Universidad Zaragoza
1542

Trabajo de Fin de Grado

Diseño e Implementación de un lazo de enganche de fase (PLL) en un microcontrolador

Phase-locked loop (PLL) desing and microcontroller implementation

Autor
Cynthia Manosalvas Pillajo

Directores

José Ignacio Artigas
Luis Ángel Barragán

Departamento de Ingeniería Electrónica y Comunicaciones

2019-2020

Escuela de Ingeniería y Arquitectura
Universidad Zaragoza

Resumen

La sincronización de señales de sistemas de comunicación, eliminación de ruidos o atenuación retrasos son problemas en los se busca soluciones eficientes, ya que resultan de vital importancia para su correcto funcionamiento.

Una solución muy utilizada es elPLL, debido a sus numerosas ventajas. Según sea el problema o aplicación existen diferentes topologías que se pueden aplicar lo que le da gran versatilidad.

En este TFG se presenta los conceptos fundamentales y se hace un análisis para su comprensión. Se eligen dos de sus tipologías para hacer una comparativa entre ellas mediante simulación en Matlab y Simulink, y se implementa la que mejor prestaciones presente en un microcontrolador.

Agradecimientos

La realización de este *tfg* no habría sido posible sin el apoyo de mis directores, José Ignacio Artigas y Luis Ángel Barragán, que me han guiado a través de este proyecto. Mi madre, padre y tía que han impulsado y no han dejado de hacerlo hasta la finalización del mismo. Y en especial, a Pablo que me ha ayudado en todos los aspectos tanto académicos y personales.

Índice general

Índice de figuras

Siglas

APLL Analógico Phase-Locked Loop.

CCS Code Composer Studio.

CPU Unidad Central de Procesamiento.

DCO Digital-controlled Oscillator - Oscilador digital.

DPLL Digital Phase-Locked Loop.

FACTS Flexible AC Transmission System - Sistema flexible de Transmisión AC.

LF Low-pass Filter - Filtro paso bajo.

NCO Numeric-controlled Oscillator - Oscilador numérico.

NVIC Nested Vectored Interrupt Controller.

PD Phase Detector - Detector de fase.

PLL Phase-Locked Loop.

PWM Pulse Width Modulation.

SOGI-PLL Second Order Generalized Integrator Phase-Locked Loop.

STIR Software Trigger Interrupt Register - Registro de interrupción Software Trigger.

TFG Trabajo de Fin de Grado.

VCO Voltage-controlled Oscillator - Oscilador controlado por tensión.

Capítulo 1

Introducción

1.1. Motivación y Contexto

En en los años 20 con los receptores superheterodinos se empeza a dar los primeros pasos en la definición actual del PLL. En esencia, estos receptores consistían en un oscilador local, un mezclador y un amplificador de audio. Para operar el oscilador debía ajustarse exactamente a la misma frecuencia que la señal de entrada, esta se convertía en una frecuencia intermedia de 0 Hz. La salida del mezclador contenía información demodulada. La interferencia no era síncrona con el oscilador y por eso se usaba el amplificador de audio a modo de filtro.

La primera mención del concepto *phaselock* se publica en 1932 [2], donde se utiliza el principio de estos receptores para formalizar la base del mismo.

En los años 50 [3] su desarrollo fue en aumento con la sincronización de receptores en televisión. Junto con el avance de la tecnología analógica y digital, especialmente, el PLL se establece de manera significativa en el campo de la ingeniería electrónica y de telecomunicaciones, debido a sus numerosas aplicaciones y al desarrollo de nuevas topologías.

La estructura básica de un PLL, consta de tres elementos básicos (Figura 1.1):

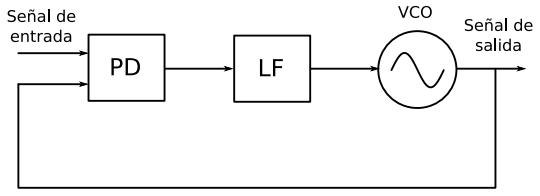


Figura 1.1: Estructura PLL básica

- Detector de fase (PD)
- Filtro paso bajo (LF)
- Oscilador controlado por tensión (VCO)

Una vez que el bucle ha enganchado, el control de tensión fija la frecuencia media del VCO igual a la frecuencia media de la señal de entrada. Para cada ciclo de entrada sólo hay un ciclo de salida del oscilador. El error de fase no será necesariamente cero, se puede tener error de fase en régimen permanente y error de seguimiento de la fase.

El objetivo del diseño es elegir el filtro apropiado que cumpla con los requisitos deseados en régimen permanente y transitorio.

Esta presentación básica de su estructura y funcionamiento es el primer paso para poder entender su importancia en el ámbito de la ingeniería y la motivación del TFG. Su diseño y construcción, así como implementación reunen buena parte de las habilidades adquiridas durante el grado y culminar así con un Trabajo de Fin de Grado que las aplique y además sea un complemento a la formación.

Algunos ejemplos de sus aplicaciones son, la sincronización de señales de datos, receptores de televisión, atenuación de retrasos, control de sistemas digitales, o control de velocidad de motores, entre otros. Esto indica, de nuevo, su multifuncionalidad en un gran abanico de aplicaciones y ámbitos en la ingeniería.

1.2. Objetivos y alcance

Una vez enunciado el contexto de trabajo, el objetivo de este TFG es probar distintas topologías del PLL para estudiar su respuesta, así como su margen de enganche. Simular su comportamiento con la herramienta de Matlab, Simulink, e implementar una de estas topologías en coma flotante con un microcontrolador de *Texas Instruments*, en lenguaje C, en el entorno de desarrollo Code Composer Studio (CCS). Además de comprobar su funcionamiento de forma experimental en el laboratorio.

Para poder alcanzar con éxito el objetivo se ha hecho un estudio previo de la estructura del PLL, un análisis matemático del mismo, así como el análisis de sus distintas topologías y métodos de desarrollo. También ha sido necesario la familiarización con el entorno de desarrollo Matlab/Simulink, CCS, y el funcionamiento del microcontrolador, manejo de sus periféricos, interrupciones, temporizadores y fuentes de reloj, principalmente.

1.3. Estructura de la memoria

La organización del resto de la memoria se estructura en cinco capítulos.

- Capítulo 2: estudio del PLL.
- Capítulo 3: diseño del PLL y simulación.
- Capítulo 4: microcontrolador y partes de interés para el TFG.
- Capítulo 5: implementación y experimentación.
- Capítulo 6: conclusiones y líneas futuras.

Capítulo 2

Estudio del PLL

En este capítulo se profundiza en la comprensión y análisis de la estructura del PLL.

2.1. Introducción

El Phase-Locked Loop (PLL) es un sistema realimentado [3] cuyo principal objetivo es la sincronización de fase entre una señal de entrada, referencia, y su salida, dentro de un margen determinado, mediante la comparación de fases. El diagrama de bloques básico se muestra en la Figura 2.1

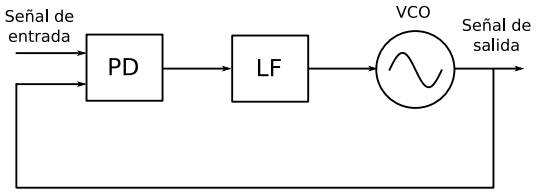


Figura 2.1: Diagrama básico de bloques PLL

Este sistema está compuesto por tres bloques básicos:

- Detector de fase (PD)
- Filtro paso bajo (LF)
- Oscilador controlado por tensión (VCO)

Para conseguir la sincronización deseada, la señal del oscilador cambia su frecuencia en respuesta a la entrada, controlada por una tensión.

El detector de fase compara la referencia con la salida del VCO, y genera una señal que cambia en proporción a la diferencia de fases. Ésta es procesada por el filtro cuya salida entra al VCO. Se repite el proceso hasta que se alcanza la sincronización, y el PLL alcanza su estado de enganchado (*locked*).

Los dos tipos básicos de PLL son los analógicos (APLL) y digitales (DPLL). Su implementación difiere en sus parámetros y tiempos de respuesta [4]. Los digitales tienen menos tiempo de respuesta con respecto a los analógicos, son más inmunes a cambios bruscos de la entrada y al ruido. Sin embargo, al tener más bloques son más caros que los analógicos.

En este TFG se va a diseñar tipologías digitales, como son la básica y SOGI-PLL, que se desarrollan y explican en el capítulo 3.

El PLL tiene un gran abanico de aplicaciones [3, 5, 6, 7] debido a su gran utilidad. Algunas de ellas son la sincronización para sistemas de comunicación, sintetizadores de frecuencia [8], aplicaciones biomédicas, como la implantación de dispositivos biomédicos [9], aplicaciones FACTS [10]; acondicionamiento de alimentación, conexión de sistemas de energías renovables a la red como los sistemas fotovoltaicos, o eólicos; o sistemas que trabajan a altas frecuencias [11].

2.2. Ecuaciones básicas/Análisis del PLL

Para poder entender el concepto del PLL es necesario realizar un análisis teórico del mismo.

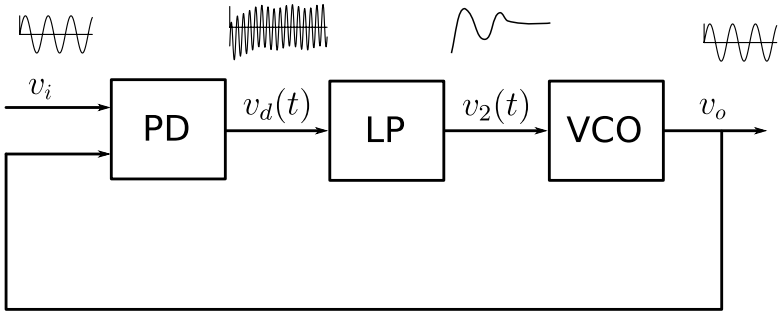


Figura 2.2: Diagrama de bloques básico

Sea la entrada del sistema PLL, V_{in}, una señal sinusoidal general

$$v_i = V_i \sin(\omega_i t + \theta_i(t)) \tag{2.1}$$

La señal de salida del bloque VCO

$$v_o = V_o \cos(\omega_o t + \theta_o(t)) \tag{2.2}$$

Suponiendo el caso básico en el que el bloque PD es un multiplicador de ganancia K_m, su salida se define como

$$v_d = K_m v_i(t) v_o(t) = K_m A[\sin(\omega_i t + \theta_i(t)) \cos(\omega_o t + \theta_o(t))] \tag{2.3}$$

donde K_m es la ganancia del PD y $A = V_i V_o$.

Utilizando la propiedad del producto del seno y coseno en términos de los anteriores parámetros,

$$\begin{aligned} 2\sin(a)\cos(b) &= 2\sin(\omega_i t + \theta_i(t))\cos(\omega_o t + \theta_o(t)) \\ &= \sin(\omega_i t + \theta_i(t) + \omega_o t + \theta_o(t)) + \sin(\omega_i t + \theta_i(t) - \omega_o t - \theta_o(t)) \end{aligned} \tag{2.4}$$

Reagrupando términos se obtiene

$$2\sin(a)\cos(b) = \sin[(\omega_i + \omega_o)t + \theta_i(t) + \theta_o(t))] + \sin[(\omega_i - \omega_o)t + \theta_i(t) - \theta_o(t)] \tag{2.5}$$

Operando con (2.5) se obtiene

$$v_d(t) = \frac{1}{2} K_m A[\sin[(\omega_i + \omega_o)t + \theta_i(t) + \theta_o(t))] + \sin[(\omega_i - \omega_o)t + \theta_i(t) - \theta_o(t)] \tag{2.6}$$

La señal de salida $v_d(t)$ comprende dos componentes, $\omega_i + \omega_o$ y $\omega_i - \omega_o$. Cuando el PLL aun no ha alcanzado la sincronización, estas frecuencias son distintas, $\omega_i \neq \omega_o$. Ambas se sitúan en la banda atenuada del filtro y el primer término del seno de (2.6) es atenuado por el filtro paso bajo.

La salida del bloque del filtro paso bajo es

$$v_2(t) = K_d \sin[(\omega_i - \omega_o)t + \theta_i(t) - \theta_o(t))] \tag{2.7}$$

con $K_d = \frac{1}{2} K_m A[\text{V/rad}]$.

Haciendo la siguiente sustitución en 2.7, $\theta_d(t) = \theta_i(t) - \theta_o(t)$ y $\omega_d = \omega_i - \omega_o$, se obtiene

$$v_2(t) = K_d \sin(\omega_d t + \theta_d(t)) \tag{2.8}$$

El VCO verifica la relación [12],

$$\omega_o = \omega_c + K_{vco} v_2(t) \tag{2.9}$$

donde ω_c es la frecuencia central del VCO y K_{vco} es la ganancia del mismo.

Cuando el PLL está sintonizado la frecuencia de entrada y salida son aproximadamente iguales, $\omega_1 \approx \omega_o$. La ecuación (2.8) queda

$$v_2(t) = K_d \sin(\theta_d(t)) \tag{2.10}$$

La solución se puede expresar tanto en el dominio del tiempo como de la frecuencia [3]. Aunque ambas resulten de interés, en el dominio del tiempo tenemos un sistema no lineal mientras que en el caso del frecuencial es lineal como se demuestra a continuación. En el siguiente apartado se desarrolla el modelo de pequeña señal del PLL. Este modelo se utilizará en el capítulo siguiente para diseñar el controlador que asegure un comportamiento adecuado del lazo de realimentación.

2.3. Modelo de pequeña señal

Las señales del diagrama de la Figura 2.2 se pasan al dominio frecuencial, para este caso se toma como entrada θ_i y salida θ_o.

Para valores pequeños de θ_d se puede aproximar que $\sin\theta_d = \theta_d$ de forma que la tensión de salida del PD se define

$$V_d(s) = K_d(\theta_i(s) - \theta_o(s)) \tag{2.11}$$

La función de transferencia del LF se denomina $F(s)$. La salida del filtro que controla la frecuencia del VCO al pasar al dominio frecuencial es

$$V_2(s) = F(s)V_d(s) \tag{2.12}$$

Para poder obtener θ_o a la salida, partiremos de (2.9), donde se observa que la desviación de la frecuencia central del VCO, $\Delta\omega$, es la señal de control del mismo, es decir,

$$\Delta\omega = K_{vco} v_2(t) \rightarrow \frac{d\theta_o(t)}{dt} = K_{vco} v_2(t) \tag{2.13}$$

Aplicando la trasformada de Laplace a cada término de 2.13 se obtiene

$$s\theta_o(s) = K_{vco} V_2(s) \rightarrow \theta_o(s) = \frac{K_{vco} V_2(s)}{s} \tag{2.14}$$

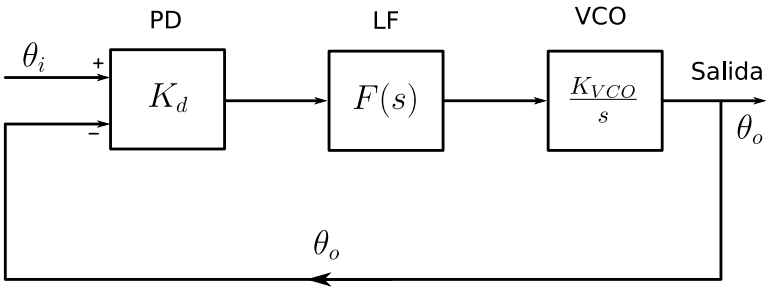


Figura 2.3: PLL básico, modelo pequeña señal

donde $s = \sigma + j\omega$ es la variable independiente de Laplace. Es digno de mención que $\frac{1}{s}$ es la transformada de Laplace de un integrador, por lo cual la fase del VCO es proporcional a la integral a la tensión de control.

