Universidad de la República, Facultad de Ingeniería
Proyecto de Fin de Carrera

PLATAFORMA DSP PARA LA SIMULACIÓN Y CONTROL DE UN FILTRO ACTIVO DE CORRIENTES ARMÓNICAS

Gabriel Barbat Cassanello
Andrés Silveyra Bruno
Gabriel Valiente Hernández

Tutor:
Dr. Gonzalo Casaravilla Ponsetí

30 de septiembre de 2007

Agradecimientos

A Controles S.A., MSc. Cesar Briozzo, Ing. Fernando Chiaramello, CCC del Uruguay S.A., Gastón Rivoir, Diego Giacosa, Renzo Biardo, Nelson Ventura, Roberto Rodríguez y Sergio Beheregaray.

Tabla de contenidos

1. Resumen

Las cargas no lineales distorsionan la corriente consumida, causando caídas de tensión armónicas en la red. En ciertos casos, esto puede afectar el funcionamiento de distintos dispositivos.

El proyecto tuvo como objetivo principal el diseño e implementación de un circuito de control de un filtro activo de corrientes armónicas, utilizando un DSP. Se controló un VSI para suministrar corrientes de forma de atenuar selectivamente los armónicos que consume una carga trifásica no lineal conectada a la red. De esta manera se podría cumplir con determinadas normas sobre distorsión armónica con un mínimo de corriente generada.

Se distinguen tres etapas bien definidas:

- Simulaciones: utilizando Matlab-Simulink.
- Hardware-in-the-Loop: interacción entre el control implementado en el DSP y la planta simulada, la cual consiste en una red eléctrica, una carga trifásica no lineal (de tres hilos) y un inversor trifásico.
- Control de un VSI real: en esta etapa también se realizó una interfaz gráfica que facilitó la realización de las pruebas del sistema final.

Los resultados obtenidos fueron satisfactorios. Se alcanzaron las metas planteados originalmente, cumpliendo con los tiempos planificados.

2. Introducción

Actualmente se está desarrollando un proyecto PDT (Proyecto de Desarrollo Tecnológico) cuyo responsable es el MSc. Cesar Briozzo y el coordinador es el Dr. Gonzalo Casaravilla, al cual contribuyen dos proyectos de grado (incluyendo éste). Ese proyecto PDT tiene como uno de sus objetivos crear un prototipo de un filtro activo para ser aplicado en un dimmer de alumbrado público.

En particular, el proyecto que aquí se documenta tuvo como objetivo principal el diseño e implementación de un circuito de control de un filtro activo utilizando un DSP. Se controló un VSI (Voltage Source Inverter) para suministrar los armónicos que consume una carga no lineal conectada a la red y así poder cumplir con las normas internacionales.

3. Fundamentos teóricos

3.1. Armónicos

La calidad de la onda de tensión se refiere a las características con que se ofrece la tensión en las instalaciones del usuario, y se evalúa a partir de parámetros tales como la frecuencia, la amplitud, la forma y la simetría en la onda. Cualquier desviación de estos valores con respecto a unos límites establecidos por la normativa correspondiente es considerada como una perturbación.

Un filtro activo, es un dispositivo electrónico que aprovecha la energía almacenada ya sea en un condensador o en una bobina, para entregar y/o volver a almacenar la energía según una consigna que busca compensar la perturbación. Dicha consigna se conoce con el nombre de referencia y su generación se da por medio de la apertura y cierre de interruptores de potencia (normalmente semiconductores). En cuanto a la perturbación que se puede compensar, esta dependerá de la estrategia de control y de la forma en que se conecte el filtro a la red.

El problema de los armónicos se crea por la utilización cada vez mayor de dispositivos electrónicos en sectores residenciales, comerciales e industriales que ha traído consigo un aumento significativo de las perturbaciones que afectan la calidad de la onda. Cargas no lineales tan comunes como variadores de velocidad, lámparas fluorescentes compactas, ordenadores, etc., se convierten en fuentes generadoras de armónicos que al no ser compensadas adecuadamente, pueden vulnerar los niveles de susceptibilidad del entorno electromagnético que los rodea, y por consiguiente ocasionar un mal funcionamiento de equipos y protecciones, un aumento de pérdidas, etc. Para solucionar estos problemas, se han utilizado técnicas que van desde la utilización de filtros pasivos sintonizados hasta la utilización de filtros activos. Si bien los primeros tienen la ventaja de ser más económicos, su selectividad no les permite compensar todo lo no deseado, además, su naturaleza pasiva trae consigo la modificación de las frecuencias de resonancia y su posible excitación puede traer graves consecuencias. Los filtros activos se presentan como la solución dinámica que más se ajusta a las necesidades de compensación.

Las cargas no lineales distorsionan la corriente consumida, causando caídas de tensión armónicas y, por tanto, tensiones distorsionadas en los nudos. Los armónicos se pueden representar y cuantificar mediante un análisis de Fourier, este es el caso para una señal no sinusoidal, periódica y de energía finita. En ese sentido, se presenta la Distorsión Armónica Total (THD en inglés), como una medida de la desviación de la forma de onda no sinusoidal con respecto a la sinusoide pura de frecuencia fundamental. Por la calidad de la información que este factor involucra, el adoptado por muchas normativas para indicar los límites de las perturbaciones armónicas es el que se halla por la ec. 3.1.

$$\mathrm{THD}_I = 100\sqrt{\sum_{h\neq 1}\left(\frac{I_h}{I_1}\right)^2} \tag{3.1}$$

donde I_1 es la componente de fundamental de la corriente e I_h es el valor del armónico *h*.

En general, la incidencia de los armónicos está determinada por la susceptibilidad que la carga o la fuente tengan a su presencia. Ciertas cargas toleran mejor la presencia de armónicos de tensión y corriente que otras. Los equipos menos susceptibles son los que generan calor (hornos de arco, calefacción), y los equipos más susceptibles son los que se deben alimentar con una fuente perfectamente sinusoidal (comunicación, procesamiento de datos). Los principales efectos negativos de los armónicos en corriente y tensión son:

- La posibilidad de amplificación de niveles de armónicos como resultado de resonancias en paralelo y en serie.
- Reducción en la eficiencia de la generación, transmisión y utilización de la energía eléctrica.
- Envejecimiento en el aislamiento de componentes eléctricos y, consecuentemente, reducción de vida útil de estos.
- Incorrecta operación de equipos eléctricos.
- Interferencia en sistemas de telecomunicación.
- Pérdidas adicionales en máquinas rotativas, condensadores y alimentadores.

Para mantener los armónicos de tensión y corriente dentro de los límites recomendados por norma se requiere de una compensación o atenuación de éstas. La atenuación debe entenderse entonces como la reducción del nivel de inyección de la perturbación en el punto de conexión de la carga no lineal a la red eléctrica, y no como la desaparición de la perturbación (ya que ésta es de presencia obligada por la característica no lineal de la carga). En ese orden de ideas, las técnicas de mitigación evitan que las perturbaciones se propaguen al sistema eléctrico.

Algunas técnicas de eliminación de armónicos son:

- Conexiones de transformadores apropiados, con lo cual se logra compensar determinado contenido armónico.
- Filtros sintonizados: son empleados cuando el nivel de mitigación debe ser total, basados en la conexión en paralelo de un circuito tanque. Siendo ésta una de las soluciones mas comunes presenta limitaciones como ser: posible excitación de resonancias paralelo entre el filtro y la impedancia del sistema; el filtrado está acotado para un cierto ancho de banda y la respuesta del filtro es muy sensible a la variabilidad de los parámetros de la carga y el sistema.
- Filtros activos de potencia, estos son dispositivos diseñados para mejorar la calidad del suministro de energía eléctrica en particular la forma de onda en sistemas de distribución. Dependiendo de su forma de conexión pueden ser series o paralelos o combinación de ambos. En términos de su estrategia de control y forma de conexión es la posibilidad de compensación que presentan. La figura 3.1 muestra el esquema unipolar de la conexión del filtro activo paralelo de corrientes armónicas, donde i_L es la corriente que entrega la red, i_C es la corriente que consume la carga e i_F es la corriente que toma el filtro. Este método es el estudiado e implementado en este proyecto.

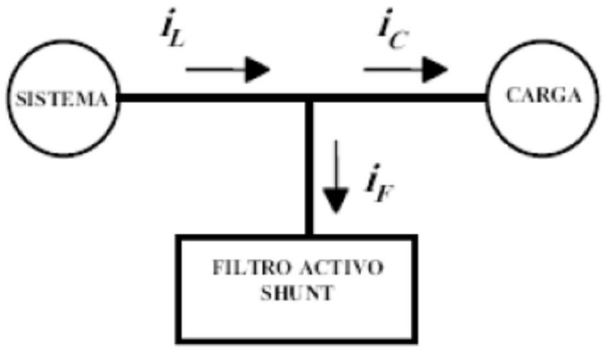


Figura 3.1 – Filtro activo, conexión al punto de compensación.

3.2. Teoría pq aplicada al filtrado de armónicos

Esta sección se refiere íntegramente al trabajo desarrollado en la tesis de doctorado del Dr. Gonzalo Casaravilla [7], pretendiendo ser esto un pequeño resumen de los punto tenidos en cuenta en la implementación de la celda básica de filtrado, no siendo esto sustituto de la lectura de la tesis para la compresión cabal de todo su trabajo.

