UNIVERSIDAD NACIONAL DE CHIMBORAZO

FACULTAD DE INGENIERÍA

ESCUELA DE INGENIERÍA ELECTRÓNICA Y TELECOMUNICACIONES

"Trabajo de grado previo a la obtención del Título de Ingeniero en Electrónica y Telecomunicaciones"

TRABAJO DE GRADUACIÓN

Título del proyecto:

“DISEÑO E IMPLEMENTACIÓN DE UN FILTRO ADAPTATIVO EN PSOC APLICANDO EL ALGORITMO RLS”

AUTORES:

DIEGO FERNANDO AMAGUAYA RAMOS

DAYSI NOEMÍ RAMÍREZ SATÁN

Director:

ING. FABIÁN GUNSHA

Riobamba – Ecuador

2016

Los miembros de tribunal de graduaciones del proyecto de investigación de título:

"DISEÑO E IMPLEMENTACIÓN DE UN FILTRO ADAPTATIVO EN PSOC APLICANDO EL ALGORITMO RLS"

Presentado por:

Diego Fernando Amaguaya Ramos

Daysi Noemí Ramírez Satán

Y dirigida por:

Ing. Fabián Celso Gunsha Maji

Una vez escuchado la defensa oral y revisado el informe final del proyecto de investigación con fines de graduación escrito en la cual se ha constado el cumplimento de las observaciones realizadas, remite la presente para uso y custodia en la biblioteca de la facultad de Ingeniería de la UNACH.

Para constancia de lo expuesto firman:

Ing. Paulina Vélez
Presidente del tribunal

..... **Firma**

Ing. Fabián Gunsha Máji
Director del proyecto

..... **Firma**

Ing. Giovanni Cuzco
Miembro del Tribunal

..... **Firma**

AUTORÍA DE INVESTIGACIÓN

La responsabilidad del contenido de este proyecto de Graduación, nos corresponde exclusivamente a: Diego Amaguaya, Daysi Ramírez e Ingeniero Fabián Gunsha; y el patrimonio intelectual de la misma a la Universidad Nacional de Chimborazo.

Diego Fernando Amaguaya Ramos

C.I. 060357032-6

Daysi Noemí Ramírez Satán

C.I. 060517019-0

AGRADECIMIENTO I

A Dios y mi familia por guiarme, cuidarme y por apoyarme incondicionalmente en cada día de mi vida.

Al Ingeniero Fabián Gunsha Director de Tesis por su asesoramiento y ayuda para la culminación del proyecto.

A mis amigos, compañeros y a todas aquellas personas que me apoyaron para obtener los resultados del proyecto.

Diego Amaguaya

AGRADECIMIENTO II

Primeramente, a Dios por haberme acompañado durante toda mi vida, especialmente en mis estudios.

A mis queridos padres Rosa y José que me apoyaron constantemente, en cada decisión que he tomado.

A mi familia, mis amigos que de alguna manera estuvieron presente y me brindaron su apoyo en el momento adecuado.

A la Universidad Nacional de Chimborazo por brindarme la oportunidad de obtener mi formación profesional.

Al nuestro Director de Tesis Ingeniero Fabián Gunsha, quien nos brindó su conocimiento para el desarrollo de nuestra tesis.

Daysi Ramírez

DEDICATORIA I

A mi padre y a mi madre.

Diego Amaguaya

DEDICATORIA II

A mi madre Rosa, por ser una fuente de guía y estimulo, por enseñarme a tener fortaleza para continuar con mis propósitos sin importar las dificultades que se presenten.

A mi padre José por apoyar mi decisión de continuar con mi formación académica y por el aporte hecho para culminar mi carrera.

A mis hermanos y toda mi familia, por ser fuentes de ánimo y alegría en los momentos difíciles.

A todos mis amigos y compañeros de estudio.

Daysi Ramírez

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

RESUMEN

El propósito de este trabajo de investigación es el diseño e implementación de un filtro adaptativo con el Algoritmo RLS en PSoC, con el objetivo de eliminar una interferencia que se encuentra presente en una señal de audio.

El filtro adaptativo que se utiliza consta de dos partes fundamentales que lo caracterizan: filtro programable y algoritmo adaptativo. El filtro programable consta de dos entradas como son: la señal de audio y ruido, al aplicar el algoritmo adaptativo RLS desarrolla la diferencia entre las dos señales y así proporcionará como resultado una salida filtrada es decir una señal recompuesta sin ruido.

La implementación y desarrollo del filtro Adaptativo se realiza en la tarjeta CY8CKIT-059 de la familia PSOC, que incorpora un procesador ARM Cortex-M3.

Los resultados obtenidos con la tarjeta PSOC se comparan con Matlab-Simulink para verificar la eficiencia de la señal recompuesta de audio.

UNIVERSIDAD NACIONAL DE CHIMBORAZO

CENTRO DE IDIAMAS INSTITUCIONAL

Lic. Karen Plua

1 de Julio del 2016

ABSTRACT

The purpose of this investigation is the design and implementation an adaptive filter with the RLS algorithm in PSoC; with objective eliminate the reference signal that found present in audio signal.

The adaptive filter that is used consists in two parts fundamental that characterize it: programmable filter and algorithm adaptive.

The programmable filter consists two input as are: the audio signal and noise, at apply algorithm RLS adaptive , develop the difference between two signal and so it affording as result an output filtered that is to say an restored signal without noise.

The implementation and development of the adaptive filter it is performed in the CY8KIT-059 card of the PSOC families that incorporate an ARM Cortex-M3 processor.

The results obtained with the card PSOC compared with Matlab-Simulink to verify the efficiency of audio signal recomposed.

INTRODUCCIÓN

El ruido es una perturbación que interfiere dentro de una señal muestreada alterando sus valores reales, para ello es necesario diseñar filtros que se adapten únicamente a las características espectrales de las perturbaciones dentro de la señal obtenida de tal forma que la señal original quede intacta. Este proceso se conoce como cancelación de ruido. Para la cancelación de ruido se usa técnicas de filtrado adaptativo que constituyen una herramienta fundamental del procesado de señales, siendo de gran importancia en múltiples aplicaciones en el ámbito de la Telecomunicaciones como: codificación de voz, comprensión y restauración de imágenes, cancelación de ruido, ecos acústicos, detección de radar, etc.

Los filtros adaptativos permiten, de una forma segura, confiable y dinámica, la atenuación de señales no deseadas. Estos filtros se configuran con el propósito de reducir casi a cero la señal de error, donde se quiere que la salida del filtro adaptativo sea igual a la señal deseada y sobretodo se actualicen sus coeficientes de muestra a muestra.

En esta tesis se presenta el diseño de un filtro Adaptativo con el algoritmo RLS en PSoC para la cancelación de ruido de una señal de audio. El algoritmo RLS permite encontrar los coeficientes o pesos del filtro para obtener el mínimo cuadrado de la señal de error es decir la diferencia entre la señal deseada y la señal ruido de forma recursiva. La tecnología PSoC se escoge con el fin de aprovechar las características que brinda este dispositivo, como es la velocidad de procesamiento y la versatilidad de los sistemas de entrada y salidas.

CAPÍTULO I

1. FUNDAMENTO TEÓRICO

1.1. RUIDO

Es cualquier señal de la naturaleza que aparece superpuesta a la señal útil y afecta la calidad de la información en su recepción y se puede clasificar de la siguiente manera.

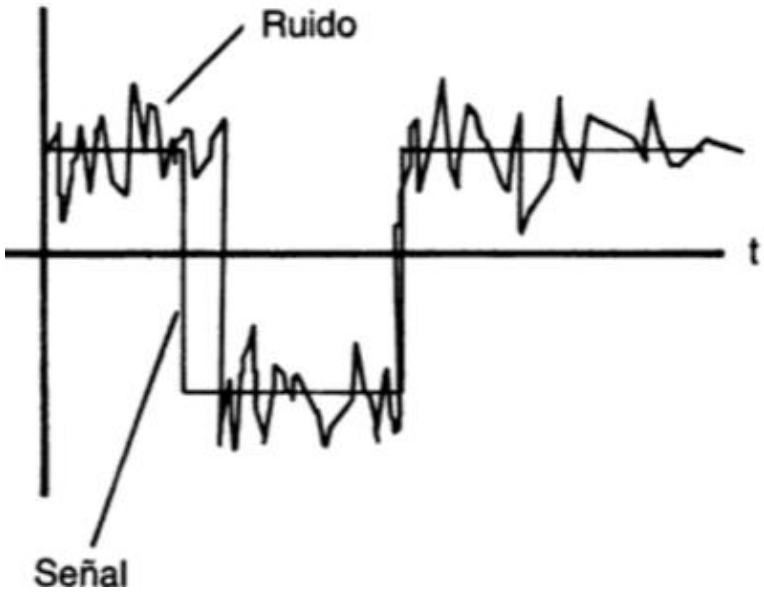


Figura 1.- Ruido.

Fuente: (Kuhlmann & Alonso, 1996).

1.1.1. Clasificación de los ruidos.

- **Ruido fluctuante**. - Ruido de presión sonora que fluctúa a lo largo del

tiempo. Las fluctuaciones pueden ser periódicas o aleatorias, este tipo de ruido se clasifica en función de tiempo.

- **Ruido Impulsivo**. - Se representa por impulsos, se caracterizan por un ascenso brusco del ruido. Estos impulsos suelen presentarse repetidamente en intervalos iguales de tiempo o bien repetitivos.
- **Ruido Continuo Constante**. - Presenta pequeñas fluctuaciones a lo largo del tiempo. Estas fluctuaciones deben ser menores de 5dB durante el periodo de observación.
- **Ruido causado por el hombre.** Las fuentes de este tipo de ruido son los mecanismos que producen chispas, como el equipo generador conmutador de energía eléctrica, los conmutadores de los motores eléctricos, las lámparas fluorescentes y los sistemas de encendido automotriz. Este ruido es de mayor intensidad en las zonas más densamente pobladas y en áreas industriales, considerado como ruido industrial.

1.2. INTRODUCCIÓN A LOS FILTROS ADAPTATIVOS

1.2.1 Filtros

Dispositivos que permite pasar, discriminar o suprimir un grupo de señales eléctricas, a través de él, a una cierta frecuencia o rango de frecuencias mientras previene el paso de otras, que puede modificar la amplitud y la fase. Los filtros pueden ser: analógicos que toman cualquier valor y los digitales toman solo valores discretos.

Figura 2.- Funcionamiento Externo Básico de un Filtro.

Fuente: Autores.

1.2.2. Filtros Digitales

Es un algoritmo implementado en hardware y/o software, como puedes ser un DSP[1] o un FPGA[2], que opera sobre una señal de entrada digital (discreta en tiempo y cuantizada en amplitud) y genera una señal digital de salida, con la finalidad de efectuar un proceso de filtrado. El término "filtro digital" se refiere al hardware o software específico que ejecuta el algoritmo (U.N.S, 2011).

Al implementar un filtro digital se debe tener en cuenta el uso de un ADC[3] para obtener una señal de entrada y un DAC[4] para enviar el resultado a una salida externa.

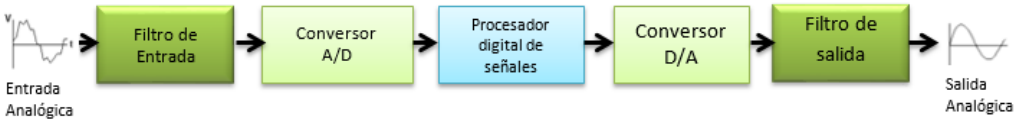


Figura 3 Diagrama de Bloque de un Filtro Digital.

Fuente: Autores.

El diseño de filtros digitales normalmente involucra una etapa de aproximación, en la que se genera una función de transferencia que satisface las especificaciones de la aplicación, y en donde normalmente se estudian respuestas tanto en el dominio de la frecuencia como del tiempo.

Luego se lleva a cabo la realización de la función de transferencia que se expresa en términos de una topología o redes de filtros, según las características del problema

[1]Procesador Digital de Señales
[2]Arreglo de Compuertas Programables en campo
[3]Convertidor Analógico Digital
[4]Convertidor Digital Analógico

y la disponibilidad de recursos en el procesador.

Los dos pasos anteriores parten de la base de un sistema de precisión infinita, y es por eso que se debe tener una etapa de implementación, relacionada con el hardware disponible y las rutinas de programación del procesador seleccionado. Para tener una precisión finita, obliga al diseñador a estudiar los efectos de los errores matemáticos en la respuesta del filtro. El desarrollo de los filtros digitales va creciendo continuamente y pueden ser:

- Filtros recursivos y no recursivos
- Filtro de abanico
- Filtro bidimensional
- Filtro adaptativo
- Filtros multidimensionales

1.2.2.1. Ventajas y Desventajas de los Filtros Digitales

Los filtros digitales juegan un papel muy importante en el procesamiento de la señal digital en el cual presentan algunas ventajas y desventajas como son:

Ventajas

- Alta confiabilidad y estables.
- La precisión del filtro solo está limitada por el tamaño de los datos.
- Menor sensibilidad a la temperatura y al envejecimiento.
- Los datos de la señal original y de la señal filtrada se puede almacenar para ser utilizados posteriormente.

- La respuesta en frecuencia de un filtro digital se puede ajustar automáticamente si se utiliza un procesador programable (DSP, FPGA, ARM[4], etc.).
- Puede filtrar varias señales con un solo hardware.
- Alta inmunidad al ruido.
- Fácil modificación de las características del filtro.
- Son de fase lineal.

Desventajas

- Requiere un número de coeficientes altos para aproximar filtros de transición brusca.
- La velocidad de operación de los filtros digitales en tiempo real depende de la velocidad del procesador utilizado y el número de operaciones matemáticas que se emplee en el proceso de filtrado.
- El retardo de fase puede no ser entero.

Los filtros digitales determinan los valores de los coeficientes, que puede describirse mediante la ecuación.

$$y(n) = \sum_{k=0}^{N} a_k y(n-k) + \sum_{k=0}^{M} b_k (n-k) \quad (1)$$

1.2.1. Filtros Adaptativos

Son sistemas que se caracterizan por modelar la relación entre señales, en tiempo real de forma iterativa. Estos filtros se basan en algoritmos matemáticos, en donde

[4]Advanced RISC Machani

pueden ajustar a sus variables de forma automática para así obtener una señal deseada.

La diferencia de los filtros adaptativos con filtros digitales, pueden cambiar sus coeficientes de acuerdo con el algoritmo adaptativo basado en la señal de error y pueden ser implementados por software y hardware. El algoritmo es quien disminuye el valor cuadrático de la señal de error e(n), para así alcanzar la convergencia y adaptación del sistema.

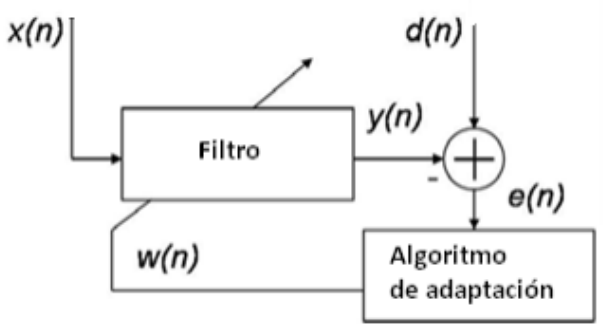


Figura 4.- Filtro Adaptativo.

Fuente: (Vazquess Burgos, 2010)

En la **figura 4** podemos observar que x(n) señal de entrada, d(n) señal deseada, y(n) salida del filtro, e(n) señal de error. La señal de error es la diferencia entre la señal deseada d(n) y la salida del filtro y(n).

1.2.1.1. Estructuras del Filtrado Adaptativo

Las características que presentan estos filtros adaptativos, son de gran utilidad, no solo en el área de procesamiento digital de señales sino también en el área de control. Existen cuatro tipos de sistemas adaptativos.

- Identificador del sistema
- Modelado inverso

- Predictor lineal
- Cancelador de ruido

Identificador del sistema

Es una aplicación que permite obtener un modelo lineal de un sistema desconocido, donde la señal de entrada del sistema y el filtrado adaptativo tiene la misma señal de entrada, la salida del sistema indica la señal deseada para el filtro adaptativo.

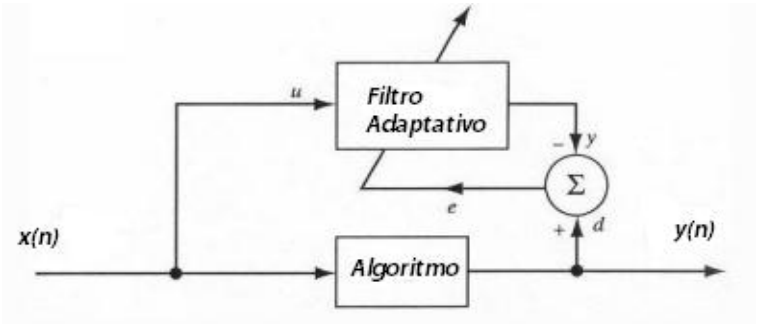


Figura 5.- Filtro Adaptativo-Identificador.

Fuente: (Vazquess Burgos, 2010).

Modelado Inverso

La estructura tiene un modelo inverso que representa el que mejor se adapta a un sistema desconocido con ruido. En el caso de sistemas tiene una función de transferencia igual al reciproco (inverso) de la función de transferencia del filtro. Esta estructura es utilizada para la ecualización e igualación de canales de transmisión de datos.

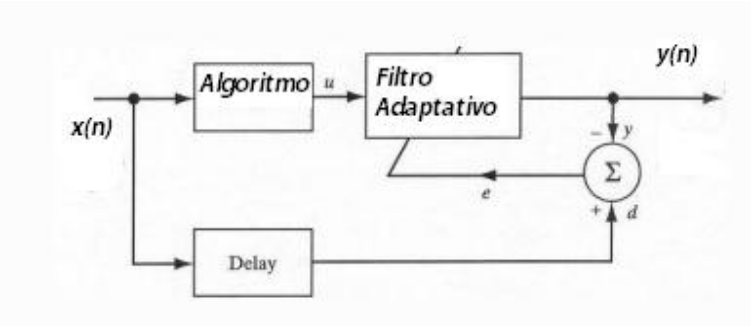


Figura 6.- Filtro Adaptativo-Modelado Inverso.

Fuente: (Vazquess Burgos, 2010).

Predictor Lineal

La función del filtro adaptativo es proveer la mejor predicción del valor presente de una señal aleatoria. El valor presente de la señal es la respuesta deseada para el filtro, donde los valores pasados de la señal son la entrada, la salida del filtro adaptativo corresponde al error, la señal d(n) es igual a la señal de entrada x(n), el error es la diferencia entre la señal deseada y la salida del filtro adaptativo y(n) como se aprecia en la **figura 7**.

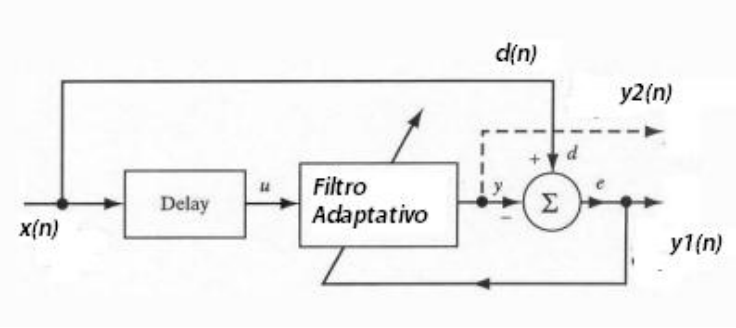


Figura 7.- Filtro Adaptativo- Predictor Lineal.

Fuente: (Vazquess Burgos, 2010).

Cancelación de Ruido

Es usado para eliminar una señal no deseada, interferencia o ruido de una señal de

interés.

La cancelación comienza a optimizar la salida hasta cierto grado. La señal deseada o primaria sirve como la respuesta deseada para el filtro adaptativo.

La señal de referencia ingresada a la entrada del filtro, es un ruido o conjunto de ruido que pueden ser localizados cerca de la señal deseada, donde el ruido en la señal deseada es débil y la componente del ruido en la señal de referencia resulta ser una proporción de la señal interferente (Vazquess Burgos, 2010).

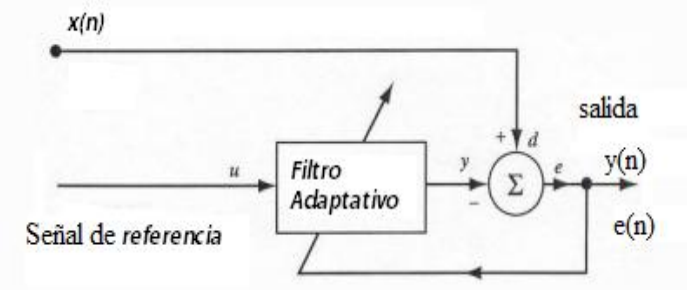


Figura 8.- Filtro Adaptativo- Cancelador de Ruido.

Fuente: (Vazquess Burgos, 2010)

Usualmente, no es admisible el sustraer ruido de una señal recibida debido a que una operación de este tipo podría causar resultados desastrosos, al incrementar la potencia promedio de ruido a la salida. sin embargo, cuando el filtro y la sustracción son controlados por un proceso adaptativo es posible conseguir un desempeño superior del sistema comparado al filtrado directo.

Básicamente, un cancelador adaptativo de ruido es un sistema de retroalimentación de lazo cerrado adaptativo con dos entradas.

En la **tabla 1** indican algunas aplicaciones de los cuatro tipos de configuración del

filtrado adaptativo se toman estas aplicaciones de los diversos campos de control, sismología, electrocardiografía comunicaciones y radar.

Tabla 1.- Aplicaciones de Filtros Adaptativos.

Fuente: Autores.

Tipo de Filtro Adaptativo	Aplicaciones
Identificación	Identificación de sistemas
	Modelado en capas de la tierra
Modelado Inverso	De convolución predictiva
	Ecualización Adaptativa
	Ecualización ciega
Predicción	LPC[5]
	ADPCM[6]
	Análisis auto regresivo de espectro
	Detección de señales
Cancelación de Ruido	Cancelación adaptativa de ruido
	Cancelación de eco
	Modelo adaptativo de haz

1.2.1.2. Filtro FIR [7]

Son aquellos que obtienen la salida a partir de las entradas actuales y anteriores, a los FIR se les pueden representar por la ecuación 2 donde M es la longitud del filtro.

[5]Codificación por Predicción Lineal
[6]Modulación por Código Pulsado Diferencial
[7] Respuesta Finita al Impulso

$$y(n) = \sum_{k=0}^{M-1} b_k x(n-k) \quad (2)$$

$$y(n) = b_0 x(n) + b_1 x(n-1) + \cdots + b_{M-1} x(n-[M+1]) \quad (3)$$

El orden del filtro está dado por M, es decir el número de coeficientes del filtro b_k, la salida es expresada como la convolución de la señal de entrada x(n) como un filtro h(n).

Ante un estímulo impulsional, la respuesta es finita. La salida y(n) puede escribirse como la convolución de la entrada x(n) con la respuesta impulsional h(n).

$$y(n) = \sum_{K=0}^{M-1} h(k) x(n-k) \quad (4)$$

Por la función de transferencia seria:

$$\sum_{K=0}^{M-1} h(k) z^{-k} \quad (5)$$

Donde se observa como un polinomio de grado M-1 en la variable z^{-1}. Existen varias formas de representar a un filtro, unas de ellas son: forma directa, cascada y por realización polifásica, la más usual es la forma directa ilustrada en la **figura 9**.

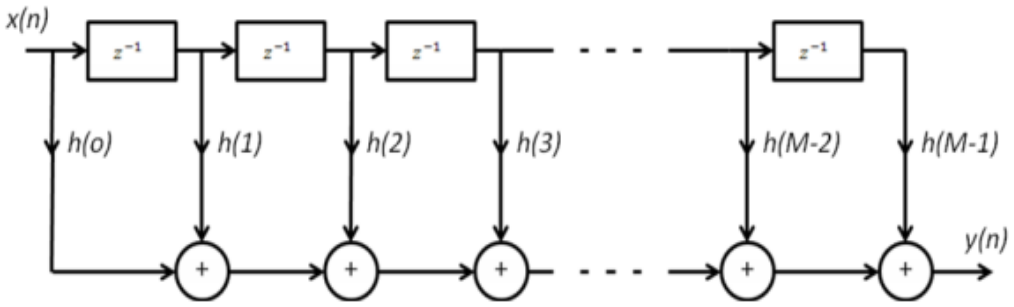


Figura 9.- Estructura de un Filtro FIR.

Fuente: (Vàzquez, 2010).

Como podemos observar en la **figura 9**, el sistema requiere de M-1 posiciones de memoria para almacenar el número de entradas anteriores, tiene una complejidad de M multiplicaciones y un M-1 sumas para cada punto de salida. “Las salidas consisten en una combinación lineal ponderada de M-1, valores anteriores de entrada y el valor actual de la entrada ponderada” (G.Proakis, 2007).

1.2.1.3. Filtro IIR[8].

Los filtros IIR, son llamados así porque su respuesta al impulso da en un número infinito de muestras, son filtros que si tienen retroalimentación.

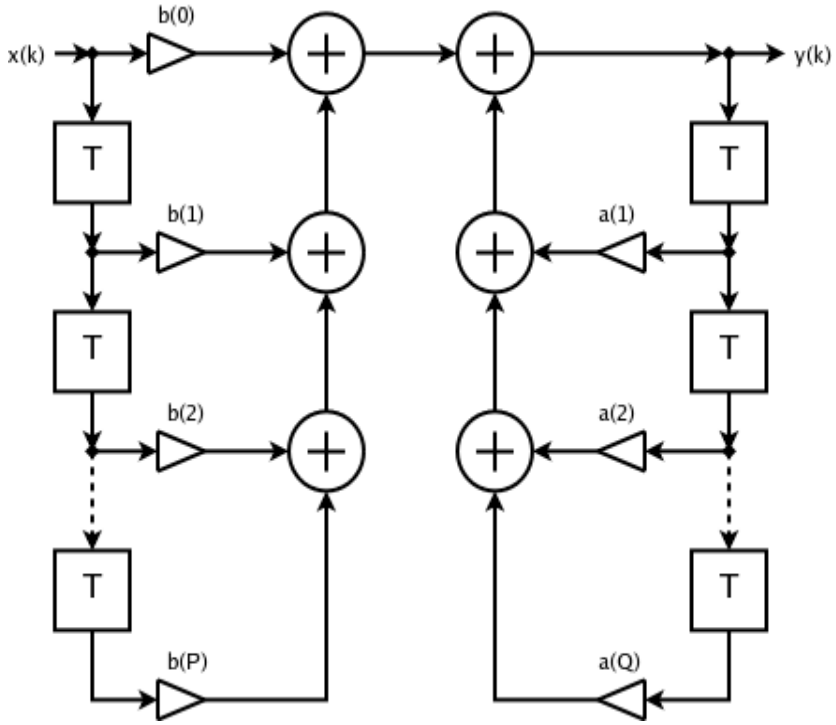


Figura 10.- Estructura del Filtro IIR.

Fuente: https: (Vàzquez, 2010)

[8]Respuesta Infinita de Impulso

Se pensaría que en este tipo de filtros requieren más cálculos para ejecutar el filtrado de la señal, en una parte de los valores de entrada; también se encuentran términos de salida en la expresión del filtro.

"De hecho, pasa todo lo contrario, ya que un filtro recursivo requiere una expresión de menor orden (por lo tanto, muchos menos términos a tratar por el procesador) que su equivalente filtro no- recursivo" (Watanabe Ruiz, 2012).

$$H(z) = \frac{\sum_{k=0}^{M} b_k z^{-k}}{1+\sum_{k=1}^{N} a_k . z^{-k}} \quad (6)$$

Los filtros IIR se caracterizan por la ecuación 6. Estos filtros IIR son eficientes, su cálculo es menor y requieren de un menor número de coeficientes.

1.3. ALGORITMO RECURSIVE LEAST SQUARES (RLS[9]).

1.3.1. RLS.

El algoritmo RLS, son utilizados en filtros adaptativos que proporcionan un método para determinar los coeficientes del filtro, y así obtener el mínimo cuadrado de la señal de error en forma recursiva. El algoritmo adaptativo RLS es una extensión del algoritmo mínimos cuadrados (LM[10]), que tiene un filtro de Kalman, quien minimiza el error cuadrático. Este filtro es uno de los más utilizados para el tratamiento de convergencia, desajuste y sobre todo para la depuración de señales, permitiendo el paso de señales deseadas y elimina el ruido.

[9] Mínimos Cuadrados Recursivos
[10] Mínimos Cuadrados

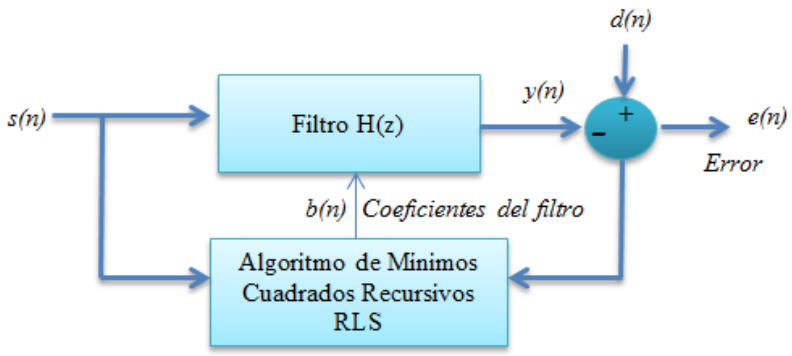


Figura 11.- Estructura del Filtro RLS.

Fuente: Autores.

En el algoritmo RLS los coeficientes del filtro son b_M (n), M es la longitud del filtro y la señal de error se define de la siguiente ecuación.

$$\varepsilon(n) = \sum_{i=0}^{n} e^2 = \sum_{i=0}^{n} (d(i) - y(i))^2 \quad (15)$$

$$e(n) = d(n) - y(n) \quad (16)$$

Al modificar el método de mínimos cuadrados mediante la inclusión de una exponencial "factor de olvido" λ^{n-1}, ($0 < \lambda \leq 1$) a cada término de error, se obtendrá las siguientes ecuaciones.

$$\varepsilon'(n) = \sum_{i=0}^{n} \lambda^{n-i} - e^2 \quad (17)$$

$$\varepsilon'(n) = \sum_{i=0}^{n} \lambda^{n-i} (d(i) - y(i))^2 \quad (18)$$

$$\varepsilon'(n) = \sum_{i=0}^{n} (d'(i) - y'(i))^2 \quad (19)$$

Donde $d'(i) = \sqrt{\lambda^{n-1} d(i)}$, $y'(i) = \sqrt{\lambda^{n-1} y(i)}$. El propósito del factor de olvido o factor de ponderación λ es ponderar los puntos de datos más recientes y permite a los coeficientes del filtro adaptarse a las características estadísticas variantes en el tiempo. La salida del filtro FIR está dada por la suma de convolución.

$$y(n) = \sum_{k=}^{M-1} b_k f(n-k) \tag{20}$$

1.3.2. Coeficientes del Filtro RLS.

Dado que el algoritmo será recursivo en el tiempo, es necesario introducir un índice de tiempo en el vector de coeficientes del filtro y en la secuencia de error. Así, definimos el vector de coeficientes del filtro $b_k(n), k = 0,1 \ldots\ldots M-1$, que minimiza el error $\varepsilon'(n)$, se encuentran mediante el establecimiento de las derivadas con respecto a cada uno de los coeficientes de filtro $b_k(n)$ igual a cero. (G.Proakis, 2007)

$$k = 0, 1, \ldots\ldots, M-1,$$

La minimización ε_M con respecto al vector de coeficientes del filtro $b_k(n)$, que proporciona un conjunto de ecuaciones lineales de los coeficientes $b_k(n)$.

$$R(n)b(n) = P(n) \tag{21}$$

Donde R(n) es la matriz de correlación (estimada) de la señal, P(n) es el vector de correlación cruzada, a R(n) entre d(n) y x(n). Por medio de esta ecuación 21, se

puede encontrar a los coeficientes que minimizan error con respecto al vector de coeficientes del filtro $b(n)$ (G.Proakis, 2007).

$$b(n) = R(n)^{-1}P(n) \tag{22}$$

Donde $b(n) = [b_0(n)\ b_1(n)\ b_2(n) \ldots\ldots b_{M-1}(n)]^T$, y M x 1 es un vector de coeficientes de filtro. Al elegir λ se debe tomar en cuenta lo siguiente: Si λ < 1, menor será la contribución de las muestras previas y esto hace que el filtro sea más sensible a las muestras recientes, esto significa que habrá más fluctuaciones en los coeficientes. Si $\lambda = 1$ es conocido como una ventana creciente del algoritmo RLS. Y puede ser calculado de forma recursiva como se expresa en la ecuación 23.

$$R(n) = \lambda \mathrm{R(n-1)} + \mathrm{f(n)}f^T(n) \tag{23}$$

Al aplicar el lema de la inversión de matrices permite calcular a los coeficientes del filtro de forma recursiva (Rodriguez & Caamaño, 2008) .

$$(A + BCB^T)^{-1} = A^{-1} - A^{-1}(C^{-1} + B^T A^{-1})^{-1} B^T A^{-1} \tag{24}$$

Si $A = R^{-1}(n-1)$, $B = f(n)$ y $C = \lambda^{-1}$, al reemplazar en la ecuación 23 obtendremos lo siguiente:

$$R^{-1}(n) = \lambda^{-1}[R^{-1}(n-1) - \frac{R^{-1}(n-1)f(n)f^T R^{-1}(n-1)}{\lambda + f^T(n)R^{-1}(n-1)f(n)}] \tag{25}$$

Al aplicar la matriz de inversión se podrá definir la recursividad con la ecuación (26).

$$P(n) = \lambda \mathrm{P}(\mathrm{n}-1) + \mathrm{d}(\mathrm{n})\mathrm{f}(\mathrm{n}) \quad (26)$$

Al combinar las ecuaciones (18), (19) con la ecuación (16) se obtendrá la base para el diseño del filtro RLS con el vector de ganancia de Kalman que se definirá como:

$$k(n) = \frac{R^{-1}(n-1)f(n)}{\lambda + f^T R^{-1}(n-1)f(n)} \quad (27)$$

1.3.2.1. Resumen de Ecuaciones del algoritmo RLS

Las ecuaciones resumidas que describen la operación del filtrado con el algoritmo, quien utiliza el filtro de Kalman para hallar su solución.

- Analizar la entrada al filtro f(n).
- Calcular la salida del filtro utilizando el anterior conjunto de coeficientes del filtro b(n-1).

$$y(n) = f^T(n)b(n-1) \quad (28)$$

- Calcular el error

$$e(n) = d(n) - y(n) \quad (29)$$

- Calcular la ganancia de Kalman vectorial.

$$k(n) = \frac{R^{-1}(n-1)f(n)}{\lambda + f^T R^{-1}(n-1)f(n)} \quad (30)$$

- Actualización de la matriz inversa de correlación R^{-1}

$$R^{-1}(n) = x^{-1}[R^{-1}(n-1) - k(n)f^{T}(n)R^{-1}(n-1)] \quad (31)$$

- Actualización de los coeficientes del filtro.

$$b(n) = b(n-1) + k(n)e(n) \quad (32)$$

Donde M es la longitud del filtro y el vector de la señal de entrada en el instante n se designa como:

$$x_M(n) = [x(n), x(n-1), x(n-2), \ldots., (n-M+1)] \quad (33)$$

1.3.3. Algoritmo RLS rápidos

"Los algoritmos RLS en la forma directa y los de raíz cuadrada tienen una complejidad de M^2, por lo que se han estudiado diferentes métodos para simplificarlo, esto se logra evitando las multiplicaciones de matrices en el cálculo del vector de ganancia de Kalman k(n)" (Apolo Castillo & Córdova Medina, 2010).

Este algoritmo es menos costoso computacionalmente, la simplificación es atribuida por Kaczmarz donde $\gamma(k)$ es un vector.

$$\gamma(k) = c.\frac{r(k-1)}{\alpha+\gamma^{T}(k-1)r(k-1)} \quad (34)$$

Donde α es una constante positiva que permite evitar indeterminaciones cuando el vector de regreso se anula y c permite sintonizar la velocidad de convergencia del algoritmo y debe cumplir con $0 \leq c \leq 2$.

1.4. Filtro de Kalman

Es un algoritmo de procesamiento de datos óptimo recursivos. Su optimización minimiza un criterio determinado e incorpora toda la información que se le suministra para determinar el filtrado. Recursivo porque no requiere almacenar todos los datos previos y procesarlos cada vez que llega una nueva información.

El filtro de Kalman es el principal algoritmo para estimar los sistemas dinámicos presentados en forma de espacio-estado ya que el sistema es descrito por un conjunto de variables denominadas de estados. La solución es óptima del filtro es combinar toda la información observada y el conocimiento previo acerca del comportamiento del sistema para producir una estimación del estado de tal manera que el error es minimizado estadísticamente. El término recursivo significa que el filtro recalcula la solución cada vez que una nueva observación o medida ruidosa es incorporada en el sistema (Cordoba; Duran, 2010).

El filtro de Kalman permite estimar el estado $x \epsilon \mathbb{R}^n$ de un proceso controlado en tiempo discreto que es administrado por una ecuación en diferencia lineal estocástica

$$x_{k=} A\, x_{\,k-1} + B_k + W_{k-1} \tag{35}$$

Con una medición $z \epsilon \mathbb{R}^n$ que es:

$$x_{k=} Hx_{\,k} + V_k \tag{36}$$

Las variables aleatorias W_k y V_k representa el ruido del proceso y de la medición. Se asumen que son independientes una de la otra.

$$p(W) \sim N(0, Q) \tag{37}$$

$$p(V) \sim N(0, R) \tag{38}$$

La covarianza del ruido del proceso Q y la covarianza del ruido de la medición R, son matrices que pueden cambiar con cada paso del tiempo.

"La matriz A de $n * n$ relaciona el estado de $k - 1$ con el estado actual. La matriz B de $n * 1$ se relaciona con la entrada opcional de control $u \epsilon \mathbb{R}^n$ al estado x. La matriz H de $m * m$ relaciona el estado con la medición" (Castañeda Cardenas, Nieto Areas, & Ortiz Bravo, 2013).

Los orígenes computacionales del filtro.

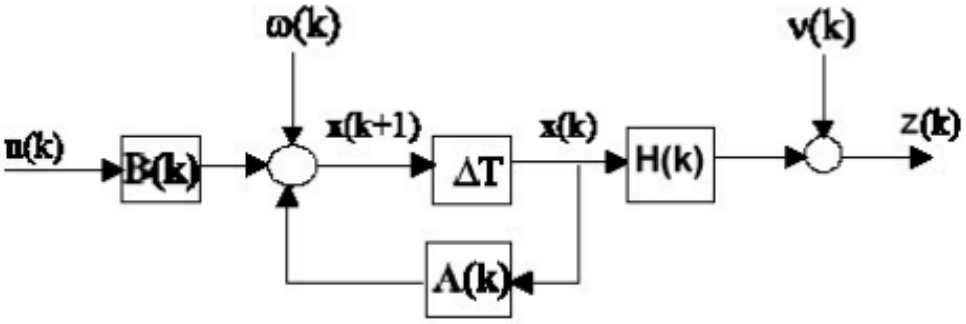


Figura 12.- Relación de Ruidos.

Fuente: (Castañeda Cardenas, Nieto Areas, & Ortiz Bravo, 2013)

Se define $x' \epsilon \mathbb{R}^n$ como el estado estimado a priori en el paso k de un paso previo de proceso, y $x' \epsilon \mathbb{R}^n$ es el estado estimado a posteriori de la medición z_k y se define el error como:

$$e_k^- = x_k - x'^-_k \tag{39}$$

$$e_k = x_k - x'_k \tag{40}$$

Matriz de correlación del error a priori es:

$$P_k^- = E[e_k^- e_k^{-T}] \tag{41}$$

La covarianza del error a posteriori es:

$$P_k = E[e_k e_k^t] \tag{42}$$

El filtro de Kalman tiene una retroalimentación, el filtro estima el estado del proceso en un tiempo y después obtiene la retroalimentación en forma de mediciones (ruidosas).

Las ecuaciones del filtro de Kalman se dividen en dos grupos de ecuaciones: actualización de tiempo y actualización de mediciones.

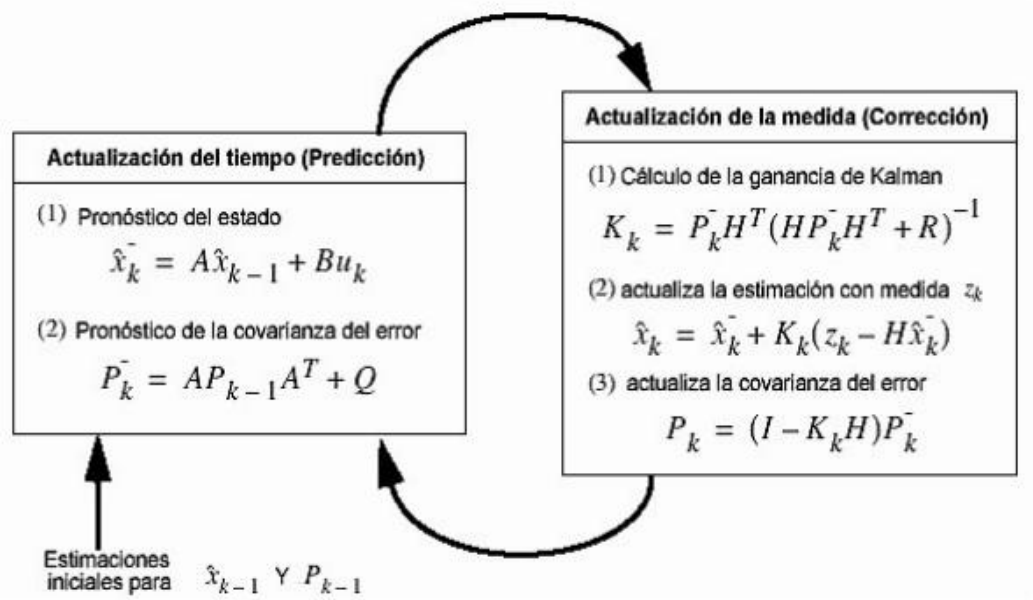


Figura 13.- Funcionamiento del filtro de Kalman.

Fuente: (Prado Obando, 2005)

Ecuaciones de mediciones de tiempo son responsables de proyectar hacia adelante el estado actual, estimaciones de error y covarianza para obtener los estimados de priori del siguiente paso en el tiempo. (Castañeda Cardenas, Nieto Areas, & Ortiz Bravo, 2013)

"Ecuaciones de actualización de medida son vistas como ecuaciones correctoras. El algoritmo del filtro de Kalman reúne los algoritmos de predicción-corrección para la solución de problemas numéricos" (Castañeda Cardenas, Nieto Areas, & Ortiz Bravo, 2013).

1.5. TARJETA DE DESARROLLO PSOC[11].

1.5.1. PSOC.

Son sistemas programables en un solo chip (microcontrolador) que se asemeja a un lego, totalmente dinámico y muy versátil, cuenta con innumerables dispositivos electrónicos que se puede modificar internamente ya sea análogos y digitales para luego programarlos en el lenguaje C o Asambler.

