5V	Operación desde 2.7 a 5.5V
Consumo	Activo: 2mA, Pausado: 3µA	Activo: 1.2mA, Pausado: 1µA, Hibernado: 200nA	Activo: 2mA, Pausado: 2µA, Hibernado: 20nA	Activo: 2mA, Pausado: 2µA, Hibernado: 300nA
ADC	1 ADC Delta-Sigma (6 a 14 bits) 131 ksps @ 8 bits Precisión de voltaje de ±1.53%	1 ADC Delta-Sigma (12 a 20 bits) 192 ksps @ 12 bits Precisión de voltaje de ±0.1%	1 ADC SAR (12 bits) 192 ksps @ 12 bits; 1 Msps @ 12 bits	1 ADC Delta-Sigma (12 a 20 bits) 2 ADCs SAR (12 bits) 192 ksps @ 12 bits; 1 Msps @ 12 bits Precisión de voltaje de ±1.0%
DAC	2 DACs (6 a 8 bits)	4 DACs (8 bits)	4 DACs (8 bits)	4 DACs (8 bits)

	PSoC®1	PSoC®3	PSoC®4	PSoC®5LP
Comunicación	FS USB 2.0, I2C, SPI, UART	FS USB 2.0, I2C, SPI, UART, CAN, LIN, I2S	FS USB 2.0, I2C, SPI, UART, LIN, I2S	FS USB 2.0, I2C, SPI, UART, LIN, I2S
Sistema de programación	Requiere cubo ICE y Flex-Pods	JTAG on-chip, depura y simula; SWD, SWV	JTAG on-chip, depura y simula; bootloader, SWD, SWV	JTAG on-chip, depura y simula; SWD, SWV
Pines de E/S	Hasta 64	Hasta 72	Hasta 36	Hasta 72

Tabla 1. Características más relevantes de las diversas familias PSoC®
Fuente: elaboración propia

Sistemas de desarrollo para PSoC®'s

Cypress fabrica diversos sistemas de desarrollo y tarjetas de evaluación de bajo costo para cada una de las familias de PSoC® existentes en el mercado, con diversos periféricos y funcionalidades. Entre estos se pueden mencionar los siguientes:

- Para PSoC®1 se tienen dos sistemas de desarrollo y dos kits de evaluación.
- Para PSoC®3 se tienen dos sistemas de desarrollo y nueve kits de expansión.
- Para PSoC®4 se tienen cuatro sistemas de desarrollo compatible con diversas tarjetas Arduino disponibles en el mercado, más dos kits de evaluación.
- Para PSoC®5 se tiene un sistema de desarrollo que soporta además PSoC®1 y PSoC®3 y un kit de evaluación.

CY3210-PSoCEval1

Sistema de desarrollo para PSoC®1 (ver figura 4) que incluye una unidad de programación MiniProg, una tarjeta de desarrollo con LCD, un potenciómetro y LED, entre otros, un chip PSoC®1 de referencia CY8C29466-24PXI y otro de referencia CY8C27443-24PXI. (U$40).

CY3214-PSoCEvalUSB

En la figura 5 puede verse otro sistema de desarrollo para PSoC®1 con capsense (sensor capacitivo), una tarjeta de desarrollo con LCD, un potenciómetro y varios LED, entre otros, que incluye además una unidad de programación MiniProg y un chip PSoC®1 de referencia CY8C24794-24LFXI (U$40).

Figura 4. Kit CY3210-PSoCEval1

Fuente: [1]

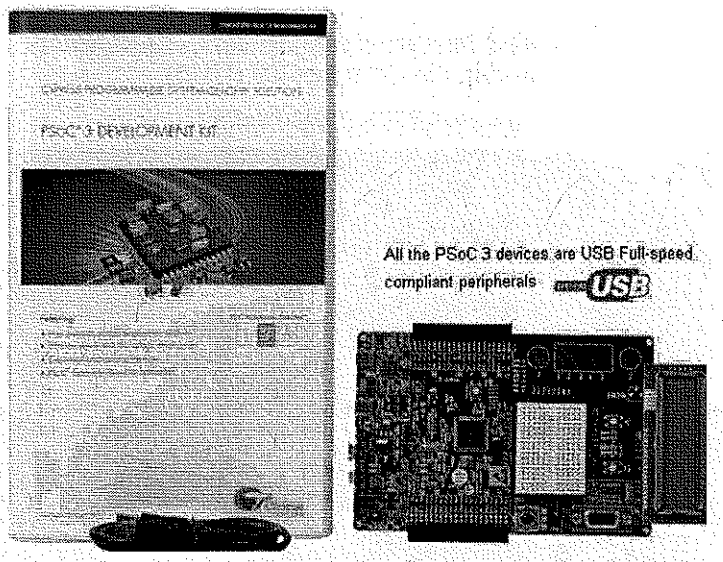
Figura 5. Kit CY3214-PSoCEvalUSB

Fuente: [1]

CY8CKIT-030 PSoC®3

Sistema de desarrollo para PSoC®3 que incluye internamente la unidad de programación del PSoC®, con conversor USB-Serial, LCD, potenciómetro y LED, entre otros, y un chip de la familia PSoC®3 de referencia CY8C3866AXI-040 soldado a la tarjeta de desarrollo (U$99) (ver figura 6).

Figura 6. CY8CKIT-030 PSoC®3

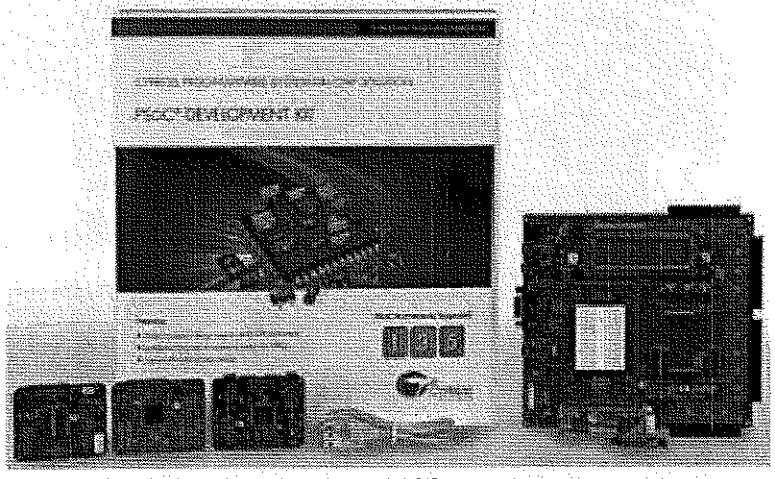


Fuente: [1]

CY8CKIT-001 PSoC®

Sistema de desarrollo para PSoCs 1, 3, 4 y 5 (ver figura 7), con sensor capacitivo que incluye una unidad de programación MiniProg3, LCD, potenciómetro, pulsadores y varios LED, entre otros, un procesador PSoC®1 de la familia CY8C28, PSoC®3 de la familia CY8C38 y PSoC®5LP de la familia CY8C58LP (versión anterior CY8C55), adaptador de corriente (U$250). Para el PSoC®4 es necesario adquirir de manera adicional el kit CY8CKIT-038 PSoC4200.

Figura 7. CY8CKIT-001 PSoC®

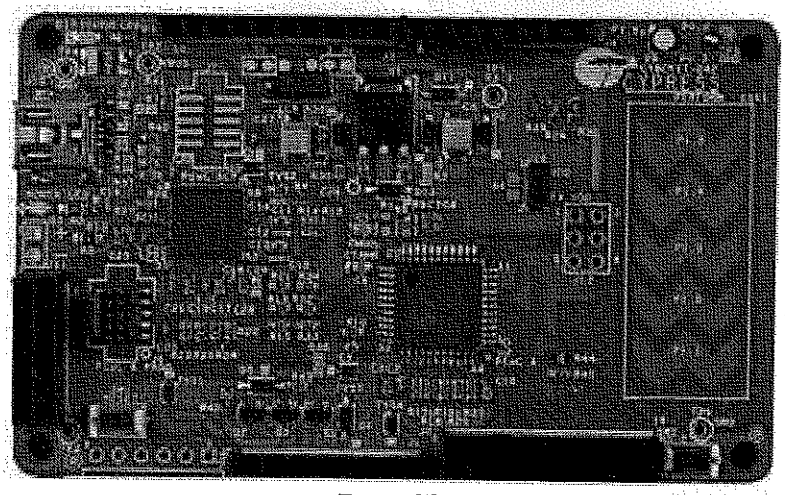


Fuente: [1]

CY8CKIT-042 PSoC4

Sistema de desarrollo para PSoC®4 denominado Pioneer Kit (ver figura 8), que incluye sensor capacitivo, compatible con tarjetas Arduino, con unidad de programación del PSoC® interna y un chip PSoC® serie 4200 (U$25).

Figura 8. CY8CKIT-042 PSoC®4

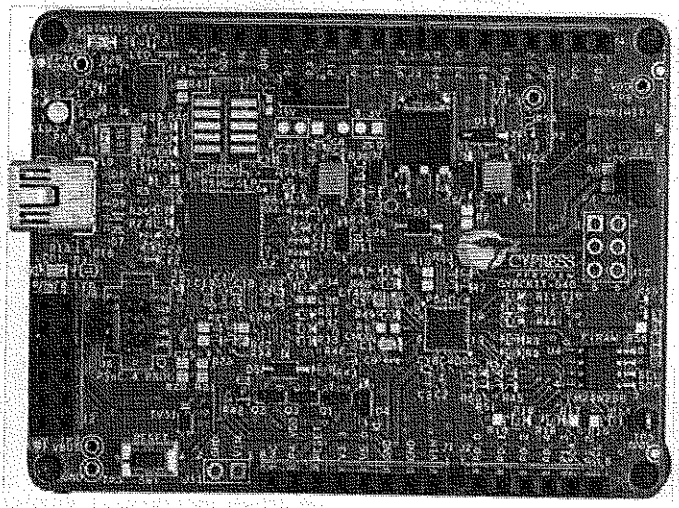


Fuente: [1]

CY8CKIT-040 PSoC4000

Pionner Development Kit (figura 9). Sistema de desarrollo para PSoC®4 con sensor capacitivo externo, incluye internamente la unidad de programación del PSoC® y un procesador de la familia PSoC®4 de referencia CY8C4000 (U$30).

Figura 9. CY8CKIT-040 PSoC4000

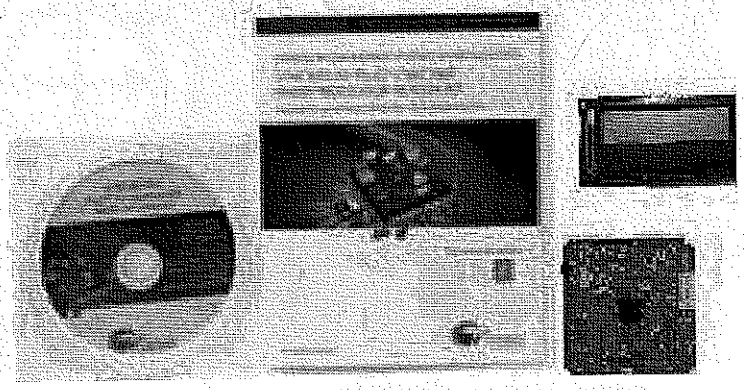


Fuente: [1]

CY8CKIT-038 PSoC4200

Pioneer Kit (figura 10). Sistema de desarrollo para PSoC®4 para ser utilizado con el kit CY8CKIT-001; incluye un chip PSoC® CY8C4245AXI-483 y una LCD I2C (U$39).

Figura 10. CY8CKIT-038 PSoC4200

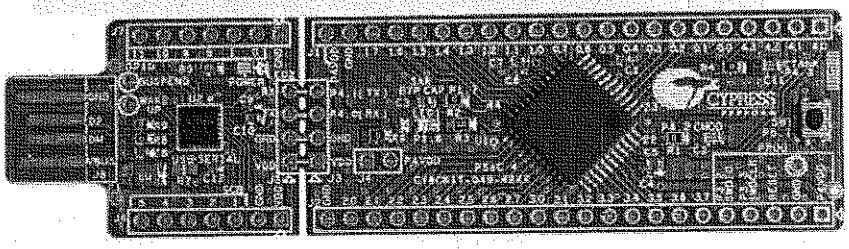


Fuente: [1]

CY8CKIT-049 PSoC4

Prototyping Kit. Sistema de desarrollo de muy bajo costo para la familia PSoC®4 (ver figura 11). Este es un sistema de programación por bootloader y conexión por puerto USB, usando el conversor Cypress USB-Serial de la familia CY7C6521x; incluye un procesador PSoC4100 o PSoC4200 (U$4).

Figura 11. CY8CKIT-049 PSoC®4



Fuente: [1]

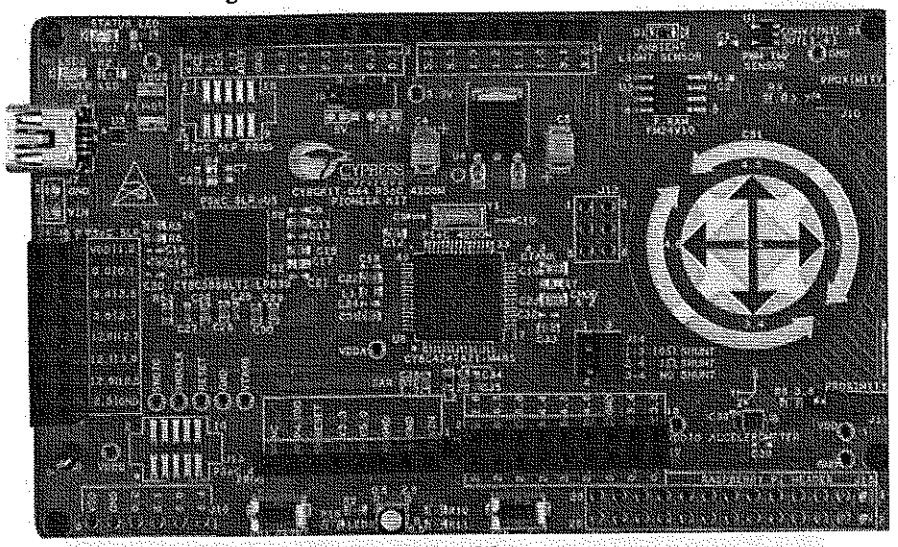
CY8CKIT-044 PSoC®4 M-SERIES

Pioneer Kit para la serie M de la familia PSoC®4, incluye sensor capacitivo con un nuevo diseño denominado CapSense Gesture Pad, un PSoC®4 de la familia 4200M, sensor de luz, LED RGB, acelerómetro de tres ejes y sensor de temperatura (U$25).

CY8KIT-059 PSoC®5LP

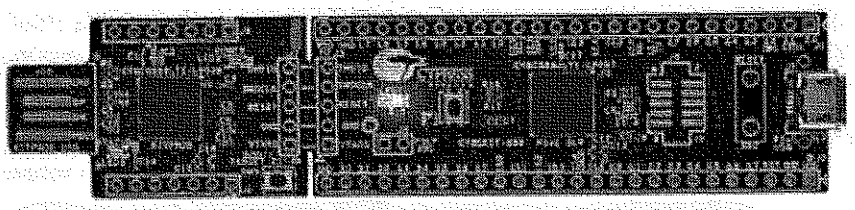
Prototyping Kit. Sistema de desarrollo de bajo costo para la familia PSoC®5LP (ver figura 13), programador y sistema de debugger incluido tipo snap-away, conversor USB-Serial, un procesador CY8C5888LTI-LP097 de 68 pines (U$10).

Figura 12. CY8CKIT-044 PSoC®4 M-SERIES



Fuente: [1]

Figura 13. CY8KIT-059 PSoC®5LP

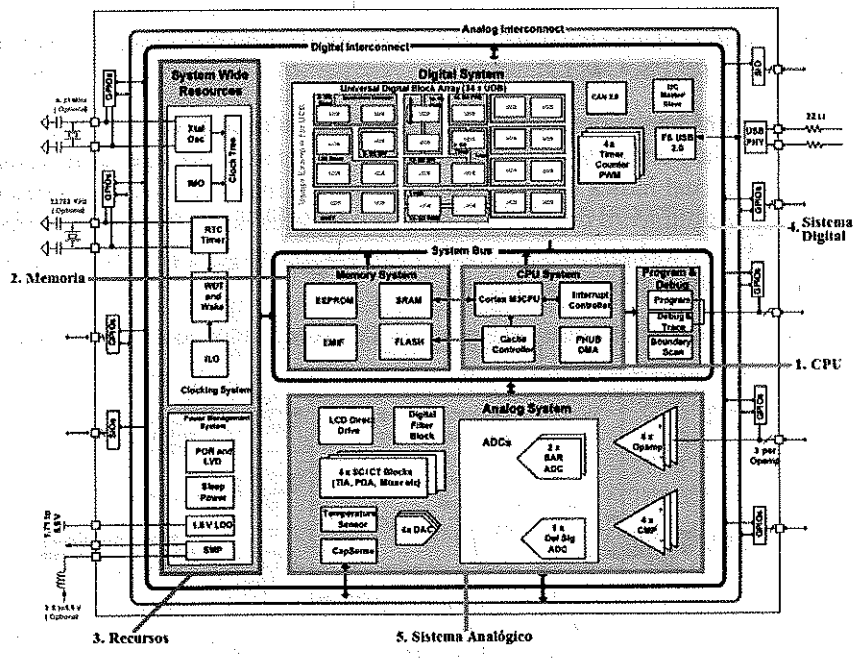


Fuente: [1]

Arquitectura de la familia PSoC®5LP

En este capítulo se trataran de manera general los diferentes subsistemas que componen al sistema en chip PSoC®5LP que se muestra esquemáticamente en la figura 14, explicando sus componentes más importantes.

Figura 14. Esquema general del PSoC®5LP



Fuente: [2, p. 24]

Los subsistemas que componen el PSoC®5LP son los siguientes:

1. Unidad central de procesamiento (CPU).
2. Memoria.
3. Recursos del sistema.
4. Sistema digital.
5. Sistema analógico.

Unidad central de procesamiento (CPU)

El PSoC®5LP viene dotado de un procesador ARM Cortex-M3, el cual tiene unas propiedades diseñadas específicamente para microcontroladores que lo hacen muy eficiente para las diversas exigencias en las que se vea comprometido.

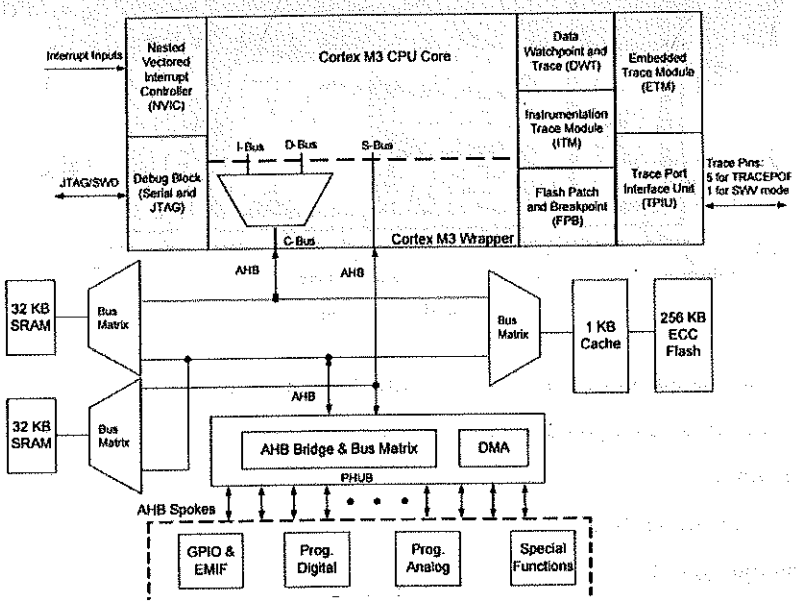
La familia de procesadores ARM Cortex establece una arquitectura estándar dirigida a un amplio espectro de tecnologías. Esta familia se basa en la arquitectura ARMv-7 que tiene tres distintos perfiles dirigidos a aplicaciones específicas [3]:

- Perfil A: diseñado para plataformas de aplicación abierta de alto rendimiento.
- Perfil R: diseñado para sistemas embebidos de alta gama, en las que es necesario el rendimiento en tiempo real.
- Perfil M: diseñado para los sistemas tipo microcontrolador profundamente embebidos.

Este último perfil es el más relevante para el tema tratado en este libro, por ser el que tiene incorporado el PSoC®5LP.

El ARM Cortex-M3 es un procesador de 32 bits, tanto la ruta de acceso como el banco de registros y la interfaz de memoria son de 32 bits, es un procesador con arquitectura Harvard, lo cual indica que los buses de instrucciones y datos están separados y aumentan el rendimiento del procesador. Las interfaces de los buses aunque están separadas comparten el mismo espacio de memoria que es de 4 GB. En la figura 15 se observa el esquema general del procesador.

Figura 15. Diagrama de bloques del procesador Cortex-M3 utilizado en el PSoC®5LP



Fuente: [2, p. 38]

Registros

Los registros del procesador Cortex M3 son 16, numerados desde R0 a R15, con tres registros de asignación fija, R13 (Stack Pointer), R14 (Link Register) y R15 (Program Counter).

Los registros de R0 a R12 son registros generales multipropósito de 32 bits para operaciones con datos, de los cuales los registros R0 a R7 son considerados registros secundarios o (low), ya que ciertas instrucciones del set de instrucciones 16-bit Thumb® únicamente pueden ser ejecutadas por estos registros, los registros R8 a R12 son los registros primarios (high), puesto que manejan todas las demás instrucciones [4].

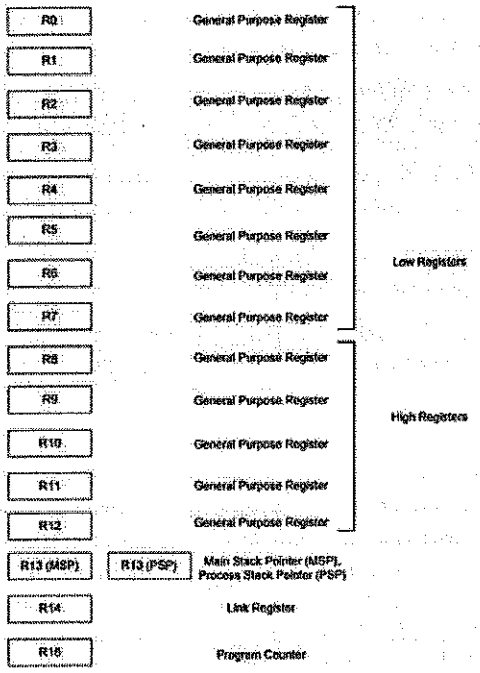
Registro R13 SP (Stack Pointer): apuntador de pila que a su vez se subdivide en dos (Main Stack Pointer) y (Process Stack Pointer), el primero es el predeterminado, el segundo es usado durante la ejecución del código.

Registro R14 (Link Register): se usa en las subrutinas y su función es almacenar la dirección de retorno.

Registro R15 PC (Program Counter): almacena la dirección de secuencia durante la ejecución del programa.

En la figura 16 se puede observar el mapa de registros del procesador Cortex M3.

Figura 16. Registros de un procesador Cortex-M3

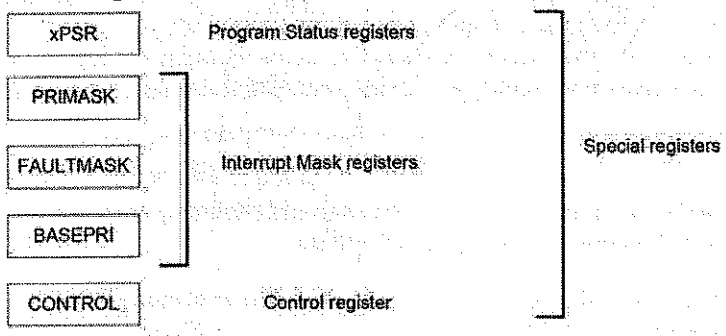


Fuente: [2, p. 40]

Registros especiales

El procesador Cortex M3 también dispone de registros de asignación especial a los cuales se puede acceder únicamente con instrucciones específicas del set de instrucciones. Los registros especiales se pueden observar en la figura 17 y se describen a continuación.

Figura 17. Registros especiales procesador Cortex-M3



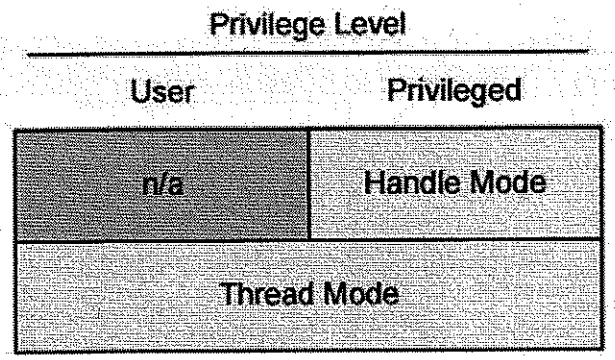
Fuente: [2, p. 41]

- Registro xPRS (Program Status Registers): tiene como objetivo dar soporte a las operaciones aritméticas almacenando las banderas de procesos, como por ejemplo resultado cero (zero flag) o bandera de arrastre (carry flag), también el estado de la ejecución y el número de la interrupción.
- Registro PRIMASK (Interrupt Mask Registers): deshabilita todas las interrupciones excepto las interrupciones no enmascaradas (NMI) y Hard Fault.
- Registro FAULTMASK (Interrupt Mask Registers): deshabilita todas las interrupciones excepto las interrupciones no enmascaradas (NMI).
- Registro BASEPRI (Interrupt Mask Registers): deshabilita todas las interrupciones de un nivel especifico de prioridad.
- Registro CONTROL (Control Register): define el nivel de privilegio en el modo de operación y la selección del puntero de la pila (Stack Pointer).

Modos de operación

El procesador tiene dos modos de operación y dos niveles de privilegio. Los modos de operación se denominan Thread y Handler y los niveles de privilegio se denominan Privileged y User. Dependiendo del modo de operación se puede determinar si el procesador ejecuta un programa normal o un manejo de interrupciones, y con los niveles de privilegio se puede determinar qué accesos se tiene a las áreas sensibles de la memoria o áreas críticas. A continuación se presentan los modos de operación (figura 8).

Figura 18. Modos de Operación



Fuente: [2, p. 42]

Nested vectored interrupt controller (NVIC)

El procesador Cortex-M3 incluye un controlador de interrupciones llamado Nested Vectored Interrup Controller (NVIC) cuyas características principales son:

- Soporte de interrupciones anidadas y vectorizadas.
- Soporte del cambio dinámico de prioridades.
- Reducción de latencia de las interrupciones.
- Enmascaramiento de la interrupción.

Mapa de memoria

El mapa de memoria del procesador Cortex-M3 está predefinido y tiene una longitud de 4 GB dividido en regiones; esto permite que los controladores de interrupción y los componentes de depuración sean accedidos con simples instrucciones de acceso de memoria (ver figura 19).

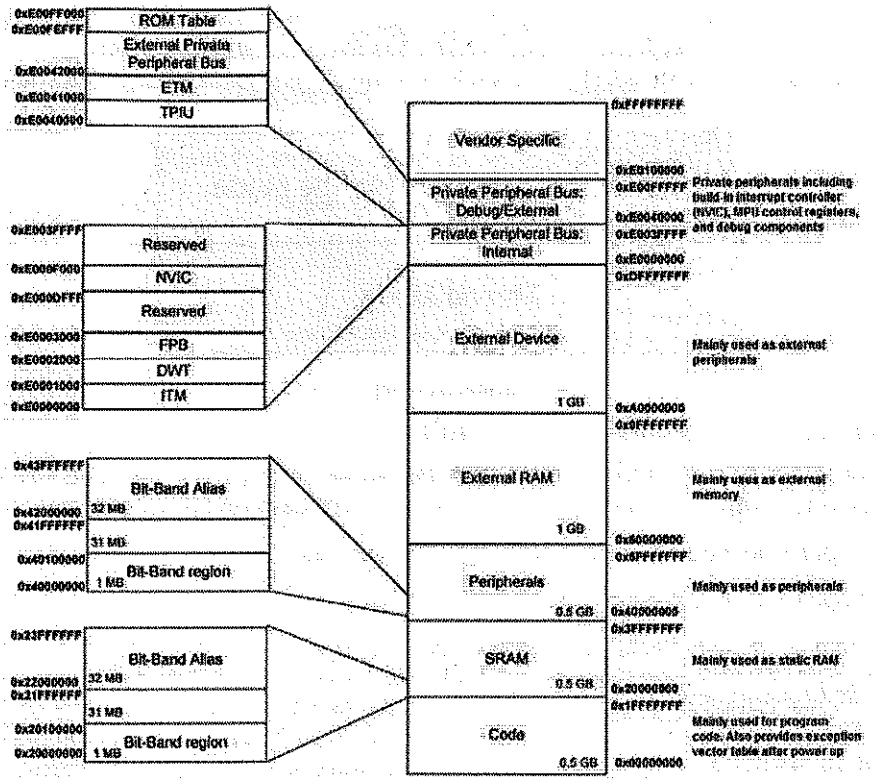
Set de instrucciones

El Cortex M3 soporta el set de instrucciones Thumb® - 2; esta es una de las características más importantes del procesador, puesto que permite la ejecución de instrucciones tanto de 16 bits como de 32 bits que se pueden usar juntas para lograr una alta eficiencia en la ejecución del código [5].

Para obtener lo mejor de cada set de instrucciones, muchas aplicaciones se pueden desarrollar mezclando los códigos en ARM 32 bits y Thumb 16 bits. Estos arreglos pueden lograr una eficiencia mayor, pero si no son bien estructurados los cambios de set de instrucciones pueden causar deficiencias y retardos en la ejecución del programa.

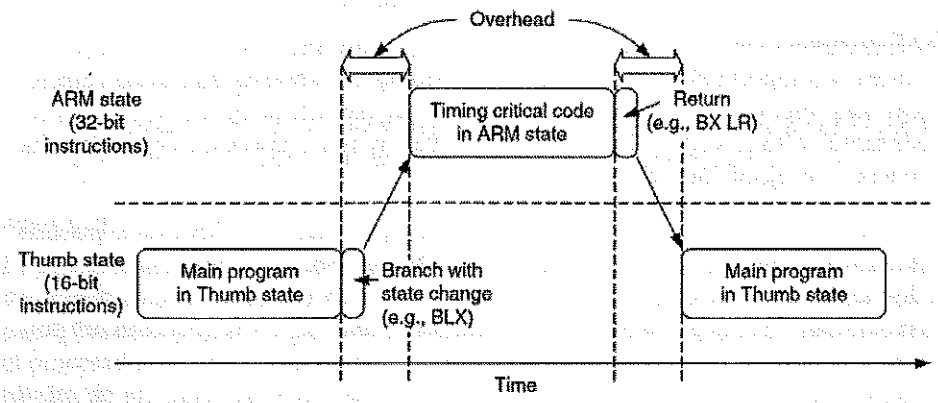
En la figura 20 se muestra el cambio de set de instrucciones en la ejecución de un programa (para mayor información y detalle se sugiere la consulta del manual de referencia del Cortex-M3).

Figura 19. Mapa de memoria del procesador Cortex_M3



Fuente: [2, p. 46]

Figura 20. Cambio entre set de instrucciones en procesadores Cortex-M3



Fuente: [4, p. 19]

Por último, se presenta en la tabla 2 un resumen con las características más importantes del procesador Cortex-M3.

Características	Cortex – M3
Arquitectura	ARMv7-M (Harvard)
Set de instrucciones	Thumb/Thumb – 2
Pipeline	3 Stage + Branch Speculation
Interrupciones	NMI + 1 a 240 Interrupciones Físicas
Latencia de interrupción	12 Ciclos
Sleep Modes	Integrado
Protección de memoria	8 Region Memory Protection Unit
Dhyrestone	1,25 DMIPS/MHz
Consumo de potencia	0,19mW/MHz
Área	0,86mm2

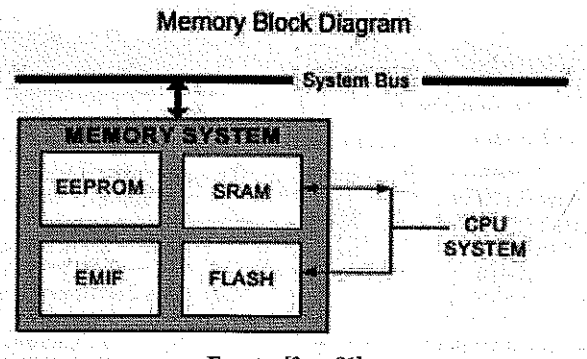
Tabla 2. Características más relevantes de los procesadores Cortex-M3
Fuente: elaboración propia.

Memoria

El sistema de memoria del PSoC®5LP se muestra en la figura 21, y tiene los siguientes recursos:

- Memoria estática RAM (SRAM): es usada para almacenar información de manera temporal. El código puede ser ejecutado desde la memoria estática, el sistema tiene 64 KB de memoria organizada en 16 bloques de 4KB accesibles por el procesador o controlador de acceso directo a memoria (DMA).
- Memoria programable flash: en el PSoC® la memoria flash es un espacio no volátil donde se almacena el firmware, la memoria consiste en 256 KB de memoria libre que el usuario puede manejar.
- Memoria EEPROM: El PSoC® incorpora una memoria EEPROM, la cual es una memoria de tipo no volátil para uso del usuario y es de un tamaño de 2KB.
- Memoria externa: el PSoC® contiene una interfaz denominada EMIF (External Memory Interface) que permite la conexión de memorias externas al PSoC®. Esta interfaz permite leer y escribir datos en la memoria externa y soporta cuatro tipos de memoria: SRAM síncrona y asíncrona, cellular RAM y NOR Flash que pueden ser de 8 a 16 bits de ancho.

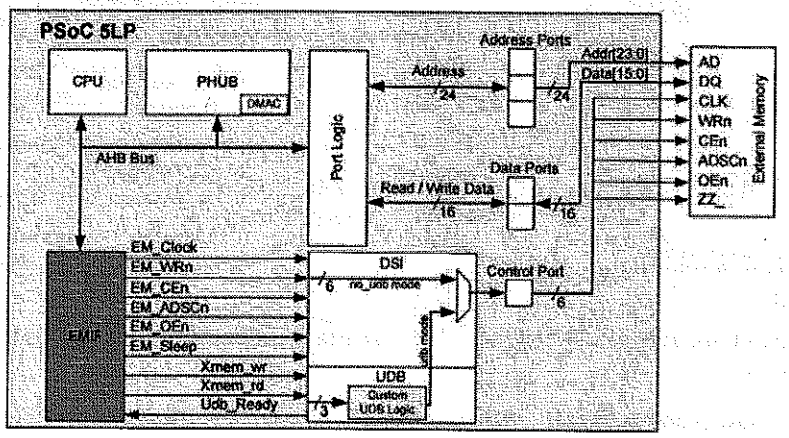
Figura 21. Sistema de memoria del PSoC®5LP



Fuente: [2, p. 81].

En la figura 22 se muestra la interfaz para memorias externas EMIF.

Figura 22. Diagrama de bloques de la interfaz EMIF



Fuente: [2, p. 95].

Recursos del sistema

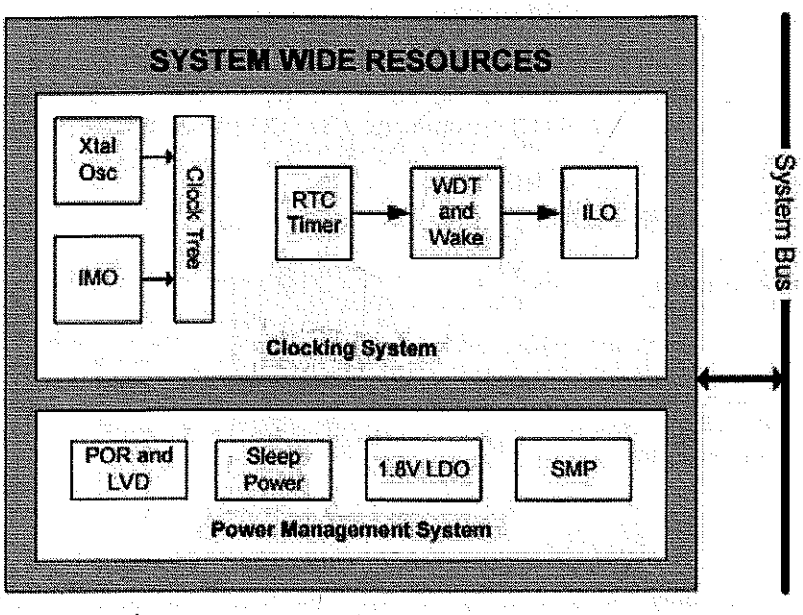
Los recursos del sistema se dividen en dos: un sistema de relojes y un sistema de manejo de la alimentación. Estos sistemas proveen al PSoC®5LP de dos de los recursos más importantes y esenciales para correcto funcionamiento de cualquier microcontrolador, la señal de reloj que genera los ciclos de reloj interno y la estabilidad en la alimentación (ver figura 23).

El sistema de reloj gestiona toda la base de tiempos para la ejecución de las tareas dentro del PSoC®5LP, el usuario tiene a disposición diferentes frecuencias de reloj (ver figura 24).

Resumiendo lo anterior, puede decirse que la familia PSoC®5LP dispone de tres tipos de reloj internos:

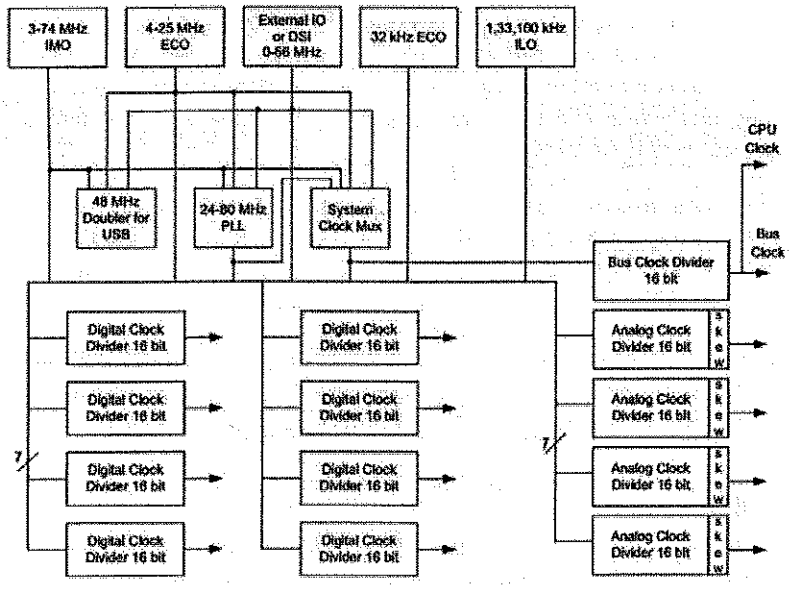
- Oscilador principal interno (IMO), opera entre 3MHz y 74MHz con una tolerancia del ±1% a 3MHz y del 7% a ±74MHz.

Figura 23. Recursos del sistema



Fuente: [2, p. 105].

Figura 24. Diagrama de bloques del sistema de reloj

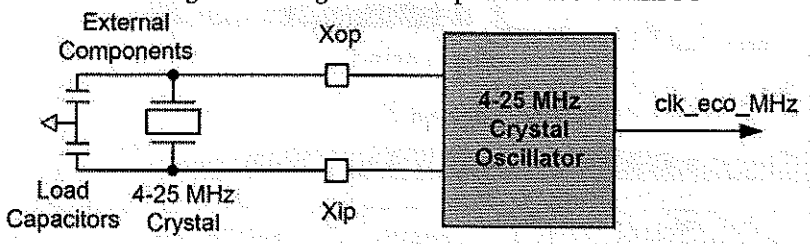


Fuente: [5, p. 23].

- Oscilador interno de baja velocidad (ILO) que puede operar a 1 kHz, 33kHz o 100 kHz.
- 24 a 80 MHz Phase–Locked Loop (PLL).

También es posible contar con tres fuentes externas de reloj que otorgan una mayor precisión, uno operando entre 4 y 25 MHz denominado MHzECO. En la figura 25 puede verse la conexión típica de este tipo de oscilador, otro a 32.768 kHz para aplicaciones Real Time Clock (RTC) denominado kHzECO y finalmente otro entre 0 y 66MHz denominado DSI.

Figura 25. Diagrama de bloques oscilador MHzECO



Fuente: [2, p. 110].

El sistema de manejo de la alimentación está conformado por entradas separadas para la alimentación analógica, digital y puertos de entrada/salida. El sistema también proporciona un regulador para el modo sleep, regulador para el bus de comunicaciones I2C y regulador para el modo hibernación.

Dentro de este sistema se cuenta también con un conversor de tipología Boost cuyo esquema se muestra en la figura 26. Este sistema puede tomar un voltaje externo proveniente de una batería o una celda solar y producir un voltaje de salida mayor; se debe conectar una bobina externa y un diodo schottky cuando se opera a más de 3,6v para su correcto funcionamiento.

El conversor Boost maneja dos modos de operación: Activo y Standby. El primero es el modo de operación normal donde el regulador Boost proporciona alimentación regulada al PSoC®5LP; el segundo modo se utiliza durante el modo Sleep para proporcionarle al PSoC®5LP la energía mínima para operar.

El PSoC®5LP incorpora también un monitor de voltaje que genera dos interrupciones cuando el voltaje está por encima o por debajo de la referencia deseada LVI (Low Voltage Interrupt) y HVI (High Voltage Interrupt) (ver figura 27).

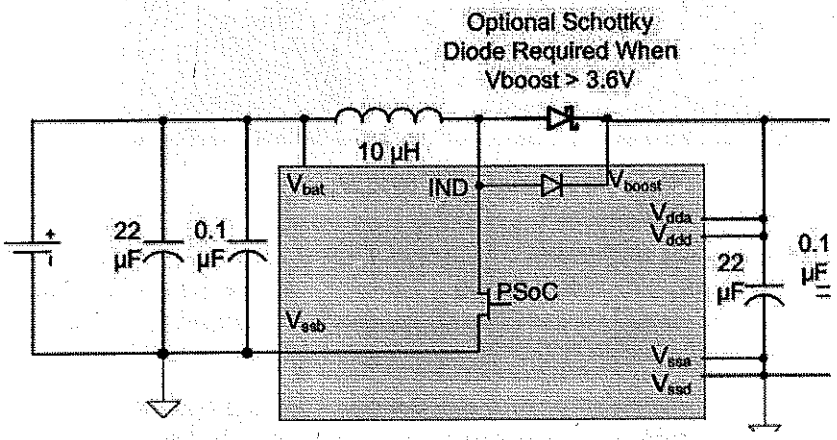
Sistema digital

El sistema digital programable del PSoC®5LP es el sistema más importante, puesto que es en esta arquitectura donde se desarrolla todo el procesamiento digital de la señal. Su configuración permite una flexibilidad muy generosa pues se puede progra-

mar desde pequeñas operaciones especializadas hasta algoritmos de gran exigencia computacional. La estructura del sistema digital del PSoC®5LP se muestra en la figura 28.

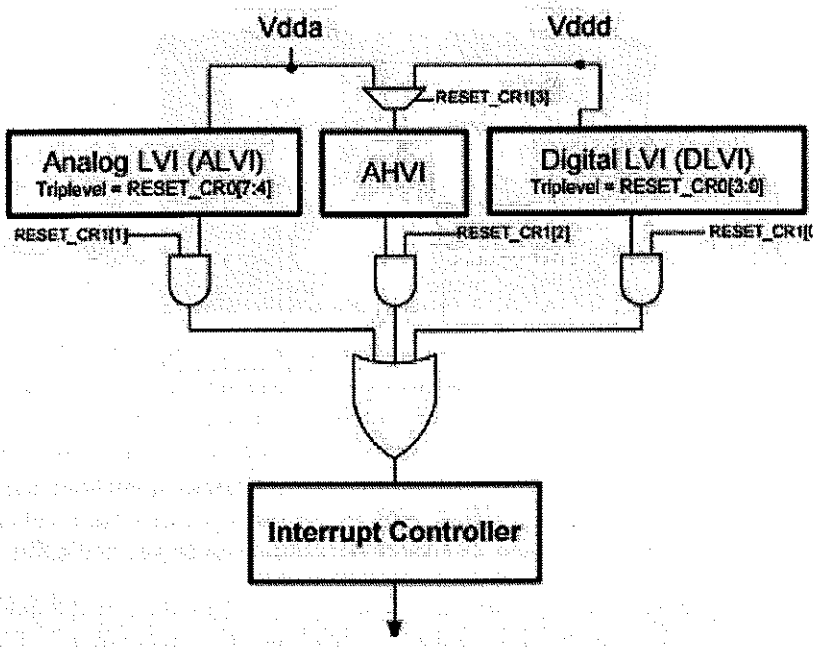
El sistema digital del PSoC®5LP está compuesto por 24 bloques digitales universales UDB, un controlador de red local CAN 2.0, bloques de comunicaciones I2C y USB 2.0 y cuatro bloques dedicados para implementar Timer, Counter y PWM.

Figura 26. Conversor Boost



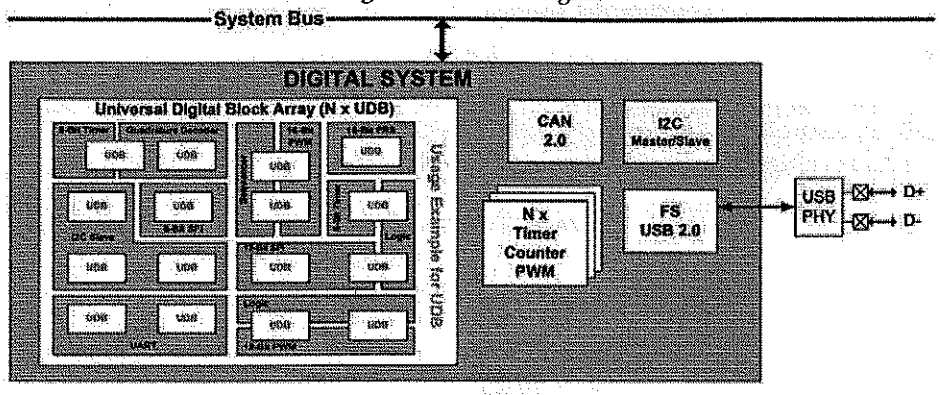
Fuente: [2, p. 123].

Figura 27. Diagrama de bloques del monitor de voltaje



Fuente: [2, p. 125].

Figura 28. Sistema digital



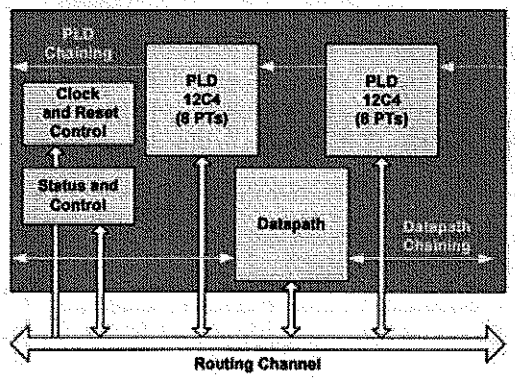
Fuente: [2, p. 166].

Bloques digitales universales

Los bloques digitales universales o Universal Digital Blocks (UDB) representan el núcleo de los dispositivos digitales y son la parte esencial y la gran novedad que ofrece el PSoC®. Con ellos es posible construir bloques más avanzados, son las unidades mínimas con las que cuenta el PSoC® para el manejo de funciones digitales.

Cada bloque contiene en su interior dos arreglos lógicos programables (PLD), un camino de datos, registros de control y estado, y un módulo de reset y control de reloj (ver figura 29).

Figura 29. Bloque universal digital



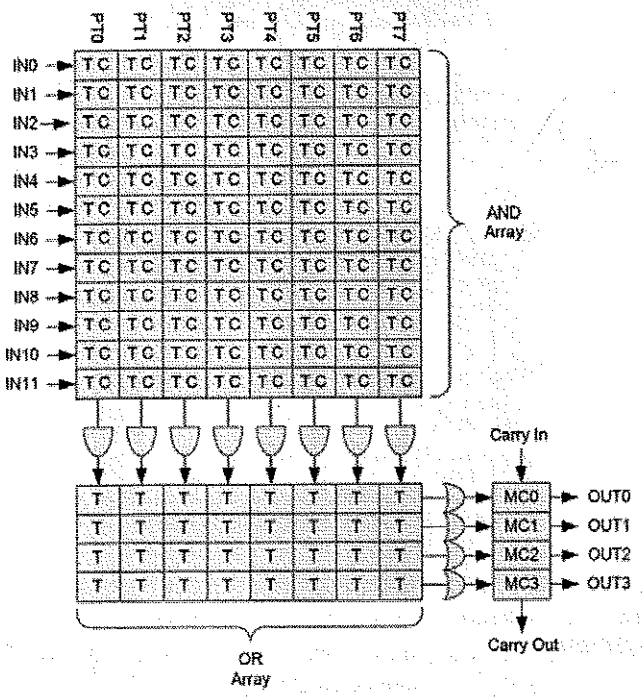
Fuente: [2, p. 168].

Cada PLD tiene 12 entradas configuradas en 8 productos lógicos; estos son sumados en un arreglo de 4 funciones OR y luego son transferidos por una macrocelda final que puede implementar una salida secuencial o combinacional (ver figura 30).

Cada PLD puede ser usado de manera individual o en conjunto con cualquier elemento del arreglo de bloques UDB; es decir, si una función requiere usar 7 PLDs y 5

caminos de dos es posible implementarlo, en total se tienen 48 PLDs a disposición, por lo tanto permite una flexibilidad de diseño muy importante.

Figura 30. Esquema del PLD interno



Fuente: [2, p. 169].

Camino de datos

El camino de datos (figura 31) es otro de los elementos importantes del sistema digital. En conjunto con los PLD, se disponen 24 de ellos y cada uno consta de dos registros de datos, dos registros acumuladores, dos registros de colas, una unidad aritmética y lógica (ALU) y demás componentes que hacen de este camino de datos una herramienta muy versátil al momento de procesar las señales digitales.

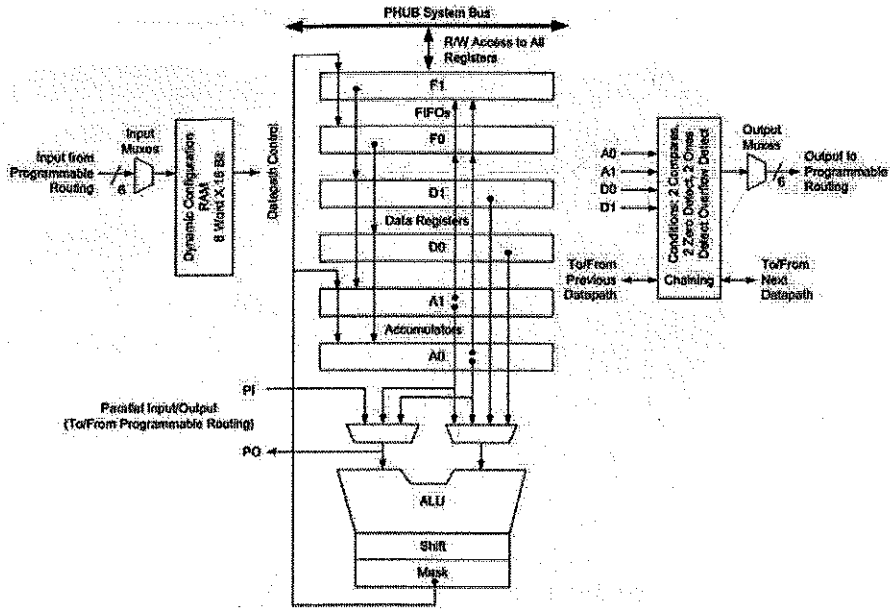
Arreglo de bloques universales digitales

Los bloques (UDB) son mapeados a través de una red o arreglo de bloques en forma de matriz, denominada Universal Digital Block Array (UDBA). Esta configuración permite la interconexión de todos los bloques y la flexibilidad para las funciones digitales que se deseen implementar.

Sistema digital de interconexión

El sistema digital de interconexión o Digital System Interconection (DSI) es un sistema especializado en el manejo de las señales digitales que se conectan a los UDB (Universal Digital Blocks), ya sean internas o provenientes de los pines de entrada/

Figura 31. Camino de datos



Fuente: [5, p. 42].

salida (GPIOs) del PSoC®. En la figura 32 puede verse un diagrama de bloques que muestra la interacción existente entre los bloques UDB Array (UDBA), DSI (Digital System Interconection) y los bloques digitales universales (UDB) del PSoC®5LP.

Sistema analógico

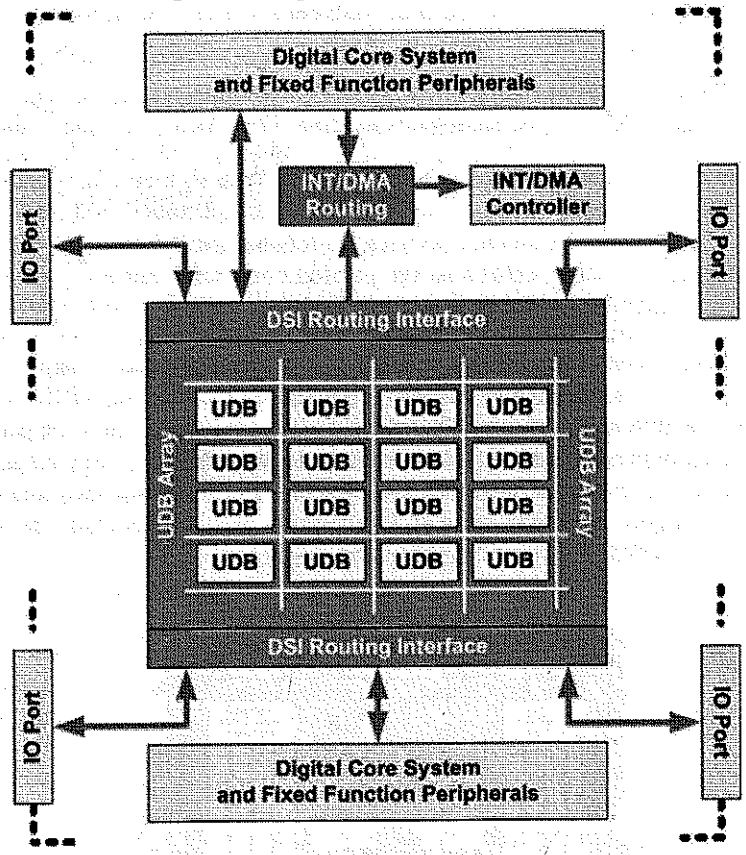
El sistema analógico programable del PSoC®5LP es el segundo sistema más importante, puesto que en su estructura modular es posible realizar el procesamiento de señales analógicas, tanto entrantes como salientes del sistema. Incluye además, entre otros, multiplexores analógicos, comparadores analógicos, mezcladores analógicos, amplificador de muestreo y retención, conversores analógico digital (ADC) delta sigma y SAR, conversores digital analógico (DAC), sensor capacitivo, sensor de temperatura y filtros digitales (DFB) (ver figura 33).

El PSoC®5LP tiene una arquitectura flexible de enrutamiento entre las señales analógicas de las GPIO y los bloques de recursos analógicos internos como el ADC y el DAC. Uno de los puntos fuertes de esta arquitectura de enrutamiento flexible es que permite la configuración dinámica de las conexiones de entrada/salida a los diferentes bloques analógicos, igual que ocurre con los bloques digitales.

El sistema de enrutamiento analógico del PSoC® tiene las siguientes características:

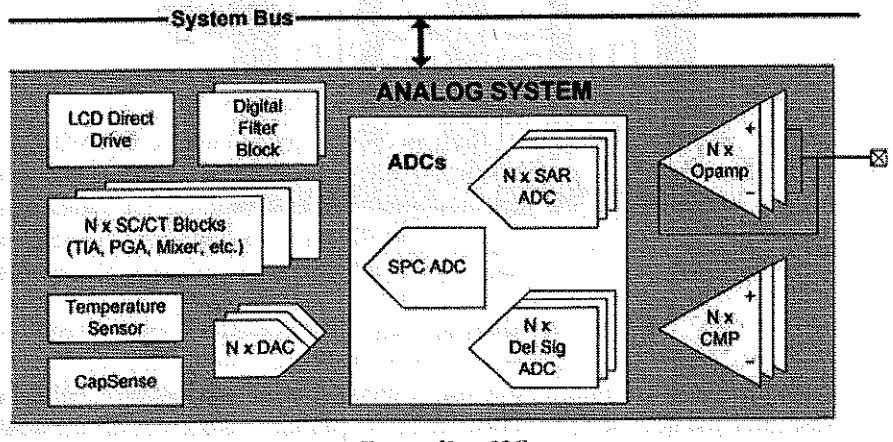
- Flexibilidad, es decir, la arquitectura de enrutamiento analógica es configurable.
- Enrutamiento dedicado para el driver LCD.

Figura 32. Arquitectura digital programable



Fuente: [2, p. 209].

Figura 33. Diagrama de bloques sistema analógico



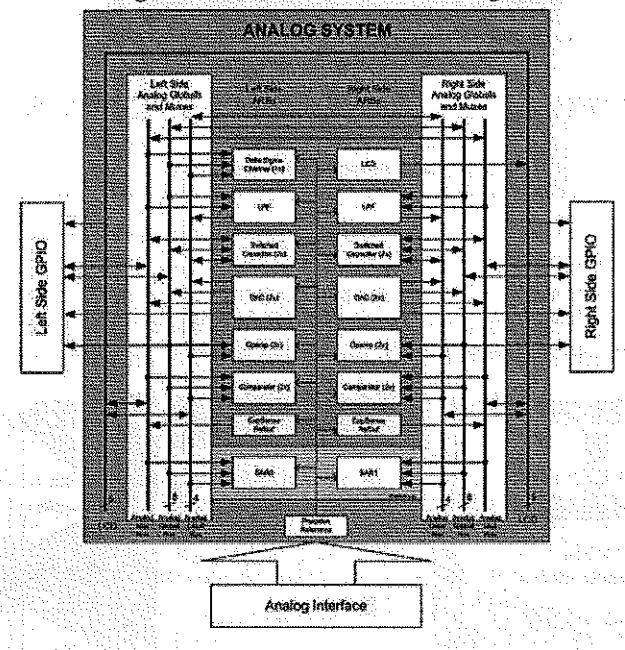
Fuente: [2, p. 296].

- Ocho bloques analógicos globales (AGs) y un bus analógico multiplexado (AMUXBUS) para las GPIO de cada lado del sistema, el cual está dividido en cuatro cuadrantes.
- Opciones de enrutamiento flexibles dentro del sistema analógico para interconectar bloques de recursos analógicos usando el bus local analógico (abus).

La operación del sistema analógico puede resumirse de la siguiente manera: los bloques analógicos globales (AGs) y el bus analógico multiplexado (AMUXBUS) van a los GPIO y proporcionan un camino para las señales entre estos y los bloques de recursos analógicos (ARBs); estos a su vez pueden conectarse entre sí por medio de los buses locales analógicos (abus).

El bus para pantalla de cristal líquido (LCDBUS) es utilizado para manejar las señales directamente hacia una pantalla LCD. Además, se tiene un bus VREF que lleva los voltajes de referencia para los diferentes bloques analógicos que son generados por el bloque de referencia de precisión. Finalmente, los interruptores y los multiplexores analógicos establecen conexiones entre los buses de enrutamiento analógico y los ARB. En la figura 34 puede verse el diagrama de bloques detallado del sistema analógico del PSoC®5LP.

Figura 34. Detalles del sistema analógico



Fuente: [2, p. 312].

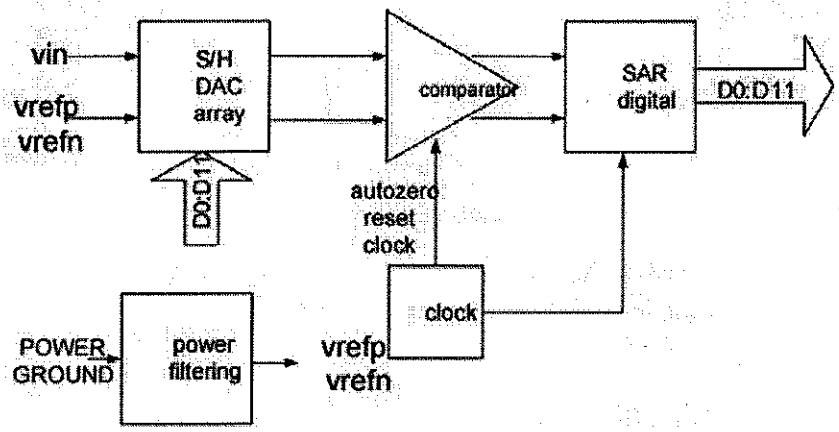
Uno de los bloques más importantes de este sistema son los conversores analógico digital (ADC) con los que cuenta el PSoC®5LP. El primero de ellos es el conversor Delta-Sigma. Esta tecnología permite obtener un excelente desempeño en la conver-

sión, puesto que tiene un rango de resolución amplio, una alta relación señal a ruido (SNR) y figura de ruido. Sus características principales se enlistan a continuación:

- De 8 a 20 bits de resolución.
- Ganancia de entrada configurable desde 0,25 a 256.
- Entrada diferencial o singular.
- Selección de referencia interna o externa.
- Modo continuo o incremental.

Otro conversor que se tiene a disposición es el de registro de aproximaciones sucesivas o (ADC-SAR), cuya característica principal es que está diseñado para trabajar con una resolución media, pero puede trabajar a altas velocidades de muestreo, teniendo así hasta 12 bits de resolución pero un muestreo que llega a 1 MSPS. También puede muestrear la señal y compararla con la salida del conversor digital analógico, con lo cual se logra seleccionar la resolución adecuada de conversión. En la figura 35 se puede observar el esquema de este conversor.

Figura 35. ADC-SAR

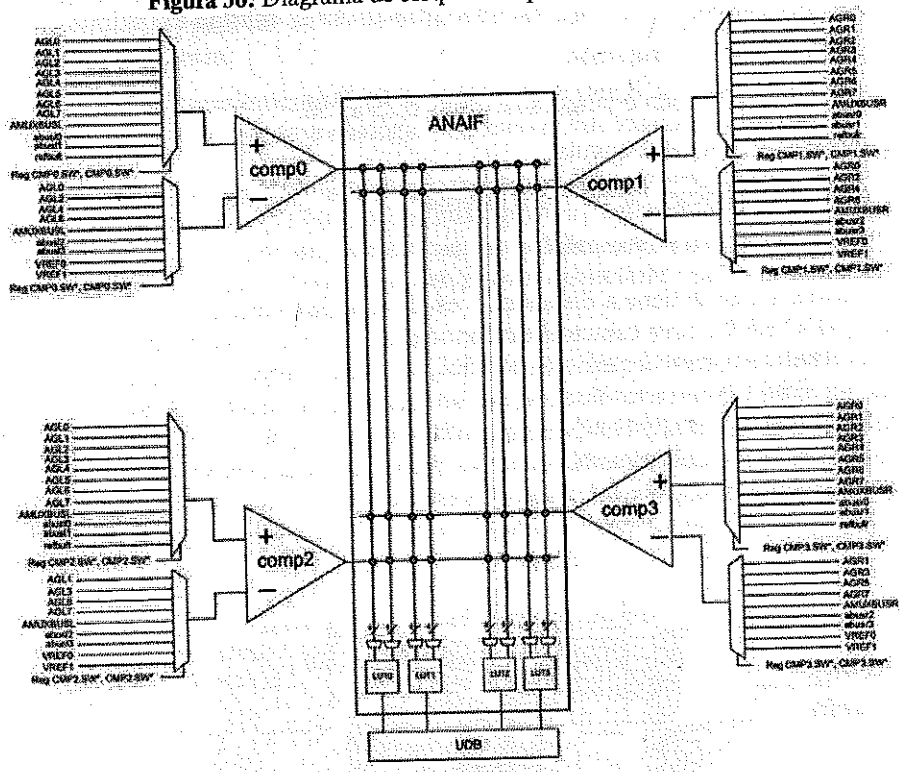


Fuente: [5, p. 54].

El PSoC®5LP también cuenta con 4 comparadores y 4 amplificadores operacionales discretos que pueden ser programados para diferentes aplicaciones. El diagrama esquemático del comparador se muestra en la figura 36. A cada comparador se le puede conectar cualquier entrada desde los pines analógicos del PSoC® y su comparación es registrada en un bloque digital.

Los amplificadores operacionales pueden manejar cualquier entrada del exterior dentro de ciertos límites o también pueden ser conectados internamente; tienen una leve deficiencia dado que cada uno de ellos tiene predefinidos su pines de entrada y salida, por lo que el usuario no puede configurar este parámetro a su acomodo, algunas de sus características más importantes se muestran a continuación:

Figura 36. Diagrama de bloques comparadores analógicos



Fuente: [2, p. 328].

- Capacidad de contacto de 25 mA.
- Ancho de banda de 3 MHz para una carga de 200 pF.
- Bajo ruido.
- Slew Rate de 3 V/µS
- Offset menor de 0.5 mV.