El diagrama de bloques queda como aparece en la Figura 2.3.

A continuación se obtiene las funciones de transferencia del PLL tras el análisis.

- Función de transferencia en bucle abierto:

$$G(s) = \frac{\theta_o(s)}{\theta_e(s)} = \frac{K_{vco}K_dF(s)}{s} \tag{2.15}$$

- Función de transferencia en bucle cerrado:

$$H(s) = \frac{\theta_o(s)}{\theta_i(s)} = \frac{G(s)}{1+G(s)} = \frac{K_{vco}K_dF(s)}{s+K_{vco}K_dF(s)} \tag{2.16}$$

- Función de transferencia del error:

$$E(s) = \frac{\theta_e(s)}{\theta_i(s)} = \frac{1}{1+G(s)} = 1 - H(s) = \frac{s}{s+K_{vco}K_dF(s)} \tag{2.17}$$

2.4. Márgenes de enganche y de captura

Existen unos márgenes de trabajo entre las cuales el PLL se encuentra en sintonía o es capaz de sincronizar, estos márgenes son el margen de enganche y de captura, respectivamente.

El margen de captura ($\Delta\omega_H$) es el rango en el que el PLL es capaz de engancharse a una señal inicialmente no sincronizada. Se puede calcular para el PLL clásico como [5]:

$$\Delta\omega_H = K_{vco}K_dF(0) \tag{2.18}$$

El margen de enganche ($\Delta\omega_L$) , rango de frecuencia en que la señal permanece enganchada, se puede estimar [5] como:

$$\Delta\omega_L \approx K_{vco}K_dF(\infty) \tag{2.19}$$

El margen de captura es siempre menor que el margen de enganche, y ambos están centrados respecto de la frecuencia central del VCO, como se puede ver en la Figura 2.4

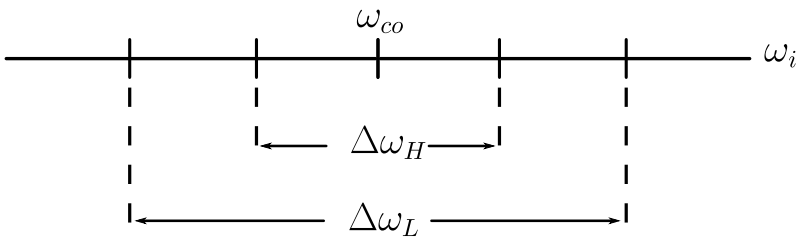


Figura 2.4: Márgenes de enganche y captura

2.5. Bloques del PLL básico

2.5.1. Detector de fase (PD)

Es un circuito capaz de generar una señal proporcional al desfase entre dos señales de entrada [13]. Se pueden distinguir dos tipos:

- Dispositivos multiplicadores.
  Se usan exclusivamente en PLLs lineales. El error DC de la salida es la media del producto de forma de onda de la señal de entrada y la forma de onda del oscilador. Con un diseño apropiado son capaces de trabajar con señales de entrada con ruido.

- Dispositivos secuenciales.
  Genera el error de la tensión de salida dependiente solo del intervalo de tiempo entre una transición de la forma de onda de las señal y de la forma de onda del VCO. Son circuitos digitales lógicos lo que les permite operar con forma de ondas rectangulares binarias. Algunos ejemplos son los detectores *XOR*, *JK-señales* o los basados en frecuencia con salida en tensión o corriente.

2.5.2. Filtro paso bajo (LF)

Como se pudo observar en la Figura 2.3, la función de transferencia de LF se definió de manera genérica como $F(s)$. Este filtro paso bajo tiene como objetivo filtrar la frecuencia suma de entrada y salida y de esa manera conseguir la sincronía que se busca. Este filtro puede ser de tipo activo o pasivo, de primer, segundo, tercer, u orden superior, o combinación de los mismos, según sean los requisitos de nuestro diseño.

2.5.3. Oscilador controlado por tensión (VCO)

El VCO es una parte esencial en todo PLL. Es un dispositivo no lineal que genera una oscilación periódica [5]. Esta oscilación es controlada por una tensión para reducir el error de fase hasta que sea 0 o próximo a 0, ajustando así la frecuencia del mismo con respecto a la de la entrada. Para los PLLs digitales el VCO puede ser sustituido por un oscilador controlado numéricamente (NCO) o digitalmente (DCO), donde la tensión es reemplazada por un valor digital y su salida es una onda digital oscilante.

Capítulo 3

Diseño del PLL y Simulación

3.1. Introducción

En este capítulo se explica el diseño de las topologías PLL básico y SOGI-PLL. Se hace una comparativa entre ellas y se elige el método a implementar.

3.2. Topologías

3.2.1. *PLL-básico*

Esta topología tiene como detector de fase un multiplicador [14].

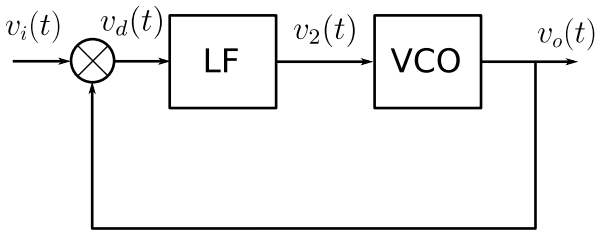


Figura 3.1: Diagrama de bloques del PLL-básico

Se reconstruye la señal de entrada a partir de su componente fundamental estimando su amplitud, fase y frecuencia.

En la Figura 3.1, el bloque LF de filtrado suele incluir un controlador C que asegure ciertas prestaciones al lazo cerrado.

Presenta inmunidad a pequeñas variaciones de la señal de entrada o internas, ruidos, armónicos o variaciones de frecuencia pequeñas. Se aplica en alimentación de sistemas electrónicos, detección de picos de tensión o detección de perturbaciones, entre otros.

A continuación, se van a desarrollar tres diseños partiendo de la función de transferencia en bucle abierto (2.15) vista en el capítulo anterior, utilizando la herramienta *Sisotool* de Matlab.

La señal de entrada de todos los diseños será una señal sinusoidal de 50 Hz. La herramienta tiene como parámetros el margen de fase deseado al igual que el ancho de banda. Se establece

para los diseños un margen de fase de 60^{o} y un ancho de banda de 20 rad/s, para que el desfase introducido por el filtro sea pequeño.

- Diseño 1

  Sea

  $$G(s) = \frac{K_{vco} K_d F(s)}{s} \tag{3.1}$$

  Se usa como filtro un controlador PI de la forma

  $$F(s) = K_p + K_i \frac{1}{s} = \frac{K_p s + K_i}{s} \tag{3.2}$$

  Con $K_d = 0{,}5$ y $K_{vco} = 0{,}1$, se obtiene que $K_p = 11{,}547$ y $K_i = 346{,}4102 \cdot K_p$

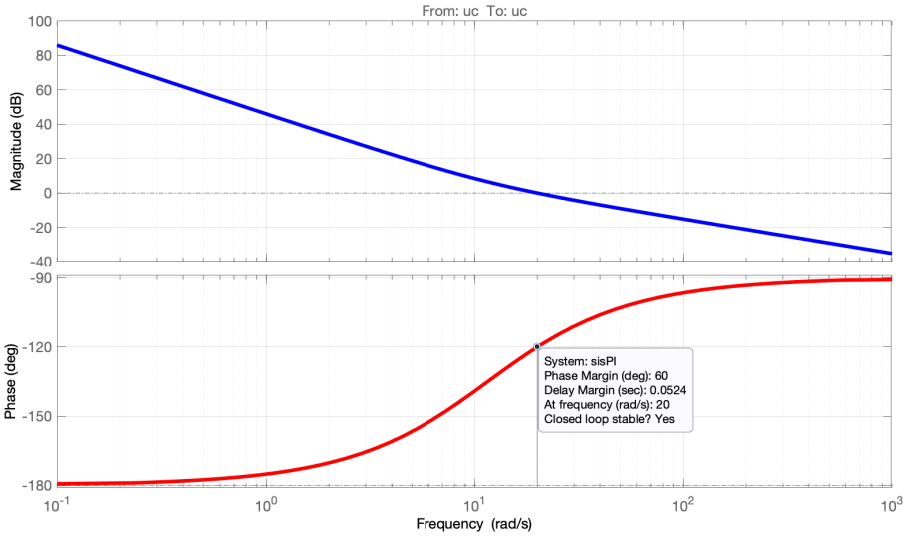


Figura 3.2: Diagrama de bode PLL-básico (PI)

Mirando el diagrama de bode (Figura 3.2), se observa que se cumple con las especificaciones impuestas al sistema y este es estable en bucle cerrado.

- Diseño 2

  En este diseño se va a implementar un filtro de primer orden (3.3), y un PI.

  $$F(s) = \frac{1}{1 + \tau s} \quad , \quad \tau = \frac{1}{\omega_c} \tag{3.3}$$

  La frecuencia de corte del filtro, ω_c, debe ser menor que ω_i para reducir la influencia del nivel de continua de la entrada [15].

  Se define de nuevo $G(s)$ incluyendo el filtro de primer orden

  $$G(s) = \frac{K_{vco} K_d}{s(1 + \tau s)} \tag{3.4}$$

  Con los mismos valores de K_d, K_{vco} del diseño 1, y $\omega_c = 2\pi 10$, se obtiene $K_p = 4{,}3765$ y $K_i = 410{,}0721 \cdot K_p$.

  Mirando el diagrama de bode de la Figura 3.3, se observa que se cumple con las especificaciones impuestas al sistema y este es estable en bucle cerrado.

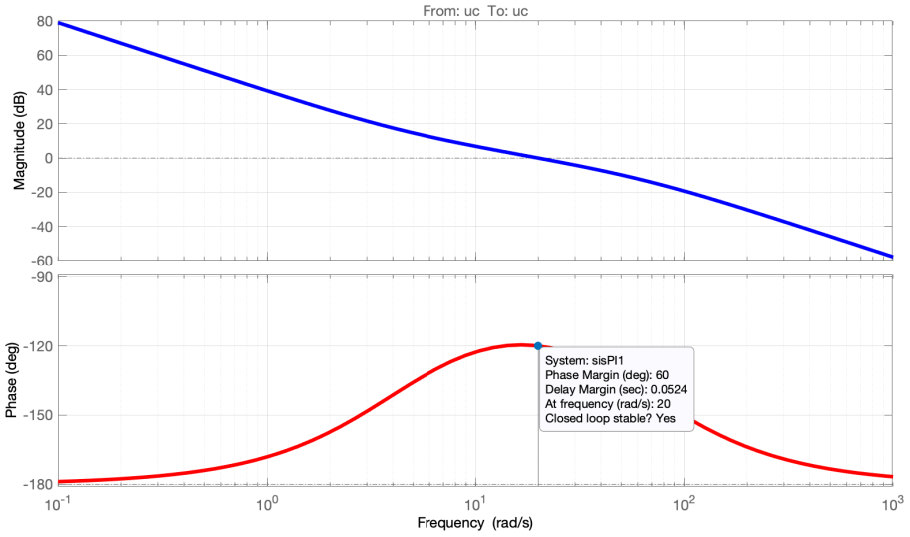


Figura 3.3: Diagrama de bode PLL-básico, (PI+1^{o}orden)

- Diseño 3

  En este diseño se va a implementar un filtro paso bajo con un filtro *Butterworth* de orden 2 (3.5), y un PI.

$$F(s) = \frac{a_0 + a_1 s + a_2 s^2}{b_0 + b_1 s + b_2 s^2} \tag{3.5}$$

  Los coeficientes se obtienen con la función *butter* de Matlab tomando $f_c = 10$ Hz.

  Se define de nuevo $G(s)$ incluyendo el filtro de segundo orden

$$G(s) = \frac{K_{vco} K_d}{s} \left(\frac{a_0 + a_1 s + a_2 s^2}{b_0 + b_1 s + b_2 s^2} \right) \tag{3.6}$$

  Con los mismos valores de K_d, K_{vco} del diseño 1, se obtiene $K_p = 0{,}35$ y $K_i = 401{,}9782 \cdot K_p$ Mirando el diagrama de bode Figura 3.4, se observa que se cumple con las especificaciones impuestas al sistema y este es estable en bucle cerrado.

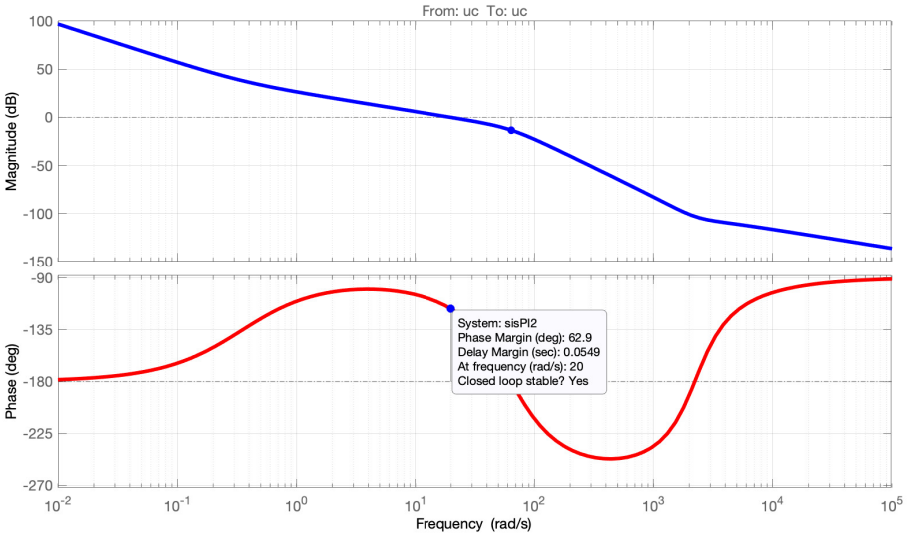


Figura 3.4: Diagrama de bode PLL-básico (PI+2^{o}orden)

3.2.2. *SOGI-PLL*

El uso del SOGI-PLL ha crecido en los últimos años desde su publicación en [16], debido a las deficiencias del PLL-básico, como errores en la estimación de la frecuencia, poca inmunidad al ruido o pérdida de estabilidad en cambios en la frecuencia o fluctuaciones de la señal de entrada.