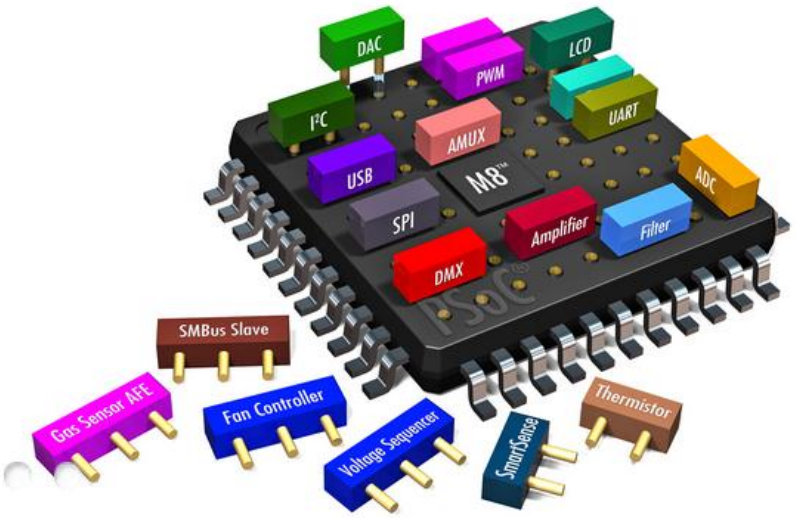


Figura 14.- PSoC.

Fuente: http://www.cypress.com/file/113041/downloa

[11]Sistema Programable en un solo Chip

1.5.1.1. Familias del PSOC.

La empresa Cypress va desarrollando e incorporando nuevas características a cada familia de PSoC, hasta el momento existen tres familias diferentes.

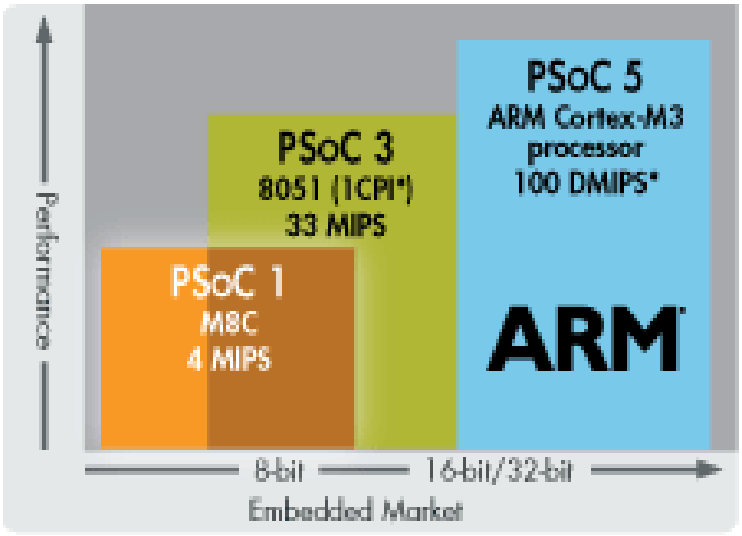


Figura 15.- Familias de PSoC.

Fuente:http://psocenespanol.blogspot.com/2011/09/el-nucleo-de-psoc.html.

- Familia CY8C2XXXX (PSOC 1): M8C processor speeds up to 24 MHz
- Familia CY8C3XXXX (PSOC 3): Single cycle 8051 CPU[12] core.
- Familia CY8C5XXXX (PSOC 5): 32-bit ARM Cortes-M3 CPU core.

PSoC 1

Tiene un procesador M8C que alcanza una velocidad de 24Mz, su arquitectura es de tipo Harvard que logra simplificar la programación en tiempo real.

[12]Unidad Central de Procesos
[13]Dhrystone Million Instructions per Second

PSoC 3

Tiene un set de instrucciones tipo RISC[14], con una arquitectura altamente configurable su velocidad es de 67 MHz, mayor que el PSoC 1, cuenta con un vector programable para el control de interrupciones.

PSoC 5

Cuenta con una CPU ARM córtex-M3, que es un procesador de bajo consumo de potencia de 32 bits, su arquitectura es de tipo Harvard, diseñado para un manejo de interrupciones a gran velocidad.

1.5.2. Arquitectura general de PSoC.

Los PSoC tienen una arquitectura altamente configurable en un sistema-en-chip. Integran circuitos analógicos y digitales, configurables, que son controlados por un microcontrolador, reduciendo así: el tiempo de diseño, el espacio en la placa, el consumo de energía y sobre todo el costo **figura 16**.

1.5.3. Herramientas de programación para los PSoC

Para el desarrollo de los programas de PSoC existen diferentes softwares de programación según su categoría, como es: Para PSoC 1 se utiliza un entorno gráfico de programación llamado PSoC Designer, a diferencia de PSoC 3 y PSoC 5 que utilizan las herramientas de desarrollo PSoC Creator y PSoC Programmer.

[14]Computador con Conjunto de Instrucciones Reducidas

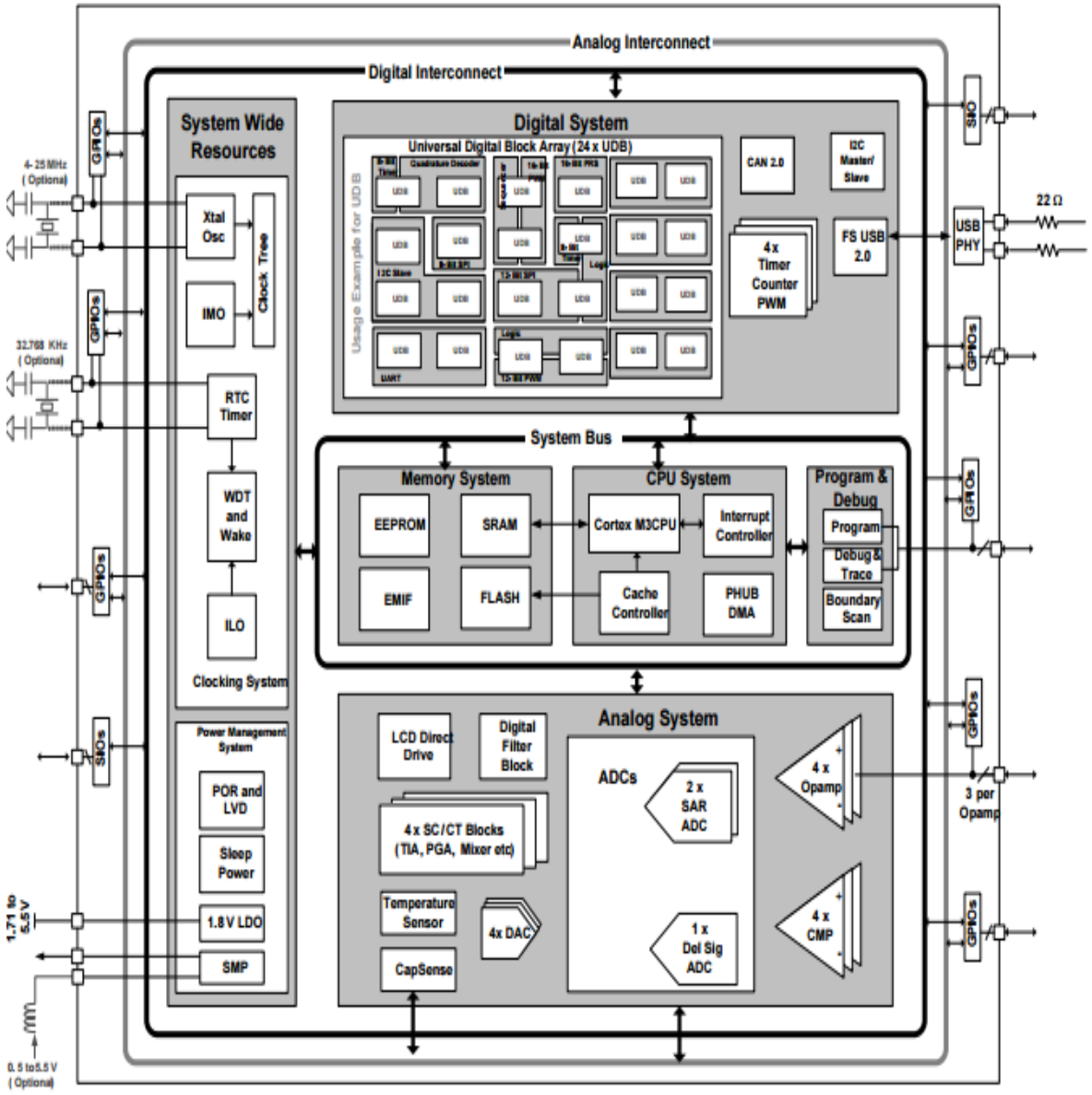


Figura 16.- Arquitectura del PSoC.

Fuente: http://red.uao.edu.co/bitstream/10614/7810/1/T05809.pdf

PSoC Programmer.

Es una interfaz gráfica de usuario simple que conecta al hardware de programación para programar y configurar dispositivos PSoC (reloj y dispositivos de Cypress). EL software programador de PSoC es compatible con todos los dispositivos de hardware y la programación PSoC de Cypress.

PSoC Creator.

Es un entorno IDE[15] para el desarrollo de software, combinado con un editor de diseño gráfico el cual forma un único y poderoso entorno de diseño de hardware / software. Este enfoque de co-diseño ayuda a los diseñadores a crear diseños nuevos, software embebido, sin limitaciones de tamaño de código, además utiliza una herramienta de compilación que es el Keil para PSoC 3 y GNU[16] para ARM como es el caso de PSoC 5.

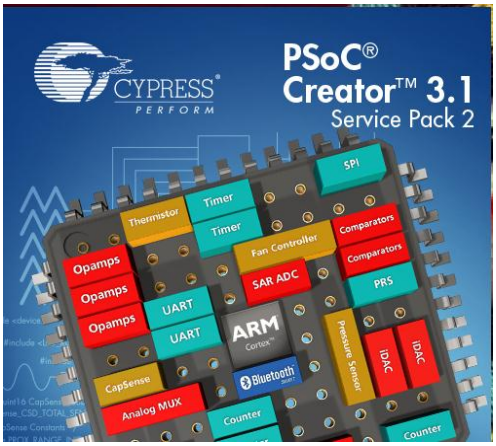


Figura 17.- PSoC Creator 3.1.

Fuente: http://www.cypress.com

[15]Entorno de Diseño Integrado
[16]Sistema Operativo de tipo Unix

Principales Herramientas de desarrollo de PSoC Creator.

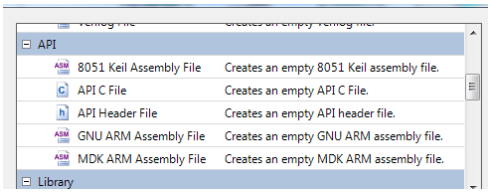


Figura 18.- Herramienta de desarrollo firmware.

Fuente: Autores, PSoC Creator.

- Captura de Esquemáticos: Proporcionan al usuario una interfaz amigable para describir sus diseños en forma de arrastrar y soltar.
- Herramientas para desarrollo de firmware: Con código comprimido (mediante las APIS[17]) y resultado de sintaxis.
- Biblioteca de bloques completo: Incluyendo I2C[18], USB[19], UART[20], SPI[21] y BLE[22]. Cuenta con más de 120 bloques predefinidos con datasheet que además incluye los APIS a utilizar para cada bloque.
- Herramientas de Depuración (Debbug): Con breakpoints para verificar el funcionamiento del proyecto.
- Herramienta Verilog: Para la descripción de boques personalizados también mediante diagrama de máquina de estados o esquemas de captura.
- Configuración de todos los recursos-bloques: a través de la interfaz gráfica de usuario.
- Editor y compilador: Para C y Assembler se usa compiladores ARM.

Para el desarrollo del filtro Adaptativo se utiliza un kit de PSoC más avanzado como es un PSoC 5, el cual permitirá cumplir con los objetivos de este proyecto.

[17]Interfaz de Programación de Aplicaciones
[18]Inter-Integrated Circuit.
[19]Bus Universal en Serie.
[20]Transmisor-Receptor Asíncrono Universal
[21]Serial Peripheral Interface
[22]Boque Lineal Ecualizador

1.5.4. CY8CKIT-059 PSOC 5LP.

Es un sistema programable basado en ARM Cortex-M3, que puede manejar gran cantidad de canales de adquisición de datos, entradas y salidas analógicas en cada pin de propósito general (GPIO[23]), opera múltiples funciones en diferentes tiempos brindando una flexibilidad para diseñar soluciones personalizadas en el sistema.

Figura 19 **.-**CY8CKIT-059 **PSoC 5.**

Fuente: http://www.cypress.com/documentation/development-kitsboards/cy8ckit-059-psoc-5lp-prototyping-kit-onboard-programmer-and

1.5.4.1. Características Técnicas del CY8CKIT-059 PSoC 5

- CPU de 32 bits ARM Cortex-M3
- 32 entradas de interrupción
- Pulsador
- 3x ADC (12 bits 1Msps[24] SAR[25] ADC, de 20 bits DelSig ADC).

[23]General Purpose Input/Output.
[24]Millones de Instrucciones por segundo.
[25]Registro de Aproximaciones Sucesivas.

- 4x DACs 8 bits y programables bloques de matriz analógicas
- 24 canales de acceso directo de memoria(DMA[26])
- Filtro digital de 24 bits (DFB[27])
- Sistema de E / S
- Periféricos digitales
- Voltaje de operación (1.71V a 5.50V)
- Reloj programable de precisión
- Memoria flash de 256 KB, con características de cache y seguridad
- Memoria flash adicional de 32 KB para el código de corrección de errores (ECC[28])
- Memoria RAM de 64 KB
- 2 KB de EEPROM
- Interfaz UART, SPI, I2S, LIN 2.0, I2C
- conector micro-USB para permitir el desarrollo de aplicaciones USB.
- Cable USB
- Led

1.5.4.2. Estructura de PSoC 5 CY8CKIT-059

En la **figura 20** indica la estructura interna que tiene PSoC 5LP donde está integrado los periféricos analógicos y digitales que son configurables, todo esto en un solo chip.

[26]Acceso Directo de Memoria
[27]Bloque de Filtro Digital.
[28]Codigo de Corrección de Errores.

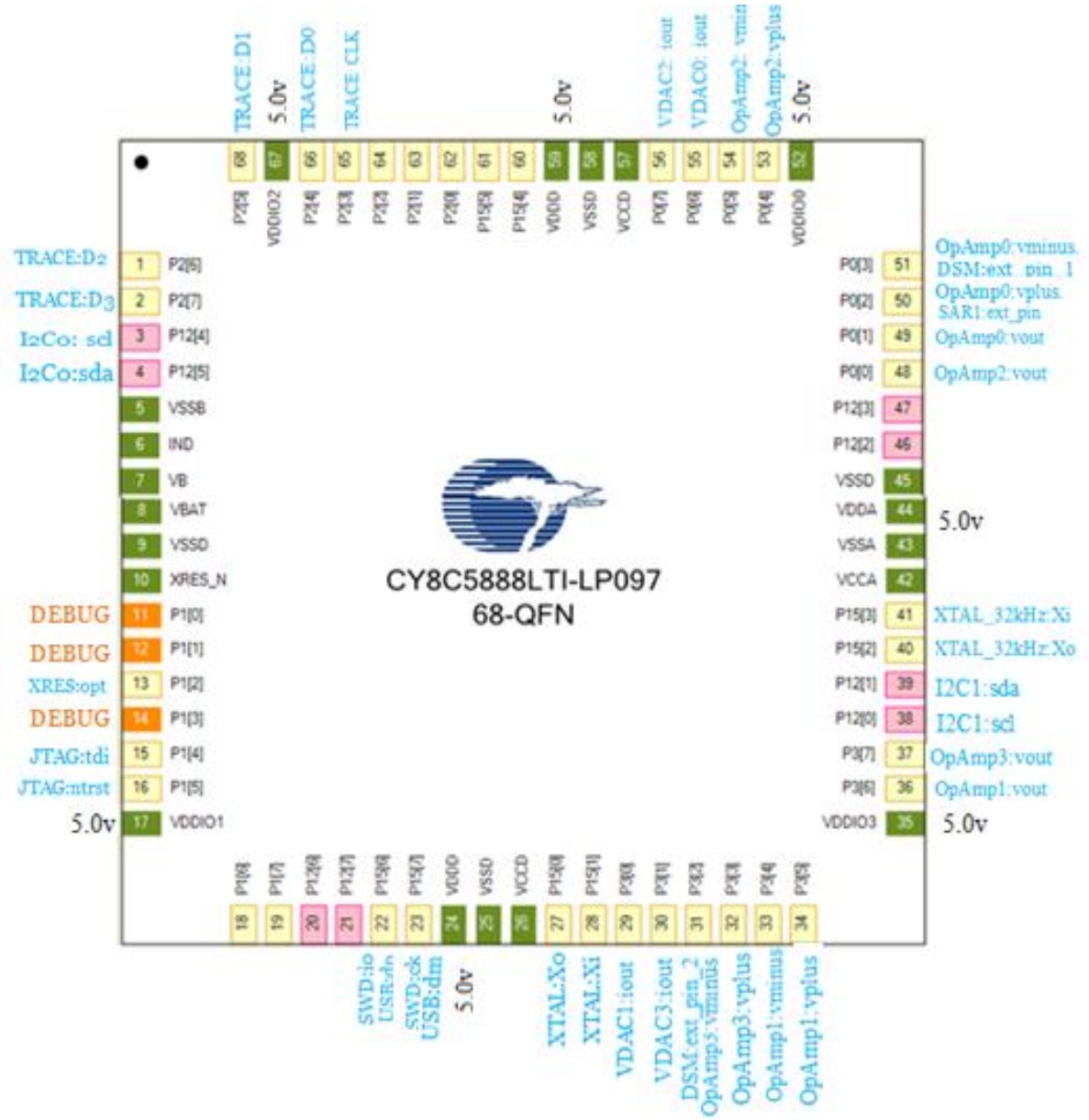


Figura 20.- Estructura Interna.

Fuente: Autores.

CPU-ARM Cortex-M3

Es un procesador de 32 bits que tiene alta densidad de código y rendimiento con el conjunto de instrucciones Thumb-2. Los ARM son para aplicaciones en tiempo real que permite desarrollar plataformas a bajo costo de alto rendimiento.

DMA (Acceso Directo a Memoria).

Está constituido por 24 canales y 128 descriptores de transacción. Los TDS[29] contienen toda la información necesaria para la transferencia de datos. El DMA es utilizado para mover datos de 8, 16, 32 bits de una fuente de destino sin intervención de la CPU. Este se utiliza cuando requiera datos de un ADC y permita a la CPU hacer simultáneas tareas.

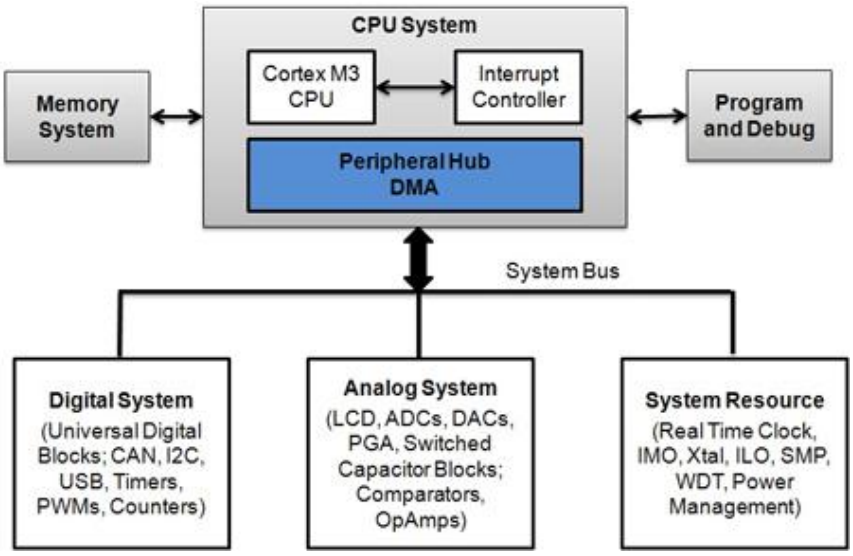


Figura 21.- Arquitectura del PSOC 5 LP

Fuente: http://www.embedded.com/print/4414628

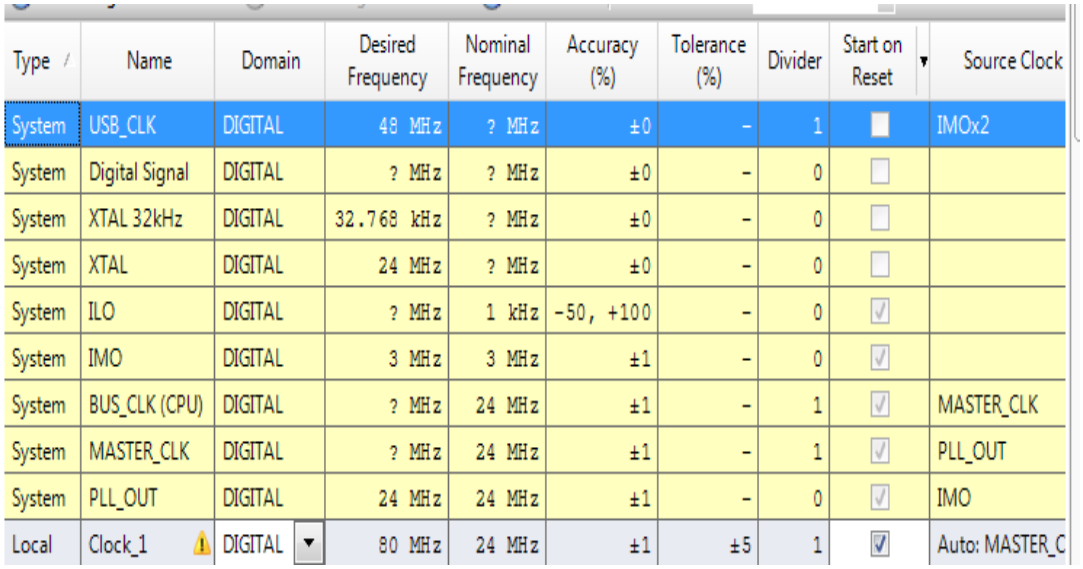
DMAC

Controla las transferencias de datos utilizando los periféricos HUB[30] (PHUB). Es un bus de alto rendimiento para acceder a diferentes periféricos y así transferir los datos.

[29]Descriptores de Transacción.
[30]Bus de Datos de Conexión.

Sistemas de reloj

El sistema que se muestra en la **figura 22** se encarga de configurar el reloj para todo tipo de dispositivos. Permite al usuario elegir con precisión una amplia gama de frecuencias.



Type	Name	Domain	Desired Frequency	Nominal Frequency	Accuracy (%)	Tolerance (%)	Divider	Start on Reset	Source Clock
System	USB_CLK	DIGITAL	48 MHz	? MHz	±0	-	1	☐	IMOx2
System	Digital Signal	DIGITAL	? MHz	? MHz	±0	-	0	☐	
System	XTAL 32kHz	DIGITAL	32.768 kHz	? MHz	±0	-	0	☐	
System	XTAL	DIGITAL	24 MHz	? MHz	±0	-	0	☐	
System	ILO	DIGITAL	? MHz	1 kHz	-50, +100	-	0	☑	
System	IMO	DIGITAL	3 MHz	3 MHz	±1	-	0	☑	
System	BUS_CLK (CPU)	DIGITAL	? MHz	24 MHz	±1	-	1	☑	MASTER_CLK
System	MASTER_CLK	DIGITAL	? MHz	24 MHz	±1	-	1	☑	PLL_OUT
System	PLL_OUT	DIGITAL	24 MHz	24 MHz	±1	-	0	☑	IMO
Local	Clock_1	DIGITAL	80 MHz	24 MHz	±1	±5	1	☑	Auto: MASTER_C

Figura 22.- Sistemas de Reloj de PSoC.

Fuente: Software PSoC Creator.

Puede generar, dividir y distribuir el reloj en todo el sistema de PSoC, en algunos casos no requiere cristal externo ya que la OMI[31] y PLL[32] puede generar un reloj hasta los 80 MHz. algunos bloques de PSoC pueden generar y distribuir el reloj en forma automática, todo esto se debe a la capacidad que tiene PSoC. Cabe recalcar que el oscilador externo es de 4 a 25 MHz.

Sistema de alimentación.

Este sistema consiste en una alimentación de entrada y salida para analógicos (VDDA[33]) como digitales (VDDD[34]).

[31]Internal Main Oscillator.
[32]Phase-Locked Loop
[33]Voltaje Periféricos Analógico.
[34]Voltaje Periféricos Digitales.

Puertos de entradas y salidas de PSoC.

Los puertos tienen el nombre de GPIO (E/S de propósito general), lo que significa que, no hay pines pre-asignados para las funciones de SPI, UART o ADC. El usuario es libre de elegir las E/S y funciones que desee implementar, son utilizados para conectar al PSoC 5. Tiene más de siete puertos de ocho pines, en total tiene 56 GPIO que son utilizados para E/S analógicas y digitales.

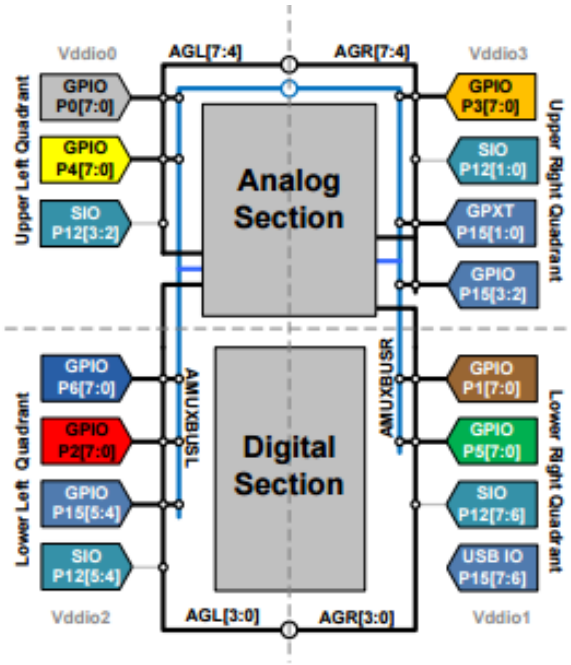


Figura 23.-Puertos E/S de PSoC 5LP.

Fuente: http://www.cypress.com/file/136316/download

Interrupciones GPIO en PSoC.

Proporciona un mecanismo para la respuesta de eventos hardware. Una interrupción es quien detiene la ejecución del programa principal y salta a ejecutar un nuevo código específico donde el **ISR**[35] hace posible la interrupción.

[35]Interrupt Service Routine

Cada bloque de GPIO se puede configurar para que realice una interrupción, se puede utilizar varias interrupciones a la vez, dependiendo de cómo programe el usuario.

Bloques Digital Programable

Estos sistemas crean componentes (periféricos) nuevos según las necesidades del programador, utilizando el software PSoC Creator, los bloques son interconectados entre sí y con cualquier pin en el dispositivo dando así un alto nivel de flexibilidad en el diseño y se dividen en bloques digitales universales.

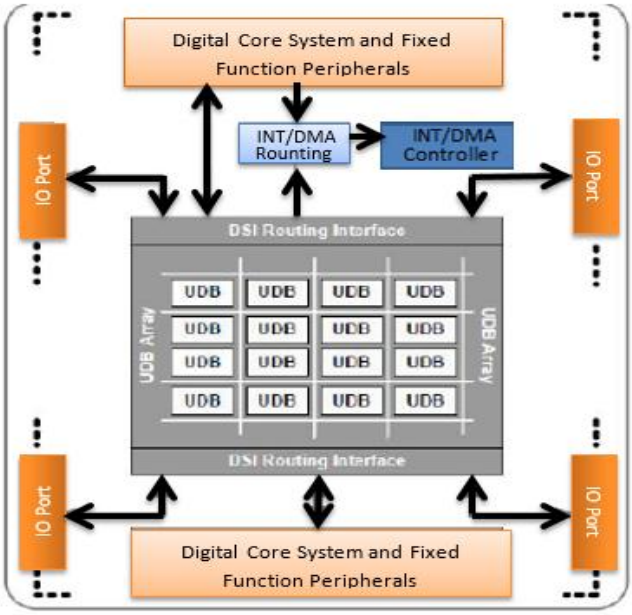


Figura 24.- Diagrama de los UDB.

Fuente: Autores.

Bloques digitales universales (UDB) - Estos forman el núcleo de funcionalidad del sistema programable digital, son una colección de lógica no comprometida (PLD[36]) y la lógica estructural (Datapath) que se pueden utilizar para implementar máquinas de estado, crear tablas de búsqueda, funciones aritméticas, etc.

[36]Programable Logic Device.

Componentes digitales en PSoC.

Estos componentes digitales encontramos en PSoC Creator, las características de estos componentes dependen de la selección de chip es decir el microprocesador y se clasifican en:

- Comunicaciones (I^2C, UART, SPI)
- Funciones (PWM[37], Timer, contadores)
- Filtros
- Lógica (NOT, OR, XOR, AND), LCD[38]

Componentes analógicos en PSoC.

Los componentes analógicos programables de PSoC Creator son:

- Amplificadores operacionales
- Comparadores
- Amplificadores de ganancia programable
- Mezcladores
- LCD de segmento
- Multiplexores analógicos

APIS

Son interfaces de programación que permiten crear librerías para facilitar la programación en el entorno de PSoC Creator. Estas APIS son muy importantes y pueden ser utilizadas por otro software de manera independiente. Estas librerías contienen un conjunto de funciones, procedimientos y métodos que puede llamar desde el programa de desarrollo.

[37]Modulacion por Ancho de Pulso.
[38]Pantalla de Cristal Líquido.

CAPÍTULO II

2. METODOLOGÍA

2.1. TIPO DE ESTUDIO

- Empírica: Se basa en la recolección de datos de algunos diseños de filtros para analizar sus características y así dar solución al problema principal.
- Método científico: Se realizan pasos, técnicas y procedimientos para realizar el algoritmo RLS.
- Metodología de medición: Para determinar la eficiencia del filtro en PSoC y demostrar su correcto funcionamiento.

2.2. POBLACIÓN Y MUESTRA

2.2.1.1. POBLACIÓN.

La población está representada por los diferentes tipos de ruido que pueden existir en la señal de una transmisión de información. En este caso tomaremos en cuenta como ruido a las señales triangulares, cuadráticas, sinodales y señales audibles.

2.2.1.2. MUESTRA

Para determinar el tamaño de la muestra se toma todas las señales que van a ser filtradas para así tener una señal sin ruido.

2.2.1.3. HIPÓTESIS

"Con el diseño y la implementación de un filtro adaptativo en PSOC aplicando el algoritmo RLS se permitirá eliminar el ruido de una señal de audio".

La hipótesis planteada es de tipo descriptiva, ya que involucra dos variables y declara que el diseño e implementado el filtro adaptativo con el algoritmo RLS para audio podrá desarrollar prácticas, eliminando el ruido de una señal de audio.

2.3. OPERACIONALIZACIÓN DE VARIABLES.

Variables	Concepto	Dimensiones	Indicadores	Técnicas e Instrumentos
Dependiente Funcionamiento PSoC	**PSoC** Es un microcontrolador programable, incorpora un sistema configurable dentro de un único chip con funciones analógicas y digitales.	Hardware Software	1.- Procesamiento de envió de datos. 2.- Herramienta de software de programación.	Tarjeta de desarrollo PSoC 5lp Micro C

Independiente Algoritmo	**Algoritmo RLS** Es un algoritmo recursivo de mínimos cuadrados que se usa en filtros adaptativos, son variantes en el tiempo de forma que se adapten a cambios en su entorno.	- Lenguajes de programación	1.-Modelo matemático. 2.-Modelo Programado. .	-Señal recompuesta sin ruido.

2.4. PROCEDIMIENTOS

Para el desarrollo de un filtro adaptativo en PSoC aplicando el algoritmo RLS, se debe seguir una serie de pasos, el cual permitirá cumplir con cada uno de los objetivos planteados, además permite conocer el orden cronológico del diseño y funcionamiento del filtro adaptativo **figura 26**.

2.1. PROCEDIMIENTOS Y ANÁLISIS

2.1.1. Diseño de las placas electrónicas para el Filtro Adaptativo.

2.1.1.1. Diseño de sujetadores

Para las señales de entrada se utilizan dos sujetadores que permiten añadir un nivel de voltaje de corriente continua (C.C) a un voltaje de corriente alterna (C.A), está constituida por dos resistencias y dos capacitores. Los capacitores dividen la fuente AC y DC, bloqueando así el voltaje DC, para el diseño se utiliza el software de Proteus.

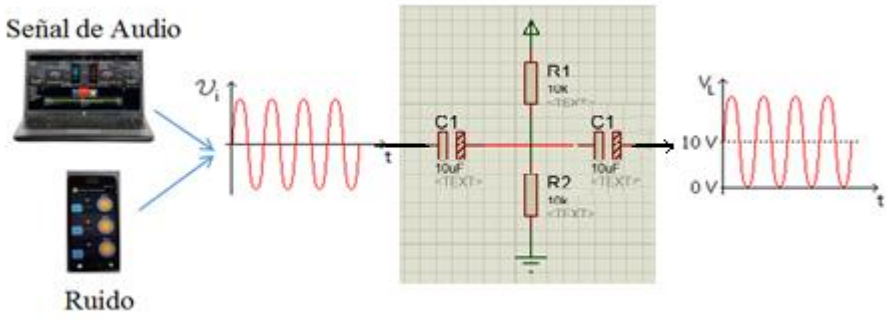


Figura 25.- Sujetador de Voltaje.

Fuente: Autores

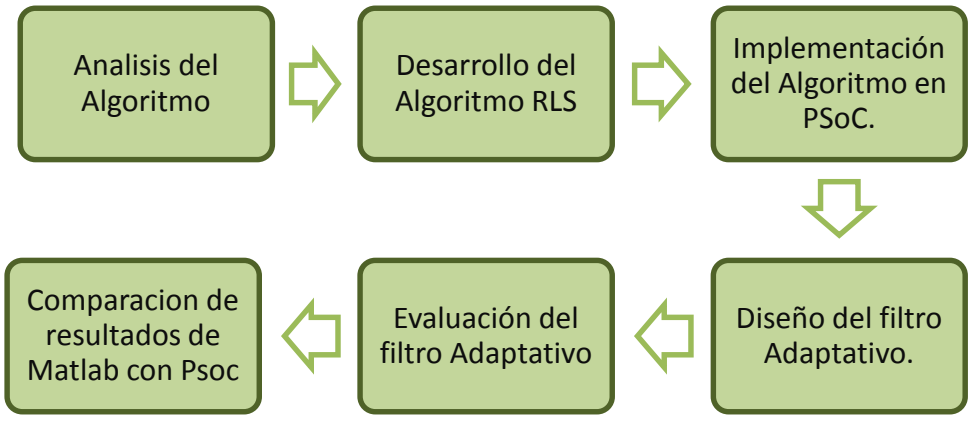


Figura 26.- Esquema del Desarrollo de Filtro Adaptativo.

Fuente: Autores.

2.1.1.2. Diseño de un regulador de voltaje.

Es un circuito electrónico que tiene la capacidad de regular y entregar una cantidad óptima de voltaje 5V.

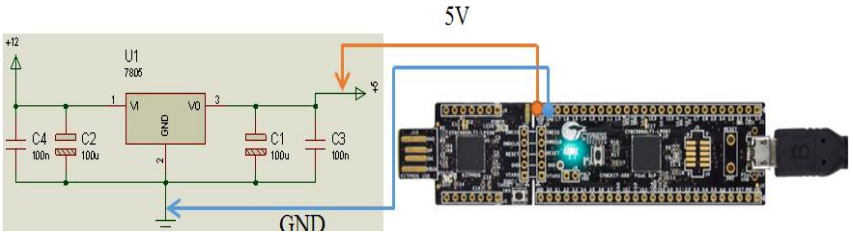


Figura 27.- Regulador de Voltaje.

Fuente: Autores.

2.1.2. Procesamiento del algoritmo RLS.

2.1.2.1. Análisis del algoritmo RLS.

El algoritmo RLS se basa en la inversión de matrices con el fin de proveer una rápida convergencia y cancelar el ruido.

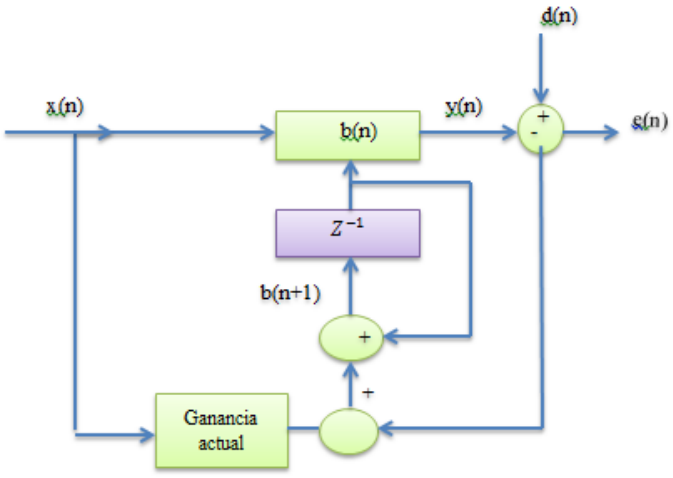


Figura 28.-Diagrama de bloques del Algoritmo RLS.

Fuente: Autores.

En la **figura 28** se observa la capacidad que tiene el algoritmo y cómo se comportan los coeficientes $b(n)$ que son generados por el algoritmo RLS, con estos valores obtendremos el valor de $y(n)$.

$$d(n) = x(n) + v(n) \tag{43}$$

Las señales de entrada $d(n)$ es igual a la suma de la señal de audio más el ruido. La señal de referencia es el ruido la cual es una señal periódica $v(n)$, esta señal de referencia no tiene correlación con la señal de entrada $x(n)$.

$$e(n) = d(n) - y(n) \tag{44}$$

El error es la diferencia entre la señal deseada y la estimada, $n = 1,2, \ldots . M,$ donde n se incrementa en un valor de acuerdo a la longitud del filtro, el tiempo real de procesamiento, para eliminar el error con respecto a los valores de los coeficientes del filtro $b(n)$ proporcionando asi el conjunto de ecuaciones lineales, donde $R(n)$es la matriz de correlación y $P(n)$ es el vector de correlación cruzada.

$$R(n).b_M(n) = P(n) \tag{45}$$

Internamente se va calculando los valores $R(n-1)$y $P(n-1)$ e incorporando el vector de referencia $x(n)$. De esta manera la matriz $R(n)$, $P(n)$se representan recursivamente.

$$R(n) = \lambda.R(n-1)x(n).x^T(n) \tag{46}$$

$$P(n) = \lambda.P(n-1) + d(n).x(n) \tag{47}$$

El factor de ponderación o factor de olvido $\lambda = 1$ da una memoria infinita y si $\lambda < 1$ da más pesos a las muestras más recientes que a las antiguas.

La memoria infinita del algoritmo RLS es quien promedia el valor de cada coeficiente y mejora el rendimiento final de eliminación de ruido.

$$e(n) = d(n) - b^T(n).x(n) \tag{48}$$

$e(n)$ es el error a priori, o innovación, que resulta usar los coeficientes previos $b(n-1)$, es necesaria esta cantidad porque el peso actualizado no está disponible hasta la llegada de la próxima muestra.

$$b(n+1) = b(n) + k(n).e(n) \tag{49}$$

$$k_M(n) = R_M^{-1}(n).x_M(n) \tag{50}$$

El vector de Kalman se puede generar recursivamente.

$$k(n) = \frac{R^{-1}(n-1)f(n)}{\lambda + f^T R^{-1}(n-1)f(n)} \tag{51}$$

El algoritmo RLS, tiene un mejor desempeño en señales senoidales.

2.1.3. Diseño e implantación de un filtro Adaptativo.

2.1.3.1. Implementación del algoritmo RLS en el software de PSoC.

El algoritmo RLS se programa en el software de PSoC para ello se utiliza la herramienta de creación de las APIS, el cual permite implementar el código del Algoritmo RLS en PSoC. Las APIS son librerías que tienen unas colecciones de programas que facilitan la ejecución de funciones relacionadas entre sí, el cual facilita tener una conexión con el algoritmo generado en lenguaje C.

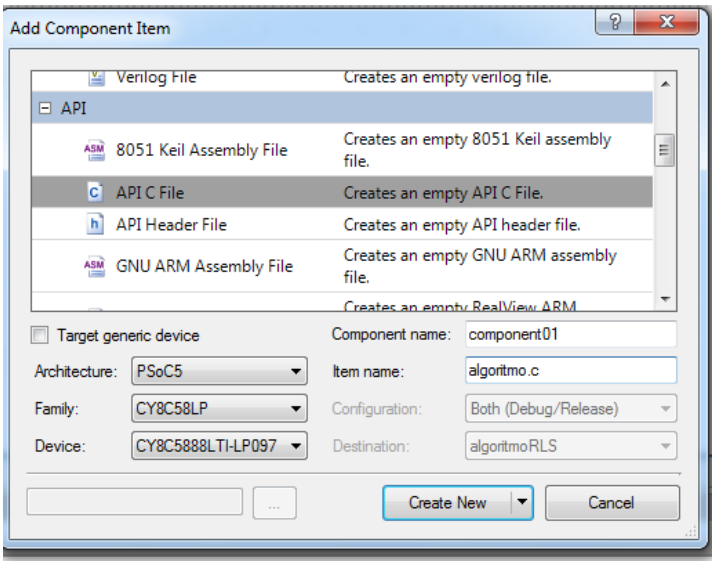


Figura 29.- Generación de APIS.

Fuente: Software PSoC Creator.

Descripción del código RLS en el software PSoC Creator.

La estructura para el desarrollo del algoritmo RLS en el software PSoC Creator corresponde a una serie de pasos como se muestra en la **figura 30**.

Ingreso de señales por `filtro_U.In3` y `filtro_U.In4`.