Debe de quedar claro que todo el fundamento de por qué esta manera de operar sirve para realizar el filtrado no se expondrá aquí, pues la misma requiere varios tópicos extensamente comentados en la tesis. Sí se expondrán las hipótesis de trabajo y las operaciones que realiza el filtro.

Hipótesis de trabajo

- En todo momento se trabajará con sistemas de tres hilos por lo que no habrá componente homopolar ni serán consideradas las topologías que sí las contemplan. Esta es la situación usual en los sistemas eléctricos de potencia donde las corrientes homopolares se bloquean con transformadores adecuados.

- No se eliminará todo el residuo armónico (todos los armónicos de tensión o corriente que no son de la frecuencia de red del sistema eléctrico), sino que la idea es filtrar solo lo estrictamente necesario. Por otra parte si bien un filtro activo podría cumplir simultáneamente con otras tareas como ser compensar reactiva y desbalances de cargas, los mismos no serán considerados.
- La topología de compensación será, en cuanto a medida de la corriente de carga, tipo feedfoward (figura 3.2). Como se concluye en [7], el método óptimo a utilizar es entonces el llamado método serie, en el que las distintas celdas selectivas van en cascada (figura 3.3).

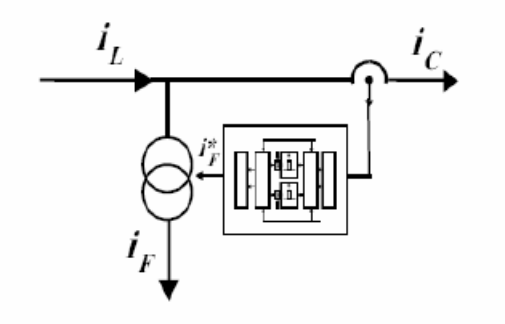


Figura 3.2 – Esquema feedforward.

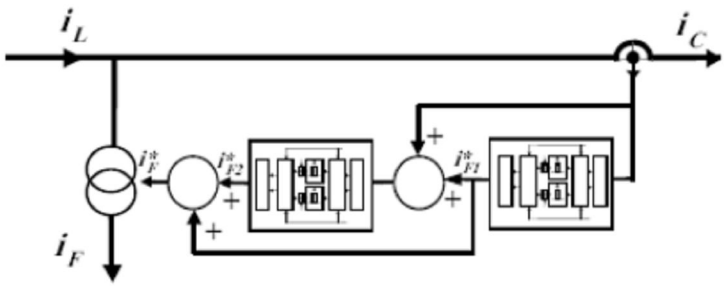


Figura 3.3 – Método de compensación feedforward – Serie – Compensación de dos secuencias.

Descripción de la celda

Basándose en los capítulos 1, 2 y 3 de [7], se interpreta la forma de realizar el filtrado. No se pretende entrar en detalle con la teoría, sino mostrar básicamente los algoritmos del filtrado selectivo basado en la teoría pq [15]. Originalmente se distinguen dos etapas, que conjugan la misma celda básica operatoria pero se diferencia en la etapa de filtrado, se distinguen:

- "Celda de Filtrado Selectivo", donde la celda se encarga de generar el contenido armónico correspondiente al armónico que trata de compensar, éste involucra dos filtros pasabajos, Butterworths de 2^o orden.
- "Celda de Filtrado Residual", se encarga de generar un contenido armónico que trata de compensar en el punto de conexión a todas la armónicas existentes en vez de ser selectivo, éste involucra dos filtros Butterworth de 2do orden, pero pasaalto.

Ambas etapas de filtrado involucran las siguientes operaciones:

- Transformada de Clarke: Una de las transformadas más comunes de un sistema trifásico a uno bifásico ortogonal (debido a la ausencia de componente homopolar), queda entonces definida su transformada directa a través de la expresión 3.2:

$$\begin{bmatrix} i_o \\ i_\alpha \\ i_\beta \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} i_R \\ i_S \\ i_T \end{bmatrix} \quad (3.2)$$

En el caso estudiado, no aparece la componente i_0, pues no se tiene homopolar en el sistema.

- Cálculo pq (Modulación): Aquí se definen la potencia instantánea homopolar "p_o" (la cual no se usa aquí), potencia real instantánea "p" y potencia instantánea imaginaria "q" que conforman el enunciado de la teoría p q.

$$\begin{bmatrix} p_o \\ p \\ q \end{bmatrix} = \begin{bmatrix} v_o & 0 & 0 \\ 0 & v_\alpha & v_\beta \\ 0 & v_\beta & -v_\alpha \end{bmatrix} \begin{bmatrix} i_o \\ i_\alpha \\ i_\beta \end{bmatrix} \tag{3.3}$$

El cálculo se realizará considerando, para el caso de una secuencia positiva (para secuencia negativa es lo mismo cambiando de signo la frecuencia), empleándose lo mismo para la inversión:

$$\begin{bmatrix} v_\alpha(t) \\ v_\beta(t) \end{bmatrix}_+ = \begin{bmatrix} +sin(w_c t) \\ -cos(w_c t) \end{bmatrix} \tag{3.4}$$

- Etapa de filtrado: Distinguida según sea filtrado residual o selectivo.
- Multiplicación por -1
- Inverso de cálculo pq (Demodulación): No considerando la componente homopolar la inversión se realiza a partir de:

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \frac{1}{v_\alpha^2 + v_\beta^2} \begin{bmatrix} v_\alpha & v_\beta \\ v_\beta & -v_\alpha \end{bmatrix} \begin{bmatrix} p \\ q \end{bmatrix} \tag{3.5}$$

- Inversa de la Transformada de Clarke:

$$\begin{bmatrix} i_R \\ i_S \\ i_T \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \frac{1}{\sqrt{2}} & 1 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} i_o \\ i_\alpha \\ i_\beta \end{bmatrix} \tag{3.6}$$

Las figura 3.4 y 3.5 muestran esquemas gráficos de las operaciones involucradas en la celda de filtrado residual y selectivo respectivamente.

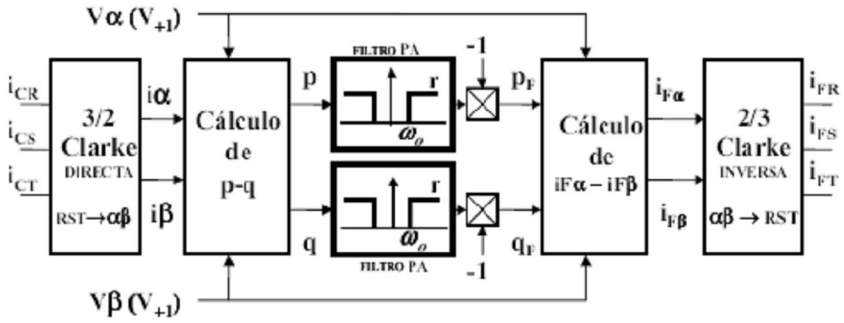


Figura 3.4 – Celda de filtrado residual.

Para la realización del filtrado de una corriente trifásica dada se sigue el esquema comentado en [7]. Aquí se muestra el filtrado total de las componentes armónicas 3,5,7,9. La figura 3.6 sugiere el algoritmo de cálculo para tal fin. Entiéndase a cada bloque como una celda básica de filtrado selectivo, concatenadas según el método "serie" y a su vez en cascada con éstas el filtrado residual, que se interconectan como sugiere la figura 3.7.

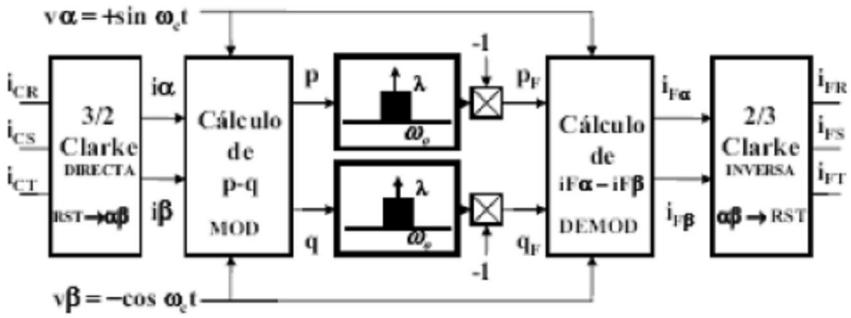


Figura 3.5 – Celda de filtrado selectivo.

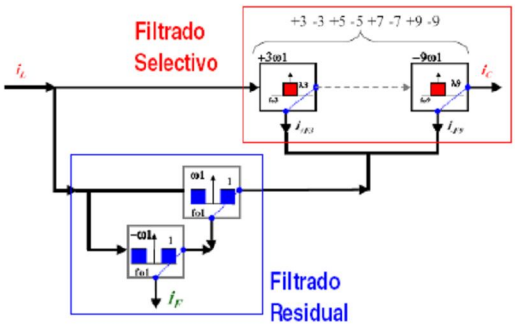


Figura 3.6 - Ejemplo de filtrado – Feedfoward – Selectivo múltiple y Residual Serie.