Algunas de sus ventajas frente a otras topologías, son la no inclusión de retraso, mayor inmunidad ante cambios de frecuencia [17] o su uso en sistemas trifásicos [18]. En este apartado se va desarrollar la aplicación en un sistema de una sola fase (*single-phase SOGI*).

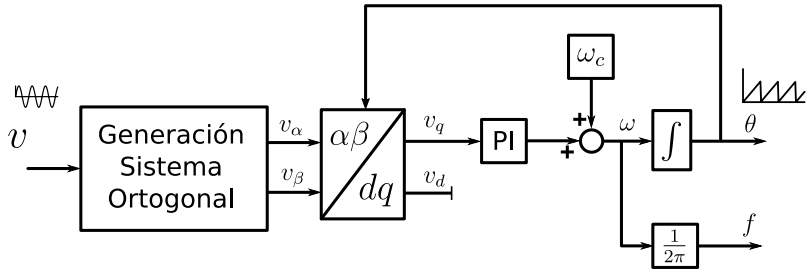


Figura 3.5: Diagrama de bloques del SOGI-PLL

La estructura de la Figura 3.5 correspondiente al SOGI (bloque generación sistema ortogonal) genera un sistema ortogonal de tensiones v_α y v_β a partir de la tensión de entrada. Este sistema puede obtenerse mediante métodos como la transformada de Clark, Hilbert, o la inversa de Park, entre otras [19]. Las señales de salida v_α y v_β tienen un desfase entre ellas de 90^{o}, y v_α tiene la misma fase y magnitud que la componente fundamental de la señal de entrada como se puede observar en la Figura 3.6) del modelo del pequeña señal de la estructura del SOGI.

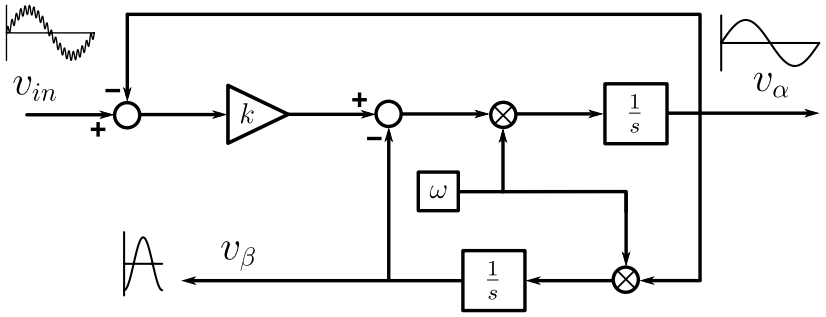


Figura 3.6: Diagrama de bloques del modelo de pequeña SOGI

A continuación se usa la transformada de Park, cuya relación entre la entrada y salida es [19]

$$\begin{bmatrix} v_d \\ v_q \end{bmatrix} = \begin{bmatrix} \sin\theta & \cos\theta \\ -\cos\theta & \sin\theta \end{bmatrix} \begin{bmatrix} v_\alpha \\ v_\beta \end{bmatrix} \tag{3.7}$$

Este método es una alternativa para generar un sistema ortogonal de tensión comparado con otros [19, 20] que son más complejos, no lineales, con ningún o poco filtrado y dependientes de la frecuencia. Además su implementación es simple [16].

La función de transferencia en bucle abierto del bloque SOGI es $GI(s)$

$$GI(s) = \frac{\omega s}{s^2 + \omega^2} \tag{3.8}$$

donde ω es la frecuencia de resonancia del *SOGI*

Las funciones en bucle cerrado se pueden obtener aplicando las propiedades de los bloques

$$G_\alpha(s) = \frac{v_\alpha(s)}{v_{in}(s)} = \frac{k\omega s}{s^2 + k\omega s + \omega^2} \tag{3.9}$$

$$G_\beta(s) = \frac{v_\beta(s)}{v_{in}(s)} = \frac{k\omega^2}{s^2 + k\omega s + \omega^2} \tag{3.10}$$

donde k es el ganancia que afectará al ancho de banda.

La ganancia regula el nivel de filtrado del sistema, si ésta es pequeña($k < 1$) el filtrado será mayor pero se ralentiza el sistema. Por el contrario, si esta es mayor el filtrado es menor pero el sistema es más rápido.

El diseño del filtro paso bajo es igual al diseño 1 del PLL-básico. Utilizando de nuevo la herramienta *Sisotool* de Matlab con los mismos parámetros de ancho de banda y desfase, partiendo de la linealización del SOGI [21].

$$F_{sogi} = \frac{1}{\tau_{sogi} \cdot s + 1} \quad , \quad \tau_{sogi} = \frac{2}{k\omega} \tag{3.11}$$

Se define de nuevo $G(s)$ con la estructura del SOGI

$$G(s) = \frac{K_{vco} K_d}{s} \left(\frac{1}{\tau_{sogi} \cdot s + 1} \right) \tag{3.12}$$

Se obtiene $K_p = 0{,}35$ y $K_i = 323{,}7491 \cdot K_p$

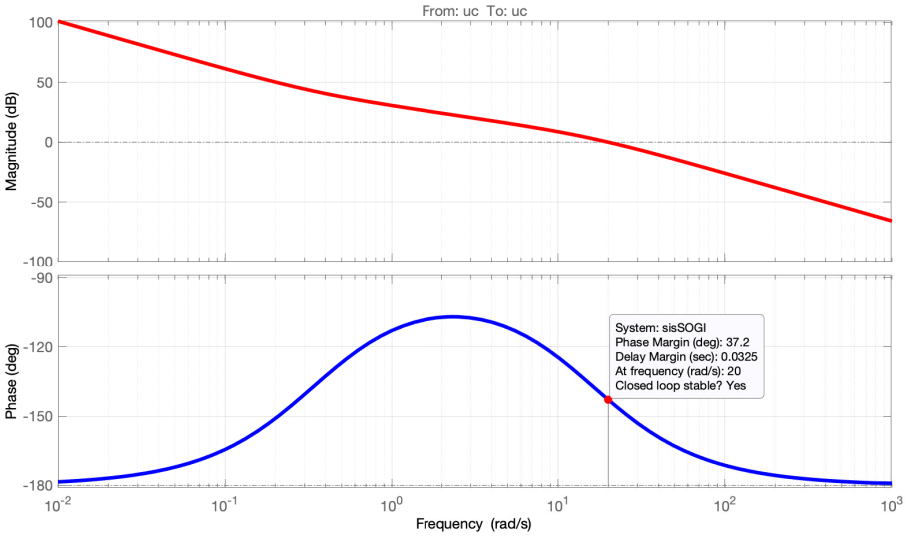


Figura 3.7: Diagrama de bode SOGI-PLL

3.2.3. Análisis resultados simulación

En este apartado se va a comparar la respuesta de cada diseño. Como se enunció en el capítulo anterior los márgenes de trabajo son función del filtro paso bajo, y en las siguientes figuras se observa esa influencia.

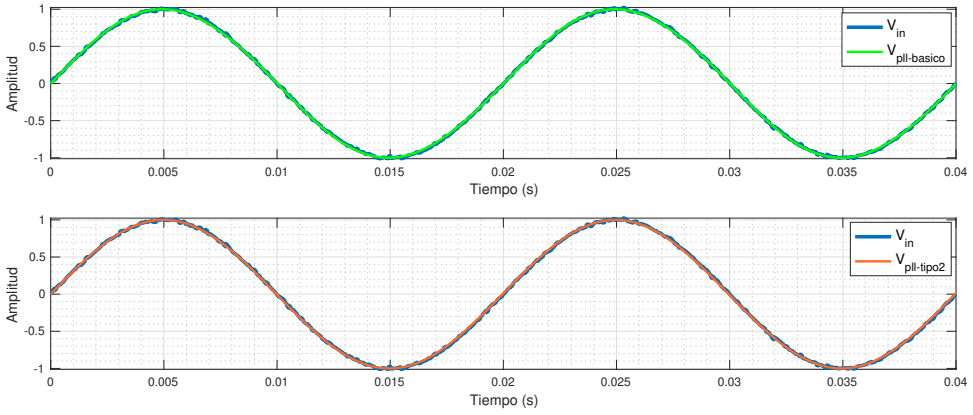


Figura 3.8: Tensión de entrada y salida, diseño 1 y 2

En la Figura 3.8 se observa el enganche de la tensión de salida del diseño 1 y 2 con una entrada sinusoidal con ruido tenue. En ambos casos el ruido es atenuado en la salida.

En la Figura 3.9 se observa el enganche de la tensión de salida del diseño 3 y SOGI con la misma señal entrada sinusoidal anterior. En ambos casos el ruido es atenuado en la salida, y su respuesta es mejor que en los diseños anteriores, ya que en los pasos por cero el error es mucho menor. Esto se observa mejor en las siguientes figuras.

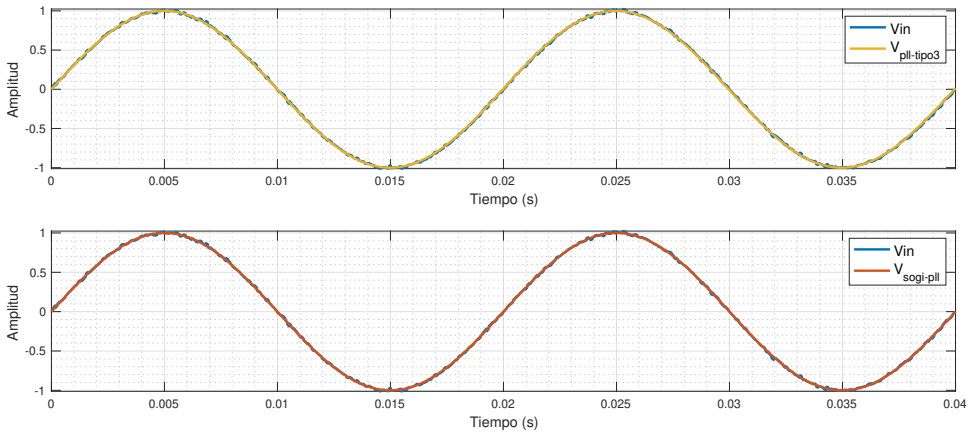


Figura 3.9: Tensión de entrada y salida, diseño 3 y SOGI

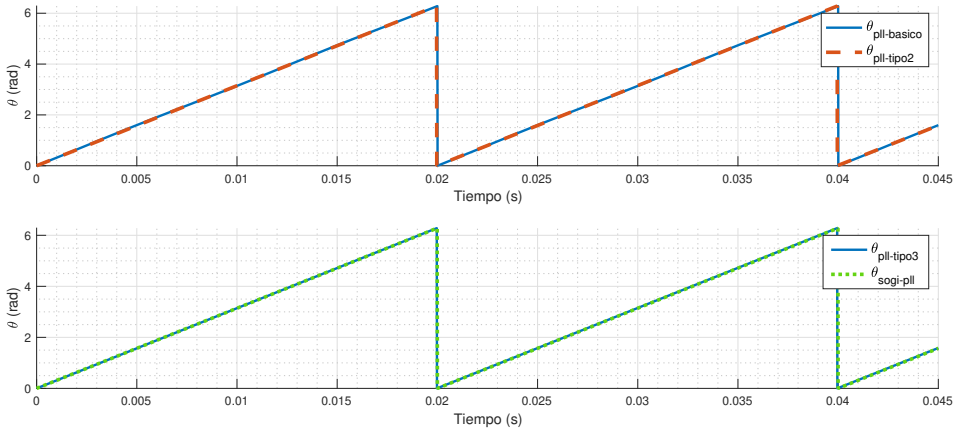


Figura 3.10: θ_{out}

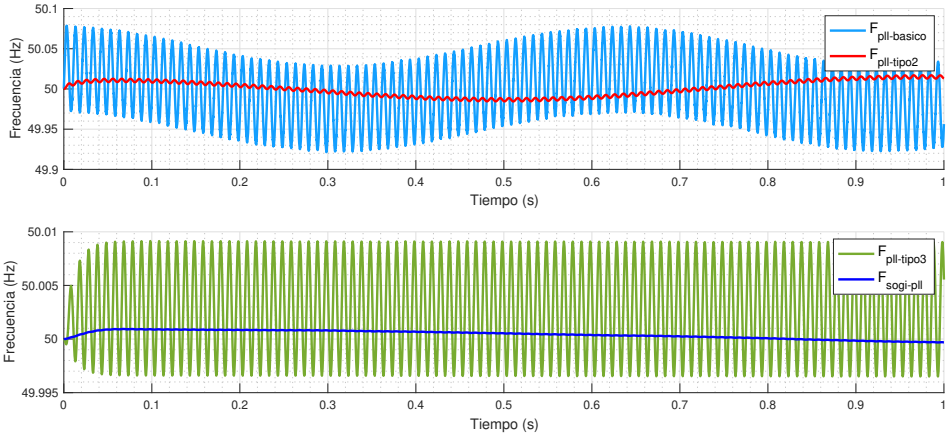


Figura 3.11: Frecuencia de salida

En la Figura 3.10 donde se observa el ángulo de salida se puede ver como .....

En la Figura 3.11 se observa de manera más clara la diferencia entre los 4 diseños. Aunque en los 4 casos el PLL engancha con la señal de entrada, y en media la frecucia de salida es 50 Hz, el error difiere para cada uno de ellos. Observando la frecuencia del Pll-básico este error oscila entre 49.92 y 50.08, para el Pll-básico tipo 2 esta variación es menor, entre 49.98 y 50.02. El error de Pll-básico tipo 3 disminuye aún más, entre 49.997 y 50.009. Sin embargo, para el diseño del SOGI-PLL este error oscila entre 49.999 y 50.001 .

En conclusión, conforme se eleva el orden del filtro paso bajo mejor es la respuesta y menor el error. El caso del SOGI es el que menor error tiene y cuya variabilidad es del orden de 0.1. Por tanto es esta tipología la que se va a implementar en el microcontrolador MSP432 P401R.

Capítulo 4

Microcontrolador MSP432

4.1. Introducción

Para el desarrollo de nuestro algoritmo del PLL, se ha elegido el *MSP432 LaucnhPad, Development Kit* de Texas Instruments (Figura 4.1).