Los ciclos if son para introducir un índice de tiempo en el vector de coeficientes del filtro y en la secuencia de error.

```
if (filtro_ConstB.Width < 0.0) {
  filtro_B.Sum3 = ceil(filtro_ConstB.Width);
} else {
  filtro_B.Sum3 = floor(filtro_ConstB.Width);
}

if (rtIsNaN(filtro_B.Sum3) || rtIsInf(filtro_B.Sum3)) {
  filtro_B.Sum3 = 0.0;
} else {
  filtro_B.Sum3 = fmod(filtro_B.Sum3, 4.294967296E+9);
}
```

```
      if (s7_iter <= (filtro_B.Sum3 < 0.0 ? -(int32_T)(uint32_T)-
filtro_B.Sum3 :
                      (int32_T)(uint32_T)filtro_B.Sum3)) {
        rtb_Buffer_idx_0 = filtro_DW.Buffer_CircBuff;
        filtro_DW.Buffer_CircBuff = filtro_U.In3;
```

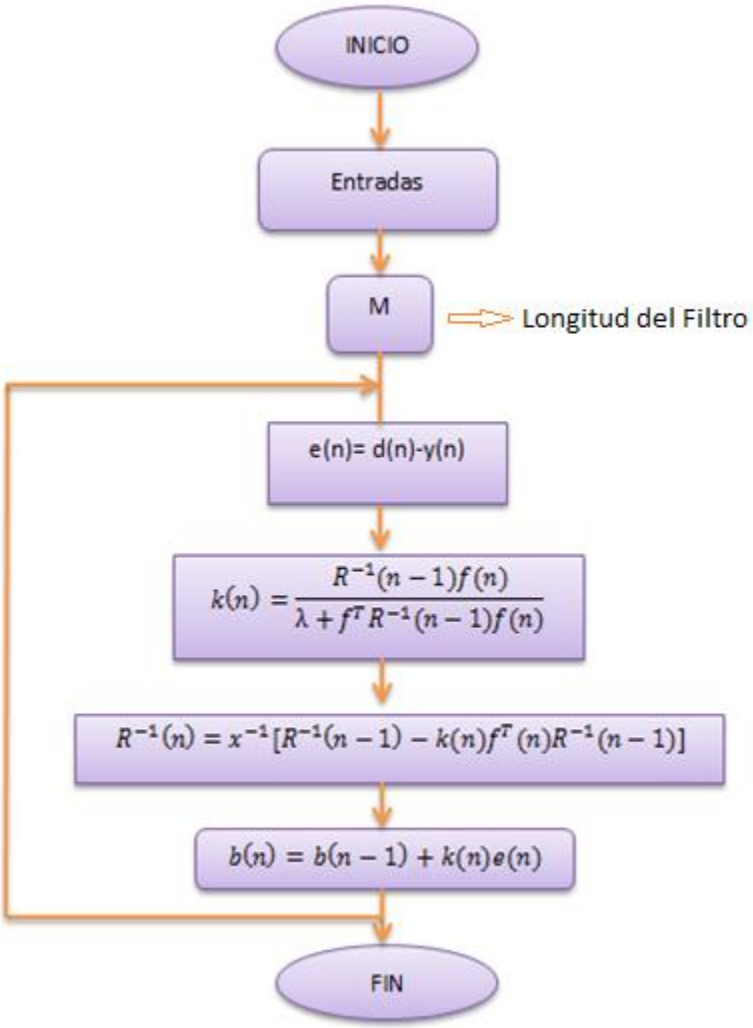


Figura 30.- Diagrama de flujo del RLS.

Fuente: Autores.

La entrada es un vector en un instante n de tiempo, donde la longitud del filtro está dada por M = 2

```
  if (filtro_P.Reset_Value) {
          filtro_DW.FilterTaps_DSTATE[0] =
filtro_P.FilterTaps_InitialCondition;
          filtro_DW.FilterTaps_DSTATE[1] =
filtro_P.FilterTaps_InitialCondition;
        }
```

Coeficientes del filtro que minimiza la suma ponderada del error.

```
  filtro_B.Sum3 = rtb_Buffer_idx_0 * filtro_DW.FilterTaps_DSTATE[0]+

          filtro_U.In3 * filtro_DW.FilterTaps_DSTATE[1];
```

Donde el error se define como la diferencia entre la señal deseada y la estimada

```
  filtro_Y.Out1 = filtro_U.In4 - filtro_B.Sum3;
```

La minimización del error con respecto al vector de coeficientes del filtro que proporciona el conjunto de ecuaciones lineales donde la matriz de correlación (estimada) de la señal se define como:

```
        if (filtro_P.Adapt_Value) {
          if (filtro_P.Reset_Value) {
            filtro_DW.Correlation_DSTATE[0] =
              filtro_P.Correlation_InitialCondition[0];
            filtro_DW.Correlation_DSTATE[1] =
              filtro_P.Correlation_InitialCondition[1];
            filtro_DW.Correlation_DSTATE[2] =
              filtro_P.Correlation_InitialCondition[2];
            filtro_DW.Correlation_DSTATE[3] =
              filtro_P.Correlation_InitialCondition[3];
          }
```

El vector de la ganancia de Kalman se lo representa como:

```
if (filtro_P.Adapt_Value) {
  rtb_Gain_idx_0 = (filtro_DW.Correlation_DSTATE[0] +
                    filtro_DW.Correlation_DSTATE[0]) /
    filtro_P.RLSFilter_lambda * filtro_P.Gain_Gain;
  rtb_Gain_idx_1 = (filtro_DW.Correlation_DSTATE[2] +
                    filtro_DW.Correlation_DSTATE[1]) /
    filtro_P.RLSFilter_lambda * filtro_P.Gain_Gain;
```

```
          rtb_Gain_idx_2 = (filtro_DW.Correlation_DSTATE[1] +
                            filtro_DW.Correlation_DSTATE[2]) /
            filtro_P.RLSFilter_lambda * filtro_P.Gain_Gain;
          rtb_Gain_idx_3 = (filtro_DW.Correlation_DSTATE[3] +
                            filtro_DW.Correlation_DSTATE[3]) /
            filtro_P.RLSFilter_lambda * filtro_P.Gain_Gain;
  }
```

Actualización en el tiempo para $R_M^{-1}(n)$ de forma recursiva.

```
  filtro_DW.Correlation_DSTATE[0] = rtb_Gain_idx_0 -
rtb_Product2_idx_0 *
            rtb_MathFunction_idx_0;
          filtro_DW.Correlation_DSTATE[2] = rtb_Gain_idx_2 -
rtb_Product2_idx_0 *
            filtro_B.rtb_Product2_m;
          filtro_DW.Correlation_DSTATE[1] = rtb_Gain_idx_1 -
filtro_B.Sum3 *
            rtb_MathFunction_idx_0;
          filtro_DW.Correlation_DSTATE[3] = rtb_Gain_idx_3 -
filtro_B.Sum3 *
            filtro_B.rtb_Product2_m;
```

La salida del filtro adaptativo en el instante *n* basada en el uso de los coeficientes del filtro en el instante n-1 se deduce a la ecuación de actualización en el tiempo como:

```
  rtb_Product2_idx_0 = rtb_Gain_idx_0 * rtb_Buffer_idx_0 +
rtb_Gain_idx_2 * filtro_U.In3;
          filtro_B.rtb_Product2_m = rtb_Gain_idx_1 *
rtb_Buffer_idx_0 +
            rtb_Gain_idx_3 * filtro_U.In3;
```

2.1.3.2. Estructura del filtro adaptativo en PSoC.

En la **figura 31** se observa dos señales, una señal de audio más ruido y la señal de ruido que va ser filtrado, estas señales ingresan al ADC de la tarjeta PSoC, en donde se encarga de cuantificar la señal y asignarle un valor que se muestra en formato digital, esta información es procesada con el filtro RLS y su resultado es enviado a un DAC, el cual efectúa la conversión o transformación de la información en

formato digital a análogo, la señal de información resultante tiende a atenuarse durante el proceso de filtrado es por ello que se utiliza un amplificador para tener una mayor ganancia en la salida del filtro adaptativo.

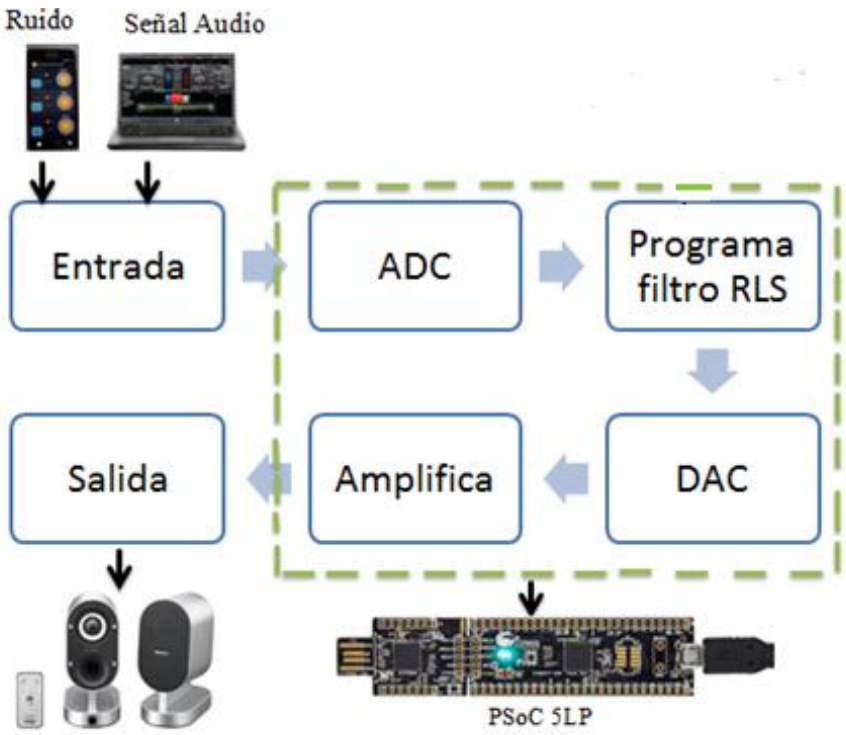


Figura 31.- Diagrama de Bloques de Filtro Adaptativo.

Fuente: Autores.

En la **figura 31** se presenta, de forma general cada una de las etapas necesarias para la implementación del filtro adaptativo.

2.1.3.3. Diseño del filtro Adaptativo.

Para el diseño del filtro se utiliza el software PSoC Creator 3.3, al abrir un nuevo proyecto, se selecciona el tipo de la tarjeta y el procesador, para este caso se selecciona la tarjeta PSoC 5LP y el procesador CY8C5888LTI-LP097. La implementación de un

filtro adaptativo para la cancelación de ruido de una señal de audio se basa en algunos parámetros como son: un filtro adaptativo con el algoritmo RLS, PSoC con su software de desarrollo PSOC Creator.

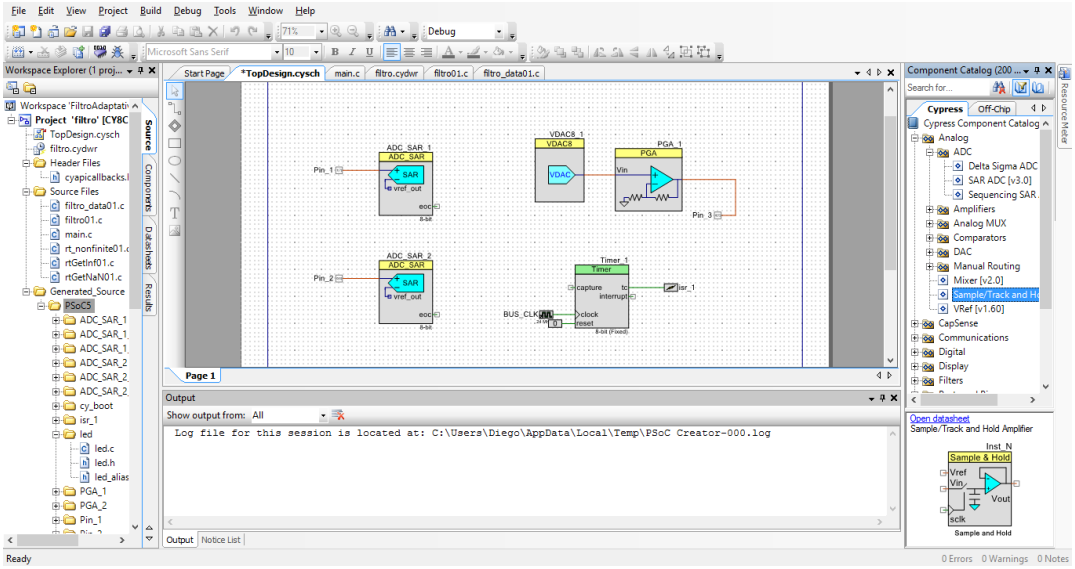


Figura 32.- Diseño de un Filtro Adaptativo en PSoC Creator.

Fuente: Autores.

En la **figura 32** se observa el diseño del filtro adaptativo que está constituido por varios bloques como son: dos ADC_SAR, Timer, una interrupción (isr), un reloj (CLK), un DAC y un amplificador, para cumplir con el diseño se procedió a realizar los siguientes pasos:

Configuración y Programación de un ADC_SAR.

El diseño del filtro Adaptativo consta de dos bloques analógicos de tipo ADC_SAR que serán para las señales de entrada, estos bloques tienen una resolución seleccionable (8,10 o 12 bits), con una frecuencia de muestreo máximo de 1.6 Mbps.

La **figura 33** indica que el pin1 es para la señal de audio más ruido y el pin2 es para

la señal de error.

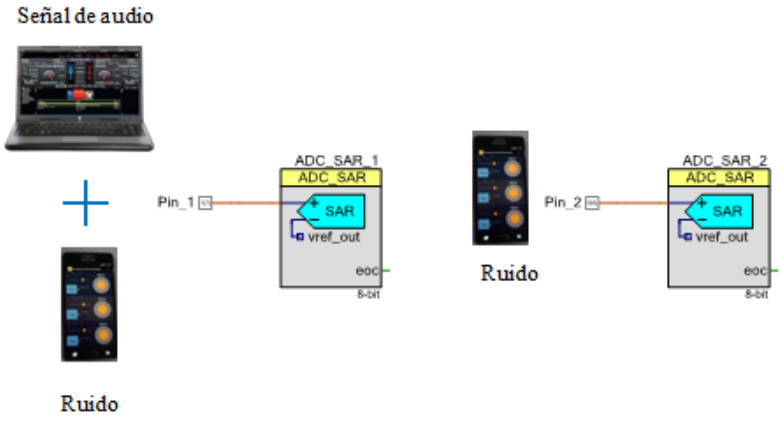


Figura 33.- Conexión del ADC_SAR con las señales de entrada.

Fuente: Autores.

En la **figura 34** indica la configuración del ADC_SAR con una resolución de 8 bits, en **input range**, un VDDA, que tendrá un valor máximo de 5V como voltaje de referencia. La referencia negativa seleccionada internamente permite tener una mayor frecuencia de reloj.

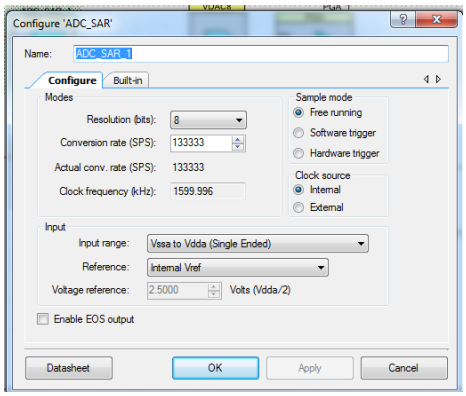


Figura 34.- Configuración del ADC_SAR.

Fuente: Software PSoC Creator.

En la **figura 35**, se puede observar la programación de los dos ADC_SAR.

```
70
71      ADC_SAR_1_Start();
72      ADC_SAR_1_StartConvert();
73      ADC_SAR_2_Start();
74      ADC_SAR_2_StartConvert();
```

Figura 35.- Configuración de los ADC_SAR.

Fuente: Software PSoC Creator.

Configuración y Programación del Timer.

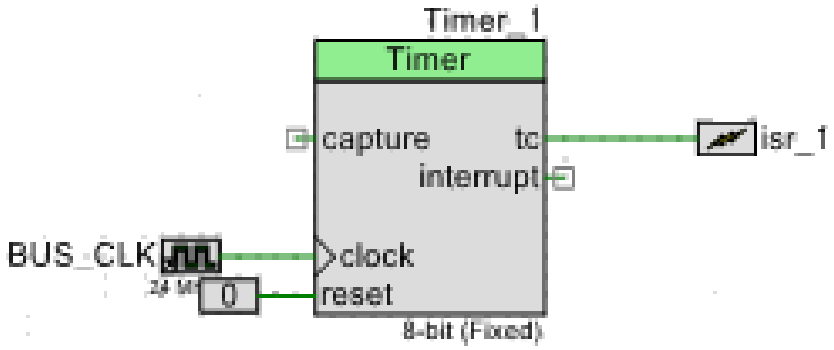


Figura 36.- Conexión del Timer.

Fuente: Software PSoC Creator.

En el diseño del filtro adaptativo consta de un Timer que permite activar las interrupciones cada cierto periodo de tiempo.

En la **figura 37**, se puede observar que el Timer está configurado a una resolución de 8 bit con un periodo de 10us lo cual permite tener una frecuencia de muestreo de 100 KHz, en la **figura 38** muestra el código para activa el Timer y la interrupción.

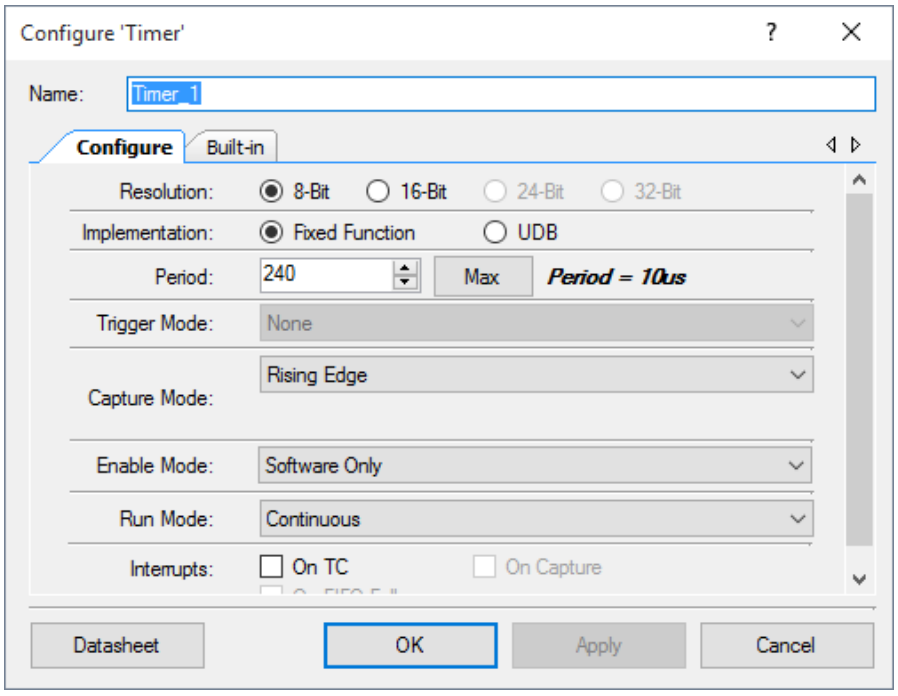


Figura37.- Configuración del Timer.

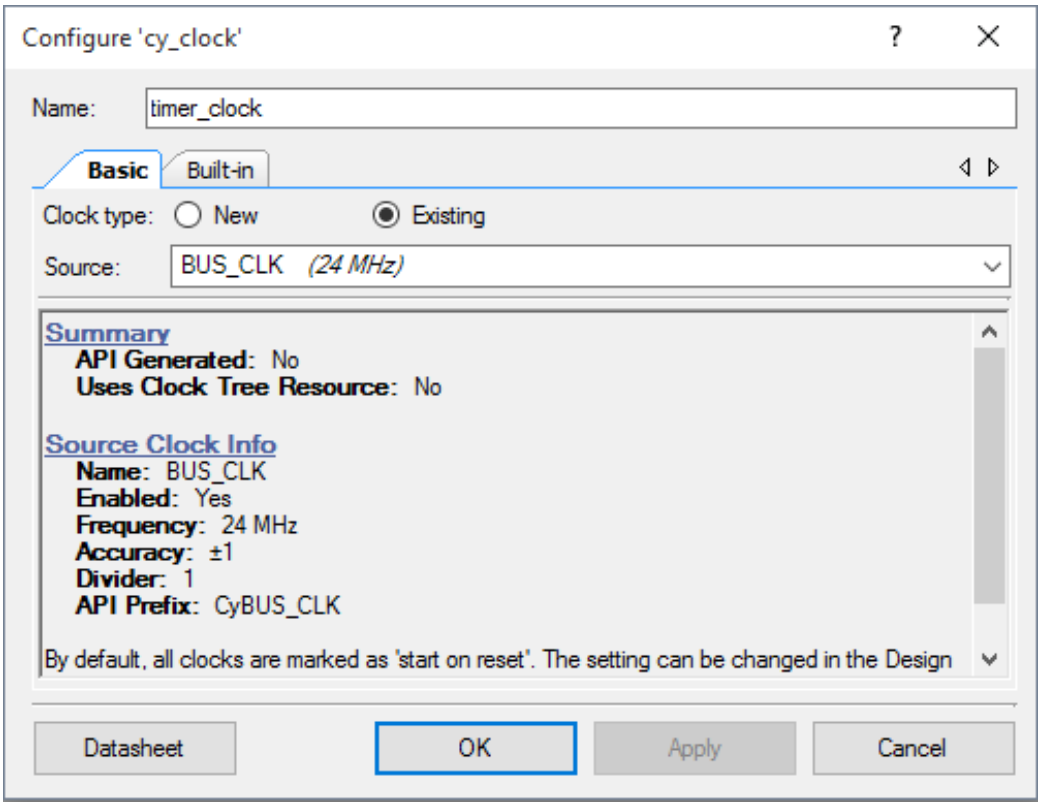
Fuente: Software PSoC Creator.

```
76
77
78
79      Timer_1_Start();
80      isr_1_StartEx(filtroVDAC);
81
82
83      CyGlobalIntEnable; /* Enable global interrupts. */
84
```

Figura 38.- Programación de la Interrupción y Timer.

Fuente: Software PSoC Creator.

Nota: Cabe mencionar que para el funcionamiento del Timer es necesario un reloj y su configuración se dejó con uno ya existente en PSoC como muestra la **figura 39** para así aprovechar la frecuencia del BUS_CLK que es de 24 MHZ, evitando cargar con más procesos al microcontrolador.



Fuente 39.- Configuración del Reloj.

Fuente: Software PSoC Creator.

Configuración de la interrupción.

Con respecto a la programación del bloque de interrupción, **figura 38**, se utiliza un solo comando para activar la interrupción. En la **figura 40**, se observa cómo está configurada la interrupción.

Configuración y programación de VDAC 8.

Para la conversión de datos de digital- analógica, se utiliza un bloque VDAC, este bloque recompondrá la señal filtrada, con una resolución de 8bits y un rango de voltaje de 500 mV, como se observa en la **figura 41**.

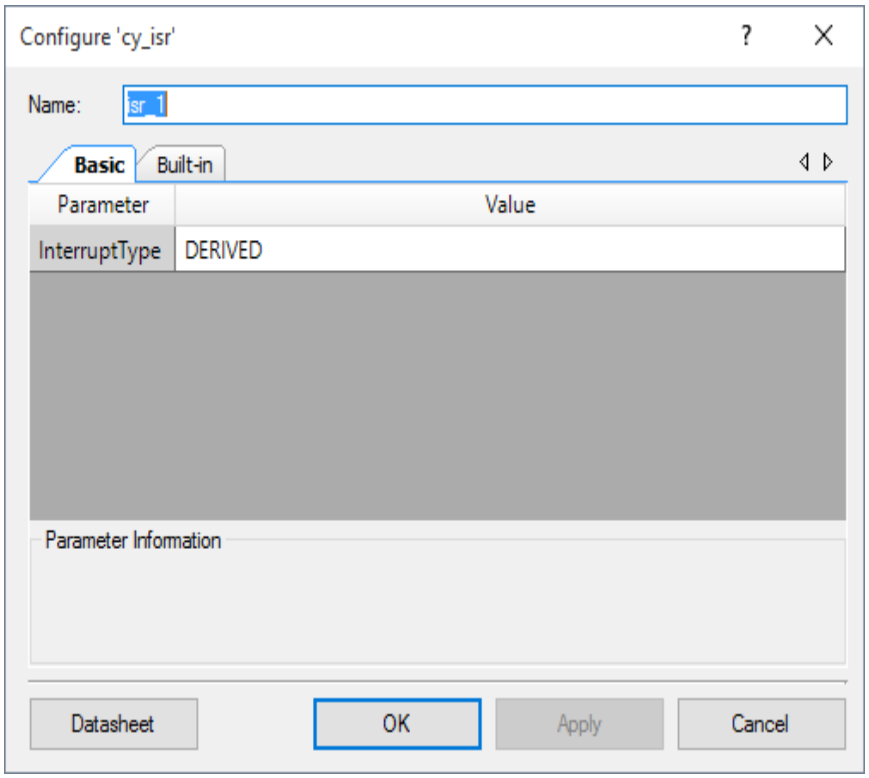


Figura 40 Configuración de la interrupción.

Fuente: Software PSoC Creator.

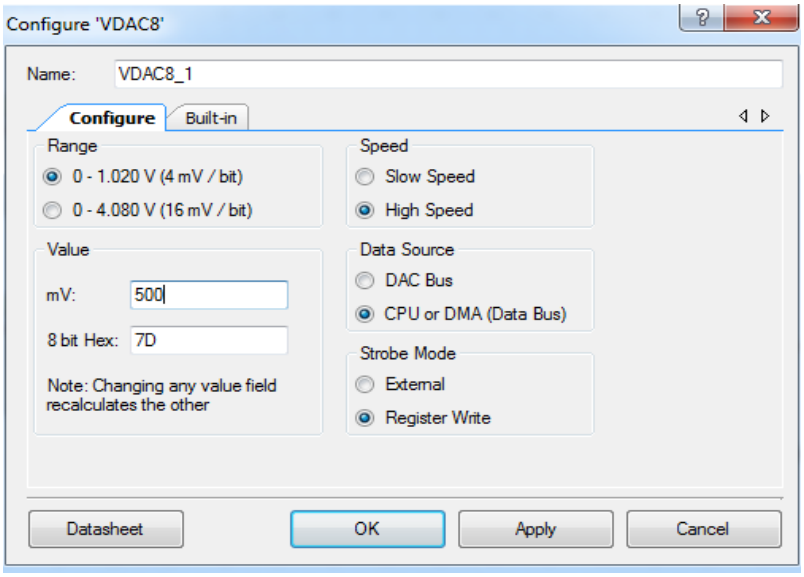


Figura 41 Configuración de VDAC.

Fuente: Software PSoC Creator.

Se debe tener en cuenta, en una conversión digital analógico o DAC, el número de bits seleccionados debe ser el mayor posible para tener una buena recomposición de la señal, en este caso se seleccionó 8 bits, debido a que entre mayor sea el número de bits menor es la frecuencia de reloj, limitando la recomposición de la señal.

Con la siguiente formula se obtiene la resolución de la señal para su recomposición.

$$Resoluciòn = \frac{V}{2^n - 1}$$

$$Resoluciòn = \frac{0.5\,V}{2^8 - 1}$$

$$Resoluciòn = \frac{0.5V}{255}$$

$$Resoluciòn = 1{,}9\,mV$$

Al aplicar la formula la resolución es de 1,9 mV para la recomposición de la señal. La programación del bloque VDAC se puede observar en la **figura 42**, a los bloques de VDAC se debe incluir la librería específica de los VDAC e inicializar por medio del comando VDAC8_Start.

```
13  //#include <DVDAC_1.h>
14  #include <VDAC8_1.h>
15  #include <VDAC8_2.h>
```

```
78      VDAC8_1_Start();
79      VDAC8_2_Start();
80      //Timer_1_Start();/////////////
81      isr_1_StartEx(filtroVDAC);
```

Figura 42 Inicialización del VDAC.

Fuente: Software PSoC Creator.

La suma de un valor de 128, **figura 43**, sirve para la reconstrucción de la señal, como mencionamos anteriormente, el número de bits para este diseño, es de 8 bits en donde 2^8=256 dividimos para 2 obteniendo así un valor resultante de 128.

```
60
61      //DVDAC_1_SetValue(filtro_Y.Out2+52);
62      VDAC8_1_SetValue(filtro_Y.Out2+128);
63      VDAC8_2_SetValue(filtro_Y.Out3+128);
64  }
```

Figura 43.- Programación de los VDAC.

Fuente: Software PSoC Creator..

Hay que tener en cuenta que mientras menor sea la resolución, mejor será la recomposición de la señal, al utilizar un número mayor de bits en el DAC genera más ruido, es por ello se utiliza un rango de voltaje 0.5V para la recomposición de la señal.

Configuración y Programación de un Amplificador.

El bloque está configurado con una ganancia de cuatro, para amplificar la señal de salida de baja potencia, en la **figura 44** se observa su configuración.

2.1.1. Inicialización del programa del filtro Adaptativo con el Algoritmo RLS.

2.1.1.1. Descripción del código en PSoC Creator.

Los <**# include**> permiten manipular a los ficheros del programa en C y se deben incluir al inicio del código. Como se observa en el código siguiente se incluye a los

bloques y a las APIS que se generó a un inicio.

```
#include <filtro.h>
#include <filtro_private.h>
#include <project.h>
#include <VDAC8_1.h>
#include <filtro.h>
#include <rtwtypes.h>
```

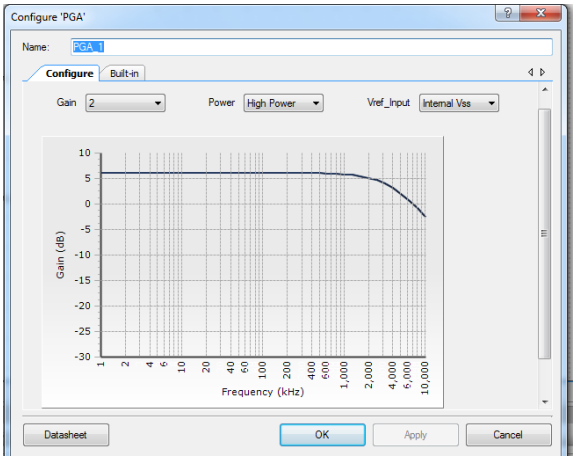


Figura 44 .-Configuración del Amplificador.

Fuente: Software PSoC Creator.

El comando **#define** permite definir varias macros (controlan y definen expresiones en el código) y realizar modificaciones en el texto del código.

```
#define REQUEST_PER_BURST        (1u)
#define BYTES_PER_BURST          (1u)
#define UPPER_SRC_ADDRESS
CYDEV_PERIPH_BASE
#define UPPER_DEST_ADDRESS
CYDEV_PERIPH_BASE
```

Para las variables tenemos de tipo entero de 32 bits y de real_T, permite devolver la parte real de un número complejo.

```
int32_T s7_iter;
real_T rtb_Product2;
real_T rtb_kn;
real_T rtb_Gain_e;
real_T rtb_MathFunction;
int32_T exitg1;
```

Estas líneas de código generan un llamado a la API del filtro RLS.

```
(void) memset(((void *) &filtro_B), 0,
              sizeof(B_filtro_T));

(void) memset((void *)&filtro_DW, 0,
              sizeof(DW_filtro_T));

volatile int IsrOverrun = 0;
static boolean_T OverrunFlag = 0;

void rt_OneStep(void)
{

  if (OverrunFlag++) {
    IsrOverrun = 1;
    OverrunFlag--;
    return;
  }
```

Ingresan las señales por las variables, **fiter_U.Int3** que es la señal de audio + ruido y **filter_U. Int4** es el ruido y la salida es **filter_Y.Out1**. Todo esto está incluido dentro de una interrupción, con un periodo de 10us que da una frecuencia de muestreo de 100KHz.

```
CY_ISR(filtroVDAC)
{
  rt_OneStep();
  filtro_U.In3 = ADC_SAR_1_GetResult8();
  filtro_U.In4 = ADC_SAR_2_GetResult8();

    VDAC8_1_SetValue(filtro_Y.Out1+128);
 }
```

Inicio de cada uno de los bloques que están constituidos para el diseño del filtro Adaptativo con el algoritmo RLS.

```
ADC_SAR_1_Start();
ADC_SAR_1_StartConvert();
ADC_SAR_2_Start();
ADC_SAR_2_StartConvert();

PGA_1_Start();
PGA_2_Start();
VDAC8_1_Start();
Timer_1_star();
isr_1_StartEx(filtroVDAC);
```

2.1.2. Diseño final del filtro adaptativo con el algoritmo RLS en PSoC.

El desarrollo del programa final y el esquema del circuito véase en el anexo 2 y 3.

2.1.3. Diseño del filtro Adaptativo con Simulink-Matlab.

El software Simulink-Matlab es una plataforma con lenguaje de programación gráfico que permite la manipulación del filtro adaptativo programado para evaluar la eficiencia del filtro. En la **figura 45** se observa la conexión del bloque del filtro adaptativo RLS con las señales de entrada y salida, el cual facilita la comprobación de resultados con PSoC.

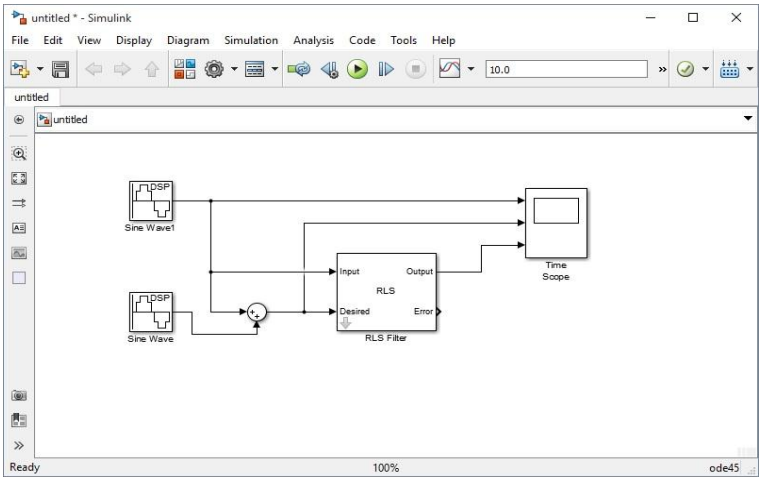


Figura 45.- Diseño de Filtro Adaptativo RLS en Simulink.

Fuente: Autores, Simulink-Matlab.

2.2. COMPROBACIÓN DE LA HIPÓTESIS.

La comprobación de la hipótesis se realizó mediante el método estadístico CHI-CUADRADO, para constatar el cumplimiento de la hipótesis.

2.2.1. PLANTEAMIENTO DE LA HIPÓTESIS

Hipótesis Nula (Ho): Con el diseño e implementación de un filtro adaptativo en PSoC, aplicando el Algoritmo RLS, no eliminara el ruido de una señal de audio.

Hipótesis Alternativa (H1): Con el diseño e implementación de un filtro adaptativo en PSoC, aplicando el Algoritmo RLS permitirá eliminar el ruido de una señal de audio.

2.2.2. PLANTEAMIENTO DE LA HIPÓTESIS ESTADÍSTICA.

Las pruebas se las realizaron con un 95% de confiabilidad, es decir se trabajó con un nivel de significancia de $\propto = 0{,}05$.

2.2.3. DETERMINACIÓN DEL VALOR ESTADÍSTICO DE PRUEBA.

Si el valor del CHI-CUADRADO es menor o igual que el CHI-CUADRADO crítico entonces se acepta la hipótesis nula, caso contrario se la rechaza.

$$X^2 \leq Valor\ Crítico$$

Para aceptar o rechazar esta hipótesis se tomaron en cuenta la comparación del filtro Adaptativo en la tarjeta PSoC con el bloque del filtro adaptativo de Simulink-Matlab.

En la **tabla 2** se muestra los valores obtenidos en la comparación de estos dos sistemas tanto de Simulink con de la tarjeta PSoC.

	RUIDO	ERROR	SALIDA	TOTAL
SIMULINK-MATLAB	0,4262599	-0,0024567	-0.0049838	0,4760988
TARJETA PSoC	0,4396563	-0.0032435	-0,0028989	0,4335139
TOTAL	0,8659162	-0,0032435	-0,0078828	0.8547899

Tabla 2.- Comparación de Tarjeta de PSoC y Matlab.

Fuente: Autores.

Para obtener las frecuencias esperadas se multiplica el total de cada columna, por el total de cada la fila y se divide entre el total de cada fila, y columna, como se puede observar en la **tabla 4**.

	RUIDO	ERROR	SALIDA
MATLAB	0,4262599	-0,0013797	-0,0029524
TARJETA PSoC	0,4396563	-0,0013312	-0,0021839
TOTAL	0,8659162	-0,0030558	-0,0030753

Tabla 3.- Frecuencias esperadas de Matlab y PSoC.

Fuente: Autores.

Para obtener los grados de libertad se hace una diferencia con el número de fila menos uno por el número de columnas menos uno, obteniendo así un numero dos de grado de libertad, como podemos observar en la **tabla 5**.

En la **tabla 4** indica los valores obtenidos para nuestra comprobación de la hipótesis a través del método **CHI-CUADRADO**.

GRADOS DE LIBERTAD	PROBABILIDADES			
	0,995	0,990	0,975	0,950
1	0,0000	0,0000	0,0010	0,0039
2	0,0100	0,0201	0,0506	0,1026
3	0,0717	0,1148	0,2158	0,3518
4	0,2070	0,2971	0,4844	0,7107
5	0,4118	0,5543	0,8313	1,1435
6	0,6757	0,8721	1,2373	2,1674
7	0,9893	1,2390	1,6899	2,7326
8	1,3444	1,6465	2,1797	3,3251
9	1,7349	2,0879	2,7004	3,9403
10	2,1558	2,5582	3,2470	4,5748
11	2,6032	3,0535	3,8157	5,2260

Tabla 4.- Valores Críticos Método Chi-Cuadrado.

Fuente: Autores.

Numero de fila	2
Número de columna	3
X^2	0,30753003
Gados de Libertad	2
Nivel de Significancia	0,05
Probabilidad	0,95

VALOR CRÍTICO	0,103

Tabla 5.- Resultados del Método Estadístico del CHI-CUADRADO.

Fuente: Autores.

Al aplicar la fórmula de **CHI-CUADRADO** da un valor de 0,30753. En la **tabla 4** se muestra los valores obtenidos en cada uno de los escenarios donde se realizaron las pruebas.

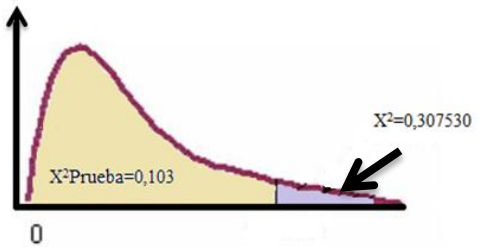


Figura 46.- Resultado de Comparación X^2prueba con X^2tabla

Fuente: Autores.

En la **figura 46** indica la prueba del CHI-CUADRADO, donde es la comparación del valor crítico (X^2 tabla) con X^2(X^2 prueba).

Con el resultado obtenido X^2 donde es mayor que el valor crítico, esta diferencia indica que se rechaza la hipótesis nula y se acepta la hipótesis alternativa que es:

"Hipótesis Alternativa (H1)": Con el diseño e implementación de un filtro adaptativo en PSoC, aplicando el Algoritmo RLS permitirá eliminar el ruido de una señal de audio.

CAPITULO III

3. RESULTADO

A fin de comprobar el funcionamiento del filtro adaptativo aplicando el algoritmo RLS para la cancelación de ruido, se tomó tres escenarios: en el primer escenario se utilizó una señal de entrada (audio) a la cual se le sumo una señal periódica como ruido, en el segundo escenario, es realizar la misma suma, pero con una señal de audio como ruido y en el tercer escenario se tomó a la señal periódica como señal de entrada, se utilizó al audio como ruido.

ESCENARIO 1.

En este escenario se realiza la suma de una señal de audio más la señal periódica (como ruido), **figura 47**, la señal periódica tiene una frecuencia de 1,5 KHz, una amplitud de 800 mV pico a pico y una frecuencia de muestreo de 100 KHz.

En la **figura 48** se puede observar la señal obtenida ya aplicando el filtro adaptativo con el algoritmo RLS.

ESCENARIO 2

En este escenario, **figura 49**, se realiza la suma de una señal de audio más otra señal de audio que ingresa como ruido, con una frecuencia de muestreo de 100 KHz.

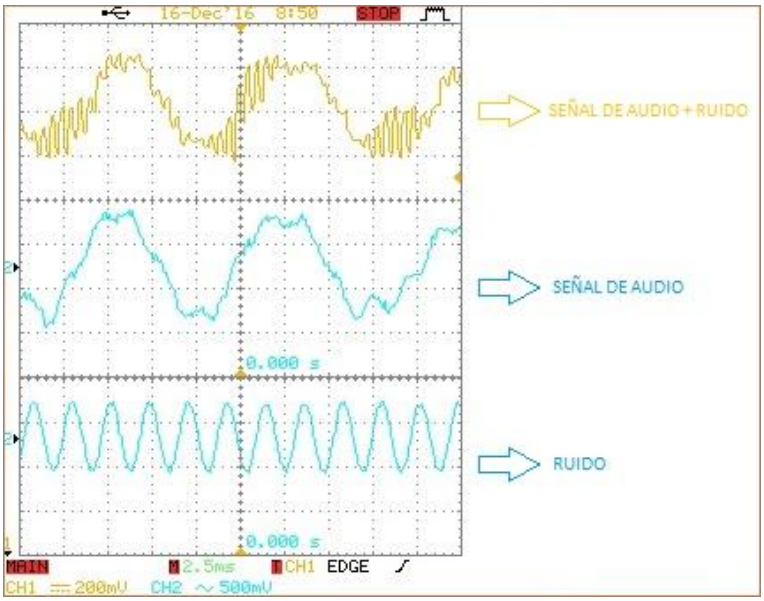


Figura 47.-Ruido, Señal de Audio, Sumatoria de las dos Señales.

Fuentes: Autores - Osciloscopio.

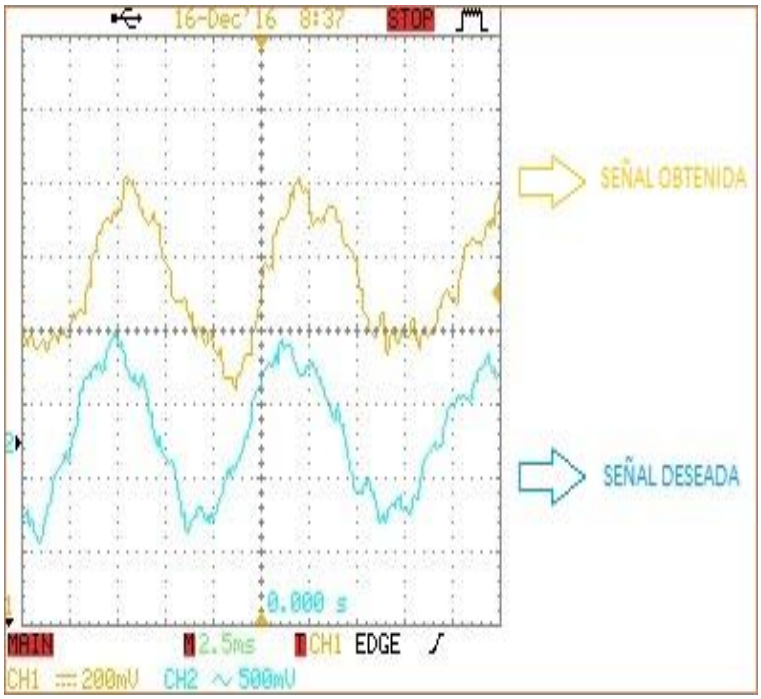


Figura 48.- Señal Obtenida, Señal Deseada.

Fuentes: Autores - Osciloscopio.

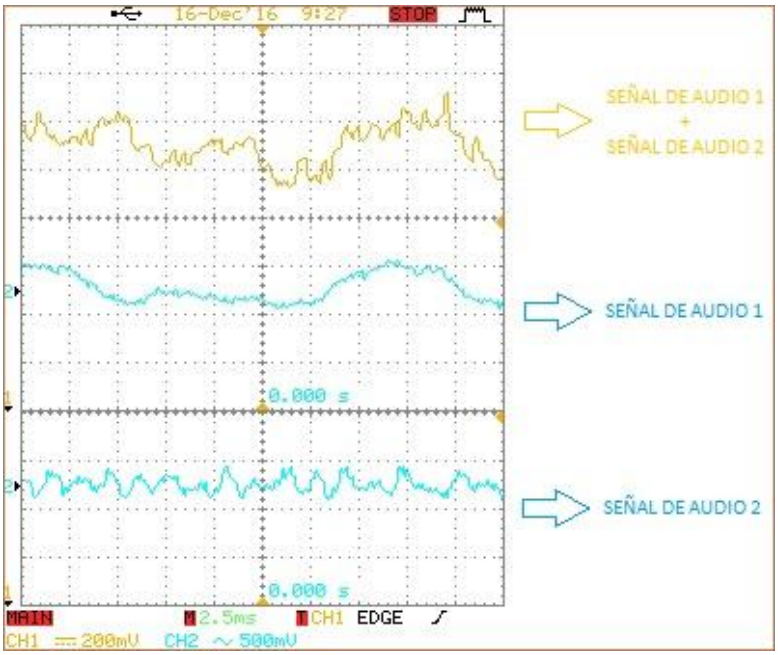


Figura 49.- Ruido (Señal de Audio 1), Señal de Audio 2, Suma de las dos señales.