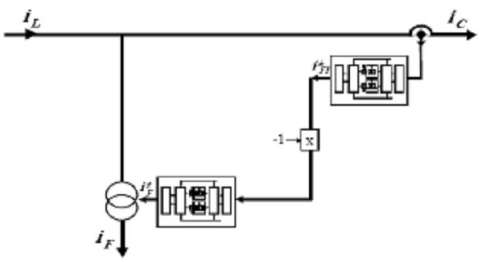


Figura 3.7 – Ejemplo de filtrado – Concatenado – Selectivo múltiple y Residual Serie.

3.3. Otros bloques

En esta sección se intentó resumir la teoría pq aplicada al filtrado selectivo de armónicos de corriente. Aunque esto es el núcleo del sistema de control, se necesitan otros bloques, que aunque se podrían denominar como complementarios, son tan importantes como el bloque de filtrado. Estos son en particular:

- PLL, para tener la fase de la red en tiempo real. En las simulaciones y primer hardware in-the-loop se implementó un bloque el cual genera la fase a partir de los cruces por cero de la tensión de fase R. Sin embargo, en las últimas etapas del proyecto se implementó el denominado SSI-PLL (extraído de [13]), el cual tiene varias ventajas que se verán en otra sección.
- El método de control del VSI, para obtener con este las corrientes que se calcularon con la teoría pq. El método adoptado es el control vectorial, utilizado en [3] (adjuntamos una extracción en el CD). Las modificaciones realizadas para este proyecto se explican en la sección 5.3.7.
- Control de la tensión en el condensador, consta del agregado de un bloque PI al bloque de filtrado residual.

4. Simulaciones

4.1. Introducción

Actualmente existen en el medio herramientas de simulación que permiten saber de forma más o menos precisa cómo se comportará el sistema a desarrollar. Para este proyecto se utilizó Simulink, de Matlab, pues es una herramienta muy potente y a la vez fácil de utilizar.

Es importante al momento de simular tomar modelos que sean representativos de la realidad, sin caer en una complejidad innecesaria. En el transcurso de este proyecto hubo algunos temas que no se previeron oportunamente, en particular la tensión a la que podía llegar el bus de continua. Por esta razón no se puede hacer comparaciones directas en varios resultados obtenidos en las simulaciones con los resultados con el sistema real.

En esta sección se presenta brevemente la teoría utilizada en el diseño del filtro de armónicos de corriente. Luego se verán las simulaciones realizadas, cuya complejidad fue creciendo al pasar el tiempo.

4.2. Implementación del Filtro Activo ideal

Inicialmente se implementó un bloque de filtrado el cual consiste principalmente en un bloque de filtrado selectivo y otro de filtrado residual, tal como se muestra en la figura 4.2.1. Este bloque calcula la corriente que debe tomar el filtro, a partir de la corriente hacia la carga.

Como se ve en la teoría, es necesario realizar la transformada de Clarke. En la figura se ve como aparece la transformada solo a la entrada del filtro y la anti-transformada a la salida. Hay que notar que la transformación no se hace en cada celda, pues se estarían realizando operaciones de manera innecesaria.

Los bloques mencionados tienen como entrada la señal a filtrar, tal como se explicó en la sección anterior, y la fase. En esta etapa se supondrá frecuencia constante e igual a 50Hz, por lo que podemos generar nuestra fase sin saber cómo es la tensión en cada momento.

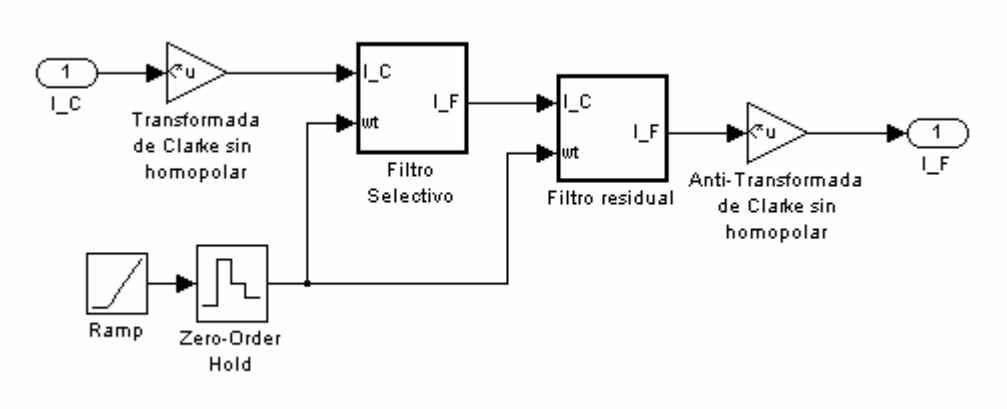


Figura 4.2.1 – Bloque de filtrado

También cabe mencionar que el bloque carga parámetros desde el "workspace", necesarios para su configuración.

Filtro Selectivo

El bloque de filtrado selectivo consta de varias celdas selectivas (SFBC) en serie, las cuales se activan independientemente mediante un vector en memoria creado para ese fin (vector de ganancias). La corriente final se obtiene mediante la suma de las salidas de todas estas celdas. Se optó por que el bloque de filtrado selectivo tuviera solo celdas capaces de filtrar armónicos impares del 3 al 25 inclusive.

En la figura 4.2.2 se observa la vista del filtro por dentro. Es importante aclarar que cada "SFBC_i" en la figura no es en realidad una celda básica, sino que está constituida por dos SFBC independientes: la secuencia positiva y negativa (figura 4.2.3). El bloque

“ganancias1“ lee en memoria el vector de ganancias para asignarle una cierta ganancia a cada celda. Si la ganancia es cero, entonces el bloque directamente se desactiva (mediante su entrada enable) para no tener que hacer cuentas innecesariamente.

La fase de entrada se multiplica por el número de armónico correspondiente para luego tener las sinusoidales de la frecuencia necesaria para el cálculo en cada celda.