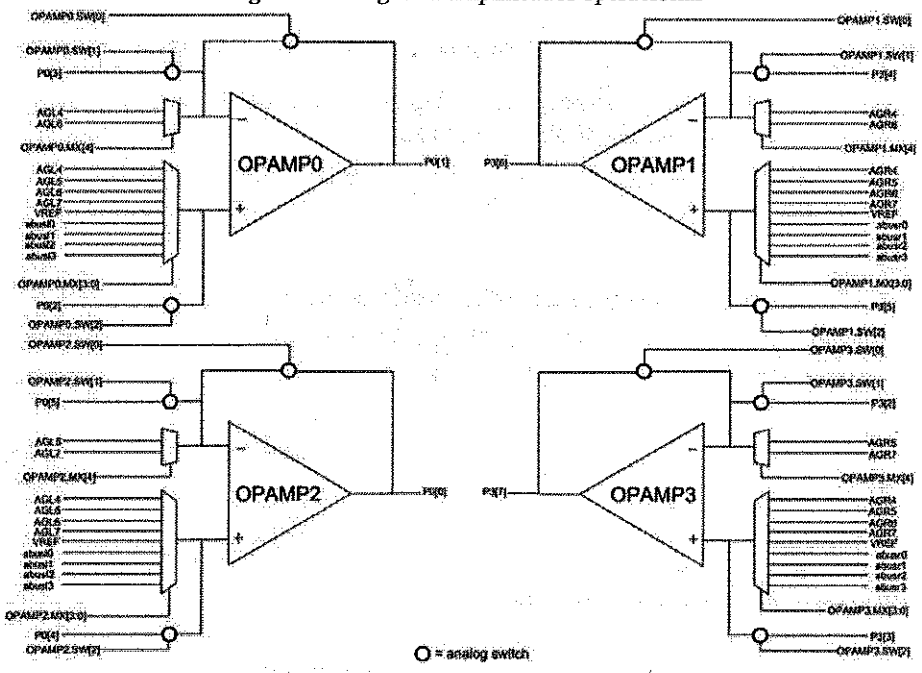
El diagrama de bloques de los amplificadores operaciones (OpAmp) se muestra en la figura 37.

Actualmente la tecnología de sensores capacitivos ha tenido un auge muy marcado, gracias a que los artículos de consumo electrónico cada vez más están enfocados en el manejo a través del tacto. El PSoC®5LP incluye como una de sus grandes novedades la tecnología de sensor capacitivo con su módulo CapSense®; este módulo ofrece la posibilidad de programar este tipo de tecnologías (figura 38).

Entre las características más importantes del CapSense se incluyen:

- Recursos para permitir la exploración de dos sensores capacitivos simultáneamente.

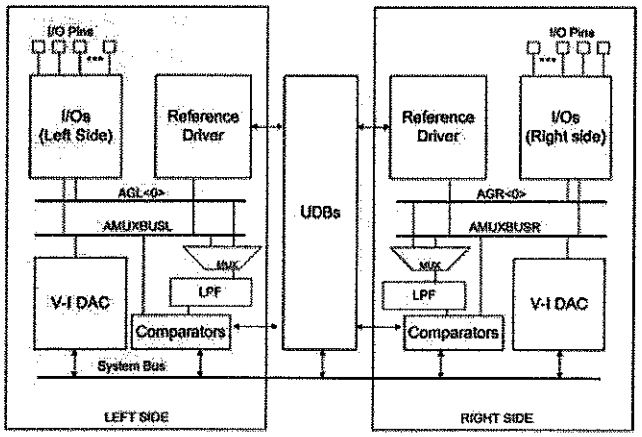
Figura 37. Diagrama amplificador operacional



Fuente: [2, p. 332].

- Filtro paso bajo configurable para eliminar ruido de conmutación.
- Buffer de referencia con alta corriente para permitir mediciones más rápidas.

Figura 38. Diagrama de bloques del módulo CapSense



Fuente: [2, p. 349].

Nomenclatura utilizada para nombrar los PSoC®s

Cypress, al igual que la gran mayoría de empresas fabricantes de semiconductores,

microprocesadores y microcontroladores, utiliza su propia nomenclatura para darle nombre a un dispositivo. Esta nomenclatura normalmente incluye información "escondida" que puede ser útil para el usuario final a la hora de seleccionar un dispositivo.

La estructura general de la nomenclatura asignada por Cypress para nombrar un PSoC®5LP es la siguiente: CY8Cabcdefg-LPxxx.

Donde:

CY8C:

Prefijo utilizado por Cypress para identificar los PSoC®s

a) Arquitectura

- 2: PSoC®1
- 3: PSoC®3
- 4: PSoC®4
- 5: PSoC®5

b) Subfamilia dentro de la arquitectura

- 2: CY8C52LP
- 4: CY8C54LP
- 6: CY8C56LP
- 8: CY8C58LP

c) Velocidad máxima

- 6: 67 MHz
- 8: 80 MHz

d) Capacidad de memoria flash

- 5: 32kB
- 6: 64kB
- 7: 128kB
- 8: 256kB

ef) Tipo de encapsulado

- AX: TQFP
- LT: QFN
- PV: SSOP
- FN: CSP

g) Rango de temperatura de operación

- C: Comercial
- I: Industrial
- Q: Extendido
- A: Automotriz

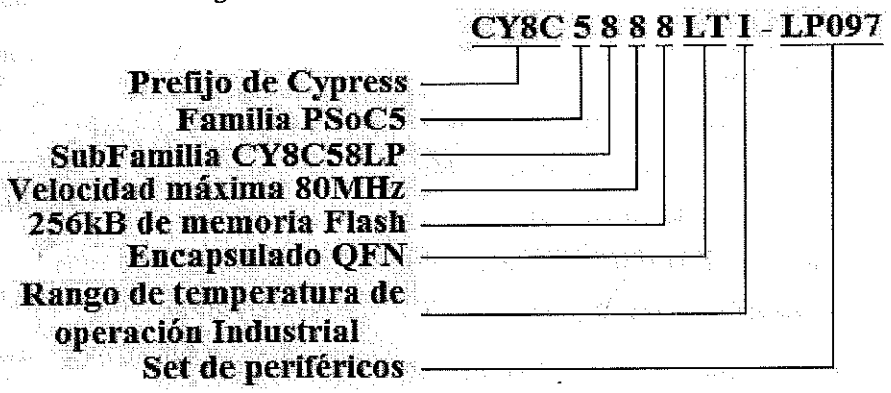
xxx: Set de periféricos

Valor numérico de tres dígitos que no tiene ningún significado

Así, por ejemplo, la referencia CY8C5888LTI-LP097 utilizada para el desarrollo de los problemas incluidos en este libro presenta las características mostradas en la figura 39:

Familia PSoC® CY8C58LP

Figura 39. Nomenclatura utilizada en los PSoC®s



La familia PSoC®5LP maneja diversas opciones en cuanto a dispositivos, tal como se mostró en la figura 3 del capítulo 1. Para el desarrollo de este libro se escogió un miembro de la familia CY8C58LP el CY8C5888LTI-LP097, incluido en el kit CY8CKIT-059 por su bajo costo y sus excelentes características frente a otros miembros de la gran familia PSoC®5LP.

Características generales del CY8C5888LTI-LP097

Rendimiento:

- Procesador Cortex-M3 de 32 bits con controlador DMA
- Controlador DMA (Direct Memory Access) de 24 canales
- Filtro digital (DFB) de 24 bits

Características de operación:

- Voltaje de operación desde +1,71v a +5,5v
- Temperatura de operación desde -40°C a +85°C
- Oscilador desde DC hasta 80MHz
- Consumo de energía: 3,1 mA a 6MHz y 15 mA a 48MHz
  2 µA en modo sleep
  300 nA en modo de hibernación con retención de datos en la RAM
- Regulador tipo Boost para entrada de 0,5v salida de 5v

Memoria:

- Flash de 256kB, con caché y características de seguridad
- Flash adicional de 32kB para corrección de error de código (ECC)
- RAM de 64kB
- E2PROM de 2kB

Periféricos digitales:

- 4 temporizadores de 16 bits c/u, contadores y bloques PWM (TCPWM)
- Módulo I2C con velocidad de hasta 1Mbps
- Puerto USB 2.0 full speed (FS)
- Bus full CAN 2.0b, con 16 buffers de recepción y 8 buffers de transmisión
- 24 bloques universales programables (UDB) que permiten crear:

Temporizadores de 8, 16, 24 o 32 bits, contadores y PWM

Interfaces I2C, UART, SPI, I2S y LIN 2.0

CRC (Cyclic Redundancy Check)

Generador de secuencias seudoaleatorias (PRS)

Decodificadores de cuadratura

Funciones lógicas a nivel de compuertas

Fuentes de reloj:

- Oscilador interno que opera entre 3 y 74 MHz, con precisión del 1% a 3 MHz
- Oscilador con cristal externo entre 4 y 25 MHz
- Generador de reloj con PLL interno hasta 80 MHz
- Oscilador interno de baja potencia de 1,33 y 100 kHz

- Oscilador con cristal externo de 32 kHz para aplicaciones de tiempo real
- 12 divisores de reloj para ser utilizados por periféricos del PSoC®

Periféricos analógicos:

- Conversor analógico digital (ADC) delta sigma con resolución de 8 a 20 bits
- 2 ADC por aproximaciones sucesivas (SAR) con resolución de 12 bits
- 8 conversores digital analógico (DAC) con resolución de 8 bits
- 4 comparadores
- 4 Amplificadores operacionales (OpAmp)
- 4 bloques analógicos programables que permiten implementar:

Amplificadores de ganancia programable (PGA)

Amplificadores de trasimpedancia (TIA)

Mezcladores

Circuitos de muestreo y retención

- Módulo CapSense® que soporta hasta 62 sensores capacitivos
- Voltaje de referencia interno de 1,024v ±0,1 %

Sistema de E/S:

- 48 líneas de E/S de las cuales 38 son de propósito general (GPIO)
- 8 líneas con altas prestaciones (SIO):

Corriente de 25 mA en modo sink

Umbral de voltaje entrada y de salida programable

Puede operar como un comparador de propósito general

- 2 pines USBIO que pueden ser utilizados como GPIO
- Enrutamiento de cualquier periférico analógico o digital a cualquier GPIO
- Uso de cualquier GPIO como drive para manejo directo de LCD de 46 x 16 segmentos
- Soporte del módulo CapSense® en cualquier GPIO
- Interfaces de voltaje de 1,2v a 5,5v

Programación y depuración:

- Soporta las interfaces: JTAG de 4 hilos, SWD (serial wire debug) de 2 hilos, SWV (single wire viewer) y Traceport de 5 hilos.

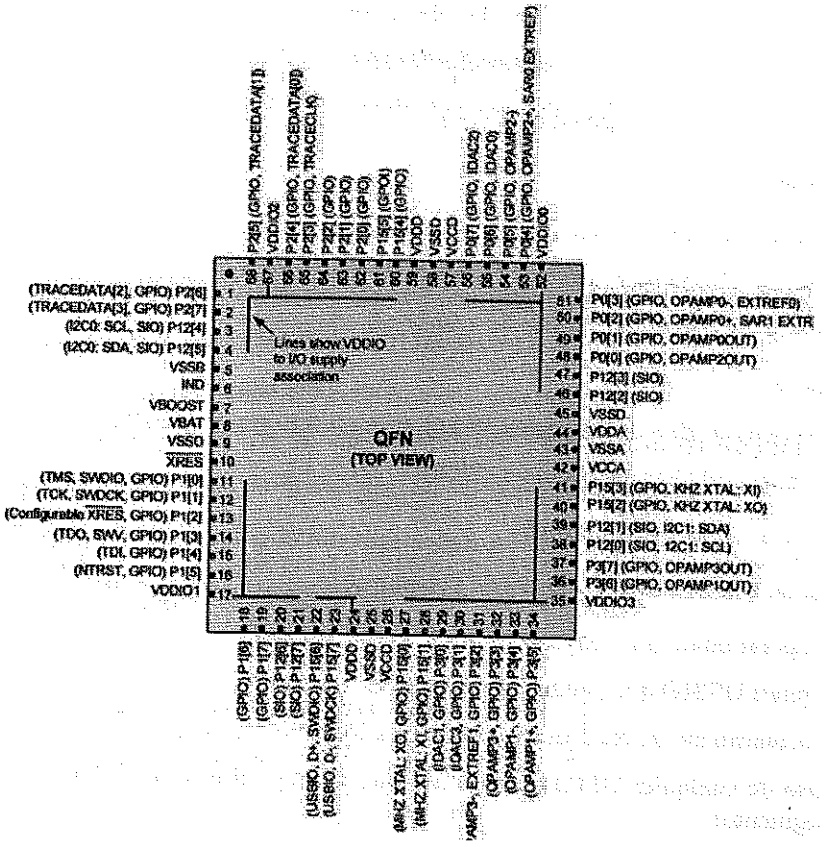
- Depuración ARM embebida en el core
- Programación por bootloader por medio de las interfaces I2C, SPI, UART y USB, entre otras

Encapsulado QFN de 68 pines

Distribución y función de pines del CY8C5888LTI-LP097

En la figura 40 puede verse la distribución de pines y el nombre asignados a cada uno de ellos para el PSoC®5LP de referencia CY8C5888LTI-LP097, que como se indicó en el apartado anterior maneja un encapsulado QFN de 68 pines.

Figura 40. Distribución de pines del PSoC® CY8C5888LTI-LP097



Fuente: [5, p. 7].

En el manual de usuario el fabricante sugiere conectar a la tierra digital (VSSD) lo que se denomina en el encapsulado QFN como pad térmico expuesto, que es gran superficie metálica de forma cuadrada y con similares características a la de un pad gigante, y a cuyo alrededor se encuentran los pads de conexión del dispositivo. Con

esta sugerencia se logra un mejor rendimiento mecánico, térmico y eléctrico. Si este pad no está conectado a tierra, debe quedar eléctricamente flotante y no conectarse a ninguna otra señal.

- VDDA:

Voltaje de entrada para todos los periféricos analógicos y para el regulador del bloque analógico. VDDA debe ser el voltaje más alto utilizado en el dispositivo, las demás fuentes de alimentación deben tener un valor menor o igual a este pin.

- VDDD:

Voltaje de entrada para todos los periféricos digitales y para el regulador del bloque digital. El valor de VDDD debe ser mayor o igual a VDDA.

- VDDIO0, VDDIO1, VDDIO2, VDDIO3:

Voltaje de alimentación para los pines de E/S. Cada pin VDDIO debe estar vinculado a un valor de voltaje de operación válido (1,71v a 5,5v), y debe ser menor o igual al valor del pin VDDA.

- VCCA:

Salida del regulador utilizado por los bloques analógicos, que requiere un condensador de 1μF conectado a VSSA. Esta salida no está diseñada para alimentar circuitos externos.

- VCCD:

Salida del regulador utilizado por los bloques digitales, que necesita un condensador de 1μF conectado a VSSD. Esta salida no está diseñada para alimentar circuitos externos.

- VSSA:

Señal de tierra de todos los periféricos analógicos.

- VSSD:

Señal de tierra de toda la lógica digital de los pines de E/S digitales.

- XRES y configurable XRES:

Pin de reset externo, activo en bajo con resistencia interna de pull-up. El pin P1 [2] puede ser configurado para operar como el pin XRES.

- IDAC0, IDAC1, IDAC2, IDAC3:

Pines de salida de baja resistencia y alta corriente de salida utilizado por los DAC (IDAC) de corriente.

- OPAMP0OUT, OPAMP1OUT, OPAMP2OUT, OPAMP3OUT:

Pines de salida de alta corriente de los amplificadores operacionales (OpAmp). El fabricante no recomienda su uso para conexión con el módulo CapSense.

- EXTREF0, EXTREF1:

Voltaje de entrada de referencia externo utilizado por el sistema analógico interno.

- SAR0 EXTREF, SAR1 EXTREF:

Pines de referencia externa de los ADC por aproximaciones sucesivas (SAR).

- OPAMP0-, OPAMP1-, OPAMP2-, OPAMP3-:

Pines de entrada inversora de los amplificadores operacionales (OpAmp).

- OPAMP0+, OPAMP1+, OPAMP2+, OPAMP3+:

Pines de entrada no inversora de los amplificadores operacionales (OpAmp).

- GPIO:

Estos pines proveen la interface entre la CPU y los periféricos digitales, analógicos, interrupciones, driver LCD y CapsSense del PSoC®.

- SIO:

Estos pines proveen la interface entre la CPU, periféricos digitales e interrupciones del PSoC® con comparadores analógicos, salidas de alta corriente en modo sink y salidas de alta impedancia.

- I2C0: SCL, I2C1: SCL

La línea I2C SCL permite sacar al PSoC® del modo sleep. Cuando esta funcionalidad no se requiere, puede utilizarse cualquier GPIO para general la señal SCL del bus I2C.

- I2C0: SDA, I2C1: SDA.

La línea I2C SDA permite sacar al PSoC® del modo sleep. Cuando esta funcionalidad no se requiere, puede utilizarse cualquier GPIO para general la señal SDA del bus I2C.

- MHz XTAL: Xo, MHz XTAL: Xi

Pines de conexión con cristal externo de 4 a 25 MHz. En la figura 41 puede verse la conexión típica para este tipo de oscilador sugerida por el fabricante.

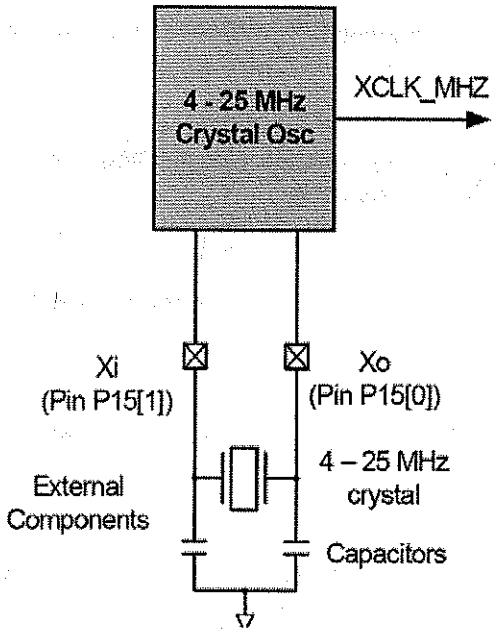
- kHz XTAL: Xo, kHz XTAL: Xi

Pines de conexión con el cristal externo de 32,768 Hz. En la figura 42 puede verse la conexión típica para este tipo de oscilador sugerida por el fabricante.

- SWDCK:

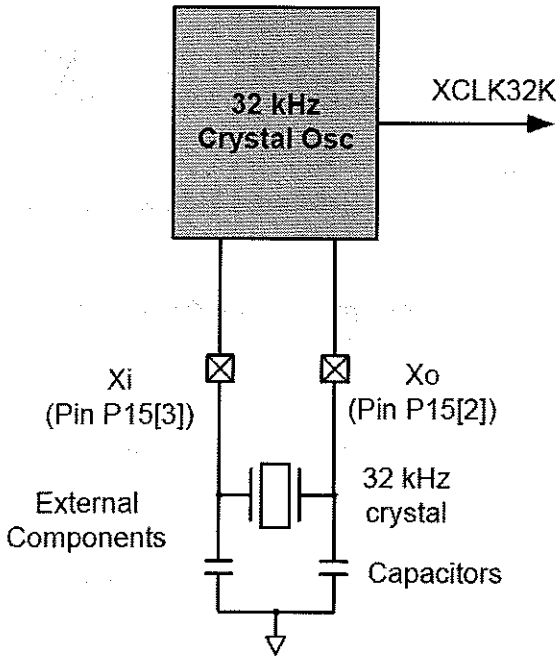
Señal de reloj de entrada conectada al puerto de programación y depuración SWD (serial wire debug).

Figura 41. Diagrama de bloques conexión MHzECO



Fuente: [5, p. 24].

Figura 42. Diagrama de bloques conexión 32kHxECO



Fuente: [5, p. 25].

- SWDIO:

Señal de datos bidireccional conectada al puerto de programación y depuración SWD (serial wire debug).

- NTRST:

Pin opcional del sistema de programación y depuración JTAG conectado al pin de reset del puerto JTAG (Test Reset).

- TCK:

Señal Test Clock de entrada conectada al puerto del sistema de programación y depuración JTAG.

- TDI:

Señal Test Data conectada al puerto del sistema de programación y depuración JTAG.

- TDO:

Señal de salida Test Data Out conectada al puerto del sistema de programación y depuración JTAG.

- TMS:

Señal Test Mode Select conectada al puerto del sistema de programación y depuración JTAG.

- TRACECLK:

Señal de reloj conectada al Cortex-M3 TRACEPORT.

- TRACEDATA[3:0]

Señales de salida de datos conectadas al Cortex-M3 TRACEPORT.

- SWV:

Salida de datos conectada al SWV (single wire viewer).

- USBIO, D+:

Pin de conexión de la señal D+ al puerto USB 2.0, el cual se polariza por medio del pin VDDD. Puede ser utilizado como un pin de E/S digital.

- USBIO, D-:

Pin de conexión de la señal D- al puerto USB 2.0, el cual se polariza por medio del pin VDDD. Puede ser utilizado como un pin de E/S digital.

- Ind:

Conexión de la bobina utilizada por el módulo boost pump (figura 43).

- VBOOST:

Conexión al sensor de potencia del módulo boost pump (figura 43).

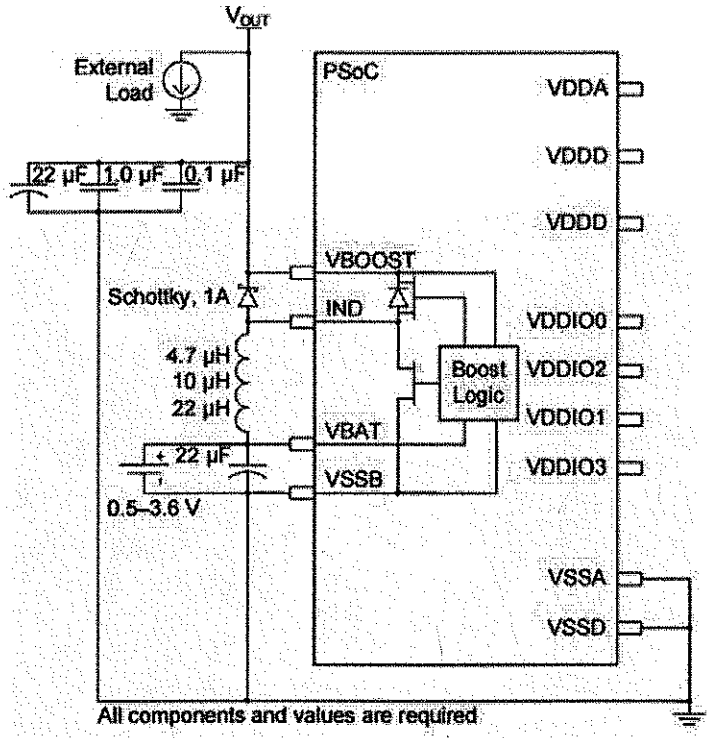
- VBAT:

Batería externa para alimentar al módulo boost pump (figura 43).

- VSSB:

Señal de tierra del módulo boost pump (figura 43).

Figura 43. Conexión típica conversor Boost

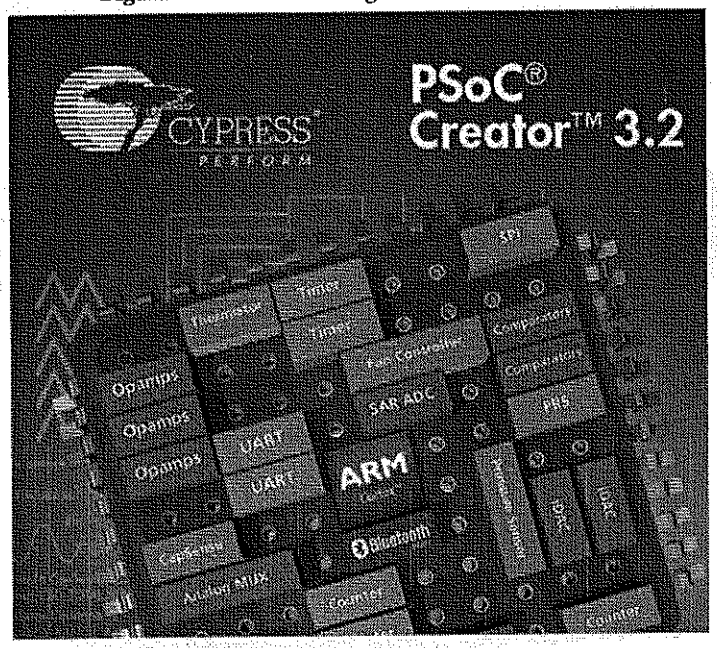


Fuente: [5, p. 29].

Software de programación para el PSoC®5LP

La compañía Cypress Semiconductor, fabricante del PSoC®5LP, proporciona un software de desarrollo gratuito denominado PSoC® Creator (disponible en www.cypress.com), con el cual se puede realizar toda la configuración, programación y simulación del PSoC® (figura 44).

Figura 44. Ventana de carga del PSoC Creator 3.2



Fuente: elaboración propia.

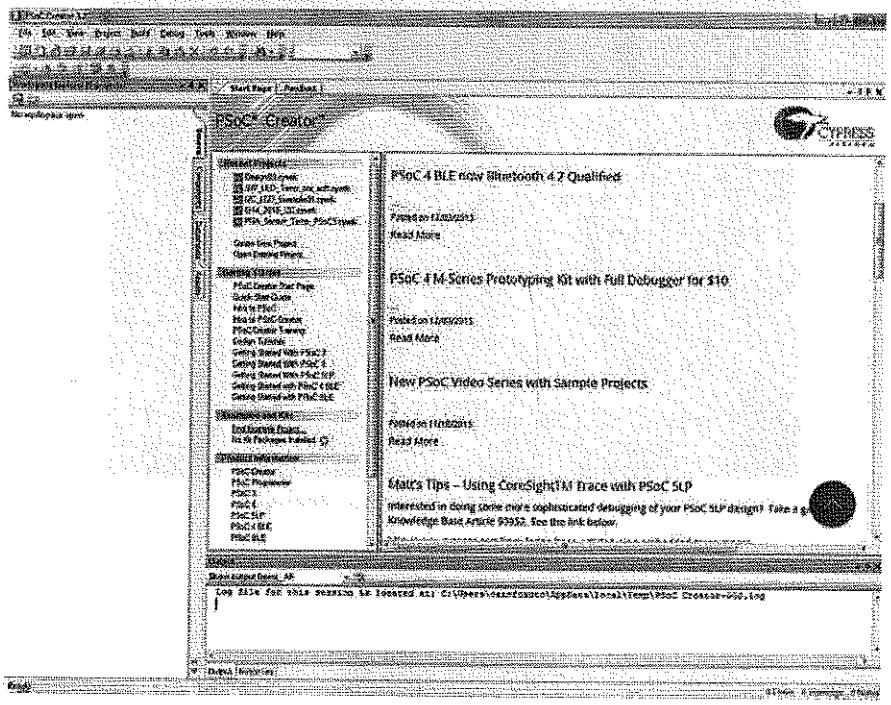
El PSoC® Creator es un entorno que permite el codiseño del software y del hardware que se vaya a programar en todos los dispositivos PSoC®3, PSoC®4, PSoC®4BLE y PSoC®5LP. Cuenta con una lista de más de 200 componentes que han sido predi-

señados, los cuales se pueden configurar dependiendo de las exigencias particulares de cada implementación. Además de esto, el PSoC® Creator permite el diseño de componentes nuevos que se requieran para funciones específicas que no sean cubiertas por los componentes existentes.

Los componentes de PSoC® Creator son "chips virtuales" analógicos y digitales, representados por un icono que los usuarios pueden arrastrar y utilizar en un diseño y luego configurar para adaptarse a las exigencias de la aplicación por desarrollar: además, incluye un conjunto completo de bibliotecas API que pueden ser generadas dinámicamente. Una vez que se ha configurado el sistema PSoC®, el firmware de aplicación puede ser escrito, compilado y depurado dentro del PSoC® Creator o exportado a IDEs de IAR, Keil y Eclipse. Los usuarios pueden crear componentes personalizados para ser usados dentro del PSoC® Creator, usando diagramas, máquinas de estado o con Verilog para optimizar aún más el uso de hardware.

En cada proyecto que se realice se deben utilizar por lo menos tres plantillas en el PSoC® Creator. La plantilla de diseño esquemático que permite incluir y configurar los componentes utilizados del catálogo, la plantilla de los recursos del sistema que configura los recursos del PSoC® y la plantilla de edición de código C que permite la edición del programa (firmware) que gobierna el diseño. En la figura 45 se presenta la página de inicio del PSoC® Creator v3.2.

Figura. 45 Página inicial PSoC® Creator v3.2



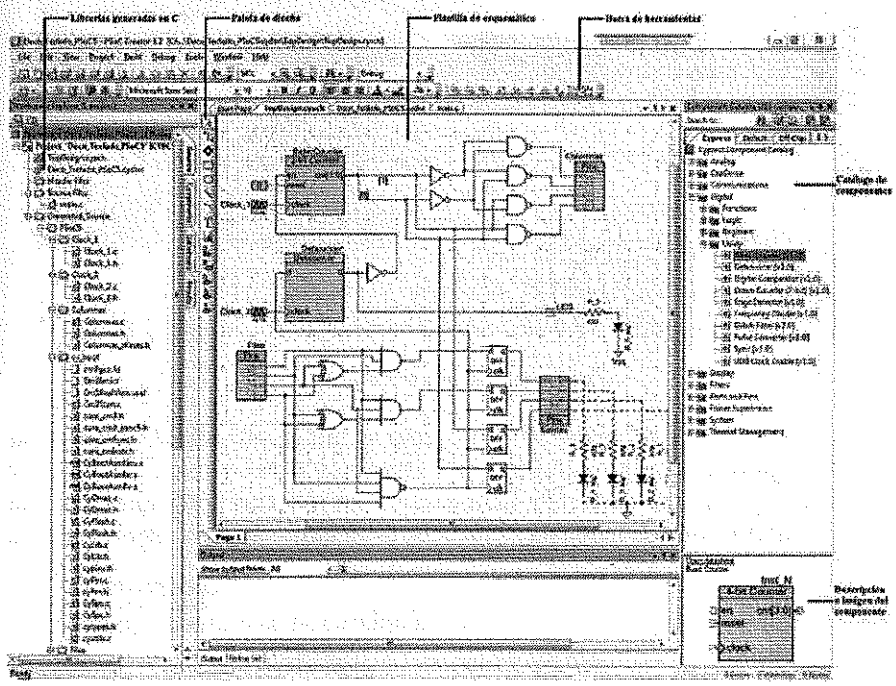
Fuente: elaboración propia.

Plantilla diseño esquemático

En la figura 46 se puede observar la plantilla de diseño esquemático. Esta plantilla tiene la finalidad de permitir la selección y configuración de todos los componentes que van a intervenir en el diseño. Estos componentes son seleccionados del catálogo de componentes, los cuales están clasificados por funcionalidades, incluye además un manual de usuario (datasheet) que contiene información del componente o componentes utilizados en el diseño.

También es allí donde se realizan todas las conexiones físicas entre componentes. Es posible además agregar información gráfica y de texto (comentarios) para darle claridad al diseño; esto puede hacerse con las herramientas de la paleta de diseño o con los elementos incluidos en la pestaña off-Chip del catálogo de componentes. Uno de los componentes esenciales son los pines de entrada/salida que permiten la configuración de puertos genéricos GPIO (general purpose input output) del PSoC® (ver capítulo 5), ya sean como conexiones analógicas o digitales. Esta plantilla siempre tiene como nombre de archivo la siguiente estructura: NOMBRE_PROYECTO.cysch. Para el caso del ejemplo de la figura 46 este se denomina Deco_Teclado_PSoC5.cysch.

Figura 46. Plantilla diseño esquemático



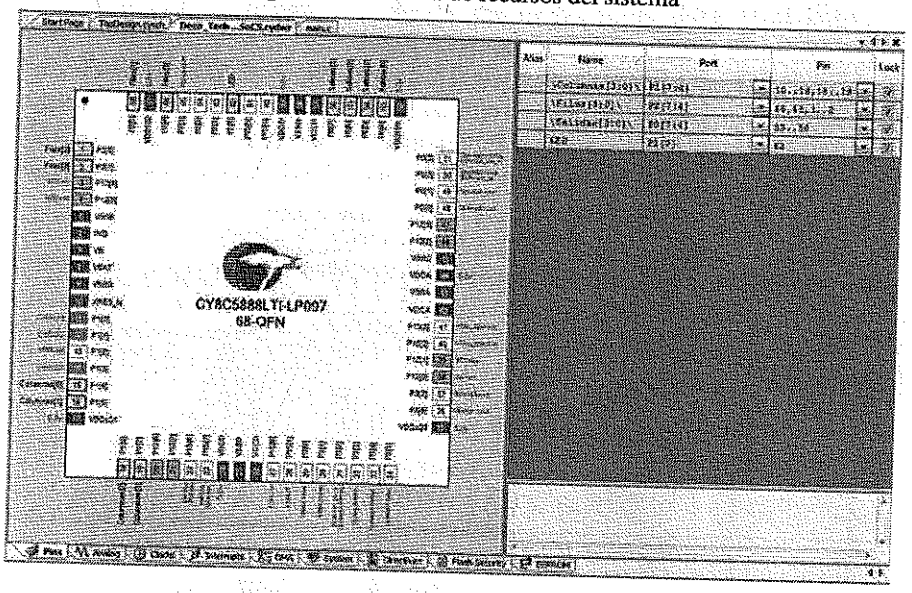
Fuente: elaboración propia.

Plantilla recursos del sistema

La plantilla de recursos del sistema (DWR) permite la configuración de todos los recursos que están relacionados con la operación del sistema del PSoC®. Esta plantilla se divide en varias secciones para la configuración específica de los recursos; estas secciones son: pines, relojes, interrupciones, canales DMA, sistema, directivas, seguridad de la memoria ROM y otras que pueden ser necesarias de acuerdo con el diseño que se esté realizando.

La pestaña Pins está activa por defecto y es la más utilizada; en esta se seleccionan los pines que utilizará cada uno de los recursos incluidos en la plantilla diseño esquemático Esta plantilla tiene como nombre de archivo la estructura NOMBRE_PROYECTO.cydwr. En la figura 47 se puede observar la plantilla de recursos del sistema, denominada Deco_Teclado_PSoC5.cydwr.

Figura 47. Plantilla de recursos del sistema



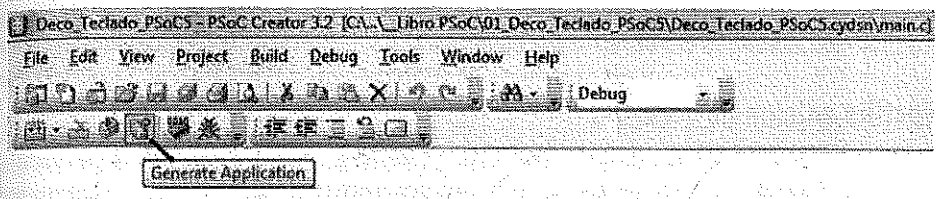
Fuente: elaboración propia.

Para finalizar el proceso de creación del proyecto se une la plantilla de diseño con todas las configuraciones allí realizadas, con los recursos del sistema utilizados y configurados en la respectiva plantilla. Esta tarea se ejecuta con la opción Generate Application, cuarto icono de izquierda a derecha de la barra de herramientas (figura 48).

El resultado de la operación de generación de la aplicación básicamente puede resumirse en dos tareas:

- La primera es la conexión interna o la generación por medio de bloques UDB (cuando sea necesario) o de los bloques analógicos programables de los compo-

Figura 48. Selección de herramienta para generar la aplicación

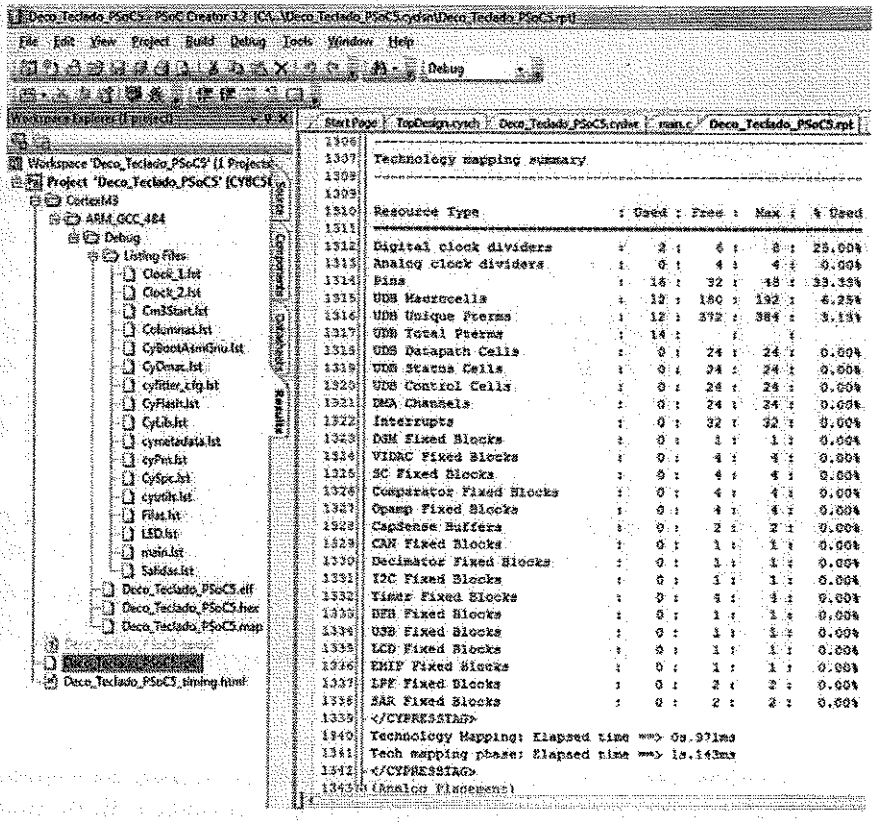


Fuente: elaboración propia.

nentes utilizados en la aplicación. El resumen del uso de recursos del PSoC® para la aplicación puede verse en la pestaña Result, seleccionando el archivo con la estructura NOMBRE_PROYECTO.rpt; en la figura 49 se puede observar parte de la información de uso de los recursos del PSoC® haciendo uso del archivo denominada Deco_Teclado_PSoC5.rpt.

- La segunda tarea no menos importante es la generación de librerías y funciones de cada uno de los componentes utilizados en la aplicación, con esta tarea se hace posible el uso de las APIs de los componentes que las incluyen, por ejemplo

Figura 49. Resumen uso de recursos del PSoC® en una aplicación



Fuente: elaboración propia.

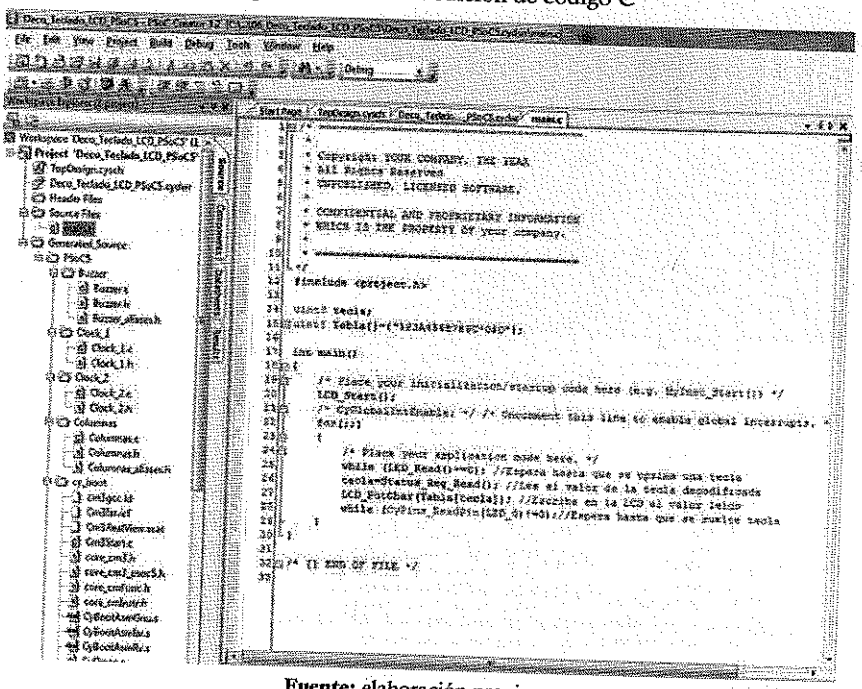
en una GPIO digital se tiene la posibilidad de leer o escribir un pin del PSoC® haciendo uso de las APIs (application programming interfaces): Pin_Read() o Pin_Write() respectivamente (en el capítulo 5 se profundizará sobre estas).

Plantilla edición de código C

La plantilla de edición de código C permite la escritura del algoritmo que será ejecutado en el PSoC®, el lenguaje que se usa para programar el dispositivo es el lenguaje C, dado que es un lenguaje universal, ampliamente utilizado en el mundo de los sistemas embebidos. También es posible programar el dispositivo en lenguaje de ensamblador, para lo cual se debe cambiar la plantilla de programación a este lenguaje.

Los componentes específicos que se deseen diseñar deben ser descritos en el lenguaje de descripción de hardware Verilog, para lo cual se debe incluir esta plantilla en el proceso de creación del componente adicional. Esto solo se hace necesario si el usuario desea crear su propio componente. La plantilla de edición de código permite también la inclusión de puntos de quiebre utilizados en el proceso de depuración del programa.

Figura 50. Plantilla edición de código C



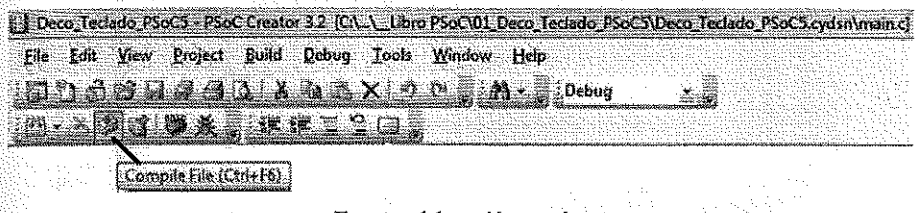
Fuente: elaboración propia.

Algunos componentes fueron diseñados para ser configurados por medio del lenguaje de programación. Estos componentes incluyen en su hoja de datos las interfaces aplicativas de programación o application programming interfaces (API), las cuales

describen qué argumentos se deben incluir para realizar su configuración. Si el componente requiere la inclusión de API, estas deben necesariamente ser introducidas en la plantilla de edición de código C. En la figura 50 se presenta la plantilla de edición de código C, la cual tiene como nombre de archivo main.c.

En la plantilla de edición de código también es posible hacer verificación de la sintaxis del algoritmo implementado en el PSoC® con la función Compile File (Ctrl+F6), tercer icono de izquierda a derecha de la barra de herramientas (figura 51), si hay algún error de sintaxis o de cualquier otro tipo este será indicado en la zona inferior de la plantilla de código denominada Output.

Figura 51. Selección de herramienta para compilar



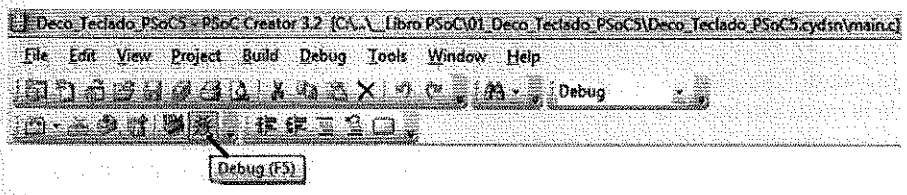
Fuente: elaboración propia.

Cuando el compilador reporta la no existencia de errores de sintaxis en el programa fuente, genera un archivo con el listado del programa (programa objeto) y otro con los códigos de operación de cada una de las instrucciones utilizadas en la aplicación (este archivo será utilizado en la fase de programación física del microcontrolador).

Debugger

Una de las funciones más importantes del PSoC® Creator es depuración debugger in circuit, que permite observar el comportamiento del diseño mientras el dispositivo está en funcionamiento y localizar posibles errores de lógica. Esta función únicamente opera si el PSoC® está conectado al PC por medio del programador o el sistema de desarrollo. Para realizar la simulación se hace uso del sexto icono de izquierda a derecha de la barra de herramientas (figura 52) o por medio del teclado utilizando la tecla de funciones F5.

Figura 52. Selección de herramienta para simular

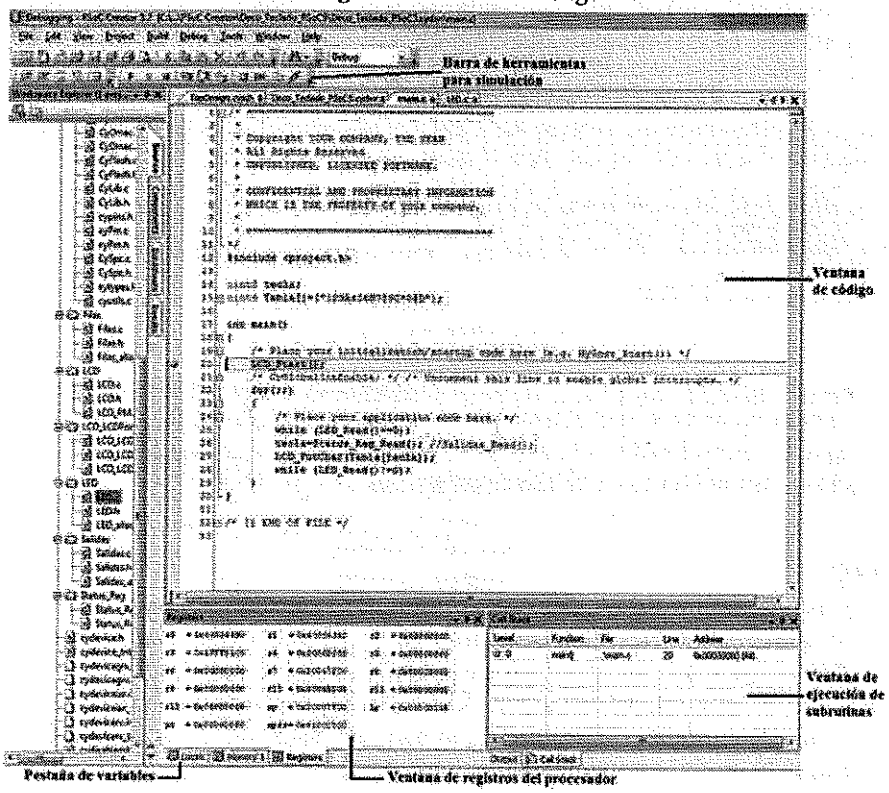


Fuente: elaboración propia.

La simulación permite ejecutar el código con la función Resume Execution (F5), pausarlo (Ctrl+Alt+Break), ejecutarlo paso a paso entrando a las subrutinas (Step Into tecla F11), ejecutarlo paso a paso sin ingresar a las subrutinas (Step Over tecla

F10), modificar el código, compilar y ejecutar nuevamente la simulación con la función Rebuild and Run (Ctrl+Shift+F5), generar un reset en el PSoC® (Shif+Alt+F5), evaluar variables del programa en la pestaña Locals, ver los registros que intervienen y sus valores (pestaña Registers), la memoria que está siendo usada y sus valores que puede ser vista en la pestaña Memory 1 y las funciones creadas por el código (ventana Call Stack). En la figura 53 se muestra la ventana de trabajo de la función Debug.

Figura 53. Función Debug



Fuente: elaboración propia.

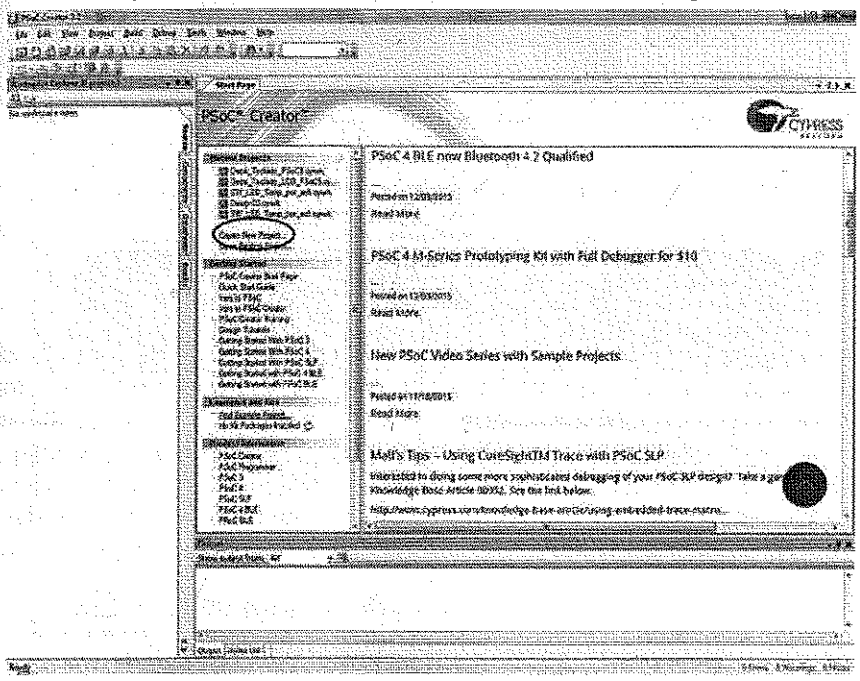
Creación de un proyecto con PSoC® Creator

Dado que el PSoC® Creator es un entorno de desarrollo integrado, todo el proceso desde la creación hasta la programación y simulación de la aplicación se realiza con él; de esta manera el usuario no tendrá que saltar de aplicación en aplicación.

Para la creación de un proyecto con el PSoC® Creator se deben seguir los siguientes pasos:

- Luego de ejecutar el PSoC® Creator, en la ventana de presentación inicial se selecciona la opción Create New Proyect (figura 54).

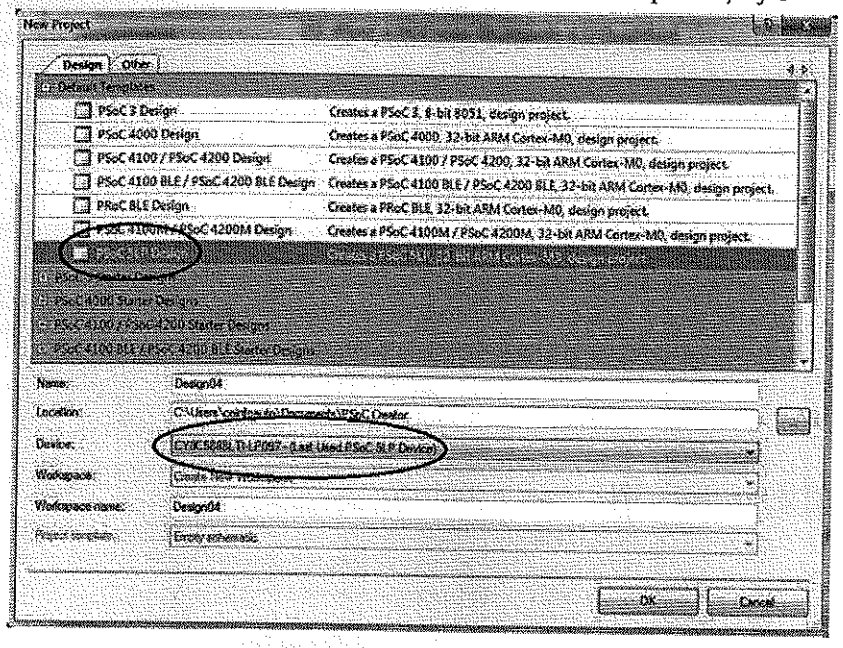
Figura 54. Creación de un proyecto con PSoC® Creator: paso 1



Fuente: elaboración propia.

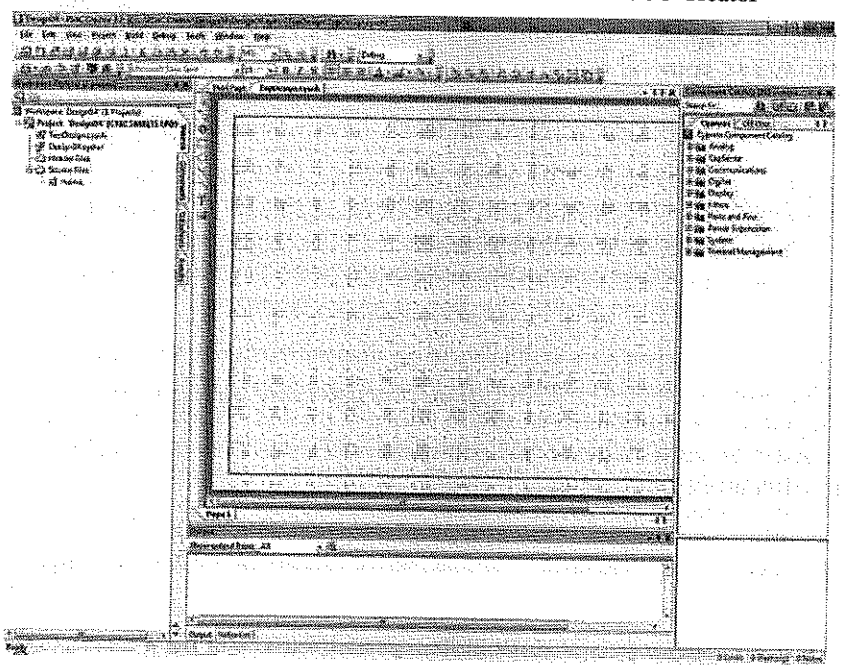
- Luego se despliega una ventana que permite seleccionar la familia del PSoC® por utilizar, como se indicó al inicio de este capítulo, el software de desarrollo PSoC® Creator está diseñado para soportar diversas familias de PSoC®s, en este caso se selecciona la opción PSoc5LP Design (figura 55).
- En la misma ventana se le asigna un nombre al proyecto (para este caso es ejemplo 1) y la ubicación de este, por defecto la ruta de ubicación del proyecto es: "Mis Documentos", allí en el momento de la instalación el PSoC® Creator crea una carpeta denominada PSoC Creator, pensada para almacenar todos los proyectos del usuario (figura 55). Su uso no es obligatorio.
- Finalmente se selecciona la referencia del dispositivo que se va a utilizar en el diseño, para este caso la referencia del PSoC® utilizado es CY8C5888LTI-LP097. Cuando no es la primera vez que se utiliza el PSoC® Creator, este almacena la última referencia utilizada, de esta manera el usuario evita hacer este último paso del proceso (figura 55).
- Se confirma y finalmente se crea el proyecto para que el usuario inicie el proceso de diseño en cada una de las plantillas mencionadas en este capítulo (figura 56).

Figura 55. Creación de un proyecto con PSoC® Creator: pasos 2, 3 y 4



Fuente: elaboración propia.

Figura 56. Presentación ventana de diseño inicial PSoC® Creator

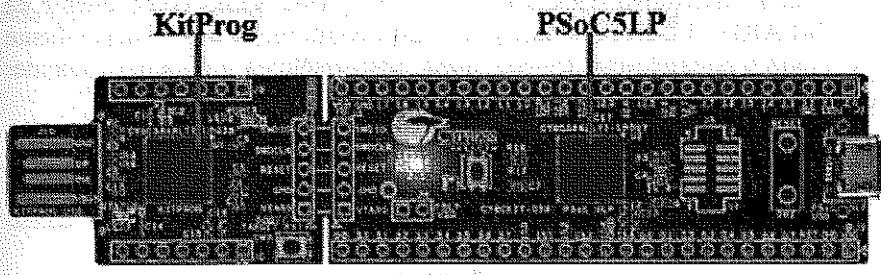


Fuente: elaboración propia.

Kit de desarrollo CY8CKIT-059 para PSoC®5LP

El kit CY8CKIT-059 para la familia PSoC®5LP (figura 57), utilizado para el desarrollo de los ejemplos en este libro, está diseñado para uso como plataforma de desarrollo de hardware o software en un entorno de laboratorio, de bajo costo y fácil uso, para ser utilizada directamente con el software de desarrollo PSoC® Creator. Esta tarjeta de desarrollo incluye un PSoC®5LP de referencia CY8C5888LTI-LP097 y el sistema denominado Cypress KitProg, que permite programar, hacer debug y la funcionalidad bridging que permite comunicación USB-UART y USB-I2C.

Figura 57. Board kit CY8CKIT-059



Fuente: [6, p. 4].

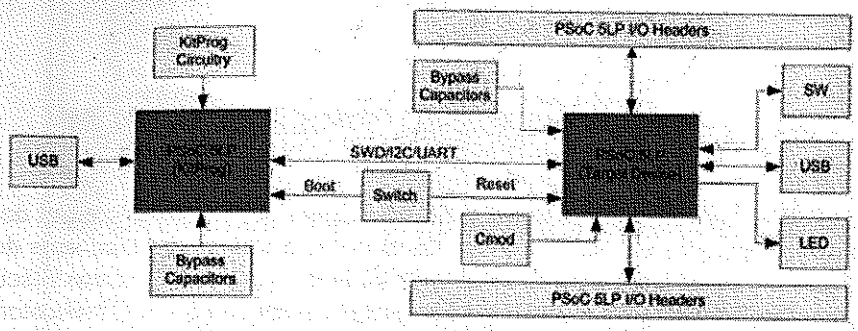
La tarjeta de desarrollo incluye además las siguientes características:

- Conector micro USB para aplicaciones con puerto USB
- Condensadores CMOD para habilitar la función CapSense
- Condensadores de bypass para mejor la calidad del ADC
- LED azul conectado a una GPIO
- Pulsador N.A. (normalmente abierto) conectado a una GPIO
- Condensadores de carga para permitir la conexión de cristal externo de 32kHz
- Alimentación directamente del puerto USB del PC

Diagrama de bloques del CY8CKIT-059

En la figura 58 puede verse de forma esquemática en un diagrama de bloques la estructura del kit CY8CKIT-059.

Figura 58. Diagrama de bloques del kit CY8CKIT-059



Fuente: [6, p. 13].

- KitProg:

  Es un programador/depurador integrado, que permite al usuario programar y depurar el dispositivo PSoC®5LP. También puede actuar como puente de la USB-UART y USB-I2C. Cuando se utiliza como un módulo independiente (separando o cortando las tarjetas), puede utilizarse para programar cualquier dispositivo de las familias PSoC® 3, 4 y 5LP a través de la interfaz SWD.

- Conectores de extensión:

  El kit de desarrollo permite el uso de todos los pines del PSoC®, esto por medio de dos conectores de extensión laterales; de esta manera se da al usuario acceso a todas las capacidades de dispositivo.

- Conector micro USB:

  El conector micro USB permite acceso al bloque USB interno del PSoC®5LP. Este conector permite desarrollar aplicaciones USB.

- LED de usuario:

  El LED incorporado puede utilizarse para mostrar el estado de una de las salidas del dispositivo PSoC®5LP, específicamente el pin P2.1. Este LED está conectado entre el pin y tierra por medio de una resistencia de 820 ohm. Esto incluye la modulación del brillo del LED para notificar diferentes estados del dispositivo.

- Pulsador (SW):

  Este kit tiene un pulsador conectado al pin P2.2 del PSoC®5LP en uno de sus extremos, —el otro está conectado a tierra—. Puede ser usado para generar estados a la entrada del PSoC®.

- Botón de Reset:

Este botón se utiliza para dos tareas en el kit de desarrollo:

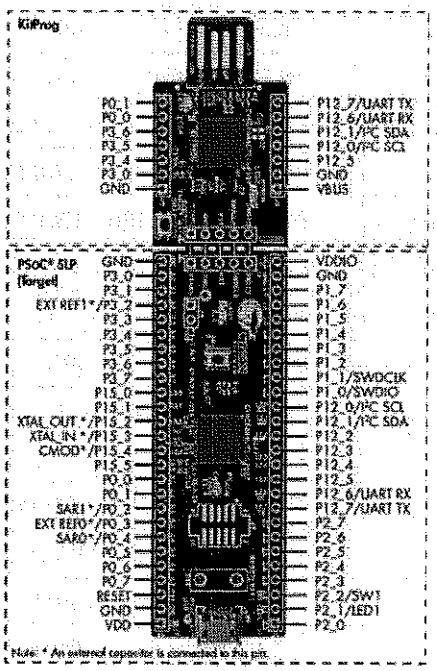
- Reset del PSoC®5LP: cuando se presiona, conecta a tierra la línea XRES del PSoC® y reestablece el dispositivo.
- Bootload KitProg: cuando se presiona el botón de reset mientras se conecta el kit al puerto USB del PC hace que el KitProg entre en modo bootloader.

Detalles del hardware del kit

La distribución de pines del kit CY8CKIT-059 puede verse en la figura 59, las dos partes que lo conforman tienen diferentes funcionalidades. La parte correspondiente al KitProg distribuye los pines en dos conectores de siete pines cada uno denominados J8 y J9 que el usuario debe soldar a la tarjeta. La parte correspondiente al PSoC®5LP se distribuye en dos conectores de 26 pines cada uno denominados J1 y J2, que el usuario también debe soldar, y que permiten el acceso a los pines del PSoC®.

Las dos partes en las que está dividido el kit están unidas por medio de caminos de cobre sobre el circuito impreso, que pueden ser separadas para utilizar de manera independiente el KitProg de la tarjeta con el PSoC®. Al separarse, estas partes pueden comunicarse nuevamente haciendo uso de los conectores J3 y J7 de cinco pines cada uno.

Figura 59. Tarjeta CY8CKIT-059



Fuente: [6, p.17].

Distribución de pines de la tarjeta con el PSoC®5LP

La zona del kit donde está ubicado el PSoC®5LP permite al usuario el uso formal del PSoC®, para conectar a este los periféricos o elementos externos que la aplicación que está desarrollando le exija. Como se indicó en el apartado anterior se dispone de dos conectores J1 y J2 de 26 pines cada uno para permitir esta conexión.

En las tablas 3 y 4 puede verse la ubicación y funcionalidad de cada uno de los pines que se incluyen en la tarjeta. El PSoC®5LP de referencia CY8C5888LTI-LP097 utilizado en la tarjeta es un dispositivo de 68 pines; no son incluidos todos los pines en los dos conectores, dado que algunos de ellos se requieren solo para la operación del dispositivo y son cableados directamente en la tarjeta.

Conector J1		
Pin	Señal	Descripción
J1_01	P2.0	GPIO
J1_02	P2.1	GPIO/LED1
J1_03	P2.2	GPIO/SW1
J1_04	P2.3	GPIO
J1_05	P2.4	GPIO
J1_06	P2.5	GPIO
J1_07	P2.6	GPIO
J1_08	P2.7	GPIO
J1_09	P12.7	GPIO/UART TX
J1_10	P12.6	GPIO/UART RX
J1_11	P12.5	GPIO/I2C0 SDA
J1_12	P12.4	GPIO/I2C0 SDL
J1_13	P12.3	GPIO
J1_14	P12.2	GPIO
J1_15	P12.1	GPIO/I2C_SDA
J1_16	P12.0	GPIO/I2C_SDL
J1_17	P1.0	GPIO/SWDIO*
J1_18	P1.1	GPIO/SWDCLK*
J1_19	P1.2	GPIO
J1_20	P1.3	GPIO*

so formal del
la aplicación
e dispone de
.

de los pines
I-LP097 uti-
los pines en
peración del

Conector J1		
Pin	**Señal**	**Descripción**
J1_21	P1.4	GPIO
J1_22	P1.5	GPIO
J1_23	P1.6	GPIO
J1_24	P1.7	GPIO
J1_25	GND	Tierra
J1_26	VDDIO	Alimentación

Tabla 3. Conector J1 tarjeta con PSoC®
Fuente: [6, p.22].

Conector J2		
Pin	**Señal**	**Descripción**
J2_01	VDD	Alimentación
J2_02	GND	Tierra
J2_03	RESET	Reset
J2_04	P0.7	GPIO
J2_05	P0.6	GPIO
J2_06	P0.5	GPIO/AO2 V-
J2_07	P0.4	GPIO/ AO2 V+
J2_08	P0.3	GPIO/AO0 V-
J2_09	P0.2	GPIO/AO0 V+
J2_10	P0.1	GPIO/AO0 Vout
J2_11	P0.0	GPIO/AO2 Vout
J2_12	P15.5	GPIO
J2_13	P15.4	GPIO/CMOD
J2_14	P15.3	GPIO/XTAL_IN
J2_15	P15.2	GPIO/XTAL_OUT
J2_16	P15.1	GPIO
J2_17	P15.0	GPIO
J2_18	P3.7	GPIO/AO3 Vout

Continúa

Conector J2		
Pin	Señal	Descripción
J2_19	P3.6	GPIO/AO1 Vout
J2_20	P3.5	GPIO/AO1 V+
J2_21	P3.4	GPIO/AO1 V-
J2_22	P3.3	GPIO/AO3 V+
J2_23	P3.2	GPIO/AO3 V-
J2_24	P3.1	GPIO
J2_25	P3.0	GPIO
J2_26	GND	Tierra

Tabla 4. Conector J2 tarjeta con PSoC®
Fuente: [6, p.22].

La alimentación de la tarjeta hace uso del voltaje que provee el puerto USB del PC, donde es conectado el kit. Este voltaje, que es de +5v, puede ser utilizado para alimentar dispositivos externos que tengan un bajo consumo de energía; para esto se puede hacer uso del pin VDD (J1_26). Al separar esta tarjeta de la tarjeta KitProg puede polarizarse externamente el PSoC® con un valor de voltaje diferente o de mayor potencia si la aplicación lo requiere.

Distribución de pines de la tarjeta KitProg

La parte del kit denominada KitProg, que es utilizada principalmente para la programación del PSoC®, incluye además la posibilidad de operar como un puente de comunicación USB-UART o un puente USB-I2C. Se dispone de dos conectores J8 y J9 de siete pines cada uno para tener a disposición del usuario la funcionalidad de puente de comunicaciones.

Las tablas 5 y 6 muestran la ubicación y funcionalidad de cada uno de los pines que se incluyen en la tarjeta KitProg.