Fuentes: Autores - Osciloscopio.

En esta **figura 50** se observa a señal obtenida aplicando la señal de audio como ruido.

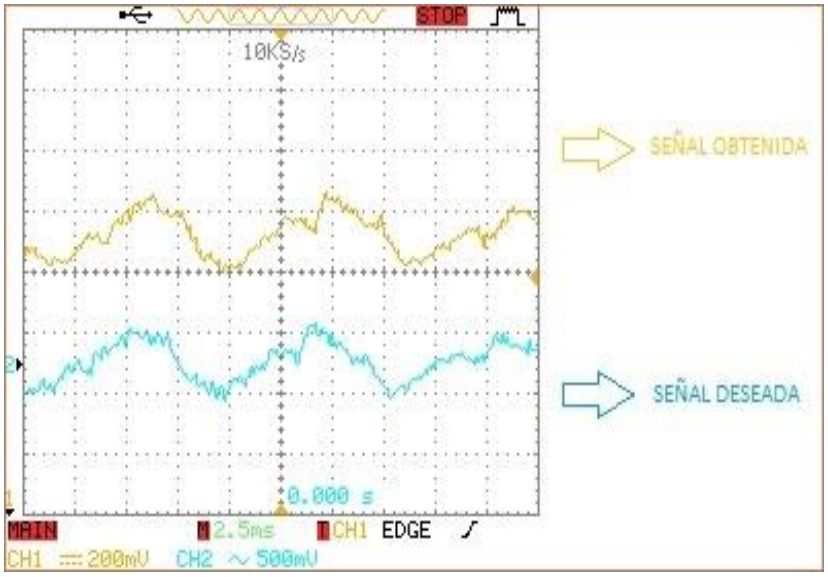


Figura 50.- Señal Obtenida ingresando a la señal de audio como Ruido.

Fuentes: Autores - Osciloscopio.

ESCENARIO 3.

En este escenario, **figura 51**, se realiza la suma de una señal periódica a una frecuencia de 1,5 KHz y una amplitud de 800 mV pico a pico más una señal de audio (como ruido), con una frecuencia de muestreo de 100 KHz.

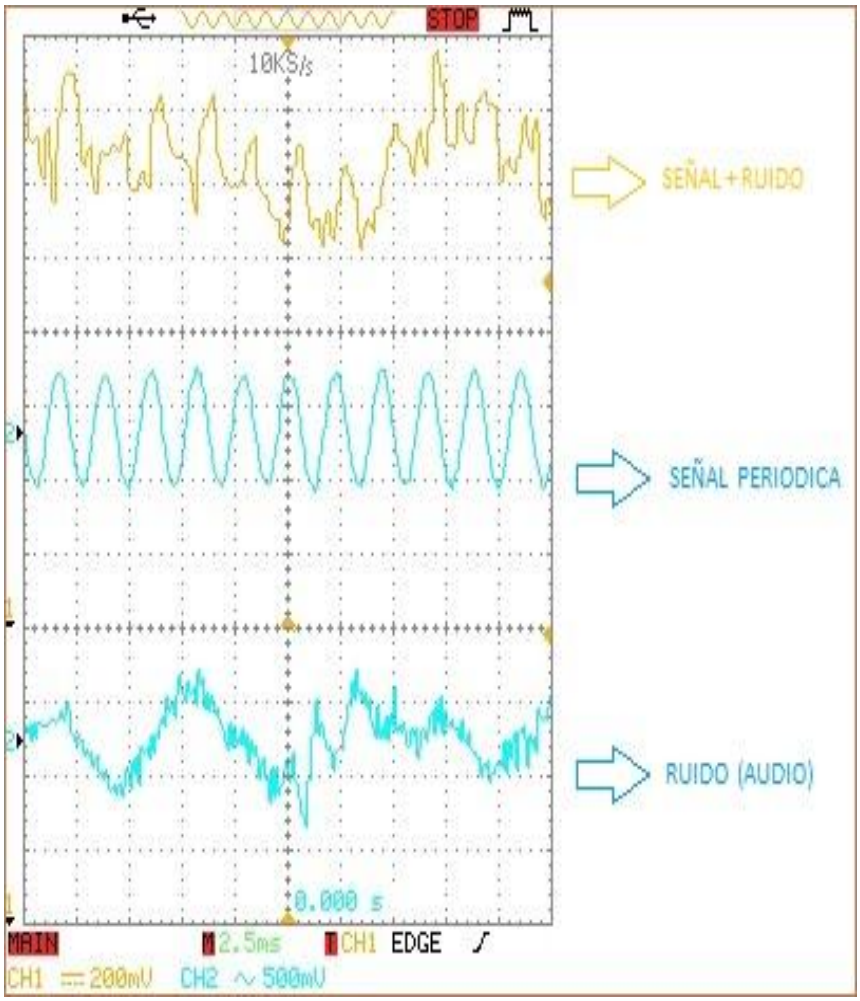


Figura 51.- Ruido (Audio), Señal Periódica, Sumatoria de las dos señales.

Fuentes: Autores - Osciloscopio.

La señal obtenida del Filtro Adaptativo con el algoritmo RLS se muestra en la **figura 52**.

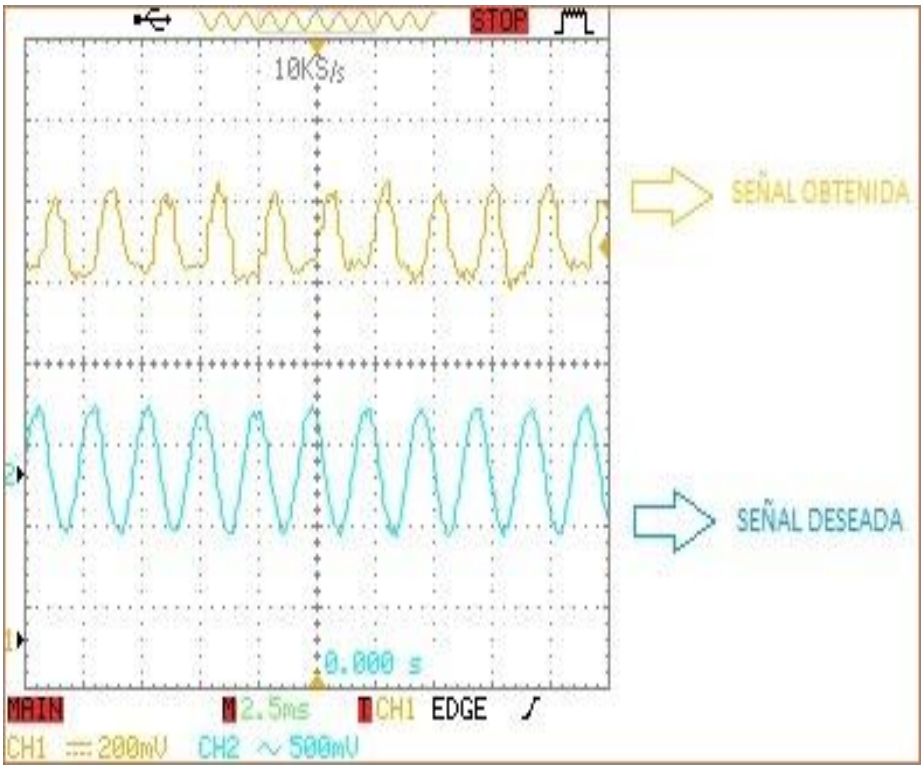


Figura 52.- Señal Obtenida, ingresando como Ruido al Audio.

Fuente: Autores - Osciloscopio.

En los tres escenarios se observa una atenuación del ruido en la salida del filtro adaptativo con el algoritmo RLS y a su vez se tiene como resultado una señal con menor amplitud, pero con una forma semejante a la señal deseada.

3.1.1. Comparación del filtro Adaptativo de PSoC con Simulink-Matlab.

Para la comparación se utilizó señales periódicas tanto para la generación de ruido como para la señal de entrada, donde la señal de entrada tiene una frecuencia de 5.5Hz con una amplitud de 800mV y la señal de ruido es de 20Hz con una amplitud de 500mV las características que se ingresó en cada uno de los filtros son:

- Orden de filtro $= 2$

- Factor de olvido $\lambda = 0.9999$

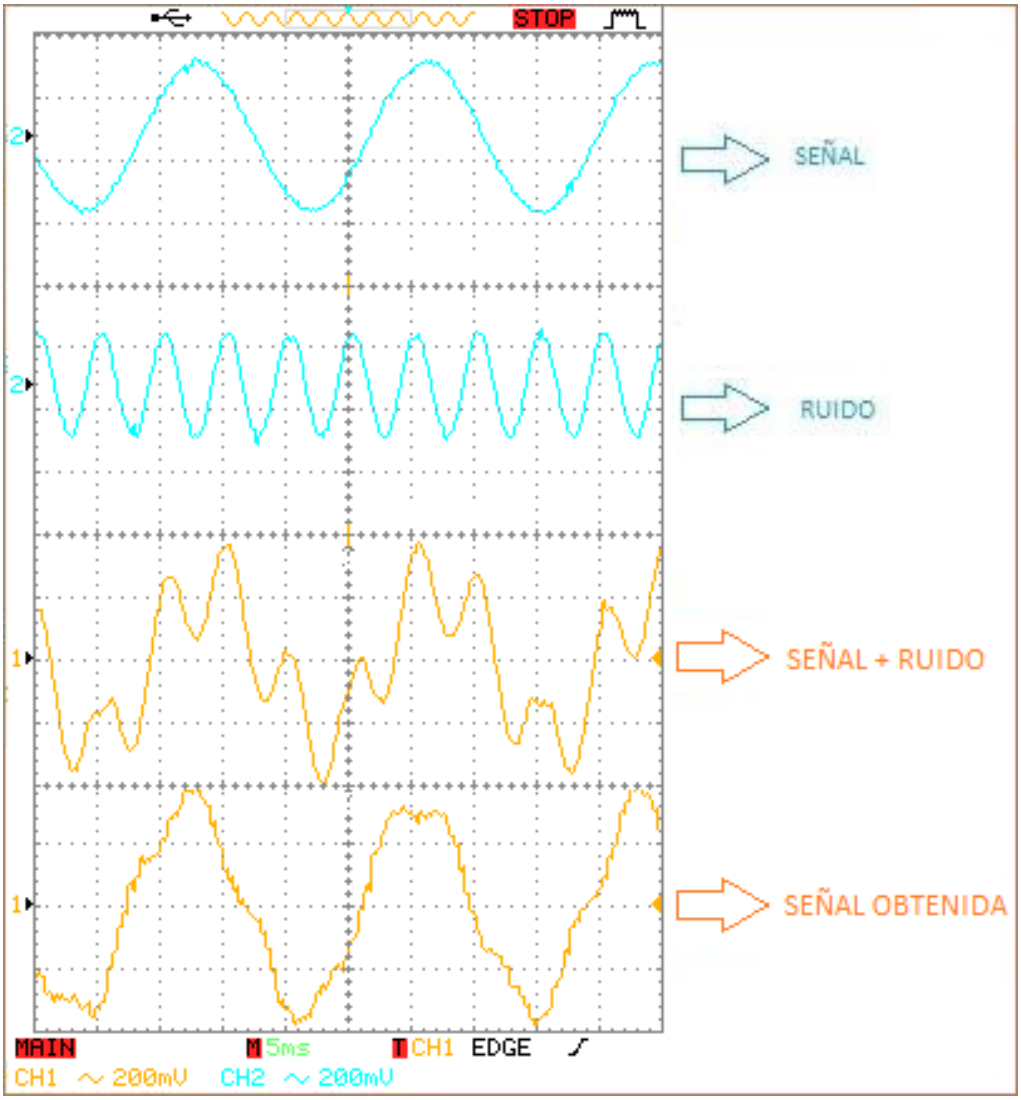


Figura 53.-Suma de señales Periódicas en PSoC

Fuente: Autores - Osciloscopio.

Para la comparación del filtro Adaptativo en PSoC con el Bloque de Simulink-Matlab se ingresó las características con las que se encuentra las señales de entrada de la tarjeta PSoC. En la **figura 53** podemos observar la señal deseada, la señal más ruido

y la señal filtrada.

Como se observa en la **figura 53** la señal de salida en Psoc tiene una adecuada atenuación del ruido y una variación de la forma de la señal a diferencia del diseño de Simulink-Matlab **figura 54** en la cual se observa una recomposición casi perfecta, ya que los cálculos matemáticos se los realiza en un procesador de mayor capacidad de cálculo.

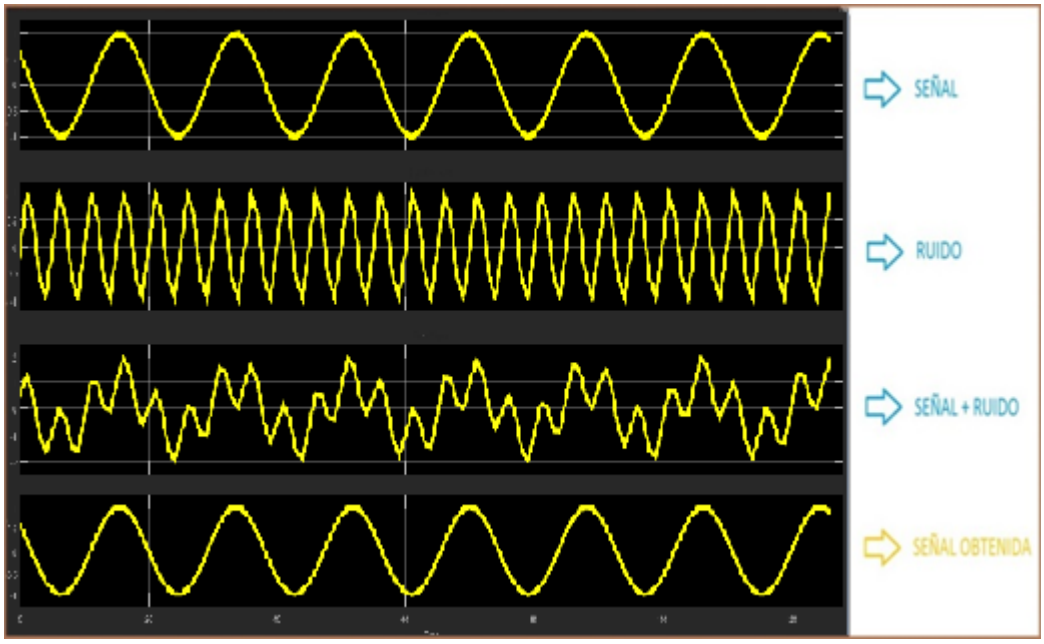


Figura 54.- Señales de entrada, ruido y salida, generadas por Matlab.

Fuente: Autores- Matlab.

CAPÍTULO IV

4. DISCUSIÓN

Los filtros adaptativos son sistemas que están basados en algoritmos matemáticos, en donde ajustan sus variables de forma automática para obtener una señal deseada, estos filtros pueden ser implementados por software, donde sus variables son señales digitales de diferentes características, pueden sustraer el ruido de una señal empleando la técnica del filtrado adaptativo con el algoritmo RLS, obteniendo así una señal casi perfecta, sin ruido. El ruido son señales que perturban la transmisión y procesamiento de la señal de un sistema de comunicación, en donde el ruido se vuelve un problema creciente que afectan a las personas y eliminarlo o controlarlo es un reto tecnológico por su complejidad temporal, frecuencial y espacial que presenta. El algoritmo RLS es quien disminuye el valor cuadrático de la señal de error para alcanzar la convergencia y así una adaptación. Además, cuenta con un factor de olvido comprendido entre cero y uno, que permite dar importancia a las muestras más recientes, bridando al filtro la capacidad de olvidad el pasado y adaptarse a los cambios de los escenarios aplicados. Existen muchas aplicaciones para el modelo de filtros Adaptativos. Uno de los más importantes y utilizados en el desarrollo de la tesis es la cancelación de ruido adaptativo, quien sustrae el ruido de una señal, empleando la técnica de filtrado adaptativo, obteniendo así una señal casi perfecta.

CAPÍTULO V

5. CONCLUSIONES Y RECOMENDACIONES

5.1. Conclusiones

- El algoritmo RLS eliminara el ruido de la señal de audio con un procesador de mejores características que el ARM cortex-M3 en velocidad de procesamiento y tratamiento de valores de punto flotante.
- El algoritmo RLS es una excelente forma de filtrar una señal de error usando una señal deseada d(n) como modelo.
- Las ecuaciones del filtro Adaptativo, requiere de un costo computacional elevado y una velocidad de cálculo muy alta, debido a los cálculos de la matriz de auto-correlación inversa P(n) y el vector de ganancia de Kalman k(n).
- Mientras más cerca este a la unidad el factor de olvido (λ), se obtendrá una gran eliminación de ruido o un mejor seguimiento de la variación de los parámetros.
- La desventaja del filtro adaptativo con el algoritmo RLS implementado se refleja en la alta velocidad de convergencia que requiere para la estimación de los parámetros lo cual aumenta la complejidad computacional para las operaciones matriciales que realiza.
- Cuando el factor de olvido $\lambda<1$ el algoritmo olvida las medidas más antiguas

y hace que el algoritmo tenga una memoria limitada, mientras que cuando el factor de olvido λ=1 el algoritmo tiene una memoria infinita.

- La tarjeta de desarrollo PSoC brinda una alternativa para el procesamiento de señales, además de su acondicionamiento análogo de la señal, posee módulos de filtrado, amplificación, multiplexores entre otros, que son de gran ayuda para el diseño y desarrollo.

5.2. Recomendaciones

- Para la implementación del filtro Adaptativo con el algoritmo RLS se debe utilizar una tarjeta de desarrollo con un microprocesador de mayor velocidad de cálculo para que pueda realizar las operaciones matemáticas que posee el algoritmo RLS de una forma más eficiente.
- Encontrar el valor óptimo del orden del filtro y su factor de olvido, para así obtener una mejor recomposición de la señal con una mayor atenuación del ruido.
- Para la cancelación de ruido se requiere de sistema de retroalimentación de lazo cerrado adaptativo con dos entradas, para la señal de audio y la señal de error.
- Al usar la librería orientada a las operaciones matemáticas en PSoC se debe tomar en cuenta que su activación es de forma externa y no mediante código.

CAPITULO VI

6. PROPUESTA

6.1. TITULO DE LA PROPUESTA

"DISEÑO E IMPLEMENTACION DE UN FILTRO ADAPATATIVO CON EL ALGORITMO QR-RLS EN PSOC"

6.2. INTRODUCCIÓN

Los filtros digitales se encuentran en innumerables aplicaciones como: sistemas de control, comunicación y cibernética, estos sistemas se pueden implementar mediante procesadores de señales.

Al existir un gran desarrollo de los diseños de filtros Adaptativos, ha llevado a los diseñadores a probar con diversas arquitecturas de hardware como son: DSP, ARM, FPGA, etc.

Estas arquitecturas cuentas con una gran capacidad de realizar operaciones aritméticas y lógicas en paralelo. Para este diseño propuesto se utilizará una Arquitectura de PSoC para implementar el algoritmo de filtrado adaptable QR-RLS en el cual su ventaja es mejorar el rendimiento del filtro y evitan explícitamente la inversión de matrices, siendo robustas y más asequibles su implementación de hardware a diferencia del algoritmo RLS.

Este algoritmo QR-RLS encuentra una matriz Q, donde pondrá a cero todos os elementos de la fila inferior, esto es posible por una serie de rotaciones de Givens. Estos coeficientes de Given serán representados por un conjunto de

ecuaciones. Debido al paralelismo que tiene el algoritmo QR-RLS hay reducción de costo computacional.

6.3. OBJETIVOS

6.3.1. OBJETIVO GENERAL

Diseñar e implementar un filtro adaptativo con el algoritmo QR-RLS en PSoC.

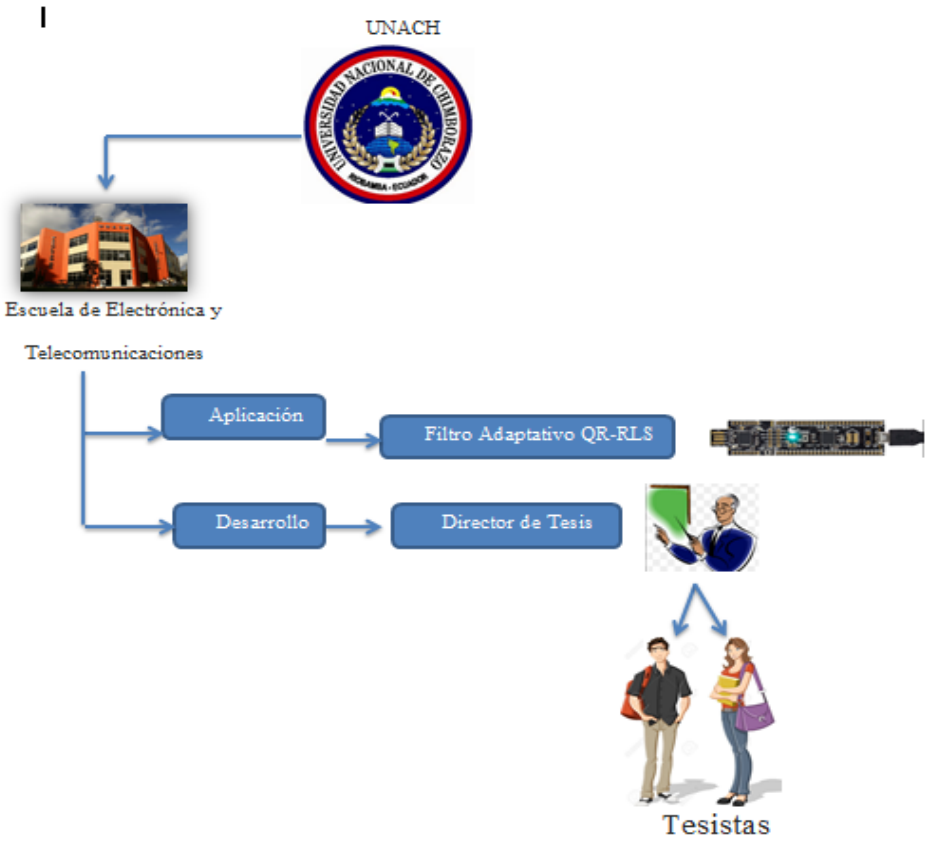
6.3.2. OBJETIVO ESPECIFICO

- Analizar el algoritmo QR-RLS para los filtros adaptativos.
- Desarrollar el algoritmo QR-RLS del filtro adaptativo en un PSoC para la cancelación de ruido.
- Comprobar el funcionamiento del algoritmo QR-RLS en audio.
- Comparar los resultados obtenidos del filtro adaptativo en PSOC con Matlab.

6.4. DESCRIPCIÓN DEL PROBLEMA

Las señales eléctricas de una transmisión pueden verse perturbadas durante él envió de información, estas perturbaciones pueden ser diferentes de una transmisión a otra; esto ocurre cuando no se conoce las características de la señal o se conocen, pero se sabe que son cambiantes con el tiempo.

6.5. DISEÑO ORGANIZACIONAL



6.6. MONITOREO Y EVALUACION DE LA PROPUESTA.

El monitoreo y la evaluación de la propuesta se la realizara a través de pruebas del sistema de filtro adaptativo con el algoritmo QR-RLS para la cancelación de ruido.

El desarrollo de este algoritmo QR-RLS en PSoC es beneficioso ya que esta arquitectura brinda todo un conjunto de dispositivos de un diseñador necesita en el momento de programar, este diseño sería mejor el resultado ya que al desarrollar este filtro adaptativo en PSoC ocupa menos proceso matemático y por ello se tendrá una señal filtrada adecuada.

CAPÍTULO VII

7. BIBLIOGRAFÍA

Apolo Castillo, H. N., & Córdova Medina, A. E. (2010). *Universidad Politecnica Salesiana*. Recuperado el 31 de Marzo de 2016, de http://dspace.ups.edu.ec/bitstream/123456789/418/15/UPS-CT001878.pdf

Castañeda Cardenas, J. A., Nieto Areas, M. A., & Ortiz Bravo, V. A. (1 de Abril de 2013). *Scientia et Technica.* Recuperado el 4 de Abril de 2016, de Analisis y Apliación del Filtro de Kalman a una señal con ruido aleatorio: file:///C:/Users/Noe/Downloads/Dialnet-AnalisisYAplicacionDelFiltroDeKalmanAUnaSenalConRu-4320424%20(3).pdf

Cordoba, U. N. (s.f.). *Procesamiento Digital de Señales.* Recuperado el 26 de 04 de 2016, de Filtros Adaptativos-RLS-LMS-Filtro Kalman: http://www.dsp.efn.unc.edu.ar/documentos/Filtros_adaptivos.pdf

Duran, I. (2010). *Implementacion del Algoritmo NLMS en un DSP.* Mexico.

G.Proakis, J. (2007). *Tratamiento Digital de Señales* (Vol. Cuarta Edision). Madrid: Printed.

Prado Obando, G. (2 de Junio de 2005). *Tecnicas Recursivas para Estimación Dinamica.* Recuperado el 26 de Abril de 2016, de Fundacion Universitaria

Konrad Lorenz: http://www.konradlorenz.edu.co/images/stories/suma_digital_matematicas/Trabajo_Grado_Guillermo_Prado_613003.pdf

Rodriguez, P., & Caamaño, F. (2008). *HighSpeed Download Packet Acces.* Recuperado el 6 de Abril de 2016, de Universidad de la Republica: http://iie.fing.edu.uy/publicaciones/2008/RC08/RC10.pdf

Rojas, E. (s.f.). Kits Educacionales PSOC de señal mixta. *Universidad Catolica Nuestra Sra. de la Asunción.*, 1-9.

U.N.S. (2011). *Ingelec*. Recuperado el 23 de Marzo de 2016, de http://www.ingelec.uns.edu.ar/pds2803/Materiales/Cap07/07-Cap07.pdf

Vazquess Burgos, I. S. (10 de Junio de 2010). *Escuela Superior de Ingenieria Mecanica Electrica.* Recuperado el 5 de Abril de 2016, de Filtrado Adaptativo Difuso con un DSP TMS320C6713: http://tesis.ipn.mx/bitstream/handle/123456789/9493/293.pdf?sequence=1

Vazquez, I. L. (2010). *Filtros Digitales.* Mexico.

Watanabe Ruiz, C. (Septiembre de 2012). *Universidad Politecnica de Madrid.* Recuperado el 5 de Abril de 2016, de Escuela Universitaria de Ingenieria Tecnica de Telecomunicaciones: http://oa.upm.es/13819/1/PFC_CESAR_E_WATANABE_RUIZ.pdf

CAPITULO VIII

8. ANEXOS

ANEXO 1

DATASHEET DE PSoC® 5LP: CY8C58LP

General Description

PSoC® 5LP is a true programmable embedded system-on-chip, integrating configurable analog and digital peripherals, memory, and a microcontroller on a single chip. The PSoC 5LP architecture boosts performance through:

- 32-bit ARM Cortex-M3 core plus DMA controller and digital filter processor, at up to 80 MHz
- Ultra low power with industry's widest voltage range
- Programmable digital and analog peripherals enable custom functions
- Flexible routing of any analog or digital peripheral function to any pin

PSoC devices employ a highly configurable system-on-chip architecture for embedded control design. They integrate configurable analog and digital circuits, controlled by an on-chip microcontroller. A single PSoC device can integrate as many as 100 digital and analog peripheral functions, reducing design time, board space, power consumption, and system cost while improving system quality.

Features

- Operating characteristics
  - Voltage range: 1.71 to 5.5 V, up to 6 power domains
  - Temperature range (ambient) –40 to 85 °C[1]
  - DC to 80-MHz operation
  - Power modes
    - Active mode 3.1 mA at 6 MHz, and 15.4 mA at 48 MHz
    - 2-µA sleep mode
    - 300-nA hibernate mode with RAM retention
  - Boost regulator from 0.5-V input up to 5-V output
- Performance
  - 32-bit ARM Cortex-M3 CPU, 32 interrupt inputs
  - 24-channel direct memory access (DMA) controller
  - 24-bit 64-tap fixed-point digital filter processor (DFB)
- Memories
  - Up to 256 KB program flash, with cache and security features
  - Up to 32 KB additional flash for error correcting code (ECC)
  - Up to 64 KB RAM
  - 2 KB EEPROM
- Digital peripherals
  - Four 16-bit timer, counter, and PWM (TCPWM) blocks
  - I^2C, 1 Mbps bus speed
  - USB 2.0 certified Full-Speed (FS) 12 Mbps
  - Full CAN 2.0b, 16 Rx, 8 Tx buffers
  - 20 to 24 universal digital blocks (UDB), programmable to create any number of functions:
    - 8-, 16-, 24-, and 32-bit timers, counters, and PWMs
    - I^2C, UART, SPI, I2S, LIN 2.0 interfaces
    - Cyclic redundancy check (CRC)
    - Pseudo random sequence (PRS) generators
    - Quadrature decoders
    - Gate-level logic functions
- Programmable clocking
  - 3- to 74-MHz internal oscillator, 1% accuracy at 3 MHz
  - 4- to 25-MHz external crystal oscillator
  - Internal PLL clock generation up to 80 MHz
  - Low-power internal oscillator at 1, 33, and 100 kHz
  - 32.768-kHz external watch crystal oscillator
  - 12 clock dividers routable to any peripheral or I/O
- Analog peripherals
  - Configurable 8- to 20-bit delta-sigma ADC
  - Up to two 12-bit SAR ADCs
  - Four 8-bit DACs
  - Four comparators
  - Four opamps
  - Four programmable analog blocks, to create:
    - Programmable gain amplifier (PGA)
    - Transimpedance amplifier (TIA)
    - Mixer
    - Sample and hold circuit
  - CapSense® support, up to 62 sensors
  - 1.024 V ±0.1% internal voltage reference
- Versatile I/O system
  - 46 to 72 I/O pins – up to 62 general-purpose I/Os (GPIOs)
  - Up to eight performance I/O (SIO) pins
    - 25 mA current sink
    - Programmable input threshold and output high voltages
    - Can act as a general-purpose comparator
    - Hot swap capability and overvoltage tolerance
  - Two USBIO pins that can be used as GPIOs
  - Route any digital or analog peripheral to any GPIO
  - LCD direct drive from any GPIO, up to 46 × 16 segments
  - CapSense support from any GPIO
  - 1.2-V to 5.5-V interface voltages, up to four power domains
- Programming, debug, and trace
  - JTAG (4-wire), serial wire debug (SWD) (2-wire), single wire viewer (SWV), and Traceport (5-wire) interfaces
  - ARM debug and trace modules embedded in the CPU core
  - Bootloader programming through I^2C, SPI, UART, USB, and other interfaces
- Package options: 68-pin QFN and 100-pin TQFP
- Development support with free PSoC Creator™ tool
  - Schematic and firmware design support
  - Over 100 PSoC Components™ integrate multiple ICs and system interfaces into one PSoC. Components are free embedded ICs represented by icons. Drag and drop component icons to design systems in PSoC Creator.
  - Includes free GCC compiler, supports Keil/ARM MDK compiler
  - Supports device programming and debugging

Note
1. The maximum storage temperature is 150 °C in compliance with JEDEC Standard JESD22-A103, High Temperature Storage Life.

More Information

Cypress provides a wealth of data at www.cypress.com to help you to select the right PSoC device for your design, and to help you to quickly and effectively integrate the device into your design. For a comprehensive list of resources, see the knowledge base article KBA86521, How to Design with PSoC 3, PSoC 4, and PSoC 5LP. Following is an abbreviated list for PSoC 5LP:

- Overview: PSoC Portfolio, PSoC Roadmap
- Product Selectors: PSoC 1, PSoC 3, PSoC 4, PSoC 5LP
  In addition, PSoC Creator includes a device selection tool.
- Application notes: Cypress offers a large number of PSoC application notes covering a broad range of topics, from basic to advanced level. Recommended application notes for getting started with PSoC 5LP are:
  - AN77759: Getting Started With PSoC 5LP
  - AN77835: PSoC 3 to PSoC 5LP Migration Guide
  - AN61290: Hardware Design Considerations
  - AN57821: Mixed Signal Circuit Board Layout
  - AN58304: Pin Selection for Analog Designs
  - AN81623: Digital Design Best Practices
  - AN73854: Introduction To Bootloaders
- Development Kits:
  - CY8CKIT-001 provides a common development platform for any one of the PSoC 1, PSoC 3, PSoC 4, or PSoC 5LP families of devices.
  - CY8CKIT-050 is designed for analog performance. It enables you to evaluate, develop and prototype high precision analog, low-power and low-voltage applications powered by PSoC 5LP.

Both kits support the PSoC Expansion Board Kit ecosystem. Expansion kits are available for a number of applications including CapSense, precision temperature measurement, and power supervision.

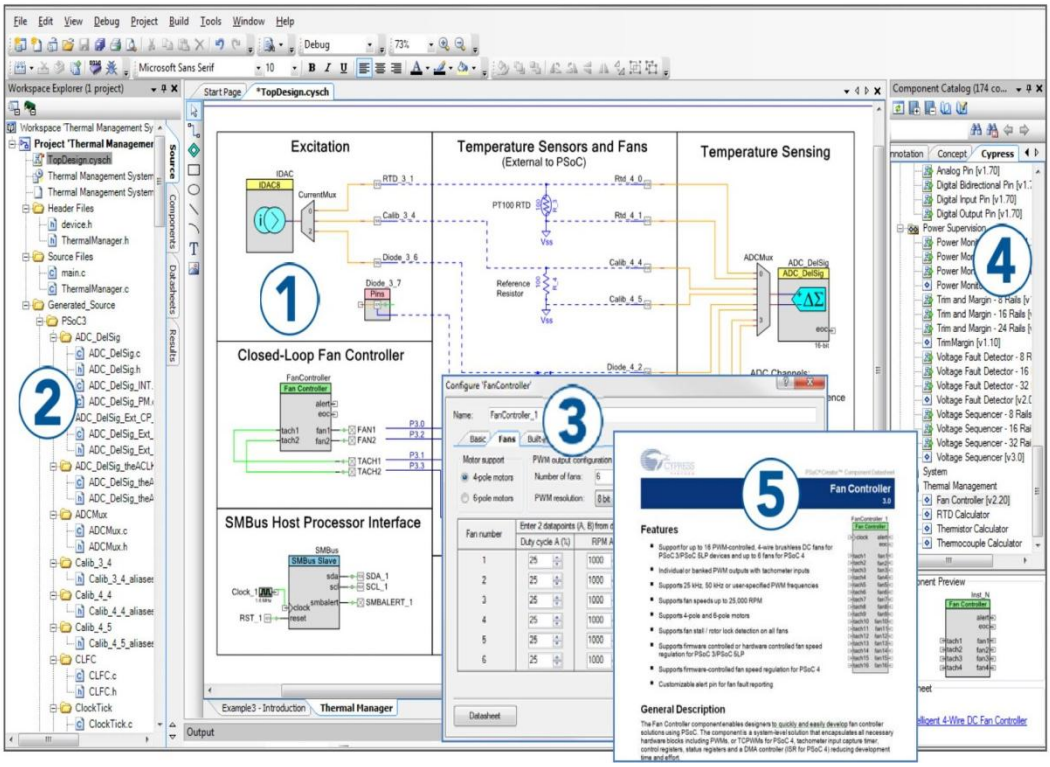
The MiniProg3 device provides an interface for flash programming and debug.

PSoC Creator

PSoC Creator is a free Windows-based Integrated Design Environment (IDE). It enables concurrent hardware and firmware design of PSoC 3, PSoC 4, and PSoC 5LP based systems. Create designs using classic, familiar schematic capture supported by over 100 pre-verified, production-ready PSoC Components; see the list of component datasheets. With PSoC Creator, you can:

1. Drag and drop component icons to build your hardware system design in the main design workspace
2. Codesign your application firmware with the PSoC hardware, using the PSoC Creator IDE C compiler
3. Configure components using the configuration tools
4. Explore the library of 100+ components
5. Review component datasheets

Figure 1. Multiple-Sensor Example Project in PSoC Creator



Contents

1. Architectural Overview

Introducing the CY8C58LP family of ultra low power, flash Programmable System-on-Chip (PSoC) devices, part of a scalable 8-bit PSoC 3 and 32-bit PSoC 5LP platform. The CY8C58LP family provides configurable blocks of analog, digital, and interconnect circuitry around a CPU subsystem. The combination of a CPU with a flexible analog subsystem, digital subsystem, routing, and I/O enables a high level of integration in a wide variety of consumer, industrial, and medical applications.

Figure 1-1. Simplified Block Diagram

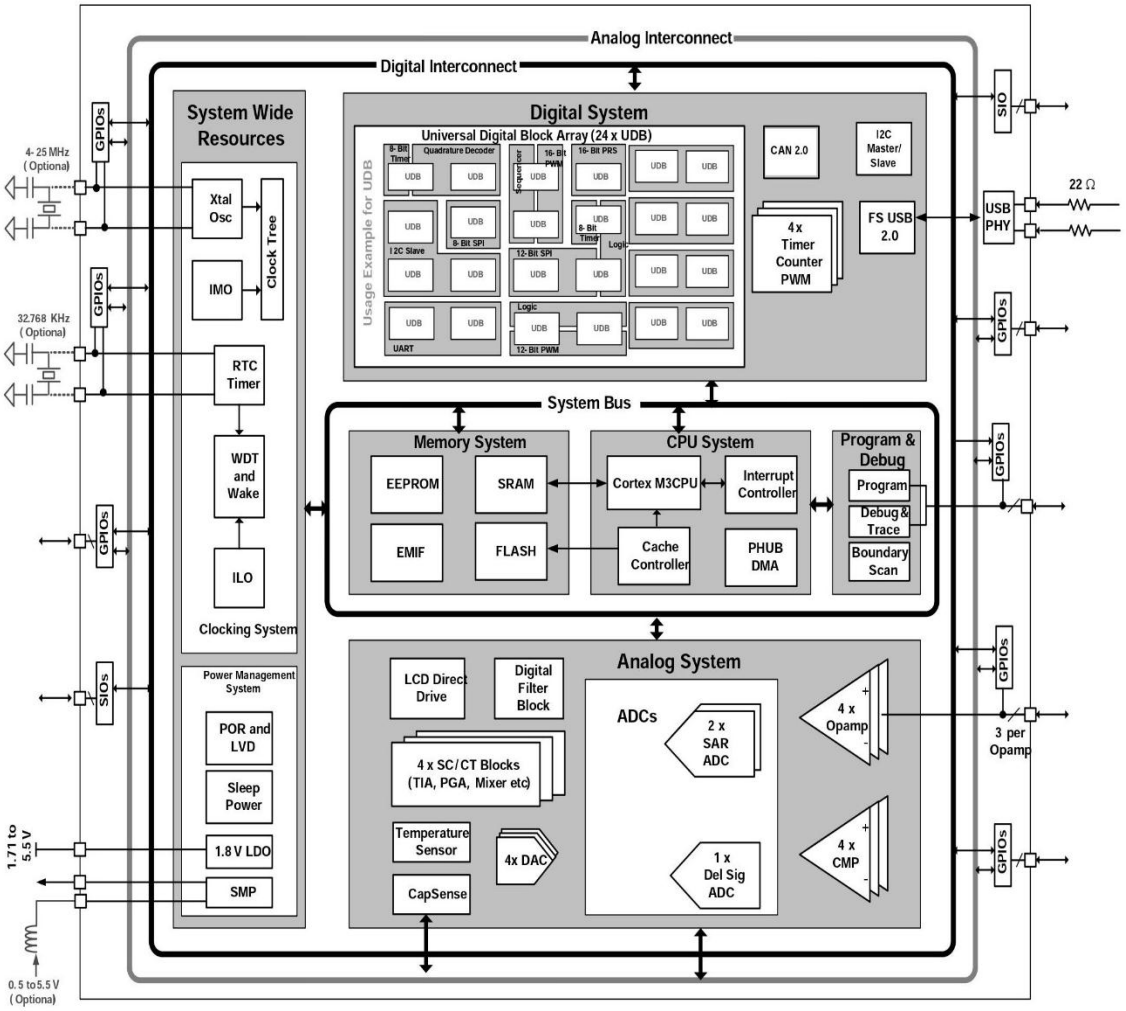


Figure 1-1 illustrates the major components of the CY8C58LP family. They are:

- ARM Cortex-M3 CPU subsystem
- Nonvolatile subsystem
- Programming, debug, and test subsystem
- Inputs and outputs
- Clocking
- Power
- Digital subsystem
- Analog subsystem

PSoC's digital subsystem provides half of its unique configurability. It connects a digital signal from any peripheral to any pin through the digital system interconnect (DSI). It also provides functional flexibility through an array of small, fast, low power UDBs. PSoC Creator provides a library of pre-built and tested standard digital peripherals (UART, SPI, LIN, PRS, CRC, timer, counter, PWM, AND, OR, and so on) that are mapped to the UDB array. You can also easily create a digital circuit using boolean primitives by means of graphical design entry. Each UDB contains programmable array logic (PAL)/programmable logic device (PLD) functionality, together with a small state machine engine to support a wide variety of peripherals.

In addition to the flexibility of the UDB array, PSoC also provides configurable digital blocks targeted at specific functions. For the CY8C58LP family, these blocks can include four 16-bit timers, counters, and PWM blocks; I^2C slave, master, and multimaster; Full-Speed USB; and Full CAN 2.0.

For more details on the peripherals see the "Example Peripherals" section on page 37 of this datasheet. For information on UDBs, DSI, and other digital blocks, see the "Digital Subsystem" section on page 37 of this datasheet.

PSoC's analog subsystem is the second half of its unique configurability. All analog performance is based on a highly accurate absolute voltage reference with less than 0.1% error over temperature and voltage. The configurable analog subsystem includes:

- Analog muxes
- Comparators
- Analog mixers
- Voltage references
- ADCs
- DACs
- Digital filter block (DFB)

All GPIO pins can route analog signals into and out of the device using the internal analog bus. This allows the device to interface up to 62 discrete analog signals. One of the ADCs in the analog subsystem is a fast, accurate, configurable delta-sigma ADC with these features:

- Less than 100-µV offset
- A gain error of 0.2%
- Integral non linearity (INL) less than ±2 LSB
- Differential non linearity (DNL) less than ±1 LSB
- SINAD better than 84 dB in 16-bit mode

This converter addresses a wide variety of precision analog applications including some of the most demanding sensors.

The CY8C58LP family also offers up to two SAR ADCs. Featuring 12-bit conversions at up to 1 M samples per second, they also offer low nonlinearity and offset errors and SNR better than 70 dB. They are well-suited for a variety of higher speed analog applications.