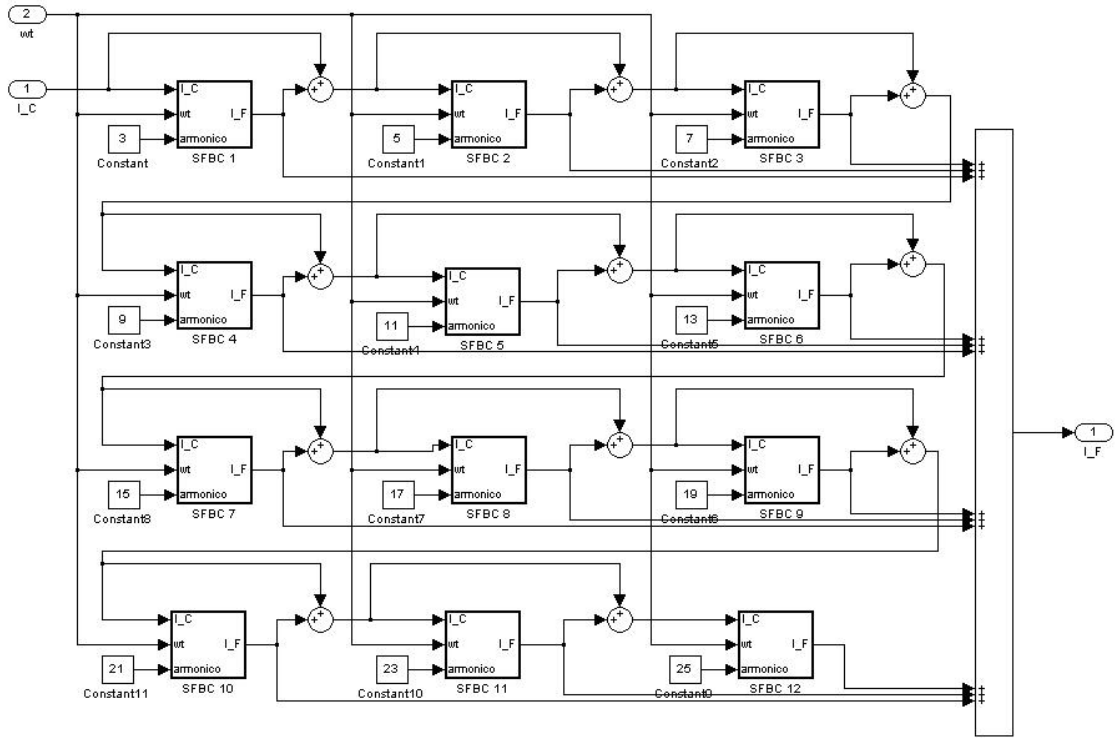


Figura 4.2.2 – Celdas selectivas

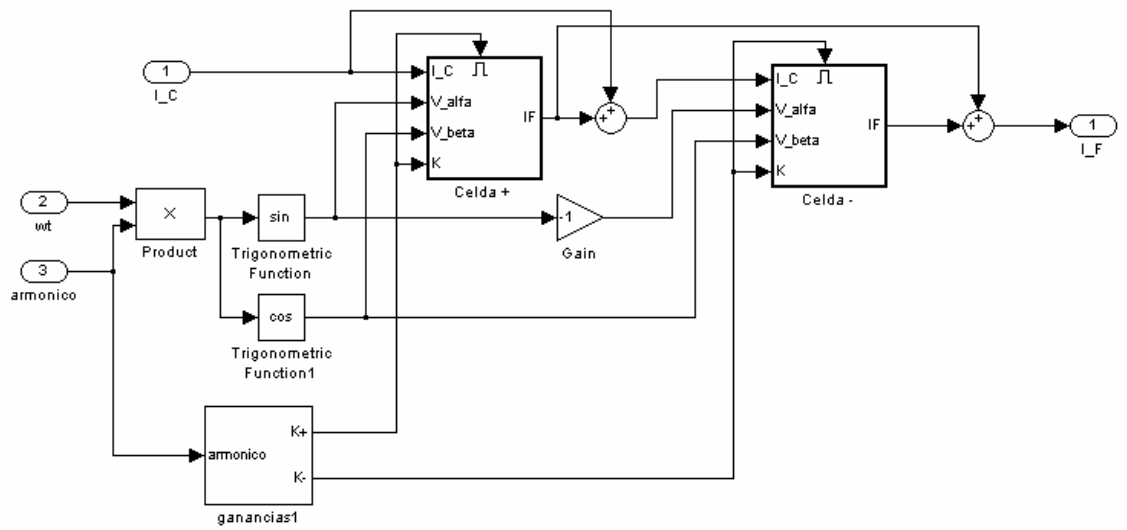


Figura 4.2.3 – Celda selectiva, para una secuencia positiva y negativa

En la figura 4.2.4 se muestra cómo se implementaron las SFBC. Hay que notar que todas son idénticas, lo que las hace diferentes son sus entradas. No hay nada complicado en su construcción, solo está implementada la teoría al pie de la letra, salvo un par de detalles:

- Los filtros pasabajos se implementaron con filtros pasaaltos. Con esto se tiene una mejor respuesta en cuanto a la fase, según se dice en [14]).
- El demodulador (o transformación de corrientes y tensiones de Clarke a las potencias p y q) y el demodulador (antitransformación) son idénticos pues las tensiones alfa y beta son sinusoidales de amplitud 1.

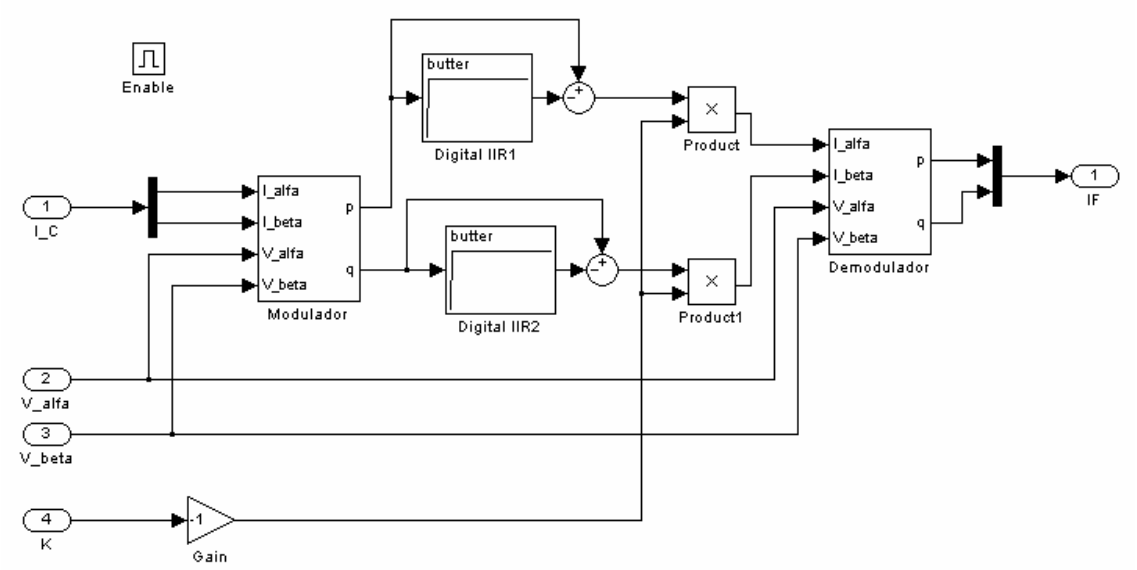


Figura 4.2.4 – Celda selectiva básica, SFBC

Filtro Residual

Con lo visto hasta ahora no hace falta mostrar con mucho detalle como se implementó el filtro residual. Solo consta de un par de celdas residuales (RFBC), las cuales se diferencias del las SFBC por los filtros IIR butterworth, los cuales son pasaaltos en vez de pasabajos. Ver la figura 4.2.5.

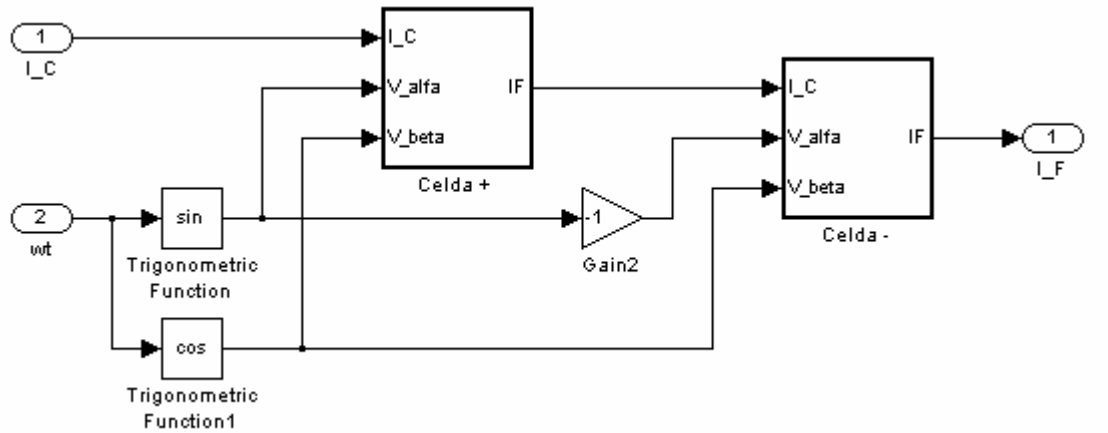


Figura 4.2.5 – Celdas para el filtrado residual, RFBC

Ensayos

Se realizaron numerosos ensayos para observar el comportamiento del filtro en el caso más ideal. En la figura 4.2.6 se observa uno de ellos, en el que se generó una corriente de 5A (eficaces) de fundamental, un 50% de 5º armónico de secuencia negativa y un 10% de armónico 19 de secuencia positiva. El filtro se configuró con ganancia 1 para esas secuencias, con lo que se pretende filtrar el 100% de cada secuencia. En la figura se observa las tres corrientes de interés para la fase R: corriente a la carga, por el filtro y línea (red). Para las otras fases el resultado es similar.

La corriente de red en este caso simplemente se calculó como I_C+I_F pues estamos en el caso ideal de que logramos generar la corriente por el filtro exactamente igual a la que calculamos.

Cabe mencionar que para la generación de las señales de entrada al filtro se utilizó una función de Matlab ya implementada (*carga_generico.m* implementado por G. Casaravilla, revisado por D. Sosa). Se utilizó una frecuencia de muestreo de 51,2khz (1024 muestras por período de red).

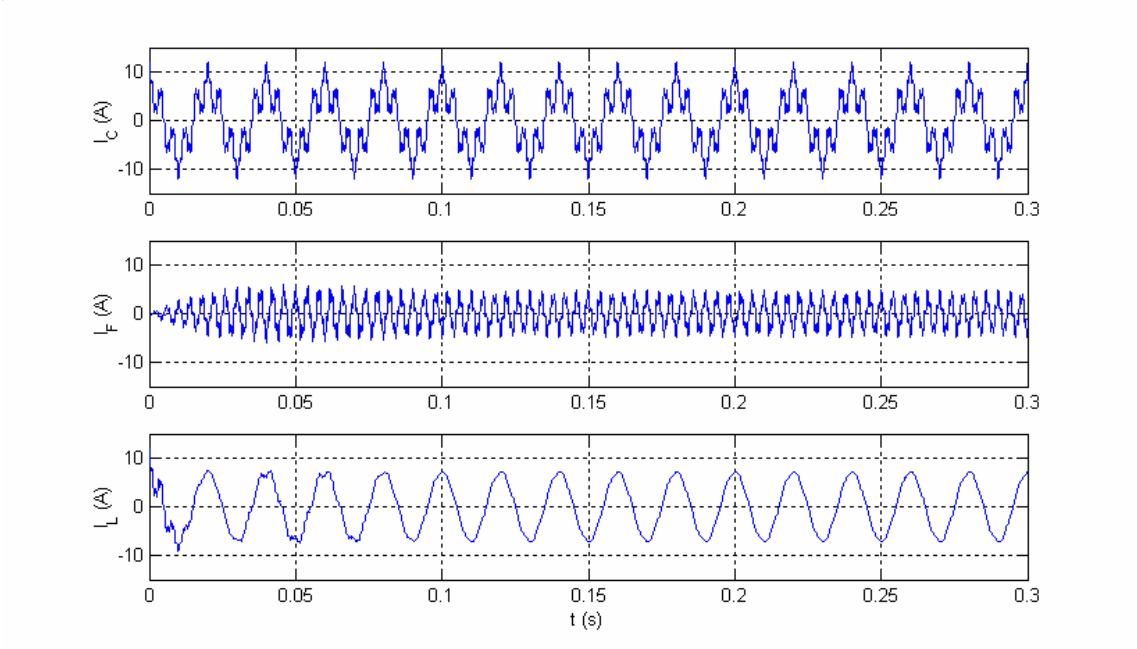


Figura 4.2.6 – Ejemplo de filtrado

En las formas de onda se observa un transitorio de aprox. 100ms debido a los filtros IIR. Luego la corriente por el filtro queda estable, dejando prácticamente sinusoidal a la corriente IL. En la figura 4.2.7 se observa el análisis en frecuencia para las señales en régimen.