Conector J8		
Pin	Señal	Descripción
J8_01	GND	Tierra
J8_02	P3.0	GPIO
J8_03	P3.4	GPIO
J8_04	P3.5	GPIO

Conector J8		
Pin	Señal	Descripción
J8_05	P3.6	GPIO
J8_06	P0.0	GPIO
J8_07	P0.1	GPIO

Tabla 5. Conector J8 tarjeta KitProg
Fuente: [6, p.23]

Conector J9		
Pin	Señal	Descripción
J9_01	VBUS	Alimentación
J9_02	GND	Tierra
J9_03	P12.5	GPIO
J9_04	P12.0	GPIO/I2C SCL
J9_05	P12.1	GPIO/I2C SCDA
J9_06	P12.6	GPIO/UART RX
J9_07	P12.7	GPIO/UART TX

Tabla 6. Conector J9 tarjeta KitProg
Fuente: [6, p.23]

Conexión entre la tarjeta KitProg y la tarjeta con el PSoC®5LP T2

La funcionalidad más importante de la tarjeta KitProg es la posibilidad de operar como un sistema de programación independiente para las familias de PSoC®3, PSoC®4 y PSoC®5LP. Esto se da ya que se pueden separar físicamente las dos partes del kit CY8CKIT-059, el KitProg y la tarjeta del PSoC®5LP.

En las tablas 7 y 8 pueden verse las señales y funciones asociadas a cada una de las tarjetas

Conector J3 (Tarjeta PSoC®)		
Pin	Señal	Descripción
J3_01	VTARG	Alimentación
J3_02	GND	Tierra

Continúa

Conector J3 (Tarjeta PSoC®)		
Pin	Señal	Descripción
J3_03	XRES	Reset
J3_04	P1.1	SWDCLK
J3_05	P1.0	SWDIO

Tabla 7. Conector J3 tarjeta con PSoC®5LP
Fuente: [6, p.23].

Conector J7 (KitProg)		
Pin	Señal	Descripción
J3_01	VDD	Alimentación
J3_02	GND	Tierra
J3_03	P12.4	PROG_XRES
J3_04	P12.3	PROG_SWDCLK
J3_05	P12.2	PROG_SWDIO

Tabla 8. Conector J7 del lado del KitProg
Fuente: [6, p.23].

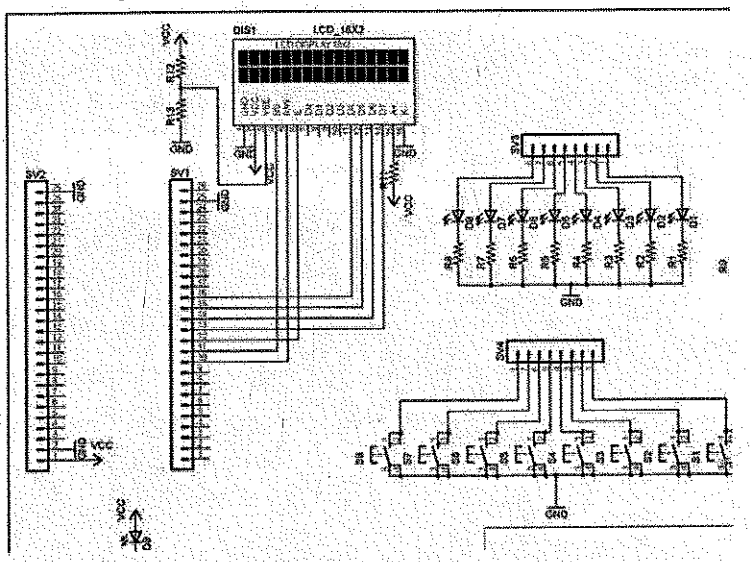
Tarjeta de periféricos para el kit CY8CKIT-059 T1

Aunque el kit CY8CKIT-059 es muy versátil por su diseño sencillo, carece de periféricos externos para su operación, escasamente cuenta con un pulsador N.A y un LED de color azul como periféricos para iniciar rápidamente con el uso del PSoC®.

Para solucionar esta falencia se propone la implementación de una tarjeta con los periféricos o elementos utilizados en la mayoría de aplicaciones típicas, esto es: una LCD de 16x2 con luz de fondo (back light), un grupo de 8 LED, un dipswitch de 8 interruptores, 8 pulsadores N.A., un buzzer de +5v, un potenciómetro y un conector de 8 pines para conexión de un teclado matricial de hasta 4x4 teclas o cualquier otro elemento que se requiera conectar con el PSoC®. Todo esto facilita la implementación rápida de proyectos. Cuando se necesite otro tipo de periféricos con los conectores J1 y J2 de la tarjeta CY8CKIT-059, se pueden realizar conexiones rápidas hacia el exterior. La tarjeta será utilizada para resolver todos los ejemplos presentados en este libro.

En la figura 60 se muestra el diagrama esquemático de la tarjeta propuesta, los conectores SV1 y SV2 se utilizan para realizar la conexión a la tarjeta CY8CKIT-059. La LCD utilizada es conectada al puerto P12 (líneas P12.0 a P12.6) del PSoC®5LP,

Figura 60. Diagrama esquemático tarjeta con periféricos para el CY8CKIT-059

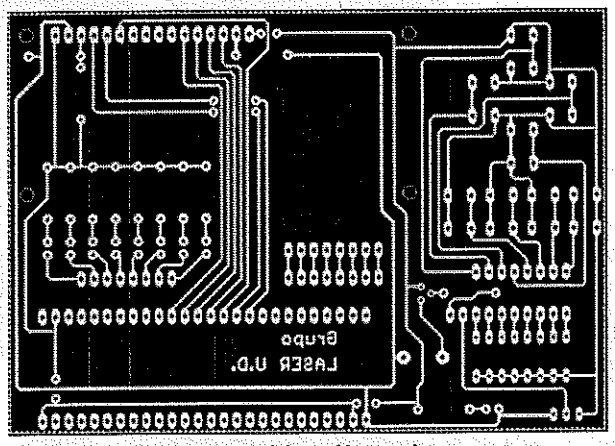


Fuente: elaboración propia.

por lo que esta conexión debe tenerse en cuenta al utilizarse dicho bloque en el PSoC Creator. Se utiliza un conector de 16 pines hembra para tener la posibilidad de retirar la LCD de la tarjeta cuando no se requiera en una aplicación y se deban utilizar las GPIO del puerto P12. Los demás elementos de la tarjeta disponen de conectores propios para que el usuario seleccione las GPIO por utilizar con ellos, y así aprovechar una de las bondades del PSoC®, que es la posibilidad de configurar estas líneas de acuerdo con las necesidades de la aplicación.

Para reducir costos de fabricación y simplificar el diseño, se implementó un circuito impreso de una sola cara de tamaño 11,7cm x 8,2 cm, como se muestra en la figura 61.

Figura 61. Circuito impreso tarjeta con periféricos para el CY8CKIT-059



Fuente: elaboración propia.

La tabla 9 muestra el listado completo de componentes utilizados en la fabricación de la tarjeta, las cantidades y sus respectivos valores o referencia comercial.

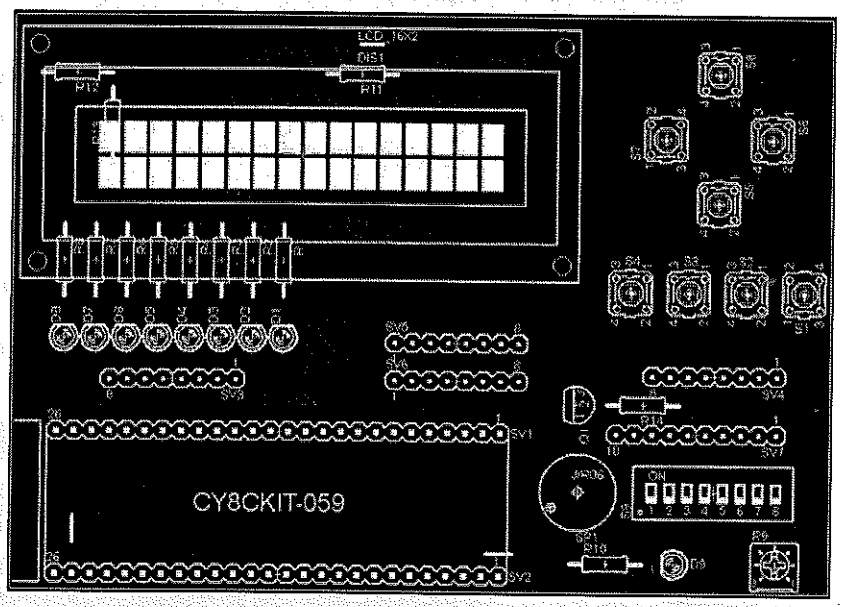
Componente	Nombre	Valor o referencia	Cantidad
Resistencia 1/4W	R1, R2, R3, R4, R5, R6, R7, R8	470Ω	8
	R10	330Ω	1
	R11	100Ω	1
	R12	2.2kΩ	1
	R13	390Ω	1
	R14	1kΩ	1
Potenciómetro	R9	10kΩ	1
Transistor	Q1	2N2222 o 2N3904	1
LED	D1, D2, D3, D4, D5, D6, D7, D8, D9	3mm rojo	9
LCD	DIS1	16x2 con BL QY1602 o similar	1
Pulsador	S1, S2, S3, S4, S5, S6, S7, S8	N.A 4 pines	8
Dip Switch	S9	8 interruptores	1
Buzzer	SP1	Buzzer 6v	1
Conector	SV1, SV2	26 pines hembra	2
	SV3,SV4, SV6	8 pines hembra	3
	SV5	8 pines macho	1
	SV7	10 pines hembra	1
	DIS1	16 pines hembra	1

Tabla 9. Listado de componentes tarjeta prototipo
Fuente: elaboración propia.

En la figura 62 puede verse el screen o máscara de componentes que permiten identificar fácilmente cada uno de los componentes sobre la superficie de la tarjeta diseñada.

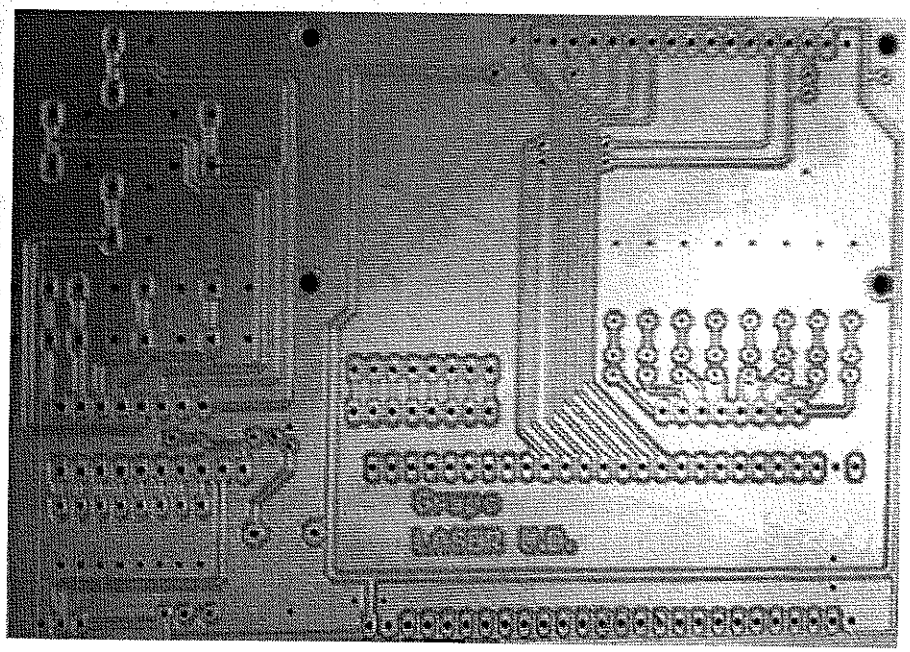
Luego de la fabricación de circuito impreso (ver resultado en la figura 63), debe tenerse en cuenta que lo mostrado corresponde a una tarjeta prototipo a la que no se le incluyó la capa de antisolder ni el screen. Se sugiere al lector incluirlos en su implementación, ya que la capa de antisolder impide que el circuito impreso se oxide por el

Figura. 62. Máscara de componentes tarjeta prototipo



Fuente: elaboración propia.

Figura 63. Circuito impreso real

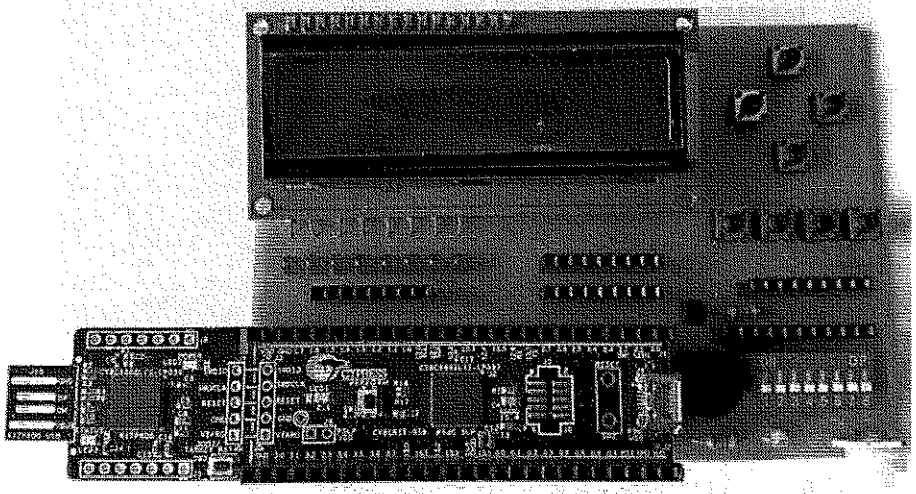


Fuente: elaboración propia.

continuo contacto con la humedad del ambiente y de la piel del usuario; además, facilita el proceso de soldadura de los componentes, el screen como se indicó antes facilita la identificación y la ubicación correcta de todos los componentes de la tarjeta.

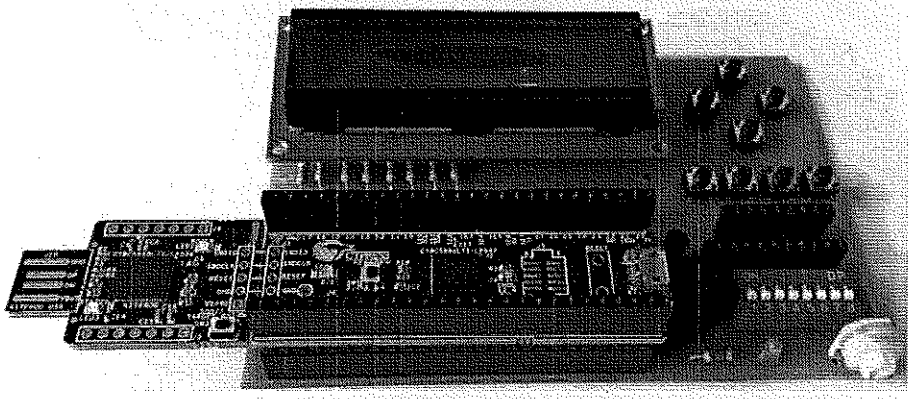
Al finalizar el montaje de cada uno de los componentes, el resultado obtenido para la tarjeta prototipo puede verse en las figuras 64 y 65. En esta última se incluye la tarjeta del kit CY8CKIT-059.

Figura 64. Vista frontal de la tarjea prototipo



Fuente: elaboración propia.

Figura 65. Vista lateral de la tarjeta prototipo



Fuente: elaboración propia

Puertos de entrada/salida (E/S)

Las líneas de E/S de un PSoC® son flexibles, cada una de ellas puede ser configurada para trabajar con señales digitales o señales analógicas, y presentar una gran cantidad de modos de operación configurables por el usuario. El PSoC® de referencia CY8C5888LTI-LP097 incluido en kit CY8CKIT-059 tiene 48 líneas de E/S, como se indicó en el capítulo 2.

Las líneas de E/S salida se dividen en tres grupos: puertos de entrada salida de propósito general (GPIO), puerto de entrada salida especial (SIO) y pines de entrada salida USB (USBIO).

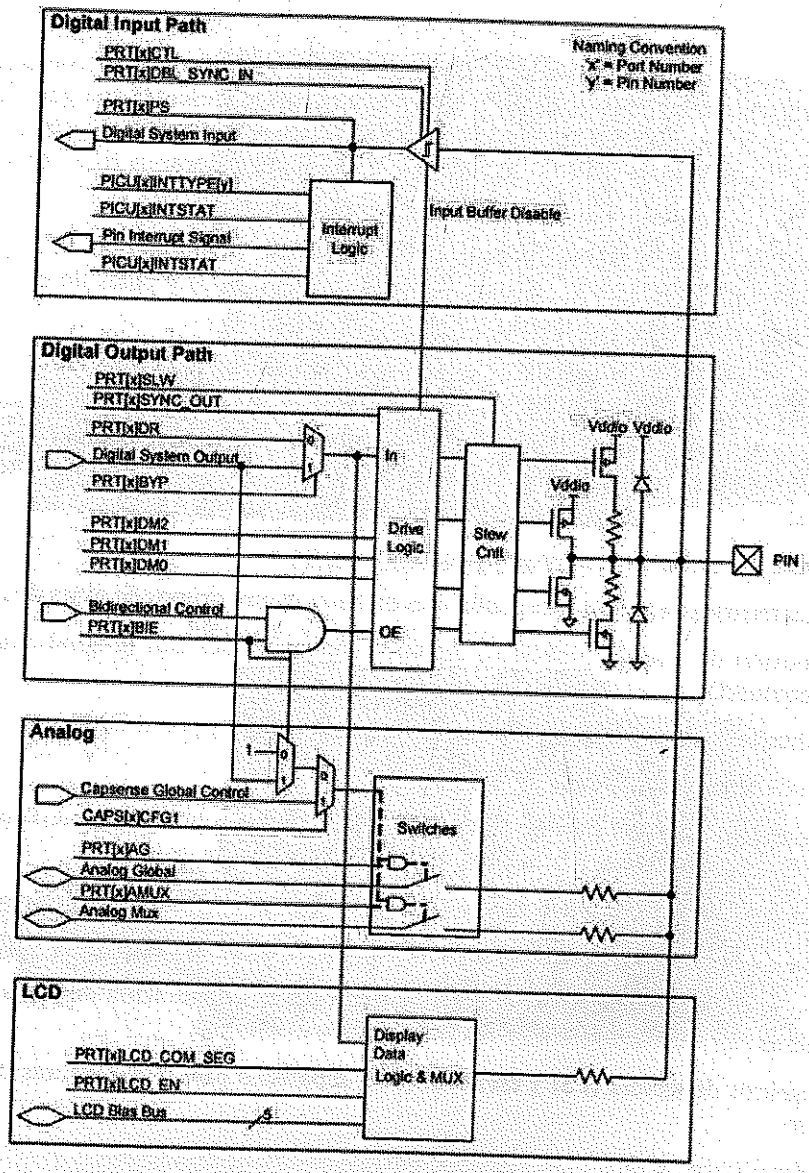
Puertos de E/S de propósito general (GPIO)

En la figura 66 puede verse el diagrama de bloques de un pin de un puerto de E/S de propósito general (GPIO). Las características para este tipo de puertos son las siguientes:

- Estado inicial del puerto programable por el usuario.
- Características y voltaje independientes en hasta cuatro grupos de pines de E/S.
- Conexión de pines a periféricos digitales por medio del DSI (Digital System Interconect).
- Utilizados como entradas, salidas o ambos por la CPU y por el DMA.
- 8 tipos de drives.
- Cada pin puede generar de manea independiente una solicitud de interrupción configurada por flanco de bajada, de subida o cualquiera de los dos.
- Vector de interrupción dedicado a cada pin o puerto GPIO.
- Control de slew rate para salidas digitales.
- Acceso a registros de configuración y control para configuración de pines o de puerto.
- Registros de lectura (PS) y escritura (DR) separados para evitar errores de acceso
- Drivers para LCD *
- Uso del módulo CapSense *

- Capacidad de entrada o salida analógica *
- Soporte con interface JTAG *
- Capacidad de corriente de 8mA en modo sink y 4mA en modo source *
- Voltaje de operación bajo desde 1,71v *

Figura 66. Diagrama de bloques de un pin GPIO



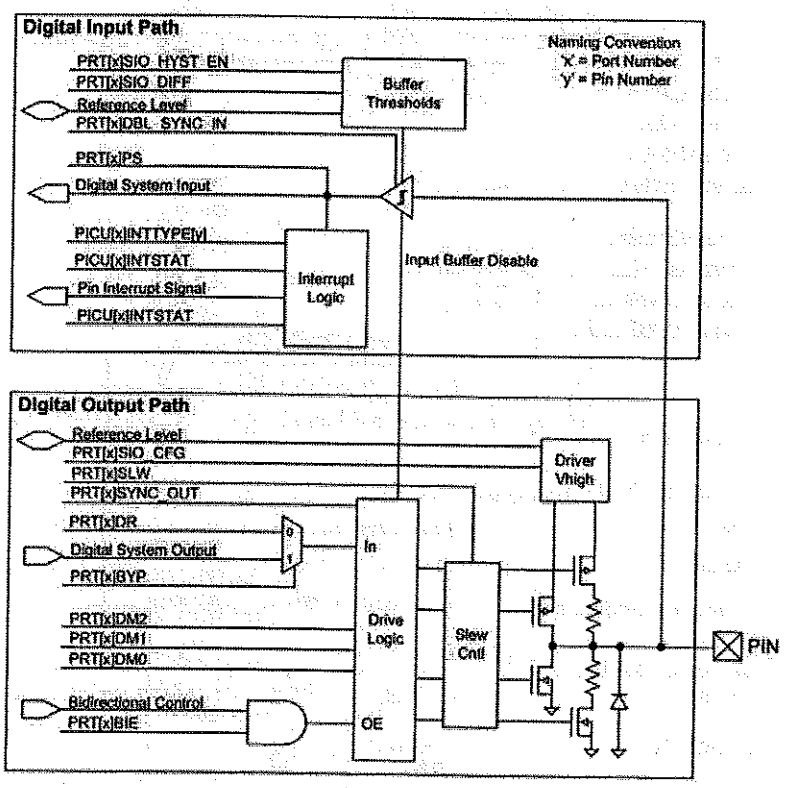
Fuente: [5, p. 33].

Puertos de E/S especiales (SIO)

En la figura 67 puede verse el diagrama de bloques de un pin de un puerto de E/S especial (SIO). Las características mostradas en el aparte anterior para los GPIO (excepto las marcadas con un *) aplican también para los SIO. Además de estas características para este tipo de puertos se tienen las siguientes:

- Driver superior al manejado por los GPIO.
- Capacidad de corriente de 25mA en modo sink y 4mA en modo source.
- Acepta valores de 5v sin importar el valor de VDD.
- No opera como entrada analógica, tampoco soporta CapSense o driver LCD.
- Tolera sobre voltajes a 5,5v.

Figura 67. Diagrama de bloques de un pin SIO



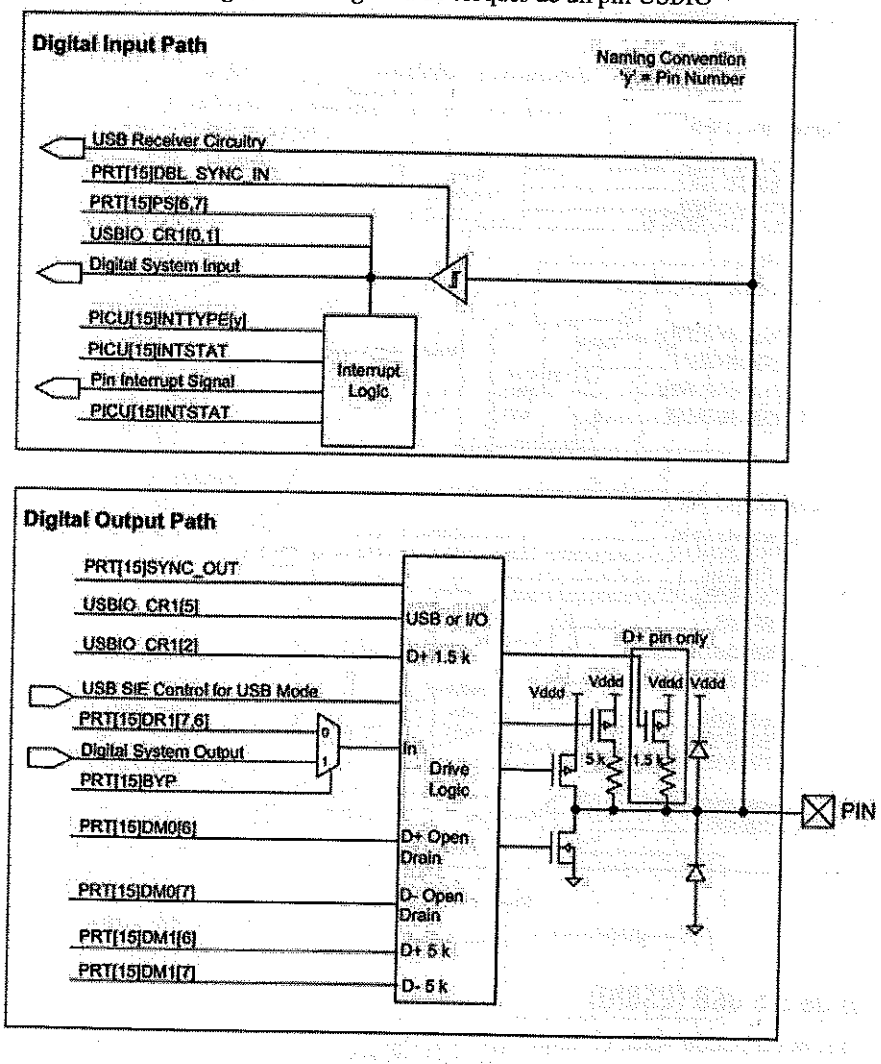
Fuente: [5, p. 34].

Puerto de E/S USB (USBIO)

En la figura 68 puede verse el diagrama de bloques de un pin del puerto de E/S USB (USBIO). Las características para este puerto son las siguientes:

- Soportan USB 2,0 full speed.
- Corriente 25 mA en modo sink y 24 mA source operando como líneas de E/S generales.
- Utilizados como entradas, salidas o ambos por la CPU y por el DMA.
- Utilizados como entradas, salidas o ambos por periféricos digitales.
- Salida digital tipo CMOS.
- Cada pin puede generar de manea independiente una solicitud de interrupción configurada por flanco de bajada, de subida o cualquiera de los dos.

Figura 68. Diagrama de bloques de un pin USBIO

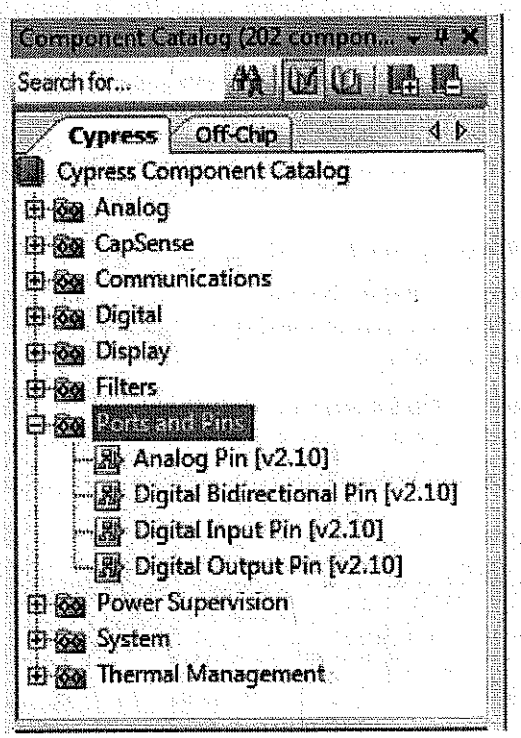


Fuente: [5, p. 34].

Uso de los puertos de E/S con el PSoC Creator

Las líneas de E/S son manejadas por medio de un bloque en el software de desarrollo PSoC Creator, denominado cy_pins_v2_1. Para esto, el usuario selecciona la carpeta Port and Pins del catálogo de componentes (figura 69), allí se tienen cuatro opciones de acuerdo con el uso que le vaya a dar al pin, pines o puerto: Analog Pin (pin analógico), Digital Bidireccional Pin (pin digital bidireccional), Digital Input Pin (pin digital de entrada) y Digital Output Pin (pin digital de salida), en cada caso se muestra la versión del bloque (para lo mostrado en la figura 69 es la versión 2.10). Dado que periódicamente Cypress puede actualizar este y otros bloques, la actualización puede consistir en agregar alguna función al bloque o mejorar algún parámetro del mismo.

Figura 69. Selección de un pin de E/S

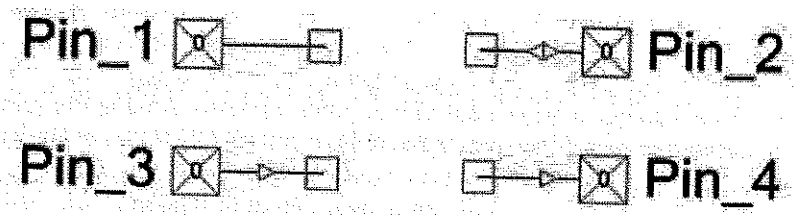


Fuente: elaboración propia.

En la figura 70 se muestran las cuatro posibilidades luego de seleccionar un pin. El PSoC Creator por defecto nombra los pines con el formato: Pin_x, siendo x un valor numérico que inicia desde uno y va hasta el último valor posible de acuerdo con la cantidad de pines incluidos en el diseño.

Los símbolos mostrados en la figura 70 tienen el siguiente significado: el Pin_1 corresponde a un pin analógico (puede ser de entrada o de salida), el Pin_2 a un pin digital bidireccional, el Pin_3 a uno digital de entrada y el Pin_4 a uno digital de salida.

Figura 70. Tipo de puertos de E/S soportado por el PSoC Creator



Fuente: elaboración propia.

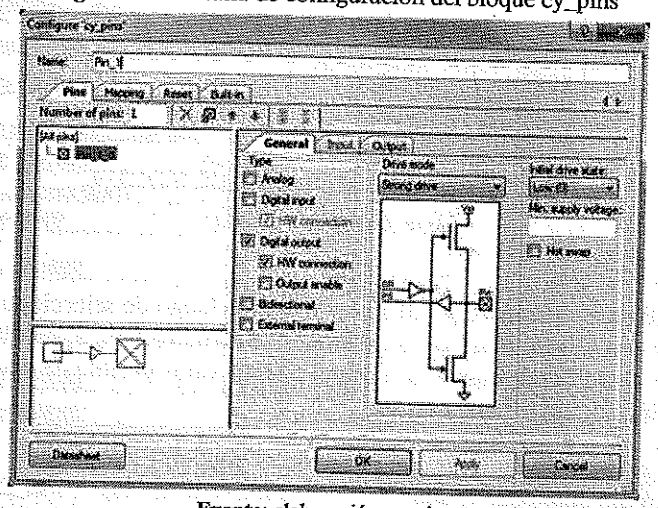
Configuración

Cada pin o puerto luego de ser seleccionado permite la configuración de sus parámetros (el tipo de pin y el sentido de flujo de datos esta preconfigurado). Así que pueden ser utilizados por ejemplo para conectarse con potenciómetros, LED, pulsadores, periféricos tanto internos como externos, sensores, etc.

En la figura 71 puede verse la ventana de configuración del bloque cy_pins que se activa al hacer doble clic sobre el símbolo. En esta ventana el usuario puede ver o configurar:

- El nombre del bloque (Name).
- Cantidad de pines del bloque (Number of pins) (por defecto este valor es uno), que puede aumentarse hasta 8; de esta forma, se maneja como un grupo de pines o un puerto de 8 bits.
- Modificar el alias del pin (el nombre dado en la figura 71 es Pin_1_0); este alias es la identificación del pin cuando se requiere acceso a él en formato de bit.
- Abrir el manual del bloque cy_pins como un datasheet en formato pdf.

Figura 71. Ventana de configuración del bloque cy_pins

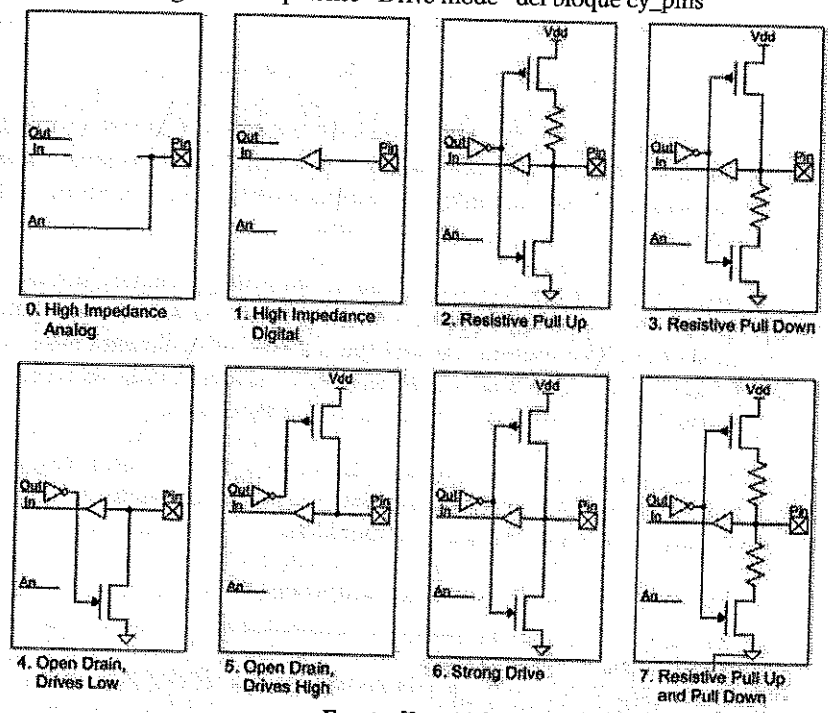


Fuente: elaboración propia.

En la pestaña "General" el usuario puede configurar el tipo de pin (Type) entre cinco valores posibles, los cuatro primeros corresponden a pin analógico, digital de entrada, digital de salida o bidireccional; la opción cinco permite extender el pin para conectar elementos externos (solo de manera gráfica) disponibles en la pestaña "Off-Chip" de la ventana del "Catálogo de componentes". La opción "HW Connection" ("Conexión con Hardware interno"), disponible solo para entrada o salida digital, determina si el bloque tendrá comunicación con el exterior (podrá ser utilizado con las funciones API del bloque) o con periféricos internos del PSoC® (opción por defecto). En este caso el bloque genera señales físicas utilizadas por los periféricos conectados al bloque.

La opción "Drive mode" de la misma pestaña permite seleccionar entre ocho posibles configuraciones el tipo de drive utilizado por el pin (figura 72). Dependiendo el tipo de pin seleccionado se tiene un valor inicial por defecto así:

Figura 72. Opciones "Drive mode" del bloque cy_pins

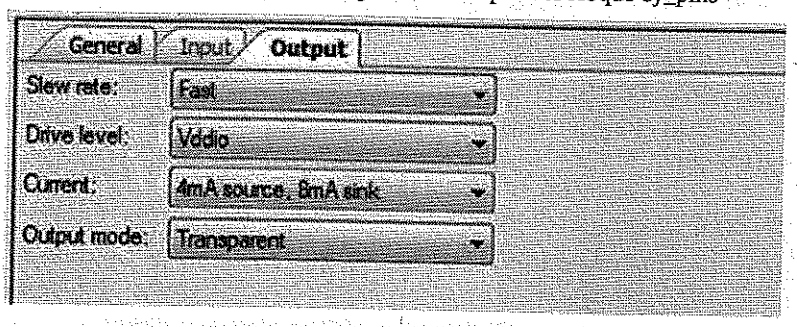


Fuente: [2, p. 147].

- Entrada digital valor inicial: High Impedance Digital (Alta Impedancia Digital).
- Pin analógico valor inicial: High Impedance Analog (Alta Impedancia Analógica).
- Pin Bidireccional valor inicial: Open Drain, Drives Low.
- Salida digital valor inicial: Strong drive.

La pestaña "Output" mostrada en la figura 73, que está activa únicamente para la configuración de salida digital o pin bidireccional, permite configurar cuatro parámetros: Slew rate, Drive level, Current y Output mode.

Figura 73. Pestaña de configuración Output del bloque cy_pins



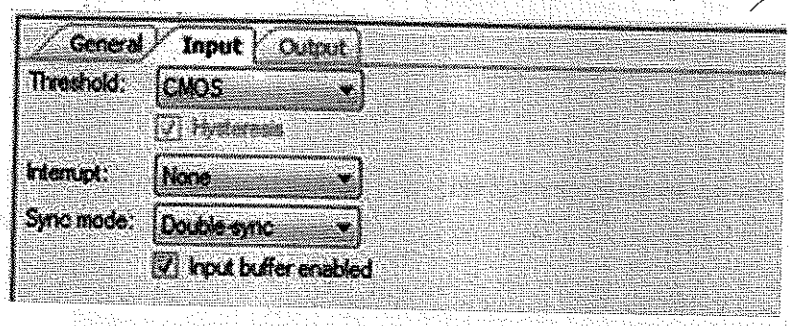
Fuente: elaboración propia.

El parámetro slew rate determina la velocidad de respuesta del pin.; se puede seleccionar entre dos valores posibles Slow y Fast (valor por defecto). El primero es utilizado para aplicaciones que requieran valores de respuesta menores a 1MHz; el segundo, para respuestas mayores a esta velocidad. El parámetro Drive level selecciona la fuente de alimentación para el pin que puede ser Vref (para pines SIO) o Vddio (valor por defecto).

Con el parámetro Current el usuario selecciona el valor de corriente de salida del pin, estos pueden ser: 4mA en modo source y 8mA en modo sink (valor por defecto) o 4mA en modo source y 25mA en modo sink (para pines SIO). Finalmente el parámetro Output mode reduce la inclinación de señales de salida de alta velocidad, los valores posibles son Transparent (valor por defecto) y Single-Sync.

Para configurar una entrada digital o un pin bidireccional se tiene otra pestaña denominada "Input" (figura 74), que permite realizar cambios en cuatro parámetros: Threshold, Interrupt, Sync mode e Input buffer enabled.

Figura 74. Pestaña de configuración Input del bloque cy_pins



Fuente: elaboración propia.

El valor del voltaje de entrada soportado por el pin puede ser modificado. Para esto se hace uso del parámetro Thereshold que permite siete opciones diferentes: CMOS (valor por defecto), CMOS o LVTTL, LVTTL, 0.4xVddio (para pines SIO), 0.5xVddio (para pines SIO), 0.5xVref (para pines SIO) y Vref (para pines SIO).

El parámetro Interrupt permite habilitar la solicitud de interrupciones del bloque y además el tipo de interrupción generada. Los valores posibles para este parámetro son: None (valor por defecto), Rising Edge (flanco de subida), Falling Edge (flanco de bajada) y Both Edges (ambos flancos).

Con el parámetro Sync mode se puede generar una señal de sincronización (Doublesync) para que los pines se sincronicen con todas las señales internas del PSoC®. Aunque esta es la forma ideal de operación de los pines, se tiene la opción de deshabilitar esta función (Transparent).

Finalmente, el parámetro Input buffer enabled permite habilitar (valor por defecto) o deshabilitar un buffer digital de entrada. Este buffer es necesario para la lectura del pin o para utilizar el nivel lógico presente en este.

Rutinas utilizadas para el manejo de las líneas de E/S

El PSoC Creator tiene una serie de rutinas desarrolladas por Cypress que permiten el acceso por software a los bloques utilizados en la aplicación, denominadas de forma general como Aplication Programming Interface (API). Estas rutinas se incluyen automáticamente en el proyecto al momento de generarlo, el usuario simplemente debe hacer el llamado de la función teniendo en cuenta los parámetros utilizados por esta.

Para el caso particular de las líneas de E/S se tiene la posibilidad de manejarlas con dos tipos de funciones, pin por pin, es decir, como una línea de E/S de un bit, o como un puerto con el que se interactúa como un valor de 1 a 8 bits dependiendo de la cantidad de líneas configuradas.

Para el caso en el que se manejan pin a pin se tiene una serie de funciones globales, es decir, operan independientes del bloque utilizado: El usuario puede encontrar estás funciones en el Datasheet denominado cy_boot_v5_0.pdf en la carpeta System Reference Guides, a la cual puede tenerse acceso de forma directa seleccionando la pestaña Datasheets en el PSoC Creator (figura 75).

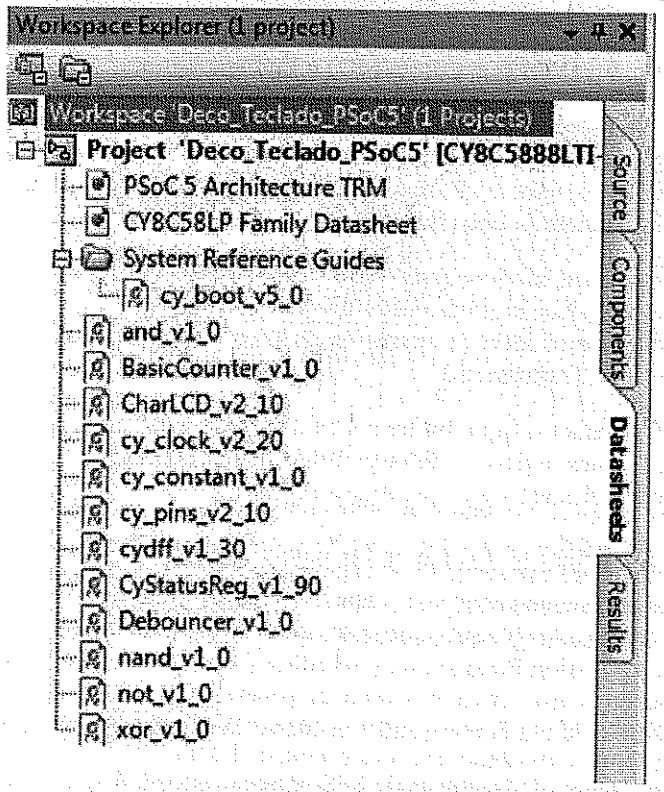
Las funciones que pueden ser utilizadas para acceso a pines son:

- CyPins_ReadPin():

  Lee el valor actual de un pin del PSoC®. Se utiliza como parámetro de entrada el alias del pin que se va a leer. La función retorna dos posibles valores: '0' que equivale a un nivel bajo en el pin y 'No-0' que equivale a un nivel lógico alto. Por ejemplo, para toma una decisión basada en el estado de un pin se podría utilizar la siguiente línea de código:

```
while (CyPins_ReadPin(LED_0)!=0);
```

Figura 75. Pestaña de acceso a datasheets de los bloques utilizados en la aplicación



Fuente: elaboración propia.

- CyPins_SetPin():

  Escribe un '1' en un pin del PSoC®. Se utiliza como parámetro de entrada el alias del pin que se va a escribir. Por ejemplo, para hacer '1' el pin con el alias LED_0 la sintaxis sería la siguiente:

  ```
  CyPins_SetPin(LED_0);
  ```

  CyPins_ClearPin():

  Escribe un '0' en un pin del PSoC®. Se utiliza como parámetro de entrada el alias del pin que se va a escribir. Por ejemplo, para hacer '0' el pin con el alias LED_0 la sintaxis sería la siguiente:

  ```
  CyPins_ClearPin(LED_0);
  ```

- CyPins_SetPinDriveMode():

  Permite seleccionar el tipo de drive utilizado por el pin. Entre las diversas posibilidades mostradas en la figura 72, para cada caso la función maneja una constante predefinida. Cada una de ellas es mostrada en la tabla 10.

Nombre de la constante	Valor de la constante	Drive
CY_PINS_DM_ALG_HIZ	0	Analog HiZ
CY_PINS_DM_DIG_HIZ	1	Digital HiZ
CY_PINS_DM_RES_UP	2	Resistive pull up
CY_PINS_DM__RES_ DWN	3	Resistive pull down
CY_PINS_DM_OD_LO	4	Open drain – drive low
CY_PINS_DM_OD_HI	5	Open drain – drive high
CY_PINS_DM_STRONG	6	Strong CMOS Output
CY_PINS_DM__RES_ UPDWN	7	Resistive pull up/down

Tabla 10. Constantes para la sección del Drive mode
Fuente: [5].

Por ejemplo, para configurar un pin de salida con el drive tipo Strong CMOS Output la sintaxis utilizada sería:

```
CyPins_SetPinDriveMode(LED_0,CY_PINS_DM_STRONG);
```

- CyPins_ReadPinDriveMode():

Lee el tipo de drive configurado en un pin específico del PSoC®. Se utiliza como parámetro de entrada el alias del pin por interrogar, la función retorna una constante con los mismos valores de la función anterior incluidas en la tabla 10. Si por ejemplo quiere conocerse el tipo de drive configurado en el pin con el alias LED_0, cuyo valor sería almacenado en la variable drive, la sintaxis sería la siguiente

```
drive=CyPins_ReadPinDriveMode(LED_0);
```

Para el caso de las funciones propias del bloque cy_pins, hay que tener en cuenta que la primera parte del nombre de la función debe ser reemplazada por el nombre que el usuario le haya asignado al bloque. De estas funciones se puede tener información en el manual correspondiente del bloque (cy_pins_v2_10.pdf), en la sección titulada Aplication Programming Interface. Las funciones que pueden ser utilizadas para acceso a bloques de pines son:

- Pin_Read():

Lee el valor actual de todos los pines asociados al bloque cy_pins utilizado. La función no utiliza ningún parámetro de entrada y retorna un valor entero de 8 bits sin signo justificado a la derecha. Por ejemplo, para leer el estado de un bloque de entrada de 4 bits denominado Sensores y enviarlo a la variable in se puede escribir la siguiente línea de código:

```
in=Sensores_Read();
```

- Pin_Write():

Escribe un valor de 8 bits justificado a la derecha en todos los pines asociados al bloque cy_pins utilizado. La función utiliza como parámetro de entrada un número entero de 8 bits sin signo. Por ejemplo, para hacer que todos los pines de un bloque de salida de 8 bits denominado Columnas se hagan '1' se puede escribir la siguiente línea de código:

```
Columnas_Write(0xFF);
```

- Pin_SetDrive_Mode():

Permite seleccionar el tipo de drive utilizado por cada pin del bloque cy_pins. Igual que ocurre con la función CyPins_SetPinDriveMode(), vista antes. Para cada caso la función maneja una constante predefinida, que incluye el alias del pin por configurar (ver tabla 11).

Nombre de la constante	Drive
Pin_1_DM_STRONG	Strong Drive
Pin_1_DM_OD_HI	Open drain – drive high
Pin_1_DM_OD_LO	Open drain – drive low
Pin_1_DM_RES_UP	Resistive pull up
Pin_1_DM_RES_DWN	Resistive pull down
Pin_1_DM_RES_UPDWN	Resistive pull up/down
Pin_1_DM_DIG_HIZ	Digital HiZ
Pin_1_DM_ALG_HIZ	Analog HiZ

Tabla 11. Constantes de la función Pin_SetDrive_Mode() para la sección del Drive mode
Fuente: elaboración propia.

Por ejemplo, para configurar un pin de salida con el drive tipo Strong Drive del bloque cy_pins denominado LED, la sintaxis utilizada sería:

```
LED_SetDriveMode(LED_0_DM_STRONG)
```

- Pin_ClearInterrupt():

Retorna el valor del registro de estado interrupciones del puerto asociado al bloque cy_pins, y luego borra este registro. Si se quiere, por ejemplo, saber el valor del registro estado de las interrupciones del bloque denominado Columnas la sintaxis por utilizar sería:

```
estado=Columna_ClearInterrupt();
```

Ejemplos de aplicación de los puertos de E/S

Diseño de un decodificador de teclado matricial 4x4

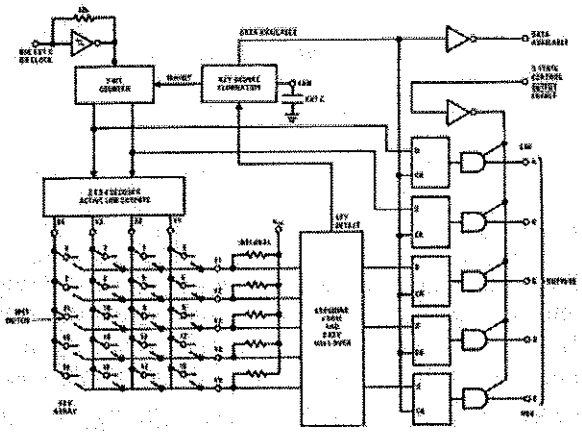
El uso de teclados es muy común en aplicaciones donde es necesaria la escritura o la selección de información, como datafonos, cajeros electrónicos, sistemas de seguridad con clave de acceso, configuración de parámetros en equipos, etc. En el siguiente ejemplo se muestra el diseño de un decodificador de teclado matricial de 4x4 teclas por hardware, aprovechando las bondades de los bloques digitales del PSoC®5LP. De esta manera no se hace uso de los recursos de la CPU del PSoC®, dado que todo el proceso se realiza únicamente por hardware por medio de los bloques digitales universales (UDB).

Solución

Para decodificar un teclado matricial, el método convencional es realizar el barrido de las columnas del teclado por medio de la rotación de '0' lógico a una frecuencia adecuada que depende de qué tan "sensible" se necesite la operación del teclado, utilizando para esto una combinación de contador y decodificador. Las filas del teclado son conectadas a un codificador que tiene sus entradas conectadas a Vcc por medio de resistencias de pull up; de esta manera se asegura un '1' lógico en cada una de las entradas. Este codificador detecta cuándo una entrada cambia a '0'; al ocurrir esto se genera una señal que detiene temporalmente al contador, manteniendo durante un breve momento la combinación fila columna, con lo que puede generarse un código binario equivalente a la combinación detectada.

Se plantea la solución basada en el decodificador comercial 74C922 [7] fabricado por Fairchild Semiconductor y por National Semiconductor, que puede decodificar un teclado matricial de hasta 4x4 teclas (16 teclas). En la figura 76 puede verse el diagrama circuital del decodificador, la idea del diseño propuesto es tomar las partes básicas del decodificador comercial e implementarlas realizando las adaptaciones necesarias de acuerdo con las funcionalidades del PSoC.

Figura 76. Diagrama circuital del decodificador 74C922



Fuente: [7, p. 3].

Se toman para el diseño con sus respectivas adecuaciones el bloque decodificador de 2 a 4 con salidas activas en bajo, que tiene como entrada un contador ascendente de 2 bits. Este conjunto es utilizado para el manejo de las columnas de teclado, el codificador de 4 a 2 para el manejo de las filas del teclado, además 4 flip flops tipo D para generar y almacenar el código binario de salida del decodificador, el bloque que elimina el rebote generado por el contacto mecánico de las teclas y finalmente la señal de dato disponible o tecla oprimida que puede ser utilizada para generar interrupciones. La funcionalidad de poner en alta impedancia la salida del decodificador de teclado no fue implementada.

- Bloque decodificador de 2 a 4:

  Se hace necesario el diseño de este bloque con compuertas lógicas ya que en el catálogo de componentes no está disponible. Para esto se tiene en cuenta la tabla de verdad de un decodificador convencional de dos entradas y cuatro salidas con salida activa en bajo (tabla 12).

A	B	O_0	O_1	O_2	O_3
0	0	0	1	1	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	1	1	1	0

Tabla 12. Tabla de verdad de un decodificador de 2 a 4
Fuente: elaboración propia.

De la tabla 12 se deducen por álgebra de Boole las ecuaciones (1) a (4), que serán las que se implementen en el diseño de este bloque, para obtener así el diagrama circuital mostrado en la figura 77:

$$O_0 = \overline{\overline{A}\bullet\overline{B}} \rightarrow (1)$$

$$O_1 = \overline{\overline{A}\bullet B} \rightarrow (2)$$

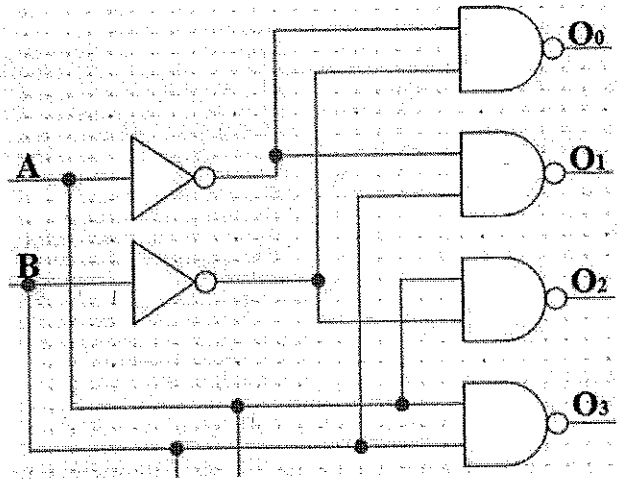
$$O_2 = \overline{A\bullet\overline{B}} \rightarrow (3)$$

$$O_3 = \overline{A\bullet B} \rightarrow (4)$$

- Bloque codificador 4 a 2:

  Igual que ocurre con el bloque decodificador es necesario el diseño de este bloque con compuertas lógicas, ya que en el catálogo de componentes tampoco está disponible. Para esto se tiene en cuenta la tabla de verdad de un codificador convencional de cuatro entradas y dos salidas que puede apreciarse en la tabla 13; a los estados restantes no incluidos en la tabla se les asignó el valor de salida 00.

Figura 77. Diagrama circuital decodificador 2 a 4



Fuente: elaboración propia.

A	B	C	D	X	Y
0	1	1	1	0	0
1	0	1	1	0	1
1	1	0	1	1	0
1	1	1	0	1	1

Tabla 13. Tabla de verdad de un codificador de 4 a 2
Fuente: elaboración propia.

Con los datos de la tabla 13 se deducen las ecuaciones (5) y (6) que serán utilizadas en la implementación de este bloque; de esta manera se obtiene el diagrama circuital mostrado en la figura 78:

$$X = A.B.(C \oplus D) \rightarrow (5)$$

$$Y = A.C.(B \oplus D) \rightarrow (6)$$

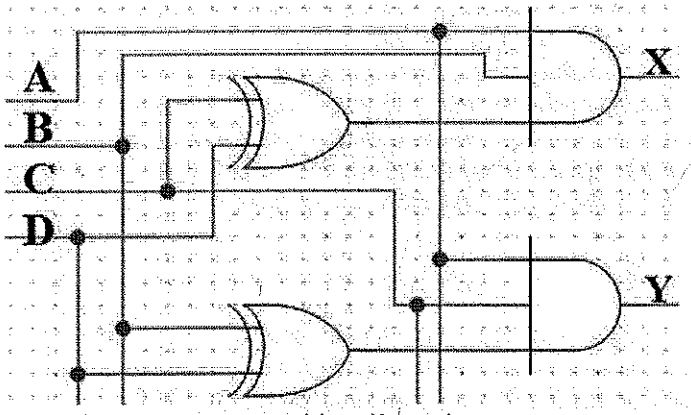
- Bloque para eliminar rebote mecánico:

El rebote generado por el contacto mecánico de las teclas es eliminado por medio de un bloque incluido en el catálogo de componentes denominado Debouncer, cuya función primordial de acuerdo con lo indicado en su datasheet es eliminar oscilaciones no deseadas en una señal digital.

- Contador ascendente de dos bits:

Este contador tiene como función realizar el barrido de las columnas del teclado

Figura 78. Diagrama circuital codificador 4 a 2



Fuente: elaboración propia.

realizando un conteo ascendente cíclico de cero a tres. El contador genera el dato de entrada que requiere el bloque decodificador de 2 a 4. La frecuencia de operación del reloj del contador depende de la "sensibilidad" que se le quiera dar al teclado. El contador es detenido por la señal generada por el bloque para eliminar rebote (Debouncer), para que el arreglo de flip flops tipo D tenga tiempo de capturar y almacenar el dato que finalmente mostrará el código binario equivalente de la tecla oprimida.

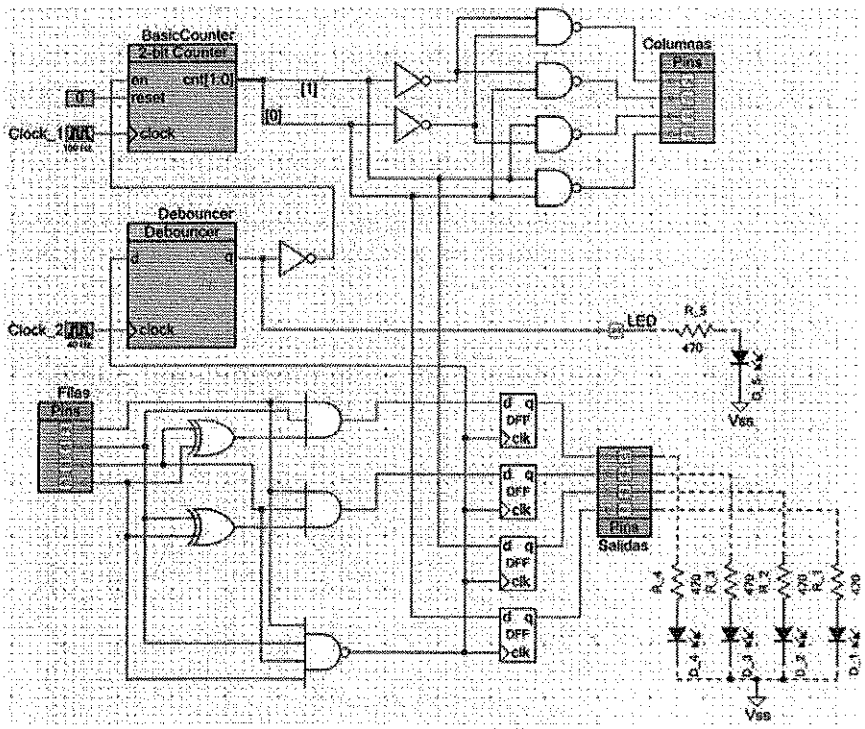
En la figura 79 puede verse el diagrama circuital con el diseño completo del decodificador. En este, además de lo explicado anteriormente, se incluyen tres bloques de E/S salida denominados "Columnas", "Filas" y "Salidas", que permiten conectar físicamente las columnas y las filas del teclado y además obtener el código binario de la tecla oprimida.

La señal de dato disponible o tecla oprimida está nombrada en el diagrama como LED; esta señal es un pulso que se mantiene en un nivel lógico alto mientras esté oprimida una tecla, y luego vuelve a nivel lógico bajo mientras el teclado esté en reposo, es decir, que ninguna tecla se haya oprimido.

Las funciones lógicas en general no requieren ningún tipo de configuración; para el diseño realizado únicamente debe modificarse la cantidad de entradas para aquellas compuertas que necesiten más de dos entradas. Esto puede realizarse haciendo doble clic sobre la compuerta seleccionado el parámetro NumTerminals y cambiando el valor.

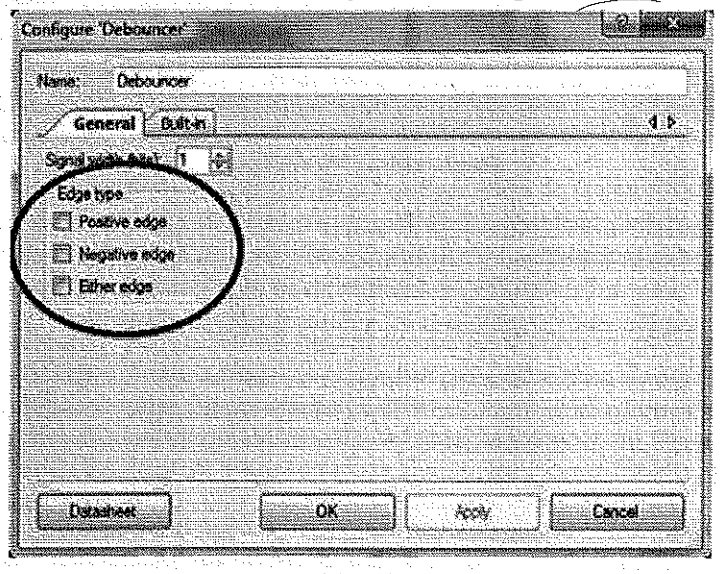
Al bloque Debouncer, que puede encontrarse en el catálogo de componentes en la carpeta Digital, subcarpeta Utility, se le deben modificar algunos parámetros, dado que por defecto tiene habilitadas algunas funcionalidades que no son útiles para este diseño en particular. En la ventana de configuración del bloque mostrada en la figura 80 se debe eliminar la selección de las opciones: Positive edge, Negative edge y Either edge del parámetro Edge type, el cual detecta un tipo de flanco particular, subida, bajada o ambos y genera la señal de salida temporizada de acuerdo con esta detección.

Figura 79. Diagrama final del decodificador de teclado matricial



Fuente: elaboración propia.

Figura 80. Ventana de configuración del bloque Debouncer

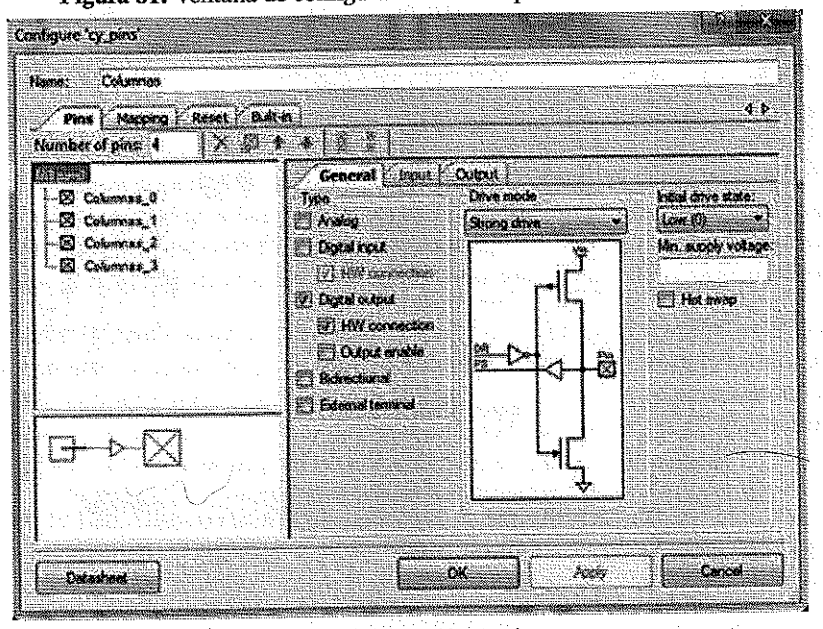


Fuente: elaboración propia.

Para el diseño propuesto el requisito es mantener la salida en alto mientras se esté oprimiendo una tecla, así que no son necesarias estas opciones.

Para el conjunto encargado de manejar las columnas del decodificador se utilizó un bloque contador (BasicCounter), el bloque decodificador de 2 a 4 (que se explicó anteriormente) y un bloque de líneas de E/S de cuatro bits (Pins) configuradas como salidas. Al bloque contador es necesario modificarle el parámetro Width (haciendo doble clic sobre el bloque) que determina el número de bits del contador. Por defecto este parámetro es cuatro y para el diseño debe modificarse a dos, además se debe seleccionar una frecuencia de operación para el contador. Para este caso se definió finalmente el valor de 100Hz.

Figura 81. Ventana de configuración del bloque de salida Columnas



Fuente: elaboración propia.

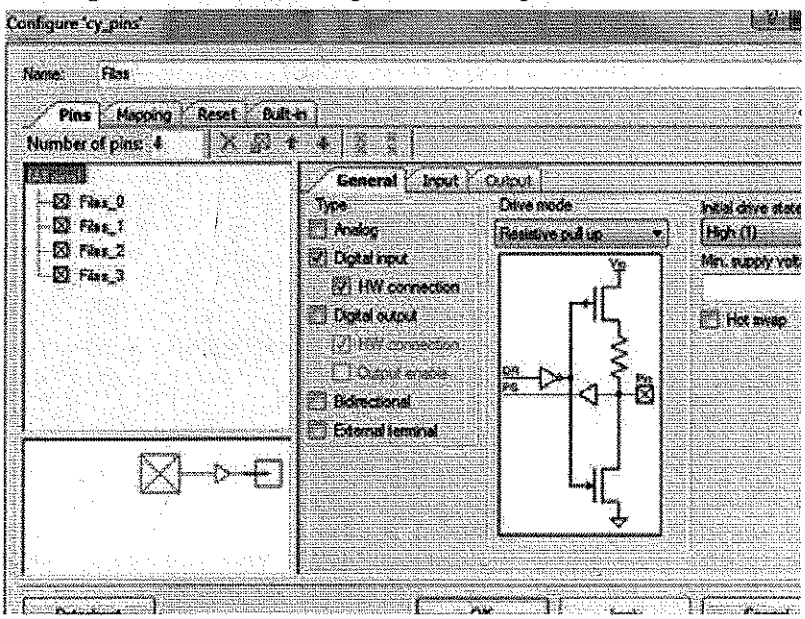
El bloque Pins llamado Columnas, utilizado para la conexión física de las columnas del teclado, debe ser modificado, los parámetros por modificar son: Name: Columna y Number of pins: 4. Los demás parámetros no se modifican; en la figura 81 puede verse la ventana de configuración con los valores finales configurados en el bloque.

Para la conexión de las filas se generó el conjunto conformado por un bloque de E/S de cuatro bits configurado como entrada, el bloque codificador de 4 a 2 (explicado anteriormente) y un arreglo de flip flops tipo D para almacenar el dato de la tecla decodificada.

En la figura 82 puede verse la configuración final del bloque Pins denominado "Filas". A este bloque se le modificaron para el diseño los parámetros: Name: Fila, Number of pins: 4, Drive mode: Resistive pull up. Este último parámetro asegura un

nivel lógico alto en la entrada del bloque, que es necesario para el correcto funcionamiento del decodificador.

Figura 82. Ventana de configuración del bloque de entrada "Filas"



Fuente: elaboración propia.