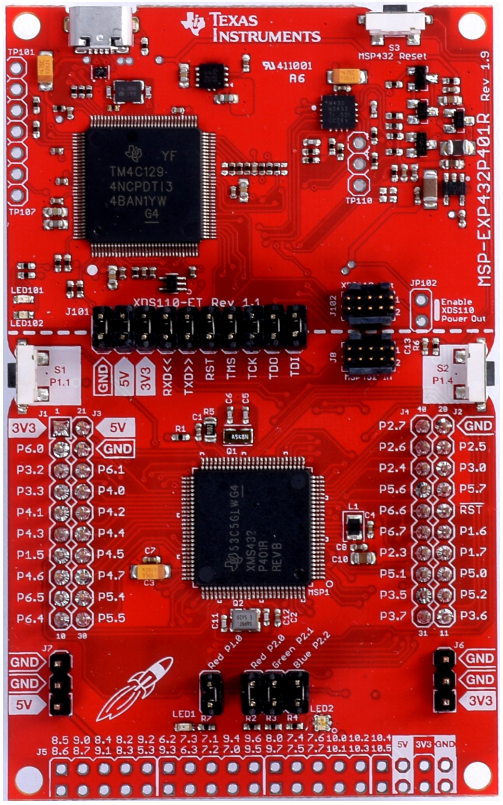


Figura 4.1: MSP-EXP432P401R LaunchPad™ Development Kit [1]

Su microcontrolador es el MSP432 P401R. Es de bajo consumo y cuenta con un núcleo Arm® de 32-bit, Cortex®-M4F. Es un módulo de evaluación fácil de usar, contiene todo lo necesario para desarrollar, depurar la programación y el código, y medir consumos. Este dispositivo admite aplicaciones de bajo consumo que requieren una velocidad de CPU elevada, memoria, puertos analógicos

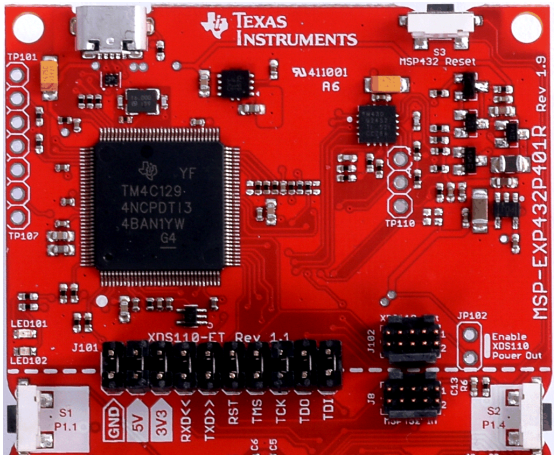


Figura 4.2: Placa de prueba y depuración [1]

entre otros. Cuenta con 40 pines, su núcleo es de bajo consumo con un reloj de hasta 48 MHz, y se puede añadir nuevos módulos como el *BoosterPack* de *Texas Instruments*.

Tiene dos botones y dos LEDs para la interacción con el usuario, además de un canal UART por USB para ordenador. Una memoria flash de 256 KB, 64 KB de SRAM y 32 KB de ROM. Cuatro temporizadores de 16 bits con captura, comparación o PWM, y dos temporizadores de 32 bits y un RTC. Tiene 8 canales de comunicación serie tipo I2C, SPI, UART, e IrDA. Cuenta, además, con un módulo ADC de precisión.

Este kit tiene como soporte entornos de desarrollo profesionales como *Code Composer Studio*, *Keil µVision* e *IAR Embedded Workbench*. En este TFG se utilizará CCS.

Además, la placa de prueba y depuración (*Onboard Debug Probe*, Figura 4.2) que incorpora elimina la necesidad de programadores caros y soporta casi todos los dispositivos derivados de los Arm que incorpora *Texas Instruments* en sus dispositivos.

4.2. Características

En esta apartado se van a exponer las características del microcontrolador que resultan de especial interés en el TFG.

En la Figura 4.3 aparecen todas la funciones que tiene el microcontrolador.

4.2.1. Núcleo

El núcleo Arm® 32-bit Cortex®-M4F CPU (Figura 4.4) está basado en la arquitectura ArmV7-M, cuenta con una unidad de coma flotante, que permite operaciones de procesamiento de datos (C float) de simple precisión. El convertidor analógico-digital tiene una precisión de 14 bits y tiene una velocidad de 1 Msps. Cuenta con fuentes de reloj flexibles, y dos temporizadores de 32 bits con capacidad para generar interrupciones y otros cuatro temporizadores de propósito general.

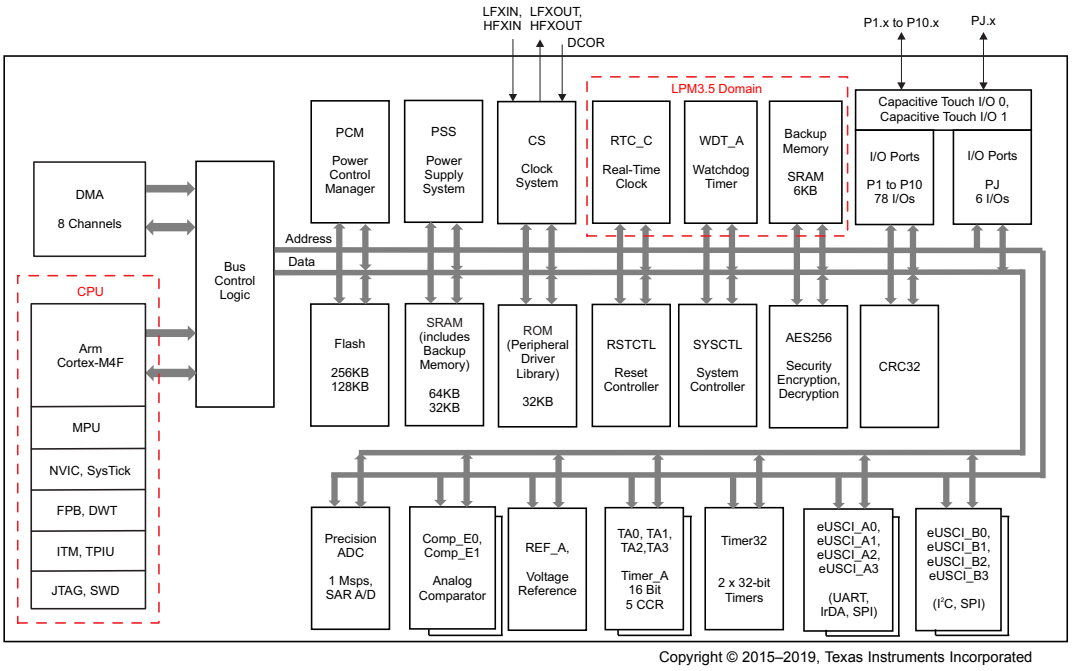


Figura 4.3: MSP432P401R Diagrama de Bloque Funciones [1]

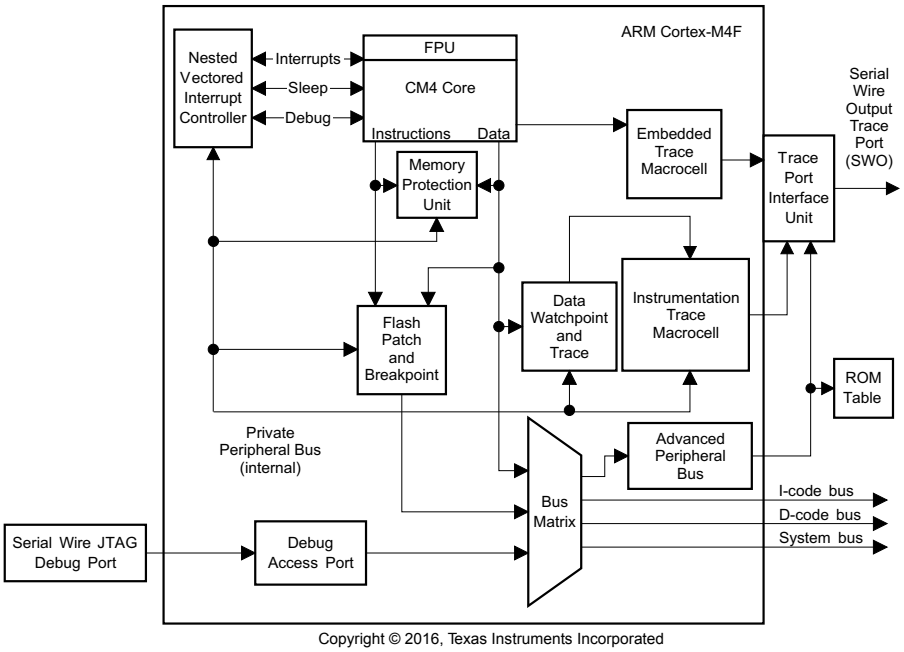


Figura 4.4: Diagrama de Bloques CPU [1]

4.2.2. Interrupciones

Las interrupciones se manejan mediante un bloque Nested Vectored Interrupt Controller (NVIC). Puede generarse hasta un total de 64 interrupciones. Cada interrupción tiene un nivel de prioridad programable codificada de 0 a 7, donde el 7 corresponde al nivel más bajo de prioridad y el 0 al más alto. Esta prioridad se puede cambiar de forma dinámica, y se pueden agrupar por niveles de prioridad. La detección de las señales de interrupción puede ser por nivel o por pulso, y este controlador cuenta además con interrupción no enmascarable (NMI) externa. Los resgistros del NVIC, que aparecen en aparecen en la Figura 4.5, pueden ser de tres tipos: read-write(escritura-lectura), read-only (sólo lectura) o write-only (sólo escritura).

La CPU lanza todas las interrupciones, los periféricos de estas interrupciones pasan al estado pendiente si el NVIC detecta una señal de interrupción en alto y no se activa o si detecta un flanco en la misma. El software escribe la interrupción pendiente en el bit correspondiente o en el STIR para generar una interrupción pendiente por software. La interrupción deja de estar pendiente cuando el procesador maneja la ISR y pasa a estar en activo. Entonces, dependiendo de si la interrupción es por nivel o por pulso procederá de una manera u otra para manejarla y desactivar el bit correspondiente.

Offset	Acronym	Register Name	Type	Reset	Section
100h	ISER0	Irq 0 to 31 Set Enable Register	read-write	00000000h	Section 2.4.3.1
104h	ISER1	Irq 32 to 63 Set Enable Register	read-write	00000000h	Section 2.4.3.2
180h	ICER0	Irq 0 to 31 Clear Enable Register	read-write	00000000h	Section 2.4.3.3
184h	ICER1	Irq 32 to 63 Clear Enable Register	read-write	00000000h	Section 2.4.3.4
200h	ISPR0	Irq 0 to 31 Set Pending Register	read-write	00000000h	Section 2.4.3.5
204h	ISPR1	Irq 32 to 63 Set Pending Register	read-write	00000000h	Section 2.4.3.6
280h	ICPR0	Irq 0 to 31 Clear Pending Register	read-write	00000000h	Section 2.4.3.7
284h	ICPR1	Irq 32 to 63 Clear Pending Register	read-write	00000000h	Section 2.4.3.8
300h	IABR0	Irq 0 to 31 Active Bit Register	read-only	00000000h	Section 2.4.3.9
304h	IABR1	Irq 32 to 63 Active Bit Register	read-only	00000000h	Section 2.4.3.10
400h	IPR0	Irq 0 to 3 Priority Register	read-write	00000000h	Section 2.4.3.11
404h	IPR1	Irq 4 to 7 Priority Register	read-write	00000000h	Section 2.4.3.12
408h	IPR2	Irq 8 to 11 Priority Register	read-write	00000000h	Section 2.4.3.13
40Ch	IPR3	Irq 12 to 15 Priority Register	read-write	00000000h	Section 2.4.3.14
410h	IPR4	Irq 16 to 19 Priority Register	read-write	00000000h	Section 2.4.3.15
414h	IPR5	Irq 20 to 23 Priority Register	read-write	00000000h	Section 2.4.3.16
418h	IPR6	Irq 24 to 27 Priority Register	read-write	00000000h	Section 2.4.3.17
41Ch	IPR7	Irq 28 to 31 Priority Register	read-write	00000000h	Section 2.4.3.18
420h	IPR8	Irq 32 to 35 Priority Register	read-write	00000000h	Section 2.4.3.19
424h	IPR9	Irq 36 to 39 Priority Register	read-write	00000000h	Section 2.4.3.20
428h	IPR10	Irq 40 to 43 Priority Register	read-write	00000000h	Section 2.4.3.21
42Ch	IPR11	Irq 44 to 47 Priority Register	read-write	00000000h	Section 2.4.3.22
430h	IPR12	Irq 48 to 51 Priority Register	read-write	00000000h	Section 2.4.3.23
434h	IPR13	Irq 52 to 55 Priority Register	read-write	00000000h	Section 2.4.3.24
438h	IPR14	Irq 56 to 59 Priority Register	read-write	00000000h	Section 2.4.3.25
43Ch	IPR15	Irq 60 to 63 Priority Register	read-write	00000000h	Section 2.4.3.26
F00h	STIR	Software Trigger Interrupt Register	write-only	00000000h	Section 2.4.3.27

Figura 4.5: Registro NVIC [1]

4.2.3. Fuentes de reloj

El modulo del sistema del reloj admite sistemas de bajo coste y de bajo consumo. El modulo del reloj se puede configurar para operar sin ningún otro componente externo, con 2 cristales externos o con resonadores, o con una resistencia externa controlada por software.

Las fuentes de reloj que incluye este sistema son, un oscilador de baja frecuencia (LFXTCLK) que se puede usar con cristales de baja frecuencia de reloj de 32768 Hz, estándar, resonadores o fuentes de reloj externas en el rango de 32 kHz o por debajo. Un oscilador de alta frecuencia (HFXTCLK) que se puede usar con cristales estándar o resonadores en el rango entre 1 MHz y 48 MHz. Un oscilador controlado digitalmente (DCOCLK) de frecuencias programable, con 3 MHz por defecto.Un oscilador de baja frecuencia y muy bajo consumo (VLOCLK) con una frecuencia típica de 9.4 KHz. Un oscilador interno de baja frecuencia y bajo consumo (REFOCLK) con frecuencias seleccionables de 32.768 kHz o 128 kHz. Un oscilador interno de bajo consumo (MODCLK),de 25 MHz frecuencia típica, y oscilador interno (SYSOSC) con 5 MHz, frecuencia típica.

Además, se encuentran disponibles cinco señales de reloj que se pueden obtener del modulo del reloj. El primero es el ACLK (*Auxiliary Clock*) cuya frecuencia máxima de operación es 128 kHz, y que se puede dividir para generar la frecuencia deseada. El segunddo es el MCLK (*Master clock*) usado por la CPU y directamente por algunos módulos de los periféricos, y puede dividirse por 1, 2, 4, 8, 16, 32, 64, o 128. El tercero es el HSMCLK (*Subsystem master clock*) se puede seleccionar por software,por módulos periféricos individuales. El cuarto es el SMCLK(*Low speed subsystem master clock*) que utiliza el HSMCLK como fuente de reloj. Su frecuencia está limitada a la mitad del máximo de frecuencia del HSMCLK.Y el quinto, el BCLK(*Low speed backup domain clock*), su frecuencia está restringida a un máximo de 32768 kHz.