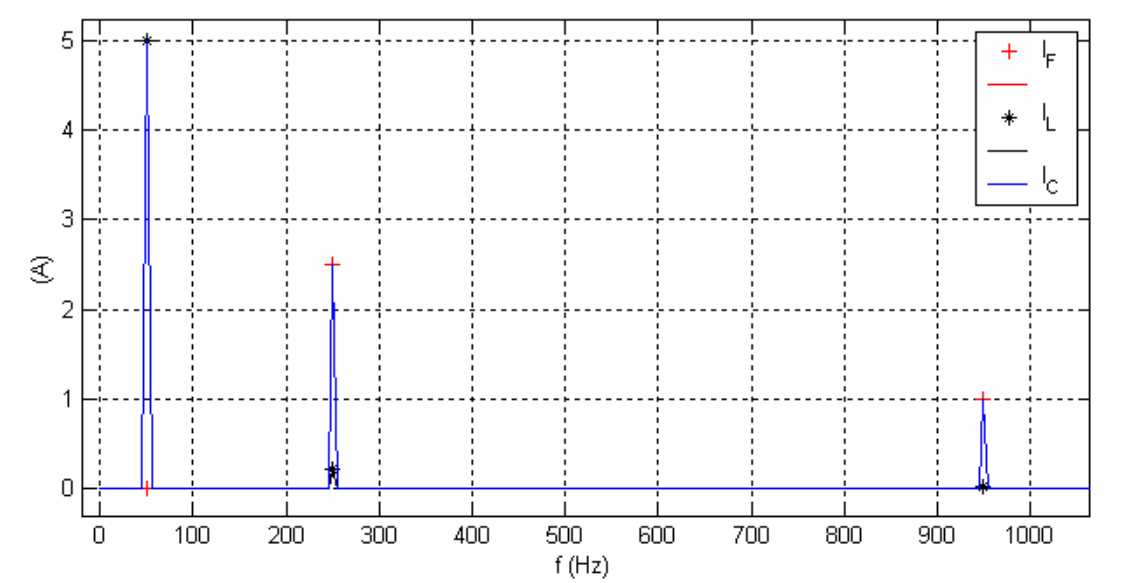


Figura 4.2.7 – Espectro de la figura 4.2.6

Se puede observar que la corriente I_F realmente no tiene fundamental y que los armónicos 5 y 19 tienen el mismo valor eficaz que el de los de I_C. Sin embargo la corriente I_L no tiene esos armónicos completamente anulados: el 5 se redujo a un 8.85% del valor inicial y el 19 a un 2.25%. Se deduce entonces que hay un desfasaje en la corriente que hace que las

corrientes no se anulen por completo. En las siguientes secciones se verán implementaciones para intentar corregir este problema.

Otro factor a tener en cuenta en el caso del filtro activo es su selectividad. En la figura 4.2.8 se muestra el resultado obtenido para el caso particular de filtro IIR de orden 2, con ancho de banda 15Hz. La corriente de carga que generamos en este caso tiene armónicos 5º, 7º y 9º de secuencia positiva de igual valor. Se observa que la selectividad no es completa, pues en este caso queremos filtrar completamente la secuencia positiva del 7º armónico dejando como están las secuencias positivas de los armónicos 5º y 9º, pero de todos modos aparece algo de éstos dos armónicos en I_F

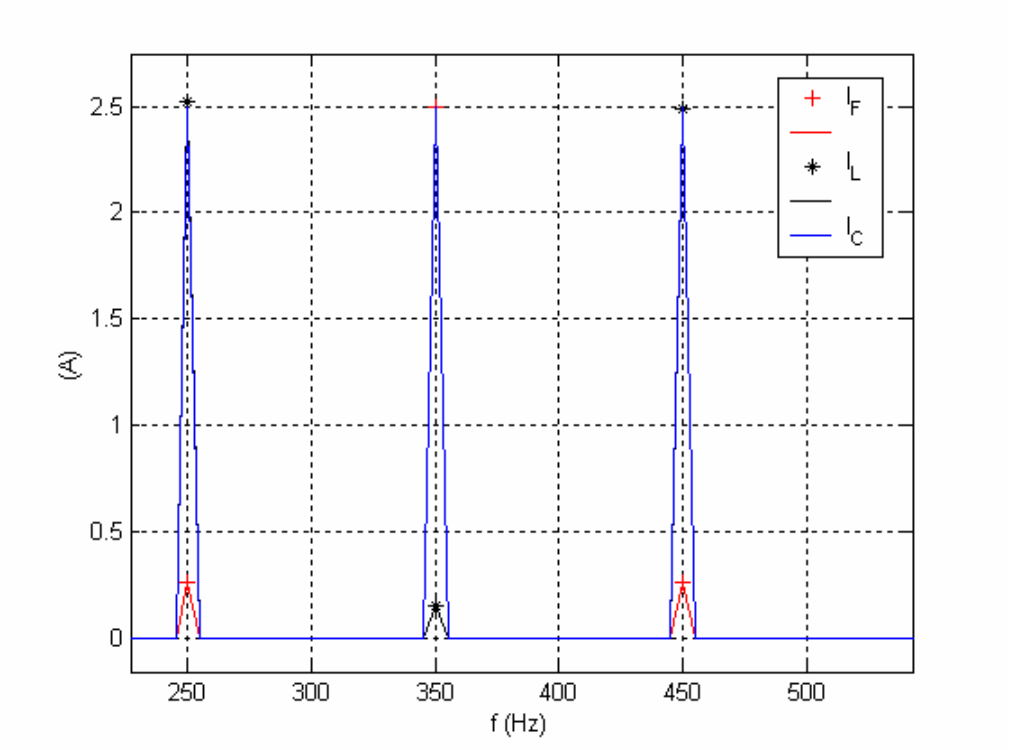


Figura 4.2.8 – Problema del desfasaje

Por otro lado, no olvidemos que el filtrado puede hacerse con ganancia entre 0 y 1. En la figura 4.2.9 aparece una pequeña muestra de este aspecto, para el caso anterior. Se observa que esta facultad funciona correctamente. Se impuso ganancia 0.5 para el armónico 7.

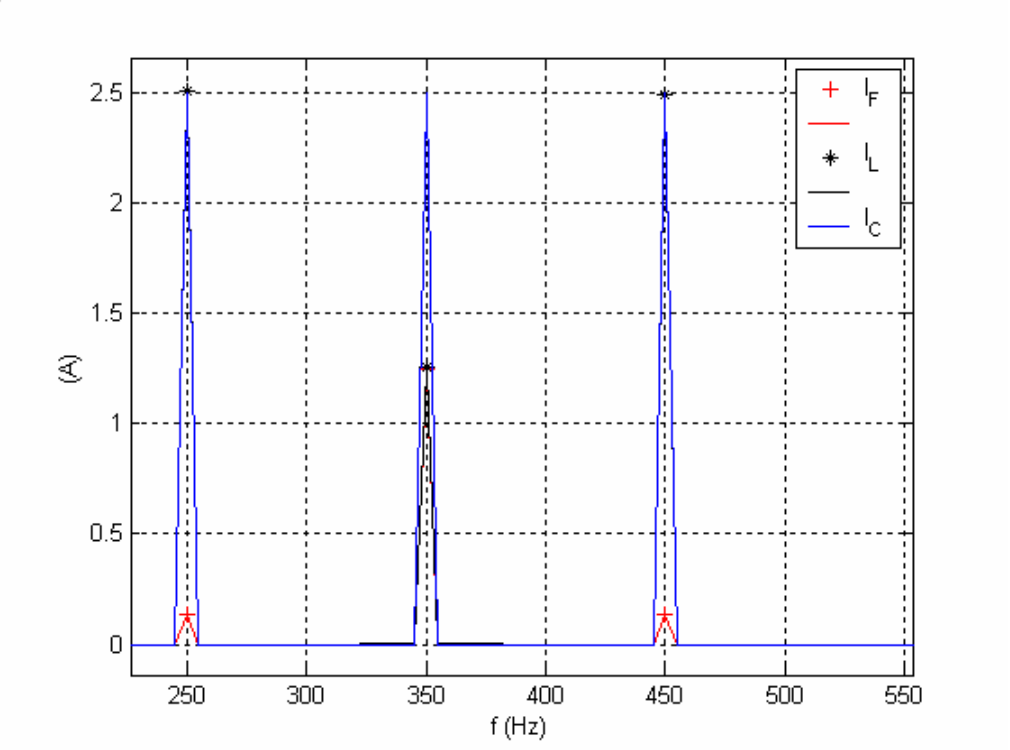


Figura 4.2.9 – Prueba de la ganancia

En principio, es de esperar que al agregar más complejidad al sistema, los resultados empeoren. En las siguientes secciones se mostrará que tanto se alejarán de la idealidad.