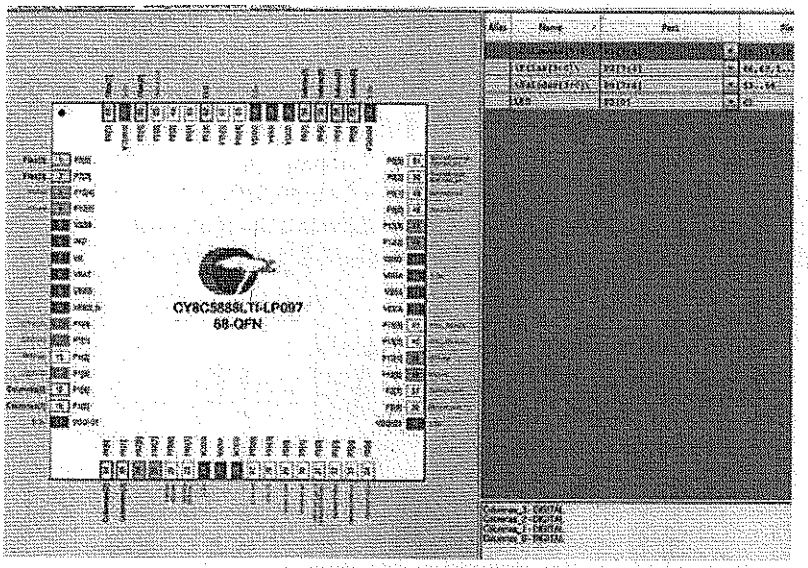
Finalmente para mostrar el resultado de la decodificación se utiliza un bloque de E/S de cuatro bits configurado como salida, denominado "Salidas", a este bloque se le modificaron los parámetros: Name: Salidas, Number of pins: 4 y se habilitó la opción External Terminal, que permite conectar del catálogo de componentes elementos externos al PSoC® denominados "Off-Chip". Para este caso se utilizaron cuatro resistencias de la carpeta de componentes Passive, cuatro LEDs de la carpeta Diodes y el símbolo de conexión a tierra Ground de la carpeta Power.

El segundo paso del proceso consiste en la configuración de la plantilla de los recursos del sistema (vista en el capítulo tres de este libro). Para este caso se deben configurar 4 bloques, todos correspondientes a líneas de E/S; estos bloques se nombraron en el diseño como "Columnas", "Filas", "Salidas" y "LED".

La tarea por realizar en este último paso es asociar los bloques cy_pins con los pines físicos del PSoC®. Como ya se indicó, cualquier pin puede ser utilizado para el proceso, en este caso se seleccionaron las siguientes opciones: Columnas P1[7:4], Filas P2[7:4], Salidas P0[7:4] y LED P2[0]. En la figura 83 puede verse la configuración final de los pines del PSoC®, y además gráficamente los pines utilizados en el circuito integrado.

Para este diseño no se hace uso de ninguna rutina de software; por lo tanto, con esto finalizaría el proceso de diseño. Para terminar se genera la aplicación y se progra-

Figura 83. Configuración de la plantilla de los recursos del sistema



Fuente: elaboración propia.

ma el PSoC®. Para la verificación del decodificador implementado se conectan al bloque Salidas cuatro LED de la tarjeta prototipo, al bloque LED indicador de tecla oprimida un LED y se conectan los bloques Filas y Columnas a las filas y columnas teclado para realizar las pruebas correspondientes.

Visualización de datos en display de 7 segmentos

Los displays de 7 segmentos son ampliamente utilizados para visualizar información en formato numérico, en aplicaciones como sistemas de generación de turnos ("turneros"), relojes digitales, visualización de tiempo y marcadores en eventos deportivos, contadores, voltímetros, tacómetros, etc. En el siguiente ejemplo se presenta el uso de un display de 7 segmentos decodificado directamente con el PSoC5®LP; así se elimina el uso de decodificadores externos, con lo que, entre otras cosas, se disminuyen costos y tiempos de diseño. El diseño incluirá además el uso del decodificador de teclado matricial implementado en el ejemplo anterior.

Solución

Para visualizar información en un display de 7 segmentos se requiere un decodificador que convierta el valor numérico en un equivalente que encienda o apague cada uno de los segmentos del display de acuerdo con el valor por decodificar. Para realizar este proceso en un sistema que incluya algún tipo de procesador y que además se exija su uso sin un decodificador, pueden tenerse múltiples opciones, como por ejemplo procesos de comparación para determinar la equivalencia del valor a decodificar, o dado que se está trabajando con información en formato de bits determinará el estado lógico de cada uno de ellos hasta llegar a un valor equivalente. Estas soluciones son válidas, pero en el primer caso genera tiempo de respuesta diferentes

ya que dependiendo del valor por decodificar se podrán necesitar más o menos comparaciones. El segundo método utiliza siempre el mismo tiempo, que dependerá del número de bits utilizados.

La solución planteada para este ejemplo se basa en el uso de tablas. Este método tiene la gran ventaja de que el tiempo de decodificación no depende del valor de entrada, pues este siempre es igual, además no se requiere ningún tipo de comparación. La tarea se resume en leer una tabla que contiene los datos de salida ya codificados, en este caso en formato de siete segmentos.

El primer paso del diseño es determinar los datos que serán incluidos en la tabla, dado que existe un estándar para nombrar cada uno de los segmentos de un display de siete segmentos. Se seguirá este estándar que utiliza las letras de la 'a' a la 'g' para esta tarea. Cada segmento será manejado por una línea de E/S del PSoC®, entonces se supondrá que la línea de E/S que maneje el segmento 'a' será la menos significativa y la que maneje el segmento 'g' la más significativa; esto implica que se utilizará un bloque cy_pin como un puerto de E/S de 7 bits.

En la tabla 14 se muestran cada uno de los valores decodificados que serán visualizados en el display de siete segmentos; estos valores corresponden a cada una de las teclas de teclado matricial de 4x4, es decir, los números del 0 al 9, las letras de la A a la D y los símbolos * y #. Para la tabla mostrada un valor lógico bajo indica segmento apagado y un valor lógico alto indica segmento encendido. Dependiendo del tipo de display utilizado en la aplicación, cátodo común o ánodo común, se utilizarán los valores indicados en la columna Hx o su complemento respectivamente.

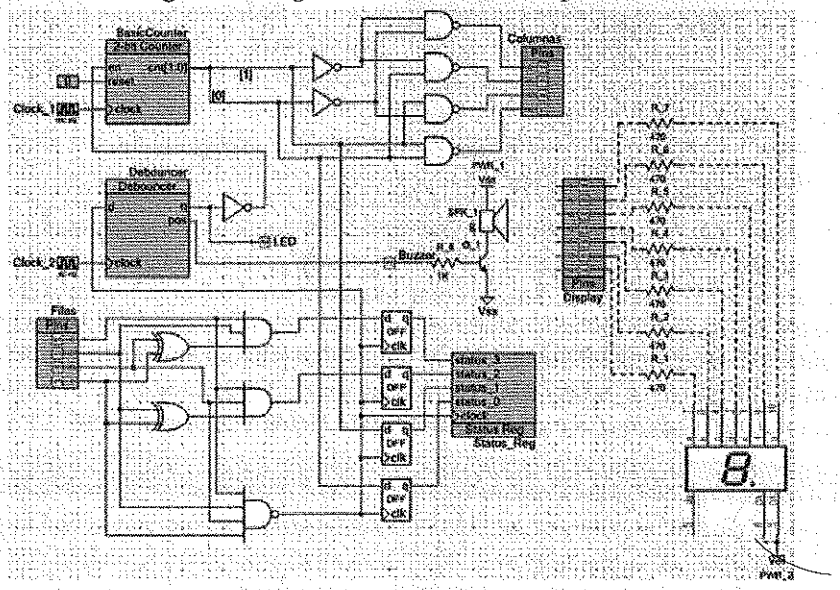
	g	f	e	d	c	b	a	Hx	
0	0	1	1	1	1	1	1	3F	
1	0	0	0	0	1	1	0	06	
2	1	0	1	1	0	1	1	5B	
3	1	0	0	1	1	1	1	4F	
4	1	1	0	0	1	1	0	66	
5	1	1	0	1	1	0	1	6D	
6	1	1	1	1	1	0	1	7D	
7	0	0	0	0	1	1	1	07	

	g	f	e	d	c	b	a	Hx	
8	1	1	1	1	1	1	1	7F	
9	1	1	0	1	1	1	1	6F	
A	1	1	1	0	1	1	1	77	
B	1	1	1	1	1	0	0	7C	
C	0	1	1	1	0	0	1	39	
D	1	0	1	1	1	1	0	5E	
*	1	1	1	0	0	0	0	70	
#	1	1	1	0	1	1	0	76	

Tabla 14. Valores decodificados para display de siete segmentos
Fuente: elaboración propia.

Con la tabla 14 y utilizando el diseño del decodificador de teclado —con algunas modificaciones— del ejemplo mostrado en el aparte "Diseño de un decodificador de teclado matricial 4x4", se plantea el segundo paso de la solución, que consiste básicamente en incluir un bloque de E/S para la conexión del display de 7 segmentos. Este bloque se encargará por medio de un algoritmo de software implementado en el PSoC® de realizar la decodificación basada en la información de la tabla 14 y de la trasferencia de los datos al display. El diseño final planteado puede verse en la figura 84, al que se le agrego un buzzer que sonará cada vez que se oprima una tecla, que no estaba incluido dentro de los requisitos del planteamiento del problema.

Figura 84. Diagrama final de la solución planteada



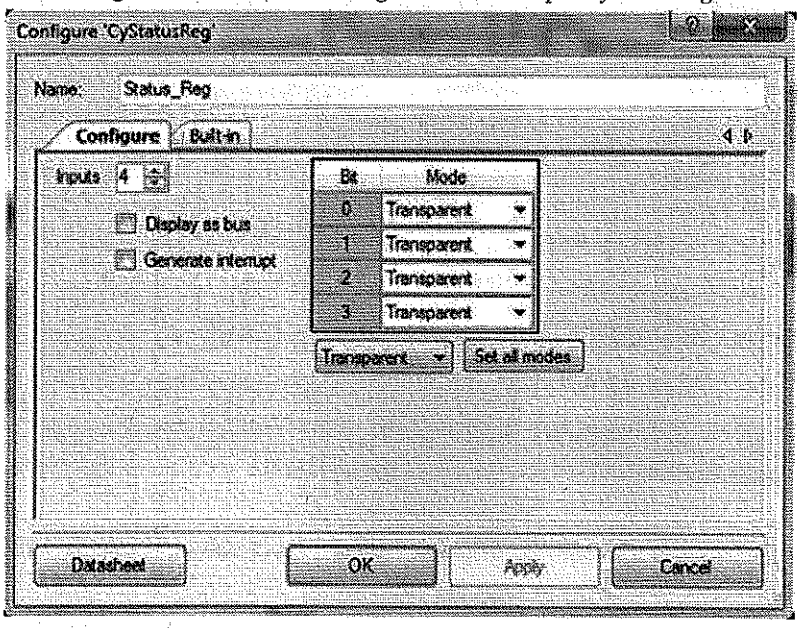
Fuente: elaboración propia.

Para el decodificador de teclado se realizan dos modificaciones respecto al diseño original. La primera está relacionada con la posibilidad de utilizar internamente el valor decodificado, es decir, que por medio de instrucciones de software el usuario pueda leer este valor; para esto se elimina el bloque cy_pins denominado Salidas y se reemplaza por el bloque Status Register, cuya función de acuerdo con lo indicado en su datasheet es la de permitir consultar el estado de señales digitales internas, que puede encontrarse en el catálogo de componentes en la carpeta Digital, subcarpeta Registers.

En la figura 85 puede verse la configuración final del bloque Status Register. A este bloque se le modificaron dos parámetros, Name: Status_Reg e Inputs: 4. Este último determina la cantidad de señales digitales que se quieren consultar; para este caso son cuatro dado que este valor corresponde a la cantidad de salidas generadas por el decodificador de teclado. Existen además dos modos de operación para el bloque (Mode): Transparent y Sticky, para el primer caso se realiza la lectura del estado de la

entrada y se mantiene hasta una nueva lectura, y en el segundo caso se almacena el estado sin importar si se realiza o no una lectura de este dato; al realizarse la lectura el valor almacenado se borra. Para la solución planteada se considera adecuado el modo por defecto Transparent.

Figura 85. Ventana de configuración del bloque CyStatusReg



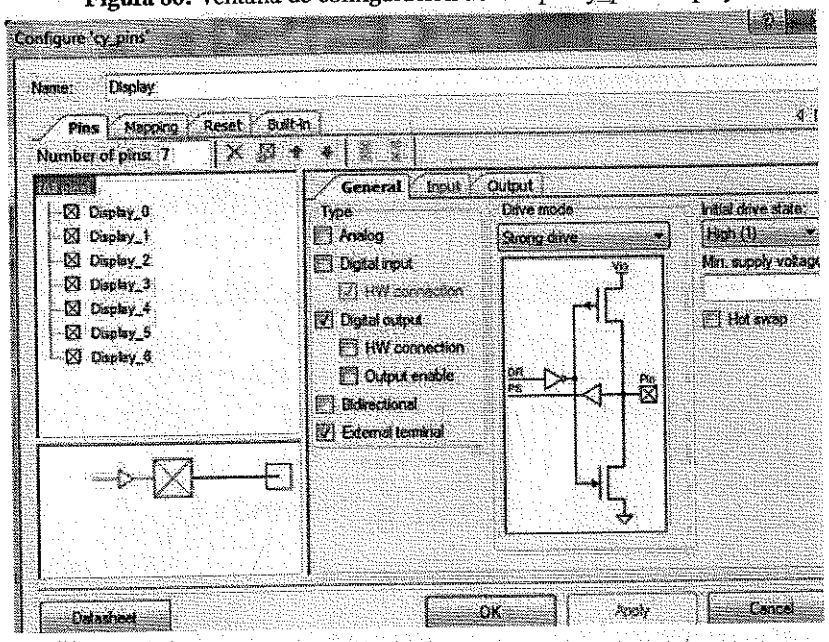
Fuente: elaboración propia.

La segunda modificación del decodificador de teclado está relacionada con el bloque Debouncer; esta modificación es opcional dado que está asociada a la tarea de manejar el buzzer que se mencionó anteriormente. La modificación consiste en habilitar la opción Positive edge del parámetro Edge type, que genera un pulso temporizado cada vez que se oprime una tecla. A esta señal se conecta un pin de E/S denominado Buzzer donde irá conectado el buzzer de la tarjeta de desarrollo utilizada para verificar la solución del ejemplo.

Finalmente, para visualizar el valor de la decodificación en el display de siete segmentos (se escogió un display de ánodo común); se utiliza un bloque de E/S de siete bits configurado como salida, denominado Display (figura 86). A este bloque se le modificaron los parámetros: Name: Display, Number of pins: 7, opción HW connection deshabilitada, Initial drive state: High (1) y se habilitó la opción External Terminal para conectar del catálogo de componentes Off-Chip siete resistencias y el símbolo de conexión a la fuente de alimentación Vdd de la carpeta Power.

La opción HW connection se deshabilita ya que el bloque será manejado por instrucciones de software dentro del algoritmo implementado en el PSoC®, no por señales de hardware, y el estado inicial de los pines es alto puesto que al iniciar el display

Figura 86. Ventana de configuración del bloque cy_pins Display



Fuente: elaboración propia.

estará apagado, y por el tipo de display utilizado en la solución este será el estado inicial adecuado.

Con el diseño anterior completo ahora se configura la plantilla de los recursos del sistema (ver figura 87). Para este caso se deben configurar 5 bloques, todos correspondientes a líneas de E/S; estos bloques se nombraron en el diseño como: Columnas, Filas, Display, Buzzer y LED.

Figura 87. Configuración de los recursos del sistema

Alias	Name	Port
	\Columnas[3:0]\	P1[7:4]
	\Display[6:0]\	P3[6:0]
	\Filas[3:0]\	P2[7:4]
	Buzzer	P2[1]
	LED	P2[0]

Fuente: elaboración propia.

El último paso para finalizar el diseño, luego de generada la aplicación, es la escritura del código que se encargará de la lectura del dato generado por el teclado y su posterior decodificación para ser enviada a la display de siete segmentos.

El algoritmo implementado es sencillo, ya que aprovecha las múltiples funciones que el PSoC Designer incluye. Básicamente realiza tres tareas, la primera es estar pendiente del momento cuando se oprime una tecla; esta información es generada por el decodificador de teclado por medio de la señal LED. Luego de detectar una tecla oprimida el algoritmo lee el valor almacenado en el bloque Status Register, este valor será un numero entre cero y 15 que dependerá de la tecla oprimida que se utiliza luego para leer la tabla y enviarlos así al puerto de E/S denominado Display, luego se espera hasta que la tecla deje de oprimirse, para repetir el ciclo completo constantemente.

A continuación puede verse el código final escrito para la solución del problema:

Ejemplo 5-2:

```
/* ==========================================================
 *
 * Programa para visualizar en un display de siete segmentos el valor
 * de la tecla oprimida en un teclado matricial de 4 x 4 teclas,
 * la decodificación del display se realiza por software.
 *
 * ==========================================================
*/
#include <project.h>
/* Definición de variables */
uint8 tecla;
uint8 Tabla[]={
06,0x5B,0x4F,0x77,0x66,0x6D,0x7D,0x7C,07,0x7F,0x6F,0x39,0x70,0x3F
0x76,0x5E
};
int main()
{
for(;;)
  {
  while (LED_Read()==0); //Espera hasta que se oprima una tecla
  tecla=Status_Reg_Read(); //Lee valor de la tecla oprimida
  Display_Write(~Tabla[tecla]); //Envía valor de la tabla al display
```

```
        while (CyPins_ReadPin(LED_0)!=0); //Espera hasta que se suelte tecla
      }
    }
```

La verificación del diseño y del algoritmo se realizará luego de programar el PSoC®, conectando el teclado, el display de siete segmentos y el buzzer, al kit CY8CKIT-059.

Manejo de tiempo con rutinas por software

La temporización, o la generación de intervalos de tiempo específicos en una aplicación es una tarea muy utilizada en sistemas embebidos, por ejemplo, el control de tiempo de diversas acciones, como captura de información cada determinado tiempo, conteo de eventos externos, etc. En muchas ocasiones los elementos de hardware del dispositivo utilizado, que permiten realizar esta tarea, están ocupados o se requieren para otras tareas. En ese caso se hace necesario el uso de instrucciones del procesador para generar estos intervalos de tiempo, ya que el tiempo de ejecución de las mismas es un parámetro que puede conocer quién diseña la aplicación.

- Se pide para este ejemplo diseñar un programa que haga oscilar uno de los pines del PSoC®5LP a una frecuencia de 1Hz, haciendo uso de retardos de tiempo por software.

Solución

Típicamente cuando se necesita solucionar un problema de temporización por software, el diseñador debe entrar a conocer en detalle el procesador con el que va a trabajar, para determinar el tiempo que tarda en ejecutarse una instrucción y así al unir varias también el tiempo total que tardan en ejecutarse. En el caso del PSoC® esto no es necesario ya que el PSoC Creator incluye dentro de sus librerías una serie de funciones que permiten manejar de manera directa y precisa este proceso, sin tener que conocer el detalle de los tiempos de ejecución del código.

Estas funciones son API detalladas en el datasheet Cy_Boot_5_0.pdf, del cual ya se ha hablado con anterioridad. Las funciones que pueden ser utilizadas para manejo de tiempo son:

- CyDelay():

  Genera un retardo en unidades de milisegundos, utiliza como parámetro de entrada un entero sin signo de 32 bits. La función no retorna ningún valor. Por ejemplo, para generar un retardo de 200ms se utiliza la siguiente línea de código:

  ```
  CyDelay(200);
  ```

- CyDelayUs():

  Genera un retardo en unidades de microsegundos, utiliza como parámetro de entrada un entero sin signo de 16 bits. La función no retorna ningún valor. Por ejemplo, para generar un retardo de 500us se utiliza la siguiente línea de código:

```
CyDelayUs(500);
```

- CyDelayFreq():

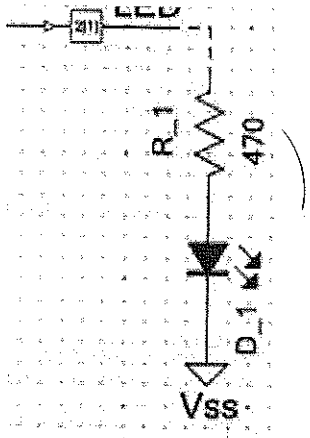
  Selecciona el valor de la frecuencia del bus de reloj del PSoC® en Hz, que se utiliza para generar el tiempo de la función CyDelay(), utiliza como parámetro de entrada un entero sin signo de 32 bits, si el valor es igual a cero se selecciona la frecuencia por defecto.

- CyDelayUs():

  Genera un retardo de tiempo basado en los ciclos de máquina del procesador; utiliza como parámetro de entrada un entero sin signo de 32 bits. La función no retorna ningún valor. Debe tenerse en cuanta que al valor utilizado en la función se le agregan entre 8 y 23 ciclos.

Se hace uso de un bloque de E/S de un bit, configurado como salida, con nombre LED, la opción HW connection deshabilitada y la opción External terminal habilitada, se conecta a este bloque una resistencia, un LED y la tierra del circuito (ver figura 88).

Figura 88. Diagrama circuital con la solución planteada



Fuente: elaboración propia.

Se selecciona el bit P2[1] del PSoC®5LP para conectar el LED (ver figura 89), ya que en el kit CY8CKIT-059 este LED viene conectado de fábrica, lo que facilita la implementación de la solución.

Figura 89. Configuración de los recursos del sistema

Alias	Name	Port	Pin	Lock
	LED	P2[1]	68	☑

Fuente: elaboración propia.

Finalmente el algoritmo implementado utiliza la función API para generar el tiempo y las funciones de lectura y escritura de pines que sirven para cambiar el estado de la salida actual del pin del PSoC®.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 5-3:

```
/* ==========================================================
 *
 * Programa para hacer oscilar un pin del PSoC® a una frecuencia
 * de 1Hz. El proceso de temporización se realiza por software.
 *
 * ==========================================================
 */
#include <project.h>
int main()
{
  for(;;)
  {
    CyDelay(500); //Genera retardo de tiempo de 500ms
    LED_Write(~LED_Read()); //Lee el valor del puerto y lo complementa
  }
}
```

Igual que para los ejemplos anteriores, la verificación del diseño y del algoritmo se realiza luego de programar el PSoC®, haciendo uso del kit CY8CKIT-059.

- Para complementar el tema de la temporización por software se plantea un segundo ejemplo en el que se pide modificar el tiempo de oscilación del pin del PSoC®. El tiempo cambiará de acuerdo con cuál de cuatro pulsadores N.A. se oprima así: pulsador uno tiempo 50ms, pulsador dos tiempo 100ms, pulsador tres tiempo 250ms y pulsador cuatro 500ms (valor de tiempo por defecto).

Solución

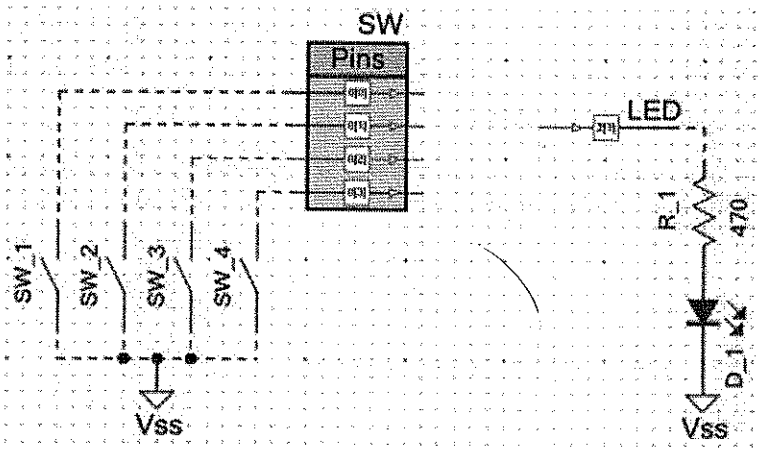
Los pulsadores N.A. son interruptores mecánicos que cierran un contacto temporalmente al oprimirse, similar a lo que ocurre con una tecla en un teclado; de esta manera se puede establecer un nivel lógico para el pulsador en estado de reposo (sin oprimir) y otro nivel para el momento cuando el pulsador se oprime.

La solución planteada para el ejemplo hace uso de la posibilidad de verificar el nivel

lógico generado por un pulsador o por un grupo de ellos. En este caso se organizan como un grupo que se maneja por un bloque de E/S de cuatro bits, el que se revisa constantemente por software para determinar si alguno cambió su estado de reposo; si este evento ocurrió se verifica cuál cambió su estado y se actualiza el tiempo de oscilación del pin de salida que tiene conectado el LED.

Para implementar la solución se hace uso de un bloque de E/S de cuatro bits configurado como entrada, con nombre SW, la opción HW connection deshabilitada, la opción External terminal habilitada, allí se conectan cuatro interruptores que se encuentran en el catálogo de componentes Off-Chip en la carpeta Electro-Mechanical y la tierra del circuito, y en la opción Drive mode se selecciona Resistive pull up. Se utiliza otro bloque de E/S de un bit, configurado como salida, con el nombre LED, la opción HW connection deshabilitada y la opción External terminal habilitada. Se conecta a este bloque una resistencia, un LED y la tierra del circuito (ver figura 90).

Figura 90. Diagrama circuital: manejo de pulsadores y LED



Fuente: [Autores]

Se selecciona el bit P2[1] del PSoC®5LP para conectar el LED que, como se indicó en el ejemplo anterior, viene conectado de fábrica a este pin el kit utilizado y los bits P0[3:0] para la conexión de los pulsadores (figura. 91).

Figura 91. Configuración de pines para manejo de pulsadores y LED

Alias	Name	Port	Pin	Lock
	\SW[3:0]\	P0[3:0]	48..51	☑
	LED	P2[1]	63	☑

Fuente: elaboración propia.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 5-4:

```
/* ==========================================================
 *
 * Programa para hacer oscilar un pin del PSoC®, el tiempo de
 * oscilación dependerá del último pulsador oprimido, así:
 * pulsador-1 50ms, pulsador-2 100ms, pulsador-3 250ms, pulsador-4
 * 500ms (tiempo por defecto). El proceso de temporización se
 * realiza por software.
 *
 * ==========================================================
 */
#include <project.h>
/* Definición de variables */
uint16 tiempo;
uint8 estado;
int main()
{
  tiempo=500; //Tiempo inicial de 500ms
  for(;;)
  {
    estado=SW_Read(); //Lee estado de los pulsadores
    if (estado==0x0F){ //Si no se oprime ninguno se procede a generar
      CyDelay(tiempo); //el tiempo y a cambiar el estado del pin
      LED_Write(~LED_Read());
    }else{ //Si se detecta un pulsador oprimido se verifica cual fue
      switch (estado){
        case 0x0E: //Si pulsador oprimido es el uno
          tiempo=50; //Se cambia tiempo a 50ms
        break;
        case 0x0D: //Si pulsador oprimido es el dos
```

```
            tiempo=100; //Se cambia tiempo a 100ms
          break;
          case 0x0B: //Si pulsador oprimido es el tres
            tiempo=250; //Se cambia tiempo a 250ms
          break;
          case 7: //Si pulsador oprimido es el cuatro
            tiempo=500; //Se cambia tiempo a 500ms
          break;
          default:
          break;
        }
      }
    }
  }
```

La verificación del diseño y del algoritmo se realiza luego de programar el PSoC®, conectando los pulsadores N.A. a la tarjeta de desarrollo que incluye el CY8CKIT-059.

Bloque LCD alfanumérica

La visualización de información en formato alfanumérico es utilizada en múltiples aplicaciones, en las cuales el usuario debe leer o visualizar datos que él escribe, como datafonos, contadores, multímetros, etc. Para esto se fabrican pantallas de cristal líquido (LCD) que permiten visualizar información en formato ASCII, además caracteres Kanji y griegos, utilizando para esto un controlador estándar creado por Hitachi de referencia HD44780.

Típicamente cuando se hace uso de una LCD, el usuario debe ir al manual de usuario para conocer tanto la información física, es decir, distribución y funcionalidad de pines, como la información de cada uno de los comandos o funciones que soporta. El PSoC Creator le ahorra una gran parte de esta tarea al usuario, ya que contiene un bloque donde vienen incluidas múltiples funciones para la configuración y el manejo de la LCD.

Descripción general

Las características detalladas para el bloque LCD denotado como LCD_Char pueden encontrarse en el datasheet con nombre CharLCD_v2_10.pdf; las siguientes son las más importantes:

- Uso de todos los comandos del controlador estándar Hitachi HD44780.
- Soporte de todos los tamaños de LCD comerciales, como por ejemplo la típica LCD de 2x16 caracteres.
- Uso de solo siete líneas de E/S para su operación.
- Incluye un editor de caracteres para que el usuario cree hasta 8 propios.
- Soporte de barras horizontales y verticales para aplicaciones que requieran mostrar procesos de llenado, tiempo o similares.
- Rutinas de conversión de datos para visualización directa de estos.

Como se mencionó antes, para la conexión física de la LCD con el PSoC® se utilizan solamente siete líneas de E/S, cuatro de ellas para la conexión con el bus de datos de la LCD y las restantes tres para la conexión con las líneas de control; esto implica que se utilice la interface de cuatro bits que soporta la LCD.

En la tabla 15 pueden verse las señales utilizadas y sus respectivas funciones, para la

conexión de la LCD al PSoC®. Para el caso específico de la conexión realizada a la tarjeta de desarrollo implementada para el montaje de los ejemplos de este libro se utilizaron los pines mostrados en la columna "Pin utilizado del kit CY8CKIT-059".

Pin Lógico	Pin LCD	Función	Pin utilizado del kit CY8CKIT-059
LCDPort_0	DB4	Bit de datos 0	P12.0
LCDPort_1	DB5	Bit de datos 1	P12.1
LCDPort_2	DB6	Bit de datos 2	P12.2
LCDPort_3	DB7	Bit de datos 3	P12.3
LCDPort_4	E	Habilitación de la LCD	P12.4
LCDPort_5	RS	Selección de palabra de control o dato en ASCII	P12.5
LCDPort_6	R/!W	Lectura/Escritura de la LCD	P12.6

Tabla 15. Pines utilizados para la conexión de una LCD al PSoC®
Fuente: elaboración propia.

Uso del bloque LCD_Char

Para seleccionar en el PSoC Creator el bloque LCD denominado Character LCD [v2.10], el usuario debe buscar la carpeta Display del catálogo de componentes (figura 92); allí se tienen 12 opciones que permiten utilizar diversos periféricos para visualización de datos.

Configuración del bloque LCD T2

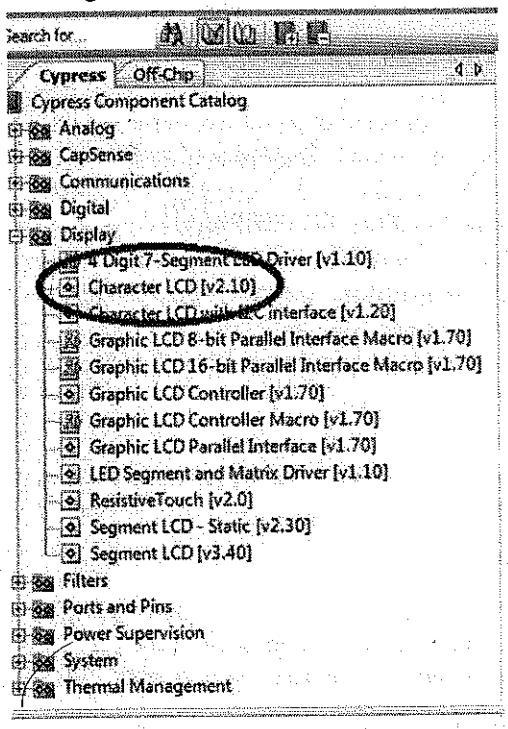
La ventana de configuración del bloque LCD permite modificar o configurar básicamente tres parámetros: nombre (Name), la opción caracteres personalizados (LCD Custom Character Set) y la habilitación/deshabilitación de rutinas para conversión de valores numéricos a caracteres ASCII (Include ASCII to Number Conversion Routines).

En la figura 93 puede verse la ventana de configuración del bloque LCD_Char con las funcionalidades antes nombradas, que se activa al hacer doble clic sobre el símbolo, como se ha hecho con otros bloques del PSoC®.

El parámetro LCD Custom Character Set permite seleccionar cuatro posibles opciones para la creación o no de caracteres personalizados:

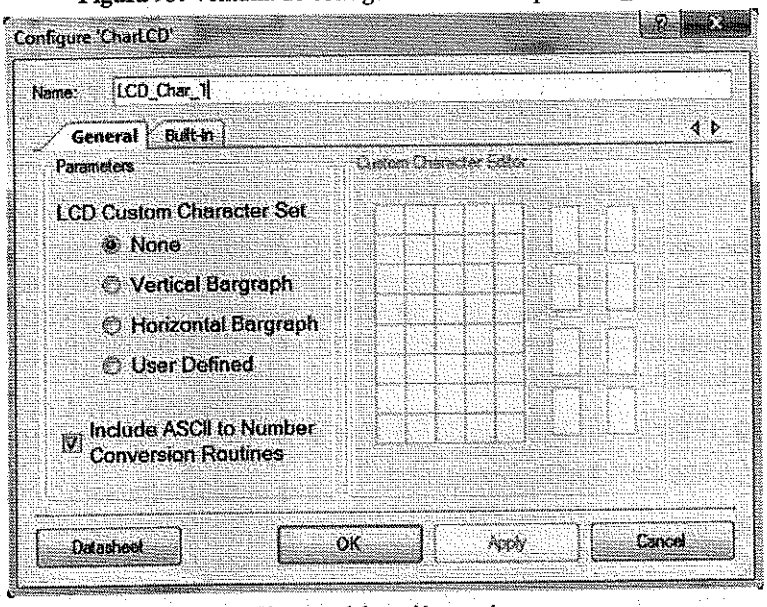
- *None*: no permite generar caracteres personalizados (valor por defecto).
- *Vertical Bar Graph*: Genera barras verticales que son manejadas por medio de funciones API propias del bloque LCD.

Figura 92. Selección del bloque LCD



Fuente: elaboración propia.

Figura 93. Ventana de configuración del bloque LCD_Char



Fuente: elaboración propia.

- *Horizontal Bar Graph*: genera barras horizontales que son manejadas por medio de funciones API propias del bloque LCD.
- *User Defined*: crea por medio del Custom Character Editor (disponible en la misma ventana de configuración) ocho caracteres personalizados, de tamaño 5x8 pixeles, que el usuario puede dibujar en esta herramienta, estos caracteres deben ser cargados a la memoria de la LCD al configurarla.

Rutinas utilizadas para el manejo del bloque LCD

El bloque LCD divide en tres grupos las funciones API para su operación; estos grupos incluyen funciones generales, funciones para manejo de caracteres personalizados y funciones para el manejo de conversión de datos ASCII. Para la sintaxis de las funciones debe tenerse en cuenta que la primera parte del nombre de la función (LCD_Char_) debe ser reemplazado por el nombre que el usuario le haya asignado al bloque.

Las funciones generales que pueden ser utilizadas para configurar la LCD y en general para su manejo son:

- LCD_Char_Start():

  Inicializa la LCD con los siguientes parámetros: interface de 4 bits, borra display, configura el autoincremento en la escritura y ubica el cursor al inicio. No utiliza ningún parámetro de entrada y tampoco retorna ningún valor.

- LCD_Char_DisplayOn():

  Enciende el display de la LCD. No utiliza ningún parámetro de entrada y tampoco retorna ningún valor.

- LCD_Char_DisplayOff():

  Apaga el display de la LCD. No utiliza ningún parámetro de entrada y tampoco retorna ningún valor.

- LCD_Char_PrintString():

  Escribe una cadena de caracteres en la LCD, en la ubicación actual del cursor. La función utiliza como parámetro de entrada un arreglo de datos en formato ASCII de 8 bits. La función no retorna ningún valor. Por ejemplo, para escribir en el bloque LCD_Char denominado LCD un mensaje codificado en ASCII se puede escribir lo siguiente:

  ```
  LCD_PrintString("Mensaje de Ejemplo");
  ```

- LCD_Char_PutChar():

  Escribe un carácter en la LCD, en la ubicación actual del cursor. La función utiliza como parámetro de entrada un dato en formato ASCII de 8 bits. Es utilizada también para transferir a la LCD los caracteres personalizados creados por el usuario. La función no retorna ningún valor; por ejemplo, si se quiere escribir

en el bloque LCD_Char, al que se ha llamado LCD, el carácter ASCII 'A' se puede utilizar la siguiente línea de código:

```
LCD_PutChar('A');
```

- LCD_Char_Position():

Mueve el cursor a una posición específica de la LCD denotada como fila-columna. La función utiliza como parámetros de entrada dos valores de 8 bits sin signo que representan fila y columna en ese orden. La función no retorna ningún valor. Si por ejemplo se quiere ubicar el cursor en la fila 0, columna 5 en un bloque LCD_Char denominado LCD se puede escribir la siguiente línea de código:

```
LCD_Position(0,5);
```

- LCD_Char_WriteDatar():

Escribe un dato en la ubicación actual del cursor en la LCD. La función utiliza como parámetro de entrada un dato de 8 bits sin signo. La función no retorna ningún valor; por ejemplo, si se quiere escribir en el bloque LCD_Char, al que se ha llamado LCD, el valor 45 hexadecimal se puede utilizar la siguiente línea de código:

```
LCD_WriteData(0x45);
```

- LCD_Char_WriteControl ():

Escribe una palabra de control o comando en la LCD. Estas palabras de control son las que se utilizan en una LCD típica. La función utiliza como parámetro de entrada una constante de 8 bits sin signo; estas constantes están predefinidas (ver tabla 16).

Nombre de la constante	Función
LCD_Char_CLEAR_DISPLAY	Borrar display
LCD_Char_CURSOR_HOME	Ubicar el cursor al inicio
LCD_Char_CURSOR_LEFT	Cursor se mueve a la izquierda
LCD_Char_CURSOR_RIGHT	Cursor se mueve a la derecha
LCD_Char_DISPLAY_ CURSOR_ON	Habilitar display y cursor encendido
LCD_Char_DISPLAY_ON_ CURSOR_OFF	Habilitar display y cursor apagado
LCD_Char_CURSOR_BLINK	Habilitar display, cursor apagado y parpadeando
LCD_Char_CURSOR_SH_ LEFT	Mover cursor/desplazar display a la izquierda

Nombre de la constante	Función
LCD_Char_CURSOR_SH_ RIGHT	Mover cursor/desplazar display a la derecha
LCD_Char_DISPLAY_2_ LINES_5x10	Habilita 2 líneas caracteres de 5x10

Tabla 16. Constantes de la función LCD_Char_ WriteControl
Fuente: elaboración propia.

Por ejemplo, si se quiere habilitar el cursor en la LCD, que por defecto no lo está, en el bloque LCD_Char, al que se ha llamado LCD la sintaxis utilizada sería:

```
LCD_WriteControl(LCD_DISPLAY_CURSOR_ON);
```

- LCD_Char_ClearDisplay():

  Borra la información visualizada en la LCD y ubica el cursor al incio. No utiliza ningún parámetro de entrada y tampoco retorna ningún valor.

El segundo grupo de funciones está relacionado con la posibilidad de crear caracteres personalizados, que son útiles cuando se requiere un icono, símbolo o letra que no está disponible en la tabla de caracteres que trae de fábrica la LCD. Para esto se tienen las siguientes funciones:

- LCD_Char_LoadCustomFonts():

  Carga los ocho caracteres personalizados, ya sean las barras horizontales, verticales o los creados por el usuario, en la memoria de la LCD, para que luego puedan ser utilizados en una aplicación. La función está disponible únicamente si alguna de las tres opciones de personalización de caracteres esta seleccionada (ver Configuración del bloque LCD). La función utiliza como parámetro de entrada un arreglo de 64 bytes (cada uno es una constante de 8 bits sin signo) que incluye los ocho caracteres personalizados (cada carácter requiere ocho bytes para su implementación). La función no retorna ningún valor. Por ejemplo, para cargar el bloque LCD_Char denominado LCD la tabla con los caracteres creados por el usuario denominada LCD_customFonts, se puede escribir la siguiente línea de código:

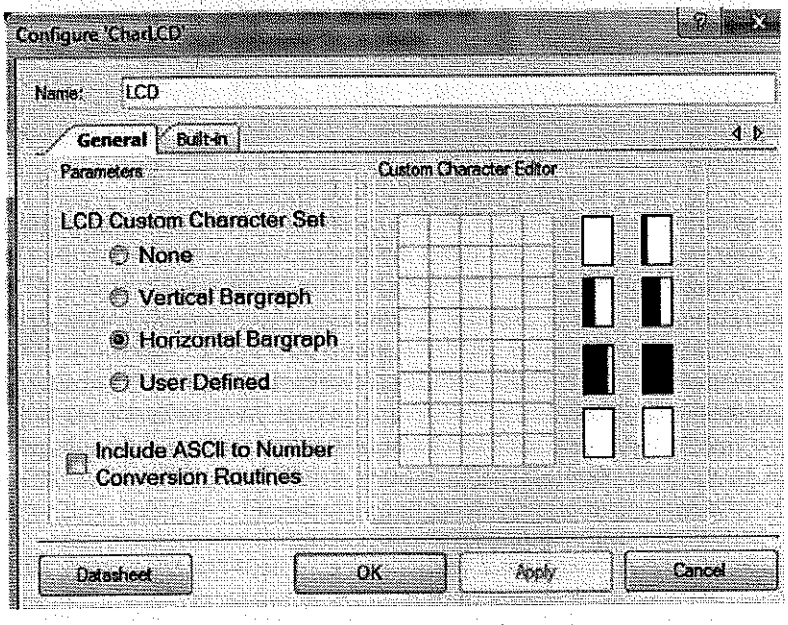
```
LCD_LoadCustomFonts(LCD_customFonts);
```

- LCD_Char_DrawHorizontalBG():

  Dibuja en la LCD un carácter, de los ocho posibles, que representa una parte de una barra horizontal. La función utiliza cuatro parámetros de entrada: fila que corresponde a un valor entero de 8 bits sin signo, columna que corresponde a un valor entero de 8 bits sin signo. Estos dos valores representan la ubicación fila-columna donde quiere ubicarse el inicio de la barra horizontal; el siguiente parámetro determina la cantidad máxima de caracteres utilizados por la barra

horizontal por fila, es un valor entero de 8 bits sin signo. El último parámetro determina la cantidad de columnas-pixel dibujadas; este valor no debe exceder la cantidad máxima de caracteres por fila (en la figura 94 pueden verse los caracteres que pueden generarse en una barra horizontal). La función no retorna ningún valor.

Figura 94. Ventana con los caracteres utilizados por la función LCD_Char_DrawHorizontalBG()



Fuente: elaboración propia.

Por ejemplo, para dibujar una barra horizontal que tenga una longitud de 12 columnas-pixel, iniciando en la fila 0 columna 5 del bloque LCD_Char denominado LCD se puede escribir la siguiente línea de código:

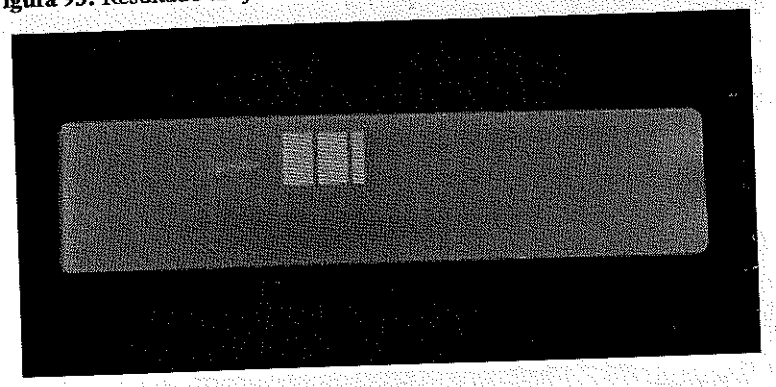
```
LCD_DrawHorizontalBG(0,5,3,12);
```

De esa línea de código se determina que: 0=representa la fila donde inicia la barra, 5= representa la columna donde inicia la barra, 3=representa el número máximo de caracteres (cada carácter equivale a cinco columnas-pixel) que pueden dibujarse y 12=representa el número de columnas-pixel dibujadas. En la figura 95 puede verse el resultado en una LCD real al programar esta línea de código en el PSoC®.

- LCD_Char_DrawVerticalBG():

Dibuja en la LCD un carácter, de los ocho posibles, que representa una parte de una barra vertical. La función utiliza cuatro parámetros de entrada: fila que

Figura 95. Resultado al ejecutar la función LCD_Char_DrawHorizontalBG()



Fuente: elaboración propia.

corresponde a un valor entero de 8 bits sin signo, columna que corresponde a un valor entero de 8 bits sin signo. Estos dos valores representan la ubicación fila-columna donde quiere ubicarse el inicio de la barra vertical, el siguiente parámetro determina la cantidad máxima de caracteres utilizados por la barra horizontal por columna; es un valor entero de 8 bits sin signo. El último parámetro determina la cantidad de filas-pixel dibujadas; este valor no debe exceder la cantidad máxima de caracteres por columna. La función no retorna ningún valor; por ejemplo, para dibujar una barra vertical que tenga una longitud de 14 filas-pixel, iniciando en la fila 1 columna 6 del bloque LCD_Char denominado LCD se puede escribir la siguiente línea de código:

```
LCD_DrawVerticalBG(1,6,2,14);
```

De esa línea de código se determina que: 1=representa la fila donde inicia la barra, 6= representa la columna donde inicia la barra, 2=representa el número máximo de caracteres (cada carácter equivale a ocho filas-pixel) que pueden dibujarse y 14=representa el número de filas-pixel dibujadas.

El tercer y último grupo de funciones está relacionado con la conversión de datos de valores numéricos a formato ASCII para ser utilizado directamente en la visualización de información:

- LCD_Char_PrintInt8():

  Escribe en la LCD dos caracteres ASCII que representan el valor numérico codificado en hexadecimal a convertir. La función utiliza como parámetro de entrada un dato de 8 bits sin signo. La función no retorna ningún valor; por ejemplo, si se quiere escribir en el bloque LCD_Char, llamado LCD, el valor 29 hexadecimal se puede escribir lo siguiente:

  ```
  LCD_PrintInt8(0x29);
  ```

- LCD_Char_PrintInt16():

Escribe en la LCD cuatro caracteres ASCII que representan el valor numérico codificado en hexadecimal a convertir, el valor visualizado incluye los ceros no significativos. La función utiliza como parámetro de entrada un dato de 16 bits sin signo. La función no retorna ningún valor; por ejemplo, si se quiere escribir en el bloque LCD_Char, llamado LCD, el valor 837 hexadecimal se puede escribir lo siguiente:

```
LCD_PrintInt16(0x837);
```

- LCD_Char_PrintInt32():

Escribe en la LCD ocho caracteres ASCII que representan el valor numérico codificado en hexadecimal a convertir, el valor visualizado incluye los ceros no significativos. La función utiliza como parámetro de entrada un dato de 32 bits sin signo. La función no retorna ningún valor; por ejemplo, si se quiere escribir en el bloque LCD_Char, llamado LCD, el valor 1A837 hexadecimal se puede escribir lo siguiente:

```
LCD_PrintInt32(0x1A837);
```

- LCD_Char_PrintNumber():

Escribe en la LCD el valor numérico codificado en decimal en formato ASCII justificado a la izquierda, es decir, los ceros no significativos no se visualizan. La función utiliza como parámetro de entrada un dato de 16 bits sin signo. La función no retorna ningún valor; por ejemplo, si se quiere escribir en el bloque LCD_Char, llamado LCD, el valor 523 se puede escribir lo siguiente:

```
LCD_PrintNumber(523);
```

- LCD_Char_PrintU32Number():

Escribe en la LCD el valor numérico codificado en decimal en formato ASCII justificado a la izquierda, es decir, los ceros no significativos no se visualizan. La función utiliza como parámetro de entrada un dato de 32 bits sin signo. La función no retorna ningún valor; por ejemplo, si se quiere escribir en el bloque LCD_Char, llamado LCD, el valor 8225 se puede escribir lo siguiente:

```
LCD_PrintU32Number(8225);
```

Ejemplos de aplicación del bloque LCD

Manejo básico de la LCD

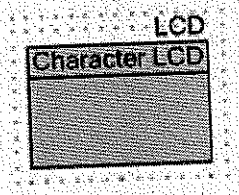
En el ejemplo se presenta el proceso de configuración y manejo del bloque LCD para visualizar un mensaje en la LCD, el mensaje estará almacenado en la memoria del PSoC®.

Solución

Aprovechando las diversas funcionalidades que incluye el bloque LCD alfanumérico mostrado en la figura 96, la tarea de diseño se simplifica en gran medida. Para este

caso en particular solamente se deben tener en cuenta dos tareas, la primera determina cómo se quiere configurar la LCD, la segunda determinar cómo se conectará físicamente la LCD al PSoC®.

Figura 96. Vista del bloque LCD en PSoC Creator



Fuente: elaboración propia.

Para el primer paso del diseño —como es obvio— se hace uso del bloque Character_LCD del catálogo de componentes del PSoC Creator, será el único bloque utilizado en el diseño. Al bloque se le modificaron dos parámetros, Name: LCD y se le deshabilito la opción Include ASCII to Number Conversion Routines que viene activa por defecto. Es conveniente hacer esto último si no se van a utilizar las rutinas de conversión de datos, ya que estas generan código adicional que ocupa espacio en la memoria.

La selección del puerto de conexión de la LCD al PSoC®5LP corresponde al segundo paso del diseño. En este caso dado que se va a utilizar el sistema de desarrollo sugerido en este libro para la implementación del problema, el puerto por utilizar será el P12[6:0] que está conectado físicamente a la LCD.

A continuación puede verse el código escrito para la solución del problema, al que se le agregó una tarea adicional que hace que el mensaje visualizado parpadee cinco veces antes de permanecer estático:

Ejemplo 6-1:

```
/* ==========================================================
*
* Programa que permite visualizar en una LCD de 2x16 un mensaje
* almacenado en la memoria del PSoC, el cual parpadea cinco veces
* antes de permanecer estático.
*
* ==========================================================
*/
#include <project.h>
/* Definición de variables */
uint8 i;
```

```
int main()
{
  LCD_Start(); //Configura la LCD
  LCD_Position(0,3); //Ubica el cursor en la fila 0 columna 3
  LCD_PrintString("Ejemplo 6-1"); //Escribe el mensaje allí
  LCD_Position(1,5); //Ubica el cursor en la fila 1 columna 5
  LCD_PrintString("PSoC5LP"); //Escribe el mensaje allí
  for (i=0;i<5;i++){
    CyDelay(500);
    LCD_DisplayOff(); //Apaga el display
    CyDelay(500);
    LCD_DisplayOn(); //Enciende el display
  }
  for(;;)
  {
  }
}
```

La verificación del diseño y del algoritmo se realizan luego de programar el PSoC®, haciendo uso de la tarjeta de desarrollo donde se incluye el kit CY8CKIT-059.

Decodificador de teclado matricial con visualización en LCD

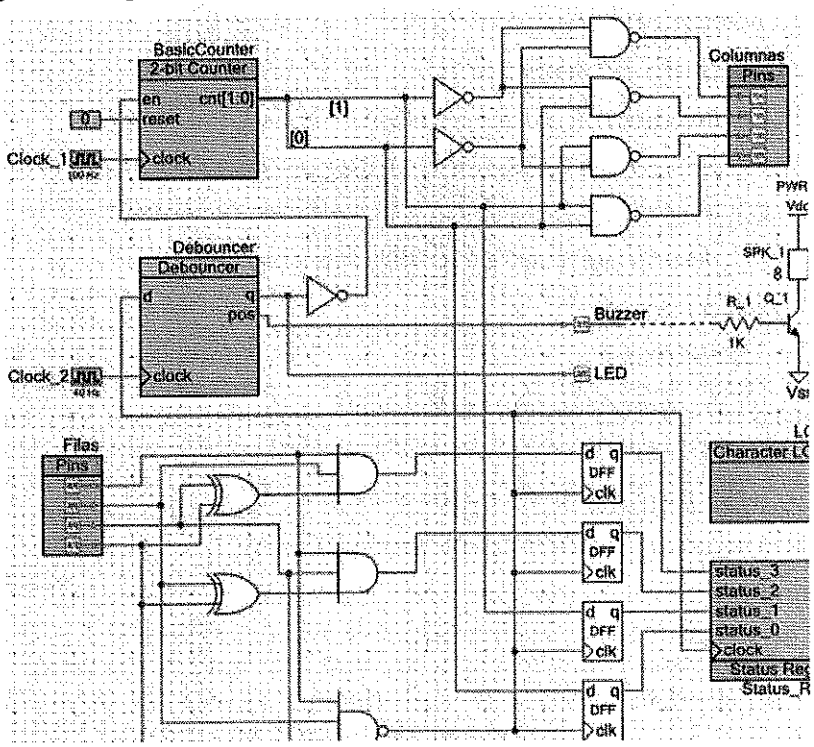
En el siguiente ejemplo se presenta la interacción entre el decodificador de teclado matricial implementado en el ejemplo del numeral 5.5.1 de este libro y una LCD alfanumérica. Se visualiza en la LCD el valor de la tecla oprimida.

Solución

Parte de la solución está desarrollada ya que se aprovecha el diseño del decodificador de teclado del ejemplo del aparte "Diseño de un decodificador de teclado matriarcal", con algunas modificaciones: en la primera se elimina el bloque Salidas y se reemplaza por el bloque Status Register; la segunda está relacionada con el bloque Debouncer, al que se le habilita la opción Positive edge del parámetro Edge type, que genera un pulso temporizado cada vez que se oprime una tecla; a esta señal se conecta un pin de E/S denominado Buzzer donde irá conectado externamente el buzzer de la tarjeta de desarrollo.

Finalmente, para visualizar el valor de la decodificación en la LCD se debe agregar un bloque Character_LCD del catálogo de componentes al diseño. El diseño final planteado puede verse en la figura 97.

Figura 97. Diagrama circuital del decodificador de teclado con visualización en LCD



Fuente: elaboración propia.

La configuración de los bloques del diseño es la siguiente: bloque Status Register, a este bloque se le modificaron dos parámetros: Name: Status_Reg e Inputs: 4, para el bloque Character_LCD se le modificaron dos parámetros: Name: LCD y se le deshabilitó la opción Include ASCII to Number Conversion Routines.

Para la conexión física de los bloques, se selecciona el puerto P12[6:0]; para conectar la LCD al PSoC, el puerto P1[7:4]; para las columnas de teclado, el puerto P2[7:4]; para las filas del teclado, el bit P2[1]; para la conexión del buzzer, y el bit P2[0] para la señal de tecla oprimida generada por el decodificador.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 6-2:

```
/* ============================================================
 *
 * Programa que permite visualizar en una LCD de 2x16 el valor
 * generado por un teclado matricial.
 *
 * ============================================================
```

```
*/
#include <project.h>
/* Definición de variables */
uint8 tecla;
uint8 Tabla[]={"123A456B789C*0#D"};
int main()
{
  LCD_Start();
  for(;;)
  {
    while (LED_Read()==0); //Espera hasta que se oprima una tecla
    tecla=Status_Reg_Read(); //Lee el valor de la tecla decodificada
    LCD_PutChar(Tabla[tecla]); //Escribe en la LCD el valor leído
    while (CyPins_ReadPin(LED_0)!=0);//Espera hasta que se suelte tecla
  }
}
```

Paso final: la verificación del diseño y del algoritmo se realizan luego de programar el PSoC®, haciendo uso de la tarjeta de desarrollo.

Manejo de caracteres personalizados

En el ejemplo se pide diseñar un programa que permita visualizar en la LCD el valor en decimal de la combinación binaria generada por un dip switch de ocho entradas conectado a un puerto del PSoC®; además, de manera visual se mostrará en una barra horizontal el valor equivalente hasta el número 80, que corresponde a la máxima cantidad que se puede visualizar en un a LCD de 2x16 caracteres.

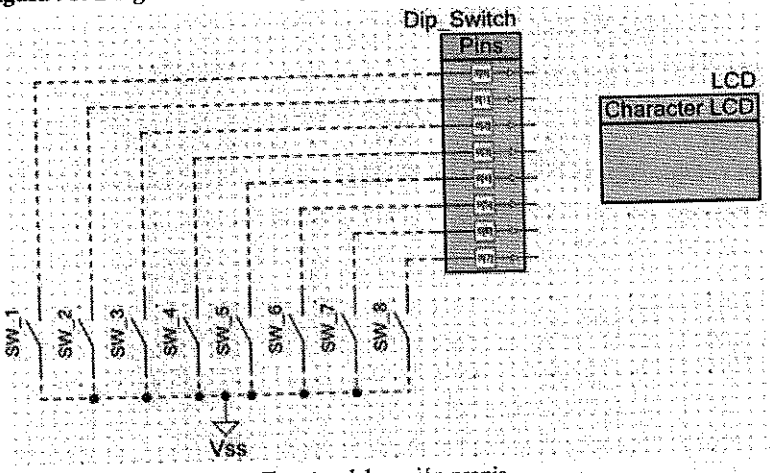
Solución

La solución planteada utiliza las funciones de conversión y visualización directa de datos que tiene el bloque LCD, además de la generación de la barra grafica horizontal.

Para el diseño se hace uso del bloque Character_LCD del catálogo de componentes, al que se le modificaron dos parámetros: Name: LCD y se seleccionó la opción Horizontal Bar Graph del parámetro LCD Custom Character Set. En la figura 98 puede verse el diagrama circuital con la solución propuesta.

Se requiere también para la solución un bloque de E/S de ocho líneas, configurado como entrada, donde se conectará el dip switch con nombre Dip_Switch, la opción

Figura 98. Diagrama circuital ejemplo de manejo de caracteres personalizados



Fuente: elaboración propia.

HW connection deshabilitada, la opción External terminal habilitada; allí se conectan ocho interruptores del catálogo de componentes y la tierra general del circuito, y finalmente en la opción Drive mode se selecciona Resistive pull up.

Para la conexión física de los bloques, se selecciona el puerto P12[6:0] para conectar la LCD al PSoC®5LP, y el puerto P0[7:0] para la conexión del dip switch.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 6-3:

```
/* ==========================================================
 *
 * Programa que permite visualizar en una LCD de 2x16 el valor
 * generado por la combinación binaria en un dip switch, este
 * valor es convertido y mostrado en la LCD en formato decimal.
 * Además, se utiliza una barra gráfica horizontal que representa
 * el valor leído limitado a un valor máximo de 80.
 *
 * ==========================================================
 */
#include <project.h>
/* Definición de variables */
uint8 valor;
int main()
```

```
{
  LCD_Start();
  LCD_PrintString("Valor=");
  valor=0;
  for(;;)
  {
    LCD_Position(0,6); //Ubica cursor en fila 0 columna 6
    LCD_PrintNumber(valor); //Convierte y visualiza el dato leído del
                            //dip switch
    LCD_DrawHorizontalBG(1,0,16,valor); //Dibuja la barra horizontal
                            //iniciando en la fila 1 columna 0
    while (valor==Dip_Switch_Read()); //Si el valor actual del dip
                            //switch no cambia espera
    valor=Dip_Switch_Read(); //Lee valor actual del dip switch
    LCD_Position(0,6); //Ubica cursor en fila 0 columna 6
    LCD_PrintString(" "); //Borra valor actual
  }
}
```

La verificación del diseño y del algoritmo se realizan luego de programar el PSoC®, haciendo uso de la tarjeta de desarrollo.

Animación simple con caracteres personalizados en la LCD

Para este ejemplo se pide diseñar un programa que muestre el desplazamiento de la imagen de un carro en la LCD, el desplazamiento se generará de izquierda a derecha, y así se producirá en la LCD una animación rudimentaria. La velocidad de desplazamiento de la imagen será ajustada por cuatro pulsadores N.A.

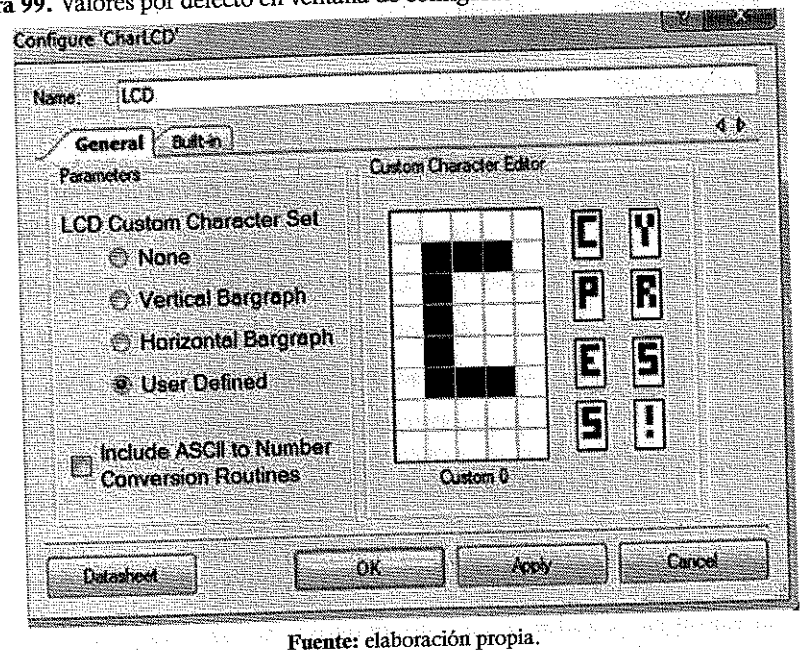
Solución

Para solucionar el problema planteado se debe hacer unos de las funciones que permiten visualizar caracteres generados por el usuario; como se indicó antes en este capítulo, el bloque LCD incluye esta funcionalidad. Para realizar este proceso se inicia dibujando los caracteres que se requieren, al realizar esta tarea inmediatamente las funciones asociadas a la visualización de caracteres personalizados se habilitan.

Se hace uso del bloque Character_LCD del catálogo de componentes, al que se le modificaron tres parámetros, Name: LCD, se seleccionó la opción User Defined del parámetro LCD Custom Character Set y se le deshabilitó la opción Include ASCII to Number Conversion Routines.

Al seleccionar la opción User Defined, en la ventana de configuración, el bloque muestra por defecto el dibujo de unos caracteres de fábrica (ver figura 99), sobre los que realiza el dibujo que el usuario desea generar.

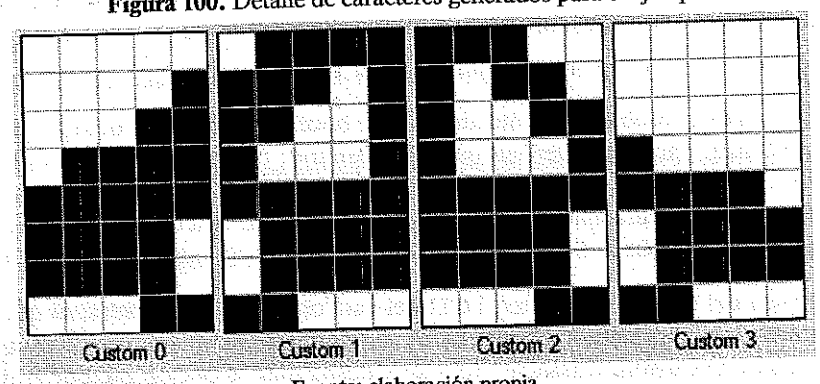
Figura 99. Valores por defecto en ventana de configuración de caracteres del bloque LCD



Fuente: elaboración propia.

La imagen utilizada para la animación se muestra en detalle en la figura 100; se utilizaron únicamente cuatro de los ocho espacios de memoria que pueden personalizarse. Cada espacio tiene un nombre con un subíndice numérico (de 0 a 7), que está predefinido en el PSoC Creator, denominado Custom_, este valor es utilizado como una constante para hacer referencia a la imagen específica que se va a utilizar en la visualización.

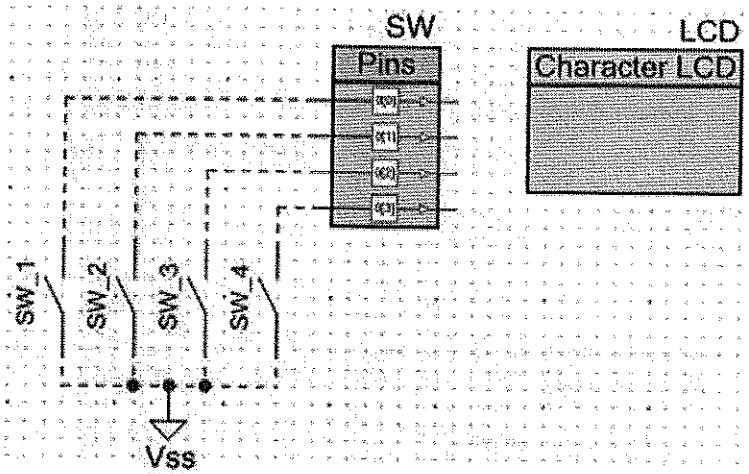
Figura 100. Detalle de caracteres generados para el ejemplo



Fuente: elaboración propia.

Se utiliza también un bloque de E/S de cuatro bits configurado como entrada, con nombre SW, con la opción HW connection deshabilitada y la opción External terminal habilitada, donde se conectan cuatro interruptores del catálogo de componentes Off-Chip y la tierra del circuito, en la opción Drive mode se selecciona Resistive pull up. Con esto se asegura un valor lógico alto en todas las entradas del bloque cuando el pulsador no está oprimido. En la figura 101 puede verse el diagrama circuital del diseño final sugerido.