En la implementación del PLL, se utilizarán el MCLK a una frecuencia de 48 MHz, que se obtiene del DCOCLK.

4.2.4. Temporizadores

El MSP432P401R cuenta con cuatro temporizadores Timer-A, y dos temporizadores independientes Timer32. Además de un *system tick timer* propio de los ARM Cortex-M.

Timer-A

Timer-A es un temporizador/contador de 16 bits con siete registros de captura/comparación. Admite múltiples capturas/comparaciones, salidas PWM, e intervalos de tiempo. Además, puede usarse para interrupciones, que pueden ser generadas o por un contador en condiciones de overflow, o por los registros.

Este módulo incluye:

- Contadores/temporizadores de 16 bits asíncronos con cuatro modos de operación : *Stop, Up, Continuos, Up/down*. Se distinguen por el valor de partida del contador, un valor máximo predeterminado o uno elegido por el usuario.
- Recurso de reloj configurable y seleccionable.
- Siete registros configurables.
- Salidas configurables con capacidad Pulse Width Modulation (PWM).

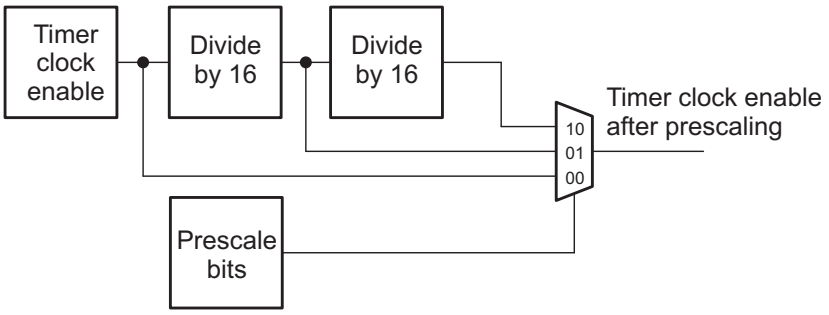


Figura 4.6: Reloj Timer32 [1]

- Captura asíncrona de entrada/salida.

El registro del Timer-A, TAxR, incrementa o decrementa el valor del mismo por medio de un flanco de reloj según su configuración. Se puede leer y escribir por software y puede generar una interrupción cuando se desborda (*overflow*).

Con respecto a sus fuentes de reloj pueden ser internos coomo el ACLK, o SMCLK, o externos desde el registro TAxCLK o INCLK. La fuente de reloj se puede dividir por 2, 3, 4, 5, 6, 7,o 8 usando el registro TAIDE.

Timer32

El Timer32 es un periférico desarrollado por Arm, cuyos registros se presentan en la Figura 4.7.

Su estructura consiste en dos contadores decrecientes programables, de 32 o 16 bits, que son capaces de generar interrupciones cuando llegan a cero. Estos contadores independientes tienen tres diferentes modos de operación cada uno,y su fuente de reloj, MCLK, se puede dividir por 1, 16 o 256 (4.6) . Las interrupciones son independientes en cada contador y también una combinada con ambos contadores.

Tiene tres modos de operación: *Free-runnig*, *Periodic timer*, y *One-shot timer*.

La primera una vez que llega al cero, continua la cuenta decremental desde el valor máximo. La segunda genera una interrupción a un intervalo constante y parte del valor asignado por el usuario como periodo una vez que llega a cero. La última, genera una interrupción sólo una vez.

Cuando se produce un *reset*, los contadores se deshabilitan, la interrupción se interrumpe y el registro de carga se pone a cero, el modo de operación pasa a ser *Free-running* y los valores de preescala se quedan en 1.

Con respecto a las interrupciones, estas se generan cuando el contador de 32 bits ha alcanzado el cero, y se desactivan cuando se escribe en el registro T32INTCLRx. El registro mantiene el valor hasta que la interrupción se desactiva. Se pueden enmascarar escribiendo un 0 en el bit de interrupción *Enable* en el registro T32CONTROLx. Para la implementación en el microcontrolador se ha elegido este módulo y su interrupción periódica cada periodo de muestreo (T_s), para controlar la conversión del ADC.

Offset	Acronym	Register Name	Type	Reset	Section
00h	T32LOAD1	Timer 1 Load Register	RW	0h	Section 18.5.1
04h	T32VALUE1	Timer 1 Current Value Register	R	FFFFFFFFh	Section 18.5.2
08h	T32CONTROL1	Timer 1 Timer Control Register	RW	20h	Section 18.5.3
0Ch	T32INTCLR1	Timer 1 Interrupt Clear Register	W	-	Section 18.5.4
10h	T32RIS1	Timer 1 Raw Interrupt Status Register	R	0h	Section 18.5.5
14h	T32MIS1	Timer 1 Interrupt Status Register	R	0h	Section 18.5.6
18h	T32BGLOAD1	Timer 1 Background Load Register	RW	0h	Section 18.5.7
20h	T32LOAD2	Timer 2 Load Register	RW	0h	Section 18.5.8
24h	T32VALUE2	Timer 2 Current Value Register	R	FFFFFFFFh	Section 18.5.9
28h	T32CONTROL2	Timer 2 Timer Control Register	RW	20h	Section 18.5.10
2Ch	T32INTCLR2	Timer 2 Interrupt Clear Register	W	X	Section 18.5.11
30h	T32RIS2	Timer 2 Raw Interrupt Status Register	R	0h	Section 18.5.12
34h	T32MIS2	Timer 2 Interrupt Status Register	R	0h	Section 18.5.13
38h	T32BGLOAD2	Timer 2 Background Load Register	RW	0h	Section 18.5.14

Figura 4.7: Registros Timer32 [1]

4.2.5. Conversores ADC

El módulo ADC tiene una precisión de 14 bits, 32 *buffers* independientes de control y conversión, lo que permite que cada *buffer* tenga además 32 muestras ADC independientes, que se convierten y guardar sin intervención de la CPU. La velocidad máxima de conversión es de 1 Msps, con una resolución de 14 bits, módules de muestreo y retención con periodos programables controlado por software o por temporizadores y las conversiones pueden ser iniciadas también por los mismos.

La tensión de referencia puede ser interna o externa. Tiene un sensor de temperatura interno y cuenta con 32 canales individuales externos de tipo diferencial o monopolar (*single-ended*).

Según sea la elección la fórmula de conversión cambia al igual que su tensión de entrada.

Para el monopolar :

$$N_{ADC} = 16384 \cdot \frac{V_{in_+} - V_{R_-}}{V_{R_+} - V_{R_-}}, 1LSB = \frac{V_{R_+} - V_{R_-}}{16384} \tag{4.1}$$

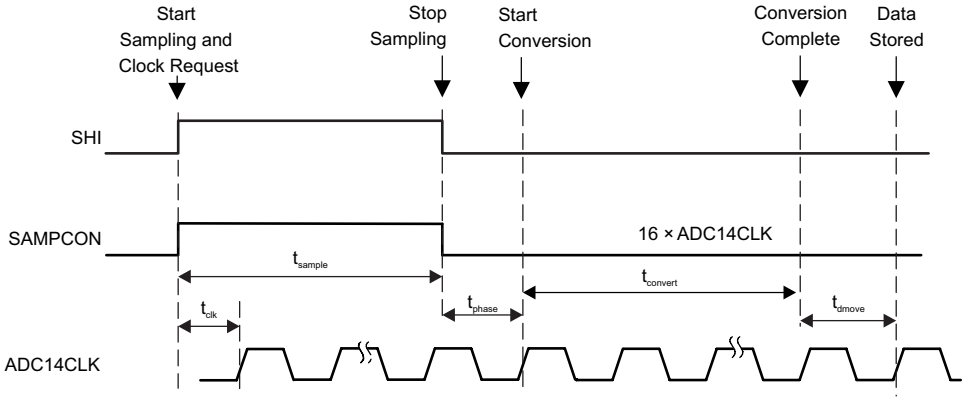
Para el diferencial:

$$N_{ADC} = \left(8192 \cdot \frac{V_{in_+} - V_{in_-}}{V_{R_+} - V_{R_-}}\right) + 8192, 1LSB = \frac{V_{R_+} - V_{R_-}}{8192} \tag{4.2}$$

El reloj del módulo se usa para la conversión y para el periodo de muestreo cuando se ha seleccionado el modo de muestreo por pulso. Para poder elegir el reloj se usa el bit ADC14SSELx, además si es de interés se puede dividir por 1, 4, 32, o 64. Este reloj puede tener como fuente los relojes MODCLK, SYSCLK, ACLK, MCLK,SMCLK,y HSMCLK. Se debe tener especial cuidado en que al escoger el reloj se de tiempo suficiente para que se pueda realizar todo el proceso de conversión, sino el resultado no será válido.

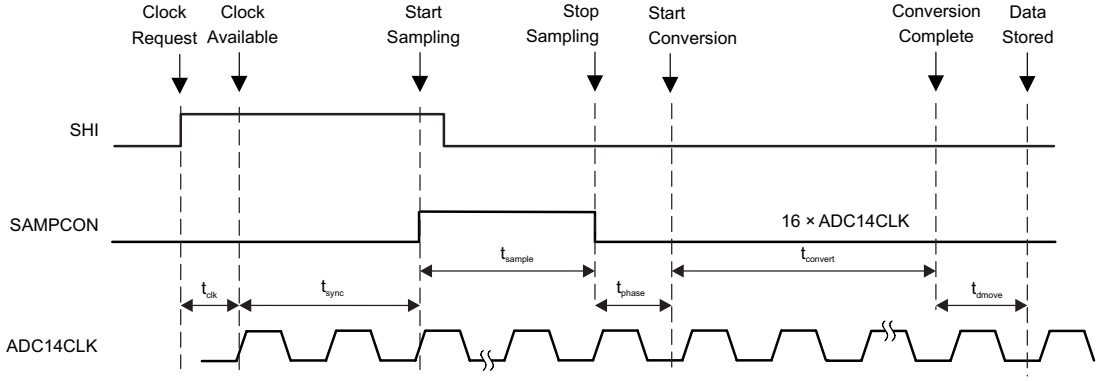
Para evitar corrientes parásitas en la entrada del puerto analogico el fabricante recomienda deshabilitar la parte digital del puerto mediante el bit PySELx, y de esa manera eleminar esta corriente parásita y reducir el consumo máximo de corriente.

PONER FOTO DEL CÓDIGO



Note: If internal ADC reference buffers are used, the SHI signal is gated while ADC14RDYIFG = 0.

Figura 4.8: *Extended Sample Mode* [1]



Note: If internal ADC reference buffers are used, the SHI signal is gated while ADC14RDYIFG = 0.

Figura 4.9: *Pulse Sample Mode* [1]

Mode	t_{clk}	t_{sync}	t_{sample}	t_{phase}	$t_{convert}$	t_{dmove}	Total
Pulse Sample Mode with ADC14MSC = 0	Up to 3 or 5 cycles(2)	2 cycles	Minimum 4 cycles	1 cycle	16 cycles for 14-bit	1 cycle	27 or 29 cycles
Pulse Sample Mode with ADC14MSC = 1	N/A(3)	N/A(3)	Minimum 4 cycles	1 cycle	16 cycles for 14-bit	1 cycle	22 cycles
Extended Sample Mode	N/A	N/A	See the device datasheet for minimum sample period	1 cycle	16 cycles for 14-bit	1 cycle	18 + sampling cycles

(1) The time for reference settling is not included

(2) Up to 3 cycles when SHI is generated based on the same clock source of the Precision ADC or 5 cycles when a different clock source is used to generate SHI signal

(3) Successive conversions

Figura 4.10: Tiempos de conversión y muestreo [1]

La conversión analógica a digital se inicia cuando hay un flanco de subida en la señal de entrada

de muestreo, SHI. Esta señal se seleciona con los bits SHSx, que incluye el bit ADC14SC, y otros siete recursos. Cuando la señal SHI se activa, se solicita el reloj del ADC y después de un tiempo(t_{clk}) de máximo 3 ciclos, la señal de reloj está disponible. Dependiendo de la resolución que requiera la conversión puede durar entre 9, 11, 14, y 16 ciclos de reloj para 8, 110, 12,y 14 bits, respectivamente. La polaridad de esta señal se puede invertir con el bit ADC14ISSH, para nuestro caso no es de interés por lo que permanerá en su valor por defecto. Para controlar el comienzo de conversión y el periodo de muestreo se utiliza la señal SAMPCON, cuando está está en alto el muestreo está activo, mientras que cuando cambia a estado bajo comienza la conversión. Una vez ésta finalice , el dato convertido se guarda en el registro ADC14MEMx, este proceso cuesta un ciclo de reloj (t_{dmove}).

Hay dos modos diferentes de muestrear, el modo de muestra extendida (*Extended Sample Mode*, Figura 4.8) y el modo de muestra por pulso (*Pulse Sample Mode*, Figura 4.9). Su principal diferencia es que en el modo extendido la señal SHI se usa directamente para controlar la señal SAMPCON, mientras que en el modo del pulso se usa para controlar el temporizador de muestreo. El tiempo total de conversión y muestreo depende del modo de muestreo que se elija, en la Figura 4.10 se puede ver cuanto valen estos tiempos.

Para la configuración en el microntrolador se utiliza una resolución de 10 bits y se ha elegido el modo de pulso con tiempo de muestro(t_{sample}) de 4 ciclos, y se utiliza el mismo reloj para la señal SHI y el reloj del ADC, HSMCLK.

Cuenta con cuatro modos de conversión quue se seleccionan mediante los bits CONSEQx :

- Un canal (*single channel*), donde la conversiín se produce solo una vez
- Un canal con repetición (*repeat-single channel*), en la que conversión se repite cada cierto tiempo.
- Secuencia (*sequence*), cuando la conversión se produce en secuencia cuando hay varios canales seleccionados.
- Secuencia repetida (*repeat-sequence*), es igual que la anterior pero se repite cada cierto tiempo.

En los casos de repetición cuando se lee el valor del registro de memoria, éste borra el valor para dejar paso al siguiente.

Los resultados de la conversión siempre se guardan en formato binario sin signo. Por lo que para los de tipo diferencial tienen un offset de 8192 añadido. Sin embargo, el formato del bit del dato (ADC14DF) en el registro ADC14CTL1,permite al usuario leer los resultados en formato sin signo o con signo expresado en complemento a 2.

Para parar la conversión en el modo de un canal con repetición es necesario poner a cero el bit ADC14ENC, en los otros modos se hace una vez finalizada la conversión.