4.3. Efecto del Muestreo

El primer punto a tener en cuenta al pasar de la teoría a la realidad es que el sistema a implementar en el DSP va a tomar muestras a cierta frecuencia, lo cual puede afectar los cálculos del filtro.

Se realizó un ensayo para determinar cuál es la mínima frecuencia que nos va a brindar resultados aceptables. En la figura 4.3.1 se muestra cómo varía el armónico 5 de la corriente I_F en función de la frecuencia de muestreo (fs) del bloque y se ve la variación del armónico 19. Ambas curvas se presentan como porcentaje del respectivo armónico de I_C. Por otro lado, no hay que confundir la frecuencia de muestreo del bloque (fs) con la frecuencia de la simulación (y de las ondas de entrada) a la cual llamamos fsim.

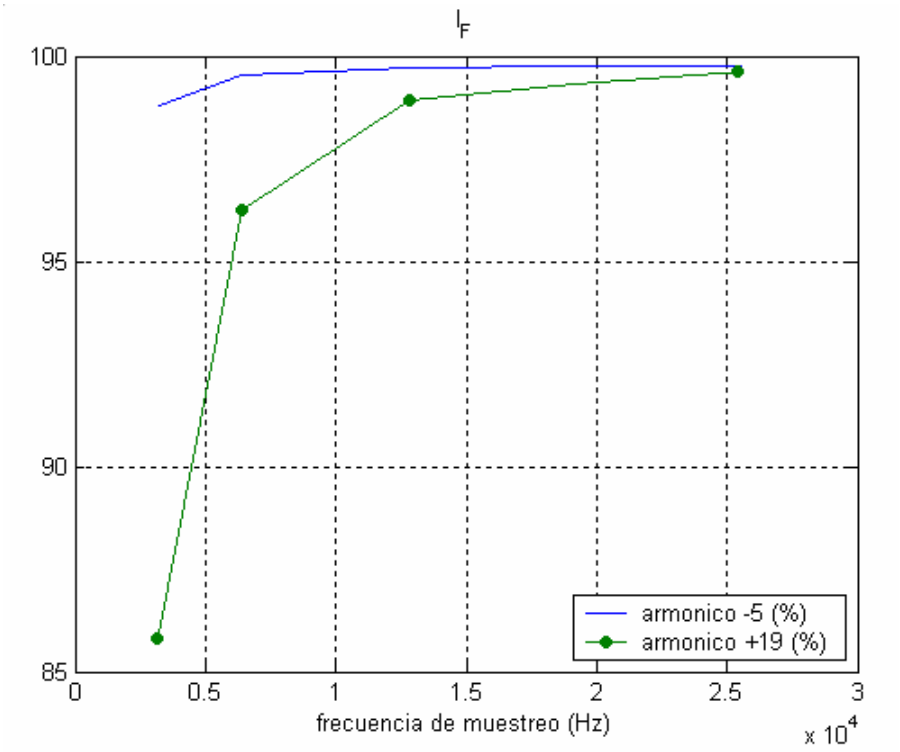


Figura 4.3.1 – Variación del filtrado con respecto a la frecuencia de muestreo

Era de esperarse que cuanto mayor sea fs mejor será el resultado, y que el armónico de mayor orden va a fijar el valor óptimo de la misma, pues será el más afectado (recordar la frecuencia de Nyquist). Si se quiere filtrar armónicos altos y con una cantidad de muestras por período de red que sea potencia de 2 (se verá que finalmente esto no fue necesario), entonces un valor adecuado será 12,8kHz (256 muestras por período). Aquí el armónico 19 casi 99% del valor ideal.

Sin embargo, en la figura 4.3.2 se observa un extraño efecto en la corriente de la red. A 6,4kHz (128 muestras por período) hay un mínimo en el 5º armónico y a frecuencias mayores comienza a aumentar. Como se mencionó en la parte anterior, esto se debe a que hay un desfasaje entre I_F e I_C, el cual (según se desprende de la figura 4.3.2) depende de la frecuencia de muestreo.

Por otro lado, en la figura 4.3.3 se observa el THD de I_L en comparación con el de I_C. Se concluye de esta figura que a partir de los 12,8kHz no se obtienen resultados mucho mejores al aumentar la frecuencia de muestreo.

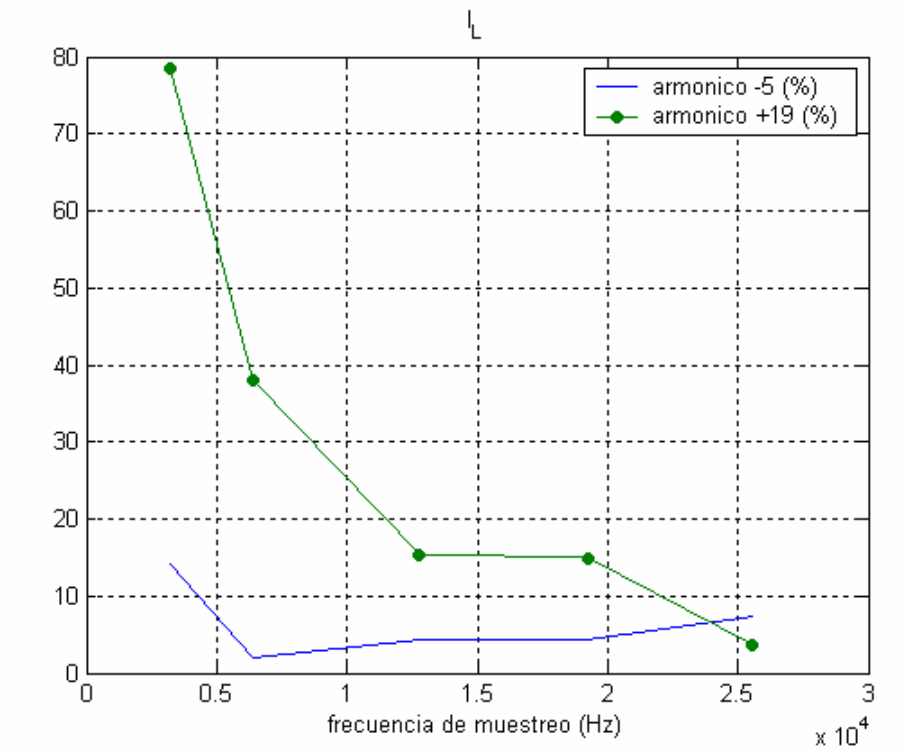


Figura 4.3.2 – Variación de la corriente de la red con la frecuencia de muestreo

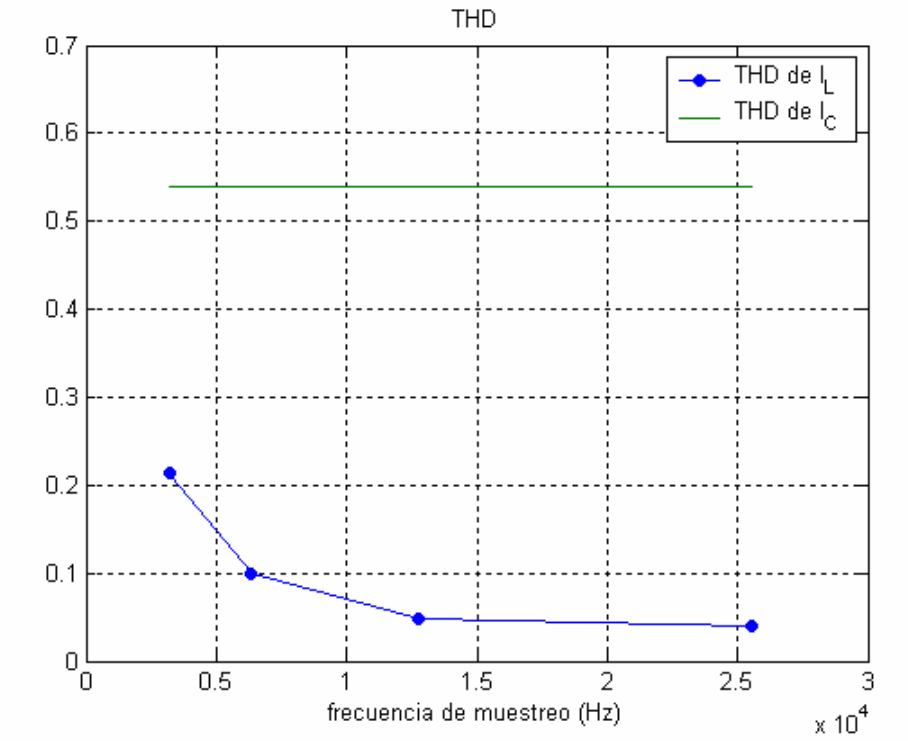


Figura 4.3.3 – Variación de la THD con respecto a la frecuencia de muestreo

4.4. PLL

En la primera etapa del proyecto se utilizó un PLL que lo único que hacía era determinar los cruces por cero para así determinar la fase de la red. Éste era muy sensible al ruido, por lo cual en la segunda etapa del proyecto se cambio por uno más robusto como lo es el SSI-PLL. Es por esta razón que en el anexo B, donde se detalla la elección del DSP, se tomó

en cuenta el PLL de los cruces por cero, dado que en ese momento no se había tenido en cuenta el SSI-PLL.