Figura 101. Diagrama circuital final del diseño sugerido



Fuente: elaboración propia.

Finamente para la conexión física de los elementos utilizados en el diseño se selecciona el puerto P12[6:0] para conectar la LCD al PSoC® y los bits P0[3:0] para la conexión de los pulsadores.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 6-4:

```
/* ==========================================================
 *
 * Programa que permite generar en una LCD de 2x16 una animación,
 * la velocidad con la que se desplaza la imagen depende del valor
 * seleccionado con uno de cuatro pulsadores N.A., conectados al
 * PSoC.
 *
 * ==========================================================
*/
```

```
#include <project.h>
/* Definición de variables */
extern uint8 const CYCODE LCD_customFonts[];
uint16 tiempo;
uint8 i,valor;
int main()
{
  LCD_Start();
  LCD_LoadCustomFonts(LCD_customFonts); //Carga tabla con
                                        // caracteres personalizados
  tiempo=500;
  for(;;)
  {
    if (SW_Read()==0x0F){ //Si no se oprime pulsador se visualiza
      for (i=36;i<60;i++){
        LCD_ClearDisplay();
        LCD_Position(0,i);
        LCD_PutChar(LCD_CUSTOM_0);//Visualiza carácter 0
        LCD_PutChar(LCD_CUSTOM_1);//Visualiza carácter 1
        LCD_PutChar(LCD_CUSTOM_2);//Visualiza carácter 2
        LCD_PutChar(LCD_CUSTOM_3);//Visualiza carácter 3
        CyDelay(tiempo);
      }
    }else{ //Si se oprime se verifica cual fue
      valor=SW_Read();
      switch (valor){
        case 0x0E: //Si pulsador oprimido es el uno
          tiempo=150; //Se cambia tiempo a 150ms
        break;
        case 0x0D: //Si pulsador oprimido es el dos
          tiempo=250; //Se cambia tiempo a 250ms
```

```
            break;
            case 0x0B: //Si pulsador oprimido es el tres
              tiempo=500; //Se cambia tiempo a 500ms
            break;
            case 7: //Si pulsador oprimido es el cuatro
              tiempo=800; //Se cambia tiempo a 800ms
            break;
            default:
            break;
          }
        }
      }
    }
```

Se plantea como ejercicio para el lector hacer que el tiempo se modifique dentro del ciclo actual de visualización y no hasta finalizar la visualización de los 16 espacios de la LCD, como se propuso en la solución mostrada.

Interrupciones en el PSoC5®LP

Una interrupción se puede definir como una señal generada por el hardware o una instrucción del software que obliga a la CPU del dispositivo utilizado, a detener la ejecución del programa que se está ejecutando (programa principal), para ejecutar una rutina, procedimiento o función que se considera como prioritaria y cuya ejecución debe realizarse en el momento en que se hace la solicitud [8].

Además de la ejecución inmediata de la rutina las interrupciones eliminan las consultas (polling) de periféricos, lo que elimina ciclos de interrogación de estos y permite la atención inmediata del periférico que lo solicite, lo que hace el programa principal más ágil.

Las interrupciones generadas por hardware pueden ser internas o externas. Las interrupciones internas son generadas por periféricos internos del microcontrolador, como por ejemplo un bloque DMA; por su parte, las interrupciones externas con generadas por periféricos que tienen contacto con el exterior, como por ejemplo un puerto de comunicaciones serie asincrónico. Estas interrupciones, ya sean internas o externas, pueden ser habilitadas o deshabilitadas por el usuario en cualquier momento por lo que se consideran como interrupciones enmascarables (maskable).

En la mayoría de controladores la o las interrupciones soportadas tienen una tabla en la cual se almacena la dirección de memoria donde se ubica la rutina para atender la interrupción. A esta tabla se le denomina tabla de vectores de interrupción. Cada vector tiene una prioridad que puede ser definida por el fabricante o como en el caso de la familia PSoC®5LP puede ser asignada por el usuario. Esto se utiliza para determinar qué interrupción se debe ejecutar cuando se generan dos o más solicitudes de interrupción simultáneamente.

En el PSoC®5LP todos los periféricos tienen la capacidad de generar interrupciones. El usuario se encarga de habilitar esta posibilidad de acuerdo con las necesidades de la aplicación. La interrupción es controlada por un bloque independiente, denominado cy_isr_v1_70, que el usuario puede configurar, y que además incluye funciones para la configuración y el manejo general de la interrupción.

Bloque de interrupción del PSoC®5LP

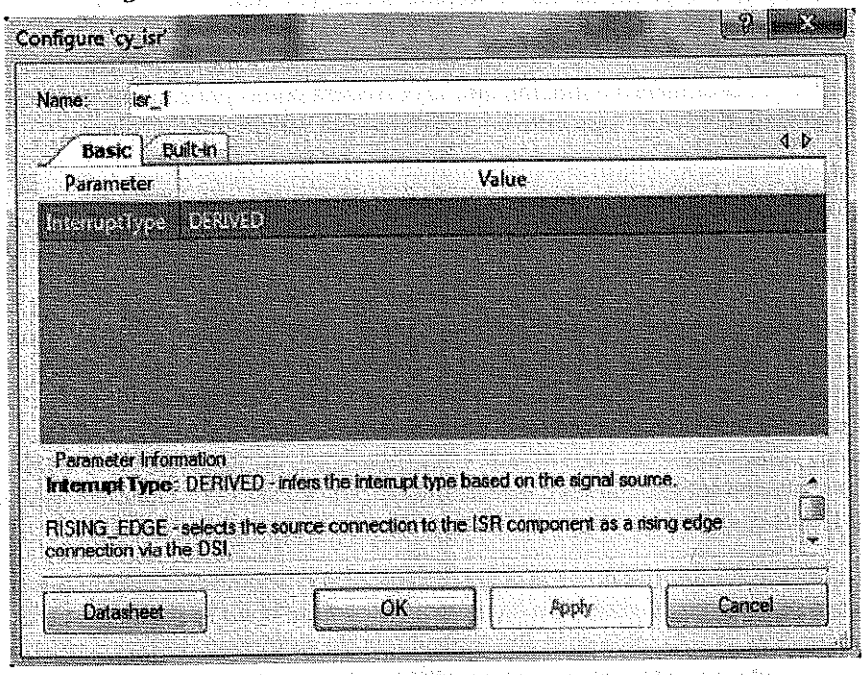
Las interrupciones en el PSoC® se manejan por medio de un bloque independiente, que puede procesar tres tipos de señales de interrupción: por nivel (la interrupción

permanece activa mientras la entrada de interrupción esté en alto), por pulso (la interrupción permanece activa durante un pulso del bus de reloj) o por flanco (la interrupción se genera cuando se detecta un flanco de subida).

Configuración del bloque de interrupción

En la figura 102 puede verse la ventana de configuración del bloque de interrupciones que puede encontrarse en la carpeta System con el nombre Interrupt [v1.70] del catálogo de componentes del PSoC Creator.

Figura 102. Ventana de configuración del bloque de interrupción



Fuente: elaboración propia.

A parte de permitir modificar el nombre del bloque, igual que ocurre con todos los bloques utilizados en el PSoC Creator, el tipo de señal de interrupción utilizada por el bloque también puede ser modificada entre tres valores posibles, relacionados con los tipos de señal que puede procesar explicados anteriormente:

- DERIVED: al generarse la petición de interrupción, el bloque inspecciona el tipo de bloque que la generó (fixed o UDB) y crea una petición tipo level o tipo rising_edge dependiendo del tipo de bloque detectado (valor por defecto)
- RISING_EDGE: genera la interrupción al detectar un flanco de subida del bloque que solicita la interrupción; este es convertido en un pulso equivalente a un periodo de la señal del bus de reloj del PSoC®, para que sea procesado por el controlador de interrupciones.

- LEVEL: forma la interrupción al detectar un nivel alto del bloque que realiza la petición, esta señal ingresa directamente al controlador de interrupciones del PSoC®.

Rutinas utilizadas para el manejo del bloque de interrupción

Las funciones API para el bloque de interrupción permiten realizar la configuración general y detallada de la interrupción por manejar. Para la sintaxis de estas funciones debe tenerse en cuenta que la primera parte del nombre de la función (ISR_) debe ser reemplazado por el nombre que el usuario le haya asignado al bloque.

Las funciones que pueden ser utilizadas para configurar y manejar el bloque de interrupción son:

- ISR_Start():

  Inicializa todo el proceso de interrupción que permite generar una petición al controlador de interrupciones. Asigna un valor por defecto tanto para el vector de interrupción, como para la prioridad. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- ISR_StartEx():

  Inicializa todo el proceso de interrupción que permite generar una petición al controlador de interrupciones del PSoC®. Asigna un valor por defecto tanto para el vector de interrupción, como para la prioridad. La diferencia con la función anterior es que permite la definición de una rutina de interrupción en el archivo main.c del proyecto y no fuera de este. Como parámetro de entrada utiliza el nombre de la rutina de interrupción que el usuario define con alguno de los macros CY_ISR o CY_ISR_PROTO. La función no retorna ningún valor. Si por ejemplo se requiere una interrupción para detectar un cambio en un bloque de E/S, se puede entonces utilizar un bloque de interrupción ISR denominado ISR_SW y la inicialización de la rutina de interrupción llamada INT_SW sería la siguiente:

```
ISR_SW_StartEx(INT_SW);
```

- ISR_SetPriority():

  Modifica el valor de prioridad de la interrupción. La función debe ejecutarse luego de las funciones ISR_Start() o ISR_StartEx(), ya que estas configuran la prioridad en un valor inicial por defecto. La función utiliza como parámetro de entrada un valor entre 0 y 7, siendo 0 el valor de mayor prioridad. La función no retorna ningún valor.

- ISR_GetPriority():

  Permite leer el valor de prioridad de la interrupción. La función no utiliza ningún parámetro de entrada y retorna un valor entero de 8 bits sin signo entre 0 y 7 que representa la prioridad de la interrupción, siendo 0 el valor de mayor prioridad.

- ISR_SetPending():

  Permite generar una petición de interrupción por software. No utiliza ningún parámetro de entrada y tampoco retorna ningún valor.

- ISR_ClearPending():

  Borra cualquier solicitud de interrupción pendiente en el controlador de interrupciones, siempre que el bloque que genera la petición de interrupción no tenga ninguna pendiente de ser atendida. No utiliza ningún parámetro de entrada y tampoco retorna ningún valor.

Ejemplos de uso del bloque de interrupción

Aplicación de una interrupción simple

Se pide modificar el tiempo de oscilación de un pin del PSoC®. El tiempo cambiará entre dos valores posibles 50ms y 800ms (valor de tiempo por defecto) cada vez que se oprima un pulsador N.A. conectado a una entrada del PSoC® configurada como interrupción.

Solución

La solución planteada para el ejemplo hace uso de un bloque de interrupción, lo que facilita todo el proceso ya que se elimina la tarea de verificar el nivel lógico generado por el pulsador, y se deja en el programa principal la tarea de generar el tiempo y de cambiar el estado del pin de E/S que tiene conectado un LED. Con esto se gana además precisión en el tiempo debido a que no se generan tiempos muertos producidos por la verificación del estado del pulsador.

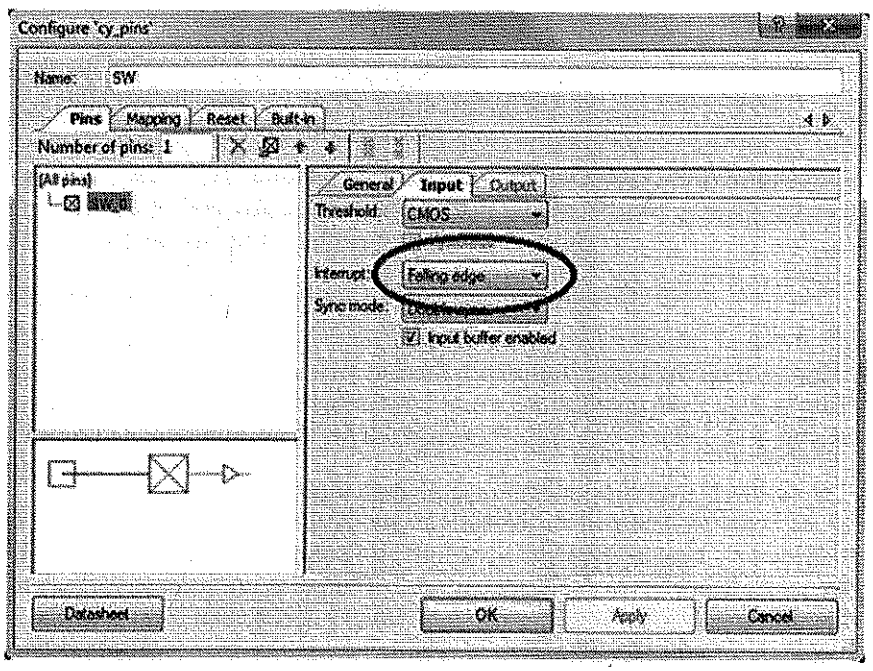
Para la implementación de la solución se hace uso de un pin de E/S configurado como entrada, con nombre SW, la opción HW connection deshabilitada, la opción External terminal habilitada, allí se conecta un interruptor del catálogo de componentes Off-Chip y la tierra del circuito; en la opción Drive mode se selecciona Resistive pull up. Finalmente, para habilitar la opción de interrupción del pin se pasa a la pestaña Input y se modifica el parámetro Interrupt, y allí se selecciona la opción Falling edge (la interrupción se genera por flanco de bajada) (ver figura 103).

Se utiliza otro bloque de E/S de un bit, configurado como salida, con el nombre LED, la opción HW connection deshabilitada y la opción External terminal habilitada; se conecta a este bloque una resistencia, un LED y la tierra del circuito. Para capturar la interrupción se requiere un bloque Interrupt, con el nombre ISR_SW. En la figura 104 se muestra el diagrama circuital de la propuesta implementada.

Se seleccionan el bit P2[1] del PSoC®5LP para conectar el LED y el bit P2[2] para la conexión del pulsador que vienen conectados de fábrica en estos pines del kit utilizado; de esta manera se facilita la implementación del ejemplo.

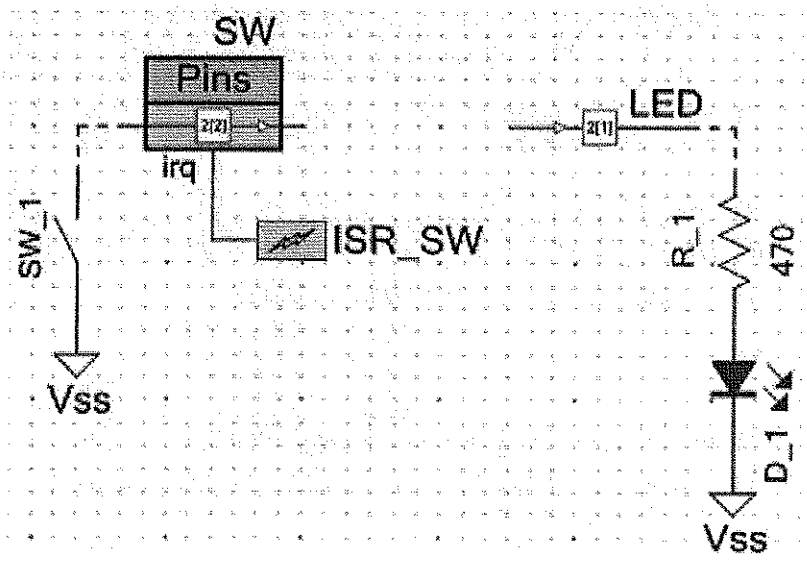
A continuación puede verse el código escrito para la solución del problema:

Figura 103. Configuración de la interrupción de un bloque de E/S



Fuente: elaboración propia.

Figura 104. Diagrama circuital del ejemplo 7-1



Fuente: elaboración propia.

Ejemplo 7-1:

```
/* ==========================================================
 *
 * Programa que permite cambiar el tiempo de oscilación de un pin
 * del PSoC. El tiempo podrá ser cambiado entre dos valores
 * posibles (50ms y 800ms) cada vez que se oprima un pulsador N.A.
 * conectado al PSoC haciendo uso de interrupciones.
 *
 * ==========================================================
 */
#include <project.h>
/* Definición de variables */
volatile uint16 tiempo;
CY_ISR(INT_SW) //Definición de rutina de interrupción
{
  if (tiempo==800){ //Verifica tiempo actual
    tiempo=50; //Actualiza valor de tiempo a 50ms
  }else{
    tiempo=800; //Actualiza valor de tiempo a 800ms
  }
  CyDelay(100); //Espera a que pulsador se estabilice
  SW_ClearInterrupt(); //Borra posibles solicitudes de interrupción
}                              //generadas por rebote o ruido
int main()
{
  CyGlobalIntEnable; /* Enable global interrupts. */
  ISR_SW_StartEx(INT_SW); //Inicializa interrupción
  tiempo=800; //Valor inicial de tiempo en 800ms
  for(;;)
  {
    CyDelay(tiempo); //Genera retardo de tiempo
    LED_Write(~LED_Read()); //Conmuta estado actual del pin
  }
}
```

La verificación del diseño y del algoritmo se realizan luego de programar el PSoC®, haciendo uso de la tarjeta de desarrollo.

Captura de interrupción en un bloque de E/S

Se pide diseñar un programa que permita cambiar el estado de 8 LED conectados a un puerto de E/S del PSoC®5LP. El cambio de estado de cada LED estará asociado a un pulsador N.A. Esto entonces implica que cada vez que se oprima algún pulsador el respectivo LED asociado a este cambiará su estado. Para el manejo del grupo de pulsadores se hará uso de las interrupciones del PSoC®.

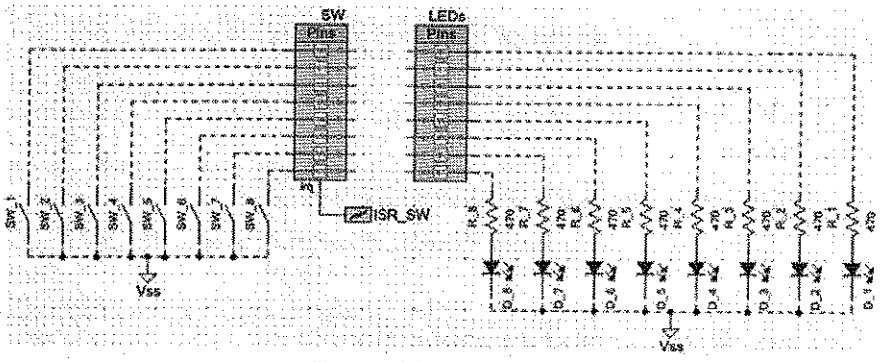
Solución

El bloque de interrupción es el corazón de la solución planteada, ya que se elimina la tarea de verificar el estado de los pulsadores, todo el algoritmo de la solución será manejado dentro de la rutina de interrupción, ya que el único proceso en el programa principal será la habilitación de las interrupción y la configuración de la utilizada en el algoritmo.

En el PSoC Creator, que permite la implementación de la solución planteada, se hace uso de un bloque de E/S de ocho bits configurado como entrada, con nombre SW, la opción HW connection deshabilitada, la opción External terminal habilitada; allí se conectan ocho interruptores y la tierra del circuito, en la opción Drive mode se selecciona Resistive pull up, y para habilitar la opción de interrupción del bloque en la pestaña Input se modifica el parámetro Interrupt, seleccionando la opción Falling edge (la interrupción de todo el bloque se genera por flanco de bajada).

Se utiliza un segundo bloque de E/S de ocho bits, configurado como salida, con el nombre LED, la opción HW connection deshabilitada y la opción External terminal habilitada. A este bloque se conectan ocho resistencias, ocho LED y la tierra del circuito. Finalmente para capturar la interrupción se hace uso de un bloque Interrupt, con el nombre ISR_SW; los demás parámetros se dejan con los valores por defecto. En la figura 105 se presenta el diagrama circuital con la solución propuesta para el ejemplo.

Figura 105. Diagrama circuital con la solución propuesta



Fuente: elaboración propia.

Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: los bits P2[7:0] para la conexión de los LED y los bits P0[7:0] para la conexión de los pulsadores.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 7-2:

```
/* ==========================================================
 *
 * Programa que permite cambiar el tiempo de oscilación de un pin
 * del PSoC. El tiempo podrá ser cambiado entre dos valores
 * posibles (50ms y 800ms) cada vez que se oprima un pulsador N.A.
 * conectado al PSoC haciendo uso de interrupciones.
 *
 * ==========================================================
 */
#include <project.h>
CY_ISR(INT_SW)
{
uint8 dato;
  dato=SW_Read(); //Lee valor de pulsadores para verificar cual
  switch (dato)     //solicitó la interrupción.
  {
    case 0xFE: //Verifica si se oprimió pulsador 1
      LEDs_Write(LEDs_Read()^0x01); //Conmuta estado de LED 1
    break;
    case 0xFD: //Verifica si se oprimió pulsador 2
      LEDs_Write(LEDs_Read()^0x02); //Conmuta estado de LED 2
    break;
    case 0xFB: //Verifica si se oprimió pulsador 3
      LEDs_Write(LEDs_Read()^0x04); //Conmuta estado de LED 3
    break;
    case 0xF7: //Verifica si se oprimió pulsador 4
      LEDs_Write(LEDs_Read()^0x08); //Conmuta estado de LED 4
```

```
        break;
        case 0xEF: //Verifica si se oprimió pulsador 5
          LEDs_Write(LEDs_Read()^0x10); //Conmuta estado de LED 5
        break;
        case 0xDF: //Verifica si se oprimió pulsador 6
          LEDs_Write(LEDs_Read()^0x20); //Conmuta estado de LED 6
        break;
        case 0xBF: //Verifica si se oprimió pulsador 7
          LEDs_Write(LEDs_Read()^0x40); //Conmuta estado de LED 7
        break;
        case 0x7F: //Verifica si se oprimió pulsador 8
          LEDs_Write(LEDs_Read()^0x80); //Conmuta estado de LED 8
        break;
        default: //Si se oprimieron más de dos pulsadores no se hace nada
        break;
      }
      while (SW_Read()!=0xFF); //Espera a que se suelte pulsador
      CyDelay(50);
      SW_ClearInterrupt(); //Borra posibles solicitudes de interrupción
    }                                //generadas por rebote o ruido.
    int main()
    {
      CyGlobalIntEnable; /* Enable global interrupts. */
      ISR_SW_StartEx(INT_SW); //Inicializa interrupción
      for(;;)
      {
        /* Place your application code here. */
      }
    }
```

Para realizar la verificación del diseño y del algoritmo se debe primero programar el PSoC®, y luego cablear cada uno de los componentes a la tarjeta de desarrollo.

Contador de objetos utilizando un sensor infrarrojo y visualización dinámica en dos displays de 7 segmentos

En el ejemplo presentado a continuación se pide implementar con el PSoC®5LP un contador de objetos que pasan en medio de un sensor de tipo óptico (pareja emisor receptor infrarrojo separados por lo menos 25cm.), se debe utilizar visualización dinámica con dos displays de 7 segmentos para mostrar la información de conteo. El contador inicia en cero al encenderse el sistema, y se incrementará cada vez que el haz de luz sea interrumpido; al llegar al máximo valor de conteo, es decir 99, el contador se reinicia, hecho que genera un conteo cíclico.

Solución

Un sensor óptico infrarrojo básicamente está conformado por una pareja emisor receptor, en la que el receptor típicamente es un optotransistor que puede ser utilizado en corte y saturación, aprovechando esta característica puede ser conectado a una entrada con interrupción para detectar el cambio de estado. Una forma de conexión que es muy utilizada para este transistor es la configuración emisor común, y se asegura así un nivel alto y bajo adecuados, pero con lógica de salida invertida; es decir, cuando la señal de luz del diodo emisor esté presente en la base del optotransistor la salida tendrá un nivel lógico bajo, si no está presente el nivel lógico será alto. Esto deberá tenerse en cuanta a la hora de configurar la interrupción en el PSoC®.

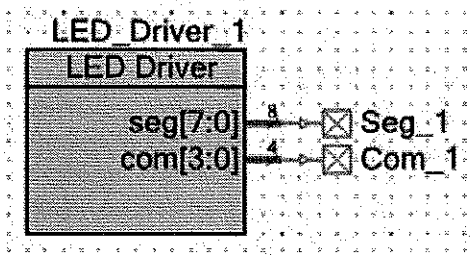
Para la visualización se pide utilizar displays de siete segmentos y además la técnica de visualización dinámica se hará uso del bloque LED_Driver, que permite realizar visualización dinámica de manera directa entre otros con displays de 7, 14 y 16 segmentos y matrices de LED, ubicado en el catálogo de componentes en la carpeta Display con el nombre 4 Digit 7-Segment LED Driver [v1.10], la información completa y detallada puede encontrarse en el datasheet del bloque llamado LED_Driver_v_1_10.pdf.

Ya que el bloque LED_Driver no ha sido presentado se hará una breve explicación del mismo para comprender el diseño sugerido. Las principales características del bloque son:

- Manejo de hasta 8 displays de 7 segmentos RGB o hasta 24 displays de 7 segmentos convencionales.
- Hasta 8 displays de 14 o 16 segmentos.
- Hasta 192 LED en matriz de 8x8 tricolor.
- Soporta señales comunes de displays o matrices activas en bajo o en alto.
- Soporta segmentos de displays o LEDs de matrices activos en bajo o en alto.
- Multiplexación no genera sobre carga a la CPU.
- Funciones para conversión de datos numéricos o tipo string para visualización directa en displays de 7, 14 o 16 segmentos.
- Control de brillo independiente para cada señal común.

En la figura 106 puede verse la imagen del bloque LED_Driver, el cual incluye de manera automática dos bloques de E/S, uno encargado del manejo de los segmentos (Seg_1) y el otro encargado de los comunes de los displays o las matrices (Com_1).

Figura 106. Bloque LED_Driver

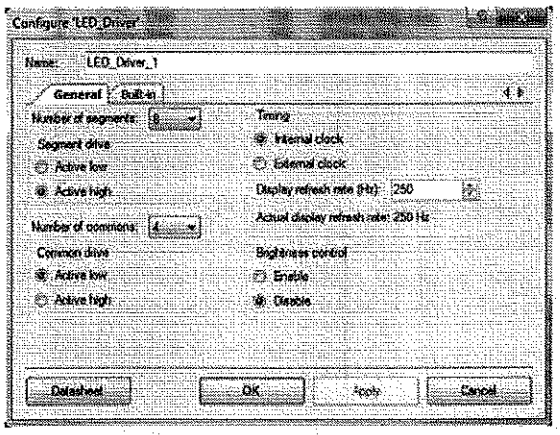


Fuente: elaboración propia.

La configuración del bloque es muy intuitiva (figura 107), en ella el usuario puede modificar entre otros los siguientes parámetros:

- Name: nombre del bloque.
- Number of segments: cantidad de segmentos a manejar (1 a 24).
- Segment drive: activo en alto o en bajo.
- Number of commons: número de pines o líneas comunes (1 a 8).
- Common drive: activo en alto o en bajo.
- Timing – Clock source: señal de reloj interna o externa
- Display refresh rate (Hz): frecuencia de refresco o barrido de los displays.
- Brightness control: se puede habilitar o deshabilitar el control de brillo de cada display.

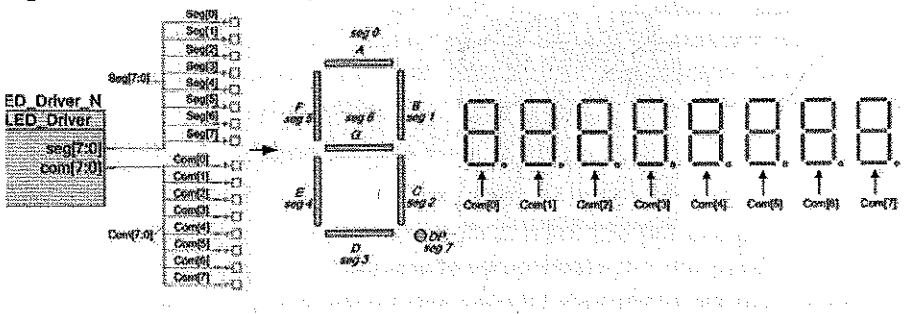
Figura 107. Ventana de configuración del bloque LED_Driver



Fuente: elaboración propia.

El orden de cada uno de los segmentos de los displays y la ubicación de estos en el arreglo está prefijado y no puede modificarse. En la figura 108 puede verse este orden para un arreglo de 8 displays de siete segmentos.

Figura 108. Orden de cada segmento de los displays y la ubicación de estos en el arreglo.



Para la sintaxis de estas funciones debe tenerse en cuenta que la primera parte del nombre de la función (LED_Driver) debe ser reemplazada por el nombre que el usuario le haya asignado al bloque. Algunas de las funciones que pueden ser utilizadas para configurar y manejar el bloque son las siguientes:

- LED_Driver_Start():

  Configura el hardware requerido por el bloque (DMA y PWM si se utiliza control de brillo) y borra los displays. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- LED_Driver_Write7SegNumberDec():

  Visualiza en todos los displays un valor codificado en decimal con signo, el signo hace uso de uno de los displays. La función utiliza cuatro parámetros de entrada: un entero con signo de 32 bits que representa el valor a visualizar, un entero sin signo de 8 bits que indica la posición desde donde se va a iniciar a visualizar la información, un entero de 8 bits sin signo que representa la cantidad de displays por utilizar y un entero de 8 bits sin signo que determina el tipo de alineación de los datos de acuerdo con lo mostrado en la tabla 17. La función no retorna ningún valor.

Nombre de la constante	Función
LED_Driver_RIGHT_ALIGN	El dígito menos significativo ocupa la posición más a la derecha.
LED_Driver_LEFT_ALIGN	El dígito más significativo está ubicado más a la izquierda.
LED_Driver_ZERO_PAD	Los dígitos de la izquierda sin valor son visualizados como ceros.

Tabla 17. Constantes de la función LED_Driver_Write7SegNumberDec()
Fuente: [Autores]

Por ejemplo, para visualizar el número 15 en dos displays de segmentos, controlados por el bloque LED_Driver de nombre Display, iniciando la visualización en el primer display, alineando el dato a la derecha se utiliza la siguiente línea de código:

```
Display_Write7SegNumberDec(15,0,2,Display_RIGHT_ALIGN);
```

- LED_Driver_Write7SegNumberHex():

  Visualiza en todos los displays un valor codificado en hexadecimal. La función utiliza cuatro parámetros de entrada: un entero sin signo de 32 bits que representa el valor por visualizar, un entero sin signo de 8 bits que indica la posición desde donde se va a iniciar a visualizar la información, un entero de 8 bits sin signo que representa la cantidad de displays por utilizar y un entero de 8 bits sin signo que determina el tipo de alineación (tabla 17). Por ejemplo, para visualizar el número 0xFA en dos displays de segmentos, controlados por el bloque LED_Driver de nombre Display, iniciando la visualización en el primer display, alineando el dato a la derecha se utiliza la siguiente línea de código:

```
Display_Write7SegNumberHex(0xFA,0,2,Display_RIGHT_ALIGN);
```

- LED_Driver_Write7SegDigitDec():

  Visualiza un digito codificado en decimal en un display de siete segmentos específico del arreglo de displays. Utiliza como parámetros de entrada un entero sin signo de 8 bits que representa el valor entre 0 y 9 a visualizar y un entero de 8 bits sin signo que representa la posición del display por utilizar. La función no retorna ningún valor. Si por ejemplo quiere visualizarse el número nueve en el display número dos en un bloque LED_Driver de nombre Display, con cuatro displays de siete segmentos, se puede utilizar el siguiente código:

```
Display_Write7SegDigitDec(9,2);
```

- LED_Driver_Write7SegDigitHex():

  Visualiza un digito codificado en hexadecimal en un display de siete segmentos específico del arreglo de displays. Utiliza como parámetros de entrada un entero sin signo de 8 bits que representa el valor entre 0x0 y 0xF a visualizar y un entero de 8 bits sin signo que representa la posición del display por utilizar. La función no retorna ningún valor.

- LED_Driver_PutDecimalPoint():

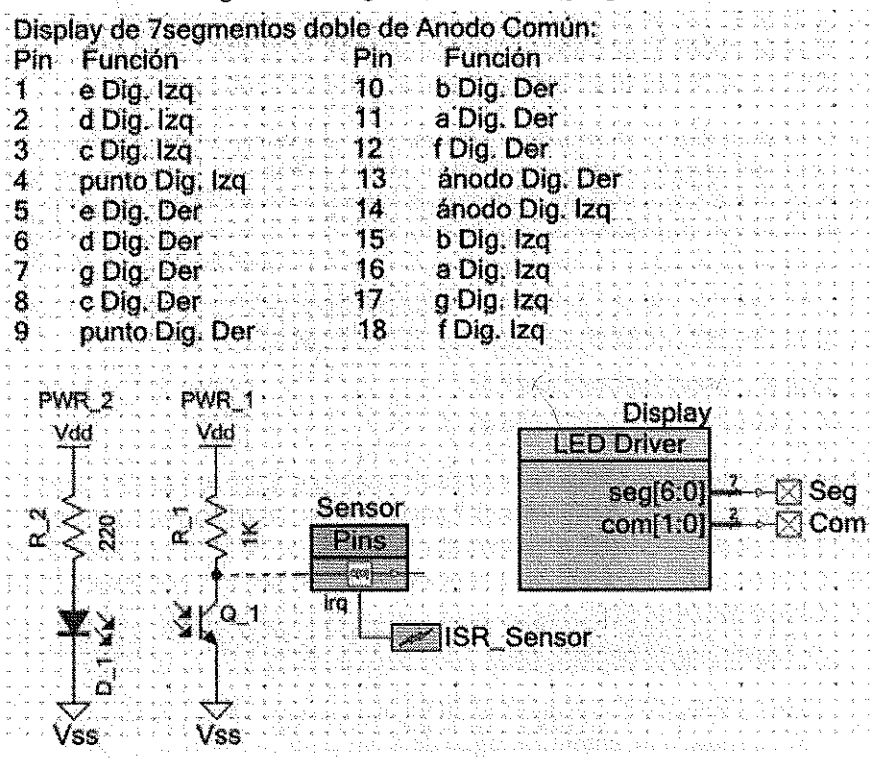
  Enciende o apaga el punto decimal en un display específico. Utiliza como parámetros de entrada un entero sin signo de 8 bits para encender o apagar el punto decimal, un valor mayor a cero lo enciende, un valor igual a cero lo apaga, y un entero de 8 bits sin signo que representa la posición del display a utilizar. La función no retorna ningún valor.

Con base en la información anterior se hace uso de tres bloques para la implementación de la solución: el primero es un bloque de E/S de un bit configurado como entrada, con nombre Sensor, la opción HW connection deshabilitada, la opción Ex-

ternal terminal habilitada; allí se conecta un sensor óptico (conformado por un diodo emisor y un optransistor), dos resistencias y la tierra del circuito, en la opción Drive mode se selecciona Resistive pull up, y para habilitar la opción de interrupción del bloque en la pestaña Input se modifica el parámetro Interrupt, seleccionando la opción Rising edge (la interrupción se genera por flanco de subida).

El segundo bloque corresponde a un LED_Driver con nombre Display, la opción Number of segments en siete (no se va a utilizar el punto decimal), Segment drive activo en bajo, Number of commons en dos (dos comunes), Common drive activo en alto, Timing – Clock source señal de reloj interna, Display refresh rate (Hz): 250Hz, Brightness control deshabilitada. El tercer y último bloque se utiliza para capturar la interrupción, se hace uso del bloque Interrupt, con el nombre ISR_Sensor, los demás parámetros se dejan con los valores por defecto. En la figura 109 se presenta el diagrama circuital con la solución propuesta para el ejemplo.

Figura 109. Diagrama circuital del ejemplo 7-3



Fuente: elaboración propia.

Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: los bits P3[6:0] para la conexión de cada uno de los segmentos del display en el bloque de E/S Seg, los bits P1[7:6] para la conexión de los comunes de los displays en el bloque de E/S Com y el bit P0[0] para la conexión del sensor.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 7-3:

```
/* ============================================================
 *
 * Programa que visualiza en dos displays de 7 segmentos el valor
 * de un contador, que se incrementa cada vez que un sensor óptico
 * conectado a una entrada del PSoC cambia de estado (cada vez que
 * se corta el haz de luz). Los displays operan utilizando la
 * técnica de visualización dinámica.
 *
 * ============================================================
*/
#include <project.h>
/* Definición de variables */
volatile uint8 cont;
CY_ISR(INT_Sensor)
{
  cont++; //Incremeta contador de objetos
  if (cont==100){cont=0;} //Reinicia contador si llega a 100
    Display_Write7SegNumberDec(cont,0,2,Display_RIGHT_ALIGN);
                                //Visualiza
                                //valor del contador en 2 displays, iniciando
                                //en pos 0, datos alineados a la derecha
  CyDelay(20);
  Sensor_ClearInterrupt(); //Borra peticiones de int. erróneas
}
int main()
{
  CyGlobalIntEnable; /* Enable global interrupts. */
  Sensor_ClearInterrupt(); //Borra posibles solicitudes iniciales
  ISR_Sensor_ClearPending();
  ISR_Sensor_StartEx(INT_Sensor); //Inicializa interrupción
```

```
    Display_Start(); //Inicializa displays de 7 segmentos
    cont=0; //Valor inicial del contador de objetos
    Display_Write7SegNumberDec(cont,0,2,Display_RIGHT_ALIGN);
                        //Visualiza
                        //valor del contador en 2 displays, iniciando
                        //en pos 0, datos alineados a la derecha
    for(;;)
    {
    }
}
```

Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC®, y luego cablear cada uno de los componentes a la tarjeta de desarrollo.

Implementación de un contador de objetos en una banda transportadora

Las bandas transportadoras son sistemas electromecánicos utilizados para el transporte de materiales de diversos tipos, las cuales aprovechan la fuerza de un motor para mover bandas o rodillos haciendo más eficiente y rápido el traslado de dichos materiales.

En el ejemplo que se muestra a continuación se implementa un contador de objetos o productos, que se movilizan por una banda transportadora hacia la zona de empaque en una fábrica típica, se debe utilizar visualización dinámica con tres displays de siete segmentos para mostrar la información del conteo. La señal por contar será generada por un sensor de tipo óptico (pareja emisor receptor infrarrojo separados por lo menos 25cm.), además se utilizará un dip switch de tres interruptores que determinará el valor de conteo máximo de acuerdo con las combinaciones binarias mostradas en la tabla 18.

Combinación	Conteo hasta
000	25
001	50
010	100
011	200
100	250
101	500
110	750
111	999

Tabla 18. Datos utilizados en el ejemplo 7-4
Fuente: elaboración propia.

Al finalizar cualquiera de las combinaciones el sistema genera una señal sonora indicando el final del conteo. Para detener la señal sonora e iniciar una nueva secuencia de conteo se debe inicializar el PSoC®; de igual manera, cada vez que se quiera cambiar el valor máximo de conteo se debe realizar esta tarea [8].

Solución

La solución puede basarse en la implementada en el ejemplo anterior, realizando las modificaciones pertinentes. Se aprovecha entonces la ventaja del uso del bloque LED_Driver para la visualización dinámica y el proceso de lectura del sensor se realiza por interrupción. Para determinar el valor de conteo se incluye un bloque de E/S donde se conectará el dip switch que generará la combinación correspondiente.

Para formalizar el diseño se hace uso de cinco bloques para su implementación: el primero es un bloque de E/S de un bit configurado como entrada, con nombre Sensor, la opción HW connection deshabilitada, la opción External terminal habilitada, donde se conecta un sensor óptico, dos resistencias, la fuente de alimentación y la tierra del circuito; en la opción Drive mode se selecciona Resistive pull up, y para habilitar la interrupción del bloque en la pestaña Input se modifica el parámetro Interrupt, seleccionando la opción Rising edge (la interrupción se genera por flanco de subida).

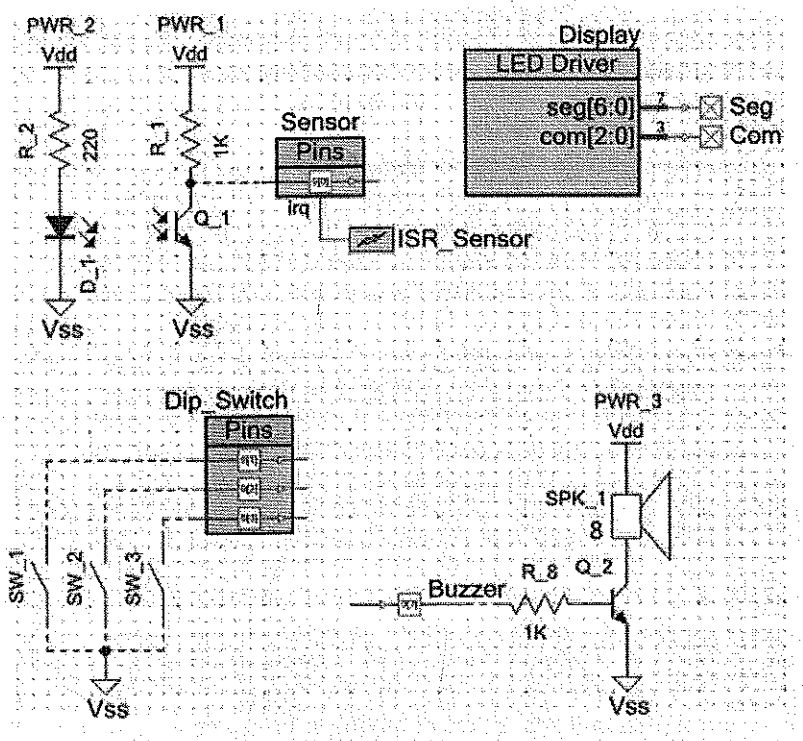
El segundo bloque corresponde a un LED_Driver con nombre Display, la opción Number of segments en siete (no se utiliza el punto decimal), Segment drive activo en bajo, Number of commons en tres (tres comunes), Common drive activo en alto, Timing – Clock source señal de reloj interna, Display refresh rate (Hz): 250Hz, Brightness control deshabilitada. El tercer bloque se utiliza para capturar la interrupción, para lo cual se hace uso del bloque Interrupt, con el nombre ISR_Sensor, los demás parámetros se dejan con los valores por defecto.

El cuarto bloque corresponde a uno de E/S de tres bits configurado como entrada, con nombre Dip_Switch, la opción HW connection deshabilitada, la opción External terminal habilitada, donde se conectan tres interruptores del dip switch y la tierra del circuito y en la opción Drive mode se selecciona Resistive pull up. El quinto y último corresponde a un bloque de E/S de un bit configurado como salida, con nombre Buzzer, la opción HW connection deshabilitada y la opción External terminal habilitada, donde se conectan una resistencia, un transistor, un buzzer, la fuente de alimentación y la tierra del circuito. En la figura 110 puede verse el diagrama circuital con la solución completa.

Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: los bits P3[6:0] para la conexión de cada uno de los segmentos del display en el bloque de E/S Seg, los bits P1[7:6] para la conexión de los comunes de los displays en el bloque de E/S Com, el bit P0[0] para la conexión del sensor, los bits P0[3:1] para la conexión del dip switch al bloque de E/S Dip_Switch y el bit P0[7] para la conexión del buzzer al bloque de E/S Buzzer.

A continuación puede verse el código escrito para la solución del problema:

Figura 110. Diagrama circuital ejemplo banda trasportadora



Fuente: elaboración propia.

Ejemplo 7-4:

```
/* ==========================================================
 *
 * Programa que visualiza en tres displays de 7 segmentos el valor
 * de un contador, que se incrementa cada vez que un sensor óptico
 * conectado a una entrada del PSoC cambia de estado, el valor de
 * conteo máximo depende del estado de un dip switch. Los displays
 * operan utilizando la técnica de visualización dinámica.
 *
 * ==========================================================
 */
#include <project.h>
/* Definición de variables */
volatile uint16 cont,secuencia;
```

```
uint16 Tabla[]={25,50,100,200,250,500,750,999};
CY_ISR(INT_Sensor)
{
  cont++; //Incrementa contador de objetos
  if (cont>secuencia){//Si llega al valor máximo
    Buzzer_Write(1); //Enciende el buzzer
    for(;;){}
  }
  Display_Write7SegNumberDec(cont,0,3,Display_RIGHT_ALIGN);
                              //Visualiza
                              //valor del contador en 2 displays, iniciando
                              //en pos 0, datos alineados a la derecha
  CyDelay(100);
  Sensor_ClearInterrupt(); //Borra peticiones de int. erróneas
}
int main()
{
  CyGlobalIntEnable; /* Enable global interrupts. */
  Sensor_ClearInterrupt(); //Borra posibles solicitudes iniciales
  ISR_Sensor_ClearPending();
  ISR_Sensor_StartEx(INT_Sensor); //Inicializa interrupción
  Display_Start(); //Inicializa displays de 7 segmentos
  cont=0; //Valor inicial del contador de objetos
  Display_Write7SegNumberDec(cont,0,3,Display_RIGHT_ALIGN);
  secuencia=Dip_Switch_Read(); //Lee valor del dip switch
  secuencia=Tabla[secuencia]; //Determina valor de conteo máximo
  for(;;)
  {
  }
}
```

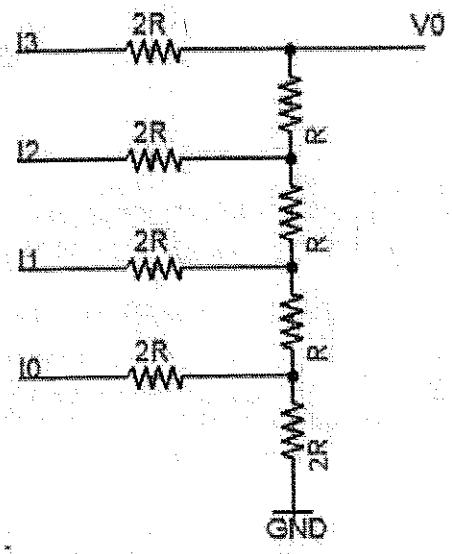
Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC®, y luego cablear cada uno de los componentes a la tarjeta de desarrollo.

Conversor analógico a digital del PSoC5®LP

El procesamiento de señales es una tarea donde un sistema que incluye algún tipo de procesador es primordial. Estas señales pueden provenir de sensores que al observar su comportamiento permiten generar una serie de acciones por parte del procesador. Dos procesos son importantes para lograrlo: la conversión analógica a digital (ADC) y la conversión digital a analógica (DAC), ya que permiten la interacción entre el mundo analógico y el mundo digital.

El proceso de conversión digital a analógica consiste en transformar un código digital en un nivel de voltaje analógico. El proceso de conversión analógica a digital se encarga de codificar en forma digital niveles de voltaje analógico que están dentro de un rango preestablecido. Un método sencillo de conversión de digital a analógico es por medio de una red resistiva denominada escalera R-2R (figura 111). Para esta implementación se utilizan resistencias de dos valores diferentes, una con el doble del valor de la otra, pudiéndose así implementar un DAC de n bits.

Figura 111. DAC por escalera R-2R



Fuente: [8, p. 139].

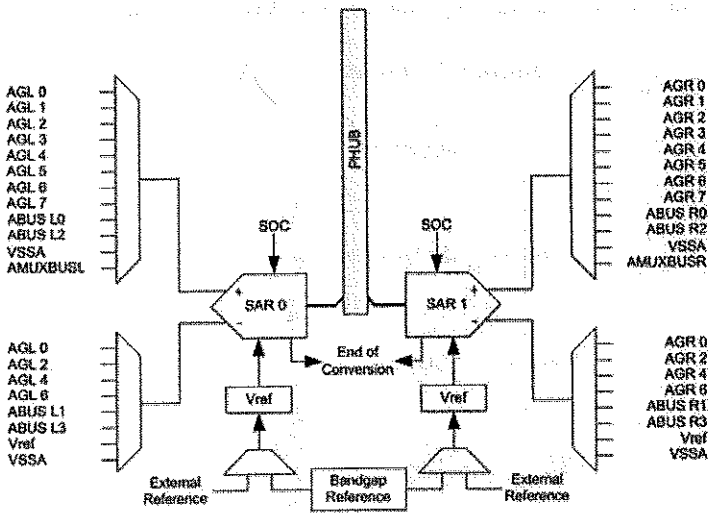
Un conversor analógico a digital incluye normalmente un circuito de conversión digital a analógico, la idea es producir códigos digitales que ingresan a través del DAC y se comparan con la señal analógica que se desea codificar. Dependiendo del resultado de la comparación, se toma alguna acción sobre el código para cambiarlo a un valor más cercano, hasta cuando no sea posible hacerlo. Por otro lado, el número de niveles que se puede lograr depende del número de bits que se utilicen en el código digital; así, el número de niveles que se podría obtener seria 2n donde n es el número de bits de entrada.

Existen varios métodos para generar los códigos digitales y para cambiarlos después de la comparación, los más conocidos son el *flash* (o de centelleo), de rampa, doble rampa y aproximaciones sucesivas. En los dos primeros, el tiempo de conversión (intervalo de tiempo necesario para que se produzca una conversión) es variable y depende del nivel de la señal que se desea codificar. En el tercer método este tiempo es el mismo para cualquier nivel, aunque el tiempo de retardo en la generación de los códigos puede ser ligeramente mayor en el método de aproximaciones sucesivas [8].

ADC por aproximaciones sucesivas

El bloque conversor analógico a digital por aproximaciones sucesivas (ADC_SAR) provee velocidades de conversión y resolución media útil en aplicaciones como controles de iluminación con LED, control de motores, lectores de tarjetas magnéticas, recolección de datos, medidores de potencia eléctrica, pulsioximetros, entre otros.

Figura 112. Diagrama de bloques del ADC por aproximaciones sucesivas



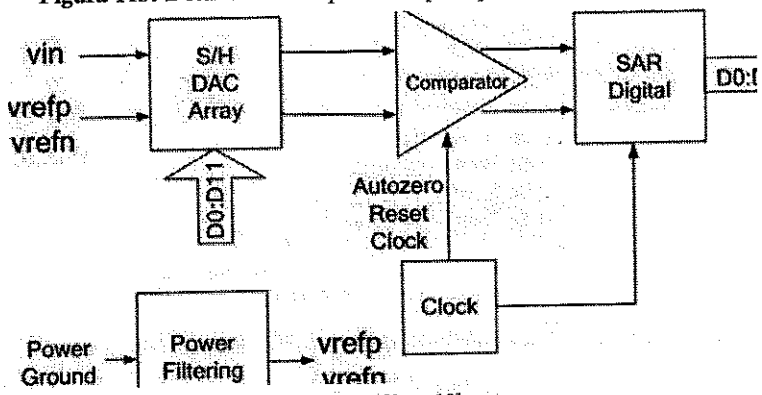
Fuente: [2, p. 387].

En la figura 112 puede verse el diagrama de bloques del ADC por aproximaciones sucesivas incluido en el PSoC®5LP, el cual tiene las siguientes características:

- 2 bloques SAR.
- Resolución de 8, 10 o 12 bits.
- Velocidad de muestra máxima de 1Msps.
- Fuente de reloj interna o externa.
- Tres modos de muestreo.
- Conversión única o continua.
- Entrada unipolar o diferencial.
- Rango de entrada de voltaje configurable.

En esencia el bloque ADC por aproximaciones sucesivas realiza el proceso de muestrear y comparar la señal analógica de entrada con la salida de un DAC que utiliza un valor binario generado por un algoritmo de búsqueda, como se muestra en la figura 113.

Figura 113. Detalles del bloque ADC por aproximaciones sucesivas



Fuente: [9, p. 19].

Para seleccionar en el PSoC Creator el bloque ADC por aproximaciones sucesivas denominado *SAR_ADC [v3.0]* (figura 114), el usuario debe buscar la carpeta *Analog* del catálogo de componentes, subcarpeta ADC y seleccionarlo entre las tres opciones disponibles allí, datasheet del bloque tiene el nombre de *ADC_SAR_v3_0.pdf.*

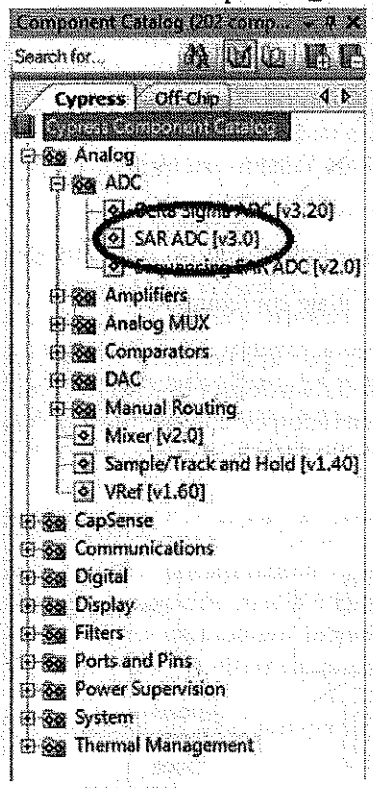
Señales de entrada/salida del bloque ADC por aproximaciones sucesivas

En la figura 115 puede verse la imagen del bloque ADC por aproximaciones sucesivas con todas las posibles señales de entrada/salida soportadas por el bloque, algunas de ellas pueden ser eliminadas en la configuración si no son necesarias para la aplicación.

- + (Analog Input):

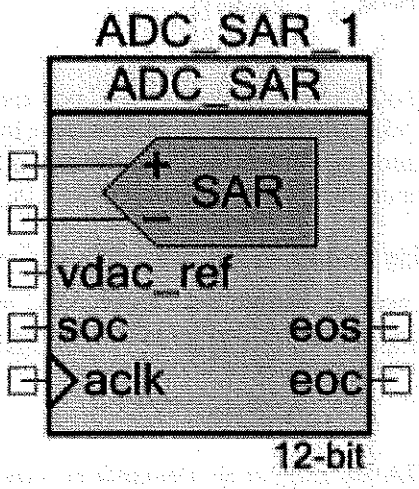
  Es la entrada positiva de la señal analógica al ADC.

Figura 114. Selección del bloque SAR_ADC [v3.0]



Fuente: elaboración propia.

Figura 115. Señales de entrada/salida del bloque ADC por aproximaciones sucesivas



Fuente: elaboración propia.

- - (Analog Input):

  Entrada negativa o de referencia al ADC. Esta entrada puede ser eliminada.

- vdac_ref (input):

  Voltaje de referencia externo denominado VDAC. Esta entrada puede ser eliminada.

- soc (input):

  Señal de inicio de conversión (*start of conversion*). Un flanco de subida externo genera el inicio de la conversión, la entrada puede ser sincronizada con la señal de reloj interna del ADC. Esta entrada puede ser eliminada.

- aclk (input):

  Señal de reloj externa que determina la tasa de conversión del ADC. Esta entrada puede ser eliminada.

- eos (output):

  Esta señal indica final de muestro (*end of sampling*) por medio de un flanco de subida; esta señal puede ser utilizada para el control de canales de entrada de un multiplexor. Esta salida puede ser eliminada.

- eoc (output):

  Esta señal indica final de conversión (*end of conversión*) por medio de un flanco de subida, puede ser utilizada por un canal DMA para tener acceso a este o también a un bloque de interrupción.

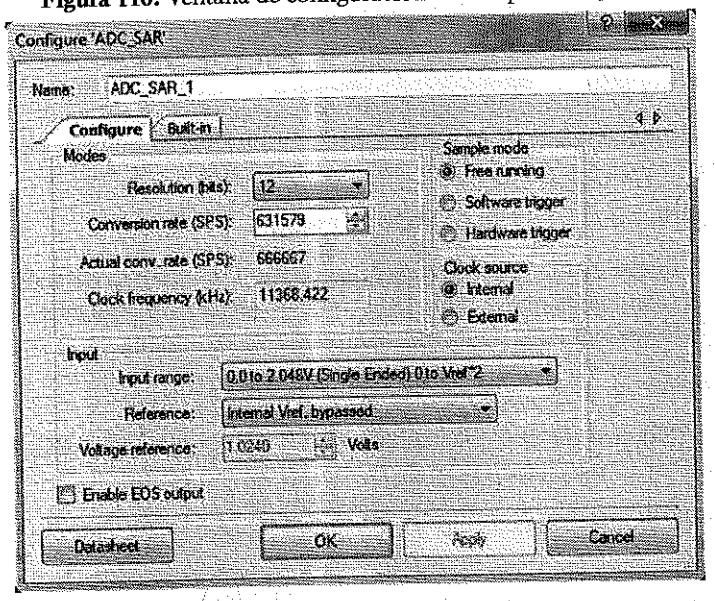
Configuración del bloque ADC por aproximaciones sucesivas

La ventana de configuración del bloque ADC_SAR permite modificar o configurar cinco parámetros generales: nombre (*Name*), modo de operación general del ADC (*Modes*), modo de muestreo (*Sample mode*), fuente de reloj (*Clock source*) y el rango del voltaje de entrada (*Input*). En la figura 116 puede verse la ventana de configuración del bloque SAR_ADC con las funcionalidades antes nombradas.

El parámetro Modes permite modificar dos valores de forma directa y un valor de manera indirecta en la configuración del ADC:

- *Resolution (bits)*: permite seleccionar entre tres valores posibles de resolución para el ADC: 8, 10 o 12 bits (valor por defecto).
- *Conversion rate (SPS)*: determina la tasa de conversión del ADC medida en muestras por segundo, el tiempo de conversión corresponde al inverso de este valor. En algunas ocasiones el valor configurado no puede ser generado por el reloj del PSoC®; como herramienta de ayuda se presenta el valor real de conversión logrado en la casilla denominada Actual conv. Rate (SPS).

Figura 116. Ventana de configuración del bloque ADC_SAR



Fuente: elaboración propia.

- *Clock frecuency (kHz)*: es una casilla de solo lectura que representa la frecuencia de la señal de reloj requerida para lograr la tasa de conversión seleccionada; el valor cambia de forma automática al modificar la tasa de conversión.

El parámetro *Sample Mode* determina el tipo de muestreo de la señal por convertir, entre conversión única y conversión continua:

- *Free runnig*: el ADC realiza conversión de manera continua, solamente requiere una instrucción del software para iniciar el proceso de manera cíclica, para detener la conversión también se hace uso de una instrucción del software.
- *Software trigger*: se realiza una única conversión cuando se envía la petición por medio de una instrucción del software.
- *Hardware trigger*: al seleccionar esta opción se habilita una entrada en el bloque denominada SOC que permite por medio de una señal externa (flanco de bajada) iniciar una única conversión.

El parámetro *Clock source* determina la fuente de reloj utilizada por el ADC entre dos posibles:

- *Internal*: utiliza una señal de reloj interna generada en el PSoC® como base de tiempo para el ADC.
- *External*: utiliza una señal externa como fuente de reloj para el ADC.

El parámetro *Input* permite configurar las características de voltaje de la señal analógica de entrada al ADC:

- *Input range*: selecciona el rango de entrada de la señal por convertir (tabla 19); independientemente del ajuste de rango seleccionado la señal analógica debe estar en la ventana entre VSSA y VDDA.

Rango de entrada	Descripción
0.0 to 2.048V (Single Ended)	Rango de entrada de la señal entre 0,0 y 2,048V cuando la señal de referencia es interna (1.024V).
0 to Vref*2	Rango de entrada de la señal entre 0,0 y Vref*2 cuando la señal de referencia es externa.
Vssa to Vdda (Single Ended)	Rango de la señal de entrada entre VSSA y VDDA cuando la señal de referencia es interna (VDDA/2).
Vssa to VDAC*2 (Single Ended)	Se utiliza como referencia la señal VDAC, el rango de la señal de entrada esta entre VSSA y VDAC*2.
0.0 ± 1.024V (Differential) -Input ± Vref	Rango diferencial de entrada de la señal entre –Input y ±1,024V cuando la señal de referencia es interna (1,024V).
0.0 ± Vdda (Differential) -Input ± Vdda	Rango diferencial de la señal de entrada entre VSSA y VDDA cuando se usa referencia interna (VDDA). Rango diferencial de entrada de la señal entre –Input y ±Vref cuando la señal de referencia es externa.
0.0 ± Vdda/2 (Differential) -Input ± Vdda/2	Rango diferencial de la señal de entrada entre VSSA y VDDA/2 cuando se usa referencia interna (VDDA/2). Rango diferencial de entrada de la señal entre –Input y ±Vref cuando la señal de referencia es externa.
0.0 ± VDAC (Differential) -Input ±VDAC	Rango diferencial de la señal de entrada entre VSSA y VDAC cuando se usa referencia interna (VDAC). Rango diferencial de la señal de entrada entre entre –Input y ±Vref cuando la señal de referencia es externa.

Tabla 19. Rangos de entrada al ADC por aproximaciones sucesivas
Fuente: [9]

- *Reference*: selecciona la fuente de referencia utilizada por el ADC, entre tres posibles, de acuerdo con lo mostrado en la tabla 20.

Fuente de referencia	Frecuencia de reloj permitida	Descripción
Internal Vref	1 – 1.6 MHz	ADC usa valor de referencia interno dependiendo del rango del voltaje de entrada seleccionado, excepto para señales entre 0.0 ± Vdda.
	1 – 9 MHz	ADC usa valor de referencia interno dependiendo del rango del voltaje de entrada seleccionado, válida para señales entre 0.0 ± Vdda.
Internal Vref, bypassed	1 -18 MHz	ADC usa valor de referencia interno dependiendo del rango del voltaje de entrada seleccionado, excepto para señales entre 0,0 ± Vdda. Debe conectarse un condensador entre 0,01uf y 10uF en el pin P0[2] para el SAR1 o en el pin P0[4] para el SAR2.
External Vref	1 -18 MHz	ADC usa valor de referencia externo en el pin P0[2] para el SAR1 o en el pin P0[4] para el SAR2.

Tabla 20. Posibles referencias para el ADC por aproximaciones sucesivas
Fuente: [9].

- *Voltaje reference*: muestra el valor del voltaje de referencia interna cuando esta es la fuente de referencia del ADC. Cuando se selecciona el valor de referencia externa refleja el valor seleccionado por el usuario.

El último parámetro *Enable EOS output* habilita/deshabilita una señal de salida que indica la finalización del muestreo de la señal.

Rutinas utilizadas para el manejo del bloque ADC_SAR

Las funciones API permiten realizar la configuración y el manejo del bloque ADC por aproximaciones sucesivas. Para la sintaxis de estas funciones debe tenerse en cuenta que la primera parte del nombre de la función (ADC_) debe ser reemplazado por el nombre que el usuario le haya asignado al bloque.

Las funciones que pueden ser utilizadas para configurar y manejar el bloque son:

- ADC_Start():

  Inicializa variables e inicializa y habilita todos los parámetros de configuración seleccionados previamente para el ADC. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- ADC_SetResolution():

  Modifica la resolución del ADC entre los tres valores posibles que soporta: 8, 10 o 12 bits. Este parámetro debe ser modificado con el conversor detenido.

Como parámetro de entrada utiliza un entero sin signo de 8 bits con alguno de los valores numéricos 8, 10 o 12 que representan la resolución seleccionada. La función no retorna ningún valor. Si por ejemplo se requiere configura una resolución de 10 bits en bloque de *ADC_SAR* denominado *ADC* se utilizaría la siguiente línea de código:

```
ADC_SetResolution(10);
```

- ADC_StartConvert():

Hace que el ADC inicie una conversión, si está configurado para conversión continua finalizará una y automáticamente iniciará una nueva. En modo de conversión por software realiza el proceso una vez. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- ADC_IsEndConversion():

Retorna el estado de la conversión del ADC de acuerdo con el parámetro de entrada utilizado (ver tabla 21), que corresponde a un entero sin signo de 8 bits. Retorna un dato de 8 bits sin signo que representa el estado solicitado.

Opción	Descripción
ADC_RETURN_STATUS	Retorna el estado del ADC entre dos posibles: si el valor retornado es cero indica que la conversión no ha finalizado, si es no cero indica que la conversión actual ha terminado.
ADC_WAIT_FOR_RESULT	Espera hasta que el ADC finalice una conversión.

Tabla 21. Parámetros utilizados para conocer estado de conversión del ADC
Fuente: [9].

- ADC_GetResult8():

Retorna el resultado de la conversión en formato de 8 bits. Si la resolución del ADC es mayor a 8 bits la función retorna el valor de la parte menos significativa de la conversión. No utiliza ningún parámetro de entrada. Retorna un valor de 8 bits con signo que representa el valor de la última conversión; por ejemplo, si se quiere leer (enviando el resultado a la variable *dato*) el resultado de un bloque *ADC_SAR* denominado *ADC*, configurado con resolución de 8 bits, se utilizaría el siguiente código:

```
dato=ADC_GetResult8();
```

- ADC_GetResult16():

Retorna el resultado de la conversión en formato de 16 bits, sin importar la resolución del ADC. No utiliza ningún parámetro de entrada. Retorna un valor de 16 bits con signo que representa el valor de la última conversión.

- ADC_CountsTo_Volts():

  Convierte el resultado generado por el ADC a un valor en voltios codificado en punto flotante. Utiliza como parámetro de entrada un entero de 16 bits con signo que almacena el resultado generado por el ADC. Retorna un valor en voltios en punto flotante equivalente al dato de entrada.

- ADC_CountsTo_mVolts():

  Convierte el resultado generado por el ADC a un valor en milivoltios. Utiliza como parámetro de entrada un entero de 16 bits con signo que almacena el resultado generado por el ADC. Retorna un valor en milivoltios como un entero de 16 bis con signo.