The output of any of the ADCs can optionally feed the programmable DFB via DMA without CPU intervention. You can configure the DFB to perform IIR and FIR digital filters and several user defined custom functions. The DFB can implement filters with up to 64 taps. It can perform a 48-bit multiply-accumulate (MAC) operation in one clock cycle.

Four high-speed voltage or current DACs support 8-bit output signals at an update rate of up to 8 Msps. They can be routed out of any GPIO pin. You can create higher resolution voltage PWM DAC outputs using the UDB array. This can be used to create a pulse width modulated (PWM) DAC of up to 10 bits, at up to 48 kHz. The digital DACs in each UDB support PWM, PRS, or delta-sigma algorithms with programmable widths.

In addition to the ADCs, DACs, and DFB, the analog subsystem provides multiple:

- Comparators
- Uncommitted opamps
- Configurable switched capacitor/continuous time (SC/CT) blocks. These support:
  - Transimpedance amplifiers
  - Programmable gain amplifiers
  - Mixers
  - Other similar analog components

See the "Analog Subsystem" section on page 47 of this datasheet for more details.

PSoC's CPU subsystem is built around a 32-bit three-stage pipelined ARM Cortex-M3 processor running at up to 80 MHz. The Cortex-M3 includes a tightly integrated nested vectored interrupt controller (NVIC) and various debug and trace modules. The overall CPU subsystem includes a DMA controller, flash cache, and RAM. The NVIC provides low latency, nested interrupts, and tail-chaining of interrupts and other features to increase the efficiency of interrupt handling. The DMA controller enables peripherals to exchange data without CPU involvement. This allows the CPU to run slower (saving power) or use those CPU cycles to improve the performance of firmware algorithms. The flash cache also reduces system power consumption by allowing less frequent flash access.

PSoC's nonvolatile subsystem consists of flash, byte-writeable EEPROM, and nonvolatile configuration options. It provides up to 256 KB of on-chip flash. The CPU can reprogram individual blocks of flash, enabling boot loaders. You can enable an ECC for high reliability applications. A powerful and flexible protection model secures the user's sensitive information, allowing selective memory block locking for read and write protection. Two KB of byte-writable EEPROM is available on-chip to store application data. Additionally, selected configuration options such as boot speed and pin drive mode are stored in nonvolatile memory. This allows settings to activate immediately after POR.

The three types of PSoC I/O are extremely flexible. All I/Os have many drive modes that are set at POR. PSoC also provides up to four I/O voltage domains through the V_{DDIO} pins. Every GPIO has analog I/O, LCD drive, CapSense, flexible interrupt generation, slew rate control, and digital I/O capability. The SIOs on PSoC allow V_{OH} to be set independently of V_{DDIO} when used as outputs. When SIOs are in input mode they are high impedance. This is true even when the device is not powered or when the pin voltage goes above the supply voltage. This makes the SIO ideally suited for use on an I^2C bus where the PSoC may not be powered when other devices on the bus are. The SIO pins also have high current sink capability for applications such as LED drives. The programmable input threshold feature of the SIO can be used to make the SIO function as a general purpose analog comparator. For devices with FS USB, the USB physical interface is also provided (USBIO). When not using USB, these pins may also be used for limited digital functionality and device programming. All the features of the PSoC I/Os are covered in detail in the "I/O System and Routing" section on page 30 of this datasheet.

The PSoC device incorporates flexible internal clock generators, designed for high stability and factory trimmed for high accuracy. The internal main oscillator (IMO) is the master clock base for the system, and has one-percent accuracy at 3 MHz. The IMO can be configured to run from 3 MHz up to 74 MHz. Multiple clock derivatives can be generated from the main clock frequency to meet application needs. The device provides a PLL to generate system clock frequencies up to 80 MHz from the IMO, external crystal, or external reference clock. It also contains a separate, very low-power internal low-speed oscillator (ILO) for the sleep and watchdog timers. A 32.768-kHz external watch crystal is also supported for use in RTC applications. The clocks, together with programmable clock dividers, provide the flexibility to integrate most timing requirements.

The CY8C58LP family supports a wide supply operating range from 1.71 to 5.5 V. This allows operation from regulated supplies such as 1.8 ± 5%, 2.5 V ±10%, 3.3 V ± 10%, or 5.0 V ± 10%, or directly from a wide range of battery types. In addition, it provides an integrated high efficiency synchronous boost converter that can power the device from supply voltages as low as 0.5 V. This enables the device to be powered directly from a single battery. In addition, you can use the boost converter to generate other voltages required by the device, such as a 3.3 V supply for LCD glass drive. The boost's output is available on the VBOOST pin, allowing other devices in the application to be powered from the PSoC.

PSoC supports a wide range of low power modes. These include a 300-nA hibernate mode with RAM retention and a 2-µA sleep mode with RTC. In the second mode, the optional 32.768-kHz watch crystal runs continuously and maintains an accurate RTC.

Power to all major functional blocks, including the programmable digital and analog peripherals, can be controlled independently by firmware. This allows low power background processing when some peripherals are not in use. This, in turn, provides a total device current of only 3.1 mA when the CPU is running at 6 MHz.

The details of the PSoC power modes are covered in the "Power System" section on page 25 of this datasheet.

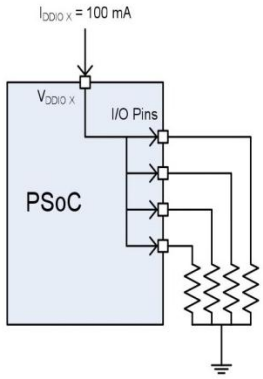
PSoC uses JTAG (4 wire) or SWD (2 wire) interfaces for programming, debug, and test. Using these standard interfaces you can debug or program the PSoC with a variety of hardware solutions from Cypress or third party vendors. The Cortex-M3 debug and trace modules include FPB, DWT, ETM, and ITM. These modules have many features to help solve difficult debug and trace problems. Details of the programming, test, and debugging interfaces are discussed in the "Programming, Debug Interfaces, Resources" section on page 57 of this datasheet.

2. Pinouts

Each VDDIO pin powers a specific set of I/O pins. (The USBIOs are powered from VDDD.) Using the VDDIO pins, a single PSoC can support multiple voltage levels, reducing the need for off-chip level shifters. The black lines drawn on the pinout diagrams in Figure 2-3 and Figure 2-4 show the pins that are powered by each VDDIO.

Each VDDIO may source up to 100 mA total to its associated I/O pins, as shown in Figure 2-1.

Figure 2-1. VDDIO Current Limit



Conversely, for the 100-pin and 68-pin devices, the set of I/O pins associated with any VDDIO may sink up to 100 mA total, as shown in Figure 2-2.

Figure 2-2. I/O Pins Current Limit

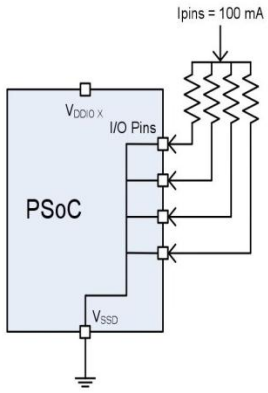


Figure 2-3. 68-pin QFN Part Pinout[2]

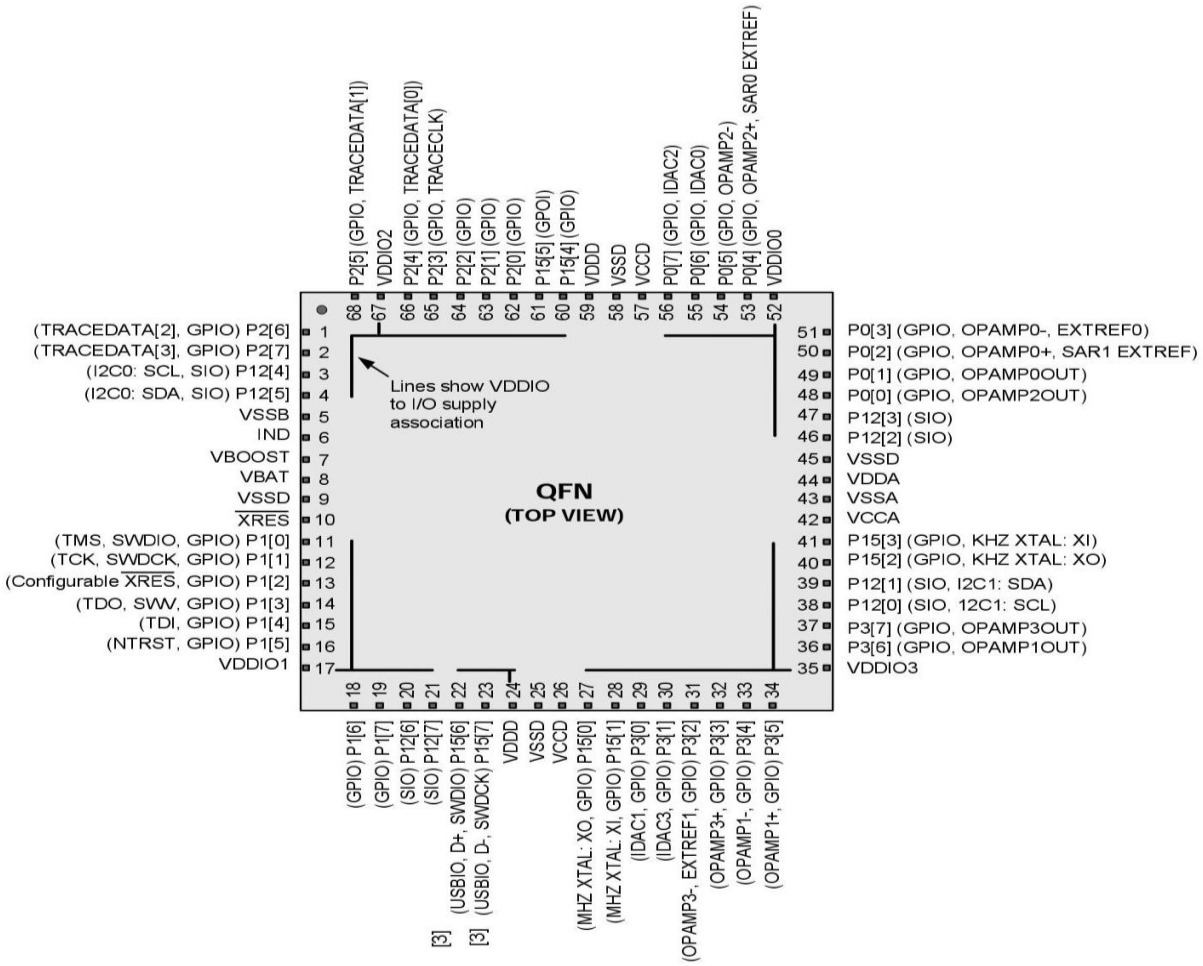


Figure 2-4. 100-pin TQFP Part Pinout

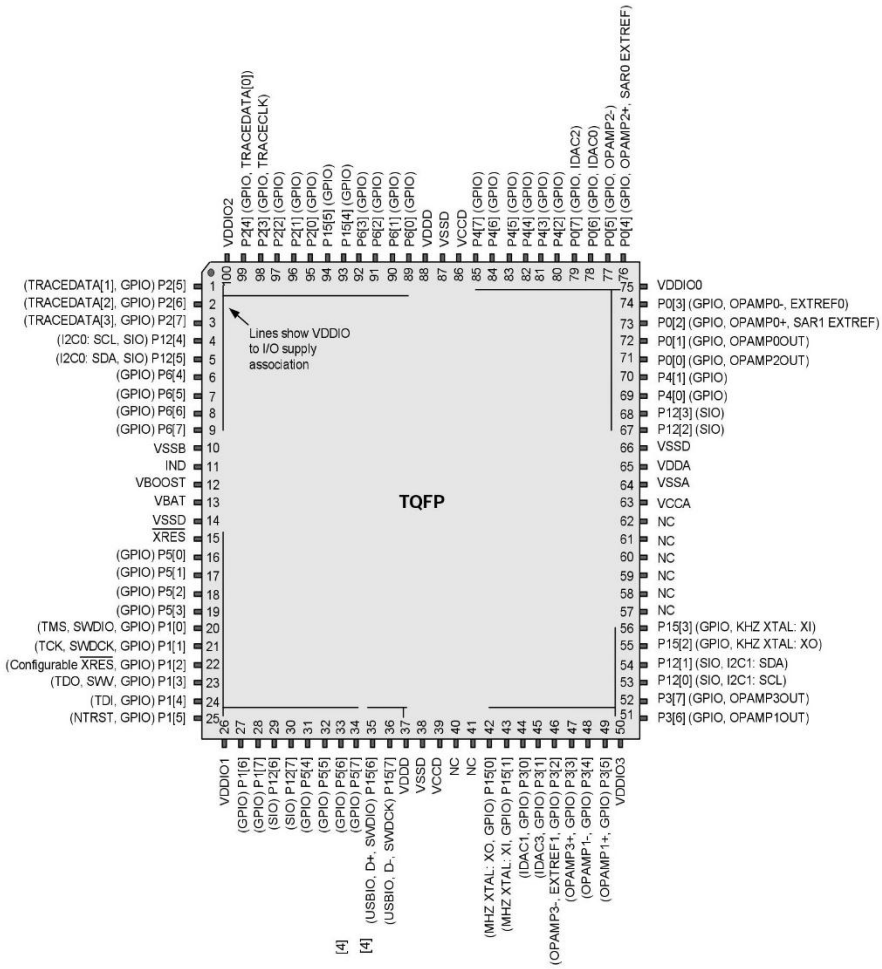
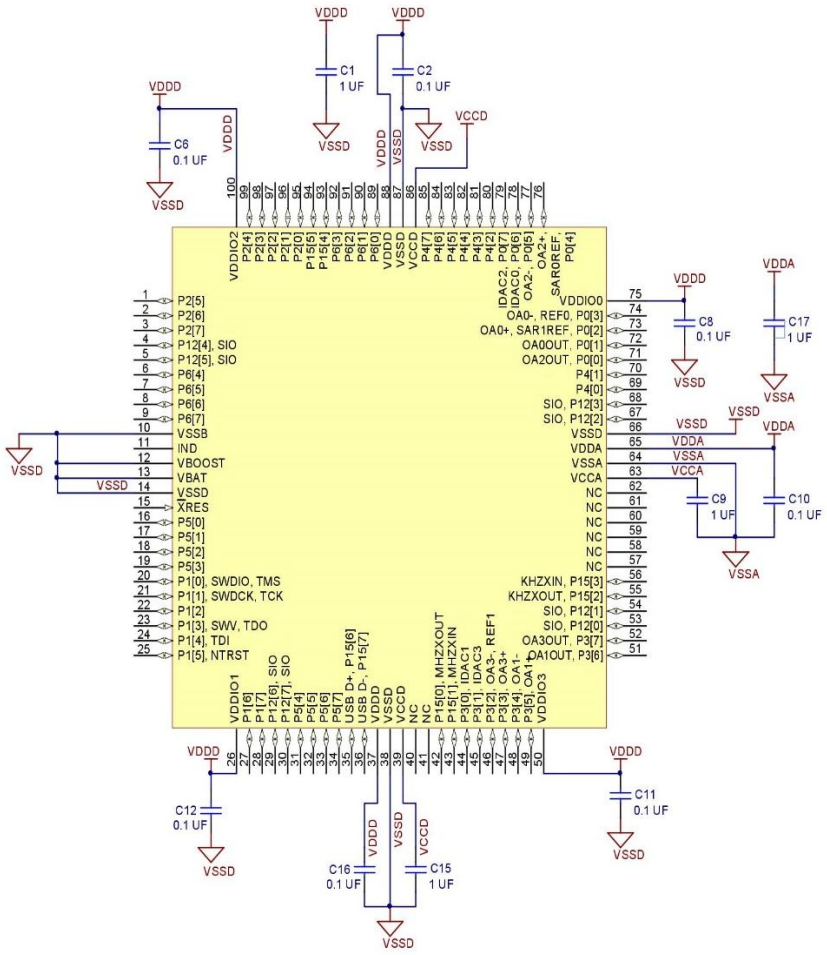


Figure 2-5 on page 9 and Figure 2-6 on page 10 show an example schematic and an example PCB layout, for the 100-pin TQFP part, for optimal analog performance on a two-layer board.

- The two pins labeled VDDD must be connected together.
- The two pins labeled VCCD must be connected together, with capacitance added, as shown in Figure 2-5 and "Power System" section on page 25. The trace between the two VCCD pins should be as short as possible.
- The two pins labeled VSSD must be connected together.

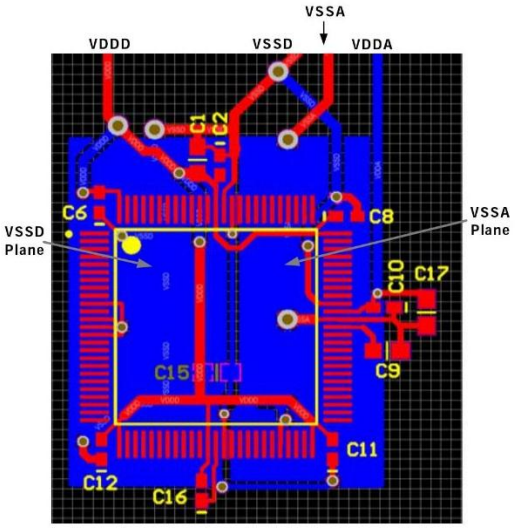
For information on circuit board layout issues for mixed signals, refer to the application note, AN57821 - Mixed Signal Circuit Board Layout Considerations for PSoC® 3 and PSoC 5.

Figure 2-5. Example Schematic for 100-pin TQFP Part with Power Connections



Note The two V_{CCD} pins must be connected together with as short a trace as possible. A trace under the device is recommended, as shown in Figure 2-6.

Figure 2-6. Example PCB Layout for 100-pin TQFP Part for Optimal Analog Performance



3. Pin Descriptions

IDAC0, IDAC1, IDAC2, IDAC3. Low-resistance output pin for high-current DACs (IDAC).

Opamp0out, Opamp1out, Opamp2out, Opamp3out. High current output of uncommitted opamp.[5]

Extref0, Extref1. External reference input to the analog system.

SAR0 EXTREF, SAR1 EXTREF. External references for SAR ADCs

Opamp0-, Opamp1-, Opamp2-, Opamp3-. Inverting input to uncommitted opamp.

Opamp0+, Opamp1+, Opamp2+, Opamp3+. Noninverting input to uncommitted opamp.

GPIO. Provides interfaces to the CPU, digital peripherals, analog peripherals, interrupts, LCD segment drive, and CapSense.[5]

I2C0: SCL, I2C1: SCL. I^2C SCL line providing wake from sleep on an address match. Any I/O pin can be used for I^2C SCL if wake from sleep is not required.

I2C0: SDA, I2C1: SDA. I^2C SDA line providing wake from sleep on an address match. Any I/O pin can be used for I^2C SDA if wake from sleep is not required.

Ind. Inductor connection to boost pump.

kHz XTAL: Xo, kHz XTAL: Xi. 32.768-kHz crystal oscillator pin.

MHz XTAL: Xo, MHz XTAL: Xi. 4 to 25-MHz crystal oscillator pin.

nTRST. Optional JTAG Test Reset programming and debug port connection to reset the JTAG connection.

SIO. Provides interfaces to the CPU, digital peripherals and interrupts with a programmable high threshold voltage, analog comparator, high sink current, and high impedance state when the device is unpowered.

SWDCK. SWD Clock programming and debug port connection.

SWDIO. SWD Input and Output programming and debug port connection.

TCK. JTAG Test Clock programming and debug port connection.

TDI. JTAG Test Data In programming and debug port connection.

TDO. JTAG Test Data Out programming and debug port connection.

TMS. JTAG Test Mode Select programming and debug port connection.

TRACECLK. Cortex-M3 TRACEPORT connection, clocks TRACEDATA pins.

TRACEDATA[3:0]. Cortex-M3 TRACEPORT connections, output data.

SWV. SWV output.

USBIO, D+. Provides D+ connection directly to a USB 2.0 bus. May be used as a digital I/O pin; it is powered from VDDD instead of from a VDDIO. Pins are Do Not Use (DNU) on devices without USB.

USBIO, D-. Provides D- connection directly to a USB 2.0 bus. May be used as a digital I/O pin; it is powered from VDDD instead of from a VDDIO. Pins are Do Not Use (DNU) on devices without USB.

VBOOST. Power sense connection to boost pump.

Note
5. GPIOs with opamp outputs are not recommended for use with CapSense.

The Cortex-M3 does not support ARM instructions for SRAM addresses.

- Bit-band support for the SRAM region. Atomic bit-level write and read operations for SRAM addresses.
- Unaligned data storage and access. Contiguous storage of data of different byte lengths.
- Operation at two privilege levels (privileged and user) and in two modes (thread and handler). Some instructions can only be executed at the privileged level. There are also two stack pointers: Main (MSP) and Process (PSP). These features support a multitasking operating system running one or more user-level processes.
- Extensive interrupt and system exception support.

4.1.2 Cortex-M3 Operating Modes

The Cortex-M3 operates at either the privileged level or the user level, and in either the thread mode or the handler mode. Because the handler mode is only enabled at the privileged level, there are actually only three states, as shown in Table 4-1.

Table 4-1. Operational Level

Condition	Privileged	User
Running an exception	Handler mode	Not used
Running main program	Thread mode	Thread mode

At the user level, access to certain instructions, special registers, configuration registers, and debugging components is blocked. Attempts to access them cause a fault exception. At the privileged level, access to all instructions and registers is allowed.

The processor runs in the handler mode (always at the privileged level) when handling an exception, and in the thread mode when not.

4.1.3 CPU Registers

The Cortex-M3 CPU registers are listed in Table 4-2. Registers R0-R15 are all 32 bits wide.

Table 4-2. Cortex M3 CPU Registers

Register	Description
R0-R12	General purpose registers R0-R12 have no special architecturally defined uses. Most instructions that specify a general purpose register specify R0-R12. ■ Low registers: Registers R0-R7 are accessible by all instructions that specify a general purpose register. ■ High registers: Registers R8-R12 are accessible by all 32-bit instructions that specify a general purpose register; they are not accessible by all 16-bit instructions.
R13	R13 is the stack pointer register. It is a banked register that switches between two 32-bit stack pointers: the main stack pointer (MSP) and the process stack pointer (PSP). The PSP is used only when the CPU operates at the user level in thread mode. The MSP is used in all other privilege levels and modes. Bits[0:1] of the SP are ignored and considered to be 0, so the SP is always aligned to a word (4 byte) boundary.
R14	R14 is the link register (LR). The LR stores the return address when a subroutine is called.
R15	R15 is the program counter (PC). Bit 0 of the PC is ignored and considered to be 0, so instructions are always aligned to a half word (2 byte) boundary.
xPSR	The program status registers are divided into three status registers, which are accessed either together or separately: ■ Application program status register (APSR) holds program execution status bits such as zero, carry, negative, in bits[27:31]. ■ Interrupt program status register (IPSR) holds the current exception number in bits[0:8]. ■ Execution program status register (EPSR) holds control bits for interrupt continuable and IF-THEN instructions in bits[10:15] and [25:26]. Bit 24 is always set to 1 to indicate Thumb mode. Trying to clear it causes a fault exception.
PRIMASK	A 1-bit interrupt mask register. When set, it allows only the nonmaskable interrupt (NMI) and hard fault exception. All other exceptions and interrupts are masked.
FAULTMASK	A 1-bit interrupt mask register. When set, it allows only the NMI. All other exceptions and interrupts are masked.
BASEPRI	A register of up to nine bits that define the masking priority level. When set, it disables all interrupts of the same or higher priority value. If set to 0 then the masking function is disabled.
CONTROL	A 2-bit register for controlling the operating mode. Bit 0: 0 = privileged level in thread mode, 1 = user level in thread mode. Bit 1: 0 = default stack (MSP) is used, 1 = alternate stack is used. If in thread mode or user level then the alternate stack is the PSP. There is no alternate stack for handler mode; the bit must be 0 while in handler mode.

4.2 Cache Controller

The CY8C58LP family has a 1 KB, 4-way set-associative instruction cache between the CPU and the flash memory. This improves instruction execution rate and reduces system power consumption by requiring less frequent flash access.

4.3 DMA and PHUB

The PHUB and the DMA controller are responsible for data transfer between the CPU and peripherals, and also data transfers between peripherals. The PHUB and DMA also control device configuration during boot. The PHUB consists of:

- A central hub that includes the DMA controller, arbiter, and router
- Multiple spokes that radiate outward from the hub to most peripherals

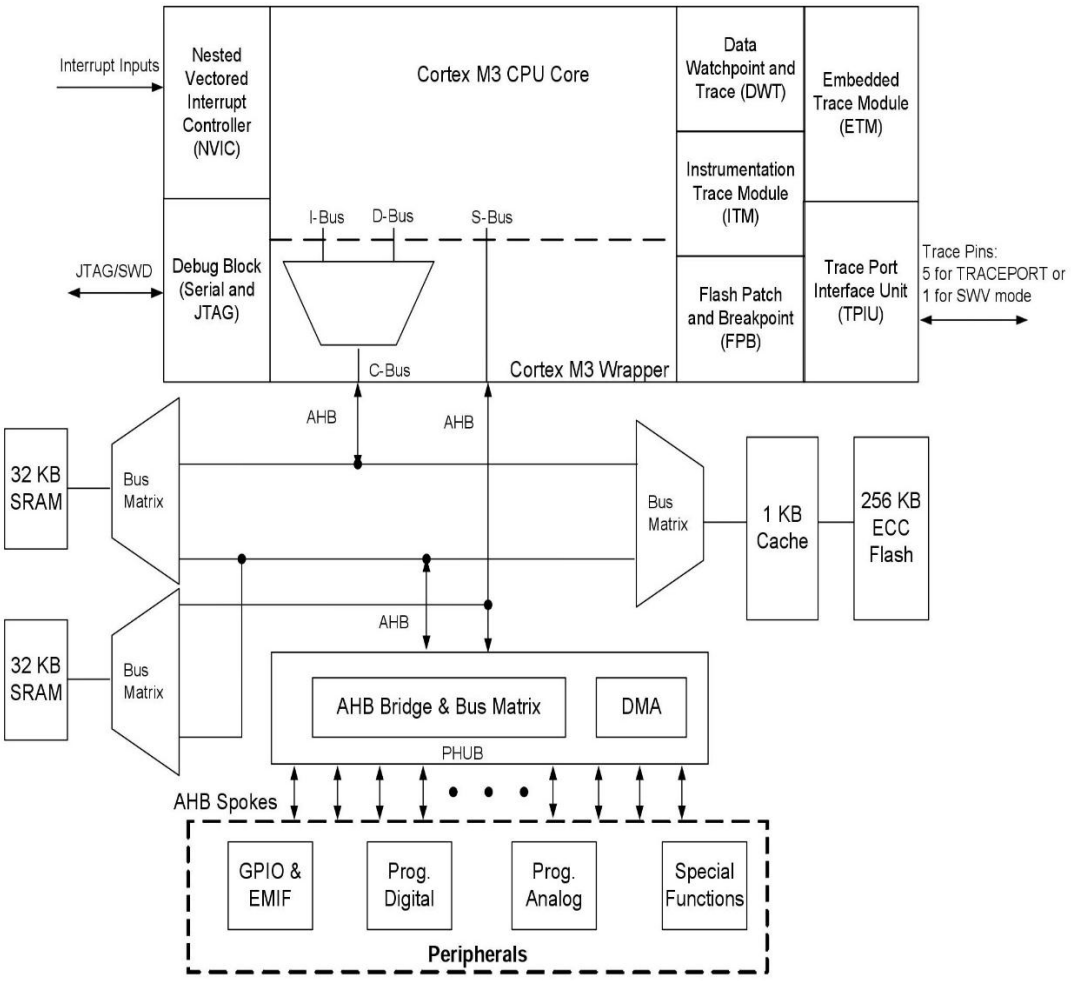
There are two PHUB masters: the CPU and the DMA controller. Both masters may initiate transactions on the bus. The DMA channels can handle peripheral communication without CPU intervention. The arbiter in the central hub determines which DMA channel is the highest priority if there are multiple requests.

4. CPU

4.1 ARM Cortex-M3 CPU

The CY8C58LP family of devices has an ARM Cortex-M3 CPU core. The Cortex-M3 is a low-power 32-bit three-stage pipelined Harvard-architecture CPU that delivers 1.25 DMIPS/MHz. It is intended for deeply embedded applications that require fast interrupt handling features.

Figure 4-1. ARM Cortex-M3 Block Diagram



The Cortex-M3 CPU subsystem includes these features:

- ARM Cortex-M3 CPU
- Programmable nested vectored interrupt controller (NVIC), tightly integrated with the CPU core
- Full featured debug and trace modules, tightly integrated with the CPU core
- Up to 256 KB of flash memory, 2 KB of EEPROM, and 64 KB of SRAM
- Cache controller
- Peripheral HUB (PHUB)
- DMA controller
- External memory interface (EMIF)

4.1.1 Cortex-M3 Features

The Cortex-M3 CPU features include:

- 4 GB address space. Predefined address regions for code, data, and peripherals. Multiple buses for efficient and simultaneous accesses of instructions, data, and peripherals.
- The Thumb®-2 instruction set, which offers ARM-level performance at Thumb-level code density. This includes 16-bit and 32-bit instructions. Advanced instructions include:
  - Bit-field control
  - Hardware multiply and divide
  - Saturation
  - If-Then
  - Wait for events and interrupts
  - Exclusive access and barrier
  - Special register access

The Cortex-M3 does not support ARM instructions for SRAM addresses.

- Bit-band support for the SRAM region. Atomic bit-level write and read operations for SRAM addresses.
- Unaligned data storage and access. Contiguous storage of data of different byte lengths.
- Operation at two privilege levels (privileged and user) and in two modes (thread and handler). Some instructions can only be executed at the privileged level. There are also two stack pointers: Main (MSP) and Process (PSP). These features support a multitasking operating system running one or more user-level processes.
- Extensive interrupt and system exception support.

4.1.2 Cortex-M3 Operating Modes

The Cortex-M3 operates at either the privileged level or the user level, and in either the thread mode or the handler mode. Because the handler mode is only enabled at the privileged level, there are actually only three states, as shown in Table 4-1.

Table 4-1. Operational Level

Condition	Privileged	User
Running an exception	Handler mode	Not used
Running main program	Thread mode	Thread mode

At the user level, access to certain instructions, special registers, configuration registers, and debugging components is blocked. Attempts to access them cause a fault exception. At the privileged level, access to all instructions and registers is allowed.

The processor runs in the handler mode (always at the privileged level) when handling an exception, and in the thread mode when not.

4.1.3 CPU Registers

The Cortex-M3 CPU registers are listed in Table 4-2. Registers R0-R15 are all 32 bits wide.

Table 4-2. Cortex M3 CPU Registers

Register	Description
R0-R12	General purpose registers R0-R12 have no special architecturally defined uses. Most instructions that specify a general purpose register specify R0-R12. ■ Low registers: Registers R0-R7 are accessible by all instructions that specify a general purpose register. ■ High registers: Registers R8-R12 are accessible by all 32-bit instructions that specify a general purpose register; they are not accessible by all 16-bit instructions.
R13	R13 is the stack pointer register. It is a banked register that switches between two 32-bit stack pointers: the main stack pointer (MSP) and the process stack pointer (PSP). The PSP is used only when the CPU operates at the user level in thread mode. The MSP is used in all other privilege levels and modes. Bits[0:1] of the SP are ignored and considered to be 0, so the SP is always aligned to a word (4 byte) boundary.
R14	R14 is the link register (LR). The LR stores the return address when a subroutine is called.
R15	R15 is the program counter (PC). Bit 0 of the PC is ignored and considered to be 0, so instructions are always aligned to a half word (2 byte) boundary.
xPSR	The program status registers are divided into three status registers, which are accessed either together or separately: ■ Application program status register (APSR) holds program execution status bits such as zero, carry, negative, in bits[27:31]. ■ Interrupt program status register (IPSR) holds the current exception number in bits[0:8]. ■ Execution program status register (EPSR) holds control bits for interrupt continuable and IF-THEN instructions in bits[10:15] and [25:26]. Bit 24 is always set to 1 to indicate Thumb mode. Trying to clear it causes a fault exception.
PRIMASK	A 1-bit interrupt mask register. When set, it allows only the nonmaskable interrupt (NMI) and hard fault exception. All other exceptions and interrupts are masked.
FAULTMASK	A 1-bit interrupt mask register. When set, it allows only the NMI. All other exceptions and interrupts are masked.
BASEPRI	A register of up to nine bits that define the masking priority level. When set, it disables all interrupts of the same or higher priority value. If set to 0 then the masking function is disabled.
CONTROL	A 2-bit register for controlling the operating mode. Bit 0: 0 = privileged level in thread mode, 1 = user level in thread mode. Bit 1: 0 = default stack (MSP) is used, 1 = alternate stack is used. If in thread mode or user level then the alternate stack is the PSP. There is no alternate stack for handler mode; the bit must be 0 while in handler mode.

4.2 Cache Controller

The CY8C58LP family has a 1 KB, 4-way set-associative instruction cache between the CPU and the flash memory. This improves instruction execution rate and reduces system power consumption by requiring less frequent flash access.

4.3 DMA and PHUB

The PHUB and the DMA controller are responsible for data transfer between the CPU and peripherals, and also data transfers between peripherals. The PHUB and DMA also control device configuration during boot. The PHUB consists of:

- A central hub that includes the DMA controller, arbiter, and router
- Multiple spokes that radiate outward from the hub to most peripherals

There are two PHUB masters: the CPU and the DMA controller. Both masters may initiate transactions on the bus. The DMA channels can handle peripheral communication without CPU intervention. The arbiter in the central hub determines which DMA channel is the highest priority if there are multiple requests.

4.3.1 PHUB Features

- CPU and DMA controller are both bus masters to the PHUB
- Eight multi-layer AHB bus parallel access paths (spokes) for peripheral access
- Simultaneous CPU and DMA access to peripherals located on different spokes
- Simultaneous DMA source and destination burst transactions on different spokes
- Supports 8-, 16-, 24-, and 32-bit addressing and data

Table 4-3. PHUB Spokes and Peripherals

PHUB Spokes	Peripherals
0	SRAM
1	IOs, PICU, EMIF
2	PHUB local configuration, Power manager, Clocks, IC, SWV, EEPROM, Flash programming interface
3	Analog interface and trim, Decimator
4	USB, CAN, I^2C, Timers, Counters, and PWMs
5	DFB
6	UDBs group 1
7	UDBs group 2

4.3.2 DMA Features

- 24 DMA channels
- Each channel has one or more transaction descriptors (TDs) to configure channel behavior. Up to 128 total TDs can be defined
- TDs can be dynamically updated
- Eight levels of priority per channel
- Any digitally routable signal, the CPU, or another DMA channel, can trigger a transaction
- Each channel can generate up to two interrupts per transfer
- Transactions can be stalled or canceled
- Supports transaction size of infinite or 1 to 64k bytes
- Large transactions may be broken into smaller bursts of 1 to 127 bytes
- TDs may be nested and/or chained for complex transactions

4.3.3 Priority Levels

The CPU always has higher priority than the DMA controller when their accesses require the same bus resources. Due to the system architecture, the CPU can never starve the DMA. DMA channels of higher priority (lower priority number) may interrupt current DMA transfers. In the case of an interrupt, the current transfer is allowed to complete its current transaction. To ensure latency limits when multiple DMA accesses are requested simultaneously, a fairness algorithm guarantees an interleaved minimum percentage of bus bandwidth for priority levels 2 through 7. Priority levels 0 and 1 do not take part in the fairness algorithm and may use 100% of the bus bandwidth. If a tie occurs on two DMA requests of the same priority level, a simple round robin method is used to evenly share the allocated bandwidth. The round robin allocation can be disabled for each DMA channel, allowing it to always be at the head of the line. Priority levels 2 to 7 are guaranteed the minimum bus bandwidth shown in Table 4-4 after the CPU and DMA priority levels 0 and 1 have satisfied their requirements.

Table 4-4. Priority Levels

Priority Level	% Bus Bandwidth
0	100.0
1	100.0
2	50.0
3	25.0
4	12.5
5	6.2
6	3.1
7	1.5

When the fairness algorithm is disabled, DMA access is granted based solely on the priority level; no bus bandwidth guarantees are made.

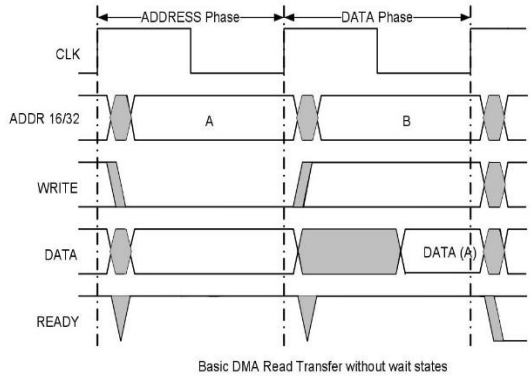
4.3.4 Transaction Modes Supported

The flexible configuration of each DMA channel and the ability to chain multiple channels allow the creation of both simple and complex use cases. General use cases include, but are not limited to:

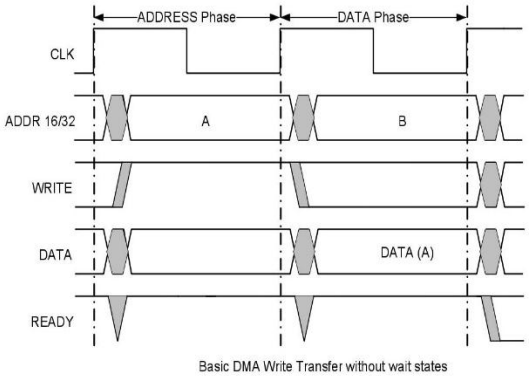
4.3.4.1 Simple DMA

In a simple DMA case, a single TD transfers data between a source and sink (peripherals or memory location). The basic timing diagrams of DMA read and write cycles are shown in Figure 4-2. For more description on other transfer modes, refer to the Technical Reference Manual.

Figure 4-2. DMA Timing Diagram



Basic DMA Read Transfer without wait states



Basic DMA Write Transfer without wait states

4.3.4.2 Auto Repeat DMA

Auto repeat DMA is typically used when a static pattern is repetitively read from system memory and written to a peripheral. This is done with a single TD that chains to itself.

4.3.4.3 Ping Pong DMA

A ping pong DMA case uses double buffering to allow one buffer to be filled by one client while another client is consuming the data previously received in the other buffer. In its simplest form, this is done by chaining two TDs together so that each TD calls the opposite TD when complete.

4.3.4.4 Circular DMA

Circular DMA is similar to ping pong DMA except it contains more than two buffers. In this case there are multiple TDs; after the last TD is complete it chains back to the first TD.

4.3.4.5 Indexed DMA

In an indexed DMA case, an external master requires access to locations on the system bus as if those locations were shared memory. As an example, a peripheral may be configured as an SPI or I^2C slave where an address is received by the external master. That address becomes an index or offset into the internal system bus memory space. This is accomplished with an initial "address fetch" TD that reads the target address location from the peripheral and writes that value into a subsequent TD in the chain. This modifies the TD chain on the fly. When the "address fetch" TD completes it moves on to the next TD, which has the new address information embedded in it. This TD then carries out the data transfer with the address location required by the external master.

4.3.4.6 Scatter Gather DMA

In the case of scatter gather DMA, there are multiple noncontiguous sources or destinations that are required to effectively carry out an overall DMA transaction. For example, a packet may need to be transmitted off of the device and the packet elements, including the header, payload, and trailer, exist in various noncontiguous locations in memory. Scatter gather DMA allows the segments to be concatenated together by using multiple TDs in a chain. The chain gathers the data from the multiple locations. A similar concept applies for the reception of data onto the device. Certain parts of the received data may need to be scattered to various locations in memory for software processing convenience. Each TD in the chain specifies the location for each discrete element in the chain.

4.3.4.7 Packet Queuing DMA

Packet queuing DMA is similar to scatter gather DMA but specifically refers to packet protocols. With these protocols, there may be separate configuration, data, and status phases associated with sending or receiving a packet.

For instance, to transmit a packet, a memory mapped configuration register can be written inside a peripheral, specifying the overall length of the ensuing data phase. The CPU can set up this configuration information anywhere in system memory and copy it with a simple TD to the peripheral. After the configuration phase, a data phase TD (or a series of data phase TDs) can begin (potentially using scatter gather). When the data phase TD(s) finish, a status phase TD can be invoked that reads some memory mapped status information from the peripheral and copies it to a location in system memory specified by the CPU for later inspection. Multiple sets of configuration, data, and status phase "subchains" can be strung together to create larger chains that transmit multiple packets in this way. A similar concept exists in the opposite direction to receive the packets.

4.3.4.8 Nested DMA

One TD may modify another TD, as the TD configuration space is memory mapped similar to any other peripheral. For example, a first TD loads a second TD's configuration and then calls the second TD. The second TD moves data as required by the application. When complete, the second TD calls the first TD, which again updates the second TD's configuration. This process repeats as often as necessary.

4.4 Interrupt Controller

The Cortex-M3 NVIC supports 16 system exceptions and 32 interrupts from peripherals, as shown in Table 4-5.

Table 4-5. Cortex-M3 Exceptions and Interrupts

Exception Number	Exception Type	Priority	Exception Table Address Offset	Function
			0x00	Starting value of R13 / MSP
1	Reset	-3 (highest)	0x04	Reset
2	NMI	-2	0x08	Non maskable interrupt
3	Hard fault	-1	0x0C	All classes of fault, when the corresponding fault handler cannot be activated because it is currently disabled or masked
4	MemManage	Programmable	0x10	Memory management fault, for example, instruction fetch from a nonexecutable region
5	Bus fault	Programmable	0x14	Error response received from the bus system; caused by an instruction prefetch abort or data access error
6	Usage fault	Programmable	0x18	Typically caused by invalid instructions or trying to switch to ARM mode
7 – 10	-	-	0x1C – 0x28	Reserved
11	SVC	Programmable	0x2C	System service call via SVC instruction
12	Debug monitor	Programmable	0x30	Debug monitor
13	-	-	0x34	Reserved
14	PendSV	Programmable	0x38	Deferred request for system service
15	SYSTICK	Programmable	0x3C	System tick timer
16 – 47	IRQ	Programmable	0x40 – 0x3FC	Peripheral interrupt request #0 - #31

Bit 0 of each exception vector indicates whether the exception is executed using ARM or Thumb instructions. Because the Cortex-M3 only supports Thumb instructions, this bit must always be 1. The Cortex-M3 non maskable interrupt (NMI) input can be routed to any pin, via the DSI, or disconnected from all pins. See "DSI Routing Interface Description" section on page 42.

The Nested Vectored Interrupt Controller (NVIC) handles interrupts from the peripherals, and passes the interrupt vectors to the CPU. It is closely integrated with the CPU for low latency interrupt handling. Features include:

- 32 interrupts. Multiple sources for each interrupt.
- Configurable number of priority levels: from 3 to 8.
- Dynamic reprioritization of interrupts.
- Priority grouping. This allows selection of preempting and non preempting interrupt levels.
- Support for tail-chaining, and late arrival, of interrupts. This enables back-to-back interrupt processing without the overhead of state saving and restoration between interrupts.
- Processor state automatically saved on interrupt entry, and restored on interrupt exit, with no instruction overhead.