El PLL es utilizado para determinar la fase de la señal de tensión R. Esto es importante para generar los armónicos con la frecuencia correcta y también para independizarse de la frecuencia en los algoritmos. Si esto no se utilizara, se tendría el problema de que al tratar de generar armónicos se estarían dando corrimientos de frecuencia, o sea se estaría tratando de compensar unos Hz más arriba o más abajo de la frecuencia real del armónico. Aunque este problema se presentaría en armónicos de alta frecuencia, dado que las variaciones de frecuencia de la red son muy bajas, normalmente menores a ±0.2Hz. Más allá de eso, se sabe de [7] que un desfasaje constante no afecta el filtrado.

Por otro lado, el control de la tensión en el condensador necesita estar en fase con la secuencia positiva de la fundamental de tensión del punto de conexión con la red, dado que la potencia se controla con esta secuencia de corriente. Un desfasaje constante provocaría una corriente mayor a la necesaria para inyectar al filtro la misma potencia activa (la cual debe ser igual a las pérdidas para mantener constante la tensión en el condensador). Esto se traduce en un aumento de la potencia reactiva.

En [13] se estudian diferentes implementaciones de PLLs para el enganche con una red trifásica. Se concluye que para nuestra aplicación el más apto es el SSI-PLL, puesto que es robusto frente a distorsiones y desbalances en la red, sin excesivas operaciones.

La figura 4.4.1 muestra el esquema general del SSI-PLL [8] (Sinusoidal Signal Integrator – Phase Locked Loop). Al igual que en las celdas de filtrado se utiliza la transformada de Clarke, pero aplicada a las tensiones de red. Luego a las tensiones en el plano $\alpha\beta$ se le aplica el filtro SSI. Luego la transformada [T] (como se explica en [8]) es la que se muestra en la ecuación 4.1. Y por ultimo se le aplica el control PI a la salida *Vq* de la transformada [T]. La salida del PI es la velocidad angular del fasor de tensión, por lo cual integrando se obtiene la fase de la tensión, la cual es necesaria para la realización del filtrado selectivo.

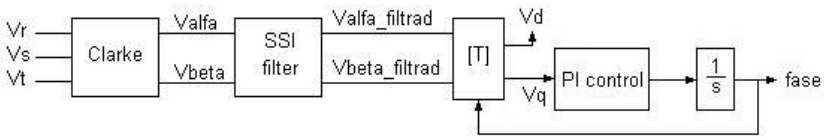


Figura 4.4.1 – SSI-PLL

$$[T] = \begin{pmatrix} cos(\theta) & sin(\theta) \\ -sin(\theta) & cos(\theta) \end{pmatrix} \tag{4.1}$$

Cabe acotar que la transformada [T] mencionada en [8] es igual a la modulación desarrollada en [7], en la que se obtienen *p* y *q*. De esta forma, el algoritmo se puede ver como el control de la potencia imaginaria instantánea (q) que se obtendría al generar un conjunto de corrientes trifásicas con la fase que se obtiene del PLL.

En la figura 4.4.2 se puede observar el filtro SSI. Este es utilizado para obtener la secuencia positiva fundamental de la tensión de la red. Este filtro hace una aproximación de Vbeta.

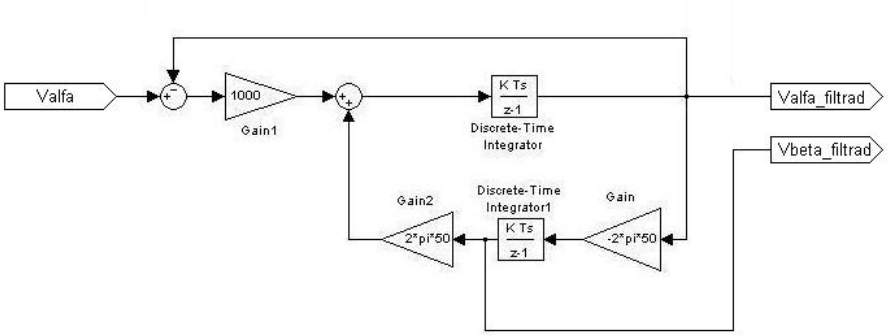


Figura 4.4.2 – Filtro SSI

El detalle del control PI y de la transformada [T], se puede ver en la figura 4.4.3.

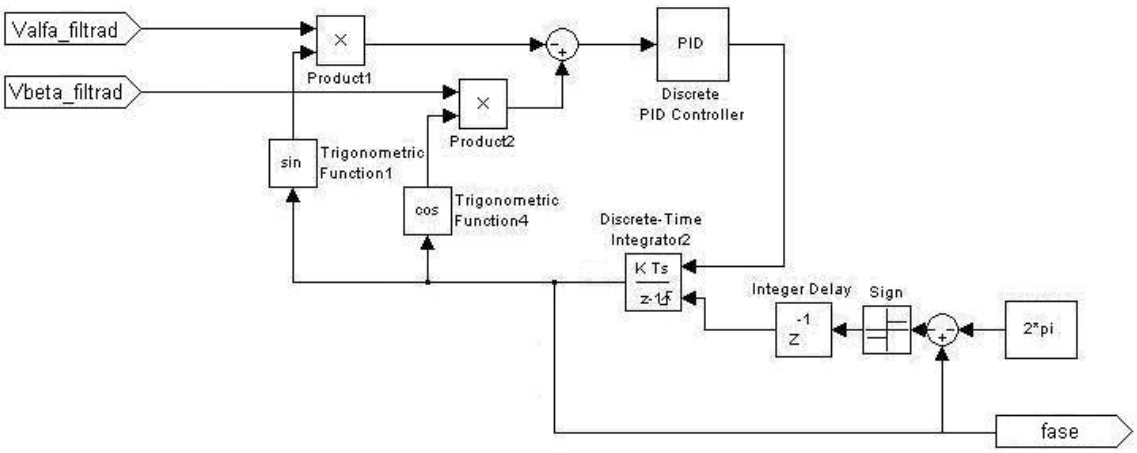


Figura 4.4.3 – Control PI y transformada [T]

Ensayos del PLL

Se evaluó el comportamiento del PLL frente a distorsiones que se presentan en la red eléctrica real, estos son variaciones en la frecuencia, huecos de tensión y saltos de fase. En la figura 4.4.4 se muestra el modelo completo con el cual se hizo este ensayo.

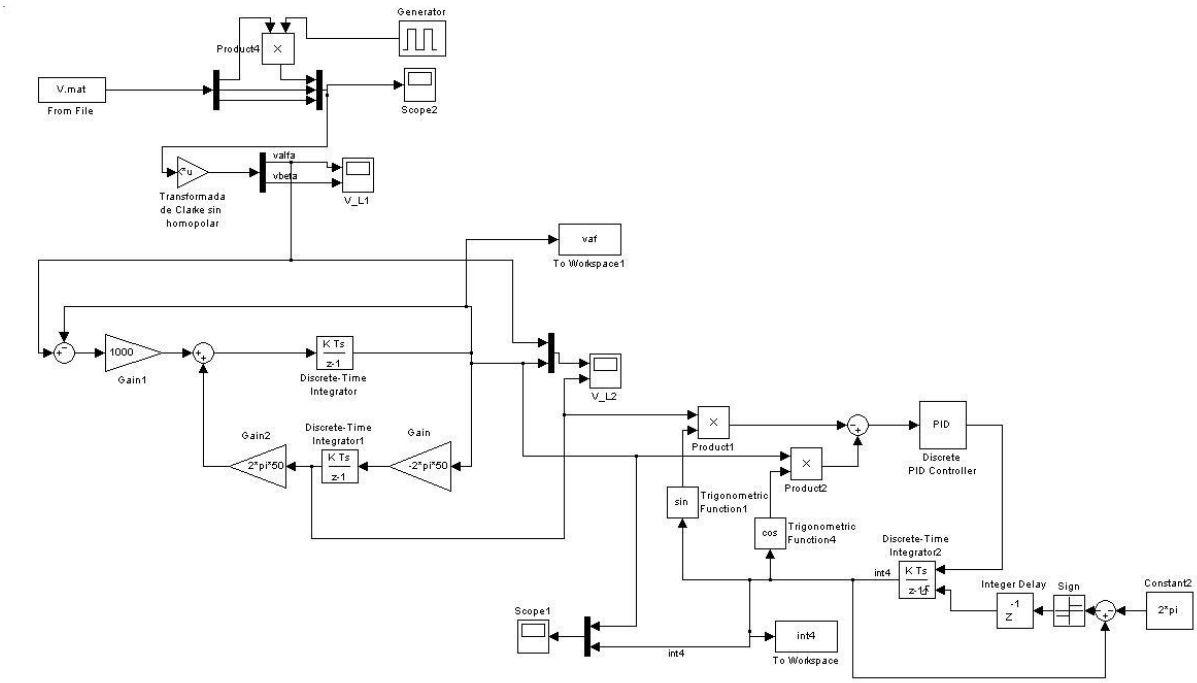


Figura 4.4.4 – Modelo de testeo del PLL

Respuesta transitoria

Antes de pasar a los ensayos, se evaluó la respuesta transitoria del SSI-PLL, en la figura 4.4.5 se muestra la señal $V_{alfafiltrada}$ (señal de la cual se debe determinar la fase) y la fase que genera el PLL. Se puede observar que el enganche se da antes de medio ciclo después del arranque. La pendiente de la fase generada es mayor antes de llegar al enganche, luego la pendiente pasa a ser la de la fase real. El motivo por el cual la amplitud de la señal $V_{alfafiltrada}$ es menor, es por el transitorio del filtro de SSI, eso sirve para ver que el filtro SSI llega al régimen luego de un ciclo.