Hay 32 registros de memoria ADC14MEMx para guardar los resultados, cada uno está asociado con un registro de control ADC14MCTLx. El bit ADC14VRSEL define la tensión de referencia ,y los bits ADC14INCHx y ADC14DIF seleccionan los canales de entrada. El bit ADC14EOS define el final se la secuencua cuando se está en modo secuencia. Los bits CSTARTADDx bits definen el primer ADC14MCTLx que se usa para cualquier conversión.

PONER el caso del TFG

Este módulo ofrece también interrupciones, cuyos recursos son los siguientes. Los bits ADC14IFGx bits, que se configura cuando su registro de memoria correspondiente tiene el resultado de una conversión. La interrupción ADC14OV que ocurre cuando el valor de una conversión se escribe en su

registro de memoria antes de que el valor anterior guardado en ese registro se lea. La interrupción ADC14TOV que se produce cuando se requiere muestreo y conversión antes de que que se complete la conversión actual. Las interrupciones ADC14LOIFG, ADC14INIFG, y ADC14HIIFG para gestionar el registro ADC14MEMx.

Todas estas interrupciones se encuentran dentro del vector de interrupción ADC14IV, donde se priorizan, combinan y gestionan las interrupciones.

4.2.6. Puertos de Entrada/Salida

Los puertos de entrada/salida de este microcontrolador son programables de forma independiente e individual. Los registros PxIN son solo de lectura y los PxOUT se configuran como entrada o salida con el registro de dirección PxDIR. Cada puerto posee un registro de datos individual, así como resistencias de *pullup* o *pulldown* que se activan en el registro PxEN.

PxDIR	PxREN	PxOUT	I/O Configuration
0	0	x	Input
0	1	0	Input with pulldown resistor
0	1	1	Input with pullup resistor
1	x	x	Output

Figura 4.11: Configuración de E/S [1]

Cada puerto necesita de dos bits (PxSEL0, PxSEL1), para seleccionar su función, de propósito general, modulo primario, modulo secundario, y modulo secundario. Esta selección no configura su pin de dirección. Todas las interrupciones de cada puerto están priorizadas siendo el bit PxIFG.0 el de mayor prioridad, y cada una de ellas corresponden con un pin. El registro de interrupción de selección por flanco (PxIES) selecciona el flanco de interrupción para cada pin de E/S. Estas interrupciones se pueden habilitar mediante el registro PxIE. Estos puertos cuentan con modos de despertar (*Wake-up*) y bajo consumo. Estos modos son LPM3, LPM4,LPM3.5 y LPM4.5. Los dos primeros se corresponden con el primer modo y los otros con el modo de bajo consumo.

Toda la información de este capítulo así como sus figuras se han obtenido de [1].

Capítulo 5

Implementación y Experimentación

5.1. Introducción

En este capítulo se va implementar el SOGI-PLL.

5.2. Discretización del SOGI-PLL

5.2.1. Modelo de Simulink discretizado

Se discretiza el modelo del SOGI-PLL en Simulink y se discretiza el PI calculado en el capítulo 3. Los valores de K_d y K_v son iguales a los del capítulo 3, y los valores discretizados del PI con $F_s = 10$ kHz, son $K_p = 0{,}35$ y $K_i = 0{,}3387$

En las figuras 5.1 y 5.2 se comprueba su correcto funcionamiento. Además, se comprueba que el muestreo se hace a la frecuencia F_s (Figura 5.3).

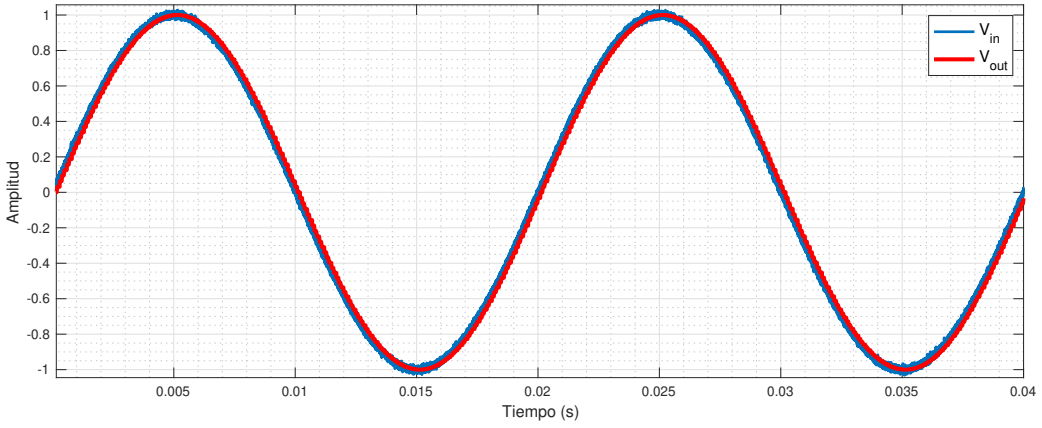


Figura 5.1: Tensión de entrada y salida

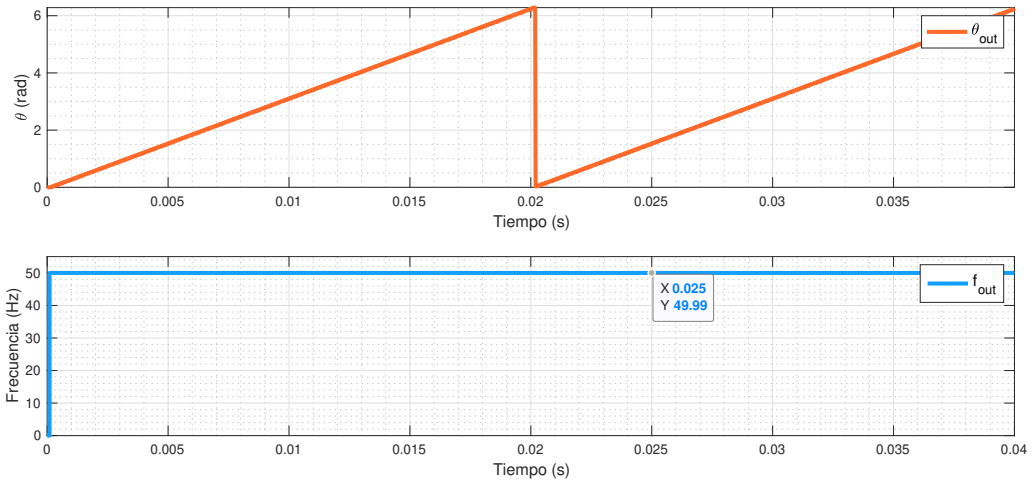


Figura 5.2: Ángulo de salida y frecuencia

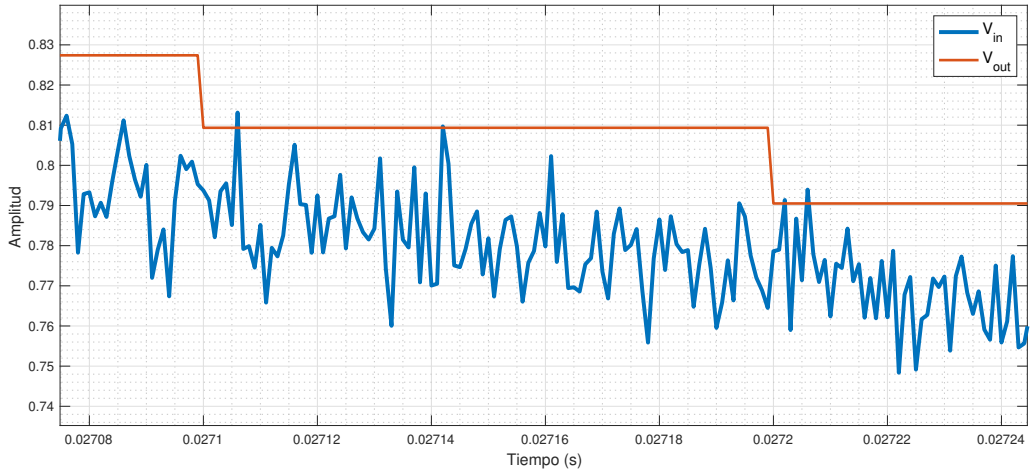


Figura 5.3: Tiempo de muestreo

5.2.2. Ecuaciones en diferencias

Siguiendo los paso de [16], para poder implementar el algoritmo primero es necesario obtener las ecuaciones en diferencias. Para ello, se transforman las funciones de transferencia del espacio en continuo al espacio discreto mediante la aproximación de integración trapezoidal (*Tustin*), con un tiempo de muestreo $T_s = 100$ µs.

Partiendo de (3.9) y (3.10), del capítulo 3, se obtienen las funciones correspondientes en el espacio discreto z,

$$G_\alpha(z) = \frac{b_0 + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{5.1}$$

$$G_\beta(z) = \frac{b_0 + 2b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{5.2}$$

Los parámetros de las funciones se calculan con la función de Matlab c2d, y así obtener las ecuaciones en diferencias de todos los bloques. Los parámetros de v_α y v_β son distintos.

El bloque del SOGI

$$v_\alpha(k) = b_0 V_{in}(k) + b_2 V_{in}(k-2) - (a_1 v_\alpha(k-1) + a_2 v_\alpha(k-2)) \tag{5.3}$$
$$v_\alpha(k-2) = v_\alpha(k-1) \tag{5.4}$$
$$v_\alpha(k-1) = v_\alpha(k) \tag{5.5}$$
$$v_\beta(k) = b_0 V_{in}(k) + b_1 V_{in}(k-1) + b_2 V_{in}(k-2) - (a_1 v_\beta(k-1) + a_2 v_\beta(k-2)) \tag{5.6}$$
$$v_\beta(k-2) = v_\beta(k-1) \tag{5.7}$$
$$v_\beta(k-1) = v_\beta(k) \tag{5.8}$$

La transformada de Park

$$v_d(k) = v_\alpha(k) sin_t(k-1) + v_\beta(k) cos_t(k-1) \tag{5.9}$$
$$v_q(k) = -v_\alpha(k) cos_t(k-1) + v_\beta(k) * sin_t(k-2) \tag{5.10}$$

El filtro paso bajo

$$v_2(k) = K_p v_q(k) - K_i v_q(k-1) + v2(k-1) \tag{5.11}$$
$$v_q(k-1) = v_q(k) \tag{5.12}$$
$$v_2(k-1) = v_2(k) \tag{5.13}$$

El bloque VCO

$$\omega_o = \omega_0 + K_v v_2(k) \tag{5.14}$$
$$\theta_0(k) = \theta(k-1) + \omega_o * T_s \tag{5.15}$$

Para comprobar su correcto funcionamiento se implementa en Matlab estas ecuaciones y se definen tres casos y se analiza la respuesta del SOGI-PLL.

- Caso 1: respuesta a un salto de fase (Figura 5.4)
- Caso 2: respuesta a una variación pequeña de frecuencia (Figura 5.5)
- Caso 3: respuesta a una variación de la amplitud de la señal de entrada (Figura 5.6)

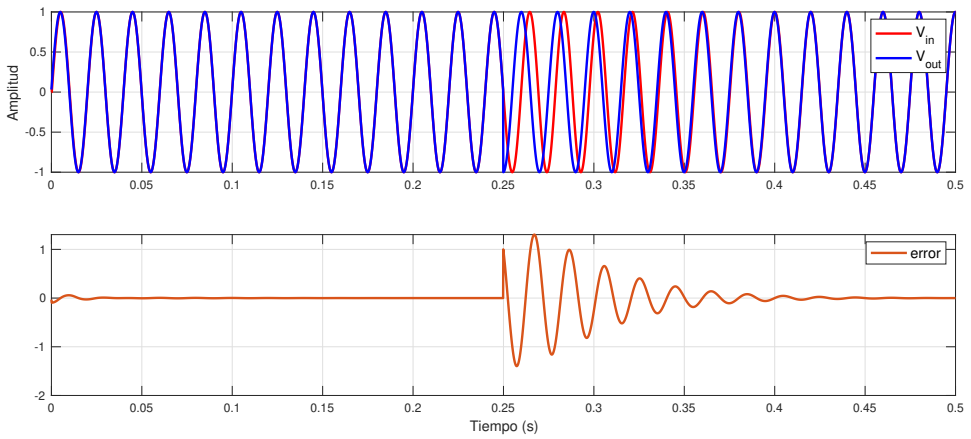


Figura 5.4: Respuesta a salto de fase

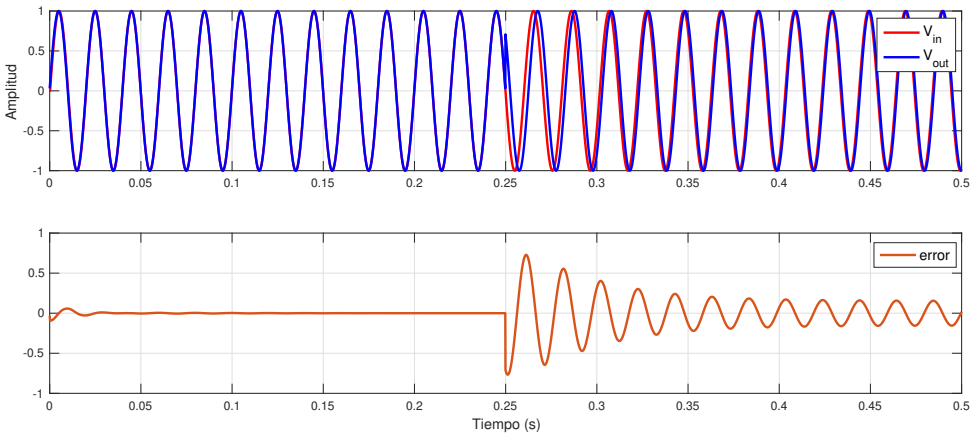


Figura 5.5: Respuesta a variación de frecuencia

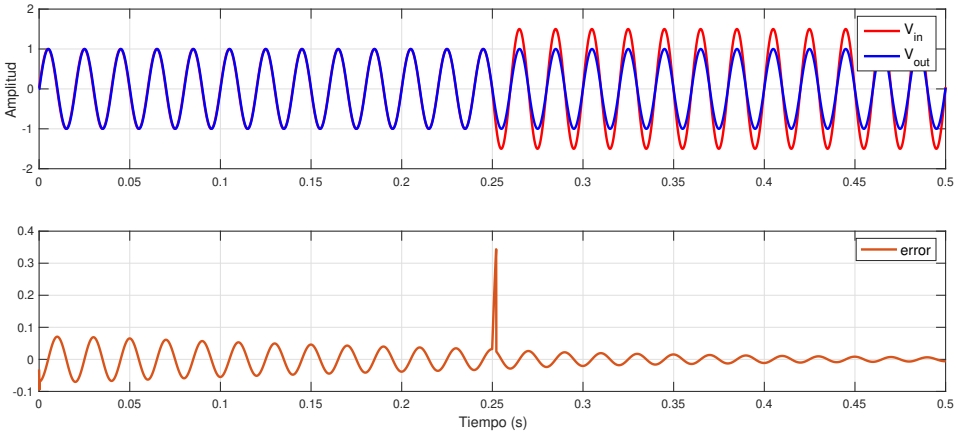


Figura 5.6: Respuesta a variación de amplitud

5.3. Implementación en C

Se va a implementar el SOGI-PLL con los parámetros antes establecidos. Se ha compilado el código para comprobar que no tiene errores de sintaxis. En el anexo se encuentra el código que se ha implementado.