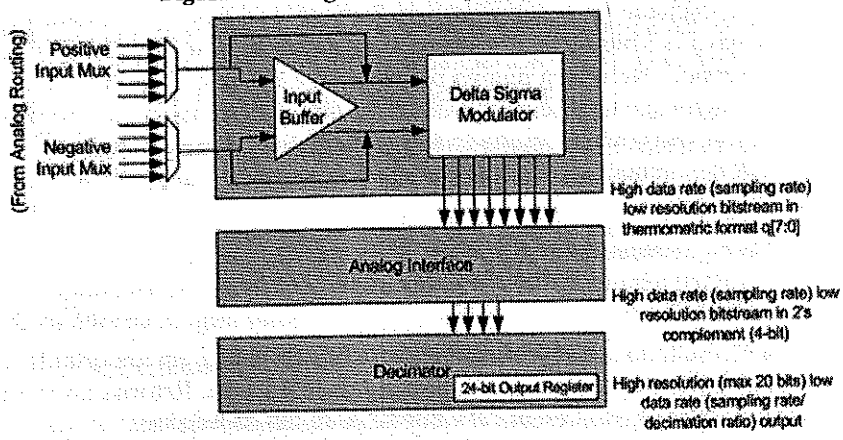
- ADC_CountsTo_uVolts():

  Convierte el resultado generado por el ADC a un valor en microvoltios. Utiliza como parámetro de entrada un entero de 16 con signo que almacena el resultado generado por el ADC. Retorna un valor en microvoltios como un entero de 32 bis con signo.

ADC delta sigma

El bloque conversor analógico a digital delta sigma (ADC_DelSig) provee una alta SNR/resolución por *oversampling*, ruido de forma, promediación y decimación, y además una alta precisión. Está conformado básicamente por un modulador y un decimador (figura 117), el primero se encarga de convertir la señal analógica de entrada a una alta velocidad de datos (*oversampling*) y una baja resolución (normalmente un bit) con lo que se obtiene el valor medio de la señal. Este flujo de daos se trasmite a través de un filtro de decimación (decimador) para obtener la salida a una baja tasa de bits, pero con una alta resolución. El filtro decimador es una combinación de un submuestreador y un filtro pasa bajo.

Figura 117. Diagrama de bloques ADC delta sigma



Fuente: [2, p. 367].

El ADC delta sigma tiene las siguientes características:

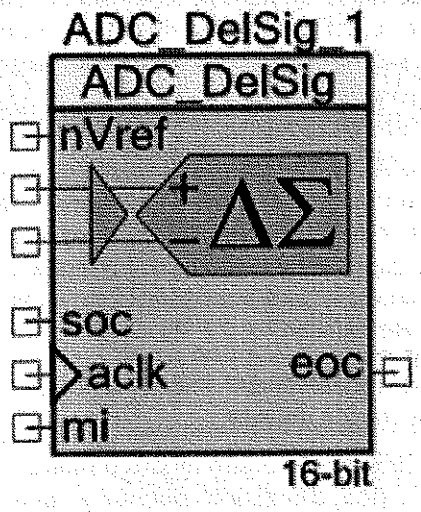
- Resolución de 8 a 20 bits.
- 11 rangos de entrada para cada resolución.
- Muestreo entre 8sps y 384ksps.
- Entrada de voltaje unipolar o diferencial.
- Cuatro modos de operación: conversión única, multi-conversión, conversión continua y multi-conversión turbo.
- Buffer de entrada de alta impedancia con ganancia seleccionable entre 1,2 4 y 8.
- Opciones de referencia interna o externa.
- Configuración de potencia automática.
- Corrección de ganancia y *offset*.

El ADC delta sigma puede ser utilizado en diversas aplicaciones que requieran alta resolución, con tasas de muestreo medias; por ejemplo, para aplicaciones de audio que necesitan resolución de 16 bits o sensores que requieren resoluciones mayores a velocidades de operación bajas.

Señales de entrada/salida del ADC delta sigma T2

La imagen de la figura 118 muestra el bloque ADC delta sigma con todas las posibles señales de entrada/salida soportadas por el bloque, algunas de ellas pueden ser eliminadas en la configuración si no son necesarias para la aplicación

Figura 118. Señales de entrada/salida del ADC delta sigma



Fuente: elaboración propia.

- nVref (input):

  Conecta la referencia del ADC a la señal VSSA general del PSoC. Esta entrada puede ser eliminada.

- + (Analog Input):

  Es la entrada positiva de la señal analógica al ADC.

- - (Analog Input):

  Entrada negativa al ADC en modo diferencial. Esta entrada puede ser eliminada.

- soc (input):

  Señal de inicio de conversión (start of conversion). Un flanco de subida externo genera el inicio de la conversión. Esta entrada puede ser eliminada.

- aclk (input):

  Señal de reloj externa que determina la tasa de conversión del ADC, la señal debe tener un ciclo útil del 50%. Esta entrada puede ser eliminada

- eoc (output):

  Esta señal indica final de conversión (end of conversión) por medio de un flanco de subida; puede ser utilizada por un canal DMA para tener acceso a este o también a un bloque de interrupción.

- mi (input):

  Señal utilizada para controlar de manera dinámica la polaridad de la señal de entrada al modulador del ADC. La polaridad se invierte cuando la señal mi (modulador input) está en nivel lógico alto. Esta entrada puede ser eliminada

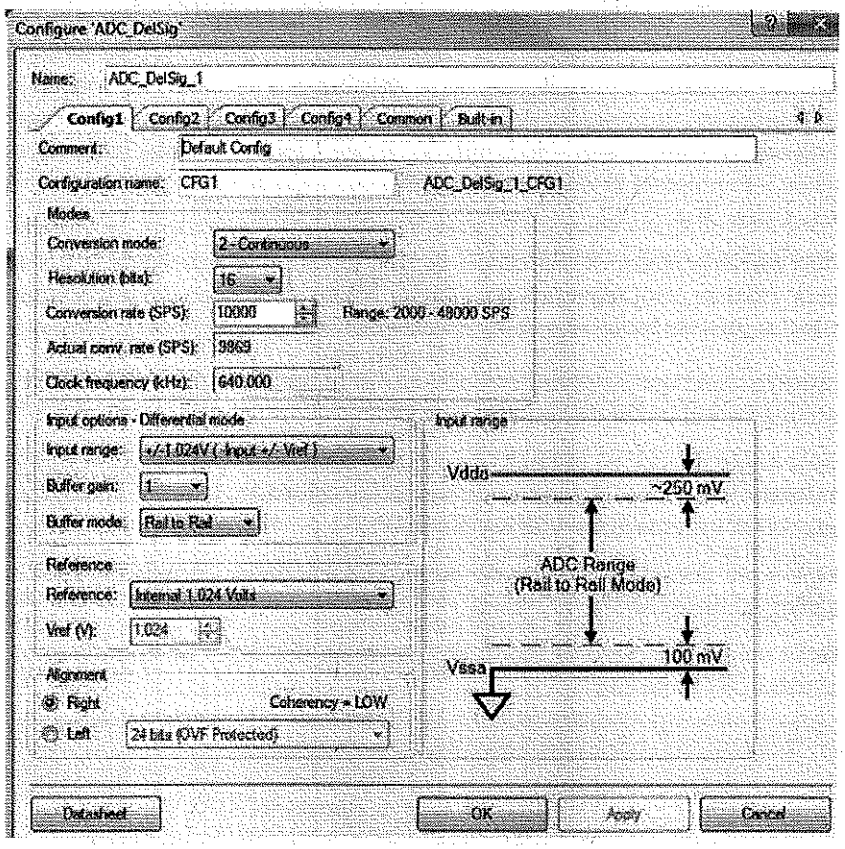
Configuración del bloque ADC delta sigma

La configuración del bloque ADC_DelSig se realiza en dos pestañas, la primera permite modificar o configurar cinco parámetros generales: nombre (*Name*), modo de operación general del ADC (*Modes*), opciones de entrada (*Input options*), (*Input*), el tipo de señal de referencia (*Reference*) y la forma de presentación del resultado (*Aligment*). En la figura 119 puede verse la ventana de configuración del bloque *ADC_DelSig* con la pestaña *Config1* seleccionada.

El parámetro *Modes* permite modificar dos valores de forma directa y un valor en forma indirecta en la configuración del ADC:

- *Conversion mode*: permite entre los cuatro modos posibles de conversión: conversión única, multi-conversión, conversión continua y multi-conversión turbo. En la tabla 22 se describe cada uno de los modos.

Figura 119. Ventana de configuración del bloque ADC_Del·Sig



Fuente: elaboración propia.

Modo	Descripción
0 – Single Sample	Realiza una conversión luego de recibir la señal correspondiente.
1 – Multi-Sample	Toma muestras de múltiples señales analógicas de entrada.
2 – Continuos	Realiza conversión de manera continua; requiere una primera señal de inicio de conversión, y a partir de esta realiza la conversión continuamente al tener un resultado. Modo por defecto.
3 – Multi-Sample (Turbo)	Toma muestras de múltiples señales analógicas de entrada; este modo de operación se utiliza para señales que requieran resoluciones entre 8 y 16 bits.

Tabla 22. Modos de conversión del ADC delta sigma
Fuente: elaboración propia

- *Resolution (bits)*: permite seleccionar la resolución del ADC entre 8 y 20 bits (el valor por defecto es 16 bits).
- *Conversion rate (SPS)*: determina la tasa de conversión del ADC medida en muestras por segundo. Dependiendo de la resolución y el modo de conversión se tiene un rango de muestreo diferente (en la venta de configuración se muestra el rango de acuerdo con la selección). Se cuenta además con una herramienta de ayuda y se presenta el valor real de conversión logrado en la casilla denominada *Actual conv. Rate (SPS)*.
- *Clock frecuency (kHz)*: es una casilla de solo lectura que representa la frecuencia de la señal de reloj requerida para lograr la tasa de conversión seleccionada, el valor cambia de forma automática al modificar la tasa de conversión.

El parámetro *Input options* permite configurar el rango del voltaje de entrada de la señal analógica y el buffer de entrada del ADC:

- *Input range*: selecciona el rango de entrada de la señal por convertir (tabla 19). Independientemente del ajuste de rango seleccionado, la señal analógica debe estar en la ventana entre VSSA y VDDA.
- *Buffer gain*: selecciona el valor de ganancia del *buffer* de entrada del ADC entre cuatro valores posibles: 1, 2 4 u 8.
- *Buffer mode*: selecciona el modo de operación del *buffer* de entrada entre tres posibles (ver tabla 23).

Buffer mode	Descripción
Bypass Buffer	Deshabilita el *buffer* de entrada del ADC, y reduce así el consumo de potencia del PSoC®. La señal de entrada puede estar en el rango de VSSA-100mV a VDDA+100mV
Rail to Rail	La señal de entrada puede estar en el rango de VSSA+100mV a VDDA-250mV
Level Shift	Se hace uso tanto del *buffer* positivo como del negativo de entrada. La señal de entrada puede estar en el rango de VSSA-100mV a VDDA-650mV

Tabla 23. Modos de operación del buffer de entrada del ADC delta sigma
Fuente: elaboración propia.

Modificando los valores del parámetro general *Reference* es posible cambiar la fuente y el valor de referencia del ADC:

- *Reference*: selecciona la fuente de referencia utilizada por el ADC, entre múltiples posibilidades (tabla 24).

Referencia del ADC	Descripción
Internal Vref 1.024 Volts	Utiliza el valor 1024V como voltaje de referencia interno.
Internal Vref, Bypassed on P0[3]	Utiliza el valor 1024V como voltaje de referencia interno con un condensador bypass conectado en el pin P0[3].
Internal Vref, Bypassed on P3[2]	Utiliza el valor 1024V como voltaje de referencia interno con un condensador bypass conectado en el pin P3[2].
External Vref on P0[3]	Utiliza un voltaje de referencia externo conectado al pin P0[3].
External Vref on P3[2]	Utiliza un voltaje de referencia externo conectado al pin P3[2].
Internal Vdda/4	Usa VDDA/4 como voltaje de referencia
Internal Vdda/4, Bypassed on P0[3]	Usa VDDA/4 como voltaje de referencia con un condensador bypass conectado en el pin P0[3].
Internal Vdda/4, Bypassed on P3[2]	Usa VDDA/4 como voltaje de referencia con un condensador bypass conectado en el pin P3[2].
Internal Vdd/3	Usa VDDA/3 como voltaje de referencia
Internal Vdda/3, Bypassed on P0[3]	Usa VDDA/3 como voltaje de referencia con un condensador bypass conectado en el pin P0[3].
Internal Vdda/3, Bypassed on P3[2]	Usa VDDA/3 como voltaje de referencia con un condensador bypass conectado en el pin P3[2].

Tabla 24. Fuentes de voltaje de referencia utilizadas por el ADC delta sigma
Fuente: elaboración propia.

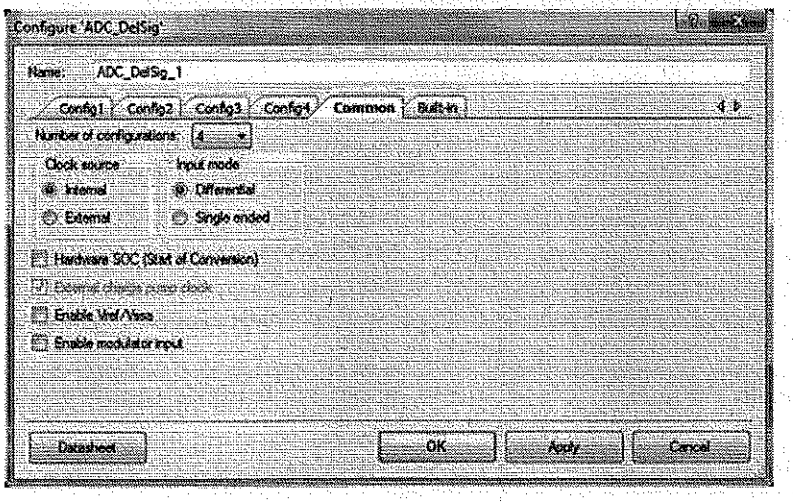
- *Vref*: muestra el valor del voltaje de referencia interna cuando esta es la fuente de referencia del ADC. Cuando se selecciona el valor de referencia externa refleja el valor seleccionado por el usuario.

Con el parámetro *Aligment* se puede seleccionar el modo de presentación del resultado de la conversión; normalmente el resultado de la conversión está ajustado a la derecha, pero en algunos casos se requiere un ajuste particular que depende de la aplicación.

Además, de acuerdo con la selección realizada se modifica de forma automática el parámetro coherencia (*Coherency*) entre tres valores posibles, *LOW, MID y HIGH*, que como su nombre lo indica le da coherencia al resultado de la conversión, ya que el valor de la conversión tiene siempre una longitud fija de 24 bits.

La segunda pestaña denominada *Common* permite realizar otras configuraciones generales del ADC delta sigma (ver figura 120).

Figura 120. Pestaña de configuración *Common* del ADC delta sigma



Fuente: elaboración propia.

El parámetro *Number of configurations* permite realizar hasta cuatro configuraciones diferentes para el ADC, las cuales pueden ser utilizadas durante la ejecución de la aplicación dependiendo de las necesidades de esta. Para hacer este cambio de configuración se hace uso de una función API del ADC.

Para seleccionar la fuente de reloj del ADC (común para todas las configuraciones), se utiliza el parámetro *Clock source*, que permite seleccionar ente dos opciones: *Internal* (valor por defecto) o *External*; en este último caso la señal de reloj debe tener un ciclo útil del 50% y ser muy estable para garantizar bajo ruido en la operación del ADC.

Con el parámetro *Input mode* se selecciona el tipo de señal analógica por convertir entre dos tipos posibles: *Differential* (valor por defecto) y *Single ended*. Esto genera en el primer caso un resultado de conversión con signo y en el segundo caso sin signo.

Es posible generar el inicio de conversión por hardware; para esto se hace uso del parámetro *Hardware SOC* (*Start of Conversion*), habilitando esta función se hace visible una señal en el bloque ADC. Con esto la conversión se inicia con el flanco de subida de una señal externa conectada al pin; por defecto este parámetro esta deshabilitado.

Para eliminar el *offset* existente en la tierra de alimentación del ADC, se tiene el parámetro *Enable Vref/Vssa*, que habilita una señal para ser conectada físicamente a la tierra del PSoC; de esta manera se elimina el pequeño error existente, que puede ampliarse si se utiliza el amplificador de ganancia de la señal de entrada.

Finalmente el parámetro *Enable modulator* input permite controlar de manera dinámica la polaridad de la señal de entrada al modulador del ADC. La polaridad se invierte cuando la señal *mi* (*modulador input*) está en nivel lógico alto.

Rutinas utilizadas para el manejo del bloque ADC_Del_Sig

Las funciones API permiten realizar la configuración y el manejo del bloque ADC delta sigma. Para la sintaxis de estas funciones debe tenerse en cuenta que la primera parte del nombre de la función (ADC_) debe ser reemplazada por el nombre que el usuario le haya asignado al bloque.

Las principales funciones que pueden ser utilizadas para configurar y manejar el bloque son:

- ADC_Start():

  Inicializa variables e inicializa y habilita todos los parámetros de configuración seleccionados previamente para el ADC; utiliza para esto la información de la pestaña de configuración *Config1*. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- ADC_StartConvert():

  Hace que el ADC inicie una conversión. En modos *Multi Sample*, *Continuous* o *Multi Sample (Turbo)* iniciará una conversión y luego de finalizarla automáticamente iniciará una nueva. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- ADC_IsEndConversion():

  Retorna el estado de la conversión del ADC de acuerdo con el parámetro de entrada utilizado (ver tabla 21), que corresponde a un entero sin signo de 8 bits. Retorna un dato de 8 bits sin signo que representa el estado solicitado.

- ADC_GetResult8():

  Retorna el resultado de la conversión en formato de 8 bits. Si la resolución del ADC es mayor a 8 bits la funciona retorna el valor de la parte menos significativa de la conversión. No utiliza ningún parámetro de entrada. Retorna un valor de 8 bits con signo que representa el valor de la última conversión; por ejemplo, si se quiere leer (enviado el resultado a la variable *dato*) el resultado de un bloque *ADC_Del_Sig* denominado *ADC*, configurado con resolución de 8 bits, se utilizaría el siguiente código:

  ```
  dato=ADC_GetResult8();
  ```

- ADC_GetResult16():

  Retorna el resultado de la conversión en formato de 16 bits, para resoluciones de 8 a 16 del ADC. No utiliza ningún parámetro de entrada y retorna un valor de 16 bits con signo que representa el valor de la última conversión.

- ADC_GetResult32():

  Retorna el resultado de la conversión en formato de 32 bits, sin importar la resolución del ADC. No utiliza ningún parámetro de entrada y retorna un valor de 32 bits con signo que representa el valor de la última conversión.

- ADC_Read8():

  Realiza todo el proceso de captura de una conversión en formato de 8 bits, lo que genera el inicio de conversión, esperando hasta que se complete la conversión, deteniendo el ADC y leyendo el resultado de la conversión. No utiliza ningún parámetro de entrada y retorna un valor de 8 bits con signo que representa el valor de la conversión; por ejemplo, si se quiere leer (enviando el resultado a la variable dato) el resultado de un bloque *ADC_Del_Sig* denominado *ADC*, configurado con resolución de 8 bits, se utilizaría el siguiente código:

  ```
  dato=ADC_Read8();
  ```

- ADC_Read16():

  Realiza todo el proceso de captura de una conversión en formato de 16 bits, con lo cual se genera el inicio de conversión, esperando hasta que se complete la conversión, deteniendo el ADC y leyendo el resultado de la conversión. No utiliza ningún parámetro de entrada y retorna un valor de 16 bits con signo que representa el valor de la última conversión.

- ADC_Read32():

  Realiza todo el proceso de captura de una conversión en formato de 32 bits, y de esta manera se genera el inicio de conversión, esperando hasta que se complete la conversión, deteniendo el ADC y leyendo el resultado de la conversión. No utiliza ningún parámetro de entrada y retorna un valor de 32 bits con signo que representa el valor de la última conversión.

- ADC_SelectConfiguration():

  Selecciona una de las cuatro configuraciones posibles que puede manejar el ADC. Luego de seleccionar la configuración se sugiere para no tener problemas con el ADC activar la función de reinicio del ADC, con lo que se debe generar todo el proceso con la función ADC_Start(). Utiliza dos valores de entrada, cada uno como un valor de 8 bits sin signo; el primero que representa la configuración seleccionada debe ser de un valor entre 1 y 4, el segundo valor determina si se genera o no reinicio del ADC. Un valor 0 indica no reinicio, un valor 1 indica reinicio del ADC. La función no retorna ningún valor; por ejemplo, si se quiere hacer que un bloque *ADC_Del_Sig* denominado *ADC* cambie a la configuración número 3 y genere un reinicio del ADC, se tendría que escribir la siguiente línea de código:

  ```
  ADC_SelectConfiguration(3,1);
  ```

- ADC_CountsTo_Volts():

  Convierte el resultado generado por el ADC a un valor en voltios codificado en punto flotante. Utiliza como parámetro de entrada un entero de 32 bits con signo que almacena el resultado generado por el ADC. Retorna un valor en voltios en punto flotante equivalente al dato de entrada.

- ADC_CountsTo_mVolts():

  Convierte el resultado generado por el ADC a un valor en milivoltios. Utiliza como parámetro de entrada un entero de 32 bits con signo que almacena el resultado generado por el ADC. Retorna un valor en milivoltios como un entero de 16 bis con signo.

- ADC_CountsTo_uVolts():

  Convierte el resultado generado por el ADC a un valor en microvoltios. Utiliza como parámetro de entrada un entero de 32 con signo que almacena el resultado generado por el ADC. Retorna un valor en microvoltios como un entero de 32 bis con signo.

Ejemplos de uso del bloque ADC

Voltímetro digital con visualización en LCD

En el ejemplo que se desarrollará a continuación se implementa un voltímetro digital de cuatro dígitos BCD (un dígito entero y tres dígitos decimales) utilizando visualización de una LCD alfanumérica de 2x16; el voltaje por medir será una señal DC que podrá variar entre 0 y +5v (manejada por un potenciómetro conectado entre +5v y GND), los datos se actualizarán cada 500ms.

Solución

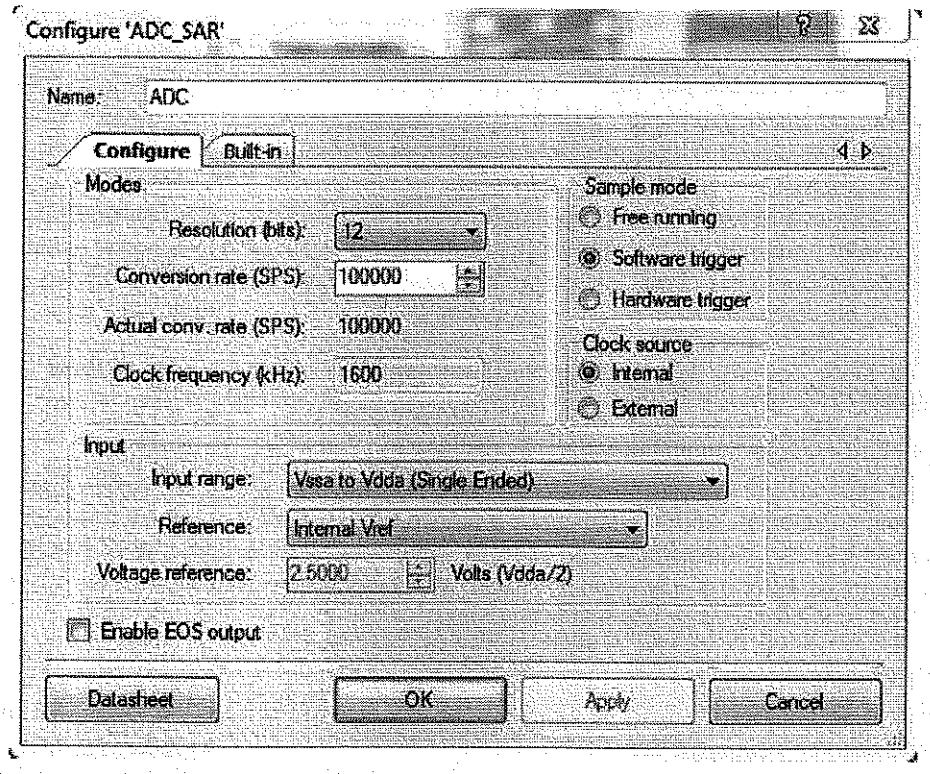
Para la solución se utilizará el bloque ADC por aproximaciones sucesivas (SAR) dado que no se requiere una alta velocidad de procesamiento. Para tener una visualización en milivoltios se utilizará la máxima resolución de este ADC que es de 12 bits; esto último no es un requerimiento del problema, así que cualquier resolución seleccionada funcionaría.

La solución en el PSoC Creator hace uso de un bloque ADC_SAR con el nombre ADC, el parámetro *Resolution (bits)* en 12, la opción *Sample mode* en *Software trigger*, que permite generar el inicio de conversión por software, el parámetro *Input range* con la opción *Vssa to Vdda (Single Ended)* seleccionada y finalmente el parámetro *Reference* con la opción *Internal Vref* seleccionada (ver figura 121).

Se utiliza un pin de E/S analógico que se conecta a la entrada del ADC y es la conexión del potenciómetro, con el nombre Potenciometro y la opción *External terminal* habilitada, en la cual se conectan un potenciómetro, la fuente de alimentación y la tierra del circuito tomadas del catálogo *off-chip* del PSoc Creator; los demás parámetros se dejan con el valor por defecto. Por último, se utiliza un bloque LCD para la visualización de los datos. En la figura 122 se presenta el diagrama circuital con la solución propuesta para el ejemplo.

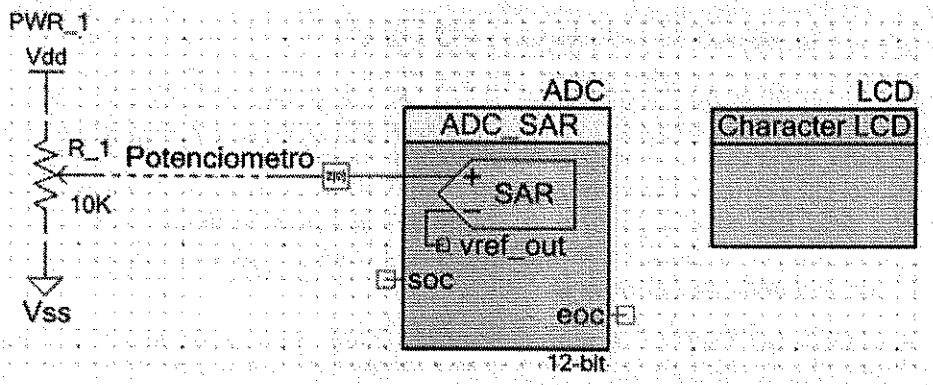
Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: los bits P12[6:0] para la conexión de la LCD, y el bit P2[0] para la conexión de la entrada analógica.

Figura 121. Ventana de configuración del ADC_SAR para el ejemplo 8-1



Fuente: elaboración propia.

Figura 122. Diagrama circuital del ejemplo 8-1



Fuente: elaboración propia.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 8-1:

```
/* ==========================================================
 *
 * Programa que visualiza en una LCD alfanumérica de 2x16 el valor
 * de voltaje medido de un potenciómetro conectado a una de las
 * entradas del PSoC, el valor mostrado tendrá un digito entero y
 * tres dígitos decimales en el rango de 0.000V a 5.000V, el dato
 * se actualiza cada 500ms.
 *
 * ==========================================================
*/
#include <project.h>
/* Definición de variables */
uint16 volt;
uint8 dato;
int main()
{
  LCD_Start();
  ADC_Start(); //Inicializa ADC SAR
  LCD_Position(0,0); //Ubica cursor al inicio de la LCD
  LCD_PrintString("Voltaje:");
  for(;;)
  {
    LCD_Position(1,8); //Ubica cursor donde se mostrar el voltaje
    ADC_StartConvert(); //Inicia conversión
    ADC_IsEndConversion(ADC_WAIT_FOR_RESULT); //Espera final de
                                                //conversión
    volt=ADC_GetResult16(); //Lee resultado de conversión
    volt=ADC_CountsTo_mVolts(volt); //Convierte dato leído a mV en Hx
    dato=volt/1000; //Convierte a decimal dato de voltaje realizando
```

```
        volt=volt%1000; //divisiones sucesivas por múltiplos de 10
        LCD_PutChar(dato+0x30); //luego envía cada digito a la LCD
        LCD_PutChar('.');
        dato=volt/100;
        volt=volt%100;
        LCD_PutChar(dato+0x30);
        dato=volt/10;
        volt=volt%10;
        LCD_PutChar(dato+0x30);
        LCD_PutChar((uint8)volt+0x30);
        LCD_PutChar('v');
        CyDelay(500); //Espera 500 ms para una nueva conversión
    }
}
```

Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC®, y luego cablear cada uno de los componentes a la tarjeta de desarrollo.

Termómetro digital

En el siguiente ejemplo se muestra el diseño e implementación de un termómetro digital haciendo uso de un sensor de temperatura de referencia LM35, que tiene como característica principal su linealidad, lo que genera una señal analógica de 10mV/°C, operando en el rango de 0°C a 100°C. El resultado de la temperatura se mostrará en formato de cuatro dígitos BCD (dos dígitos enteros y dos dígitos decimales) utilizando como visualización una LCD alfanumérica de 2x16 y actualizando el dato una vez por segundo.

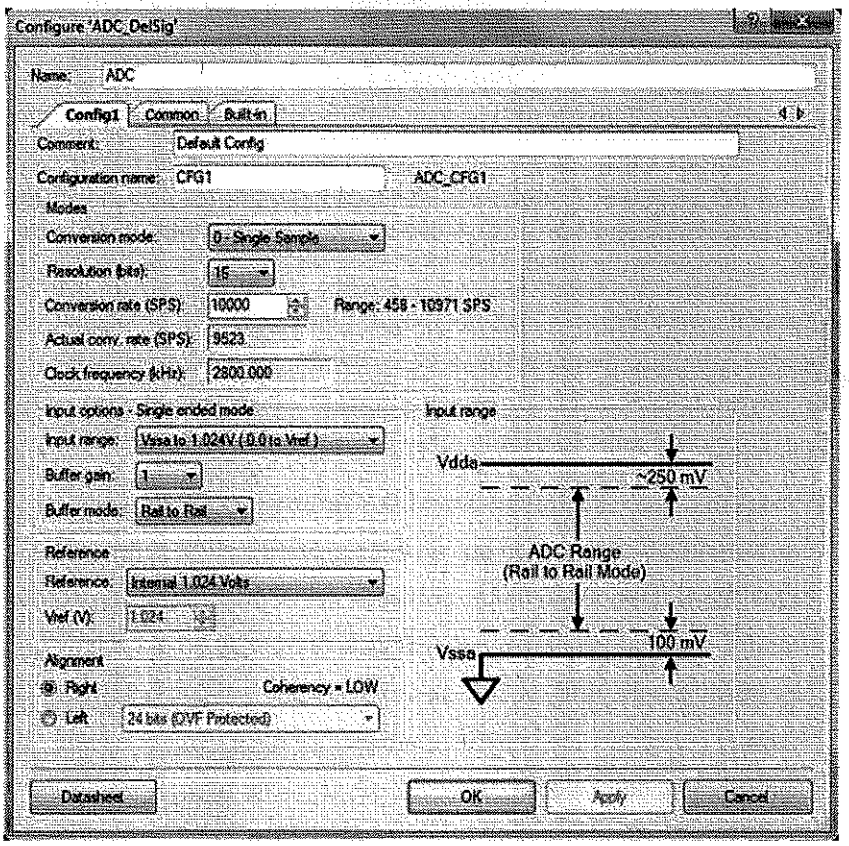
Solución

Para la solución se utilizará el bloque ADC delta sigma únicamente con fines demostrativos, dado que el ADC por aproximaciones sucesivas (SAR) también podría ser utilizado en la implementación de la solución. Para tener una visualización en décimas de grado se utilizará una resolución de 16 bits. Esto último no es un requerimiento del problema, así que cualquier resolución mayor a 10 bits funcionaría ya que daría la resolución en decimas de grado requerida.

Inicialmente se hace uso de un bloque ADC_Del_Sig con el nombre ADC. En la pestaña Common se selecciona la siguiente configuración: el parámetro *Input mode* con la opción *Single ended* seleccionada y la opción *Number of configurations* con el valor en 1; los demás parámetros con sus valores por defecto. En la pestaña *Config1* se selecciona la siguiente configuración: la opción *Conversion Mode* en *Single Sample* para

realizar conversiones únicas y tener control sobre el momento cuando se realiza el muestreo de la señal, el parámetro *Resolution (bits)* en 16, el parámetro *Input range* con la opción *Vssa to 1.024V (0.0 to Vref)* seleccionada ya que el valor máximo de voltaje generado por el sensor será de 1V por el rango de temperatura que maneja, la opción *Buffer gain* en 1, *Buffer mode* en *Rail to Rail* y finalmente el parámetro *Reference* con la opción *Internal 1.024 Volts* seleccionada. En la figura 123 puede verse la configuración realizada para la pestaña *Config1*.

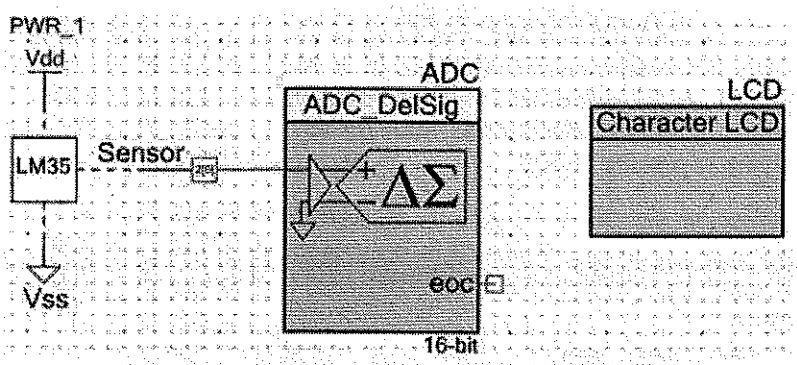
Figura 123. Ventana de configuración del ADC_Del_Sig para el ejemplo 8-2



Fuente: elaboración propia.

Se utiliza un pin de E/S analógico que se conecta a la entrada del ADC y es la conexión del sensor de temperatura, con el nombre Sensor y la opción *External terminal* habilitada, donde se conectan el sensor de temperatura, la fuente de alimentación y la tierra del circuito tomadas del catálogo *off-chip* del PSoc Creator (el sensor fue dibujado con las herramientas que posee el software dado que no está disponible en el catálogo de componentes), los demás parámetros se dejan con el valor por defecto. Por último, se utiliza un bloque LCD para la visualización de los datos. En la figura 124 se presenta el diagrama circuital con la solución propuesta para el ejemplo.

Figura 124. Diagrama circuital para la solución propuesta del ejemplo 8-2



Fuente: elaboración propia.

Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: los bits P12[6:0] para la conexión de la LCD, y el bit P2[0] para la conexión de la entrada analógica.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 8-2:

```
/* ==========================================================
 *
 * Programa que visualiza en una LCD alfanumérica de 2x16 el valor
 * de temperatura generado por un sensor analógico de referencia
 * LM35 conectado a una de las entradas del PSoC, el valor mostrado
 * tendrá dos dígitos enteros y dos dígitos decimales en el rango
 * de 00.00°C a 99.99°C, el dato se actualiza una vez por segundo.
 * Para la conversión de la señal analógica se hace uso del bloque
 * interno ADC delta sigma.
 *
 * ==========================================================
*/
#include <project.h>
/* Definición de variables */
uint32 temp;
uint8 dato;
int main()
```

```
{
  LCD_Start();
  ADC_Start(); //Inicializa ADC SAR
  LCD_Position(0,0); //Ubica cursor al inicio de la LCD
  LCD_PrintString("Temperatura:");
  for(;;)
  {
    LCD_Position(1,8); //Ubica cursor donde se mostrar el voltaje
    temp=ADC_Read16();//Inicia conversión, espera final de conversión
                      //y lee resultado de conversión
    temp=(uint16)temp*1526; //Escaliza el valor leído del ADC al rango
    temp=temp/1000; //de temperatura del sensor 0 a 100°C
                    //Ya que la resolución es de 16 bits se
                    //obtiene un valor de 100°C/65535=0.001526mV
                    //por bit esto se traduce en multiplicar por
                    //1526 y dividir luego por 10000 para evitar
                    //utilizar punto flotante.
    dato=temp/10000; //Convierte a decimal dato de voltaje realizando
    temp=temp%10000; //divisiones sucesivas por múltiplos de 10
    LCD_PutChar(dato+0x30); //luego envía cada digito a la LCD
    dato=temp/1000;
    temp=temp%1000;
    LCD_PutChar(dato+0x30);
    LCD_PutChar('.');
    dato=temp/100;
    temp=temp%100;
    LCD_PutChar(dato+0x30);
    dato=temp/10;
    LCD_PutChar(dato+0x30);
    LCD_PutChar(0xDF); //Carácter ASCII del símbolo °
    LCD_PutChar('C');
    CyDelay(1000); //Espera 1 segundo para una nueva conversión
  }
}
```

Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC®, y luego cablear cada uno de los componentes a la tarjeta de desarrollo.

Temporizador/contador del PSoC5®LP

La tarea de temporización es un proceso utilizado en la mayoría de aplicaciones que implican el uso de sistemas embebidos. Como se mostró en ejemplos de capítulos anteriores, esta temporización puede realizarse por software haciendo uso de ciclos repetitivos, utilizando para esto instrucciones del microcontrolador. Este proceso funciona y es ampliamente utilizado, pero tiene el inconveniente de que ocupa la CPU en la ejecución de dichas instrucciones, con lo que se complica la posibilidad de realizar otras tareas mientras se está generando el retardo de tiempo.

Para solucionar esta deficiencia o inconveniente, gran parte de los microcontroladores existentes en el mercando posee un módulo de hardware denominado temporizador (*timer*) que en el caso específico de la familia PSoC5®LP el bloque equivalente de temporización tiene la misma denominación: *Timer*.

Descripción general

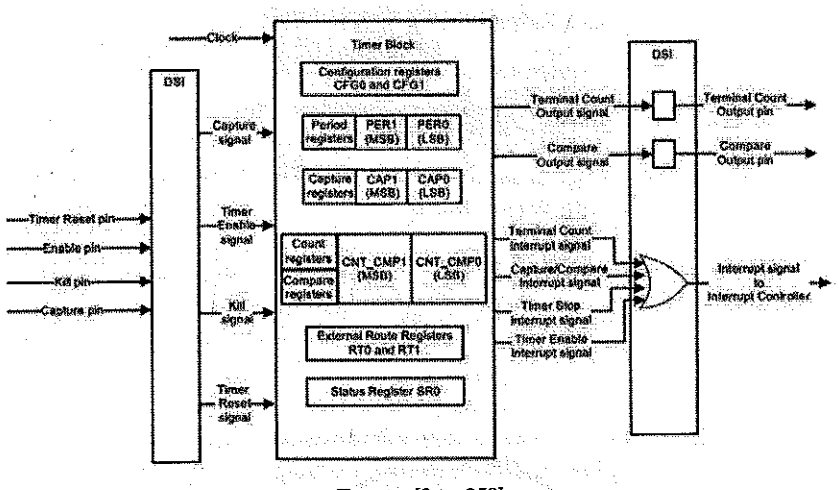
Las características detalladas para el bloque Timer pueden encontrarse en el *datasheet* con nombre *Timer_v2_70.pdf*, las siguientes son las más importantes:

- Implementación del temporizador por medio de bloques UDB.
- Bloque temporizador implementado físicamente (FF).
- Resoluciones de 8 y 16 bits para implementación FF.
- Resolución de 8, 16, 24 y 32 bits para implementaciones UDB.
- Modo de conteo del temporizador descendente.
- Fuente de reloj seleccionable por el usuario.
- Inicio de temporización por software o por combinación software/hardware.
- Captura de entrada opcional.
- Habilitación, disparo y reinicio de entradas del temporizador por sincronización con otros componentes.
- Modo de operación *one shot* (una sola vez) o continuo.
- Soporte de interrupciones por final de temporización (*Terminal count*), comparación (*Compare true*) y captura de entrada (*Capture*).

En la figura 125 puede verse el diagrama de bloques del temporizador del PSoC®, de allí pueden diferenciarse tres grandes bloques: el primero permite seleccionar desde diversas fuentes la señal de la entrada al temporizador, realizando decrementos de contador cada vez que llega un flanco de subida de la señal seleccionada. El contador al llegar a cero puede o no reiniciarse dependiendo de la configuración de este (modo continuo o *one shot*); así mismo, genera una señal que va a la salida *terminal count* (tc) del bloque temporizador.

El segundo es el encargado de almacenar toda la configuración del Timer y todos los estados generados durante su operación. El tercer bloque genera las señales que van

Figura 125. Diagrama de bloques del módulo Timer



Fuente: [2, p. 250].

al exterior del temporizador para que puedan ser utilizadas en eventos de interrupción o similares.

Además de lo anterior, el bloque temporizador puede ser utilizado para capturar el tiempo de un evento externo, medir el periodo o el ancho de pulso de una señal, encontrar el tiempo de ocurrencia de una interrupción y como un generador PWM.

Uso del bloque Timer

Para seleccionar en el PSoC Creator el bloque Timer denominado Timer [v2.70] (figura 126), el usuario debe buscar la carpeta Digital del catálogo de componentes, subcarpeta Functions y seleccionarlo entre las opciones allí disponibles.

Señales de entrada/salida del Timer

La imagen de la figura 127 muestra el bloque Timer con todas las posibles señales de entrada/salida soportadas por este; algunas de ellas pueden ser eliminadas en la configuración si no son necesarias para la aplicación.

- clock (input):

  Señal de reloj de entrada que define la frecuencia de operación del Timer y por consiguiente el tiempo generado por este. Esta entrada no puede ser eliminada.

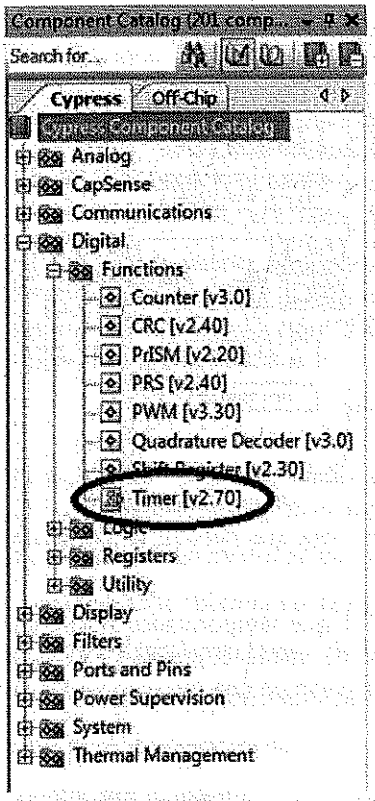
- reset (input):

  Entrada de reset sincrónico, se encarga de reiniciar los contadores internos del Timer. Esta entrada no puede ser eliminada.

- enable (input):

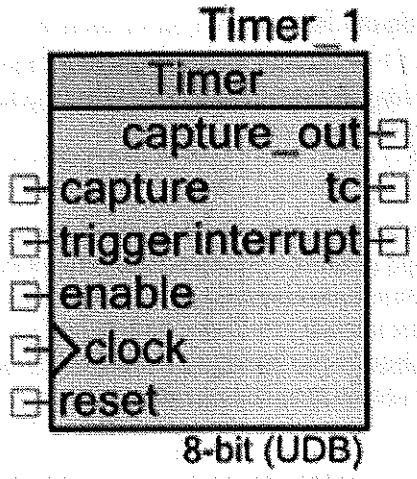
  Entrada de habilitación por hardware del Timer, activa en alto; es decir, cuando

Figura 126. Selección del bloque Timer



Fuente: elaboración propia.

Figura 127. Señales de entrada/salida del Timer



Fuente: elaboración propia.

la entrada tiene un nivel bajo el Timer está deshabilitado para temporizar. Esta entrada puede ser eliminada.

- capture (input):

  Permite capturar el valor actual de los contadores cuando se genera en la entrada alguno de los diversos tipos de excitación (flanco de subida, bajada o ambos). Esta entrada puede ser eliminada.

- trigger (input):

  Detecta una señal de entrada que es utilizada para generar retardos en la operación del Timer, señal disponible únicamente para la implementación del Timer UDB. Esta entrada puede ser eliminada.

- tc (output):

  Salida sincrónica denominada *Terminal counter* que indica la finalización del periodo de tiempo configurado. Esta salida no puede ser eliminada.

- interrupt (output):

  Genera una señal de interrupción para ser utilizada como fuente de interrupción por hardware. Esta salida no puede ser eliminada.

- capture_out (output):

  Es una señal de salida que indica cuando la captura por hardware ha sido disparada, señal disponible únicamente para la implementación del Timer UDB. Esta salida puede ser eliminada.

Configuración del bloque Timer

Dependiendo del tipo de Timer por utilizar FF (*Fixed-function*) o UDB (*Universal Digital Block*), la ventana de configuración incluye diferentes parámetros. Para la ventana de configuración del bloque Timer tipo FF se pueden modificar o configurar básicamente los siguientes parámetros: nombre (*Name*), la resolución del Timer (*Resolution*), el periodo generado a partir de la frecuencia de la señal de reloj de entrada (*Period*), modo de captura de entrada (*Capture Mode*), tipo de habilitación del Timer (*Enable Mode*), modo de operación (*Run Mode*) y el tipo de interrupción o interrupciones manejadas (*Interrupts*).

En la figura 128 puede verse la ventana de configuración del bloque Timer con las funcionalidades antes nombradas.

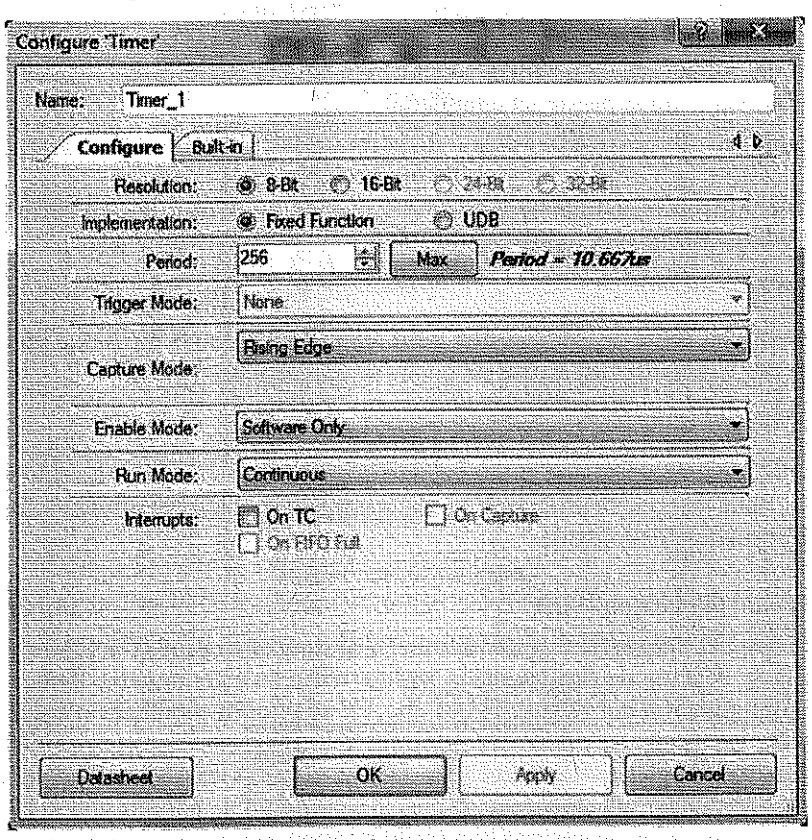
- *Resolution*: permite seleccionar entre dos valores posibles la resolución del Timer: 8 (valor por defecto) o 16 bits; este parámetro determina el valor de conteo máximo de los contadores internos del Timer.

- *Period*: determina la cantidad de conteos que realiza el contador interno del Timer; con este valor y con la frecuencia de la señal de reloj de entrada se determina el tiempo generado por el módulo.

El parámetro *Capture Mode* determina el tipo de excitación utilizada para activar la captura del Timer entre dos opciones posibles:

- *None*: deshabilita la opción de captura eliminado el pin de entrada correspondiente al bloque.
- *Rising Edge*: captura el valor del contador interno del Timer cuando aparece un flanco de subida en la entrada *capture* del bloque Timer (opción por defecto).

Figura 128. Ventana de configuración del bloque Timer FF



Fuente: elaboración propia.

El parámetro *Enable Mode* determina el tipo de habilitación del temporizador entre dos posibles:

- *Software only*: el temporizador se habilita únicamente por software, modificando el bit de habilitación en registro de control del Timer, por medio de una función API (opción por defecto).
- *Software and Hardware:* el temporizador se habilita por medio de una función API por software o por medio de una señal que ingresa a la entrada *enable* del bloque temporizador.

Con el parámetro *Run Mode* se puede seleccionar el tipo de operación del Timer entre tres valores posibles:

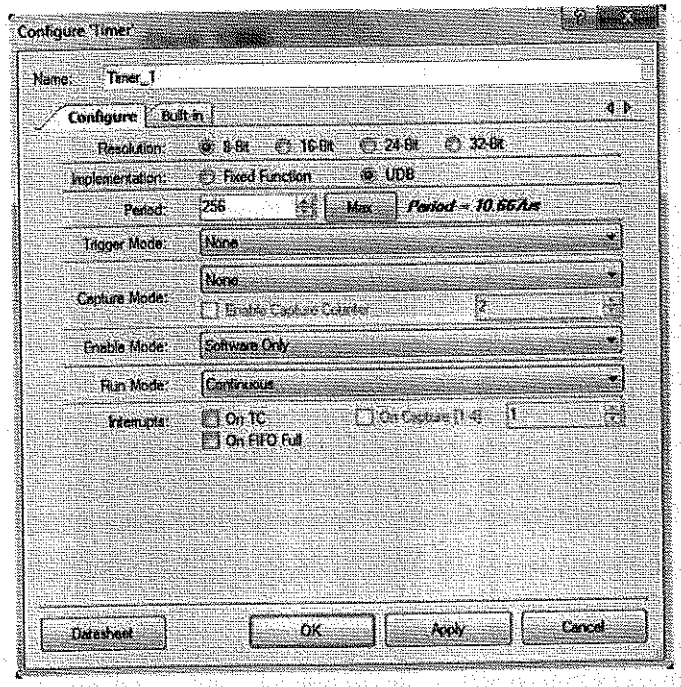
- *Continuos*: el temporizador realiza su tarea de forma continua mientras esté habilitado (opción por defecto).
- *Shot*: el temporizador realiza su tarea una vez y luego se detiene al finalizar.
- *One Shot (Halt on Interrupt)*: el temporizador realiza su tarea una vez y luego se detiene al finalizar o cuando se genera una interrupción.

El parámetro *Interrupts* para la configuración del bloque *Timer FF* solamente tiene habilitada la opción *On TC* para ser modificada, este parámetro habilita o deshabilita la interrupción generada por el Timer, por defecto está deshabilitada.

Para la ventana de configuración del bloque Timer tipo UDB se pueden modificar o configurar básicamente los siguientes parámetros: nombre (*Name*), la resolución del Timer (*Resolution*), el periodo generado a partir de la frecuencia de la señal de reloj de entrada (*Period*), modo de disparo (*Trigger Mode*), modo de captura de entrada (*Capture Mode*), tipo de habilitación del Timer (*Enable Mode*), modo de operación (*Run Mode*) y el tipo de interrupción o interrupciones manejadas (*Interrupts*).

En la figura 129 puede verse la ventana de configuración del bloque *Timer* tipo UDB con las funcionalidades antes nombradas.

Figura 129. Ventana de configuración del bloque Timer UDB



Fuente: elaboración propia.

- *Resolution*: permite seleccionar entre cuatro valores posibles la resolución del Timer: 8 (valor por defecto), 16, 24 o 32 bits; este parámetro determina el valor de conteo máximo de los contadores internos del Timer.
- *Period*: determina la cantidad de conteos que realiza el contador interno del Timer. Con este valor y con la frecuencia de la señal de reloj de entrada se determina el tiempo generado por el módulo; el valor máximo depende de la resolución seleccionada en el parámetro anterior.

Para poder capturar (contar eventos externos) una señal de entrada se puede seleccionar algún tipo de característica de la señal para esta tarea, con el parámetro *Trigger Mode*, entre cinco valores posibles:

- *None*: no se realiza ningún proceso de captura de señales (valor por defecto).
- *Rising Edge*: el conteo del Timer se realiza cuando parece un flanco de subida en la señal de disparo.
- *Falling Edge*: el conteo del Timer se realiza cuando parece un flanco de bajada en la señal de disparo.
- *Either Edge*: el conteo delTtimer se realiza cuando parece cualquier flanco en la señal de disparo.
- *Software Controlled*: el conteo del Timer se realiza por software haciendo uso de funciones API del bloque timer

El parámetro *Capture Mode* determina el tipo de excitación utilizada para activar la captura del Timer entre cinco opciones posibles:

- *None*: deshabilita la opción de captura eliminado el pin de entrada correspondiente al bloque (opción por defecto).
- *Edge*: captura el valor del contador interno del Timer cuando aparece un flanco de subida en la entrada capture del bloque Timer.
- *Falling Edge*: captura el valor del contador interno del Timer cuando aparece un flanco de bajada en la entrada capture del bloque Timer.
- *Either Edge*: captura el valor del contador interno del Timer cuando aparece cualquier flanco en la entrada *capture* del bloque Timer.
- *Software Controlled*: la captura del valor del contador interno del Timer se realiza por software haciendo uso de funciones API del bloque Timer. Esta opción habilita la posibilidad de modificar otro parámetro adicional denominado *Enable Capture Counter*, que permite determinar la cantidad de capturas por realizar.

Haciendo uso del parámetro *Enable Mode* se asigna el tipo de habilitación del temporizador entre tres valores posibles:

- *Software only*: el temporizador se habilita únicamente por software, modificando el bit de habilitación en registro de control del Timer, por medio de una función API (opción por defecto).

- *Hardware only*: el temporizador se habilita únicamente por hardware, por medio de una señal que ingresa a la entrada enable del bloque temporizador.
- *Software and Hardware*: el temporizador se habilita por medio de una función API por software o por medio de una señal que ingresa a la entrada *enable* del bloque temporizador.

Con el parámetro *Run Mode* se puede seleccionar el tipo de operación del Timer entre tres valores posibles:

- *Continuos*: el temporizador realiza su tarea de forma continua mientras esté habilitado (opción por defecto).
- *One Shot*: el temporizador realiza su tarea una vez y luego se detiene al finalizar.
- *One Shot (Halt on Interrupt)*: el temporizador realiza su tarea una vez y luego se detiene al finalizar o cuando se genere una interrupción.

El parámetro *Interrupts* para el bloque *Timer UDB* permite configurar las fuentes de interrupción soportadas por el bloque entre tres posibles:

- *On TC*: habilita o deshabilita la interrupción generada por el Timer; por defecto está deshabilitada.
- *On Capture (1-4)*: habilita o deshabilita la interrupción generada por la captura del Timer, puede seleccionarse el número de capturas que genera esta interrupción en un rango de 1 a 4.
- *On FIFO Full*: habilita o deshabilita la interrupción generada cuando se llena la FIFO de captura del Timer, cuyo tamaño depende del valor seleccionado cuando se habilita la opción Software Controlled.

Rutinas utilizadas para el manejo del bloque Timer

En general las funciones API para el bloque Timer permiten configurar, controlar y poner a andar sus diversos modos de operación, sin importar el tipo de implementación seleccionado. Para la sintaxis de estas funciones debe tenerse en cuenta que la primera parte del nombre de la función (Timer_) debe ser reemplazado por el nombre que el usuario le haya asignado al bloque.

Las funciones que pueden ser utilizadas para configurar y manejar el bloque son:

- Timer_Start():

  Inicializa variables asociadas al Timer e inicializa y habilita todos los parámetros de configuración seleccionados previamente para el Timer. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- Timer_Stop():

  Para implementación FF (*Fixed-funtion*) deshabilita el Timer y baja el consumo de potencia del bloque. Para implementación UDB deshabilita el Timer única-

mente si la opción de operación por software está habilitada. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- Timer_ReadStatusRegister():

  Lee el valor actual del registro de estado del Timer, debe tenerse en cuenta que dependiendo del tipo de implementación utilizada para el Timer (FF o UDB) la información leída es diferente por las funcionalidad existentes en cada caso. No utiliza ningún parámetro de entrada y retorna un valor de 8 bits sin signo con el valor leído del registro. Si por ejemplo se desea esperar hasta que el *Timer* con implementación *FF* (denominado Timer) se desborde (finalice el tiempo configurado), se necesita conocer el estado de la bandera TC (*Terminal Counter*) que ocupa la posición siete en el registro de estado del Timer. Para esto se podría utilizar la siguiente línea de código:

```
while ((Timer_ReadStatusRegister()&0x80)==0);
```

- Timer_WriteCounter():

  Escribe un nuevo valor directamente en el registro de conteo del Timer; esta función es válida solamente para implementación UDB. Para que el funcionamiento del Timer no genere errores antes de ejecutar esta función debe deshabilitarse el Timer. Como parámetro de entrada utiliza un entero sin signo de 8, 16 o 32 bits, que depende de la resolución seleccionada para el Timer, el cual contiene el nuevo valor asignado al Timer. La función no retorna ningún valor; si por ejemplo se requiere asignar al contador de un *Timer UDB* (denominado Timer) con resolución de 24 bits el valor 0x0FFF000 se utilizaría la siguiente línea de código:

```
Timer_WriteCounter(0x0FFF000);
```

- Time_ReadCounter():

  Obliga al Timer a realizar una captura y retornar el valor capturado solamente para implementaciones UDB. No utiliza ningún parámetro de entrada, y retorna un entero sin signo de 8, 16 o 32 bits, que depende de la resolución seleccionada para el Timer.

- Timer_ReadCapture():

  Lee el valor del contenido de captura del Timer o el valor almacenado en la FIFO en la implementación UDB. No utiliza ningún parámetro de entrada, y retorna un entero sin signo de 8, 16 o 32 bits, que depende de la resolución seleccionada para el Timer.

- Timer_SoftwareCapture():

  Obliga al Timer a generar una captura por software, el valor es almacenado en la FIFO de captura, únicamente aplica para implementaciones UDB. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- Timer_ClearFIFO():

  Borra toda la información del FIFO de captura del Timer; esta función está disponible solo para implementaciones UDB. No utiliza ningún parámetro de entrada ni retorna ningún valor.

Ejemplos de uso del bloque Timer

Operación básica del bloque Timer

El ejemplo que se presentará a continuación pretende mostrar la operación básica del bloque temporizador del PSoC5®LP. Para esto se oscilarán dos pines del PSoC® haciendo uso para cada caso de un Timer, uno con implementación FF y el otro con implementación UDB; además, se utilizará en un caso la señal TC por software y el otro caso por hardware.

Solución

Como se indicó en el planteamiento del ejemplo para la solución se utilizarán dos bloques Timer en implementaciones FF y UDB, cada uno con características de frecuencia y de configuración diferentes, los valores de tiempo de cada uno se seleccionaron libremente dado que el problema no tiene ninguna restricción al respecto.

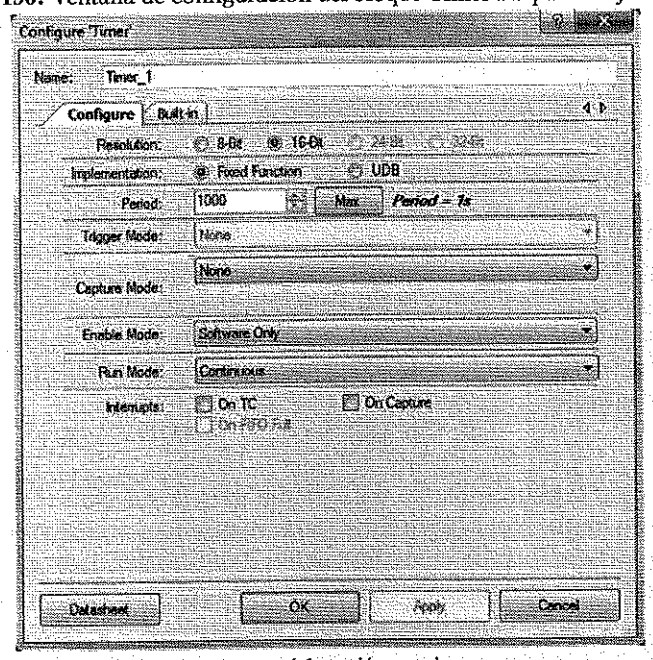
Inicialmente se hace uso de un primer bloque Timer con implementación FF (*Fixed Function*) con el nombre Timer_1, en la pestaña *Configure* se selecciona la siguiente configuración: el parámetro *Resolution* en 16-Bit, el valor *Period* en 1000 con lo que obtiene un tiempo de un segundo; para esto se debe configurar la frecuencia del bloque *timer_clock* en 1kHz. El parámetro *Capture Mode* con la opción *None* seleccionada, *Enable Mode* con la opción *Software Only* seleccionada, *Run Mode* en modo *Continuos*, y las interrupciones deshabilitadas, en la figura 130 puede verse la configuración realizada para la pestaña *Configure*.

El segundo bloque Timer utilizado trabaja con implementación UDB con el nombre Timer_2; en la pestaña *Configure* se configuran los siguientes parámetros: el parámetro *Resolution* en 16-Bit, el valor *Period* en 5000 con lo que obtiene un tiempo de 500 ms. Para esto se debe configurar la frecuencia del bloque *timer_clock1* en 10kHz. El parámetro *Trigger Mode* con la opción *None* seleccionada, *Capture Mode* con la opción *None* seleccionada, *Enable Mode* con la opción *Software Only* seleccionada, *Run Mode* en modo *Continuos*, y el parámetro *Interrupts* con todas las opciones de interrupción deshabilitadas. En la figura 131 puede verse la configuración realizada para la pestaña *Configure*.

Un tercer bloque utilizado en el diseño corresponde a uno de E/S de un bit configurado como salida, con nombre LED1, la opción *HW connection* deshabilitada, la opción *External terminal* habilitada, se conecta a este bloque una resistencia, un LED y la tierra del circuito.

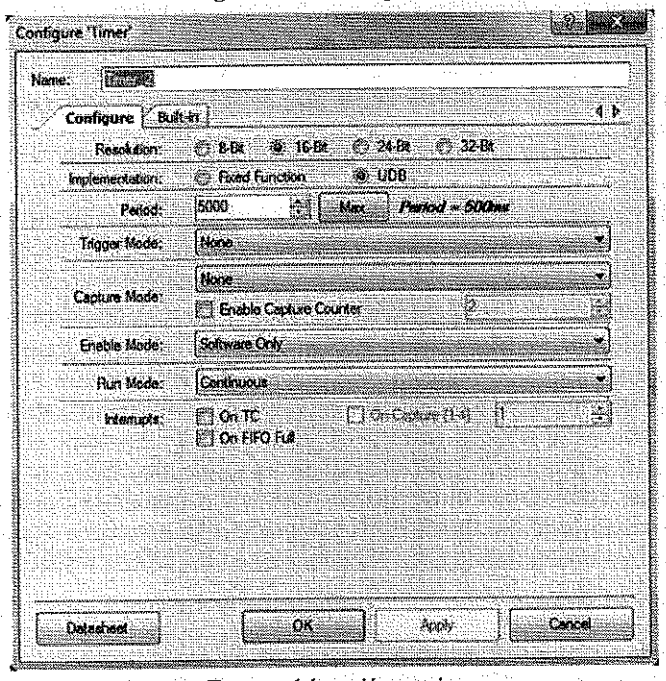
El cuarto bloque corresponde a otro de E/S de un bit configurado como salida, con nombre LED2, la opción *HW connection* habilitada, la opción *External terminal* habi-

Figura 130. Ventana de configuración del bloque Timer FF para el ejemplo 9-1



Fuente: elaboración propia.

Figura 131. Ventana de configuración del bloque Timer UDB para el ejemplo 9-1

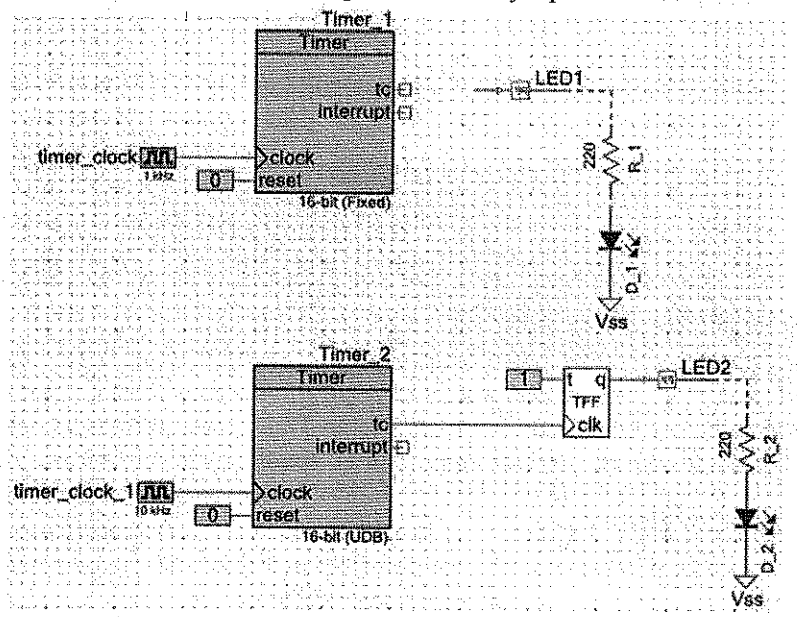


Fuente: elaboración propia.

litada, en la cual se conecta una resistencia, un LED y la tierra del circuito. Además, este bloque es conectado internamente a un FF tipo T que se encarga de recibir la señal TC (*Terminal Counter*), generada por el Timer_2 cada vez que se completa el tiempo configurado para este (500 ms), para así conmutar la salida que va hacia el LED.