If the same priority level is assigned to two or more interrupts, the interrupt with the lower vector number is executed first. Each interrupt vector may choose from three interrupt sources: Fixed Function, DMA, and UDB. The fixed function interrupts are direct connections to the most common interrupt sources and provide the lowest resource cost connection. The DMA interrupt sources provide direct connections to the two DMA interrupt sources provided per DMA channel. The third interrupt source for vectors is from the UDB digital routing array. This allows any digital signal available to the UDB array to be used as an interrupt source. All interrupt sources may be routed to any interrupt vector using the UDB interrupt source connections.

Table 4-6. Interrupt Vector Table

Interrupt #	Cortex-M3 Exception #	Fixed Function	DMA	UDB
0	16	Low voltage detect (LVD)	phub_termout0[0]	udb_intr[0]
1	17	Cache/ECC	phub_termout0[1]	udb_intr[1]
2	18	Reserved	phub_termout0[2]	udb_intr[2]
3	19	Sleep (Pwr Mgr)	phub_termout0[3]	udb_intr[3]
4	20	PICU[0]	phub_termout0[4]	udb_intr[4]
5	21	PICU[1]	phub_termout0[5]	udb_intr[5]
6	22	PICU[2]	phub_termout0[6]	udb_intr[6]
7	23	PICU[3]	phub_termout0[7]	udb_intr[7]
8	24	PICU[4]	phub_termout0[8]	udb_intr[8]
9	25	PICU[5]	phub_termout0[9]	udb_intr[9]
10	26	PICU[6]	phub_termout0[10]	udb_intr[10]
11	27	PICU[12]	phub_termout0[11]	udb_intr[11]
12	28	PICU[15]	phub_termout0[12]	udb_intr[12]
13	29	Comparators Combined	phub_termout0[13]	udb_intr[13]
14	30	Switched Caps Combined	phub_termout0[14]	udb_intr[14]
15	31	I^2C	phub_termout0[15]	udb_intr[15]
16	32	CAN	phub_termout1[0]	udb_intr[16]
17	33	Timer/Counter0	phub_termout1[1]	udb_intr[17]
18	34	Timer/Counter1	phub_termout1[2]	udb_intr[18]
19	35	Timer/Counter2	phub_termout1[3]	udb_intr[19]
20	36	Timer/Counter3	phub_termout1[4]	udb_intr[20]
21	37	USB SOF Int	phub_termout1[5]	udb_intr[21]
22	38	USB Arb Int	phub_termout1[6]	udb_intr[22]
23	39	USB Bus Int	phub_termout1[7]	udb_intr[23]
24	40	USB Endpoint[0]	phub_termout1[8]	udb_intr[24]
25	41	USB Endpoint Data	phub_termout1[9]	udb_intr[25]
26	42	Reserved	phub_termout1[10]	udb_intr[26]
27	43	LCD	phub_termout1[11]	udb_intr[27]
28	44	DFB Int	phub_termout1[12]	udb_intr[28]
29	45	Decimator Int	phub_termout1[13]	udb_intr[29]
30	46	phub_err_int	phub_termout1[14]	udb_intr[30]
31	47	eeprom_fault_int	phub_termout1[15]	udb_intr[31]

5. Memory

5.1 Static RAM

CY8C58LP static RAM (SRAM) is used for temporary data storage. Code can be executed at full speed from the portion of SRAM that is located in the code space. This process is slower from SRAM above 0x20000000. The device provides up to 64 KB of SRAM. The CPU or the DMA controller can access all of SRAM. The SRAM can be accessed simultaneously by the Cortex-M3 CPU and the DMA controller if accessing different 32-KB blocks.

5.2 Flash Program Memory

Flash memory in PSoC devices provides nonvolatile storage for user firmware, user configuration data, bulk data storage, and optional ECC data. The main flash memory area contains up to 256 KB of user program space.

Up to an additional 32 KB of flash space is available for Error Correcting Codes (ECC). If ECC is not used this space can store device configuration data and bulk user data. User code may not be run out of the ECC flash memory section. ECC can correct one bit error and detect two bit errors per 8 bytes of firmware memory; an interrupt can be generated when an error is detected. The flash output is 9 bytes wide with 8 bytes of data and 1 byte of ECC data.

The CPU or DMA controller read both user code and bulk data located in flash through the cache controller. This provides higher CPU performance. If ECC is enabled, the cache controller also performs error checking and correction.

Flash programming is performed through a special interface and preempts code execution out of flash. Code execution may be done out of SRAM during flash programming.

The flash programming interface performs flash erasing, programming and setting code protection levels. Flash in-system serial programming (ISSP), typically used for production programming, is possible through both the SWD and JTAG interfaces. In-system programming, typically used for bootloaders, is also possible using serial interfaces such as I^2C, USB, UART, and SPI, or any communications protocol.

5.3 Flash Security

All PSoC devices include a flexible flash protection model that prevents access and visibility to on-chip flash memory. This prevents duplication or reverse engineering of proprietary code. Flash memory is organized in blocks, where each block contains 256 bytes of program or data and 32 bytes of ECC or configuration data.

The device offers the ability to assign one of four protection levels to each row of flash. Table 5-1 lists the protection modes available. Flash protection levels can only be changed by performing a complete flash erase. The Full Protection and Field Upgrade settings disable external access (through a debugging tool such as PSoC Creator, for example). If your application requires code update through a boot loader, then use the Field Upgrade setting. Use the Unprotected setting only when no security is needed in your application. The PSoC device also offers an advanced security feature called Device Security which permanently disables all test, programming, and debug ports, protecting your application from external access (see the "Device Security" section on page 60). For more information on how to take full advantage of the security features in PSoC, see the PSoC 5 TRM.

Table 5-1. Flash Protection

Protection Setting	Allowed	Not Allowed
Unprotected	External read and write + internal read and write	–
Factory Upgrade	External write + internal read and write	External read
Field Upgrade	Internal read and write	External read and write
Full Protection	Internal read	External read and write + internal write

Disclaimer

Note the following details of the flash code protection features on Cypress devices.

Cypress products meet the specifications contained in their particular Cypress datasheets. Cypress believes that its family of products is one of the most secure families of its kind on the market today, regardless of how they are used. There may be methods, unknown to Cypress, that can breach the code protection features. Any of these methods, to our knowledge, would be dishonest and possibly illegal. Neither Cypress nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Cypress is willing to work with the customer who is concerned about the integrity of their code. Code protection is constantly evolving. We at Cypress are committed to continuously improving the code protection features of our products.

5.4 EEPROM

PSoC EEPROM memory is a byte addressable nonvolatile memory. The CY8C58LP has 2 KB of EEPROM memory to store user data. Reads from EEPROM are random access at the byte level. Reads are done directly; writes are done by sending write commands to an EEPROM programming interface. CPU code execution can continue from flash during EEPROM writes. EEPROM is erasable and writeable at the row level. The EEPROM is divided into 128 rows of 16 bytes each.

The CPU can not execute out of EEPROM. There is no ECC hardware associated with EEPROM. If ECC is required it must be handled in firmware.

It can take as much as 20 milliseconds to write to EEPROM or flash. During this time the device should not be reset, or unexpected changes may be made to portions of EEPROM or flash. Reset sources (see *Section 6.3.1*) include XRES pin, software reset, and watchdog; care should be taken to make sure that these are not inadvertently activated. In addition, the low voltage detect circuits should be configured to generate an interrupt instead of a reset.

6. System Integration

6.1 Clocking System

The clocking system generates, divides, and distributes clocks throughout the PSoC system. For the majority of systems, no external crystal is required. The IMO and PLL together can generate up to a 80 MHz clock, accurate to ±1% over voltage and temperature. Additional internal and external clock sources allow each design to optimize accuracy, power, and cost. All of the system clock sources can be used to generate other clock frequencies in the 16-bit clock dividers and UDBs for anything you want, for example a UART baud rate generator.

Clock generation and distribution is automatically configured through the PSoC Creator IDE graphical interface. This is based on the complete system's requirements. It greatly speeds the design process. PSoC Creator allows designers to build clocking systems with minimal input. The designer can specify desired clock frequencies and accuracies, and the software locates or builds a clock that meets the required specifications. This is possible because of the programmability inherent in PSoC.

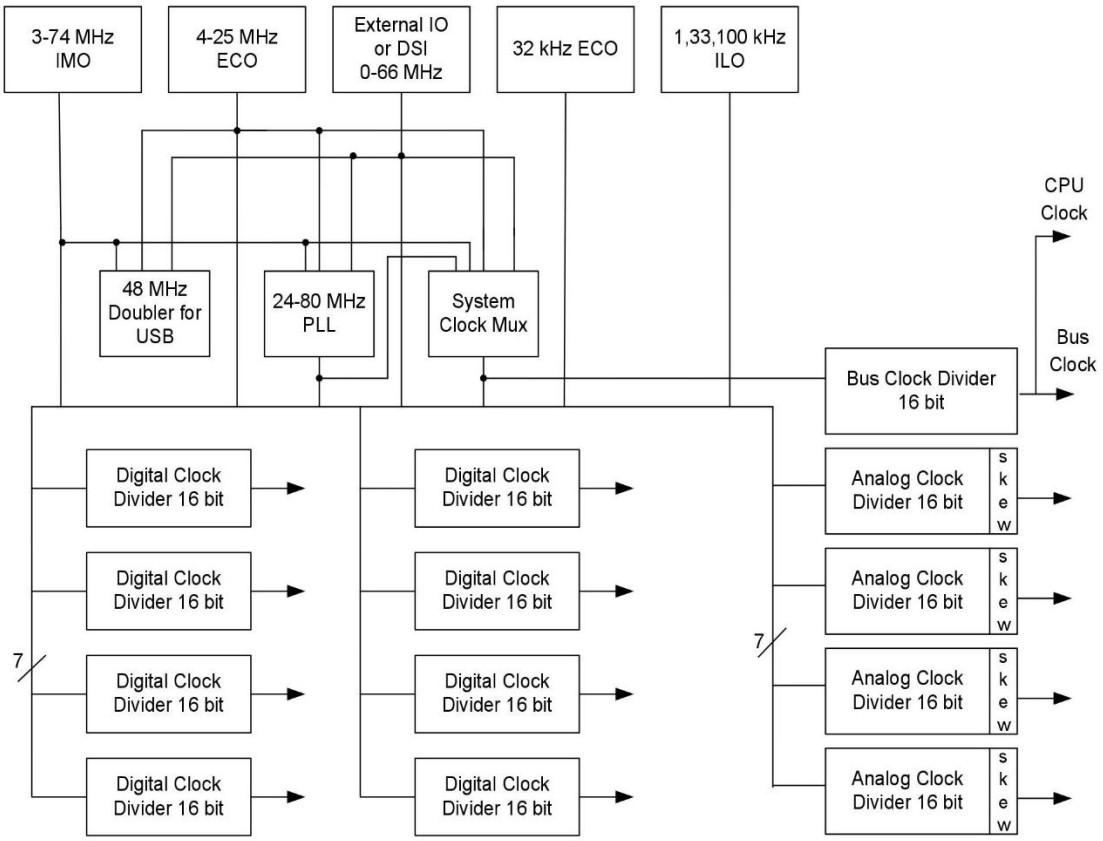
Key features of the clocking system include:

- Seven general purpose clock sources
  - 3- to 74-MHz IMO, ±1% at 3 MHz
  - 4- to 25-MHz external crystal oscillator (MHzECO)
  - Clock doubler provides a doubled clock frequency output for the USB block, see USB Clock Domain on page 25.
  - DSI signal from an external I/O pin or other logic
  - 24- to 80-MHz fractional phase-locked loop (PLL) sourced from IMO, MHzECO, or DSI
  - 1-kHz, 33-kHz, 100-kHz ILO for watchdog timer (WDT) and Sleep Timer
  - 32.768-kHz external crystal oscillator (ECO) for RTC
- IMO has a USB mode that auto-locks to the USB bus clock requiring no external crystal for USB. (USB equipped parts only)
- Independently sourced clock in all clock dividers
- Eight 16-bit clock dividers for the digital system
- Four 16-bit clock dividers for the analog system
- Dedicated 16-bit divider for the CPU bus and CPU clock
- Automatic clock configuration in PSoC Creator

Table 6-1. Oscillator Summary

Source	Fmin	Tolerance at Fmin	Fmax	Tolerance at Fmax	Startup Time
IMO	3 MHz	±1% over voltage and temperature	74 MHz	±7%	13 µs max
MHzECO	4 MHz	Crystal dependent	25 MHz	Crystal dependent	5 ms typ, max is crystal dependent
DSI	0 MHz	Input dependent	66 MHz	Input dependent	Input dependent
PLL	24 MHz	Input dependent	80 MHz	Input dependent	250 µs max
Doubler	48 MHz	Input dependent	48 MHz	Input dependent	1 µs max
ILO	1 kHz	–50%, +100%	100 kHz	–55%, +100%	15 ms max in lowest power mode
kHzECO	32 kHz	Crystal dependent	32 kHz	Crystal dependent	500 ms typ, max is crystal dependent

Figure 6-1. Clocking Subsystem



6.1.1 Internal Oscillators

6.1.1.1 Internal Main Oscillator

In most designs the IMO is the only clock source required, due to its ±1% accuracy. The IMO operates with no external components and outputs a stable clock. A factory trim for each frequency range is stored in the device. With the factory trim, tolerance varies from ±1% at 3 MHz, up to ±7% at 74 MHz. The IMO, in conjunction with the PLL, allows generation of CPU and system clocks up to the device's maximum frequency (see USB Clock Domain on page 25). The IMO provides clock outputs at 3, 6, 12, 24, 48, and 74 MHz.

6.1.1.2 Clock Doubler

The clock doubler outputs a clock at twice the frequency of the input clock. The doubler works at input frequency of 24 MHz, providing 48 MHz for the USB. It can be configured to use a clock from the IMO, MHzECO, or the DSI (external pin).

6.1.1.3 Phase-Locked Loop

The PLL allows low frequency, high accuracy clocks to be multiplied to higher frequencies. This is a tradeoff between higher clock frequency and accuracy and, higher power consumption and increased startup time.

The PLL block provides a mechanism for generating clock frequencies based upon a variety of input sources. The PLL outputs clock frequencies in the range of 24 to 80 MHz. Its input and feedback dividers supply 4032 discrete ratios to create almost any desired system clock frequency. The accuracy of the PLL output depends on the accuracy of the PLL input source. The most common PLL use is to multiply the IMO clock at 3 MHz, where it is most accurate, to generate the CPU and system clocks up to the device's maximum frequency.

The PLL achieves phase lock within 250 µs (verified by bit setting). It can be configured to use a clock from the IMO, MHzECO, or DSI (external pin). The PLL clock source can be used until lock is complete and signaled with a lock bit. The lock signal can be routed through the DSI to generate an interrupt. Disable the PLL before entering low power modes.

6.1.1.4 Internal Low-Speed Oscillator

The ILO provides clock frequencies for low power consumption, including the watchdog timer, and sleep timer. The ILO generates up to three different clocks: 1 kHz, 33 kHz, and 100 kHz.

The 1-kHz clock (CLK1K) is typically used for a background 'heartbeat' timer. This clock inherently lends itself to low power supervisory operations such as the watchdog timer and long sleep intervals using the central timewheel (CTW).

The central timewheel is a 1 kHz, free running, 13-bit counter clocked by the ILO. The central timewheel is always enabled except in hibernate mode and when the CPU is stopped during debug on chip mode. It can be used to generate periodic interrupts for timing purposes or to wake the system from a low power mode. Firmware can reset the central timewheel.

The central timewheel can be programmed to wake the system periodically and optionally issue an interrupt. This enables flexible, periodic wakeups from low power modes or coarse timing applications. Systems that require accurate timing should use the RTC capability instead of the central timewheel.

The 100-kHz clock (CLK100K) can be used as a low power system clock to run the CPU. It can also generate time intervals using the fast timewheel.

The fast timewheel is a 5-bit counter, clocked by the 100-kHz clock. It features programmable settings and automatically resets when the terminal count is reached. An optional interrupt can be generated each time the terminal count is reached. This enables flexible, periodic interrupts of the CPU at a higher rate than is allowed using the central timewheel.

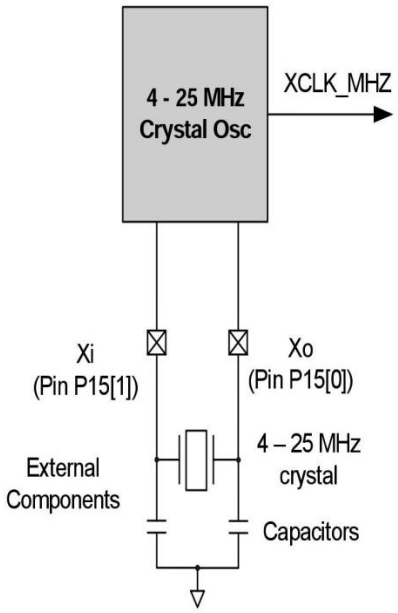
The 33-kHz clock (CLK33K) comes from a divide-by-3 operation on CLK100K. This output can be used as a reduced accuracy version of the 32.768-kHz ECO clock with no need for a crystal.

6.1.2 External Oscillators

6.1.2.1 MHz External Crystal Oscillator

The MHzECO provides high frequency, high precision clocking using an external crystal (see Figure 6-2). It supports a wide variety of crystal types, in the range of 4 to 25 MHz. When used in conjunction with the PLL, it can generate CPU and system clocks up to the device's maximum frequency (see Phase-Locked Loop on page 24). The GPIO pins connecting to the external crystal and capacitors are fixed. MHzECO accuracy depends on the crystal chosen.

Figure 6-2. MHzECO Block Diagram



6.2.1 Power Modes

PSoC 5LP devices have four different power modes, as shown in Table 6-2 and Table 6-3. The power modes allow a design to easily provide required functionality and processing power while simultaneously minimizing power consumption and maximizing battery life in low power and portable devices.

PSoC 5LP power modes, in order of decreasing power consumption are:

- Active
- Alternate active
- Sleep
- Hibernate

Active is the main processing mode. Its functionality is configurable. Each power controllable subsystem is enabled or disabled by using separate power configuration template registers. In alternate active mode, fewer subsystems are enabled, reducing power. In sleep mode most resources are disabled regardless of the template settings. Sleep mode is optimized to provide timed sleep intervals and RTC functionality. The lowest power mode is hibernate, which retains register and SRAM state, but no clocks, and allows wakeup only from I/O pins. Figure 6-5 illustrates the allowable transitions between power modes. Sleep and hibernate modes should not be entered until all VDDIO supplies are at valid voltage levels.

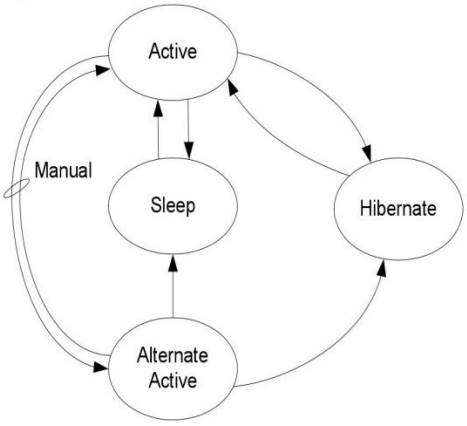
Table 6-2. Power Modes

Power Modes	Description	Entry Condition	Wakeup Source	Active Clocks	Regulator
Active	Primary mode of operation, all peripherals available (programmable)	Wakeup, reset, manual register entry	Any interrupt	Any (programmable)	All regulators available. Digital and analog regulators can be disabled if external regulation used.
Alternate Active	Similar to Active mode, and is typically configured to have fewer peripherals active to reduce power. One possible configuration is to use the UDBs for processing, with the CPU turned off	Manual register entry	Any interrupt	Any (programmable)	All regulators available. Digital and analog regulators can be disabled if external regulation used.
Sleep	All subsystems automatically disabled	Manual register entry	Comparator, PICU, I^2C, RTC, CTW, LVD	ILO/kHzECO	Both digital and analog regulators buzzed. Digital and analog regulators can be disabled if external regulation used.
Hibernate	All subsystems automatically disabled Lowest power consuming mode with all peripherals and internal regulators disabled, except hibernate regulator is enabled Configuration and memory contents retained	Manual register entry	PICU		Only hibernate regulator active.

Table 6-3. Power Modes Wakeup Time and Power Consumption

Sleep Modes	Wakeup Time	Current (Typ)	Code Execution	Digital Resources	Analog Resources	Clock Sources Available	Wakeup Sources	Reset Sources
Active	–	3.1 mA[6]	Yes	All	All	All	–	All
Alternate Active	–	–	User defined	All	All	All	–	All
Sleep	<25 µs	2 µA	No	I^2C	Comparator	ILO/kHzECO	Comparator, PICU, I^2C, RTC, CTW, LVD	XRES, LVD, WDR
Hibernate	<200 µs	300 nA	No	None	None	None	PICU	XRES

Figure 6-5. Power Mode Transitions



6.2.1.1 Active Mode

Active mode is the primary operating mode of the device. When in active mode, the active configuration template bits control which available resources are enabled or disabled. When a resource is disabled, the digital clocks are gated, analog bias currents are disabled, and leakage currents are reduced as appropriate. User firmware can dynamically control subsystem power by setting and clearing bits in the active configuration template. The CPU can disable itself, in which case the CPU is automatically reenabled at the next wakeup event.

When a wakeup event occurs, the global mode is always returned to active, and the CPU is automatically enabled, regardless of its template settings. Active mode is the default global power mode upon boot.

6.2.1.2 Alternate Active Mode

Alternate Active mode is very similar to Active mode. In alternate active mode, fewer subsystems are enabled, to reduce power consumption. One possible configuration is to turn off the CPU and flash, and run peripherals at full speed.

6.2.1.3 Sleep Mode

Sleep mode reduces power consumption when a resume time of 15 µs is acceptable. The wake time is used to ensure that the regulator outputs are stable enough to directly enter active mode.

6.2.1.4 Hibernate Mode

In hibernate mode nearly all of the internal functions are disabled. Internal voltages are reduced to the minimal level to keep vital systems alive. Configuration state is preserved in hibernate mode and SRAM memory is retained. GPIOs configured as digital outputs maintain their previous values and external GPIO pin interrupt settings are preserved. The device can only return from hibernate mode in response to an external I/O interrupt. The resume time from hibernate mode is less than 100 µs.

To achieve an extremely low current, the hibernate regulator has limited capacity. This limits the frequency of any signal present on the input pins; no GPIO should toggle at a rate greater than 10 kHz while in hibernate mode. If pins must be toggled at a high rate while in a low power mode, use sleep mode instead.

6.2.1.5 Wakeup Events

Wakeup events are configurable and can come from an interrupt or device reset. A wakeup event restores the system to active mode. Firmware enabled interrupt sources include internally generated interrupts, power supervisor, central timewheel, and I/O interrupts. Internal interrupt sources can come from a variety of peripherals, such as analog comparators and UDBs. The central timewheel provides periodic interrupts to allow the system to wake up, poll peripherals, or perform real-time functions. Reset event sources include the external reset I/O pin (XRES), WDT, and Precision Reset (PRES).

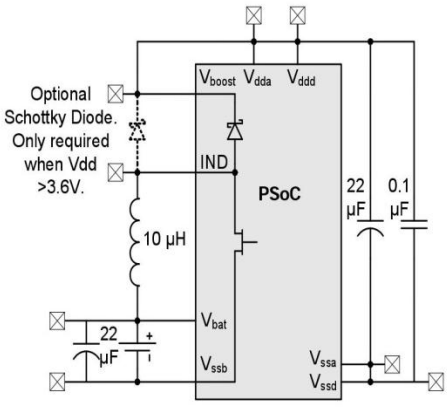
6.2.2 Boost Converter

Applications that use a supply voltage of less than 1.71 V, such as single cell battery supplies, may use the on-chip boost converter. The boost converter may also be used in any system that requires a higher operating voltage than the supply provides. For instance, this includes driving 5.0 V LCD glass in a 3.3 V system. The boost converter accepts an input voltage as low as 0.5 V. With one low cost inductor it produces a selectable output voltage sourcing enough current to operate the PSoC and other on-board components.

The boost converter accepts an input voltage VBAT from 0.5 V to 3.6 V, and can start up with VBAT as low as 0.5 V. The converter provides a user configurable output voltage of 1.8 to 5.0 V (VBOOST). VBAT is typically less than VBOOST; if VBAT is greater than or equal to VBOOST, then VBOOST will be the same as VBAT. The block can deliver up to 75 mA (IBOOST) depending on configuration.

Four pins are associated with the boost converter: VBAT, VSSB, VBOOST, and Ind. The boosted output voltage is sensed at the VBOOST pin and must be connected directly to the chip's supply inputs. An inductor is connected between the VBAT and Ind pins. You can optimize the inductor value to increase the boost converter efficiency based on input voltage, output voltage, current and switching frequency.

Figure 6-6. Application for Boost Converter



The switching frequency is set to 400 kHz using an oscillator in the boost converter block. The VBOOST is limited to 4 × VBAT.

The boost converter can be operated in two different modes: active and sleep. Active mode is the normal mode of operation where the boost regulator actively generates a regulated output voltage.

The boost typically draws 250 µA in active mode and 25 µA in sleep mode. The boost operating modes must be used in conjunction with chip power modes to minimize total power consumption. Table 6-4 lists the boost power modes available in different chip power modes.

Table 6-4. Chip and Boost Power Modes Compatibility

Chip Power Modes	Boost Power Modes
Chip -Active or alternate active mode	Boost must be operated in its active mode.
Chip -Sleep mode	Boost can be operated in either active or sleep mode. In boost sleep mode, the chip must wake up periodically for boost active-mode refresh.
Chip-Hibernate mode	Boost can be operated in either active or sleep mode. However, it is recommended not to use the boost with chip hibernate mode due to the higher current consumption. In boost sleep mode, the chip must wake up periodically for boost active-mode refresh.

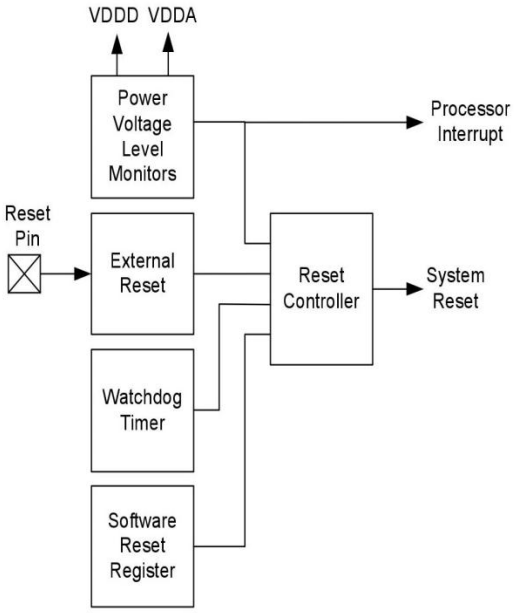
If the boost converter is not used, tie the VBAT, VSSB, and VBOOST pins to ground and leave the Ind pin unconnected.

6.3 Reset

CY8C58LP has multiple internal and external reset sources available. The reset sources are:

- Power source monitoring - The analog and digital power voltages, VDDA, VDDD, VCCA, and VCCD are monitored in several different modes during power up, active mode, and sleep mode (buzzing). If any of the voltages goes outside predetermined ranges then a reset is generated. The monitors are programmable to generate an interrupt to the processor under certain conditions before reaching the reset thresholds.
- External - The device can be reset from an external source by pulling the reset pin (XRES) low. The XRES pin includes an internal pull-up to VDDIO1. VDDD, VDDA, and VDDIO1 must all have voltage applied before the part comes out of reset.
- Watchdog timer - A watchdog timer monitors the execution of instructions by the processor. If the watchdog timer is not reset by firmware within a certain period of time, the watchdog timer generates a reset.
- Software - The device can be reset under program control.

Figure 6-7. Resets



The term **system reset** indicates that the processor as well as analog and digital peripherals and registers are reset.

A reset status register shows some of the resets or power voltage monitoring interrupts. The program may examine this register to detect and report certain exception conditions. This register is cleared after a power-on reset. For details see the Technical Reference Manual.

6.3.1 Reset Sources

6.3.1.1 Power Voltage Level Monitors

- IPOR - Initial Power-on-Reset

  At initial power on, IPOR monitors the power voltages V_{DDD}, V_{DDA}, V_{CCD} and V_{CCA}. The trip level is not precise. It is set to approximately 1 volt, which is below the lowest specified operating voltage but high enough for the internal circuits to be reset and to hold their reset state. The monitor generates a reset pulse that is at least 150 ns wide. It may be much wider if one or more of the voltages ramps up slowly.

  If after the IPOR triggers either V_{DDX} drops back below the trigger point, in a non-monotonic fashion, it must remain below that point for at least 10 µs. The hysteresis of the IPOR trigger point is typically 100 mV.

  After boot, the IPOR circuit is disabled and voltage supervision is handed off to the precise low-voltage reset (PRES) circuit.

- PRES - Precise Low-Voltage Reset

  This circuit monitors the outputs of the analog and digital internal regulators after power up. The regulator outputs are compared to a precise reference voltage. The response to a PRES trip is identical to an IPOR reset.

  In normal operating mode, the program cannot disable the digital PRES circuit. The analog regulator can be disabled, which also disables the analog portion of the PRES. The PRES circuit is disabled automatically during sleep and hibernate modes, with one exception: During sleep mode the regulators are periodically activated (buzzed) to provide supervisory

6.4.1 Drive Modes

Each GPIO and SIO pin is individually configurable into one of the eight drive modes listed in Table 6-6. Three configuration bits are used for each pin (DM[2:0]) and set in the PRTxDM[2:0] registers. Figure 6-11 depicts a simplified pin view based on each of the eight drive modes. Table 6-6 shows the I/O pin's drive state based on the port data register value or digital array signal if bypass mode is selected. Note that the actual I/O pin voltage is determined by a combination of the selected drive mode and the load at the pin. For example, if a GPIO pin is configured for resistive pull-up mode and driven high while the pin is floating, the voltage measured at the pin is a high logic state. If the same GPIO pin is externally tied to ground then the voltage unmeasured at the pin is a low logic state.

Figure 6-11. Drive Mode

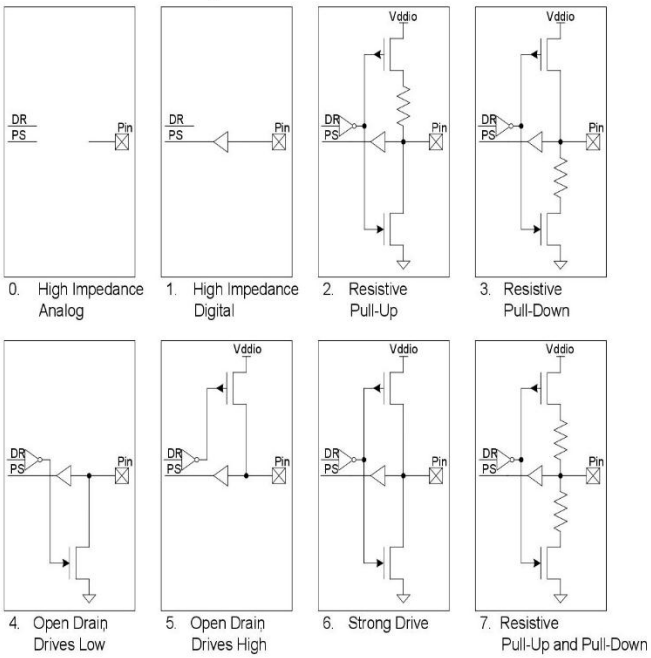


Table 6-6. Drive Modes

Diagram	Drive Mode	PRTxDM2	PRTxDM1	PRTxDM0	PRTxDR = 1	PRTxDR = 0
0	High impedance analog	0	0	0	High-Z	High-Z
1	High Impedance digital	0	0	1	High-Z	High-Z
2	Resistive pull-up[8]	0	1	0	Res High (5K)	Strong Low
3	Resistive pull-down[8]	0	1	1	Strong High	Res Low (5K)
4	Open drain, drives low	1	0	0	High-Z	Strong Low
5	Open drain, drive high	1	0	1	Strong High	High-Z
6	Strong drive	1	1	0	Strong High	Strong Low
7	Resistive pull-up and pull-down[8]	1	1	1	Res High (5K)	Res Low (5K)

7. Digital Subsystem

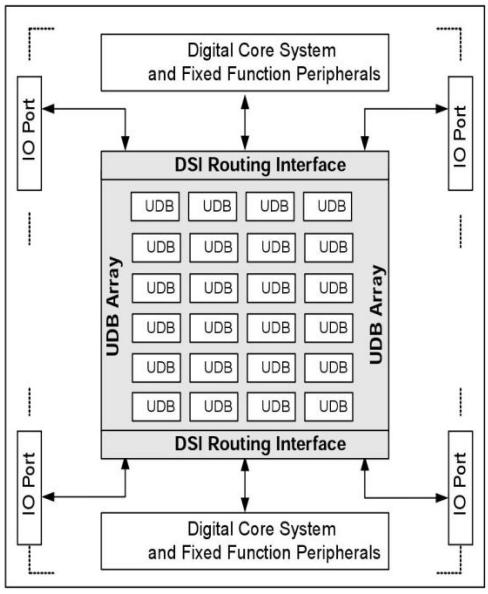
The digital programmable system creates application specific combinations of both standard and advanced digital peripherals and custom logic functions. These peripherals and logic are then interconnected to each other and to any pin on the device, providing a high level of design flexibility and IP security.

The features of the digital programmable system are outlined here to provide an overview of capabilities and architecture. You do not need to interact directly with the programmable digital system at the hardware and register level. PSoC Creator provides a high level schematic capture graphical interface to automatically place and route resources similar to PLDs.

The main components of the digital programmable system are:

- Universal Digital Blocks (UDB) - These form the core functionality of the digital programmable system. UDBs are a collection of uncommitted logic (PLD) and structural logic (Datapath) optimized to create all common embedded peripherals and customized functionality that are application or design specific.
- Universal Digital Block array - UDB blocks are arrayed within a matrix of programmable interconnect. The UDB array structure is homogeneous and allows for flexible mapping of digital functions onto the array. The array supports extensive and flexible routing interconnects between UDBs and the Digital System Interconnect.
- Digital System Interconnect (DSI) - Digital signals from Universal Digital Blocks (UDBs), fixed function peripherals, I/O pins, interrupts, DMA, and other system core signals are attached to the Digital System Interconnect to implement full featured device connectivity. The DSI allows any digital function to any pin or other feature routability when used with the Universal Digital Block array.

Figure 7-1. CY8C58LP Digital Programmable Architecture



7.1 Example Peripherals

The flexibility of the CY8C58LP family's UDBs and analog blocks allow the user to create a wide range of components (peripherals). The most common peripherals were built and characterized by Cypress and are shown in the PSoC Creator component catalog, however, users may also create their own custom components using PSoC Creator. Using PSoC Creator, users may also create their own components for reuse within their organization, for example sensor interfaces, proprietary algorithms, and display interfaces.

The number of components available through PSoC Creator is too numerous to list in the datasheet, and the list is always growing. An example of a component available for use in CY8C58LP family, but, not explicitly called out in this datasheet is the UART component.

7.1.1 Example Digital Components

The following is a sample of the digital components available in PSoC Creator for the CY8C58LP family. The exact amount of hardware resources (UDBs, routing, RAM, flash) used by a component varies with the features selected in PSoC Creator for the component.

- Communications
  - I^2C
  - UART
  - SPI
- Functions
  - EMIF
  - PWMs
  - Timers
  - Counters
- Logic
  - NOT
  - OR
  - XOR
  - AND

7.1.2 Example Analog Components

The following is a sample of the analog components available in PSoC Creator for the CY8C58LP family. The exact amount of hardware resources (SC/CT blocks, routing, RAM, flash) used by a component varies with the features selected in PSoC Creator for the component.

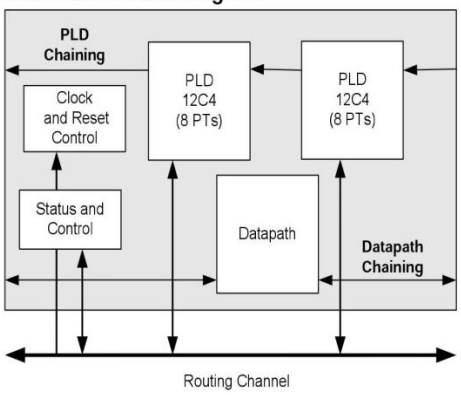
- Amplifiers
  - TIA
  - PGA
  - opamp
- ADCs
  - Delta-Sigma
  - Successive Approximation (SAR)
- DACs
  - Current
  - Voltage
  - PWM
- Comparators
- Mixers

7.2 Universal Digital Block

The Universal Digital Block (UDB) represents an evolutionary step to the next generation of PSoC embedded digital peripheral functionality. The architecture in first generation PSoC digital blocks provides coarse programmability in which a few fixed functions with a small number of options are available. The new UDB architecture is the optimal balance between configuration granularity and efficient implementation. A cornerstone of this approach is to provide the ability to customize the devices digital operation to match application requirements.

To achieve this, UDBs consist of a combination of uncommitted logic (PLD), structured logic (Datapath), and a flexible routing scheme to provide interconnect between these elements, I/O connections, and other peripherals. UDB functionality ranges from simple self contained functions that are implemented in one UDB, or even a portion of a UDB (unused resources are available for other functions), to more complex functions that require multiple UDBs. Examples of basic functions are timers, counters, CRC generators, PWMs, dead band generators, and communications functions, such as UARTs, SPI, and I^2C. Also, the PLD blocks and connectivity provide full featured general purpose programmable logic within the limits of the available resources.

Figure 7-2. UDB Block Diagram



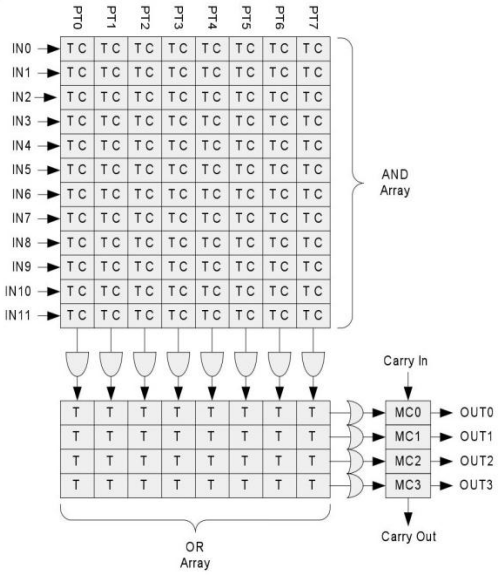
The main component blocks of the UDB are:

- PLD blocks - There are two small PLDs per UDB. These blocks take inputs from the routing array and form registered or combinational sum-of-products logic. PLDs are used to implement state machines, state bits, and combinational logic equations. PLD configuration is automatically generated from graphical primitives.
- Datapath Module - This 8-bit wide datapath contains structured logic to implement a dynamically configurable ALU, a variety of compare configurations and condition generation. This block also contains input/output FIFOs, which are the primary parallel data interface between the CPU/DMA system and the UDB.
- Status and Control Module - The primary role of this block is to provide a way for CPU firmware to interact and synchronize with UDB operation.
- Clock and Reset Module - This block provides the UDB clocks and reset selection and control.

7.2.1 PLD Module

The primary purpose of the PLD blocks is to implement logic expressions, state machines, sequencers, look up tables, and decoders. In the simplest use model, consider the PLD blocks as a standalone resource onto which general purpose RTL is synthesized and mapped. The more common and efficient use model is to create digital functions from a combination of PLD and datapath blocks, where the PLD implements only the random logic and state portion of the function while the datapath (ALU) implements the more structured elements.

Figure 7-3. PLD 12C4 Structure

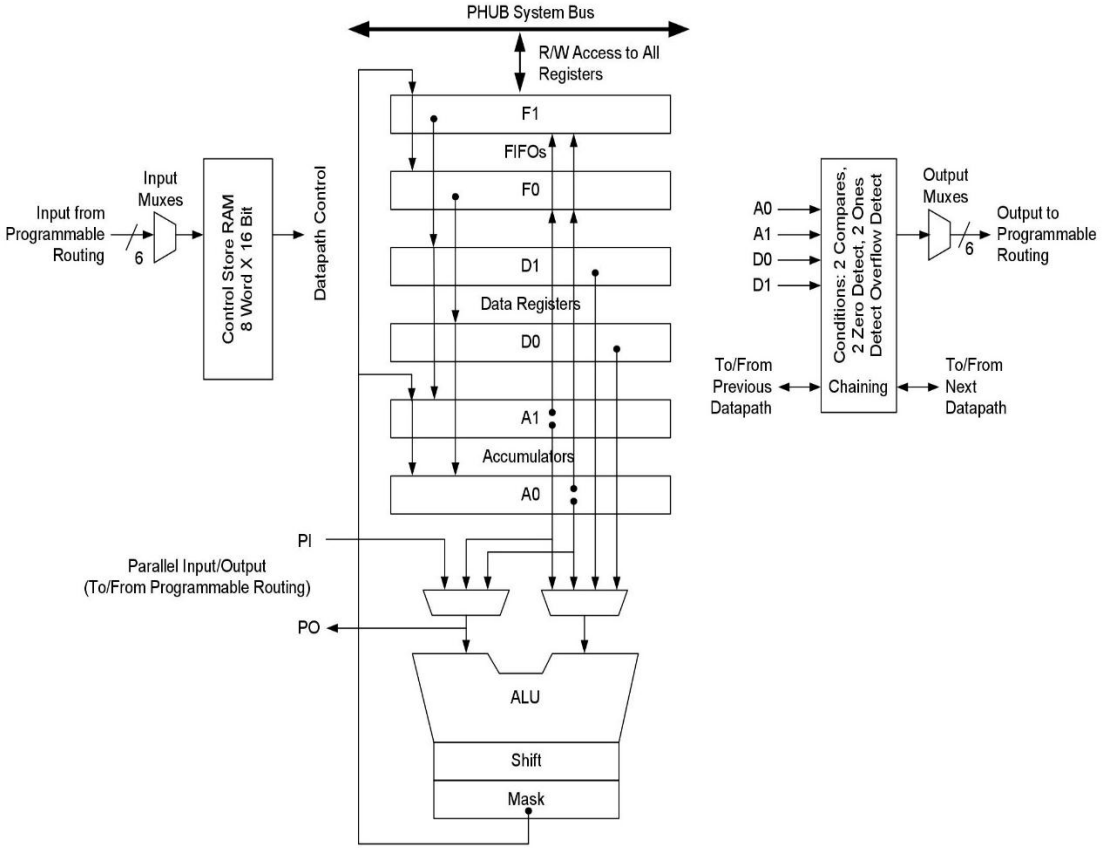


One 12C4 PLD block is shown in Figure 7-3. This PLD has 12 inputs, which feed across eight product terms. Each product term (AND function) can be from 1 to 12 inputs wide, and in a given product term, the true (T) or complement (C) of each input can be selected. The product terms are summed (OR function) to create the PLD outputs. A sum can be from 1 to 8 product terms wide. The 'C' in 12C4 indicates that the width of the OR gate (in this case 8) is constant across all outputs (rather than variable as in a 22V10 device). This PLA like structure gives maximum flexibility and insures that all inputs and outputs are permutable for ease of allocation by the software tools. There are two 12C4 PLDs in each UDB.

7.2.2 Datapath Module

The datapath contains an 8-bit single cycle ALU, with associated compare and condition generation logic. This datapath block is optimized to implement embedded functions, such as timers, counters, integrators, PWMs, PRS, CRC, shifters and dead band generators, and many others.