5.4. Experimentación

La parte experimental requiere de un generador de señales, un osciloscopio para poder observar la entrada y salida del SOGI-PLL y el microncontrolador, su montaje se puede ver en la Figura 5.7.

La parte experimental no se pudo ejecutar antes de depositar la memoria del TFG, debido a problemas con la placa de desarrollo, el entorno de desarrollo no reconocía el dispositivo proporcionado por los directores. Se contó con la ayuda del profesor del Departamento de Ingeniería Electrónica y Comunicaciones, Isidro Urriza Parroqué, ya que había trabajado previamente con la placa y con el entorno CCS. Sin embargo, aunque se probara con tres placas, no se pudo conseguir la implementación en mi equipo de trabajo. Se dirigió a los foros de *Texas Instruments* pero sin

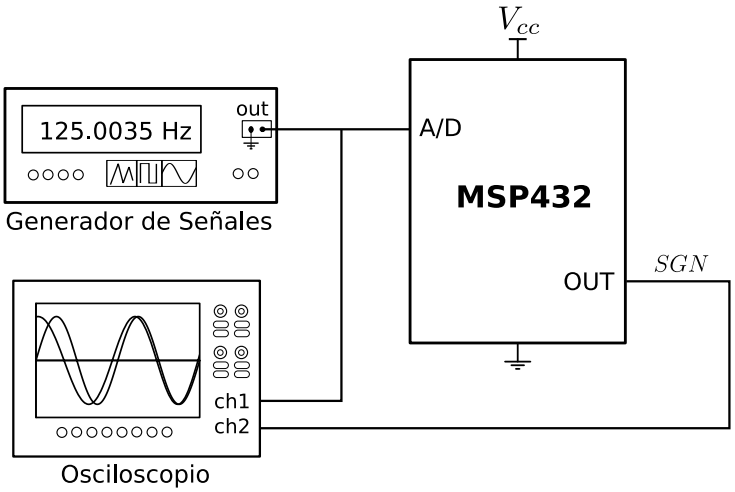


Figura 5.7: Montaje

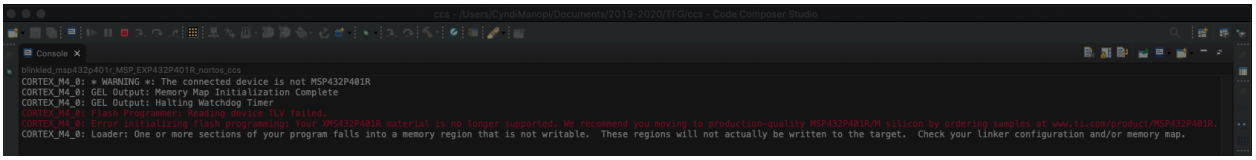


Figura 5.8: Mensajes de error CCS Console

éxito. En la Figura 5.8 se muestran los mensajes de error que aparecían, y por los que esta parte no se ha podido ejecutar.

Capítulo 6

Conclusiones y Líneas futuras

El PLL es un concepto ampliamente estudiado que tiene numerosas aplicaciones, como pueden ser, entre otras, el control de motores, generadores, convertidores de potencia, o sintonizadores digitales. El estudio de las topologías presentadas da una buena visión de su utilidad. En dicho estudio se ha analizado la influencia de los distintos parámetros que definen el comportamiento del PLL.

Por otro lado, la implementación de un algoritmo en un microcontrolador va más allá del diseño del propio algoritmo; es necesario el estudio de las interrupciones, temporizadores, y otras características del microcontrolador. Además, pueden aparecer problemas derivados del hardware utilizado.

La línea de trabajo futura primordial sería resolver los problemas con el entorno de desarrollo y comprobar el funcionamiento del algoritmo del SOGI-PLL planteado en este TFG.
Otras fuera del alcance de este trabajo serían desarrollar otras topologías [22], modificar los controles aumentando el orden o aplicando otros criterios de tiempo de respuesta o error en régimen permanente, por ejemplo. La implementación del algoritmo desarrollado en coma fija, utilizar otro microcontrolador u otros dispositivos como FPGAs [23], o DSPs [24].

Estudiar su aplicación en sistemas trifásicos, o desarrollar en concreto alguna de las aplicaciones que resulten de interés para el autor de las mencionadas en este TFG.

Capítulo 7

Anexos

Código Matlab

PLL

```
clear all;
close all;
%% PLL SIMULACION
% Rango de trabajo
%   50Hz
%   margen de enganche = f(Kv,Kpd)
%   margen de captura = f1(Kv,Kpd)
%%-------------------------------------------------------------------
% Variables
% parametros simulacion
t_final = 0.04; % tiempo de simulacion
max_step = 10e-6; % paso maximo de simulacion
% Senal de entrada
A=1;
f0 = 50;
ts=0.9e-3; % ms
fs=1/ts;
%%----------------------Bloques ------------------------------------
%----------------------Bloque PD-----------------------------------
Kpd=0.5; %V/2
%%----------------------Bloque VCO----------------------------------
% controla el enganche
Kv=0.1;
w0=2*pi*50; % frecuencia central
%%----------------------Bloque LF-----------------------------------
%% PI
SysL=tf([0 Kpd*Kv], [1 0]);
% sisotool(SysL)
% figure
% bode(sisPI)   % SIS exportado de sisotool
% grid on
%PM = 60 BW = 20
```

```
Kp = 11.547;
Ki = 346.4102*Kp;
%% PI + primer orden
sys1 = tf([0 Kpd*Kv], [1 0]);
wclp = 2*pi*10;
sys2 = tf([0 1], [(1/wclp) 1]); % primer orden
SysL = series (sys1,sys2);
% sisotool(SysL);
% figure
% bode(sisPI1)  % SIS exportado de sisotool
% grid on
%PM = 60 BW = 20
Kpp = 4.3765;
Kii = 410.0721*Kpp;
%% PI + segundo orden(butter)
sys1 = tf([0 Kpd*Kv], [1 0]);
fc = 10;
wnb = fc/(fs/2);
[b,a] = butter(2,wnb);
sys = tf(b,a,ts);
sys2 = d2c(sys);
[num,den] = tfdata(sys2,'v');
SysL = series (sys1,sys2);
% sisotool(SysL);
% figure
% bode(sisPI2)  % SIS exportado de sisotool
% grid on
%PM = 60 BW = 20
Kp3 = 0.35;
Ki3 = 401.9782*Kp3;
%% SOGI + PI
%KPD = 10;
KV=0.01;
f_sogi = 10;
w0_sogi = 2*pi*f_sogi;
t_sogi = 2/(Kpd*w0_sogi);
sys1=tf([0 KV], [1 0]);
sys_sogi=tf([0 1],[t_sogi 1]);       % linealizacion Golestan (ec. 15)
SysL=series(sys1,sys_sogi);
sisotool(SysL)
figure
bode(sisSOGI)  % SIS exportado de sisotool
grid on
%PM = 60 BW = 20
% Kp4 = 6.47498;
% Ki4 = 8.384*Kp4;
Kp4 = 0.35;
Ki4 = 323.7491*Kp4;

```

```
82  %%
83  sim('pll');
84  %----plot tensiones--------
85  fig1 = figure (1);
86  subplot(2,1,1)
87  plot(stime,Vin,'LineWidth',3);
88  hold on
89  plot(stime,V_out_basic,'LineWidth',2)
90  ylabel ('Amplitud','FontSize',12)
91  xlabel ('Tiempo (s)','FontSize',12)
92  legend ({'V_{in}','V_{pll-basico}'},'FontSize',12)
93  grid
94  grid minor
95  subplot (2,1,2)
96  plot(stime,Vin,'LineWidth',3);
97  hold on
98  plot(stime,V_out2,'LineWidth',2)
99  ylabel ('Amplitud','FontSize',12)
100 xlabel ('Tiempo (s)','FontSize',12)
101 legend ({'V_{in}','V_{pll-tipo2}'},'FontSize',12)
102 grid
103 grid minor
104
105 %%
106 fig2 = figure (2);
107 subplot(2,1,1)
108 plot(stime,Vin,'LineWidth',2);
109 hold on
110 plot(stime,V_out3,'LineWidth',2)
111 ylabel ('Amplitud','FontSize',12)
112 xlabel ('Tiempo (s)','FontSize',12)
113 legend ({'V{in}','V_{pll-tipo3}'},'FontSize',12)
114 grid
115 grid minor
116 subplot(2,1,2)
117 plot(stime,Vin,'LineWidth',2);
118 hold on
119 plot(stime,V_outs,'LineWidth',2)
120 ylabel ('Amplitud','FontSize',12)
121 xlabel ('Tiempo (s)','FontSize',12)
122 legend ({'V{in}','V_{sogi-pll}'},'FontSize',12)
123 grid
124 grid minor
125
126 %%----plot frecuencia--------
127 fig3 = figure (3);
128 subplot(2,1,1)
129 hold on
130 plot(stime,f_out_basic,'LineWidth',2)
```

```
plot(stime,f_out2,'LineWidth',2)
ylabel ('Frecuencia (Hz)','FontSize',12)
xlabel ('Tiempo (s)','FontSize',12)
legend ({'F_{pll-basico}','F_{pll-tipo2}'},'FontSize',12)
grid
grid minor
hold off
subplot(2,1,2)
hold on
plot(stime,f_outs,'LineWidth',2)
plot(stime,f_out3,'LineWidth',2)
ylabel ('Frecuencia (Hz)','FontSize',12)
xlabel ('Tiempo (s)','FontSize',12)
legend ({'F_{pll-tipo3}','F_{sogi-pll}'},'FontSize',12)
grid
grid minor
hold off
%%----plot tita-------------
fig4 = figure (4);
subplot(2,1,1)
hold on
plot(stime,tita_out_basic,'LineWidth',2)
plot(stime,tita_out2,'--','LineWidth',3)
ylabel ('\theta (rad)','FontSize',12)
xlabel ('Tiempo (s)','FontSize',12)
legend ({'\theta_{pll-basico}','\theta_{pll-tipo2}'},'FontSize',12)
grid
grid minor
hold off
subplot(2,1,2)
hold on
plot(stime,tita_out3,'LineWidth',2)
plot(stime,tita_outs,'--','LineWidth',2)
ylabel ('\theta (rad)','FontSize',12)
xlabel ('Tiempo (s)','FontSize',12)
legend ({'\theta_{pll-tipo3}','\theta_{sogi-pll}'},'FontSize',12)
grid
grid minor
hold off
```