En la figura 132 se presenta el diagrama circuital con la solución propuesta para el ejemplo. Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: el bit P1[6] para la conexión de la salida LED1 y el bit P1[7] para la conexión de la salida LED2.

Figura 132. Diagrama circuital ejemplo 9-1



Fuente: elaboración propia.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 9-1:

```
/* ==========================================================
 *
 * Programa que hace conmutar el estado de dos pines de salida del
 * PSoC, se hace uso de un Timer con implementación FF y otro con
 * implementación UDB, haciendo uso de la señal TC por software
 * para el primer Timer y la señal TC por hardware para el segundo.
 *
 * ==========================================================
```

```
*/
#include <project.h>
int main()
{
  Timer_1_Start(); //Inicializa y pone a correr el Timer_1
  Timer_2_Start(); //Inicializa y pone a correr el Timer_2

  for(;;)
  {
    while ((Timer_1_ReadStatusRegister()&0x80)==0); //Espera hasta que
                                                    //la bandera TC se active
    LED1_Write(!LED1_Read()); //Cambia el estado de la salida LED1
  }
}
```

Reloj digital con visualización en LCD

En el ejemplo que se desarrollará a continuación se implementa un reloj digital que visualiza en una LCD alfanumérica de 2x16 la hora en formato hh:mm:ss am/pm; el control de tiempo será realizado por medio de un Timer configurado a un segundo, haciendo además uso de interrupciones para la actualización del tiempo. La hora inicial al encender el reloj será las 12:00 am. Por medio de pulsadores se podrá ajustar la hora a la que el usuario desee.

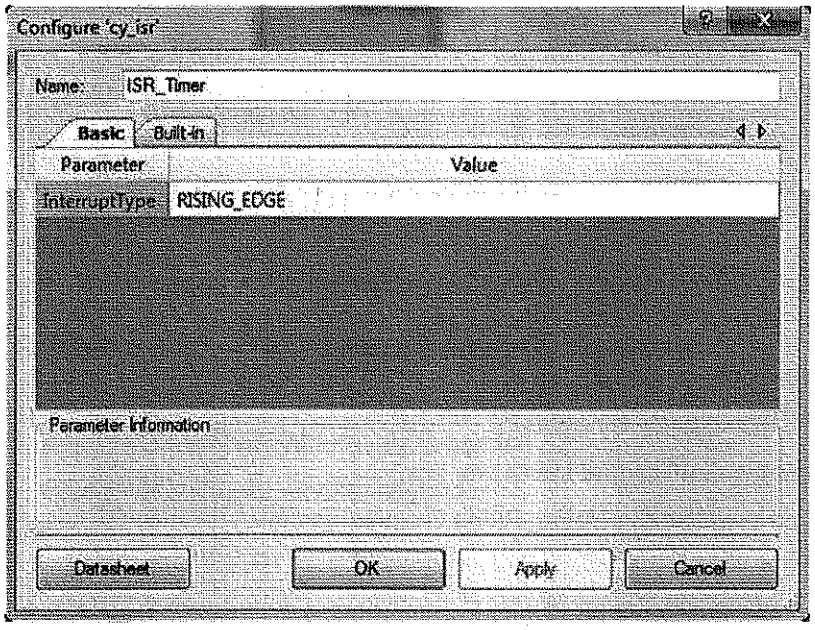
Solución

La solución planteada hace uso de un Timer con implementación FF con un periodo de 1s; la señal TC generada por el Timer se utiliza para solicitar una interrupción a la CPU del PSoC®, en la rutina de interrupción se realiza la actualización de los contadores utilizados para almacenar la hora del reloj. La información de la hora se actualiza cada segundo en una LCD, por lo que se requiere el uso de este bloque en la solución. Finalmente, para permitirle al usuario ajustar la hora del reloj se conectarán a las entradas del PSoC® dos pulsadores N.A. que operando también por interrupciones permitirán esta tarea.

Inicialmente se hace uso de un bloque Timer con implementación FF (*Fixed Function*) con el nombre Timer, y en la pestaña *Configure* se selecciona la siguiente configuración: el parámetro *Resolution* en *8-Bit*, el valor *Period* en 100 con lo que obtiene un tiempo de un segundo. Para esto se debe configurar la frecuencia del bloque *timer_clock* en 100Hz con una tolerancia del 1 %; este último parámetro determina la precisión del reloj. El parámetro *Capture Mode* con la opción *None* seleccionada, *Enable Mode* con la opción *Software Only* seleccionada, *Run Mode* en modo *Continuos*, y las interrupciones por hardware deshabilitadas.

Como es un requerimiento del diseño el empleo de las interrupciones, se hace uso de un bloque cy_isr_v1_70 con nombre ISR_Timer; a este bloque se le modifica el tipo de disparo que genera la interrupción cuyo parámetro se denomina Interrupt Type, el valor asignado al bloque es RISING_EDGE, que permite utilizar la señal TC generada por el bloque Timer por flanco de subida, que es como se genera este pulso momentáneo. En la figura 133 puede verse la configuración asignada al bloque de interrupciones asociado al Timer.

Figura 133. Ventana de configuración de la interrupción asociada al Timer



Fuente: elaboración propia.

El bloque LCD está configurado con las opciones por defecto, es decir, la opción LCD Custom Character Set en None, y habilitada la opción Include ASCII to Number Conversion Routines.

El siguiente bloque corresponde a uno de E/S de dos bits configurado como entrada, con nombre SW, la opción HW connection deshabilitada, la opción External terminal habilitada, en la cual se conectan dos pulsadores N.A. y la tierra del circuito y en la opción Drive mode se selecciona Resistive pull up; además, en la pestaña Input de este bloque se modifica el parámetro Interrupt seleccionando la opción Falling edge, para habilitar la interrupción del bloque.

Por último se incluye otro bloque de interrupción con nombre ISR_SW, para controlar la interrupción del bloque SW, al que únicamente se le modifica el nombre en la configuración. En la figura 134 puede verse el diagrama circuital de la solución propuesta para el ejemplo.

Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: los bits P12[6:0] para la conexión de la LCD y los bits P2[1:0] para la conexión de los pulsadores N.A. que permitirán al usuario el ajuste de la hora en el reloj.

A continuación puede verse el código escrito para la solución del problema:

Eejemplo 9-2:

```
/* ============================================================
 *
 * Programa que implementa un reloj digital con visualización en
 * una LCD alfanumérica de 2x16 el valor de la hora con formato
 * am/pm. Se tiene además la opción de ajustar la hora con un par
 * de pulsadores N.A. conectados a entradas del PSoC.
 *
 * ============================================================
*/
#include <project.h>
/* Definición de variables */
volatile uint8 seg,min,hora,am_pm;
void visual(void) //Define rutina de visualización
{
  LCD_Position(0,0); //Ubica cursor al inicio
  if (hora<=9) //Si variable hora menor o igual a 9 visualiza un cero
  {
    LCD_PutChar('0');
  }
  LCD_PrintDecUint16(hora); //Visualiza variable hora
  LCD_PutChar(':');
  if (min<=9) //Si variable minutos menor o igual a 9 visualiza un cero
  {
    LCD_PutChar('0');
  }
  LCD_PrintDecUint16(min); //Visualiza variable minutos
```

```
  LCD_PutChar(':');
  if (seg<=9) //Si variable segundos menor o igual a 9 visualiza un cero
  {
    LCD_PutChar('0');
  }
  LCD_PrintDecUint16(seg); //Visualiza variable segundos
  if (am_pm==0) //Dependiendo del estado de la bandera muestra am o pm
  {
    LCD_PrintString(" a.m.");
  }else{
    LCD_PrintString(" p.m.");
  }
}
CY_ISR(Int_Timer) //Define rutina de interrupción del Timer
{
  seg++; //Incrementa variable de segundos
  if (seg==60) //Ajusta variable segundos
  {
    seg=0;
    min++;
    if (min==60) //Ajusta variable minutos
    {
      min=0;
      hora++;
      if (hora==12) am_pm=~am_pm;
      if (hora==13)
      {
        hora=1;
      }
    }
  }
```

```
    visual(); //Llama rutina de visualización de datos
    ISR_Timer_ClearPending();
  }
  CY_ISR(Int_SW) //Define rutina de interrupción de los GPIOs
  {
    switch (SW_Read()){ //Lee estado de pulsadores
      case 1:
        hora++; //Incrementa variable de hora
        if (hora==12) am_pm=~am_pm; //Cambia estado de bandera
        if (hora==13)
        {
          hora=1; //Ajusta la hora
        }
      break;
      case 2:
        min++; //Incrementa variable de minutos
        seg=0;
        if (min==60)
        {
          min=0; //Ajusta los minutos
        }
      break;
      default:
      break;
    }
    CyDelay(100);
    SW_ClearInterrupt(); //Borra solicitudes de interrupción
  }
  int main()
  {
    CyGlobalIntEnable; /* Enable global interrupts. */
```

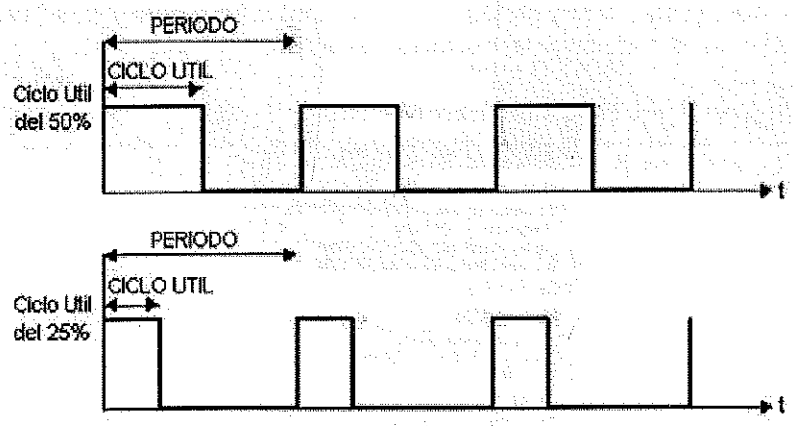
```
    ISR_Timer_StartEx(Int_Timer); // Define interrupción del Timer
    ISR_SW_StartEx(Int_SW); //Define interrupción de los GPIOs
    LCD_Start(); //Inicializa el bloque LCD
    Timer_Start(); //Inicializa y habilita en Timer
    seg=0; //Inicializa variables de tiempo
    min=0;
    hora=12;
    am_pm=0; //Inicializa bandera de formato am pm
    visual(); //Llama rutina de visualización de datos
    for(;;)
    {
      /* Place your application code here. */
    }
  }
```

Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC®, y luego cablear cada uno de los componentes a la tarjeta de desarrollo.

PWM del PSoC5®LP

El bloque PWM (modulación del ancho de pulso) es utilizado para generar una señal con un periodo constante y un ciclo útil variable, lo que permite realizar control del voltaje promedio visto por dispositivos eléctricos y electrónicos como bombillos, LED, motores, entre otros. El ciclo útil de la señal generada por el bloque PWM es el tiempo relativo utilizado en los estados altos y bajos del periodo de la señal (figura 135).

Figura 135. Señal PWM con diferentes ciclos útiles



Fuente: [8].

Los moduladores de ancho de pulso pueden tener diferentes frecuencias y resoluciones; la frecuencia está definida por el periodo y la resolución es definida por el número de pasos discretos en los que se puede dividir el periodo de dicha señal. Por ejemplo, un PWM de 8 bits permite manejar el ciclo útil de la señal en 256 pasos, es decir, 28 pasos discretos. En este caso el paso más pequeño será de 0,39 % que resulta de dividir 1 entre 256 posibles pasos [8].

Un uso común del modulador del ancho de pulso es la conversión digital a analógico usando filtros externos. El voltaje analógico generado es proporcional al ciclo útil de la señal, es decir, un ciclo útil del 50 % generará aproximadamente la mitad del voltaje analógico máximo y un ciclo útil del 25 %, un cuarto del voltaje analógico máximo. Los moduladores de ancho de pulso también se utilizan para controlar la velocidad de motores y para el control de la carga de las baterías [8].

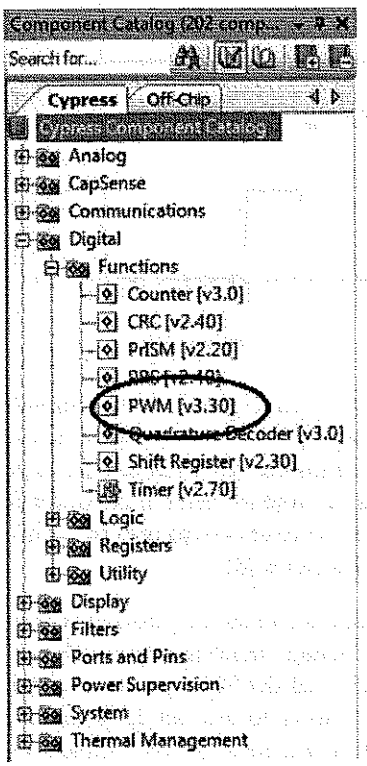
Características y uso del bloque PWM

El bloque PWM (el manual con todos los detalles de operación y funcionamiento se encuentra con el nombre de PWM_v3_30.pdf) tiene las siguientes características:

- Resolución de 8 o 16 bits.
- Múltiples modos de salida de ancho de pulso.
- Disparo configurable.
- Captura configurable.
- Habilitación por hardware o software configurable.
- Banda muerta configurable.
- Herramienta de configuración incluida en el PSoC Creator.
- Implementaciones UDB y FF, esta última solo para PSoC®3 y PSoC5®LP.

Para seleccionar en el PSoC Creator el bloque PWM denominado PWM [v3.30], el usuario debe buscar la carpeta Digital del catálogo de componentes, subcarpeta Functions y seleccionarlo entre las opciones disponibles allí (figura 136).

Figura 136. Selección del bloque PWM

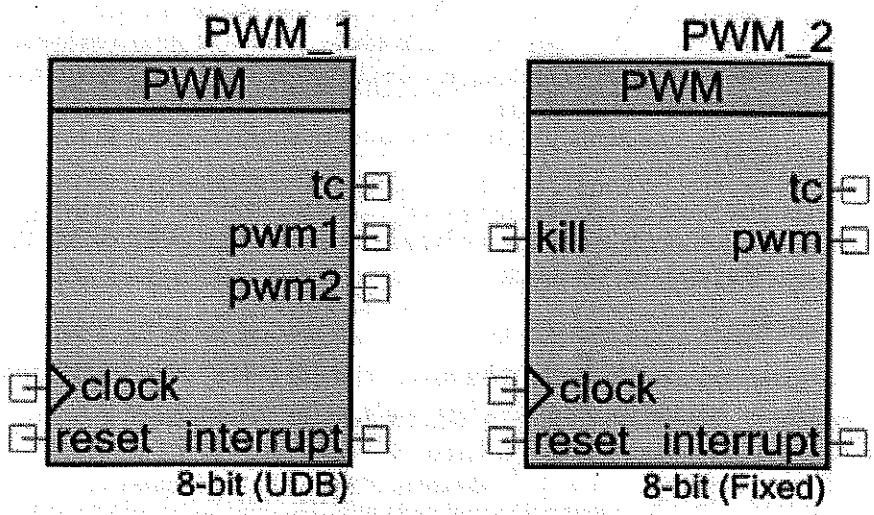


Fuente: elaboración propia.

Señales de entrada/salida del PWM T2

La figura 137 muestra el bloque PWM con todas las posibles señales de entrada/salida soportadas por las implementaciones UDB (izquierda) y FF (derecha), algunas de ellas pueden ser eliminadas en la configuración si no son necesarias para la aplicación.

Figura 137. Señales de entrada/salida del PWM



Fuente: elaboración propia.

- clock (input):

  Señal de reloj de entrada que define la frecuencia de operación del PWM, el contador del PWM se modifica por cada flanco de subida de esta señal. Esta entrada no puede ser eliminada.

- reset (input):

  Entrada de reset sincrónico, que se encarga de reiniciar los contadores internos del PWM. Esta entrada no puede ser eliminada.

- enable (input):

  Entrada de habilitación por hardware del PWM, activa en alto; es decir, cuando la entrada tiene un nivel bajo el PWM está deshabilitado. Esta entrada puede ser eliminada.

- kill (input):

  Señal activa en alto. En implementación FF habilita las salidas ph1 y ph2 y las hace cero cuando la señal kill está habilitada. En la implementación UDB las salidas pwm1, pwm2, ph1 y ph2 se hacen cero cuando la señal kill está habilitada. Esta entrada puede ser eliminada.

- cmp_sel (input):

  Señal utilizada para seleccionar por hardware la salida PWM utilizada en la implementación UDB. Cuando la entrada tiene un nivel bajo la salida pwm utiliza la señal pwm1 y cuando la entrada tiene un nivel alto la salida pwm utiliza la señal pwm2. Esta entrada puede ser eliminada.

- capture (input):

  Permite capturar el valor actual de los contadores cuando se genera en la entrada alguno de los diversos tipos de excitación (flanco de subida, bajada o ambos). Esta entrada puede ser eliminada.

- trigger (input):

  Detecta una señal de entrada que es utilizada para habilitar la operación del PWM, señal disponible únicamente para la implementación del PWM UDB. Esta entrada puede ser eliminada.

- tc (output):

  Salida sincrónica denominada Terminal counter que indica, al hacerse 1, la finalización del periodo del contador. Esta salida no puede ser eliminada.

- interrupt (output):

  Genera una señal de interrupción unificada internamente por medio de una función OR del grupo de posibles fuentes de interrupción del bloque PWM. Esta salida puede ser eliminada.

- pwm/pwm1 (output):

  Es la primera o única señal de salida generada por el bloque PWM. Esta salida puede ser eliminada.

- pwm2 (output):

  Segunda señal de salida para el bloque PWM con implementación UDB. Esta salida puede ser eliminada.

- ph1/ph2 (output):

  Salidas de fase cuando la opción de banda muerta está activa con la señal kill, señales disponibles únicamente para la implementación del PWM UDB. Estas salidas pueden ser eliminadas.

Configuración del bloque PWM

Dependiendo del tipo de PWM por utilizar FF (Fixed-function) o UDB (Universal Digital Block), la ventana de configuración incluye o habilita diferentes parámetros.

Para la ventana de configuración del bloque PWM tipo UDB se pueden modificar

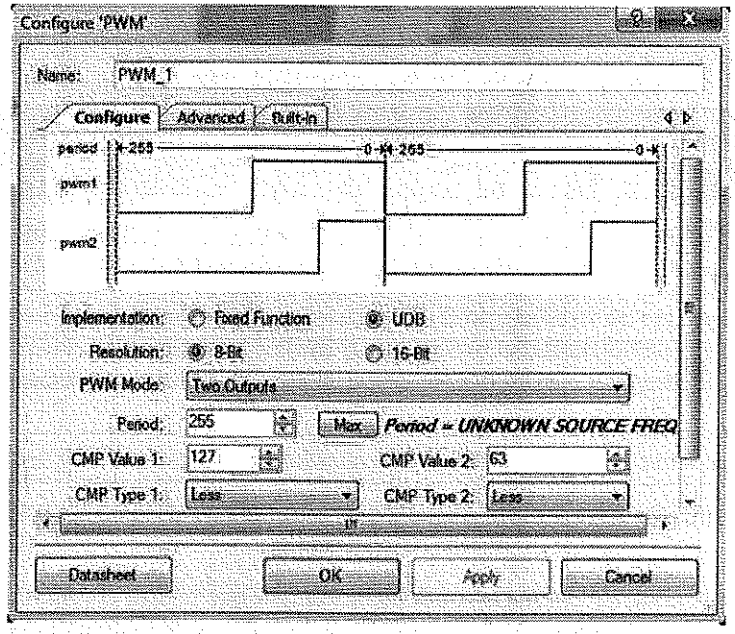
o configurar los parámetros en las pestañas Configure y Advanced. Para la pestaña Configure los parámetros por configurar son:

- Nombre (Name).
- La resolución del PWM (Resolution).
- Modo de operación del PWM (PWM Mode).
- El periodo generado a partir de la frecuencia de la señal de reloj de entrada (Period).
- Valor de comparación de la salida pwm1 (CMP Value 1).
- Valor de comparación de la salida pwm2 (CMP Value 2).
- Tipo de comparación de la salida pwm1 (CMP Type 1).
- Tipo de comparación de la salida pwm2 (CMP Type 2).
- Banda muerta (Dead Band).

En la figura 138 puede verse la ventana de configuración del bloque PWM con las funcionalidades antes nombradas para la pestaña Configure.

El parámetro Resolution permite seleccionar entre dos valores posibles la resolución manejada por el PWM: 8-Bit (valor por defecto) o 16-Bit, este parámetro determina el periodo de salida de la señal PWM generada.

Figura 138. Ventana de configuración del PWM UDB



Fuente: elaboración propia.

Con el parámetro PWM Mode es posible definir el modo de operación del PWM entre seis opciones posibles:

- *One Output:* genera una sola señal PWM, denominada pwm.
- *Two Output:* genera dos señales PWM independientes (el periodo entre ellas es común), denominadas pwm1 y pwm2 (valor por defecto).
- *Dual Edge:* genera una sola señal PWM, resultado de aplicar la función AND entre las señales internas PWM1 y PWM2. Se genera una única señal denominada pwm.
- *Center Align:* genera una señal PWM alineada al centro del periodo configurado; esto lo consigue permitiendo que el contador cuente desde cero hasta el valor máximo configurado y desde este valor hasta cero. Se genera una única señal denominada pwm.
- *Dither:* genera una señal PWM que es seleccionada de manera automática por medio de una máquina de estados entre las señales internas PWM1 y PWM2. Se selecciona el aumento del ancho del pulso entre cuatro valores posibles 0.00, 0.25, 0.50 o 0.75, con lo que el hardware selecciona entre las dos señales PWM la adecuada para que esto suceda. Se genera una única señal denominada pwm.
- *Hardware Select:* genera una señal PWM seleccionada por hardware haciendo uso de la señal cmp_sel, Cuando la señal cmp_sel está en nivel bajo se selecciona la señal interna PWM1 y cuando la señal cmp_sel está en nivel alto se selecciona la señal interna PWM2; en cualquiera de los dos casos se genera una única señal denominada pwm.

En la tabla 25 puede verse gráficamente la forma de la señal PWM generada para cada una de las seis opciones posibles del parámetro PWM Mode.

Valor parámetro PWM Mode	Forma de onda de señal PWM generada
One Output	
Two Output	
Dual Edge	
Center Align	
Dither	
Hardware Select	

Tabla 25. Forma de la señal PWM generada para cada opción del parámetro PWM Mode
Fuente: elaboración propia.

El parámetro Period determina la cantidad de conteos que realiza el contador interno del PWM, con este valor y con la frecuencia de la señal de reloj de entrada se determina el periodo de la señal generada por el módulo.

Los parámetros CMP Value 1 y CMP Value 2 definen el valor de comparación para las señales PWM1 y PWM2 respectivamente; el valor seleccionado (que debe ser menor o igual al valor seleccionado para el periodo de la señal) determina el momento de conmutación de la señal de salida. Esto finalmente determina el ciclo útil de la señal PWM generada.

Con los parámetros CMP Type 1 y CMP Type se define el tipo de comparación realizada al momento de general la señal PWM entre seis valores posibles:

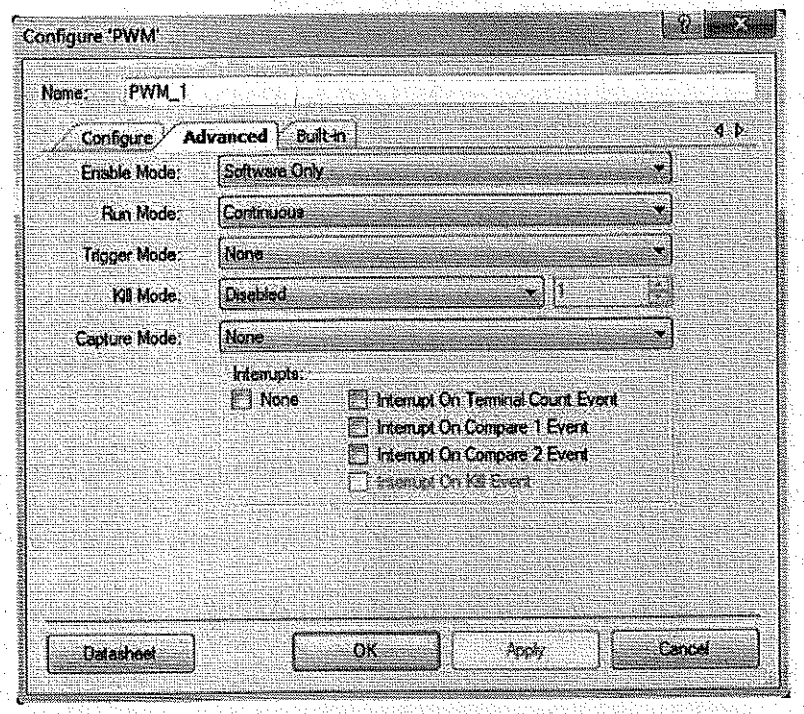
- *Less:* compara el valor actual del contador con el valor de comparación definido en el parámetro CMP Value, si el valor del contador es menor que el valor de comparación genera un nivel alto en la salida, de lo contrario forma un nivel bajo (valor por defecto).
- *Less or Equal:* compara el valor actual del contador con el valor de comparación definido en el parámetro CMP Value, si el valor del contador es menor o igual que el valor de comparación genera un nivel alto en la salida, de lo contrario crea un nivel bajo.
- *Greater:* compara el valor actual del contador con el valor de comparación definido en el parámetro CMP Value, si el valor del contador es mayor que el valor de comparación genera un nivel alto en la salida, de lo contrario forma un nivel bajo.
- *Greater or Equal:* compara el valor actual del contador con el valor de comparación definido en el parámetro CMP Value, si el valor del contador es mayor o igual que el valor de comparación produce un nivel alto en la salida, de lo contrario genera un nivel bajo.
- *Equal:* compara el valor actual del contador con el valor de comparación definido en el parámetro CMP Value, si el valor del contador es igual que el valor de comparación produce un pulso periódico en la salida del PWM.
- *Firmware Control:* esta opción proporciona un modelo de uso de recursos más flexible en la que el modo de comparación se puede establecer en tiempo de ejecución. Los modos de comparación pueden ser modificadas en cualquier momento por funciones llamadas de la API del bloque.

El parámetro Dead Band activa o desactiva la funcionalidad de banda muerta del PWM, al activar la función las señales ph1 y ph2 se habilitan como salidas del bloque. Este parámetro solo aplica para la salida pwm o pwm1. Para este se tienen tres opciones:

- *Disabled:* banda muerta deshabilitada (valor por defecto).
- *2-4 Clock Cycles:* habilita la banda muerta para la salida pwm o pwm1 durante máximo cuatro conteos.
- *2-256 Clock Cycles:* habilita la banda muerta para la salida pwm o pwm1 durante máximo 256 conteos.

La otra pestaña de configuración, denominada Advanced, del bloque PWM con implementación UDB puede verse en la figura 139, allí pueden configurarse los siguientes parámetros: forma de activación del PWM (Enable Mode), modo de inicio del PWM (Run Mode), modo de disparo del PWM (Trigger Mode), modo de operación del modo kill (Kill Mode), modo de captura de entrada del PWM (Capture Mode) y finalmente el manejo de las interrupciones del bloque PWM (Interrupts).

Figura 139. Pestaña de configuración Advanced del PWM UDB



Fuente: elaboración propia.

El parámetro Enable Mode define la forma como se habilitan las funcionalidades del PWM entre tres opciones posibles:

- *Software Only:* el PWM es habilitado únicamente por software cuando se habilita el bit correspondiente en el registro de control de este (valor por defecto).
- *Hardware Only:* el PWM es habilitado únicamente por hardware; para esto se hace uso de la señal enable que debe asegurarse a un nivel alto.
- *Hardware and Software:* el PWM se habilita con una instrucción del software o por una señal del hardware del bloque.

Con el parámetro Run Mode se define cómo se dispara el PWM para iniciar y continuar operando; este parámetro maneja tres opciones:

- *Continuous:* el PWM opera de manera continua (valor por defecto).

- *One Shot with Single Trigger:* el PWM corre durante un periodo cada vez que parece un evento de disparo.
- *One Shot with Multi Trigger:* el PWM corre durante un periodo y se detiene esperando a que aparezca un nuevo evento de disparo. El PWM no puede ser activado con un llamado de las API.

Para generar el disparo del PWM por hardware se utiliza el parámetro Trigger Mode entre cuatro valores posibles:

- *None:* disparo desactivado, el PWM opera sin necesidad de esta señal (valor por defecto).
- *Rising Edge:* el disparo del PWM se genera por flanco de subida.
- *Falling Edge:* el disparo del PWM se genera por flanco de bajada.
- *Either Edge:* el disparo del PWM se genera por flanco de subida o por flanco de bajada.

El parámetro Kill Mode define cómo el hardware se encarga de las salidas del PWM cuando la entrada kill está activa. Este parámetro permite seleccionar la configuración entre cinco valores posibles:

- *Disabled:* modo kill deshabilitado (valor por defecto).
- *Asynchronous:* las salidas del PWM están deshabilitadas mientras la entrada kill esté activa.
- *Single Cycle:* las salidas del PWM están deshabilitadas mientras la entrada kill esté activa y no se reactivan mientras finalice el periodo actual.
- *Latched:* las salidas del PWM están deshabilitadas mientras la entrada kill esté activa y permanece deshabilitado hasta que se reinicie el PWM.
- *Minimum Time:* las salidas del PWM están deshabilitadas mientras la entrada kill esté activa y no se vuelve a habilitar hasta que haya trascurrido el tiempo mínimo kill que es configurable por el usuario.

Con el parámetro Capture Mode se define qué evento del hardware causa una captura del valor del contador del PWM entre cuatro valores posibles:

- *None:* captura desactivada (valor por defecto).
- *Rising Edge:* el evento de captura del PWM se genera por flanco de subida.
- *Falling Edge:* el evento de captura del PWM se genera por flanco de bajada.
- *Either Edge:* el evento de captura del PWM se genera por flanco de subida o por flanco de bajada.

Las interrupciones del PWM son configuradas haciendo uso del parámetro Interrupts que permite seleccionar la fuente de interrupción entre cuatro posibles:

- *None:* ninguna fuente de interrupción activa.

- *Interrupt on Terminal Count Event:* interrupción por final de periodo de la señal PWM.
- *Interrupt on Compare 1 Event.*
- *Interrupt on Compare 2 Event.*

Rutinas utilizadas para el manejo del bloque PWM

Las funciones API para el bloque PWM permiten configurar, controlar y poner a andar sus diversos modos de operación, sin importar el tipo de implementación seleccionado. Para la sintaxis de estas funciones debe tenerse en cuenta que la primera parte del nombre de la función (PWM_) debe ser reemplazada por el nombre que el usuario le haya asignado al bloque.

Las funciones que pueden ser utilizadas para configurar y manejar el bloque son:

- PWM_Start():

  Inicializa variables asociadas al PWM e inicializa y habilita todos los parámetros de configuración seleccionados previamente para el PWM. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- PWM_Stop():

  Deshabilita la operación del PWM haciendo reset en el bit correspondiente en el registro de control. Si el parámetro Enable Mode tiene selecciona da la opción Hardware Only; esta función no tiene efecto sobre el PWM. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- PWM_WriteCompare():

  Escribe el valor de comparación utilizado por el PWM cuando se utilizan los modos Dither, Center Aligned o One Output para el parámetro PWM Mode. Como parámetro de entrada utiliza un entero sin signo de 8 o 16 que depende de la resolución seleccionada para el PWM, el cual contiene el nuevo valor de comparación asignado al PWM. La función no retorna ningún valor.

- PWM_WriteCompare1():

  Escribe el valor de comparación utilizado por el PWM para la salida pwm1. Como parámetro de entrada utiliza un entero sin signo de 8 o 16 que depende de la resolución seleccionada para la salida pwm1, el cual contiene el nuevo valor de comparación asignado al PWM. La función no retorna ningún valor.

- PWM_WriteCompare2():

  Escribe el valor de comparación utilizado por el PWM para la salida pwm2 y aplica solamente para implementaciones UDB. Como parámetro de entrada utiliza un entero sin signo de 8 o 16 que depende de la resolución seleccionada para la salida pwm2, el cual contiene el nuevo valor de comparación asignado al PWM. La función no retorna ningún valor.

- PWM_ReadStatusRegister():

  Lee el valor actual del registro de estado del PWM. No utiliza ningún parámetro de entrada y retorna un entero de 8 bits sin signo con el valor leído del registro, que puede ser leído en formato de bits si se requiere. Si por ejemplo se desea actualizar el valor de comparación del PWM con implementación UDB (denominado PWM), se necesita conocer el estado de la bandera TC (Terminal Counter) del registro de estado del PWM. Para esto se podría utilizar las siguientes líneas de código:

```
while (PWM_STATUS_TC==0);
PWM_WriteCompare(ciclo);
```

- PWM_ReadCounter():

  Lee el valor actual del contador del PWM (captura por software) y aplica solamente para implementaciones UDB. No utiliza ningún parámetro de entrada y retorna un entero sin signo de 8 o 16 bits, que depende de la resolución seleccionada para el PWM.

- PWM_ReadCapture():

  Lee el valor del contenido de captura del PWM almacenado en la FIFO, aplica solamente para implementaciones UDB. No utiliza ningún parámetro de entrada y retorna un entero sin signo de 8 o 16 bits, que depende de la resolución seleccionada para el PWM.

- PWM_WriteCounter():

  Escribe un nuevo valor directamente en el registro de conteo del PWM; esta función es válida solamente para implementación UDB. Como parámetro de entrada utiliza un entero sin signo de 8 o 16 que depende de la resolución seleccionada para el PWM, el cual contiene el nuevo valor asignado al PWM. La función no retorna ningún valor.

- PWMr_ClearFIFO():

  Borra toda la información del FIFO de captura del PWM; esta función está disponible solo para implementaciones UDB. No utiliza ningún parámetro de entrada ni retorna ningún valor.

Ejemplo de uso del bloque PWM

Dimmer digital controlado por PWM

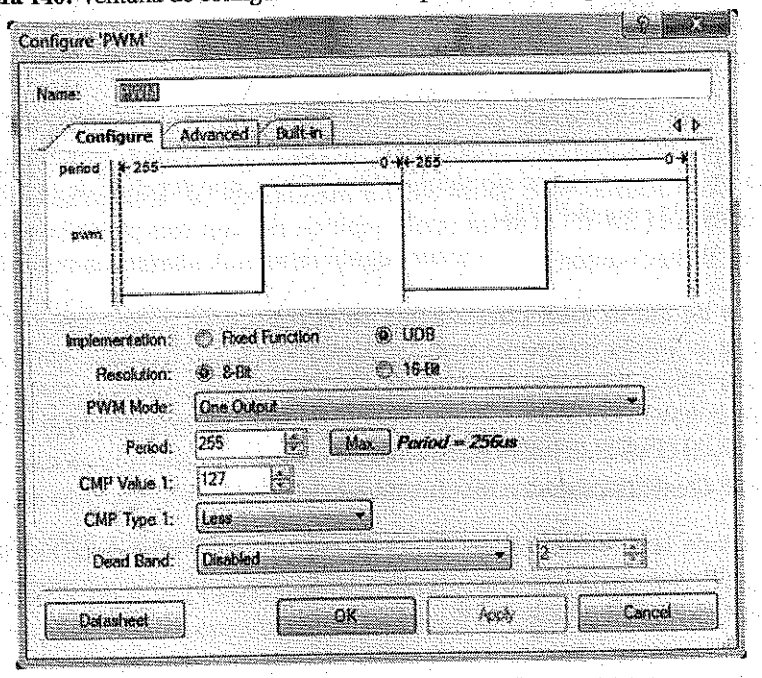
El ejemplo que se presentará a continuación muestra la implementación de un dimmer digital haciendo uso del bloque PWM del PSoC5®LP. El control del dimmer se realizará haciendo uso de dos pulsadores conectados a las entradas del PSoC®; con esto tendrá un valor de voltaje promedio que se verá reflejado con la intensidad del brillo de un LED.

Solución

Para la solución planteada se utilizará el bloque PWM, con salida única, con implementación UDB con resolución de 8 bits; esto último no es un requerimiento del problema, así que cualquier resolución aplicaría.

Inicialmente se hace uso del bloque PWM [v3.30] con el nombre PWM; en la pestaña Configure se seleccionan las siguientes opciones de configuración: el parámetro Resolution en 8-Bit, el parámetro PWM Mode con la opción One Output seleccionada, el valor Period en 255 con lo que obtiene un periodo de 256us. Para esto se debe configurar la frecuencia del bloque Clock_1 conectado a la entrada clock del PWM en 1MHz, el valor CMP Value 1 en 127 con lo que obtiene un ciclo útil inicial del 50%, el parámetro CMP Type 1 con la opción Less, así el PWM opera con lógica positiva y el parámetro Dead Band con la opción Disabled seleccionada. En la pestaña Advanced se dejan los parámetros de configuración que están por defecto. En la figura 140 puede verse la configuración realizada para la pestaña Configure.

Figura 140. Ventana de configuración del bloque PWM UDB para el ejemplo 10-1



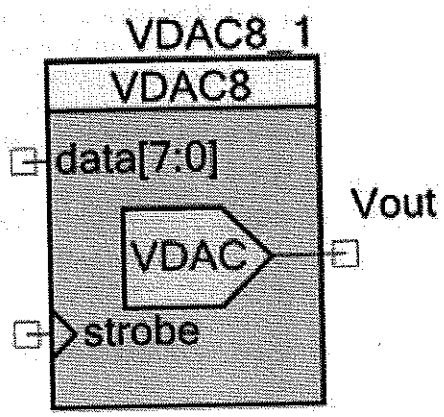
Fuente: elaboración propia.

El siguiente bloque utilizado en el diseño corresponde a uno de E/S de dos bits configurado como entrada, con nombre SW, la opción HW connection deshabilitada, la opción External terminal habilitada, donde se conectan dos pulsadores N.A. y la tierra del circuito, en el parámetro Drive mode se selecciona Resistive pull up, además, en la pestaña Input de este bloque se modifica el parámetro Interrupt seleccionando la opción Falling edge, para habilitar la interrupción del bloque de E/S.

Un tercer bloque utilizado para la implementación del diseño corresponde a uno de E/S de un bit configurado como salida, con nombre LED, la opción HW connection habilitada, la opción External terminal habilitada, se conecta a este bloque una resistencia, un LED y la tierra del circuito.

Por último, se incluye un bloque de interrupción con nombre ISR_SW, para controlar la interrupción del bloque SW que facilita en manejo de los pulsadores, al que únicamente se le modifica el nombre en la configuración. En la figura 141 puede verse el diagrama circuital de la solución propuesta para el ejemplo.

Figura 141. Diagrama circuital ejemplo 10-1



Fuente: elaboración propia.

Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: el bit P2[1] para la conexión del LED y los bits P2[3:2] para la conexión de los pulsadores N.A. que permitirán al usuario el ajuste del dimmer.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 10-1:

```
/* ==========================================================
 *
 * Programa que implementa un dimmer digital controlado por medio
 * de un par de pulsadores N.A., con ellos se puede aumentar o
 * disminuir la intensidad del brillo de un LED conectado a una de
 * las salidas del PSoC.
 *
 * ==========================================================
 */
#include <project.h>
```

```
/* Definición de variables */
volatile uint8 cont;
CY_ISR(Int_SW) //Define rutina de interrupción de los GPIOs
{
  switch (SW_Read()){ //Lee estado de pulsadores
    case 1:
    if (cont!=255) cont++; //Incrementa variable de conteo
    while (PWM_STATUS_TC==0);
      PWM_WriteCompare(cont); //Actualiza ciclo util
    break;
    case 2:
      if (cont!=0) cont--; //Decrementa variable de conteo
      while (PWM_STATUS_TC==0);
        PWM_WriteCompare(cont); //Actualiza ciclo util
    break;
    default:
    break;
  }
  CyDelay(100);
  SW_ClearInterrupt(); //Borra solicitudes de interrupción
}
int main()
{
  CyGlobalIntEnable; /* Enable global interrupts. */
  ISR_SW_StartEx(Int_SW);
  PWM_Start();
  cont=127; //Valor inicial del PWM
  for(;;)
  {
    /* Place your application code here. */
  }
}
```

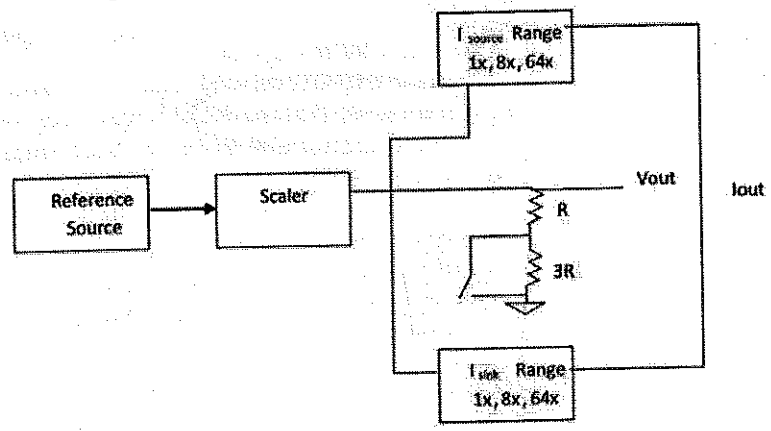
Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC®, y luego cablear cada uno de los componentes a la tarjeta de desarrollo.

Conversor digital a analógico del PSoC5®LP

El bloque conversor digital a analógico (DAC) es utilizado para generar un voltaje equivalente a una combinación binaria dada; esto es útil dado que en muchas tareas se requiere a partir de un valor digital generar valores de voltaje analógicos para utilizarlos como referencias de voltaje, o para generar formas de onda particulares.

El DAC del PSoC5®LP genera u voltaje o una corriente de salida haciendo uso de un espejo de corriente; esta última es una imagen reflejada de una fuente de referencia del DAC. La corriente generada se desvía por una escalera que da el valor de voltaje o corriente correspondiente del DAC (figura 142).

Figura 142. Diagrama de bloques del DAC del PSoC5®LP



Fuente: [5, p. 58].

Características y uso del bloque DAC

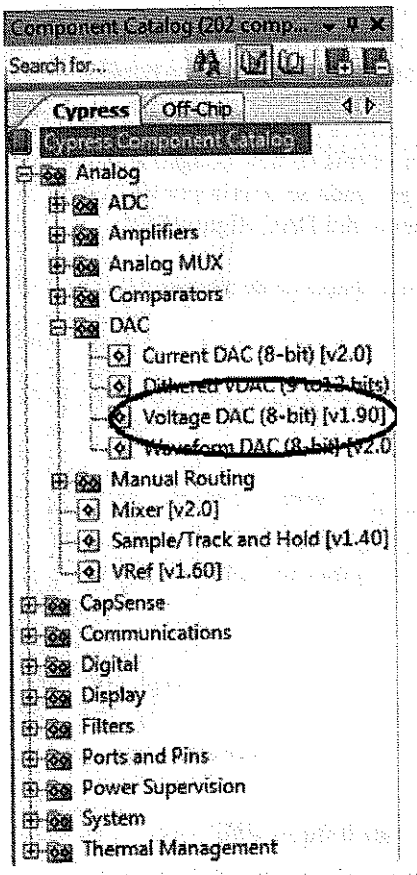
El bloque DAC (el manual con todos los detalles de operación y funcionamiento se encuentra con el nombre de VDAC8_v1_90.pdf) tiene las siguientes características:

- Cuatro bloques DAC.
- Resolución de 8.
- Dos rangos de voltaje de salida: 1.020v (4mv/bit) y 4.080v (16mV/bit).

- Fuente de reloj seleccionable.
- Velocidad de conversión de hasta 1Msps.
- Fuente de datos proveniente de la CPU, del bloque DMA o de componentes digitales.
- Resistencia dedicada para el pin de salida.

Para seleccionar en el PSoC Creator el bloque DAC de voltaje denominado Voltage DAC (8-bit) [v1.90], el usuario debe buscar la carpeta Analog del catálogo de componentes, subcarpeta DAC y seleccionarlo entre las opciones disponibles allí (figura 143).

Figura 143. Selección del bloque DAC

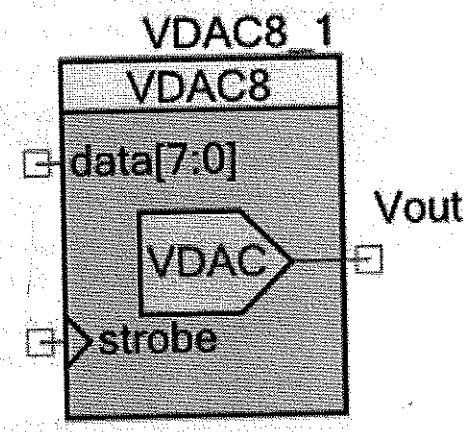


Fuente: elaboración propia.

Señales de entrada/salida del DAC de voltaje

La figura 144 muestra el bloque DAC de voltaje con todas las posibles señales de entrada/salida soportadas, algunas de ellas pueden ser eliminadas en la configuración si no son necesarias para la aplicación.

Figura 144. Señales de entrada/salida del DAC



Fuente: elaboración propia.

- Vout (analog output):

  Señal de salida analógica que genera el voltaje producido por el DAC, puede ser conectada a cualquier pin analógico del PSoC®. Esta salida no puede ser eliminada.

- data[7:0] (input):

  Bloque de entrada de ocho bits que se conecta al bus de datos del DAC, utilizado para recibir la combinación binaria que generara el voltaje de salida. Esta entrada puede ser eliminada.

- strobe (input):

  Señal de entrada que activa la carga de datos al DAC; esta carga se realiza en el flanco de subida de la señal. Los datos utilizados por el DAC pueden venir de la CPU, el DMA o de las líneas entrada data.

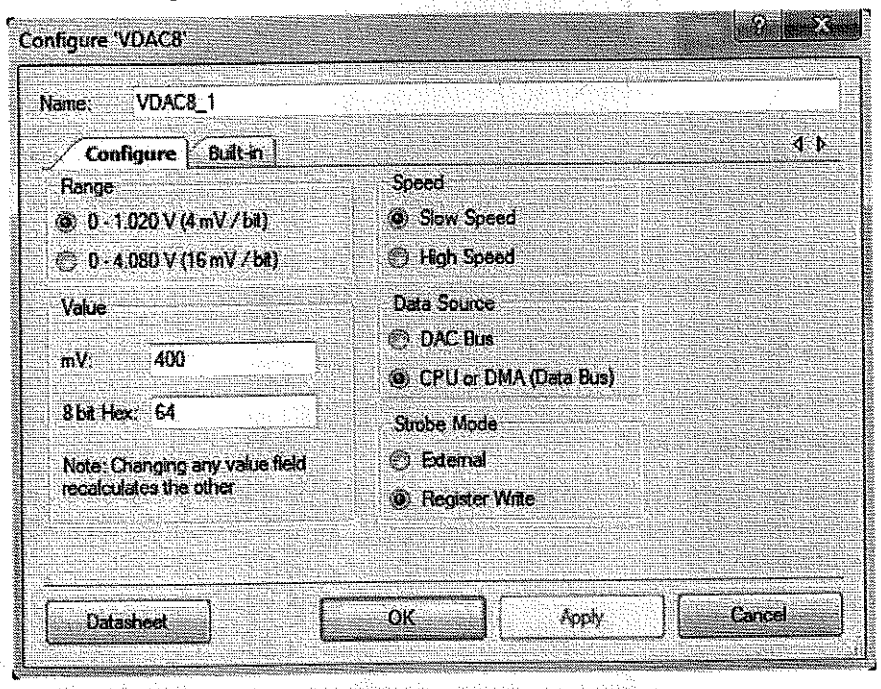
Configuración del bloque DAC de voltaje

Para la ventana de configuración del bloque DAC de voltaje se pueden modificar o configurar los siguientes parámetros:

- Nombre (Name).
- Rango del voltaje de salida (Range).
- Valor de salida inicial en mV y en hexadecimal.
- Velocidad de respuesta del DAC (Speed).
- Fuente de datos del DAC (Data Source).
- Forma de carga de datos al DAC (Strobe Mode).

En la figura 145 puede verse la ventana de configuración del bloque DAC de voltaje con las funcionalidades antes nombradas.

Figura 145. Ventana de configuración del DAC de voltaje



Fuente: elaboración propia.

El parámetro Range permite seleccionar entre dos valores posibles el rango del voltaje de salida generado por el DAC:

- 0 – 1.020v (4mV/bit) (valor por defecto).
- 0 – 4.080v (16mV/bit).

Con el parámetro Value se puede determinar el valor de salida inicial del DAC; este valor puede ser configurado en mV o como un valor hexadecimal de 8 bits.

El parámetro Speed permite seleccionar dos tipos de respuesta del DAC que modifican velocidad de respuesta y consumo de corriente:

- *Slow Speed:* el tiempo de respuesta del DAC es lento y el consumo de corriente es bajo (valor por defecto).
- *High Speed:* el tiempo de respuesta del DAC es rápido y el consumo de corriente también lo es.

Para seleccionar la fuente de los datos utilizados por el DAC se recurre al uso del parámetro Data Source; es posible seleccionar este parámetro entre dos valores posibles:

- *DAC Bus:* los datos ingresan al DAC por la entrada data, además de manera automática se habilita la señal de entrada strobe que permite determinar el momento del ingreso de los datos.
- *CPU or DMA (Data Bus):* los datos del DAC provienen de la CPU o del módulo DMA del PSoC® (valor por defecto).

El parámetro Strobe Mode permite seleccionar el tipo de activación para la escritura de datos en el DAC entre dos valores posibles:

- *External:* la carga de datos en el DAC se realiza por medio de una señal de hardware que ingresa por la línea strobe.
- *Register Write:* la carga de datos se realiza por medio de una función API del PSoC® (valor por defecto).

Rutinas utilizadas para el manejo del bloque DAC de voltaje

Las funciones API para el bloque DAC permiten configurar, controlar y poner a andar sus diversos modos de operación. Para la sintaxis debe tenerse en cuenta que la primera parte del nombre de la función (VDAC8_) debe ser reemplazadq por el nombre que el usuario le haya asignado al bloque.

Las funciones que pueden ser utilizadas para configurar y manejar el bloque son:

- VDAC8_Start():

  Inicializa variables asociadas al DAC e inicializa y habilita todos los parámetros de configuración seleccionados previamente para el DAC. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- VDAC8_SetSpeed():

  Asigna el tipo de respuesta de salida del DAC. Como parámetro de entrada utiliza un entero sin signo de 8 bits representado con las constantes: VDAC8_LOWSPEED y VDAC8_HIGHSPEED. La función no retorna ningún valor.

- VDAC8_SetRange():

  Asigna el valor del rango del voltaje de salida del DAC. Como parámetro de entrada utiliza un entero sin signo de 8 bits representado con las constantes: VDAC8_RANGE_1V y VDAC8_RANGE_4V. La función no retorna ningún valor.

- VDAC8_SetValue():

  Escribe el valor de salida que se desea asignar al DAC. Como parámetro de entrada utiliza un entero sin signo de 8 bits. La función no retorna ningún valor. Si por ejemplo se desea generar un voltaje de salida de 2v en un bloque DAC denominado VDAC, trabajando en el rango de 4.080v, se podría utilizar la siguiente línea de código:

```
VDAC8_SetValue(0x7F);
```

Ejemplo de uso del bloque DAC

Generador de señales con DAC de voltaje

El ejemplo que se presentará a continuación muestra la implementación de un generador de señales con el bloque DAC de voltaje del PSoC5®LP. Las formas de onda generadas serán: cuadrada, triangular, diente de sierra y seno; estas se seleccionaran por medio de un dip switch de dos interruptores.

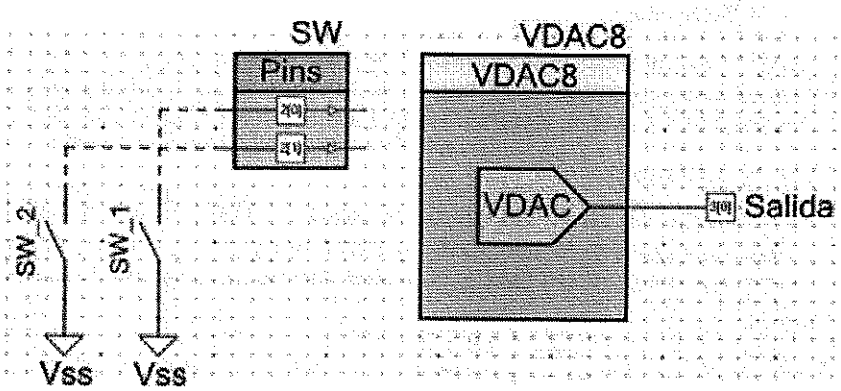
Solución

Para la solución planteada se utilizará el bloque DAC conectado a una GPIO analógica. Dado que la frecuencia de las señales generadas no es un parámetro requerido en el ejemplo, el proceso se realizará por software para tener así señales de baja frecuencia.

Inicialmente se hace uso del bloque VDAC8 [v1.90] con el nombre VDAC8, al que se le modifican dos parámetros: al parámetro Range se le selecciona el valor 0 – 4.080v (16mV/bit) y al parámetro Value se le asigna el valor 0mV; los demás parámetros permanecen sin modificaciones. A la salida del bloque VDAC8 se le conecta un bloque de E/S analógico llamado Salida.

Se utiliza también un bloque de E/S de dos bits configurado como entrada, con nombre SW, con la opción HW connection deshabilitada y la opción External terminal habilitada, en la cual se conectan dos interruptores y la tierra del circuito, en la opción Drive mode se selecciona Resistive pull up. En la figura 146 puede verse el diagrama circuital del diseño final sugerido.

Figura 146. Diagrama circuital ejemplo 11-1



Fuente: elaboración propia.

Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: el bit P3[0] para la conexión de la salida del DAC y los bits P2[1:0] para la conexión del dip switch que permitirán al usuario seleccionar la forma de onda de salida.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 11-1:

```
/* ==========================================================
 *
 * Programa que implementa un dimmer digital controlado por medio
 * de un par de pulsadores N.A., con ellos se puede aumentar o
 * disminuir la intensidad del brillo de un LED conectado a una de
 * las salidas del PSoC.
 *
 * ==========================================================
 */
#include <project.h>
/* Definición de variables */
uint8 i;
const uint8 tabla[]={
128,131,134,137,140,144,147,150,153,156,159,162,165,168,171,174,
177,180,182,185,188,191,194,196,199,201,204,206,209,211,214,216,
218,220,222,224,226,228,230,232,234,236,237,239,240,242,243,244,
246,247,248,249,250,251,251,252,253,253,254,254,254,255,255,255,
255,255,255,255,254,254,253,253,252,252,251,250,249,248,247,246,
245,244,242,241,240,238,236,235,233,231,229,227,225,223,221,219,
217,215,212,210,208,205,203,200,197,195,192,189,187,184,181,178,
175,172,169,166,164,160,157,154,151,148,145,142,139,136,133,130,
126,123,120,117,114,111,108,105,102,99,96,92,89,87,84,81,
78,75,72,69,67,64,61,59,56,53,51,48,46,44,41,39,37,35,33,
31,29,27,25,23,21,20,18,16,15,14,12,11,10,9,8,7,6,5,4,4,
3,3,2,2,1,1,1,1,1,1,1,2,2,2,3,3,4,5,5,6,7,8,9,10,12,13,14,16,
17,19,20,22,24,26,28,30,32,34,36,38,40,42,45,47,50,52,55,57,60,
62,65,68,71,74,76,79,82,85,88,91,94,97,100,103,106,109,112,116,
119,122,125,128
};
int main()
```

```
{
  VDAC8_Start(); //Inicializa el DAC
  for(;;)
  {
    switch (SW_Read()){ //Lee el dipswitch
      case 0: //Genera señal triangular
        for (i=0;i<255;i++){ //Genera pendiente positiva
          VDAC8_SetValue(i);
        }
        for (i=255;i>0;i--){ //Genera pendiente negativa
          VDAC8_SetValue(i);
        }
      break;
      case 1: //Genera señal seno
        for (i=0;i<255;i++){
          VDAC8_SetValue(tabla[i]); //Lee tabla con datos
        }
      break;
      case 2: //Genera señal cuadrada
        VDAC8_SetValue(0); //Genera ciclo bajo
        CyDelayUs(200);
        VDAC8_SetValue(255); //Genera ciclo alto
        CyDelayUs(200);
      break;
      case 3: //Genera señal diente de sierra
        for (i=0;i<255;i++){
          VDAC8_SetValue(i); //Genera pendiente positiva
        }
      break;
    }
  }
}
```

Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC® y luego cablear cada uno de los componentes a la tarjeta de desarrollo; además, se debe hacer uso de un osciloscopio para poder verificar las formas de onda. En la figura 147 puede verse una imagen tomada de un osciloscopio con el resultado obtenido para la señal seno.

Figura 147. Señal seno generada para el ejemplo 11-1

Fuente: elaboración propia.

Módulo UART del PSoC5®LP

El bloque UART del PSoC5®LP permite realizar comunicación serial, que es el proceso de envío de datos de un bit a la vez, de manera secuencial sobre una línea de comunicación, a diferencia de la comunicación paralela que requiere una línea de datos por cada bit por transmitir. Lo anterior es útil porque ahorra líneas físicas en la comunicación y además permite eliminar problemas como interferencia entre líneas, atenuación de los datos cuando la comunicación quiere realizarse entre puntos relativamente lejanos.

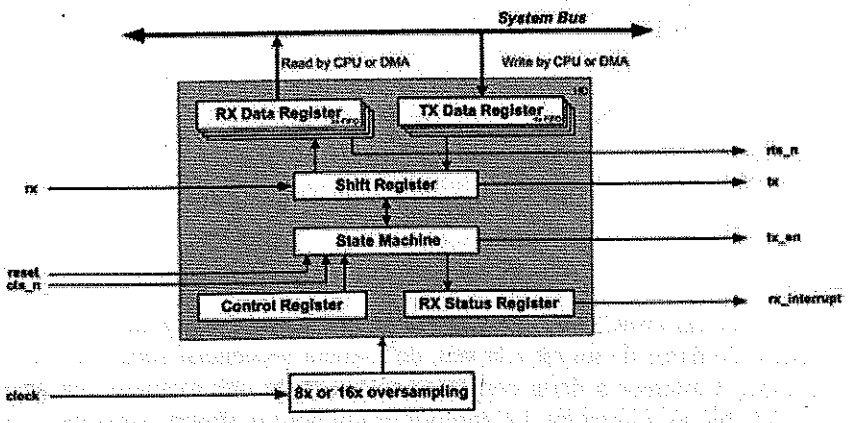
Aunque este tipo de comunicación lleva mucho tiempo en el mercado, aun es ampliamente utilizada ya que muchos equipos de comunicaciones como módems o GPS la manejan, además de muchos dispositivos OEM como conversores USB-serial, Bluethooth-Serial, LCD, módems GSM, brújulas digitales, etc.

Con el bloque UART del PSoC5®LP se pueden realizar comunicaciones en diferentes formatos: Full Duplex, Half Duplex, solo Rx o solo Tx. Todas los formatos soportados ofrecen la misma funcionalidad básica, difieren únicamente en la cantidad de recursos utilizados. Para ayudarle al procesamiento de los datos al bloque UART se incluyen buffers de Tx y Rx independientes, cada uno con longitud variable que puede ser modificada por el usuario en la ventana de configuración, con un tamaño mínimo de cuatro bytes.

La UART puede configurarse de manera sencilla eligiendo la velocidad de transmisión, paridad, número de bits de datos y el número de bits de inicio. La configuración más común para dispositivos RS232 es 8N1, que es la abreviatura de ocho bits de datos, sin paridad y un bit de parada, esta es la configuración predeterminada del bloque UART del PSoC5®LP. Por lo tanto, en la mayoría de las aplicaciones solo se tiene que ajustar la velocidad de transmisión. Un segundo uso común para el bloque UART está en redes RS485 multipunto, el bloque tiene la capacidad lógica y física de soportar esta norma.

Ya que existen variaciones de la UART comúnmente utilizada, el componente del PSoC5®LP proporciona soporte de configuración para el número de bits de datos, bits de parada, paridad, control de flujo de hardware y la generación de paridad y de detección; así es posible la comunicación con cualquier sistema actual. En la figura 148 puede verse el diagrama de bloques de la UART.

Figura 148. Diagrama de bloques del módulo UART



Fuente: [10, p. 35].

Características y uso del bloque UART T1

El bloque UART (el manual con todos los detalles de operación y funcionamiento se encuentra con el nombre de UART_v2_50.pdf) tiene las siguientes características:

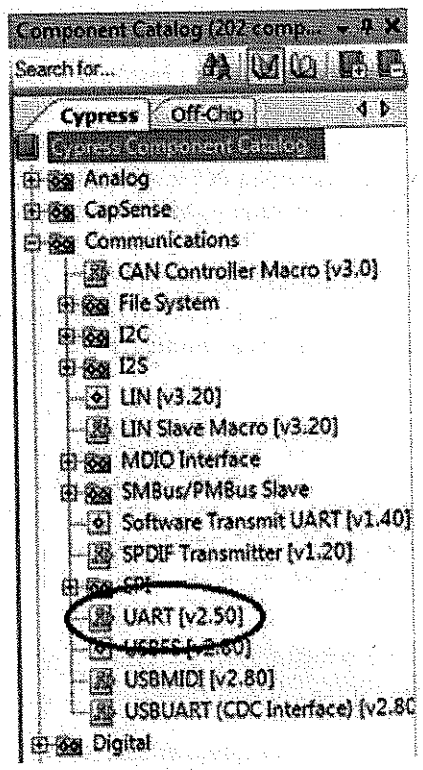
- Velocidades de comunicación (baud rates) desde 110 hasta 921600 bps o velocidades arbitrarias hasta 4Mbps.
- Fuente de reloj seleccionable por el usuario.
- Buffers de Tx y Rx independientes desde 4 hasta 65535 bytes cada uno.
- Detección de errores en la trama, paridad o por sobreescritura.
- Comunicaciones Full Duplex, Half Duplex, solo TX o solo Rx.
- Control de flujo por hardware.
- Siete fuentes de interrupción para Rx y 4 para Tx.
- Generación y detección de señal Break.
- Opciones de operación con estándar RS485.
- Sobre muestreo x8 o x16.
- Direccionamiento por hardware de 9 bits .

Para seleccionar en el PSoC Creator el bloque UART denominado UART [v2.50], el usuario debe buscar la carpeta Communications del catálogo de componentes y seleccionarlo entre las opciones disponibles allí (ver figura 149).

Señales de entrada/salida del bloque UART

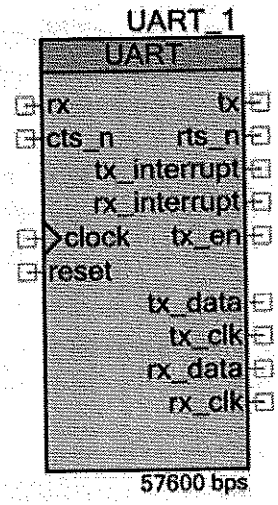
La imagen de la figura 150 muestra el bloque UART con todas las posibles señales de entrada/salida soportadas, algunas de ellas pueden ser eliminadas en la configuración si no son necesarias para la aplicación desarrollada.

Figura 149. Selección del bloque UART



Fuente: elaboración propia.

Figura 150. Señales de entrada/salida del UART



Fuente: elaboración propia.

- rx (input):

  Señal de entrada que se encarga de recibir los datos generados por otro dispositivo conectado al bus serial. Esta entrada puede ser eliminada.

- cts_n (input):

  Señal de entrada que indica cuando otro dispositivo conectado al bus serial está listo para recibir datos. Esta entrada puede ser eliminada.

- clock (input):

  Entrada que recibe una señal de reloj que define la velocidad de datos (baud rate) de la comunicación serial. Esta entrada puede ser eliminada.

- reset (input):

  Esta señal reinicia la máquina de estados interna de la UART pasando la salida al estado idle; la señal es de tipo sincrónico por lo que se requiere un flanco de subida para que tenga efecto. La entrada de reset se puede dejar flotando sin conexión externa. Esta entrada no puede ser eliminada.

- tx (output):

  Señal de salida que se encarga de enviar datos a otro dispositivo conectado al bus serial. Esta salida puede ser eliminada.

- rts_n (output):

  Señal de salida que indica cuando la UART del PSoC® está lista para recibir datos de otro dispositivo conectado al bus serial. Esta salida puede ser eliminada.

- tx_interrupt (output):

  Salida que agrupa por medio de una OR todas las fuentes de interrupción asociadas al transmisor de la UART. Esta salida puede ser eliminada.

- rx_interrupt (output):

  Salida que agrupa por medio de una OR todas las fuentes de interrupción asociadas al receptor de la UART. Esta salida puede ser eliminada.

- tx_en (output):

  Señal de salida utilizada en comunicación RS485, se encarga de habilitar la trasmisión de datos en este estándar. Esta salida puede ser eliminada.

- tx_data (output):

  Salida de datos utilizada para desplazar la información de tx para un componente CRC externo u otra lógica. Esta salida puede ser eliminada.

- tx_clk (output):

Salida de reloj utilizada para sincronizar el desplazamiento de la información de tx para un componente CRC externo u otra lógica. Esta salida puede ser eliminada.

- rx_data (output):

  Salida de datos utilizada para desplazar la información de rx para un componente CRC externo u otra lógica. Esta salida puede ser eliminada.