Figure 7-4. Datapath Top Level



7.2.2.1 *Working Registers*

The datapath contains six primary working registers, which are accessed by CPU firmware or DMA during normal operation.

Table 7-1. Working Datapath Registers

Name	Function	Description
A0 and A1	Accumulators	These are sources and sinks for the ALU and also sources for the compares.
D0 and D1	Data Registers	These are sources for the ALU and sources for the compares.
F0 and F1	FIFOs	These are the primary interface to the system bus. They can be a data source for the data registers and accumulators or they can capture data from the accumulators or ALU. Each FIFO is four bytes deep.

7.2.2.2 *Dynamic Datapath Configuration RAM*

Dynamic configuration is the ability to change the datapath function and internal configuration on a cycle-by-cycle basis, under sequencer control. This is implemented using the 8-word x 16-bit configuration RAM, which stores eight unique 16-bit wide configurations. The address input to this RAM controls the sequence, and can be routed from any block connected to the UDB routing matrix, most typically PLD logic, I/O pins, or from the outputs of this or other datapath blocks.

ALU

The ALU performs eight general purpose functions. They are:

- Increment
- Decrement
- Add
- Subtract
- Logical AND
- Logical OR
- Logical XOR
- Pass, used to pass a value through the ALU to the shift register, mask, or another UDB register

Independent of the ALU operation, these functions are available:

- Shift left
- Shift right
- Nibble swap
- Bitwise OR mask

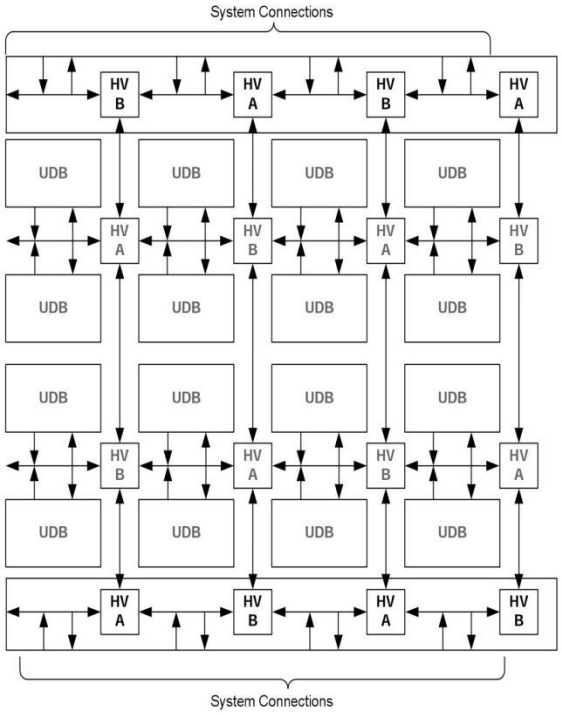
7.2.3.2 Clock Generation

Each subcomponent block of a UDB including the two PLDs, the datapath, and Status and Control, has a clock selection and control block. This promotes a fine granularity with respect to allocating clocking resources to UDB component blocks and allows unused UDB resources to be used by other functions for maximum system efficiency.

7.3 UDB Array Description

Figure 7-7 shows an example of a 16 UDB array. In addition to the array core, there are a DSI routing interfaces at the top and bottom of the array. Other interfaces that are not explicitly shown include the system interfaces for bus and clock distribution. The UDB array includes multiple horizontal and vertical routing channels each comprised of 96 wires. The wire connections to UDBs, at horizontal/vertical intersection and at the DSI interface are highly permutable providing efficient automatic routing in PSoC Creator. Additionally the routing allows wire by wire segmentation along the vertical and horizontal routing to further increase routing flexibility and capability.

Figure 7-7. Digital System Interface Structure

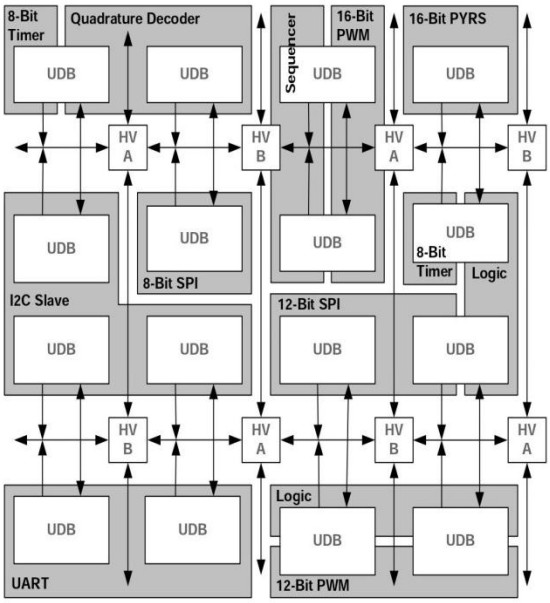


7.3.1 UDB Array Programmable Resources

Figure 7-8 shows an example of how functions are mapped into a bank of 16 UDBs. The primary programmable resources of the UDB are two PLDs, one datapath and one status/control register. These resources are allocated independently, because they have independently selectable clocks, and therefore unused blocks are allocated to other unrelated functions.

An example of this is the 8-bit Timer in the upper left corner of the array. This function only requires one datapath in the UDB, and therefore the PLD resources may be allocated to another function. A function such as a Quadrature Decoder may require more PLD logic than one UDB can supply and in this case can utilize the unused PLD blocks in the 8-bit Timer UDB. Programmable resources in the UDB array are generally homogeneous so functions can be mapped to arbitrary boundaries in the array.

Figure 7-8. Function Mapping Example in a Bank of UDBs



7.4 DSI Routing Interface Description

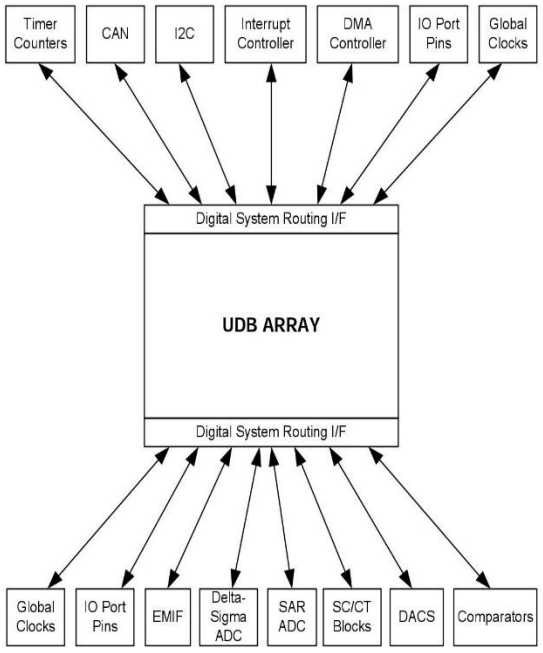
The DSI routing interface is a continuation of the horizontal and vertical routing channels at the top and bottom of the UDB array core. It provides general purpose programmable routing between device peripherals, including UDBs, I/Os, analog peripherals, interrupts, DMA and fixed function peripherals.

Figure 7-9 illustrates the concept of the digital system interconnect, which connects the UDB array routing matrix with other device peripherals. Any digital core or fixed function peripheral that needs programmable routing is connected to this interface.

Signals in this category include:

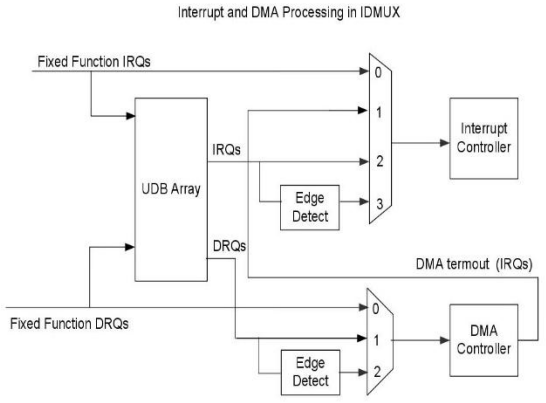
- Interrupt requests from all digital peripherals in the system.
- DMA requests from all digital peripherals in the system.
- Digital peripheral data signals that need flexible routing to I/Os.
- Digital peripheral data signals that need connections to UDBs.
- Connections to the interrupt and DMA controllers.
- Connection to I/O pins.
- Connection to analog system digital signals.

Figure 7-9. Digital System Interconnect



Interrupt and DMA routing is very flexible in the CY8C58LP programmable architecture. In addition to the numerous fixed function peripherals that can generate interrupt requests, any data signal in the UDB array routing can also be used to generate a request. A single peripheral may generate multiple independent interrupt requests simplifying system and firmware design. Figure 7-10 shows the structure of the IDMUX (Interrupt/DMA Multiplexer).

Figure 7-10. Interrupt and DMA Processing in the IDMUX



7.4.1 I/O Port Routing

There are a total of 20 DSI routes to a typical 8-bit I/O port, 16 for data and four for drive strength control.

When an I/O pin is connected to the routing, there are two primary connections available, an input and an output. In conjunction with drive strength control, this can implement a bidirectional I/O pin. A data output signal has the option to be single synchronized (pipelined) and a data input signal has the option to be double synchronized. The synchronization clock is the system clock (see Figure 6-1). Normally all inputs from pins are synchronized as this is required if the CPU interacts with the signal or any signal derived from it. Asynchronous inputs have rare uses. An example of this is a feed through of combinational PLD logic from input pins to output pins.

Figure 7-11. I/O Pin Synchronization Routing

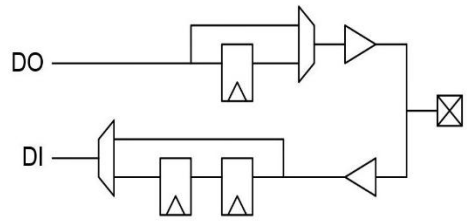
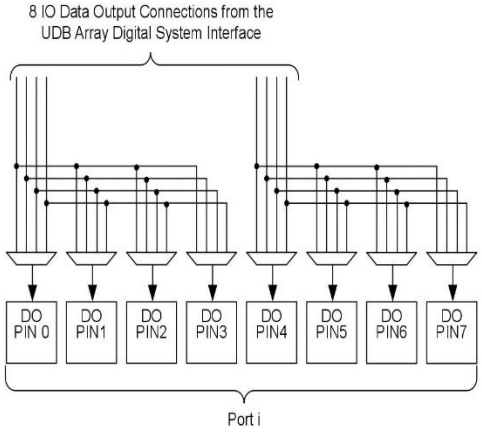
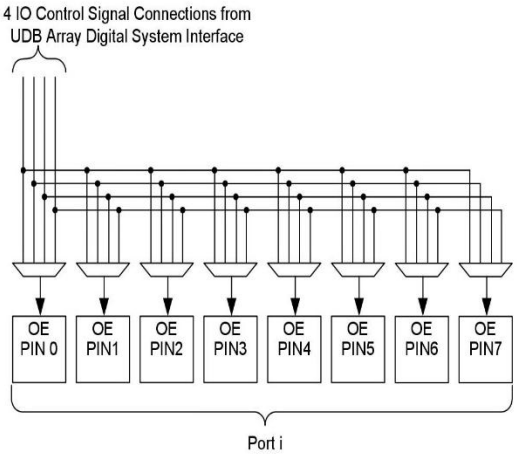


Figure 7-12. I/O Pin Output Connectivity



There are four more DSI connections to a given I/O port to implement dynamic output enable control of pins. This connectivity gives a range of options, from fully ganged 8-bits controlled by one signal, to up to four individually controlled pins. The output enable signal is useful for creating tri-state bidirectional pins and buses.

Figure 7-13. I/O Pin Output Enable Connectivity

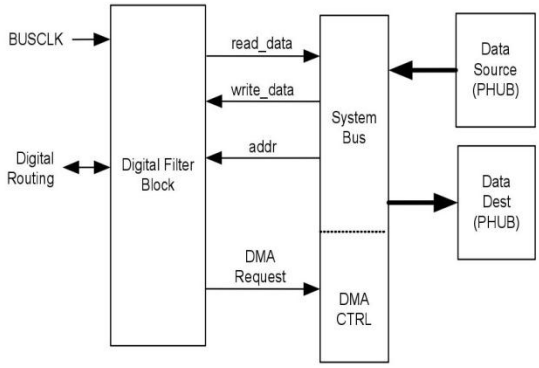


7.9 Digital Filter Block

Some devices in the CY8C58LP family of devices have a dedicated HW accelerator block used for digital filtering. The DFB has a dedicated multiplier and accumulator that calculates a 24-bit by 24-bit multiply accumulate in one system clock cycle. This enables the mapping of a direct form FIR filter that approaches a computation rate of one FIR tap for each clock cycle. The MCU can implement any of the functions performed by this block, but at a slower rate that consumes significant MCU bandwidth.

The PSoC Creator interface provides a wizard to implement FIR and IIR digital filters with coefficients for LPF, BPF, HPF, Notch and arbitrary shape filters. 64 pairs of data and coefficients are stored. This enables a 64 tap FIR filter or up to 4 16 tap filters of either FIR or IIR formulation.

Figure 7-19. DFB Application Diagram (pwr/gnd not shown)



The typical use model is for data to be supplied to the DFB over the system bus from another on-chip system data source such as an ADC. The data typically passes through main memory or is directly transferred from another chip resource through DMA. The DFB processes this data and passes the result to another on chip resource such as a DAC or main memory through DMA on the system bus.

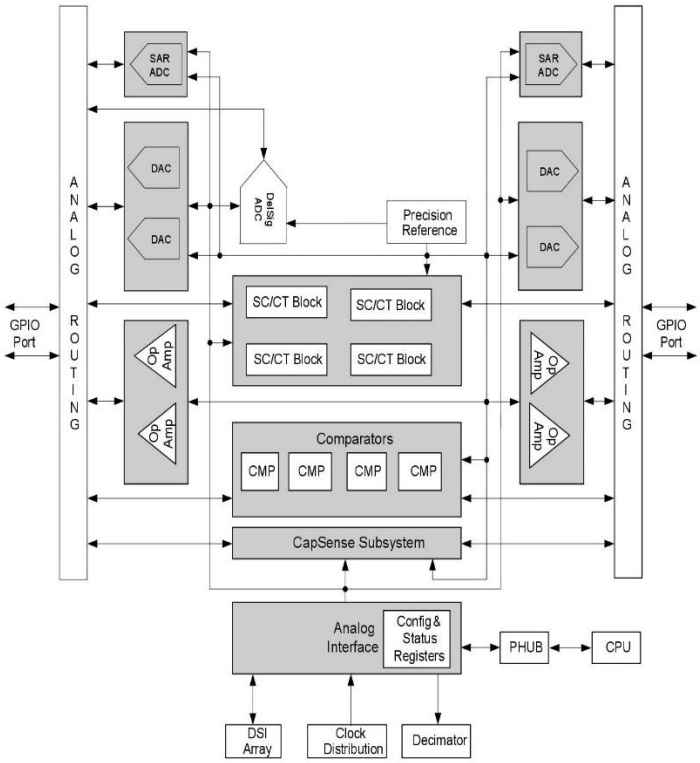
Data movement in or out of the DFB is typically controlled by the system DMA controller but can be moved directly by the MCU.

8. Analog Subsystem

The analog programmable system creates application specific combinations of both standard and advanced analog signal processing blocks. These blocks are then interconnected to each other and also to any pin on the device, providing a high level of design flexibility and IP security. The features of the analog subsystem are outlined here to provide an overview of capabilities and architecture.

- Flexible, configurable analog routing architecture provided by analog globals, analog mux bus, and analog local buses
- High resolution Delta-Sigma ADC
- Two successive approximation (SAR) ADCs
- Four 8-bit DACs that provide either voltage or current output
- Four comparators with optional connection to configurable LUT outputs
- Four configurable switched capacitor/continuos time (SC/CT) blocks for functions that include opamp, unity gain buffer, programmable gain amplifier, transimpedance amplifier, and mixer
- Four opamps for internal use and connection to GPIO that can be used as high current output buffers
- CapSense subsystem to enable capacitive touch sensing
- Precision reference for generating an accurate analog voltage for internal analog blocks

Figure 8-1. Analog Subsystem Block Diagram



The PSoC Creator software program provides a user friendly interface to configure the analog connections between the GPIO and various analog resources and also connections from one analog resource to another. PSoC Creator also provides component libraries that allow you to configure the various analog blocks to perform application specific functions (PGA, transimpedance amplifier, voltage DAC, current DAC, and so on). The tool also generates API interface libraries that allow you to write firmware that allows the communication between the analog peripheral and CPU/Memory.

8.1 Analog Routing

The PSoC 5LP family of devices has a flexible analog routing architecture that provides the capability to connect GPIOs and different analog blocks, and also route signals between different analog blocks. One of the strong points of this flexible routing architecture is that it allows dynamic routing of input and output connections to the different analog blocks.

For information on how to make pin selections for optimal analog routing, refer to the application note, AN58304 - PSoC® 3 and PSoC® 5 - Pin Selection for Analog Designs.

8.1.1 Features

- Flexible, configurable analog routing architecture
- 16 analog globals (AG) and two analog mux buses (AMUXBUS) to connect GPIOs and the analog blocks
- Each GPIO is connected to one analog global and one analog mux bus
- Eight analog local buses (abus) to route signals between the different analog blocks
- Multiplexers and switches for input and output selection of the analog blocks

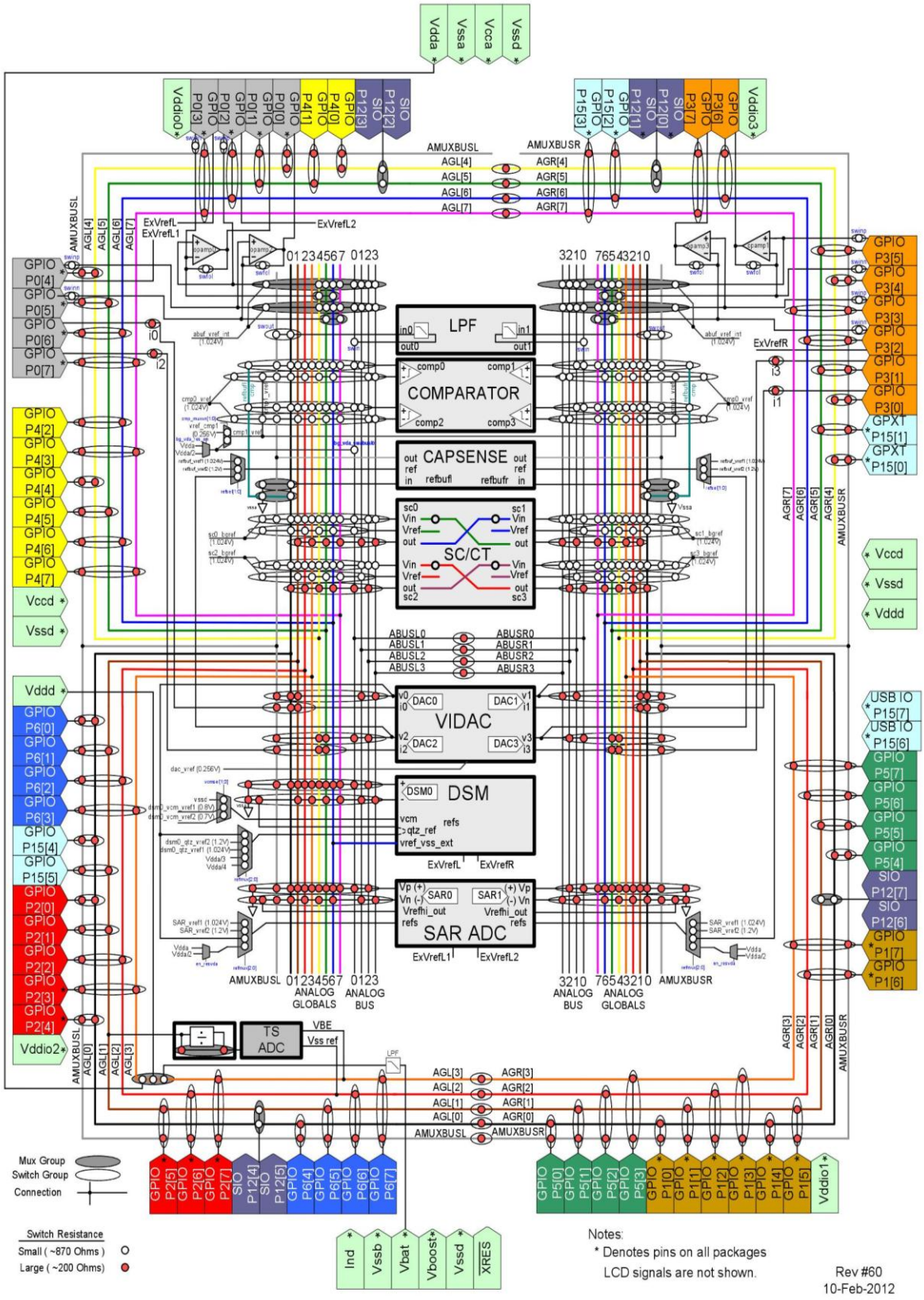
8.1.2 Functional Description

Analog globals (AGs) and analog mux buses (AMUXBUS) provide analog connectivity between GPIOs and the various analog blocks. There are 16 AGs in the PSoC 5LP family. The analog routing architecture is divided into four quadrants as shown in Figure 8-2. Each quadrant has four analog globals (AGL[0..3], AGL[4..7], AGR[0..3], AGR[4..7]). Each GPIO is connected to the corresponding AG through an analog switch. The analog mux bus is a shared routing resource that connects to every GPIO through an analog switch. There are two AMUXBUS routes in PSoC 5LP, one in the left half (AMUXBUSL) and one in the right half (AMUXBUSR), as shown in Figure 8-2.

Analog local buses (abus) are routing resources located within the analog subsystem and are used to route signals between different analog blocks. There are eight abus routes in PSoC 5LP, four in the left half (abusl [0:3]) and four in the right half (abusr [0:3]) as shown in Figure 8-2. Using the abus saves the analog globals and analog mux buses from being used for interconnecting the analog blocks.

Multiplexers and switches exist on the various buses to direct signals into and out of the analog blocks. A multiplexer can have only one connection on at a time, whereas a switch can have multiple connections on simultaneously. In Figure 8-2, multiplexers are indicated by grayed ovals and switches are indicated by transparent ovals.

Figure 8-2. CY8C58LP Analog Interconnect



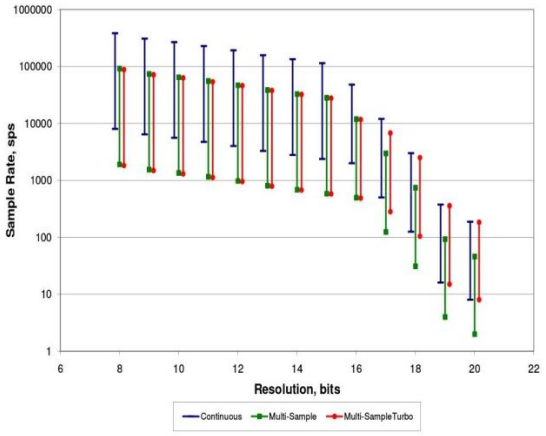
8.2 Delta-sigma ADC

The CY8C58LP device contains one delta-sigma ADC. This ADC offers differential input, high resolution and excellent linearity, making it a good ADC choice for both audio signal processing and measurement applications. The converter's nominal operation is 16 bits at 48 ksps. The ADC can be configured to output 20-bit resolution at data rates of up to 187 sps. At a fixed clock rate, resolution can be traded for faster data rates as shown in Table 8-1 and Figure 8-3.

Table 8-1. Delta-sigma ADC Performance

Bits	Maximum Sample Rate (sps)	SINAD (dB)
20	187	–
16	48 k	84
12	192 k	66
8	384 k	43

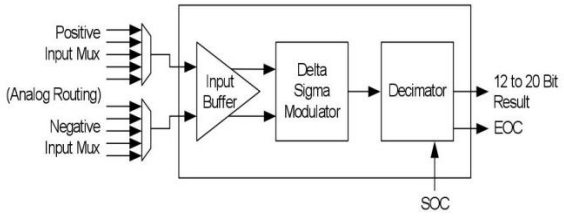
Figure 8-3. Delta-sigma ADC Sample Rates, Range = ±1.024 V



8.2.1 Functional Description

The ADC connects and configures three basic components, input buffer, delta-sigma modulator, and decimator. The basic block diagram is shown in Figure 8-4. The signal from the input muxes is delivered to the delta-sigma modulator either directly or through the input buffer. The delta-sigma modulator performs the actual analog to digital conversion. The modulator over-samples the input and generates a serial data stream output. This high speed data stream is not useful for most applications without some type of post processing, and so is passed to the decimator through the Analog Interface block. The decimator converts the high speed serial data stream into parallel ADC results. The modulator/decimator frequency response is $[(\sin x)/x]^4$.

Figure 8-4. Delta-sigma ADC Block Diagram



Resolution and sample rate are controlled by the Decimator. Data is pipelined in the decimator; the output is a function of the last four samples. When the input multiplexer is switched, the output data is not valid until after the fourth sample after the switch.

8.2.2 Operational Modes

The ADC can be configured by the user to operate in one of four modes: Single Sample, Multi Sample, Continuous, or Multi Sample (Turbo). All four modes are started by either a write to the start bit in a control register or an assertion of the Start of Conversion (SoC) signal. When the conversion is complete, a status bit is set and the output signal End of Conversion (EoC) asserts high and remains high until the value is read by either the DMA controller or the CPU.

8.2.2.1 Single Sample

In Single Sample mode, the ADC performs one sample conversion on a trigger. In this mode, the ADC stays in standby state waiting for the SoC signal to be asserted. When SoC is signaled the ADC performs four successive conversions. The first three conversions prime the decimator. The ADC result is valid and available after the fourth conversion, at which time the EoC signal is generated. To detect the end of conversion, the system may poll a control register for status or configure the external EoC signal to generate an interrupt or invoke a DMA request. When the transfer is done the ADC reenters the standby state where it stays until another SoC event.

8.2.2.2 Continuous

Continuous sample mode is used to take multiple successive samples of a single input signal. Multiplexing multiple inputs should not be done with this mode. There is a latency of three conversion times before the first conversion result is available. This is the time required to prime the decimator. After the first result, successive conversions are available at the selected sample rate.

8.2.2.3 Multi Sample

Multi sample mode is similar to continuous mode except that the ADC is reset between samples. This mode is useful when the input is switched between multiple signals. The decimator is re-primed between each sample so that previous samples do not affect the current conversion. Upon completion of a sample, the next sample is automatically initiated. The results can be transferred using either firmware polling, interrupt, or DMA.

8.2.2.4 Multi Sample (Turbo)

The multi sample (turbo) mode operates identical to the Multi-sample mode for resolutions of 8 to 16 bits. For resolutions of 17 to 20 bits, the performance is about four times faster than the multi sample mode, because the ADC is only reset once at the end of conversion.

More information on output formats is provided in the Technical Reference Manual.

8.2.3 Start of Conversion Input

The SoC signal is used to start an ADC conversion. A digital clock or UDB output can be used to drive this input. It can be used when the sampling period must be longer than the ADC conversion time or when the ADC must be synchronized to other hardware. This signal is optional and does not need to be connected if ADC is running in a continuous mode.

8.2.4 End of Conversion Output

The EoC signal goes high at the end of each ADC conversion. This signal may be used to trigger either an interrupt or DMA request.

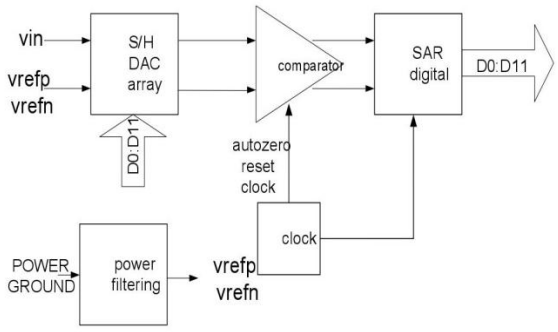
8.3 Successive Approximation ADC

The CY8C58LP family of devices has two Successive Approximation (SAR) ADCs. These ADCs are 12-bit at up to 1 Msps, with single-ended or differential inputs, making them useful for a wide variety of sampling and control applications.

8.3.1 Functional Description

In a SAR ADC an analog input signal is sampled and compared with the output of a DAC. A binary search algorithm is applied to the DAC and used to determine the output bits in succession from MSB to LSB. A block diagram of one SAR ADC is shown in Figure 8-5.

Figure 8-5. SAR ADC Block Diagram



The input is connected to the analog globals and muxes. The frequency of the clock is 18 times the sample rate; the clock rate ranges from 1 to 18 MHz.

8.3.2 Conversion Signals

Writing a start bit or assertion of a start of frame (SOF) signal is used to start a conversion. SOF can be used in applications where the sampling period is longer than the conversion time, or when the ADC needs to be synchronized to other hardware. This signal is optional and does not need to be connected if the SAR ADC is running in a continuous mode. A digital clock or UDB output can be used to drive this input. When the SAR is first powered up or awakened from any of the sleeping modes, there is a power up wait time of 10 µs before it is ready to start the first conversion.

When the conversion is complete, a status bit is set and the output signal end of frame (EOF) asserts and remains asserted until the value is read by either the DMA controller or the CPU. The EOF signal may be used to trigger an interrupt or a DMA request.

8.3.3 Operational Modes

A ONE_SHOT control bit is used to set the SAR ADC conversion mode to either continuous or one conversion per SOF signal. DMA transfer of continuous samples, without CPU intervention, is supported.

8.4 Comparators

The CY8C58LP family of devices contains four comparators. Comparators have these features:

- Input offset factory trimmed to less than 5 mV
- Rail-to-rail common mode input range (V_{SSA} to V_{DDA})
- Speed and power can be traded off by using one of three modes: fast, slow, or ultra low power
- Comparator outputs can be routed to look up tables to perform simple logic functions and then can also be routed to digital blocks
- The positive input of the comparators may be optionally passed through a low pass filter. Two filters are provided
- Comparator inputs can be connections to GPIO, DAC outputs and SC block outputs

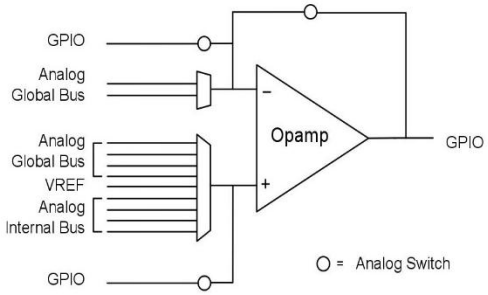
8.4.1 Input and Output Interface

The positive and negative inputs to the comparators come from the analog global buses, the analog mux line, the analog local bus and precision reference through multiplexers. The output from each comparator could be routed to any of the two input LUTs. The output of that LUT is routed to the UDB DSI.

8.5 Opamps

The CY8C58LP family of devices contain four general purpose opamps.

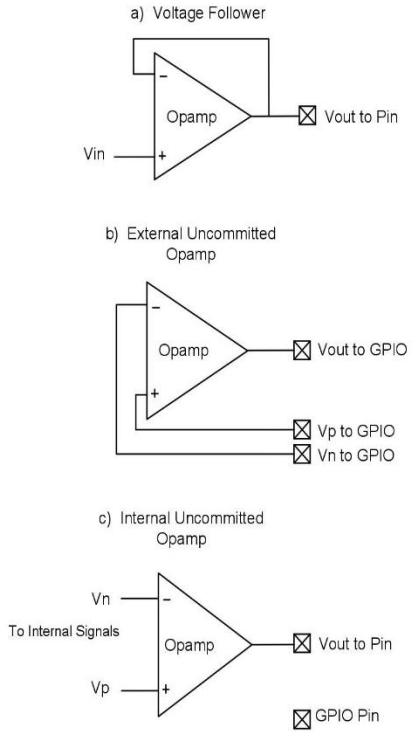
Figure 8-7. Opamp



The opamp is uncommitted and can be configured as a gain stage or voltage follower on external or internal signals.

See Figure 8-8. In any configuration, the input and output signals can all be connected to the internal global signals and monitored with an ADC, or comparator. The configurations are implemented with switches between the signals and GPIO pins.

Figure 8-8. Opamp Configurations



The opamp has three speed modes, slow, medium, and fast. The slow mode consumes the least amount of quiescent power and the fast mode consumes the most power. The inputs are able to swing rail-to-rail. The output swing is capable of rail-to-rail operation at low current output, within 50 mV of the rails. When driving high current loads (about 25 mA) the output voltage may only get within 500 mV of the rails.

8.6 Programmable SC/CT Blocks

The CY8C58LP family of devices contains four switched capacitor/continuous time (SC/CT) blocks. Each switched capacitor/continuous time block is built around a single rail-to-rail high bandwidth opamp.

Switched capacitor is a circuit design technique that uses capacitors plus switches instead of resistors to create analog functions. These circuits work by moving charge between capacitors by opening and closing different switches. Nonoverlapping in phase clock signals control the switches, so that not all switches are ON simultaneously.

The PSoC Creator tool offers a user friendly interface, which allows you to easily program the SC/CT blocks. Switch control and clock phase control configuration is done by PSoC Creator so users only need to determine the application use parameters such as gain, amplifier polarity, V_{REF} connection, and so on.

The same opamps and block interfaces are also connectable to an array of resistors which allows the construction of a variety of continuous time functions.

The opamp and resistor array is programmable to perform various analog functions including

- Naked Operational Amplifier - Continuous Mode
- Unity-Gain Buffer - Continuous Mode
- Programmable Gain Amplifier (PGA) - Continuous Mode
- Transimpedance Amplifier (TIA) - Continuous Mode
- Up/Down Mixer - Continuous Mode
- Sample and Hold Mixer (NRZ S/H) - Switched Cap Mode
- First Order Analog to Digital Modulator - Switched Cap Mode

8.6.1 Naked Opamp

The Naked Opamp presents both inputs and the output for connection to internal or external signals. The opamp has a unity gain bandwidth greater than 6.0 MHz and output drive current up to 650 µA. This is sufficient for buffering internal signals (such as DAC outputs) and driving external loads greater than 7.5 kohms.

8.6.2 Unity Gain

The Unity Gain buffer is a Naked Opamp with the output directly connected to the inverting input for a gain of 1.00. It has a -3 dB bandwidth greater than 6.0 MHz.

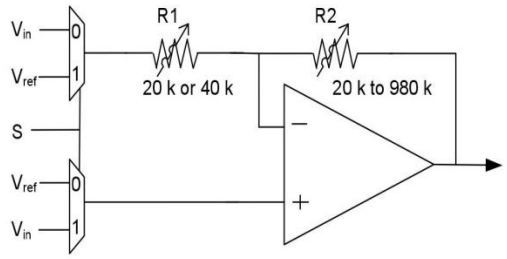
8.6.3 PGA

The PGA amplifies an external or internal signal. The PGA can be configured to operate in inverting mode or noninverting mode. The PGA function may be configured for both positive and negative gains as high as 50 and 49 respectively. The gain is adjusted by changing the values of R1 and R2 as illustrated in Figure 8-9. The schematic in Figure 8-9 shows the configuration and possible resistor settings for the PGA. The gain is switched from inverting and non inverting by changing the shared select value of the both the input muxes. The bandwidth for each gain case is listed in Table 8-3.

Table 8-3. Bandwidth

Gain	Bandwidth
1	6.0 MHz
24	340 kHz
48	220 kHz
50	215 kHz

Figure 8-9. PGA Resistor Settings



The PGA is used in applications where the input signal may not be large enough to achieve the desired resolution in the ADC, or dynamic range of another SC/CT block such as a mixer. The gain is adjustable at runtime, including changing the gain of the PGA prior to each ADC sample.

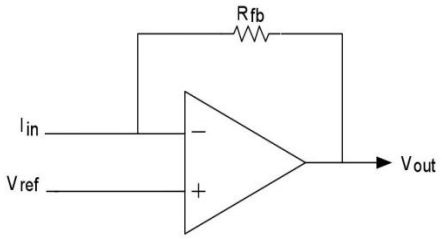
8.6.4 TIA

The Transimpedance Amplifier (TIA) converts an internal or external current to an output voltage. The TIA uses an internal feedback resistor in a continuous time configuration to convert input current to output voltage. For an input current I_{in}, the output voltage is V_{REF} - I_{in} x R_{fb}, where V_{REF} is the value placed on the non inverting input. The feedback resistor Rfb is programmable between 20 KΩ and 1 MΩ through a configuration register. Table 8-4 shows the possible values of Rfb and associated configuration settings.

Table 8-4. Feedback Resistor Settings

Configuration Word	Nominal R_{fb} (KΩ
000b	20
001b	30
010b	40
011b	60
100b	120
101b	250
110b	500
111b	1000

Figure 8-10. Continuous Time TIA Schematic



The TIA configuration is used for applications where an external sensor's output is current as a function of some type of stimulus such as temperature, light, magnetic flux etc. In a common application, the voltage DAC output can be connected to the V_{REF} TIA input to allow calibration of the external sensor bias current by adjusting the voltage DAC output voltage.

8.7 LCD Direct Drive

The PSoC Liquid Crystal Display (LCD) driver system is a highly configurable peripheral designed to allow PSoC to directly drive a broad range of LCD glass. All voltages are generated on chip, eliminating the need for external components. With a high multiplex ratio of up to 1/16, the CY8C58LP family LCD driver system can drive a maximum of 736 segments. The PSoC LCD driver module was also designed with the conservative power budget of portable devices in mind, enabling different LCD drive modes and power down modes to conserve power.

PSoC Creator provides an LCD segment drive component. The component wizard provides easy and flexible configuration of LCD resources. You can specify pins for segments and commons along with other options. The software configures the device to meet the required specifications. This is possible because of the programmability inherent to PSoC devices.

Key features of the PSoC LCD segment system are:

- LCD panel direct driving
- Type A (standard) and Type B (low power) waveform support
- Wide operating voltage range support (2 V to 5 V) for LCD panels
- Static, 1/2, 1/3, 1/4, 1/5 bias voltage levels
- Internal bias voltage generation through internal resistor ladder
- Up to 62 total common and segment outputs
- Up to 1/16 multiplex for a maximum of 16 backplane/common outputs
- Up to 62 front plane/segment outputs for direct drive
- Drives up to 736 total segments (16 backplane x 46 front plane)
- Up to 64 levels of software controlled contrast
- Ability to move display data from memory buffer to LCD driver through DMA (without CPU intervention)
- Adjustable LCD refresh rate from 10 Hz to 150 Hz
- Ability to invert LCD display for negative image
- Three LCD driver drive modes, allowing power optimization

Figure 8-11. LCD System

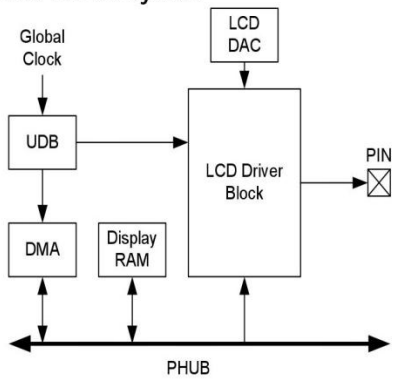
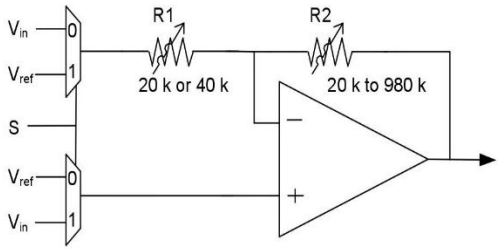


Table 8-3. Bandwidth

Gain	Bandwidth
1	6.0 MHz
24	340 kHz
48	220 kHz
50	215 kHz

Figure 8-9. PGA Resistor Settings



The PGA is used in applications where the input signal may not be large enough to achieve the desired resolution in the ADC, or dynamic range of another SC/CT block such as a mixer. The gain is adjustable at runtime, including changing the gain of the PGA prior to each ADC sample.

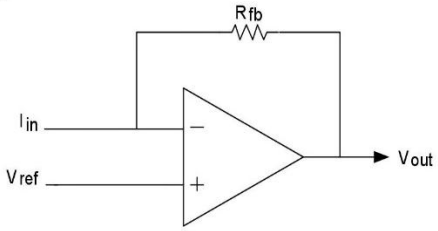
8.6.4 TIA

The Transimpedance Amplifier (TIA) converts an internal or external current to an output voltage. The TIA uses an internal feedback resistor in a continuous time configuration to convert input current to output voltage. For an input current I_{in}, the output voltage is $V_{REF} - I_{in} \times R_{fb}$, where V_{REF} is the value placed on the non inverting input. The feedback resistor Rfb is programmable between 20 KΩ and 1 MΩ through a configuration register. Table 8-4 shows the possible values of Rfb and associated configuration settings.

Table 8-4. Feedback Resistor Settings

Configuration Word	Nominal R_{fb} (KΩ
000b	20
001b	30
010b	40
011b	60
100b	120
101b	250
110b	500
111b	1000

Figure 8-10. Continuous Time TIA Schematic



The TIA configuration is used for applications where an external sensor's output is current as a function of some type of stimulus such as temperature, light, magnetic flux etc. In a common application, the voltage DAC output can be connected to the V_{REF} TIA input to allow calibration of the external sensor bias current by adjusting the voltage DAC output voltage.

8.7 LCD Direct Drive

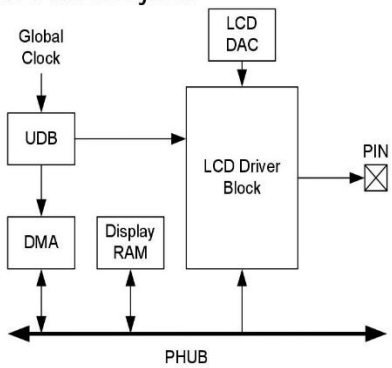
The PSoC Liquid Crystal Display (LCD) driver system is a highly configurable peripheral designed to allow PSoC to directly drive a broad range of LCD glass. All voltages are generated on chip, eliminating the need for external components. With a high multiplex ratio of up to 1/16, the CY8C58LP family LCD driver system can drive a maximum of 736 segments. The PSoC LCD driver module was also designed with the conservative power budget of portable devices in mind, enabling different LCD drive modes and power down modes to conserve power.

PSoC Creator provides an LCD segment drive component. The component wizard provides easy and flexible configuration of LCD resources. You can specify pins for segments and commons along with other options. The software configures the device to meet the required specifications. This is possible because of the programmability inherent to PSoC devices.

Key features of the PSoC LCD segment system are:

- LCD panel direct driving
- Type A (standard) and Type B (low power) waveform support
- Wide operating voltage range support (2 V to 5 V) for LCD panels
- Static, 1/2, 1/3, 1/4, 1/5 bias voltage levels
- Internal bias voltage generation through internal resistor ladder
- Up to 62 total common and segment outputs
- Up to 1/16 multiplex for a maximum of 16 backplane/common outputs
- Up to 62 front plane/segment outputs for direct drive
- Drives up to 736 total segments (16 backplane x 46 front plane)
- Up to 64 levels of software controlled contrast
- Ability to move display data from memory buffer to LCD driver through DMA (without CPU intervention)
- Adjustable LCD refresh rate from 10 Hz to 150 Hz
- Ability to invert LCD display for negative image
- Three LCD driver drive modes, allowing power optimization

Figure 8-11. LCD System



11. Electrical Specifications

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted. The unique flexibility of the PSoC UDBs and analog blocks enable many functions to be implemented in PSoC Creator components, see the component datasheets for full AC/DC specifications of individual functions. See the "Example Peripherals" section on page 37 for further explanation of PSoC Creator components.