SOGI PLL

```
clear all;
close all;
%% SOGI-PLL discretizado SIMULACION
% Rango de trabajo
%   50Hz
```

```
%   margen de enganche = f(Kv,Kpd)
%   margen de captura = f1(Kv,Kpd)
%%-------------------------------------------------------------------
% Variables
% parametros simulacion
t_fin = 0.5; % tiempo de simulacion
max_step = 10e-6; % paso maximo de simulacion
%Senal de entrada
A = 1;          % Amplitud
f0 = 50;    % frec (Hz) senal de entrada

Fs = 10000;   % Frec (Hz) muestreo
Ts = 1/Fs;    % Tiempo muestreo (s)
t = 0:Ts:t_fin;
%%----------------------Bloque VCO-----------------------------------
% controla el enganche
Kv=0.1; %Kv=0.08 para caso 2
w0=2*pi*50; % frecuencia central
%%----------------------Bloque LF------------------------------------
%Kp = 0.35;
%Ki = 323.7491*Kp;
%sis = tf([Kp Ki],[1 0]);
%sisz = c2d(sis,Ts);
%Kpz = 0.35;
%Kiz = 0.3387;
Kpz = 0.35;
Kiz = 0.3387;
%% Discretizacion y ec. en diferencia
% calculo coef. del SOGI
k = 0.5; %Kpd
wn = 2*pi*f0;
in_va = tf([k*wn 0],[1 k*wn wn^2]);
in_vb = tf([k*wn^2 ],[1 k*wn wn^2]);
in_vaz = c2d(in_va,Ts,'tustin');
in_vbz = c2d(in_vb,Ts,'tustin');
[num,den,Ts] = tfdata(in_vaz,'v');
b0=num(3);
b2=num(1);
a0=den(1);
a1=den(2);
a2=den(3);
[num,den,Ts] = tfdata(in_vbz,'v');
bb0=num(1);
bb1=num(2);
bb2=num(3);
aa0=den(1);
aa1=den(2);
aa2=den(3);
%% Simulacion Simulink
```

```
Kpd = 1;
sim('SOGIpllZ');
fig1 = figure (1);
plot(stime,Vin,'LineWidth',3);
hold on
plot(stime,Vout,'LineWidth',2)
ylabel ('Amplitud','FontSize',12)
xlabel ('Tiempo (s)','FontSize',12)
legend ({'V_{in}','V_{out}'},'FontSize',12)
grid
grid minor
figure(2)
subplot (2,1,2)
plot(stime,fout,'LineWidth',3);
ylabel ('Frecuencia (Hz)','FontSize',12)
xlabel ('Tiempo (s)','FontSize',12)
legend ('f_{out}','FontSize',12)
grid
grid minor
subplot (2,1,1)
plot(stime,tita,'LineWidth',3);
ylabel ('\theta (rad)','FontSize',12)
xlabel ('Tiempo (s)','FontSize',12)
legend ('\theta_{out}','FontSize',12)
grid
grid minor
%% Simulacion Matlab
% arrays
err = [0,0,0,0];
v2 = [0,0,0];
cos_t = [1,1,1];
sin_t = [0,0,0];
theta = [0,0,0];
wo=0;
% senales
v_a = [0,0,0,0];
v_b = [0,0,0,0];
vd = [0,0,0,0];              % salida Vd de la transf. de PARK
vq = [0,0,0,0];              % salida Vq  de la transf. de PARK
% plot
Var=[0,0,0,0];
Theta=[0,0,0,0];
V_a=[0,0,0,0];
V_b=[0,0,0,0];
V_d=[0,0,0,0];
V_q=[0,0,0,0];
V_2=[0,0,0,0];

%%
```

```
% senal de entrada
L=length(t);
%%CASO 1: Salto de fase a mitad
for n = 1:floor(L)
u(n) = A*sin(2*pi*f0*Ts*n);
end
for n = floor(L/2):L
u(n) = A*sin(2*pi*f0*Ts*n+pi/2);
end
%%CASO 2:variacion en la frecuencia
for n = 1:floor(L)
u(n) = A*sin(2*pi*f0*Ts*n);
end
for n = floor(L/2):L
u(n) = A*sin(2*pi*(f0-0.5)*Ts*n);
end
%%CASO 3: variacion en la amplitud
for n = 1:floor(L)
u(n) = A*sin(2*pi*f0*Ts*n);
end
for n = floor(L/2):L
u(n) = 1.5*A*sin(2*pi*f0*Ts*n);
end
%%
% proceso de PLL
for n = 3:t_fin/Ts
    %SOGI
    v_a(1) = b0*u(n)+b2*u(n-2)-(a1*v_a(2)+a2*v_a(3));
    v_a(3) = v_a(2);
    v_a(2) = v_a(1);

    v_b(1) = bb0*u(n)+bb1*u(n-1)+bb2*u(n-2)-(aa1*v_b(2)+aa2*v_b(3));
    v_b(3) = v_b(2);
    v_b(2) = v_b(1);
    %TF PARK
    vd(1) = v_a(1)*sin_t(2)+v_b(1)*cos_t(2);
    vq(1) = -v_a(1)*cos_t(2)+v_b(1)*sin_t(2);

    % Filtro PI
    v2(1) = Kpz*vq(1)-Kiz*vq(2)+v2(2);
    vq(2) = vq(1);
    v2(2) = v2(1);

    wo = w0+Kv*v2(1);
    theta(1) = theta(2)+wo*Ts;
    if(theta(1)>=2*pi)
        theta(1) = theta(1)-2*pi;
    end
    theta(2) = theta(1);
```

```
153         sin_t(1) = sin(theta(1));
154         cos_t(1) = cos(theta(1));
155         sin_t(2) = sin_t(1);
156         cos_t(2) = cos_t(1);
157         Theta(n+1) = theta(1);
158         V_a(n+1) = v_a(1);
159         V_b(n+1) = v_b(1);
160         V_d(n+1) = vd(1);
161         V_q(n+1) = vq(1);
162         V_2(n+1) = v2(1);
163         Var(n+1) = sin_t(1);
164     end
165     %% CASO 1
166     caso1 = figure(1);
167     error = Var-u;
168     subplot(2,1,1)
169     plot(t,Var,'r',t,u,'b')
170     ylabel('Amplitud','FontSize',12)
171     legend({'V_{in}','V_{out}'},'FontSize',12)
172     grid on
173     subplot(2,1,2)
174     plot(t,error,'r')
175     xlabel('Tiempo (s)','FontSize',12)
176     grid on
177     legend('error','FontSize',12)
178
179     %% CASO 2
180     caso2 = figure(2);
181     error = Var-u;
182     subplot(2,1,1)
183     plot(t,Var,'r',t,u,'b')
184     ylabel('Amplitud','FontSize',12)
185     legend({'V_{in}','V_{out}'},'FontSize',12)
186     grid on
187     subplot(2,1,2)
188     plot(t,error,'r')
189     xlabel('Tiempo (s)','FontSize',12)
190     grid on
191     legend('error','FontSize',12)
192     %% CASO 3
193     caso3 = figure(3);
194     error = [(Var(1:2522)-u(1:2522)),(Var(2523:5001)-1/1.5*u(2523:5001))];
195     subplot(2,1,1)
196     plot(t,u,'r',t,Var,'b')
197     ylabel('Amplitud','FontSize',12)
198     legend({'V_{in}','V_{out}'},'FontSize',12)
199     grid on
200     subplot(2,1,2)
201     plot(t,error,'r')
```

```
xlabel('Tiempo (s)','FontSize',12)
grid on
legend('error','FontSize',12)
```

Código C

```
#include "msp.h"
#include <stdint.h>
#include <stdbool.h>
#include  <math.h>

/*Declaracion de funciones*/

void configurar_CLK(void);
void configurar_Puertos(void);
void configurar_ADC(void);
void configurar_Timer32(void);

volatile int value_adc;
volatile int SGN;
void main(void)
{
    __disable_irq();
    WDT_A->CTL = WDT_A_CTL_PW + WDT_A_CTL_HOLD; // Stop watchdog timer
    configurar_CLK();
    configurar_Timer32();
    configurar_Puertos();
    configurar_ADC();


    NVIC_EnableIRQ (T32_INT1_IRQn);
    __enable_irq();

    while(1)
    {

    }
}

void configurar_CLK(void){
    // en fichero system_msp432p401r.c se puede configurar
       SYSTEM_CLOCK 4800000, la  m xima  frecuencia
    // Posiblemente no es necesario llamar a esta  f u n c i n  de
       c o n f i g u r a c i n
    CS->KEY = CS_KEY_VAL; // unlock CS registers
    CS->CTL0 = 0; // reset DCO settings
    CS->CTL0 = CS_CTL0_DCORSEL_5; // select DCO 5 (48MHz)
```

```
40      CS->CTL1 = CS_CTL1_SELS__DCOCLK | CS_CTL1_SELM__DCOCLK; // ACLK =
            REFOCLK, SMCLK = MCLK = DCOCLK
41      CS->KEY = 0; // lock CS registers
42  }
43
44  void configurar_Timer32(void){
45      TIMER32_1->LOAD = 4800-1;        /* 4800, fclk = 48MHZ  fs = 10 KHz*/
46      TIMER32_1->CONTROL = TIMER32_CONTROL_ENABLE | TIMER32_CONTROL_MODE
            | TIMER32_CONTROL_IE;    /* Enable, periodic mode, interrupt
            enable*/
47  }
48
49
50  void configurar_Puertos(void){
51      // Salidas digitales P6.0 y P6.1
52      P6->SEL1 &= ~(BIT1 + BIT0);  // P6.1 y P6.0 I/O
53      P6->SEL0 &= ~(BIT1 + BIT0);
54      P6->OUT |= BIT1 + BIT0;
55      P6->DIR |= BIT1 + BIT0;
56  }
57
58
59  void configurar_ADC(void){
60      P4->SEL1 |= BIT7;  // Enable A/D channel A6 de ADC - pin P4.7
61      P4->SEL0 |= BIT7;
62
63      // ADC14 Configuration
64      ADC14->CTL0 &= ~ADC14_CTL0_ENC;
65       /*CONFIGURE ADC14*/
66       ADC14->CTL0 |= ADC14_CTL0_SSEL__MCLK | ADC14_CTL0_SHT0_2 |
             ADC14_CTL0_ON;        /*ADC14 ON | S&H=16 | SAMPLING TIME*/
67       ADC14->CTL1 = ADC14_CTL1_RES__14BIT ;
68       ADC14->MCTL[0] |= ADC14_MCTLN_INCH_6 | ADC14_MCTLN_VRSEL_0;  //
             ref+=AVcc, channel = A6
69  }
70
71
72  void leerADC(void){
73      ADC14->CTL0 |= ADC14_CTL0_ENC | ADC14_CTL0_SC;                  /*
            ENABLE CONVERTION | SAMPLING CONVERSION*/
74      while(ADC14->CTL0 & ADC14_CTL0_BUSY);
75      value_adc = ADC14->MEM[0];
76
77  }
78  void sogiPll(int Da){
79     /* Variables */
80   float Ts = 1.0e-04;
81   float a1 = -1.9834;
82   float a2 =  0.9844;
```

```
83  float b0 = -0.0078;
84  float b2 = -b0;
85  float aa1 = -1.9834;
86  float aa2 =  0.9844;
87  float bb0 = 1.2238e-4;
88  float bb1 = 2.4476e-4;
89  float bb2 = 1.2238e-04;
90  float Kp = 336.47498 ;
91  float Ki = 336.47271;
92  float Kv = 0.8;
93  float w0 = 314.1593;     /*Hz*/
94  float wo =0.0;
95  /*--------- Se ales ---------- */
96  float Vin0,Vin1,Vin2 = 0.0;
97  float v_a0, v_a1, v_a2 = 0.0;
98  float v_b0, v_b1, v_b2 = 0.0;
99  float v_q0, v_q1 = 0.0;
100 float v2_0, v2_1 = 0.0;
101 float theta0, theta1 = 0.0;
102 float sen0, sen1 = 0.0;
103 float cos0,cos1 = 1.0;
104 /*--------- Control ----------*/
105  Vin0=(float)Da;
106 /*------ SOGI---------*/
107  v_a0 = b0*Vin0+b2*Vin2-(a1*v_a1+a2*v_a2);
108  v_a2 = v_a1;
109  v_a1 = v_a0;
110  v_b1 = bb0*Vin0+bb1*Vin1+bb2*Vin2-(aa1*v_b1+aa2*v_b2);
111  v_b2 = v_b1;
112  v_b1 = v_b0;
113 /*------ Tf PARK---------*/
114  v_q0 = -v_a0*cos1+v_b0*sen1;
115 /*------ Loop Filter---------*/
116  v2_0 = Kp*v_q0-Ki*v_q1+v2_1;
117  v_q1 = v_q0;
118  v2_1 = v2_1;
119  /*------ VCO---------*/
120  wo = w0 + Kv*v2_0;
121  theta0 = theta1 + wo*Ts;
122  if(theta0>=6.2832)
123     theta0 = theta0 - 6.2832;
124
125  theta1 = theta0;
126  sen0 = sin(theta0);
127  cos0 = cos(theta0);
128  sen1 = sen0;
129  cos1 = cos0;
130
131  SGN = (int)sen0;
```

```
132 }
133 void T32_INT1_IRQHandler(void){
134     TIMER32_1->INTCLR = 0;  // Clear interrupt flag
135     leerADC();
136     sogiPll(value_adc);
137     P6->OUT^=SGN;  // Toggle P6.0, comprobar que lo hace cad 100 us
138 }
```

Bibliografía

[1] T. Instruments, *MSP432P4xx SimpleLink™ Microcontrollers Technical Reference Manual*, 2019.

[2] F. Gardner, *Phaselock Techniques*. Wiley, 1979.

[3] V. Kroupa, *Phase Lock Loops and Frequency Synthesis*. Wiley, 2003.

[4] A. Godave, P. Choudhari, and A. Jadhav, "Comparison and simulation of analog and digital phase locked loop," in *2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pp. 1–4, 2018.

[5] D. Abramovitch, "Phase-locked loops: a control centric tutorial," in *Proceedings of the 2002 American Control Conference (IEEE Cat. No.CH37301)*, vol. 1, pp. 1–15 vol.1, 2002.

[6] F. Gardner, *Phaselock Techniques*. Wiley, 2005.

[7] D. Banerjee, *Pll Performance, Simulation and Design*. Texas Instruments, 2017.

[8] Z. Brezovic and V. Kudjak, "Pll synthesizer with very low discrete spurious," in *2008 18th International Conference Radioelektronika*, pp. 1–4, 2008.

[9] G. Bischof, B. Scholnick, and E. Salman, "Fully integrated pll based clock generator for implantable biomedical applications," in *2011 IEEE Long Island Systems, Applications and Technology Conference*, pp. 1–6, 2011.

[10] O. V. Nos, E. E. Abramushkina, and S. A. Kharitonov, "Control design of fast response pll for facts applications," in *2019 International Ural Conference on Electrical Power Engineering (UralCon)*, pp. 301–305, 2019.

[11] A. L. Makarevich, A. N. Kinash, M. S. Tokar, and V. A. Chubarov, "Performance analysis of pll components in digital synchronization systems for high-speed applications," in *2018Systems of Signal Synchronization, Generating and Processing in Telecommunications (SYNCHROINFO)*, pp. 1–3, 2018.

[12] R. Pinado, "Phase lock-loop(pll): Fundamento y aplicaciones." 2020.

[13] R. E. Best, "Phase locked loops: design, simulation, and applications.," 2007.

[14] X. Guo, W. Wu, and H.-R. Gu, "Phase locked loop and synchronization methods for grid-interfaced converters: a review," *Przeglad Elektrotechniczny*, pp. 182–187, 2011.

[15] R. M. Santos Filho, P. F. Seixas, P. C. Cortizo, L. A. B. Torres, and A. F. Souza, "Comparison of three single-phase pll algorithms for ups applications," *IEEE Transactions on Industrial Electronics*, vol. 55, no. 8, pp. 2923–2932, 2008.

[16] M. Ciobotaru, R. Teodorescu, and F. Blaabjerg, "A new single-phase pll structure based on second order generalized integrator," in *2006 37th IEEE Power Electronics Specialists Conference*, pp. 1–6, 2006.

[17] X. Song, B. Han, S. Zheng, and J. Fang, "High-precision sensorless drive for high-speed bldc motors based on the virtual third harmonic back-emf," *IEEE Transactions on Power Electronics*, vol. 33, no. 2, pp. 1528–1540, 2018.

[18] H. K. Yada and A. S. Kumar, "An so-sogi based control for a three-phase dvr under distorted grid conditions including dc offset," in *TENCON 2017 - 2017 IEEE Region 10 Conference*, pp. 3000–3005, 2017.

[19] S. Nayak, S. Gurunath, and N. Rajasekar, "Advanced single-phase inverse park pll with tuning of pi controller for improving stability of grid utility using soft computing technique," in *2016 Online International Conference on Green Engineering and Technologies (IC-GET)*, pp. 1–5, 2016.

[20] M. Z. Hasan, I. F. Shiam, T. T. Nova, and M. S. Reza, "A modified pll based on second order generalized integrator for single-phase voltage system," in *2019 International Conference on Electrical, Computer and Communication Engineering (ECCE)*, pp. 1–6, 2019.

[21] S. Golestan, M. Monfared, F. D. Freijedo, and J. M. Guerrero, "Dynamics assessment of advanced single-phase pll structures," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 6, pp. 2167–2177, 2013.

[22] R. J. Ferreira, R. E. Araújo, and J. A. Peças Lopes, "A comparative analysis and implementation of various pll techniques applied to single-phase grids," in *Proceedings of the 2011 3rd International Youth Conference on Energetics (IYCE)*, pp. 1–8, 2011.

[23] R. Rathod, P. Thaker, and N. Kolhare, "Design and simulation of dpll for fpga based carrier recovery loop: Psk receiver for digital satellite communication," in *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*, pp. 1962–1966, 2018.

[24] N. S. Bayindir, O. Kukrer, and M. Yakup, "Dsp-based pll-controlled 50-100 khz 20 kw high-frequency induction heating system for surface hardening and welding applications," *IEE Proceedings - Electric Power Applications*, vol. 150, no. 3, pp. 365–371, 2003.