- tx_clk (output):

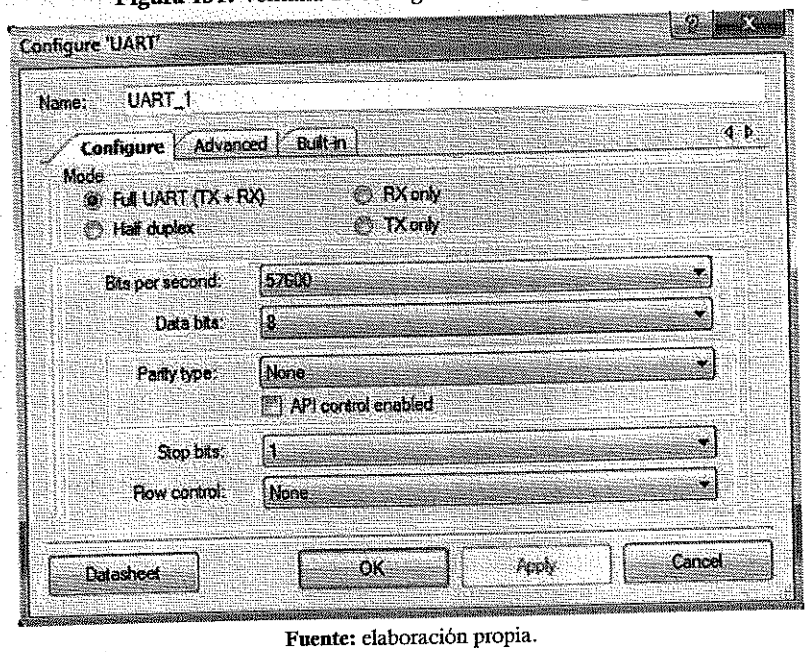
  Salida de reloj utilizada para sincronizar el desplazamiento de la información de rx para un componente CRC externo u otra lógica. Esta salida puede ser eliminada

Configuración del bloque UART

Para la ventana de configuración del bloque UART se pueden modificar o configurar los parámetros en las pestañas Configure y Advanced. Para la pestaña Configure los parámetros por configurar son: nombre (Name), modo de operación (Mode), velocidad de comunicaciones (Bits per second), cantidad de bits de datos (Data bits), tipo de paridad (Parity type), tamaño del bit de parada (Stop bits) y control de flujo (Flow control).

En la figura 151 puede verse la ventana de configuración del bloque UART con las funcionalidades antes nombradas para la pestaña Configure.

Figura 151. Ventana de configuración del bloque UART



Fuente: elaboración propia.

El parámetro Mode permite seleccionar entre cuatro valores posibles el tipo de comunicación que va a realizar el bloque UART:

- *Full UART (TX+RX):* comunicación serial bidireccional que permite trasmitir y recibir datos de manera simultánea (valor por defecto).
- *Half Duplex:* comunicación serial bidireccional que solo permite trasmitir o recibir datos.
- *RX Only:* solo se pueden recibir datos.
- *TX Only:* solo se pueden transmitir datos.

Para definir la velocidad de comunicaciones se hace uso del parámetro Bits per second que ofrece una variedad de valores entre 110 y 921600 bps; el valor por defecto está configurado en 57600bps.

Con el parámetro Data bits es posible seleccionar la longitud de los datos utilizados en la comunicación serial; este parámetro puede tener un valor entre 5 y 9, el valor por defecto es 8.

El tipo de paridad, que es opcional, también puede ser configurado; para esto se hace uso del parámetro Parity Type, que permite seleccionar entre cuatro opciones posibles, None (valor por defecto), Odd, Even o Mark/Space.

El tiempo de vida del bit de parada puede ser configurado con el parámetro Stop bits entre los valores: 1 (valor por defecto) y 2. Finalmente, para esta pestaña con el parámetro Flow control se puede habilitar o deshabilitar el control de flujo por hardware en la comunicación serial

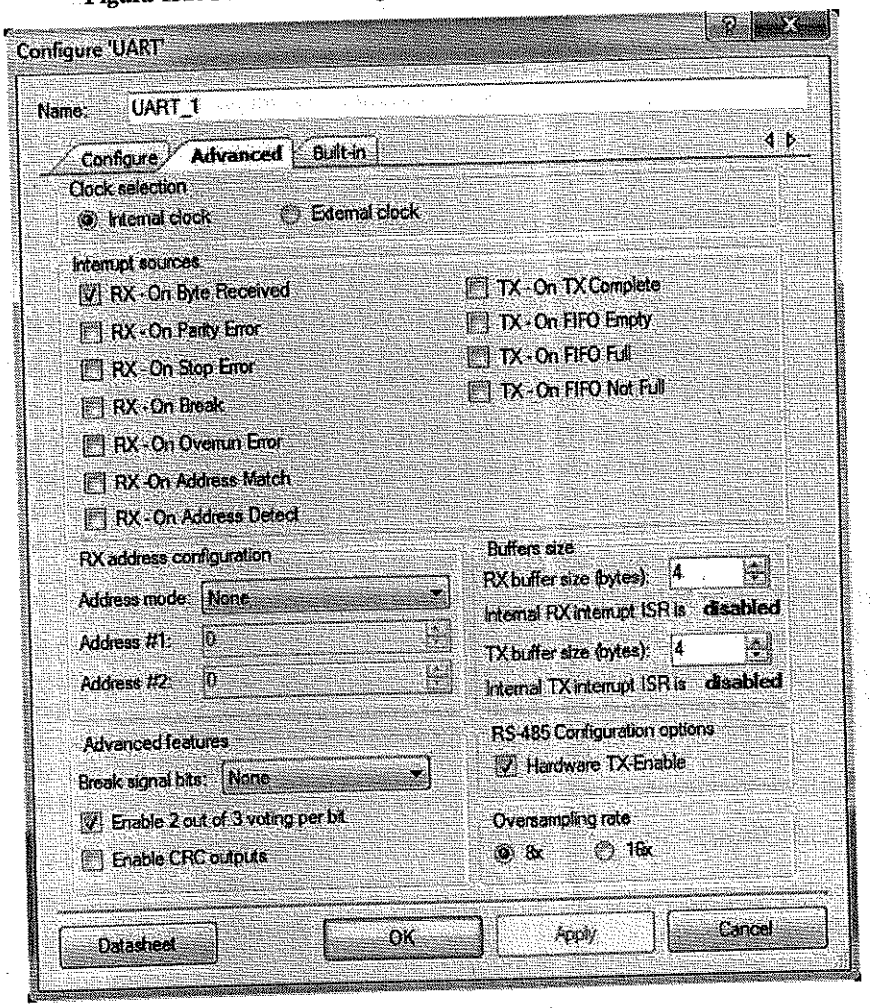
La otra pestaña de configuración denominada Advanced se ve en la figura 152; allí pueden configurarse los siguientes parámetros: selección de la fuente de reloj del bloque UART (Clock selection), selección de las posibles opciones de interrupción (Interrupt sources), configuración de la dirección del receptor (RX address configuration), tamaño de los buffers de Tx y Rx (Buffers size), comunicaciones RS485 (RS-485 Configuration options) y sobremuestreo (Oversampling rate).

El parámetro Clock selection permite escoger la fuente del reloj utilizada para generar la velocidad de comunicaciones del bloque UART; los valores posibles para este parámetro son: Internal clock que permite generar velocidades de comunicación estándar (valor por defecto) y External clock, con el que puede configurar cualquier valor de velocidad.

Las interrupciones son habilitadas haciendo uso del parámetro Interrupt sources entre 11 posibles fuentes:

- RX - On Byte Received (habilitado por defecto).
- RX - On Parity Error.
- RX - On Stop Error.

Figura 152. Pestaña de configuración Advanced del bloque UART



Fuente: elaboración propia.

- RX - On Break.
- RX - On Overrun Error.
- RX - On Address Match.
- RX - On Address Detect.
- TX - On TX Complete.
- TX - On FIFO Empty.
- TX - On FIFO Full.
- TX - On FIFO Not Full.

El parámetro Address mode define cómo el hardware y el software interactúan para manejar el direccionamiento y los datos de dispositivos seriales:

- Software Byte by Byte.
- Software Detect to Buffer.
- Hardware Byte by Byte.
- Hardware Detect to Buffer.
- None (valor por defecto).

Para configurar el tamaño de los buffers de Tx y Rx se hace uso del parámetro Buffers size; este valor se puede configurar de manera individual tanto para Tx como para Rx, en el rango de 4 (valor por defecto) a 65535.

Si el bloque UART se requiere para operar con distintivos que manejen la norma RS485 se dispone del parámetro RS-485 Configuration options que permite habilitar (valor por defecto) o deshabilitar el valor Hardware TX-Enable, utilizado para habilitar o deshabilitar los dispositivos de hardware empleados en la norma mencionada.

Rutinas utilizadas para el manejo del bloque UART

Las funciones API para el bloque UART permiten configurar, controlar y poner a andar sus diversos modos de operación. Para la sintaxis debe tenerse en cuenta que la primera parte del nombre de la función (UART_) debe ser reemplazada por el nombre que el usuario le haya asignado al bloque.

Las funciones que pueden ser utilizadas para configurar y manejar el bloque son:

- UART_Start():

  Inicializa variables asociadas al bloque UART e inicializa y habilita todos los parámetros de configuración seleccionados previamente. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- UART_ReadRxData():

  Lee el siguiente byte de datos recibidos; esta tarea la realiza sin comprobar el estado del receptor. No utiliza ningún parámetro de entrada y retorna un entero de 8 bits sin signo con el valor leído del registro de datos de Rx.

- UART_GetChar():

  Lee el último valor recibido. No utiliza ningún parámetro de entrada y retorna un entero de 8 bits sin signo con el valor recibido en el rango de 1 a 255. Si el valor recibido es 0 indica error o dato no disponible. Por ejemplo, si se quiere esperar hasta tener un dato valido recibido por el bloque UART, se podría utilizar la siguiente línea de código:

```
while ((dato=UART_GetChar())==0);
```

- UART_GetRxBufferSize():

  Lee el número de bytes recibidos disponibles en el buffer de Rx. No utiliza ningún parámetro de entrada y retorna un entero de 8 o 16 bits sin signo con la cantidad de bytes recibidos, el tamaño en bits del valor leído depende del tamaño configurado al buffer.

- UART_ClearRxBuffer():

  Borra el contenido del buffer de memoria de Rx. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- UART_WriteTxData():

  Envía un dato al transmisor del bloque UART sin realizar ningún tipo de verificación. Como parámetro de entrada utiliza un entero sin signo de 8 bits con el valor del byte por transmitir. No retorna ningún valor.

- UART_PutChar():

  Escribe un carácter en el buffer de Tx para ser enviado cuando el bus este. La función utiliza como parámetro de entrada un entero sin signo de 8 bits con el valor del byte por transmitir. No retorna ningún valor.

- UART_PutString():

  Escribe una cadena de caracteres en el buffer de Tx. La función utiliza como parámetro de entrada un arreglo de datos en formato ASCII de 8 bits. La función no retorna ningún valor. Por ejemplo para escribir en el bloque UART, con el mismo nombre un mensaje codificado en ASCII se puede escribir lo siguiente:

  ```
  UART_PutString("Mensaje de Ejemplo \r\n");
  ```

- UART_PutCRLF():

  Envía un byte junto con un retorno de carro (CR) y un retorno de línea (LF) al buffer del Tx. La función utiliza como parámetro de entrada un entero sin signo de 8 bits con el valor de byte por transmitir. No retorna ningún valor.

- UART_ClearTxBuffer():

  Borra el contenido del buffer de memoria de Tx. No utiliza ningún parámetro de entrada ni retorna ningún valor.

Ejemplos De Uso Del Bloque UART

Operación básica del bloque UART

El ejemplo que se presentará a continuación pretende mostrar la operación básica del bloque UART del PSoC5®LP. Para esto se enviará hacia un terminal de datos en Windows (por ejemplo HyperTerminal) un mensaje predefinido codificado en ASCII, luego se quedará en un ciclo de espera de datos enviados desde el terminal

de datos. Estos datos serán visualizados en una LCD alfanumérica; debe tenerse en cuenta que cada vez que se oprima la tecla Enter del teclado del PC en la LCD se debe cambiar la línea en la que se está escribiendo la información.

Solución

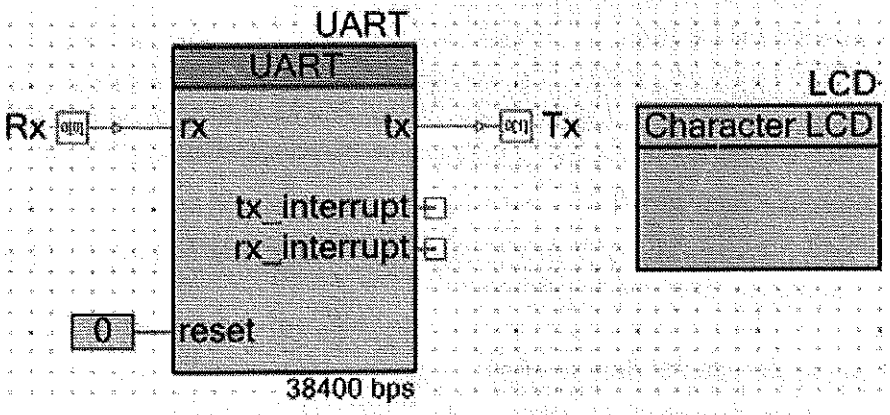
El bloque UART será utilizado en modo Full Duplex, lo que permite enviar y recibir datos simultáneamente. Las interrupciones no serán utilizadas dado que la única tarea en el ejemplo es la comunicación serial, la velocidad de comunicaciones no es un parámetro impuesto en el ejemplo así que será modificada a 38400 bps.

Inicialmente se hace uso del bloque UART [v2.50], al que se le asigna el nombre UART. En la pestaña Configure se modifica solamente el parámetro Bits per second al valor 38400 (velocidad no indicada dentro de los requisitos del ejemplo). En la pestaña Advanced se modifican dos parámetros deshabilitándolos, ya que esas funcionalidades no son necesarias para el ejemplo: Interrupt sources la opción RX-On Byte Received deshabilitada, RS-485 Configuration option con la opción Hardware TX-Enable deshabilitada.

De manera automática, cuando se selecciona el bloque UART este tiene asociadas dos líneas de E/S, las cuales traen la configuración adecuada de acuerdo con la tarea realizada, el único cambio realizado está relacionado con el nombre que cambió Rx para la línea de recepción de datos y Tx para la de trasmisión.

Se utiliza un bloque LCD para la visualización de los datos deshabilitando la opción Include ASCII to Number Conversion Routines. En la figura 153 se presenta el diagrama circuital con la solución propuesta para el ejemplo.

Figura 153. Diagrama circuital ejemplo 12-1



Fuente: elaboración propia.

Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: los bits P12[6:0] para la conexión de la LCD, el bit P0[0] para la conexión de la línea Rx y el bit P0[1] para la conexión de la línea Tx.

Nota:

Para realizar la conexión del PSoC® al PC se hace necesario el uso de un conversor USB a serial, que está incluido en la parte denominada KitProg de la tarjeta CY8C-KIT-059; para poder utilizarla es necesario separar el KitProg de la tarjeta donde se encuentra le PSoC5®LP. Ya que al diseñar la tarjeta de desarrollo mostrada en aparte "tarjeta de periféricos para el KIT CY8CKIT-059" se utilizó la línea P12[6] para la LCD y esta línea está cableada en la tarjeta CY8CKIT-059 para la Rx de la comunicación serial, lo que impediría su uso. Así, para el ejemplo se debe conectar el pin P0[0] de la tarjeta con el PSoC al pin P12[7] de la tarjeta KitProg y el pin P0[1] de la tarjeta con el PSoC al pin P12[6] de la tarjeta KitProg.

A continuación puede verse el código escrito para la solución del problema:

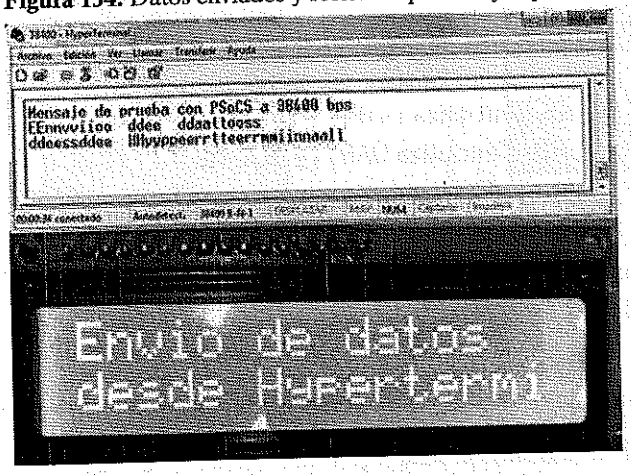
Ejemplo 12-1:

```
/* ==========================================================
 *
 * Programa que muestra la Tx y Rx de datos entre el PSoC y un PC
 * inicialmente la aplicación envía a una terminal de datos en
 * Windows un mensaje codificado en ASCII, luego espera datos
 * enviados desde la terminal para luego visualizarlos en una LCD
 * alfanumérica.
 *
 * ==========================================================
*/
#include <project.h>
/* Definición de variables */
uint8 dato,bandera;
int main()
{
  LCD_Start(); //Inicializa LCD
  UART_Start(); //Inicializa UART
  UART_PutString("Mensaje de prueba con PSoC5 a 38400 bps \r\n");
  LCD_Position(0,0);
  bandera=0; //Variable para determinar línea actual en LCD
  for(;;)
  {
    while ((dato=UART_GetChar())==0); //Espera hasta rx dato válido
    switch (dato)
```

```
    {
      case 0x0D: //Verifica si se recibió CR
        bandera=~bandera;
      break;
      case 0x0A: //Verifica si se recibió LF
        if (bandera!=0)
        {
          LCD_Position(1,0); //Ubica cursor en línea 1 de LCD
        }else{
          LCD_Position(0,0); //Ubica cursor en línea 0 de LCD
        }
      break;
      default:
        LCD_PutChar(dato); //Escribe en LCD dato recibido
        UART_PutChar(dato); //Reenvía al PC dato recibido
      break;
    }
  }
}
```

Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC®, y luego cablear cada uno de los componentes a la tarjeta de desarrollo; además, se debe hacer uso de un terminal de datos en Windows. En la figura 154 puede verse la información recibida y enviada en el terminal de datos de Windows y los datos recibidos y visualizados en la LCD controlada por el PSoC.

Figura 154. Datos enviados y recibidos para el ejemplo 12-1



Fuente: elaboración propia.

Voltímetro digital serial con visualización en PC

En el ejemplo que se desarrollará a continuación se implementa un voltímetro digital de cuatro dígitos BCD (un digito entero y tres dígitos decimales), transmitiendo los datos por UART a un terminal de datos en Windows. El voltaje por medir será una señal DC que podrá variar entre 0 y +5v (manejada por un potenciómetro conectado entre +5v y GND). Los datos se actualizarán cada 500ms.

Solución

Para la solución se utilizará un bloque ADC por aproximaciones sucesivas (SAR); para tener una visualización en milivoltios se utilizará la máxima resolución del ADC, es decir, 12 bits. Esto último no es un requerimiento del problema, así que cualquier resolución seleccionada funcionaría.

La solución en el PSoC Creator hace uso de un bloque ADC_SAR con el nombre ADC, el parámetro Resolution (bits) en 12, la opción Sample mode en Software trigger, que permite generar el inicio de conversión por software, el parámetro Input range con la opción Vssa to Vdda (Single Ended) seleccionada y finalmente el parámetro Reference con la opción Internal Vref seleccionada. Estos parámetros corresponden a los utilizados en el ejemplo 8-1, por lo que lo mostrado en la figura 121 aplica para este ejemplo.

Asociado a la entrada del ADC se utiliza un pin de E/S analógico que es la conexión del potenciómetro, con el nombre Potenciometro y la opción External terminal habilitada, donde se conectan un potenciómetro, la fuente de alimentación y la tierra del circuito, los demás parámetros se dejan con el valor por defecto.

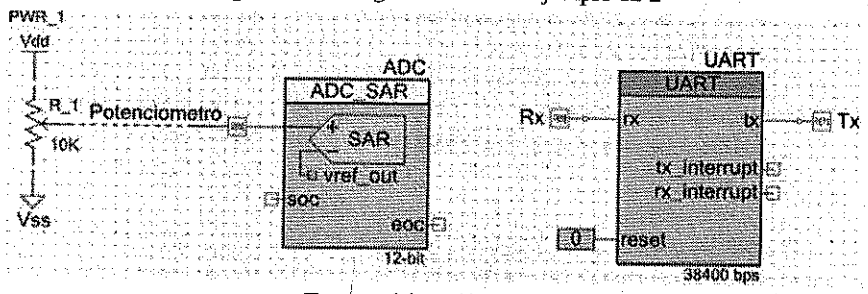
Por último se hace uso de un bloque UART [v2.50] al que se le asigna el nombre UART. En la pestaña Configure se modifica solamente el parámetro Bits per second al valor 38400 (velocidad no indicada dentro de los requisitos del ejemplo). En la pestaña Advanced se modifican dos parámetros deshabilitándolos, ya que esas funcionalidades no son necesarias para el ejemplo: Interrupt sources la opción RX-On Byte Received deshabilitada, RS-485 Configuration option con la opción Hardware TX-Enable deshabilitada.

Al igual que en el ejemplo 12-1, no debe olvidarse que de manera automática cuando se selecciona el bloque UART este tiene asociadas dos líneas de E/S, las cuales traen la configuración adecuada de acuerdo con la tarea realizada. El único cambio realizado está relacionado con el nombre que cambió Rx para la línea de recepción de datos y Tx para la de trasmisión.

En la figura 155 se presenta el diagrama circuital con la solución propuesta para el ejemplo.

Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: el bit P0[0] para la conexión de la línea Rx, el bit P0[1] para la conexión de la línea Tx, y el bit P2[0] para la conexión de la entrada analógica.

Figura 155. Diagrama circuital ejemplo 12-2



Fuente: elaboración propia.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 12-2:

```
/* ==========================================================
 *
 * Programa que visualiza en un PC por medio de una terminal de
 * datos el valor de voltaje medido de un potenciómetro conectado
 * a una de las entradas del PSoC, el valor mostrado tendrá un
 * digito entero y tres dígitos decimales en el rango de 0.000V
 * a 5.000V, el dato se actualiza en la terminal de datos del PC
 * cada 500ms.
 *
 * ==========================================================
 */
#include <project.h>
/* Definición de variables */
uint16 volt;
uint8 dato;
int main()
{
  UART_Start(); //Inicializa UART
  ADC_Start(); //Inicializa ADC SAR
  UART_PutString("\r\nVoltaje:\r\n");
  for(;;)
```

```
{
  ADC_StartConvert(); //Inicia conversión
  ADC_IsEndConversion(ADC_WAIT_FOR_RESULT); //Espera final de
                                            //conversión
  volt=ADC_GetResult16(); //Lee resultado de conversión
  volt=ADC_CountsTo_mVolts(volt); //Convierte dato leído a mV en Hx
  dato=volt/1000; //Convierte a decimal dato de voltaje realizando
  volt=volt%1000; //divisiones sucesivas por múltiplos de 10
  UART_PutChar(dato+0x30); //luego envía cada digito por UART al PC
  UART_PutChar('.');
  dato=volt/100;
  volt=volt%100;
  UART_PutChar(dato+0x30);
  dato=volt/10;
  volt=volt%10;
  UART_PutChar(dato+0x30);
  UART_PutChar((uint8)volt+0x30);
  UART_PutChar('v');
  CyDelay(500); //Espera 500 ms para una nueva conversión
  UART_PutChar(0x0D);
 }
}
```

Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC® y luego cablear cada uno de los componentes a la tarjeta de desarrollo; además, se debe hacer uso de un terminal de datos en Windows. Así mismo, para la correcta operación del ejemplo debe tenerse en cuenta la nota aclaratoria del ejemplo 12-1

Algunos módulos analógicos del PSoC5®LP

El sistema analógico programable crea combinaciones específicas de bloques estándar y avanzados para el procesamiento de señales analógicas. Estos bloques están interconectados entre sí y como todos los bloques del PSoC® pueden utilizar cualquier pin en el dispositivo, y proporcionar así un alto nivel de flexibilidad en el diseño. En este capítulo se hablará de tres de estos bloques analógicos: comparador analógico, amplificador operacional y amplificador de ganancia programable.

Bloque comparador analógico

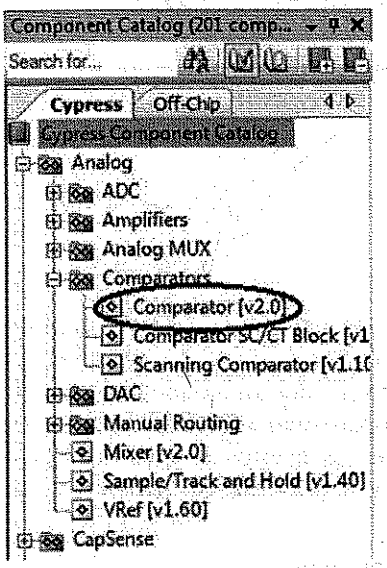
El PSoC5®5LP tiene cuatro módulos comparadores analógicos; las entradas positivas y negativas de los comparadores vienen a través de multiplexores que llevan información de las variables globales analógicas (AGS), del bus local analógico (ABUS), del bus analógico multiplexado (AMUXBUS) y de la referencia de precisión. La salida de cada comparador se enruta a través de un bloque de sincronización a una tabla de búsqueda de dos entradas (LUT) que permite realizar funciones lógicas simples; luego la salida de la LUT se encamina a la interfaz del sistema digital UDB. En la figura 156 puede verse el diagrama de bloques del comparador analógico.

El bloque comparador analógico (el manual con todos los detalles de operación y funcionamiento se encuentra con el nombre de Comp_v2_0.pdf) tiene las siguientes características:

- Selección de entradas flexible.
- Compensación de potencia por velocidad.
- Histéresis de entrada de 10mV opcional.
- Bajo voltaje de offset (<1mV).
- Polaridad de la salida configurable.
- Entrada positiva puede pasar opcionalmente por un filtro pasa bajos
- Filtro Glitch de salida.

Para seleccionar en el PSoC Creator el bloque comparador denominado Comparator [v2.0], el usuario debe buscar la carpeta Analog del catálogo de componentes, subcarpeta Comparators y seleccionarlo entre las opciones disponibles allí /ver figura 157).

Figura 157. Selección del bloque comparador

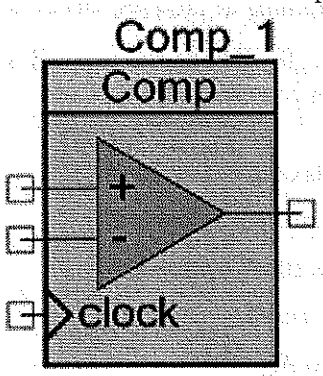


Fuente: elaboración propia.

Señales de entrada/salida del bloque Comparador

La imagen de la figura 158 muestra el bloque comparador analógico con todas las posibles señales de entrada/salida soportadas; algunas de ellas pueden ser eliminadas en la configuración si no son necesarias para la aplicación desarrollada.

Figura 158. Señales de entrada/salida del bloque comparador



Fuente: elaboración propia.

- + (analog input):

  Entrada positiva conectada normalmente a la señal de voltaje que se quiere comparar; puede conectarse a GPIO o a referencias internas de voltaje. Esta entrada no puede ser eliminada.

- - (analog input):

  Entrada negativa conectada normalmente a una referencia de voltaje; puede conectarse a GPIO o a referencias internas de voltaje. Esta entrada no puede ser eliminada.

- Salida del comparador (output):

  Señal digital de salida del comparador. Para configuración no inversora la salida está en alto cuando el voltaje de la entrada positiva es mayor al de la entrada negativa; si la configuración es inversora la salida está en alto cuando el voltaje de la entrada negativa es mayor al de la entrada positiva. Esta salida no puede ser eliminada.

- clock (input):

  La entrada de reloj sincroniza la salida del comparador con el flanco de subida de la señal de reloj ingresada, forzando la comparación. Esta entrada puede ser eliminada.

Configuración del bloque comparador

La ventana de configuración del bloque comparador permite no solo modificar parámetros de configuración, sino también ver el resultado de la configuración del bloque. Se pueden modificar o configurar los siguientes parámetros:

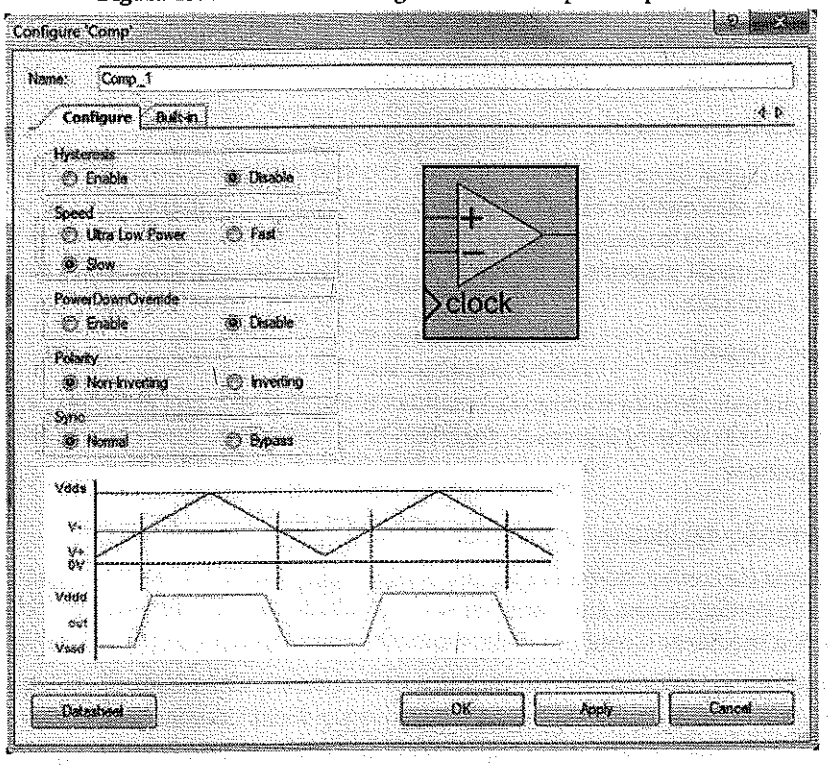
- Nombre (Name).
- Habilitar o deshabilitar histéresis (Hysteresis).
- Velocidad de respuesta (Speed).
- Bajo consumo de energía (PowerDownOverride).
- Polaridad de la salida (Polarity).
- Entrada de sincronismo (Sync).

En la figura 159 puede verse la ventana de configuración del bloque comparador analógico con las funcionalidades antes nombradas.

El parámetro Hysteresis adiciona aproximadamente 10mV de histéresis al comparador, esto ayuda a asegurar que los voltajes que se mueven lentamente o señales ligeramente ruidosas no generen oscilaciones en la salida cuando los voltajes en las entradas del comparador sean casi iguales. Los valores posibles para este parámetro son Enable o Disable (valor por defecto).

Con el parámetro Speed se puede optimizar el consumo de energía del comparador al controlar la velocidad de respuesta de este; se pueden seleccionar tres opciones para este parámetro: Ultra Low Power utilizado en aplicaciones que requieran un muy bajo consumo de energía; Slow utilizado en aplicaciones que necesiten un tiempo de respuesta mayor a 80ns (valor por defecto), y Fast utilizado en aplicaciones que demanden un tiempo de respuesta menor a 80ns.

Figura 159. Ventana de configuración del bloque comparador



Fuente: elaboración propia.

Para aplicaciones en las cuales se deba disminuir el consumo de energía durmiendo periféricos, el comparador tiene un parámetro que permite este evento: PowerDownOverride que admite seleccionar entre dos valores posibles Enable o Disable (valor por defecto).

Se puede configurar la lógica de la salida del comparador con el parámetro Polarity entre dos valores posibles: Inverting, la salida opera con lógica positiva y Non-Inverting, la salida opera con lógica negativa (valor por defecto).

El parámetro Sync permite seleccionar entre sincronizar o no la salida del comparador. Se puede seleccionar entre dos opciones posibles: Normal se genera respuesta sincronizada con el flanco de subida de la señal de reloj que ingresa por la entrada clock (valor por defecto) y Bypass se genera respuesta inmediata.

Rutinas utilizadas para el manejo del bloque comparador

Las funciones API para el bloque comparador permiten configurar, controlar y poner a andar sus diversos modos de operación. Para la sintaxis debe tenerse en cuenta que la primera parte del nombre de la función (Comp_) debe ser reemplazado por el nombre que el usuario le haya asignado al bloque.

Las funciones que pueden ser utilizadas para configurar y manejar el bloque son:

- Comp_Start():

  Inicializa variables asociadas al bloque comparador e inicializa y habilita todos los parámetros de configuración seleccionados previamente. No utiliza ningún parámetro de entrada ni retorna ningún valor.

- Comp_SetSpeed():

  Asigna la velocidad de respuesta de salida del comparador. Como parámetro de entrada utiliza un entero sin signo de 8 bits representado con las constantes: Comp_LOWPOWER, Comp_SLOWPOWER y Comp_HIGHSPEED. La función no retorna ningún valor.

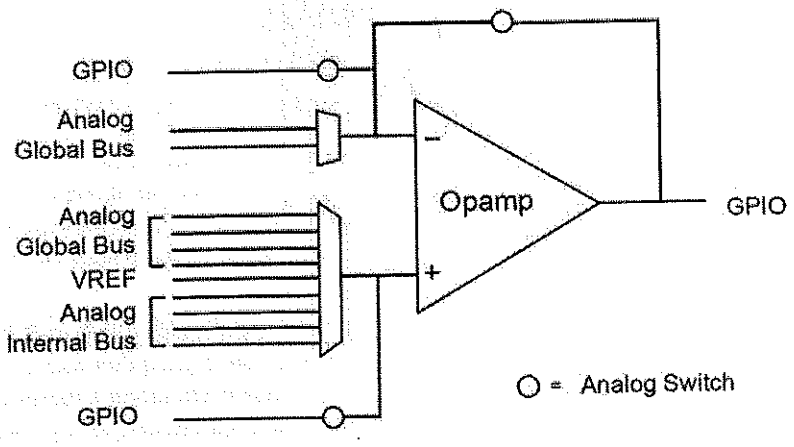
- Comp_GetCompare():

  Devuelve un valor diferente de cero cuando el voltaje conectado a la entrada positiva del comparador es mayor que el voltaje de la entrada negativa. Este valor no se ve afectado por el parámetro de polaridad. No utiliza ningún parámetro de entrada. Retorna un entero de 8 bits sin signo que representa el estado de la comparación.

Bloque amplificador operacional

El PSoC5®5LP tiene cuatro amplificadores operacionales, que son configurables como seguidores de voltaje (amplificadores con ganancia uno) para manejar cargas que requieran corrientes altas o como amplificadores operacionales convencionales. Por ejemplo, la salida de un DAC puede estar conectada a un amplificador operacional para aumentar la corriente del DAC. En la figura 160 puede verse el diagrama de bloques del amplificador operacional.

Figura 160. Diagrama de bloques del amplificador operacional del PSoC5®5LP



Fuente: [5, p. 56].

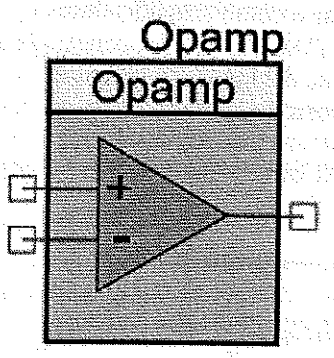
El bloque amplificador operacional (el manual con todos los detalles de operación y funcionamiento se encuentra con el nombre de OpAmp_*v1_90.pdf*) tiene las siguientes características:

- Configuración seguidor de voltaje o amplificador.
- Ancho de banda con ganancia unitaria > 3MHz.
- Bajo voltaje de offset (≤ 2mV).
- Conexión de baja resistencia en conexión a pin.
- Corriente de salida de 25mA.
- Slew rate de 3v/us.
- Potencia y ancho de banda programables.
- Conexión interna en modo seguidor (ahorra pin).
- GPIO dedicadas para entradas y salidas.

Señales de entrada/salida del bloque amplificador operacional

La figura 161 muestra el bloque amplificador operacional con todas las posibles señales de entrada/salida soportadas; algunas de ellas pueden ser eliminadas en la configuración si no son necesarias para la aplicación desarrollada.

Figura 161. Señales de entrada/salida del bloque amplificador operacional



Fuente: elaboración propia.

- + (analog input):

  Entrada no inversora típica de cualquier amplificador operacional. Esta entrada no puede ser eliminada.

- - (analog input):

  Entrada inversora típica de cualquier amplificador operacional, cuando el amplificador se configura como seguidor se conecta internamente con la salida.

Esta entrada no puede ser eliminada.

- Salida del amplificador (analog output):

  Señal de salida del amplificador operacional conectada directamente a un pin analógico externo. Esta salida no puede ser eliminada.

Cada uno de los amplificadores operaciones del PSoC® tiene asociados GPIO específicos para su conexión; por esta razón, para este bloque no existe la libertad de selección existente en otros bloques. En la tabla 26 puede verse la distribución de pines para cada uno de los amplificadores.

	Entrada +	Entrada -	Salida
opamp_0	P0[2]	P0[3]	P0[1]
opamp_1	P3[5]	P3[4]	P3[6]
opamp_2	P0[4]	P0[5]	P0[0]
opamp_3	P3[3]	P3[2]	P3[7]

Tabla 26. Pines utilizados por los cuatro amplificadores operacionales del PSoC5®LP
Fuente: elaboración propia.

Configuración del bloque amplificador operacional

La ventana de configuración del bloque amplificador operacional permite modificar o configurar los siguientes parámetros: nombre (Name), modo de operación (Mode) y potencia de salida (Power).

Con el parámetro Mode se pueden seleccionar entre los dos modos de operación soportados por el bloque: OpAmp; en este modo los tres pines típicos de un amplificador operacional están presentes (valor por defecto) o Follower en este modo el bloque se comporta como un seguidor de voltaje.

El parámetro Power permite seleccionar entre cuatro niveles de potencia para la salida del amplificador operacional: en los modos High Power y Med Power se habilita un amplificador de salida clase AB, lo que permite altas corrientes de salida en el amplificador operacional, el modo Low Power habilita un amplificador clase A que limita la corriente de salida, en el modo Low Power Over Compensated se habilita también un amplificador de salida clase A, pero incluye un circuito de compensación con una alta capacitancia de entrada.

Rutinas utilizadas para el manejo del bloque amplificador operacional

Las funciones API para el bloque amplificador operacional permiten configurar, controlar y poner a andar sus diversos modos de operación. Para la sintaxis debe tenerse en cuenta que la primera parte del nombre de la función (Opamp_) debe ser reemplazado por el nombre que el usuario le haya asignado al bloque.

Las funciones que pueden ser utilizadas para configurar y manejar el bloque son:

- Opamp_Start():

  Habilita el amplificador operacional y asigna el valor de potencia configurado previamente. No utiliza ningún parámetro de entrada ni retorna ningún valor.

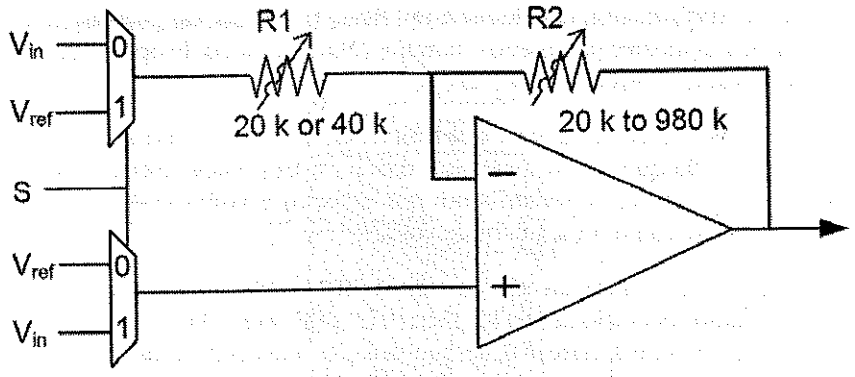
- Opamp_SetPower():

  Asigna el valor de potencia de salida del amplificador operacional. Como parámetro de entrada utiliza un entero sin signo de 8 bits representado con alguna de las siguientes constantes: Opamp_LPOCPOWER, Opamp_LOWPOWER, Opamp_MEDPOWER y Opamp_HIGHPOWER. La función no retorna ningún valor.

Bloque amplificador de ganancia programable (PGA)

El amplificador de ganancia programable (PGA) del PSoC5®5LP es un amplificador operacional continuo en el tiempo con entradas configurables para la resistencia de entrada y retroalimentación; puede ser implementado como amplificador con ganancia positiva o ganancia negativa o como parte de amplificador diferencial. En la figura 162 puede verse el diagrama de bloques del amplificador de ganancia programable.

Figura 162. Diagrama de bloques del PGA



Fuente: [5, p. 57].

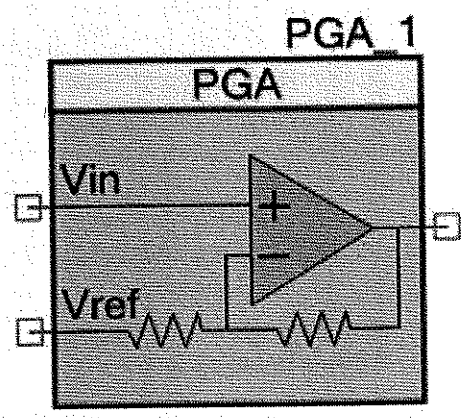
El bloque PGA (el manual con todos los detalles de operación y funcionamiento se encuentra con el nombre de PGA_v2_0.pdf) tiene las siguientes características:

- Ganancia seleccionable de 1 a 50.
- Alta impedancia de entrada.
- Referencia de entrada seleccionable.
- Características de potencia ajustables.

Señales de entrada/salida del bloque PGA

La figura 163 muestra el bloque PGA con todas las posibles señales de entrada/salida soportadas, algunas de ellas pueden ser eliminadas en la configuración si no son necesarias para la aplicación desarrollada.

Figura 163. Señales de entrada/salida del bloque PGA



Fuente: elaboración propia.

- Vin (analog input):

  Entrada conectada a la señal de voltaje que se quiere amplificar. Esta entrada no puede ser eliminada.

- Vref (analog input):

  Entrada conectada a una referencia de voltaje, que puede conectarse a una referencia interna (VSS) o externa. Esta entrada puede ser eliminada..

- Salida del PGA (analog output):

  Señal de salida del amplificador (Vout). El voltaje de salida es una función de Vin y Vref (ver ecuación 7). Esta salida no puede ser eliminada.

  $Vout=Vref+(Vin-Vref)*Gain \quad \rightarrow (7)$

Configuración del bloque PGA

La ventana de configuración del bloque PGA permite no solo modificar parámetros de configuración, sino también ver el resultado de la configuración del bloque. Se pueden modificar o configurar los siguientes parámetros: nombre (Name), ganancia del amplificador (Gain), potencia de salida (Power) y voltaje de referencia de entrada (Vref_Input).

Con el parámetro Gain se puede seleccionar el valor de la ganancia del amplificador entre nueve valores posibles (ver tabla 27).

Ganancia	Rb	Ra
1	0	40K
2	40K	40K
4	120K	40K
8	280K	40K
16	600K	40K
24	460K	20K
32	620K	20K
48	470K	10K
50	490K	10K

Tabla 27. Valores del parámetro Gain del bloque PGA
Fuente: elaboración propia.

El parámetro Power permite seleccionar entre cuatro niveles de potencia para la salida del PGA: Minimum Power, Low Power (valor por defecto), Medium Power y High Power. El valor mínimo genera una respuesta lenta en el amplificador y bajo consumo de energía, el valor máximo genera una respuesta rápida y además un alto consumo de energía en el amplificador. Finalmente, con el parámetro Vref_Input se puede seleccionar el voltaje de referencia de entrada para el amplificador entre dos opciones posibles:

- Internal Vss: la señal de tierra del PSoC es utilizada como entrada de referencia al amplificador eliminando una entrada.
- External: se activa la entrada Vef que permite ingresar una señal de referencia externa al amplificador (valor por defecto).

En la figura 164 puede verse la ventana de configuración del bloque PGA con las funcionalidades antes nombradas.

Rutinas utilizadas para el manejo del bloque PGA

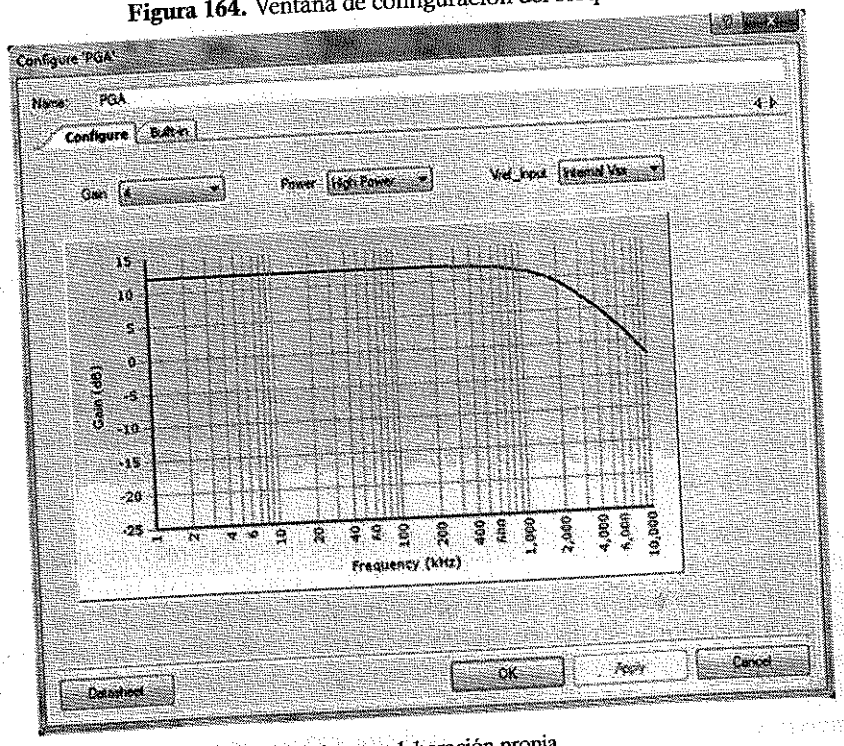
Las funciones API para el bloque amplificador de ganancia programable (PGA) permiten configurar, controlar y poner a andar sus diversos modos de operación. Para la sintaxis debe tenerse en cuenta que la primera parte del nombre de la función (PGA_) debe ser reemplazada por el nombre que el usuario le haya asignado al bloque.

Las funciones que pueden ser utilizadas para configurar y manejar el bloque son:

- PGA_Start():

  Habilita el PGA y asigna el valor de potencia configurado previamente. No utiliza ningún parámetro de entrada ni retorna ningún valor.

Figura 164. Ventana de configuración del bloque PGA



Fuente: elaboración propia.

- PGA_SetPower():

  Asigna el valor de potencia de salida del PGA. Como parámetro de entrada utiliza un entero sin signo de 8 bits representado con alguna de las siguientes constantes: PGA_MINPOWER, PGA_LOWPOWER, PGA_MEDPOWER y PGA_HIGHPOWER. La función no retorna ningún valor.

- PGA_SetGain():

  Asigna el valor de ganancia del PGA. Como parámetro de entrada utiliza un entero sin signo de 8 bits representado con alguna de las siguientes constantes: PGA_GAIN_01, PGA_GAIN_02, PGA_GAIN_04, PGA_GAIN_08, PGA_GAIN_16, PGA_GAIN_024, PGA_GAIN_32, PGA_GAIN_48 y PGA_GAIN_50. La función no retorna ningún valor.

Ejemplos de uso de los bloques analógicos

Operación básica del bloque comparador analógico

El ejemplo que se presentará a continuación muestra la operación básica del bloque comparador analógico del PSoC5®LP. Para esto se pide diseñar un control On/Off ambiental de temperatura haciendo uso de un sensor de temperatura comercial el

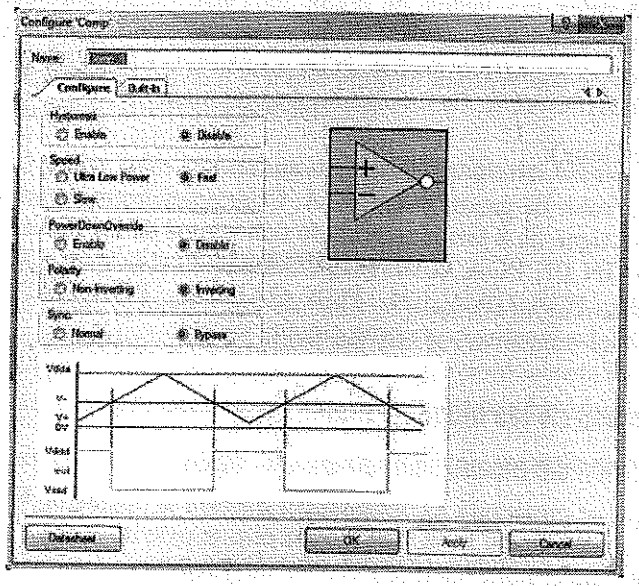
LM35, la idea es mantener en un rango de temperatura una zona determinada. Para esto se utilizará un calefactor que estará encendido mientras la temperatura sea menor a 25°C y un ventilador que se encenderá cuando la temperatura supere los 35°C. El sistema diseñado solo se encargará de la parte de control; la parte de potencia para el manejo del calefactor y del ventilador no serán contempladas en el ejemplo.

Solución

La solución planteada hará uso de dos bloques comparadores analógicos, cada uno se encargará de controlar, de manera individual, la activación o desactivación del calefactor o del ventilador. Para lograr los rangos de temperatura deseados cada comparador requiere voltajes de referencia específicos; para el caso del calefactor se necesita una referencia de voltaje de aproximadamente 250mV (25°C), y para el caso del ventilador, una referencia de aproximadamente 350mV (35°C). Estos valores se determinan recordando el rango de operación del sensor de temperatura (0°C a 100°C) que de manera lineal genera una relación de voltaje de 10mV/°C, es decir, opera en el rango de 0v a 1v.

Inicialmente se hace uso de un primer bloque comparador al que se le asigna el nombre Comp_1, encargado de activar o desactivar el calefactor, en la ventana de configuración se seleccionan los siguientes parámetros: Hysteresis deshabilitada, Speed con la opción Fast seleccionada, PowerDownOverride deshabilitada, Polarity con la opción Inverting seleccionada. Con esta última opción se logra que el comparador mantenga activado el calefactor para temperaturas menores a 25°C y el parámetro Sync seleccionada la opción Bypass. En la figura 165 puede verse el resumen de la configuración para este bloque.

Figura 165. Venta de configuración para el bloque Comp_1 del ejemplo 13-1



Fuente: elaboración propia.

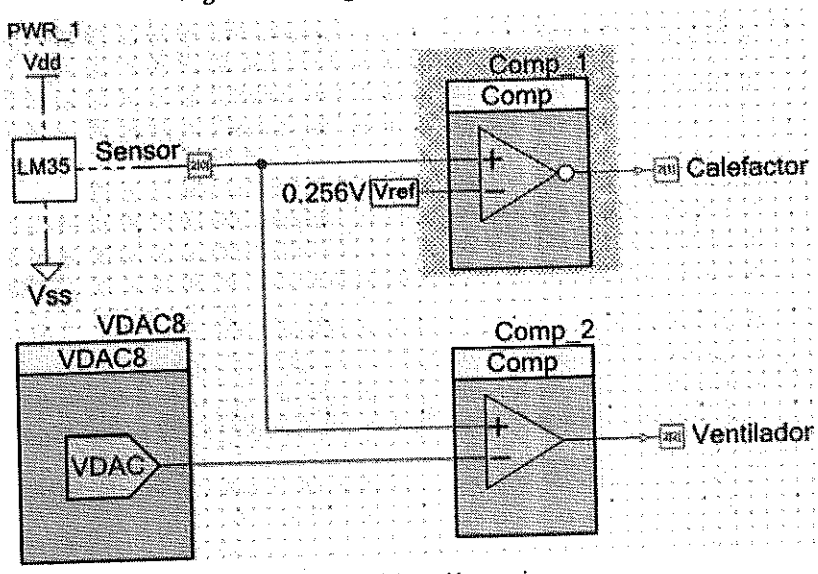
Se utiliza un segundo bloque comparador denominado Comp_2, encargado de activar o desactivar el ventilador. En la ventana de configuración se seleccionan los siguientes parámetros: Hysteresis deshabilitada, Speed la opción Fast seleccionada, PowerDownOverride deshabilitada, Polarity con la opción Non-Inverting seleccionada; así se logra que el comparador mantenga activado el ventilador para temperaturas mayores a 35°C y el parámetro Sync con la opción Bypass seleccionada.

Para las referencias de los comparadores se utiliza un bloque Vref con el valor 0,256V seleccionado (el más cercano a los 0,25v que se requieren), y un bloque DAC de voltaje VDAC8, configurado en el rango de 0-1,020v y el valor inicial en 348mV (que se mantendrá constante todo el tiempo), el más cercano a los 0,35v requeridos.

Se utiliza un pin de E/S analógico que se conecta a la entrada no inversora de los comparadores y es la conexión del sensor de temperatura, con el nombre Sensor y la opción External terminal habilitada, en la cual se conectan el sensor de temperatura, la fuente de alimentación y la tierra, los demás parámetros se dejan con el valor por defecto.

Finalmente, para las señales de control del calefactor y del ventilador se utilizan dos bloques digitales de E/S de un bit configurados como salidas, con los nombres calefactor y ventilador. Para los dos la configuración es igual, la opción HW connection deshabilitada, la opción External terminal deshabilitada. En la figura 166 se presenta el diagrama circuital con la solución propuesta para el ejemplo.

Figura 166. Diagrama circuital ejemplo 13-1



Fuente: elaboración propia.

Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: el bit P2[1] para la señal de control del calefactor (salida del bloque Comp_1), el bit P2[2] para

la señal de control del ventilador (salida del bloque Comp_2) y el bit P2[0] para la conexión del sensor de temperatura a las entradas analógicas no inversoras de los comparadores.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 13-1:

```
/* ============================================================
 *
 * Programa que controla el encendido o apagado de un calefactor
 * y un ventilador dependiendo del valor generado por un sensor de
 * temperatura de referencia LM35 conectado a una de las entradas
 * del PSoC, el calefactor se activará si la temperatura medida es
 * menor a 25°C, el ventilador se activará si el valor de la
 * temperatura es mayor a 35°C, en cualquier otro caso estarán
 * apagados.
 *
 * ============================================================
*/
#include <project.h>
int main()
{
  Comp_1_Start(); //Inicializa en Comparador 1
  Comp_2_Start(); //Inicializa en Comparador 2
  VDAC8_Start(); //Inicializa el VDAC
  for(;;)
  {
//No se requiere más código dado que los componentes
//funcionan de manera autónoma basados en su configuración
//inicial.
  }
}
```

Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC® y luego cablear cada uno de los componentes a la tarjeta de desarrollo.

Operación básica del bloque amplificador operacional T2

En el ejemplo que se presentará a continuación se muestra la operación básica del bloque Amplificador Operacional del PSoC5®LP. Para esto se diseñará un amplificador con ganancia 5 haciendo uso de un sensor de temperatura comercial el LM35, la idea es amplificar la señal generada por el sensor (10mV/°C) y enviar esto al exterior del PSoC, mostrando el resultado en un LED.

Solución

La solución planteada hará uso de un bloque amplificador operacional, al que se le calcularán las resistencias externas para la configuración no inversora típica. Dado que la operación del bloque es sencilla y similar a si se tuviese un amplificador operacional en un circuito integrado el proceso de diseño es muy simple.

Inicialmente se hace uso de un bloque Opamp [v1.90] al que se le asigna el nombre Opamp, encargado de amplificar la señal proveniente del sensor de temperatura, en la ventana de configuración se seleccionan los siguientes parámetros: Mode con la opción OpAmp seleccionada (así el bloque opera como amplificador operacional convencional) y Power con la opción High Power seleccionada.

Se utiliza un pin de E/S analógico que se conecta a la entrada no inversora del amplificador operacional que corresponde a la conexión del sensor de temperatura, con el nombre Vin y la opción External terminal habilitada, en la cual se conectan el sensor de temperatura, la fuente de alimentación y la tierra, los demás parámetros se dejan con el valor por defecto.

Para la entrada inversora se utiliza otro pin de E/S analógico donde se conectan las resistencias que determinan la ganancia del amplificador operacional, con el nombre Vin_menos y la opción External terminal habilitada, donde se conectan dos resistencias, los demás parámetros se dejan con el valor por defecto.

Finalmente, para la señal de salida se utiliza otro pin de E/S analógico con el nombre Vout, la opción HW connection habilitada, la opción External terminal habilitada; se conecta a este bloque una resistencia, un LED y la tierra del circuito. En la figura 167 se presenta el diagrama circuital con la solución propuesta para el ejemplo.

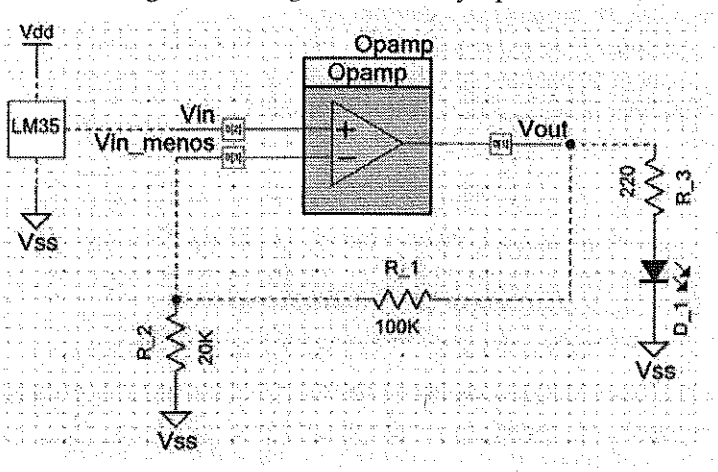
Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: el bit P0[1] para la señal de salida conectada al LED, el bit P0[2] para la conexión del sensor de temperatura a la entrada no inversora del amplificador y el bit P0[3] para la conexión de las resistencias del amplificador a la entrada inversora.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 13-2:

```
/* ==========================================================
 *
 * Programa que amplifica x 5 la señal analógica generada por un
```

Figura 167. Diagrama circuital ejemplo 13-2



Fuente: elaboración propia.

```
 * sensor de temperatura de referencia LM35 conectado a una de las
 * entradas del Opamp del PSoC, la salida del amplificador será
 * conectada a un LED.
 *
 * ============================================================
*/
#include <project.h>
int main()
{
  Opamp_Start(); //Inicializa en Amplificador Operacional
  for(;;)
  {
        //No se requiere más código dado que los componentes
        //funcionan de manera autónoma basados en su configuración
        //inicial.
  }
}
```

Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC® y luego cablear cada uno de los componentes a la tarjeta de desarrollo, el proceso puede ser corroborado con un multímetro conectado a la salida del sensor y a la salida del amplificador operacional.

Operación básica del bloque amplificador de ganancia programable

En el ejemplo que se presentará a continuación se muestra la operación básica del bloque amplificador de ganancia programable (PGA) del PSoC5®LP. Para esto se diseñará un amplificador con ganancia 4, haciendo uso de un sensor de temperatura comercial el LM35; la idea es amplificar la señal generada por el sensor (10mV/°C) y enviar esto al exterior del PSoC, mostrando el resultado en un LED.

Solución

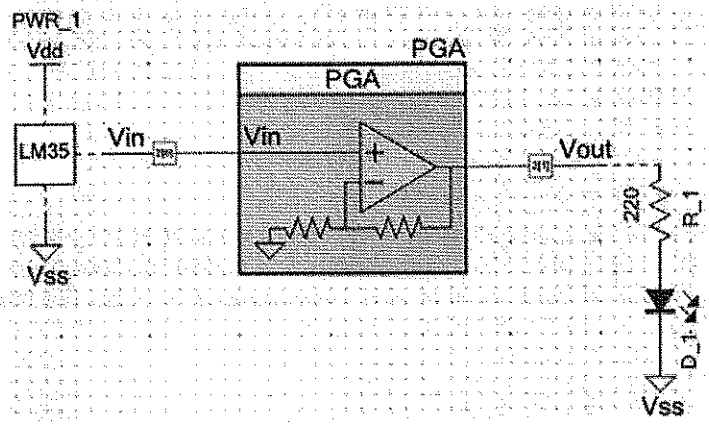
La solución planteada hará uso de un bloque PGA, al que se le seleccionará la ganancia requerida Dado que la operación del bloque es sencilla y similar a si se tuviese un amplificador operacional en un circuito integrado el proceso es similar al del ejemplo 13-2.

Inicialmente se hace uso de un bloque PGA [v2.0] al que se le asigna el nombre PGA, encargado de amplificar la señal proveniente del sensor de temperatura. En la ventana de configuración se seleccionan los siguientes parámetros: Gain con el valor 4 seleccionado, Power con la opción High Power seleccionada y Vref_Input con la opción Internal Vss seleccionada (así el bloque opera como amplificador convencional).

Se utiliza un pin de E/S analógico que se conecta a la entrada no inversora del PGA que corresponde a la conexión del sensor de temperatura, con el nombre Vin y la opción External terminal habilitada, donde se conectan el sensor de temperatura, la fuente de alimentación y la tierra; los demás parámetros se dejan con el valor por defecto.

Para la señal de salida se utiliza otro pin de E/S analógico con el nombre Vout, la opción HW connection habilitada, la opción External terminal habilitada. Se conecta a este bloque una resistencia, un LED y la tierra del circuito. En la figura 168 se presenta el diagrama circuital con la solución propuesta para el ejemplo.

Figura 168. Diagrama circuital ejemplo 13-3



Fuente: elaboración propia.

Los recursos del PSoC®5LP utilizados en el ejemplo son los siguientes: el bit P2[1] para la señal de salida conectada al LED, el bit P2[0] para la conexión del sensor de temperatura a la entrada no inversora del PGA.

A continuación puede verse el código escrito para la solución del problema:

Ejemplo 13-3:

```
/* ============================================================
 *
 * Programa que amplifica x 4 la señal analógica generada por un
 * sensor de temperatura de referencia LM35 conectado a una de las
 * entradas del PGA del PSoC, la salida del amplificador será
 * conectada a un LED.
 *
 * ============================================================
*/
#include <project.h>
int main()
{
  PGA_Start(); //Inicializa el PGA

  for(;;)
  {
      //No se requiere más código dado que los componentes
      //funcionan de manera autónoma basados en su configuración
      //inicial.
  }
}
```

Para realizar la verificación del diseño y del algoritmo se debe programar el PSoC® y luego cablear cada uno de los componentes a la tarjeta de desarrollo; el proceso puede ser corroborado con un multímetro conectado a la salida del sensor y a la salida del PGA.

Referencias

[1] Cypress Semiconductor. Programmable System-on-Chip [en línea]. Disponible en: http://www.cypress.com/psoc/?source=CY-ENG-HEADER

[2] Cypress Semiconductor. PSoC®5LP Architecture TRM (Technical Reference Manual) Document No. 001-78426 Rev. *C October 1, 2013

[3] ARM®V7-M Architecture, Reference Manual – ARM.

[4] J. Yiu and A. Frame. ARM Cortex-M3 Processor Software Development for ARM7TDMI Processor. Burlintong: Newnes, 2010.

[5] Cypress Semiconductor. PSoC5® 5LP: CY8C58LP Family Datasheet.

[6] Cypress Semiconductor. CY8CKIT-059 PSoC®5LP Prototyping Kit Guide. Doc. # 001-96498 Rev. *A Abril 2, 2015.

[7] Fairchild Semiconductor. MM74C922/MM74C923, 16-Key Encoder/20-Key Encoder [en línea]. Disponible en: https://www.fairchildsemi.com/datasheets/MM/MM74C923.pdf

[8] J. Camargo. Guia de operación y aplicaciones prácticas de los microcontroladores Freescale. Bogotá: Editorial Universidad Distrital Francisco José de Caldas, 2013

[9] Cypress Semiconductor. ADC Succesive Aproximation Register (ADC_SAR) 3.0. Document No. 001-96049 Rev. *A

[10] Cypress Semiconductor. Universal Asynchronous Receiver Transmitter (UART) 2.50. Document No. 001-97175 Rev.**

En la actualidad existe una gran variedad de microcontroladores producidos por fabricantes de semiconductores de reconocida trayectoria en el área, con características y costos similares, presentando un sin número de posibilidades en cuanto a periféricos y aplicabilidad se refieren. Estos dispositivos tienen algunas limitaciones que no permiten la operación dinámica de sus periféricos, es decir, no se puede aumentar la cantidad de periféricos preexistentes en el mismo, así, surge en este caso la necesidad de cambiar de dispositivo o incluso de fabricante haciendo más costoso y posiblemente más complejo el producto o diseño a realizar. Además si el diseño incluye componentes analógicos como amplificadores operacionales se requiere su conexión externa para aumentar así el tamaño de este.

Con la aparición en el mercado del PSoC (Program System on Chip) muchos de estos problemas son solucionados, ya que en un mismo dispositivo se tiene un procesador, periféricos digitales configurables dinámicamente (contadores, PWM de 8, 16, 32 bits), periféricos analógicos configurables (conversor digital analógico, amplificadores operacionales entre otros) y celdas lógicas programables (PLD). En este libro se presenta toda la fundamentación para que el lector pueda entrar al apasionante mundo del PSoC, iniciando desde los principios fundamentales hasta el uso de periféricos complejos. En todos los casos se presentan ejemplos de aplicación detallados que ayudan al lector a comprender de manera práctica la operación y programación del PSoC.

UD Editorial

ISBN 978-958-8972-18-3

9 789588 972183