11.1 Absolute Maximum Ratings

Table 11-1. Absolute Maximum Ratings DC Specifications[12]

Parameter	Description	Conditions	Min	Typ	Max	Units
V_{DDA}	Analog supply voltage relative to V_{SSA}		–0.5	–	6	V
V_{DDD}	Digital supply voltage relative to V_{SSD}		–0.5	–	6	V
V_{DDIO}	I/O supply voltage relative to V_{SSD}		–0.5	–	6	V
V_{CCA}	Direct analog core voltage input		–0.5	–	1.95	V
V_{CCD}	Direct digital core voltage input		–0.5	–	1.95	V
V_{SSA}	Analog ground voltage		V_{SSD} – 0.5	–	V_{SSD} + 0.5	V
V_{GPIO}[13]	DC input voltage on GPIO	Includes signals sourced by V_{DDA} and routed internal to the pin.	V_{SSD} – 0.5	–	V_{DDIO} + 0.5	V
V_{SIO}	DC input voltage on SIO	Output disabled	V_{SSD} – 0.5	–	7	V
		Output enabled	V_{SSD} – 0.5	–	6	V
V_{IND}	Voltage at boost converter input		0.5	–	5.5	V
V_{BAT}	Boost converter supply		V_{SSD} – 0.5	–	5.5	V
I_{VDDIO}	Current per V_{DDIO} supply pin		–	–	100	mA
I_{GPIO}	GPIO current		–30	–	41	mA
I_{SIO}	SIO current		–49	–	28	mA
I_{USBIO}	USBIO current		–56	–	59	mA
V_{EXTREF}	ADC external reference inputs	Pins P0[3], P3[2]	–	–	2	V
LU	Latch up current[14]		–140	–	140	mA
ESD_{HBM}	Electrostatic discharge voltage	Human body model	2000	–	–	V
ESD_{CDM}	ESD voltage	Charge device model	500	–	–	V

11.2 Device Level Specifications

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted. Unless otherwise specified, all charts and graphs show typical values.

11.2.1 Device Level Specifications

Table 11-2. DC Specifications

Parameter	Description	Conditions		Min	Typ	Max	Units
V_{DDA}	Analog supply voltage and input to analog core regulator	Analog core regulator enabled		1.8	–	5.5	V
V_{DDA}	Analog supply voltage, analog regulator bypassed	Analog core regulator disabled		1.71	1.8	1.89	V
V_{DDD}	Digital supply voltage relative to V_{SSD}	Digital core regulator enabled		1.8	–	V_{DDA}[15]	V
V_{DDD}	Digital supply voltage, digital regulator bypassed	Digital core regulator disabled		1.71	1.8	1.89	V
V_{DDIO}[16]	I/O supply voltage relative to V_{SSIO}			1.71	–	V_{DDA}[15]	V
V_{CCA}	Direct analog core voltage input (Analog regulator bypass)	Analog core regulator disabled		1.71	1.8	1.89	V
V_{CCD}	Direct digital core voltage input (Digital regulator bypass)	Digital core regulator disabled		1.71	1.8	1.89	V
Active Mode							
I_{DD}[17]	Sum of digital and analog IDDD + IDDA. IDDIOX for I/Os not included. IMO enabled, bus clock and CPU clock enabled. CPU executing complex program from flash.	V_{DDX} = 2.7 V to 5.5 V; F_{CPU} = 3 MHz[18]	T = –40 °C	–	1.9	3.8	mA
			T = 25 °C	–	1.9	3.8	
			T = 85 °C	–	2	3.8	
		V_{DDX} = 2.7 V to 5.5 V; F_{CPU} = 6 MHz	T = –40 °C	–	3.1	5	
			T = 25 °C	–	3.1	5	
			T = 85 °C	–	3.2	5	
		V_{DDX} = 2.7 V to 5.5 V; F_{CPU} = 12 MHz[18]	T = –40 °C	–	5.4	7	
			T = 25 °C	–	5.4	7	
			T = 85 °C	–	5.6	7	
		V_{DDX} = 2.7 V to 5.5 V; F_{CPU} = 24 MHz[18]	T = –40 °C	–	8.9	10.5	
			T = 25 °C	–	8.9	10.5	
			T = 85 °C	–	9.1	10.5	
		V_{DDX} = 2.7 V to 5.5 V; F_{CPU} = 48 MHz[18]	T = –40 °C	–	15.5	17	
			T = 25 °C	–	15.4	17	
			T = 85 °C	–	15.7	17	
		V_{DDX} = 2.7 V to 5.5 V; F_{CPU} = 62 MHz	T = –40 °C	–	18	19.5	
			T = 25 °C	–	18	19.5	
			T = 85 °C	–	18.5	19.5	
		V_{DDX} = 2.7 V to 5.5 V; F_{CPU} = 74 MHz	T = –40 °C	–	26.5	30	
			T = 25 °C	–	26.5	30	
			T = 85 °C	–	27	30	
		V_{DDX} = 2.7 V to 5.5 V; F_{CPU} = 80 MHz, IMO = 3 MHz with PLL	T = –40 °C	–	22	25.5	
			T = 25 °C	–	22	25.5	
			T = 85 °C	–	22.5	25.5	

Notes

15. The power supplies can be brought up in any sequence however once stable Vdda must be greater than or equal to all other supplies.
16. The V_{DDIO} supply voltage must be greater than the maximum voltage on the associated GPIO pins. Maximum voltage on GPIO pin ≤ V_{DDIO} ≤ V_{DDA}.
17. The current consumption of additional peripherals that are implemented only in programmed logic blocks can be found in their respective datasheets, available in PSoC Creator, the integrated design environment. To estimate total current, find CPU current at frequency of interest and add peripheral currents for your particular system from the device datasheet and component datasheets.
18. Based on device characterization (Not production tested).

11.4 Inputs and Outputs

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted. Unless otherwise specified, all charts and graphs show typical values.

When the power supplies ramp up, there are low-impedance connections between each GPIO pin and its V_{DDIO} supply. This causes the pin voltages to track V_{DDIO} until both V_{DDIO} and V_{DDA} reach the IPOR voltage, which can be as high as 1.45 V. At that point, the low-impedance connections no longer exist and the pins change to their normal NVL settings.

Also, if V_{DDA} is less than V_{DDIO}, a low-impedance path may exist between a GPIO and V_{DDA}, causing the GPIO to track V_{DDA} until V_{DDA} becomes greater than or equal to V_{DDIO}.

11.4.1 GPIO

Table 11-9. GPIO DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
V_{IH}	Input voltage high threshold	CMOS Input, PRT[x]CTL = 0	0.7 × V_{DDIO}	–	–	V
V_{IL}	Input voltage low threshold	CMOS Input, PRT[x]CTL = 0	–	–	0.3 × V_{DDIO}	V
V_{IH}	Input voltage high threshold	LVTTL Input, PRT[x]CTL = 1, V_{DDIO} < 2.7 V	0.7 x V_{DDIO}	–	–	V
V_{IH}	Input voltage high threshold	LVTTL Input, PRT[x]CTL = 1, V_{DDIO} ≥ 2.7 V	2.0	–	–	V
V_{IL}	Input voltage low threshold	LVTTL Input, PRT[x]CTL = 1, V_{DDIO} < 2.7 V	–	–	0.3 x V_{DDIO}	V
V_{IL}	Input voltage low threshold	LVTTL Input, PRT[x]CTL = 1, V_{DDIO} ≥ 2.7 V	–	–	0.8	V
V_{OH}	Output voltage high	I_{OH} = 4 mA at 3.3 V_{DDIO}	V_{DDIO} – 0.6	–	–	V
		I_{OH} = 1 mA at 1.8 V_{DDIO}	V_{DDIO} – 0.5	–	–	V
V_{OL}	Output voltage low	I_{OL} = 8 mA at 3.3 V_{DDIO}	–	–	0.6	V
		I_{OL} = 3 mA at 3.3 V_{DDIO}	–	–	0.4	V
		I_{OL} = 4 mA at 1.8 V_{DDIO}	–	–	0.6	V
Rpullup	Pull-up resistor		3.5	5.6	8.5	kΩ
Rpulldown	Pull-down resistor		3.5	5.6	8.5	kΩ
I_{IL}	Input leakage current (absolute value)[29]	25 °C, V_{DDIO} = 3.0 V	–	–	2	nA
C_{IN}	Input capacitance[29]	P0.0, P0.1, P0.2, P3.6, P3.7	–	17	20	pF
		P0.3, P0.4, P3.0, P3.1, P3.2	–	10	15	pF
		P0.6, P0.7, P15.0, P15.6, P15.7[30]	–	7	12	pF
		All other GPIOs	–	5	9	pF
V_H	Input voltage hysteresis (Schmitt-Trigger)[29]		–	40	–	mV
Idiode	Current through protection diode to V_{DDIO} and V_{SSIO}		–	–	100	µA
Rglobal	Resistance pin to analog global bus	25 °C, V_{DDIO} = 3.0 V	–	320	–	Ω
Rmux	Resistance pin to analog mux bus	25 °C, V_{DDIO} = 3.0 V	–	220	–	Ω

11.5 Analog Peripherals

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted.

11.5.1 Opamp

Table 11-19. Opamp DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
V_I	Input voltage range		V_{SSA}	–	V_{DDA}	V
Vos	Input offset voltage		–	–	2.5	mV
		Operating temperature –40 °C to 70 °C	–	–	2	mV
TCVos	Input offset voltage drift with temperature	Power mode = high	–	–	±30	µV / °C
Ge1	Gain error, unity gain buffer mode	Rload = 1 kΩ	–	–	±0.1	%
Cin	Input capacitance	Routing from pin	–	–	18	pF
Vo	Output voltage range	1 mA, source or sink, power mode = high	V_{SSA} + 0.05	–	V_{DDA} – 0.05	V
Iout	Output current capability, source or sink	V_{SSA} + 500 mV ≤ V_{OUT} ≤ V_{DDA} –500 mV, V_{DDA} > 2.7 V	25	–	–	mA
		V_{SSA} + 500 mV ≤ V_{OUT} ≤ V_{DDA} –500 mV, 1.7 V = V_{DDA} ≤ 2.7 V	16	–	–	mA
Idd	Quiescent current[38]	Power mode = min	–	250	400	uA
		Power mode = low	–	250	400	uA
		Power mode = med	–	330	950	uA
		Power mode = high	–	1000	2500	uA
CMRR	Common mode rejection ratio[38]		80	–	–	dB
PSRR	Power supply rejection ratio[38]	V_{DDA} ≥ 2.7 V	85	–	–	dB
		V_{DDA} < 2.7 V	70	–	–	dB
I_{IB}	Input bias current[38]	25 °C	–	10	–	pA

Figure 11-20. Opamp Vos Histogram, 7020 samples/1755 parts, 30 °C, VDDA = 3.3 V

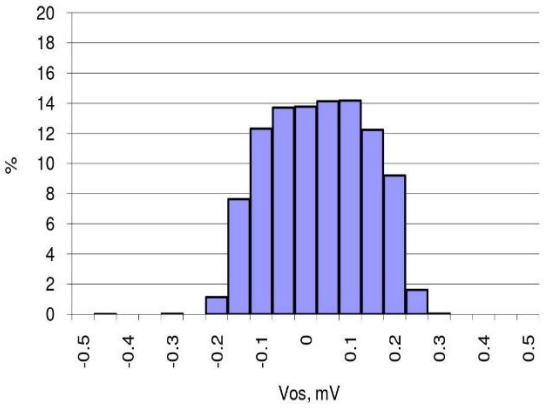
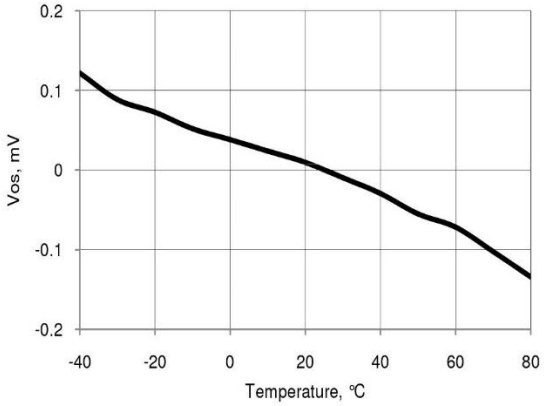


Figure 11-21. Opamp Vos vs Temperature, V_{DDA} = 5 V



11.5.4 SAR ADC

Table 11-29. SAR ADC DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
	Resolution		–	–	12	bits
	Number of channels – single-ended		–	–	No of GPIO	
	Number of channels – differential	Differential pair is formed using a pair of neighboring GPIO.	–	–	No of GPIO/2	
	Monotonicity[46]		Yes	–	–	
Ge	Gain error[47]	External reference	–	–	±0.1	%
V_{OS}	Input offset voltage		–	–	±2	mV
I_{DD}	Current consumption[46]		–	–	1	mA
	Input voltage range – single-ended[46]		V_{SSA}	–	V_{DDA}	V
	Input voltage range – differential[46]		V_{SSA}	–	V_{DDA}	V
PSRR	Power supply rejection ratio[46]		70	–	–	dB
CMRR	Common mode rejection ratio		70	–	–	dB
INL	Integral non linearity[46]	V_{DDA} 1.71 to 5.5 V, 1 Msps, V_{REF} 1 to 5.5 V, bypassed at ExtRef pin	–	–	+2/–1.5	LSB
		V_{DDA} 2.0 to 3.6 V, 1 Msps, V_{REF} 2 to V_{DDA}, bypassed at ExtRef pin	–	–	±1.2	LSB
		V_{DDA} 1.71 to 5.5 V, 500 ksps, V_{REF} 1 to 5.5 V, bypassed at ExtRef pin	–	–	±1.3	LSB
DNL	Differential non linearity[46]	V_{DDA} 1.71 to 5.5 V, 1 Msps, V_{REF} 1 to 5.5 V, bypassed at ExtRef pin	–	–	+2/–1	LSB
		V_{DDA} 2.0 to 3.6 V, 1 Msps, V_{REF} 2 to V_{DDA}, bypassed at ExtRef pin No missing codes	–	–	1.7/–0.99	LSB
		V_{DDA} 1.71 to 5.5 V, 500 ksps, V_{REF} 1 to 5.5 V, bypassed at ExtRef pin No missing codes	–	–	+2/–0.99	LSB
R_{IN}	Input resistance[46]		–	180	–	kΩ

Figure 11-36. SAR ADC DNL vs Output Code, Bypassed Internal Reference Mode

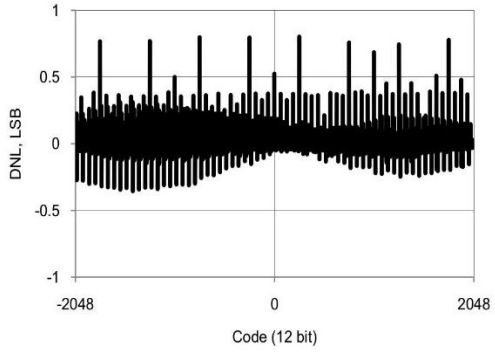
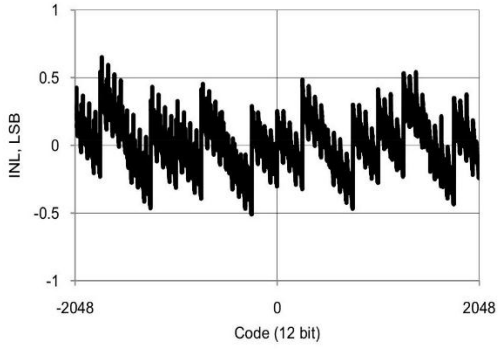


Figure 11-37. SAR ADC INL vs Output Code, Bypassed Internal Reference Mode



Notes
46. Based on device characterization (Not production tested).
47. For total analog system Idd < 5 mA, depending on package used. With higher total analog system currents it is recommended that the SAR ADC be used in differential mode.

Table 11-37. VDAC DC Specifications (continued)

Parameter	Description	Conditions	Min	Typ	Max	Units
V_{OUT}	Output voltage range, code = 255	1 V scale	–	1.02	–	V
		4 V scale, Vdda = 5 V	–	4.08	–	V
	Monotonicity		–	–	Yes	–
V_{OS}	Zero scale error		–	0	±0.9	LSB
Eg	Gain error	1 V scale	–	–	±2.5	%
		4 V scale	–	–	±2.5	%
TC_Eg	Temperature coefficient, gain error	1 V scale	–	–	0.03	%FSR / °C
		4 V scale	–	–	0.03	%FSR / °C
I_{DD}	Operating current[58]	Slow mode	–	–	100	µA
		Fast mode	–	–	500	µA

Figure 11-55. VDAC INL vs Input Code, 1 V Mode

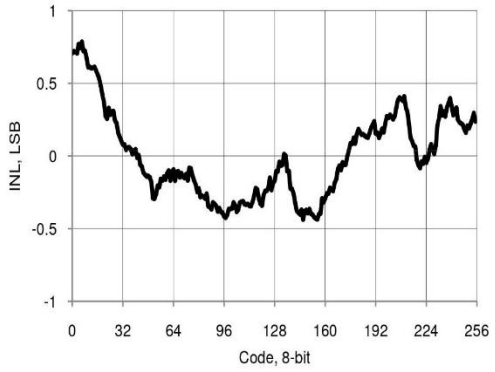


Figure 11-56. VDAC DNL vs Input Code, 1 V Mode

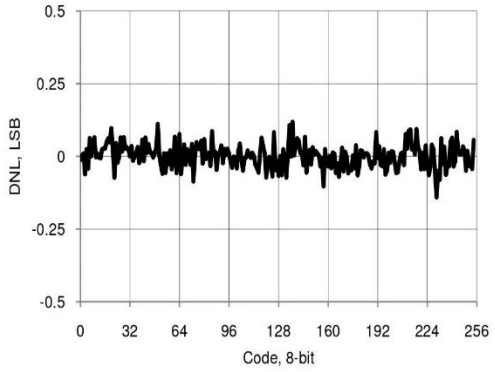


Figure 11-57. VDAC INL vs Temperature, 1 V Mode

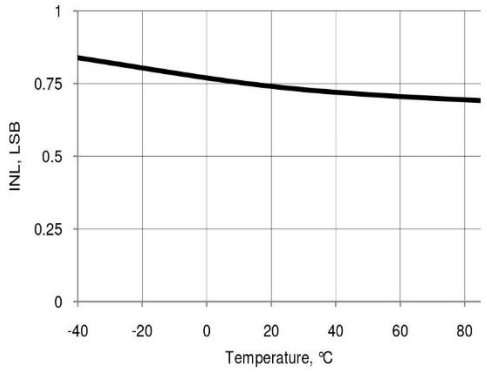
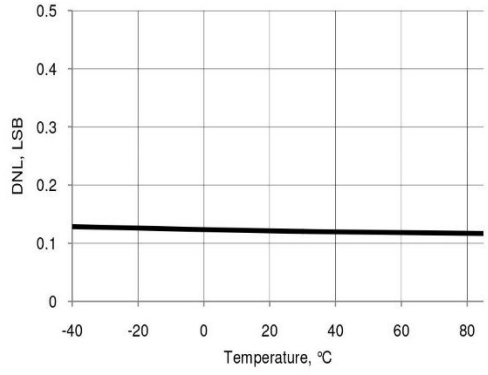


Figure 11-58. VDAC DNL vs Temperature, 1 V Mode



11.5.11 Programmable Gain Amplifier

The PGA is created using a SC/CT analog block; see the PGA component datasheet in PSoC Creator for full electrical specifications and APIs.

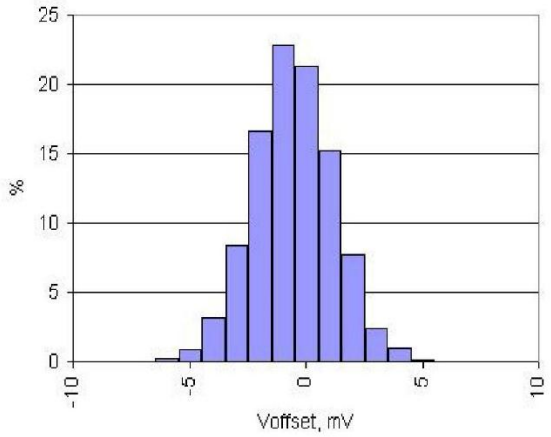
Unless otherwise specified, operating conditions are:

- Operating temperature = 25 °C for typical values
- Unless otherwise specified, all charts and graphs show typical values

Table 11-43. PGA DC Specifications

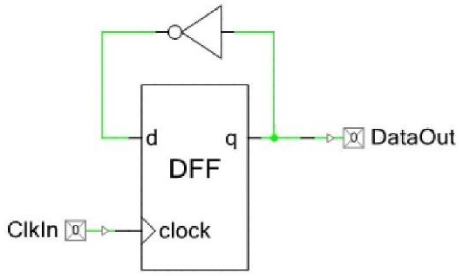
Parameter	Description	Conditions	Min	Typ	Max	Units
Vin	Input voltage range	Power mode = minimum	Vssa	–	Vdda	V
Vos	Input offset voltage	Power mode = high, gain = 1	–	–	10	mV
TCVos	Input offset voltage drift with temperature	Power mode = high, gain = 1	–	–	±30	µV/°C
Ge1	Gain error, gain = 1		–	–	±0.15	%
Ge16	Gain error, gain = 16		–	–	±2.5	%
Ge50	Gain error, gain = 50		–	–	±5	%
Vonl	DC output nonlinearity	Gain = 1	–	–	±0.01	% of FSR
Cin	Input capacitance		–	–	7	pF
Voh	Output voltage swing	Power mode = high, gain = 1, Rload = 100 kΩ to V_{DDA} / 2	V_{DDA} – 0.15	–	–	V
Vol	Output voltage swing	Power mode = high, gain = 1, Rload = 100 kΩ to V_{DDA} / 2	–	–	V_{SSA} + 0.15	V
Vsrc	Output voltage under load	Iload = 250 µA, Vdda ≥ 2.7V, power mode = high	–	–	300	mV
Idd	Operating current[62]	Power mode = high	–	1.5	1.65	mA
PSRR	Power supply rejection ratio		48	–	–	dB

Figure 11-67. PGA Voffset Histogram, 4096 samples/ 1024 parts



Note
62. Based on device characterization (Not production tested).

Figure 11-70. Clock to Output Performance



11.7 Memory

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted.

11.7.1 Flash

Table 11-62. Flash DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
	Erase and program voltage	V_{DDD} pin	1.71	–	5.5	V

Table 11-63. Flash AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
T_{WRITE}	Row write time (erase + program)		–	15	20	ms
T_{ERASE}	Row erase time		–	10	13	ms
	Row program time		–	5	7	ms
T_{BULK}	Bulk erase time (256 KB)		–	–	140	ms
	Sector erase time (16 KB)		–	–	15	ms
T_{PROG}	Total device programming time	No overhead[72]	–	5	7.5	seconds
	Flash data retention time, retention period measured from last erase cycle	Average ambient temp. T_A ≤ 55 °C, 100 K erase/ program cycles	20	–	–	years
		Average ambient temp. T_A ≤ 85 °C, 10 K erase/ program cycles	10	–	–	

11.7.2 EEPROM

Table 11-64. EEPROM DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
	Erase and program voltage		1.71	–	5.5	V

Table 11-65. EEPROM AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
T_{WRITE}	Single row erase/write cycle time		–	10	20	ms
	EEPROM data retention time, retention period measured from last erase cycle	Average ambient temp, $T_A \le 25$ °C, 1M erase/program cycles	20	–	–	years
		Average ambient temp, $T_A \le 55$ °C, 100 K erase/program cycles	20	–	–	
		Average ambient temp. $T_A \le 85$ °C, 10 K erase/program cycles	10	–	–	

11.7.3 Nonvolatile Latches (NVL)

Table 11-66. NVL DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
	Erase and program voltage	V_{DDD} pin	1.71	–	5.5	V

Table 11-67. NVL AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
	NVL endurance	Programmed at 25 °C	1K	–	–	program/ erase cycles
		Programmed at 0 °C to 70 °C	100	–	–	program/ erase cycles
	NVL data retention time	Average ambient temp. $T_A \le 55$ °C	20	–	–	years
		Average ambient temp. $T_A \le 85$ °C	10	–	–	years

11.7.4 SRAM

Table 11-68. SRAM DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
V_{SRAM}	SRAM retention voltage[73]		1.2	–	–	V

Table 11-69. SRAM AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
F_{SRAM}	SRAM operating frequency		DC	–	80.01	MHz

11.9 Clocking

Specifications are valid for –40 °C ≤ T_A ≤ 85 °C and T_J ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted. Unless otherwise specified, all charts and graphs show typical values

11.9.1 Internal Main Oscillator

Table 11-80. IMO DC Specifications[89]

Parameter	Description	Conditions	Min	Typ	Max	Units
Icc_imo	Supply current					
	74.7 MHz		–	–	730	µA
	62.6 MHz		–	–	600	µA
	48 MHz		–	–	500	µA
	24 MHz – USB mode	With oscillator locking to USB bus	–	–	500	µA
	24 MHz – non-USB mode		–	–	300	µA
	12 MHz		–	–	200	µA
	6 MHz		–	–	180	µA
	3 MHz		–	–	150	µA

Figure 11-75. IMO Current vs. Frequency

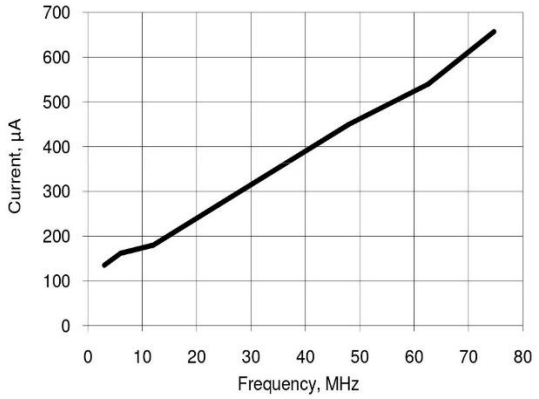


Table 11-81. IMO AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
F_{IMO}[88]	IMO frequency stability (with factory trim)					
	74.7 MHz		–7	–	7	%
	62.6 MHz		–7	–	7	%
	48 MHz		–5	–	5	%
	24 MHz – non-USB mode		–4	–	4	%
	24 MHz – USB mode	With oscillator locking to USB bus	–0.25	–	0.25	%
	12 MHz		–3	–	3	%
	6 MHz		–2	–	2	%
	3 MHz	0 °C to 70 °C	–1	–	1	%
		–40 °C to 85 °C	–1.5	–	1.5	%
	3-MHz frequency stability after typical PCB assembly post-reflow	Typical (non-optimized) board layout and 250 °C solder reflow. Device may be calibrated after assembly to improve performance.	–	±2%	–	%

Notes
88. F_{IMO} is measured after packaging, and thus accounts for substrate and die attach stresses.
89. Based on device characterization (Not production tested).

11.9.3 MHz External Crystal Oscillator

For more information on crystal or ceramic resonator selection for the MHzECO, refer to application note AN54439: PSoC 3 and PSoC 5 External Oscillators.

Table 11-84. MHzECO AC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
F	Crystal frequency range		4	–	25	MHz

11.9.4 kHz External Crystal Oscillator

Table 11-85. kHzECO DC Specifications[94]

Parameter	Description	Conditions	Min	Typ	Max	Units
I_{CC}	Operating current	Low power mode; CL = 6 pF	–	0.25	1.0	µA
DL	Drive level		–	–	1	µW

Table 11-86. kHzECO AC Specifications[94]

Parameter	Description	Conditions	Min	Typ	Max	Units
F	Frequency		–	32.768	–	kHz
T_{ON}	Startup time	High power mode	–	1	–	s

11.9.5 External Clock Reference

Table 11-87. External Clock Reference AC Specifications[94]

Parameter	Description	Conditions	Min	Typ	Max	Units
	External frequency range		0	–	33	MHz
	Input duty cycle range	Measured at $V_{DDIO}/2$	30	50	70	%
	Input edge rate	V_{IL} to V_{IH}	0.5	–	–	V/ns

11.9.6 Phase-Locked Loop

Table 11-88. PLL DC Specifications

Parameter	Description	Conditions	Min	Typ	Max	Units
I_{DD}	PLL operating current	In = 3 MHz, Out = 80 MHz	–	650	–	µA
		In = 3 MHz, Out = 67 MHz	–	400	–	µA
		In = 3 MHz, Out = 24 MHz	–	200	–	µA

Table 11-89. PLL AC Specifications

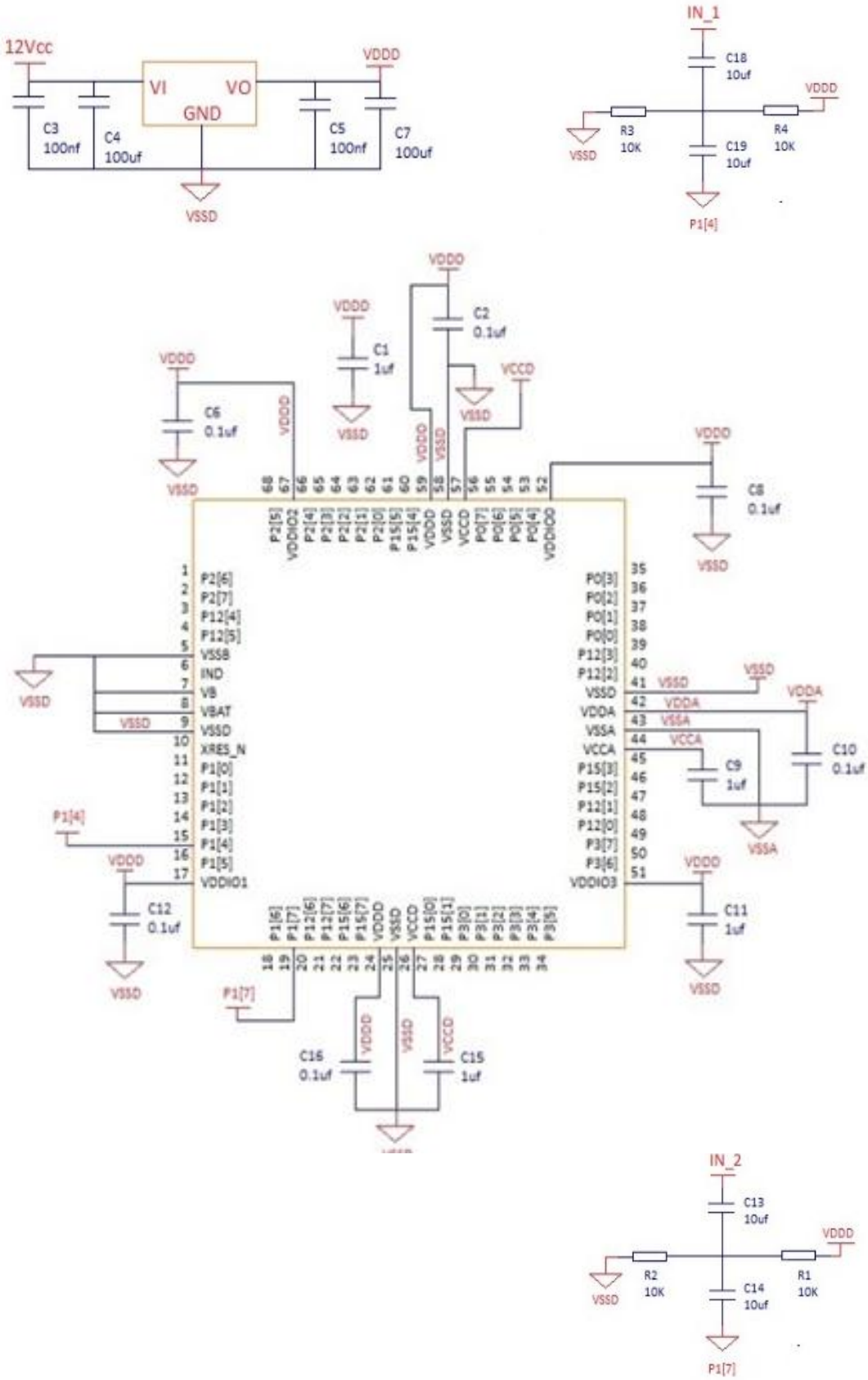
Parameter	Description	Conditions	Min	Typ	Max	Units
Fpllin	PLL input frequency[95]		1	–	48	MHz
	PLL intermediate frequency[96]	Output of prescaler	1	–	3	MHz
Fpllout	PLL output frequency[95]		24	–	80	MHz
	Lock time at startup		–	–	250	µs
Jperiod-rms	Jitter (rms)[94]		–	–	250	ps

Notes
94. Based on device characterization (Not production tested).
95. This specification is guaranteed by testing the PLL across the specified range using the IMO as the source for the PLL.
96. PLL input divider, Q, must be set so that the input frequency is divided down to the intermediate frequency range. Value for Q ranges from 1 to 16.

ANEXO 2

ESQUEMA DEL FILTRO

ADAPTATIVO


12Vcc
VI
VO
GND
VDDD
C3
100nf
C4
100uf
C5
100nf
C7
100uf
VSSD
IN_1
C18
10uf
R3
10K
R4
10K
C19
10uf
P1[4]
C1
1uf
C2
0.1uf
C6
0.1uf
VCCD
C8
0.1uf
P2[6]
P2[7]
P12[4]
P12[5]
VSSB
IND
VB
VBAT
VSSD
XRES_N
P1[0]
P1[1]
P1[2]
P1[3]
P1[4]
P1[5]
VDDIO1
P0[3]
P0[2]
P0[1]
P0[0]
P12[3]
P12[2]
VSSD
VDDA
VSSA
VCCA
P15[3]
P15[2]
P12[1]
P12[0]
P3[7]
P3[6]
VDDIO3
C9
1uf
C10
0.1uf
C11
1uf
C12
0.1uf
P1[7]
C16
0.1uf
C15
1uf
IN_2
C13
10uf
R2
10K
R1
10K
C14
10uf

ANEXO 3

PROGRAMACION DEL FILTRO ADAPTATIVO CON EL ALGORITMO RLS.

MAIN

```
#include <project.h>
#include <VDAC8_1.h>
#include <filtro.h>

int enc = 0;

volatile int IsrOverrun = 0;
static boolean_T OverrunFlag = 0;

void rt_OneStep(void)
{
  if (OverrunFlag++) {
    IsrOverrun = 1;
    OverrunFlag--;
    return;
  }

  filtro_step();
  OverrunFlag--;
}

#define UNUSED(x)                       x = x

CY_ISR(filtroVDAC)
{

  rt_OneStep();

  filtro_U.In3 = ADC_SAR_2_GetResult16();
  filtro_U.In4 = ADC_SAR_1_GetResult16();
    VDAC8_1_SetValue(filtro_Y.Out1+128);
}

int main()
{
  rtmSetErrorStatus(filtro_M, 0);
  filtro_initialize();

    ADC_SAR_1_Start();
    ADC_SAR_1_StartConvert();
    ADC_SAR_2_Start();
    ADC_SAR_2_StartConvert();
    PGA_1_Start();
    VDAC8_1_Start();
    Timer_1_Start();//////////////
    isr_1_StartEx(filtroVDAC);
```

```
    CyGlobalIntEnable;

    for(;;)
    {
    }
   }
```

FILTRO

```
#include <filtro.h>
#include <filtro_private.h>

B_filtro_T filtro_B;

DW_filtro_T filtro_DW;

ExtU_filtro_T filtro_U;

ExtY_filtro_T filtro_Y;

RT_MODEL_filtro_T filtro_M_;
RT_MODEL_filtro_T *const filtro_M = &filtro_M_;

void filtro_step(void)
{
  int32_T s7_iter;
  real_T rtb_Buffer_idx_0;
  real_T rtb_Product2_idx_0;
  real_T rtb_Gain_idx_0;
  real_T rtb_Gain_idx_1;
  real_T rtb_Gain_idx_2;
  real_T rtb_Gain_idx_3;
  real_T rtb_MathFunction_idx_0;
  int32_T exitg1;

  s7_iter = 1;
  do {
    exitg1 = 0;
    if (filtro_ConstB.Width < 0.0) {
      filtro_B.Sum3 = ceil(filtro_ConstB.Width);
    } else {
      filtro_B.Sum3 = floor(filtro_ConstB.Width);
    }

    if (rtIsNaN(filtro_B.Sum3) || rtIsInf(filtro_B.Sum3)) {
      filtro_B.Sum3 = 0.0;
    } else {
      filtro_B.Sum3 = fmod(filtro_B.Sum3, 4.294967296E+9);
    }

    if (s7_iter <= (filtro_B.Sum3 < 0.0 ? -(int32_T)(uint32_T)-
filtro_B.Sum3 :
```

```
                    (int32_T)(uint32_T)filtro_B.Sum3)) {
      rtb_Buffer_idx_0 = filtro_DW.Buffer_CircBuff;
      filtro_DW.Buffer_CircBuff = filtro_U.In3;
      if (filtro_P.Reset_Value) {
        filtro_DW.FilterTaps_DSTATE[0] =
filtro_P.FilterTaps_InitialCondition;
        filtro_DW.FilterTaps_DSTATE[1] =
filtro_P.FilterTaps_InitialCondition;
      }

      filtro_B.Sum3 = rtb_Buffer_idx_0 *
filtro_DW.FilterTaps_DSTATE[0] +
        filtro_U.In3 * filtro_DW.FilterTaps_DSTATE[1];
      if (s7_iter == 1) {
        filtro_Y.Out1 = filtro_U.In3;
      }

      filtro_Y.Out1 = filtro_U.In4 - filtro_B.Sum3;
      if (filtro_P.Adapt_Value) {
        if (filtro_P.Reset_Value) {
          filtro_DW.Correlation_DSTATE[0] =
            filtro_P.Correlation_InitialCondition[0];
          filtro_DW.Correlation_DSTATE[1] =
            filtro_P.Correlation_InitialCondition[1];
          filtro_DW.Correlation_DSTATE[2] =
            filtro_P.Correlation_InitialCondition[2];
          filtro_DW.Correlation_DSTATE[3] =
            filtro_P.Correlation_InitialCondition[3];
        }

        rtb_Gain_idx_0 = (filtro_DW.Correlation_DSTATE[0] +
                          filtro_DW.Correlation_DSTATE[0]) /
          filtro_P.RLSFilter_lambda * filtro_P.Gain_Gain;
        rtb_Gain_idx_1 = (filtro_DW.Correlation_DSTATE[2] +
                          filtro_DW.Correlation_DSTATE[1]) /
          filtro_P.RLSFilter_lambda * filtro_P.Gain_Gain;
        rtb_Gain_idx_2 = (filtro_DW.Correlation_DSTATE[1] +
                          filtro_DW.Correlation_DSTATE[2]) /
          filtro_P.RLSFilter_lambda * filtro_P.Gain_Gain;
        rtb_Gain_idx_3 = (filtro_DW.Correlation_DSTATE[3] +
                          filtro_DW.Correlation_DSTATE[3]) /
          filtro_P.RLSFilter_lambda * filtro_P.Gain_Gain;
        rtb_Product2_idx_0 = rtb_Gain_idx_0 * rtb_Buffer_idx_0 +
rtb_Gain_idx_2 *
          filtro_U.In3;
        filtro_B.rtb_Product2_m = rtb_Gain_idx_1 * rtb_Buffer_idx_0
+
          rtb_Gain_idx_3 * filtro_U.In3;
        rtb_MathFunction_idx_0 = rtb_Product2_idx_0;
        filtro_B.Sum3 = 1.0 / ((rtb_Buffer_idx_0 *
rtb_Product2_idx_0 +
          filtro_U.In3 * filtro_B.rtb_Product2_m) +
filtro_P.Constant_Value);
        rtb_Product2_idx_0 *= filtro_B.Sum3;
```

```
        filtro_B.Sum3 *= filtro_B.rtb_Product2_m;
        filtro_B.Sum2[0] = filtro_B.sum1 * rtb_Product2_idx_0 +
          filtro_DW.FilterTaps_DSTATE[0];
        filtro_B.Sum2[1] = filtro_B.sum1 * filtro_B.Sum3 +
          filtro_DW.FilterTaps_DSTATE[1];
        filtro_DW.Correlation_DSTATE[0] = rtb_Gain_idx_0 -
rtb_Product2_idx_0 *
          rtb_MathFunction_idx_0;
        filtro_DW.Correlation_DSTATE[2] = rtb_Gain_idx_2 -
rtb_Product2_idx_0 *
          filtro_B.rtb_Product2_m;
        filtro_DW.Correlation_DSTATE[1] = rtb_Gain_idx_1 -
filtro_B.Sum3 *
          rtb_MathFunction_idx_0;
        filtro_DW.Correlation_DSTATE[3] = rtb_Gain_idx_3 -
filtro_B.Sum3 *
          filtro_B.rtb_Product2_m;
      }

      filtro_DW.FilterTaps_DSTATE[0] = filtro_B.Sum2[0];
      filtro_DW.FilterTaps_DSTATE[1] = filtro_B.Sum2[1];
      s7_iter++;
    } else {
      exitg1 = 1;
    }
  } while (exitg1 == 0);
}

void filtro_initialize(void)
{

  rt_InitInfAndNaN(sizeof(real_T));

  rtmSetErrorStatus(filtro_M, (NULL));

  (void) memset(((void *) &filtro_B), 0,
                sizeof(B_filtro_T));

  (void) memset((void *)&filtro_DW, 0,
                sizeof(DW_filtro_T));

  (void) memset((void *)&filtro_U, 0,
                sizeof(ExtU_filtro_T));

  (void) memset((void *)&filtro_Y, 0,
                sizeof(ExtY_filtro_T));

  filtro_DW.Correlation_DSTATE[0] =
filtro_P.Correlation_InitialCondition[0];
  filtro_DW.Correlation_DSTATE[1] =
filtro_P.Correlation_InitialCondition[1];
  filtro_DW.Correlation_DSTATE[2] =
filtro_P.Correlation_InitialCondition[2];
```

```
  filtro_DW.Correlation_DSTATE[3] =
filtro_P.Correlation_InitialCondition[3];
  filtro_B.Sum2[0] = filtro_P.Wn_Y0;
  filtro_B.Sum2[1] = filtro_P.Wn_Y0;
  filtro_DW.Buffer_CircBuff = filtro_P.Buffer_ic;

  filtro_DW.FilterTaps_DSTATE[0] =
filtro_P.FilterTaps_InitialCondition;
  filtro_DW.FilterTaps_DSTATE[1] =
filtro_P.FilterTaps_InitialCondition;
  filtro_Y.Out1 = filtro_P.Output_Y0;

  filtro_Y.Out2 = filtro_P.Err_Y0;

}

void filtro_terminate(void)
{
}
```

FILTRO_DATA

```
#include <filtro.h>
#include <filtro_private.h>

const ConstB_filtro_T filtro_ConstB = {
  1.0
};

P_filtro_T filtro_P = {
  0.9999,                                 /* Mask Parameter:
  0.0,                                  /* Expression: flipud(ic(:))
  1.0,                                  /* Expression: 1
  { 10.0, 0.0, 0.0, 10.0 },
  0.5,                                  /* Expression: 1/2
  0.0,                                  /* Computed Parameter:
  0.0,                                  /* Computed Parameter: Err_Y0
  0.0,                                  /* Expression: 0
  0.0,                                  /* Expression: flipud(ic(:))
  1U,                                   /* Computed Parameter:
  1U,                                   /* Computed Parameter:
  1,                                    /* Expression: boolean(1)
  0                                     /* Expression: boolean(0)
};
```