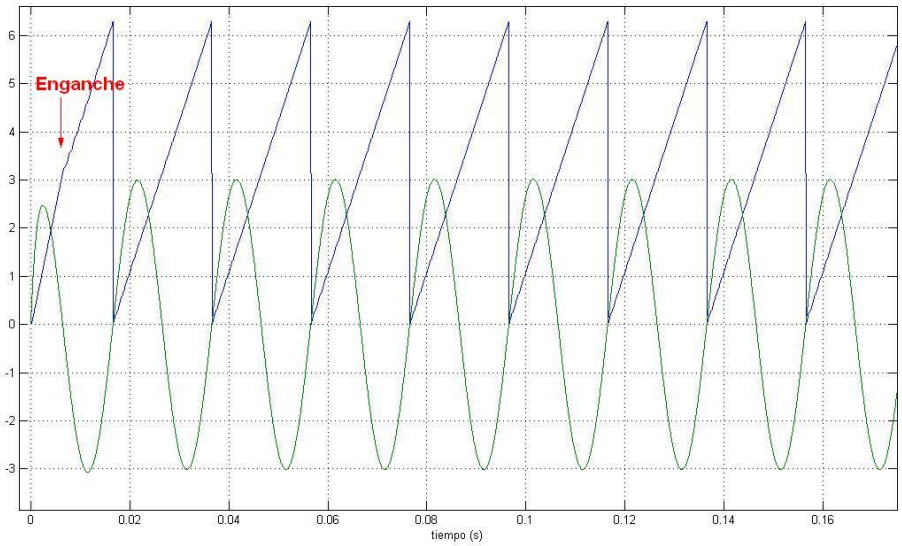


Figura 4.4.4 – Arranque del PLL

Cambios de frecuencia

Los cambios en la frecuencia de la red son poco frecuentes y de muy bajos valores, no superan lo ±0.2Hz.

En la figura 4.4.5 se muestra el resultado de una simulación de un salto de frecuencia. En la gráfica superior se puede observar la señal Valfa y la fase que detecta el PLL, y en la

gráfica inferior se observa el salto de frecuencia. El salto de frecuencia utilizado es excesivo, 5Hz, pero es adecuado para ver el comportamiento del PLL. Se puede ver que antes de los 0.14 segundos el PLL sigue estando en fase con la señal de tensión, por lo cual se concluye que en estas condiciones el PLL responde rápidamente a los cambios.

Se probó también que el PLL tenga buena sensibilidad, o sea se probó con un salto de frecuencia pequeño, en este caso de 0.1Hz, en la figura 4.4.6 se observa el comportamiento del PLL frente a este cambio y nuevamente el enganche es inmediato.

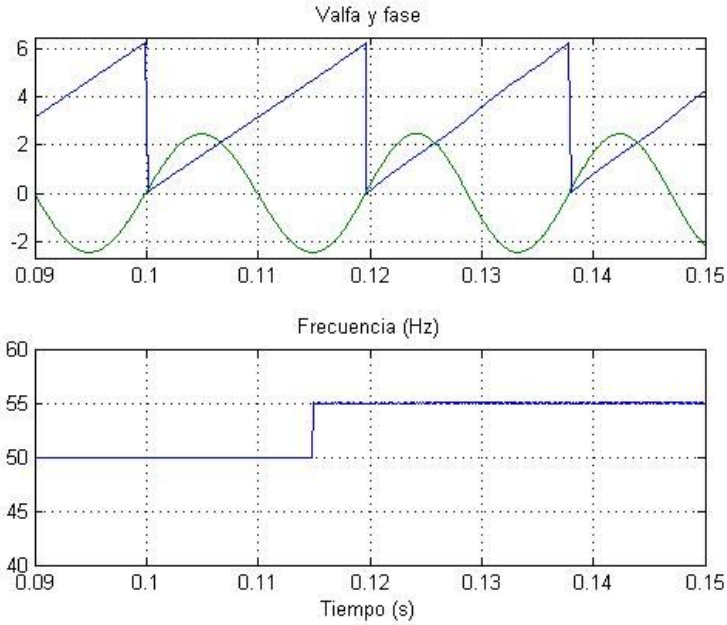


Figura 4.4.5 – Salto de frecuencia de 5Hz

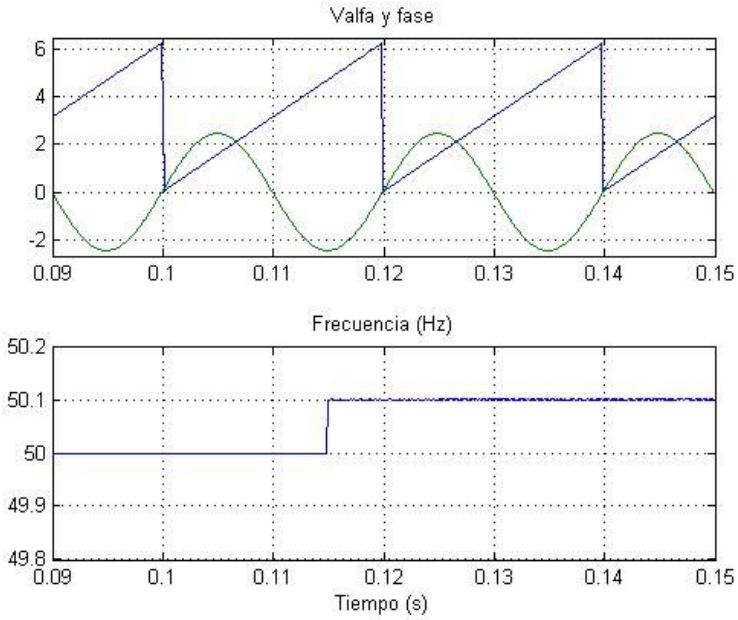


Figura 4.4.6 – Salto de frecuencia de 0.5Hz

Huecos de tensión

Un fenómeno que se presenta en las redes eléctricas son los huecos de tensión. Éstos son la caída de tensión a causa del aumento repentino de carga en la red, corto circuitos o faltas [9]. Se simuló una caída de tensión en las tres fases, en la figura 4.4.7 se puede ver el comportamiento del PLL frente a la perturbación. En esta se grafican la señal Valfa y la fase generada por el PLL (superior), y las tres fases de tensión de la red.

Se observa que en el momento que se da la caída de tensión, el PLL reacciona rápidamente y luego de un ciclo ya esta enganchado. Cuando la tensión vuelve a la normalidad se observa nuevamente que el PLL no pierde el control.

Concluimos que frente a cambios de esa naturaleza el PLL responde satisfactoriamente.

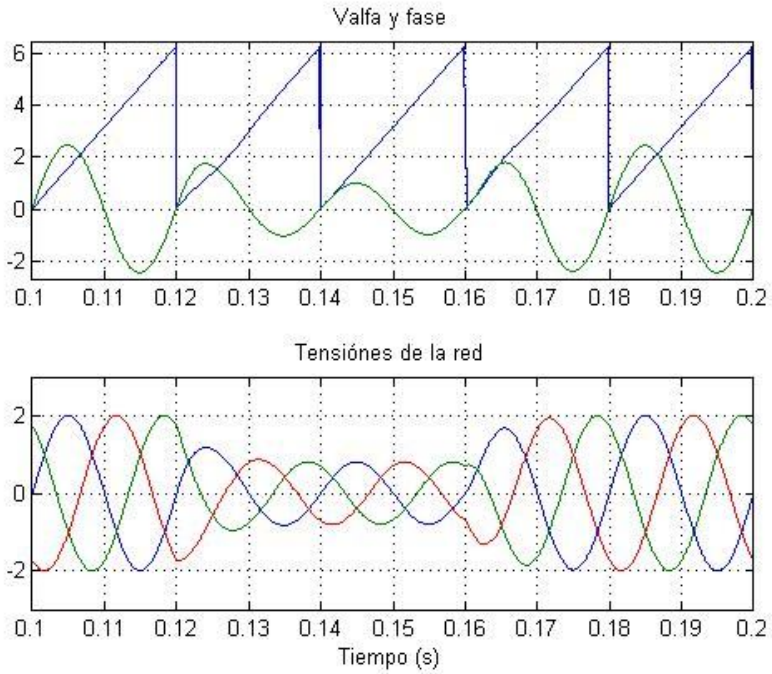


Figura 4.4.7 – Ensayo de hueco de tensión

Salto de fase

En la figura 4.4.8 se puede ver el resultado del ensayo del salto de fase. En la figura se marca el momento en el que se da el salto y cuando el PLL se encuentra en fase con la señal Valfa. Se puede ver que la respuesta del PLL se da en menos de medio ciclo, lo cual es muy rápido. En la señal de fase, se puede observar que cuando se da el salto, cambia inmediatamente la pendiente de la fase, luego en el enganche se observa el nuevo cambio de pendiente, para pasar a tener la pendiente real de la fase.

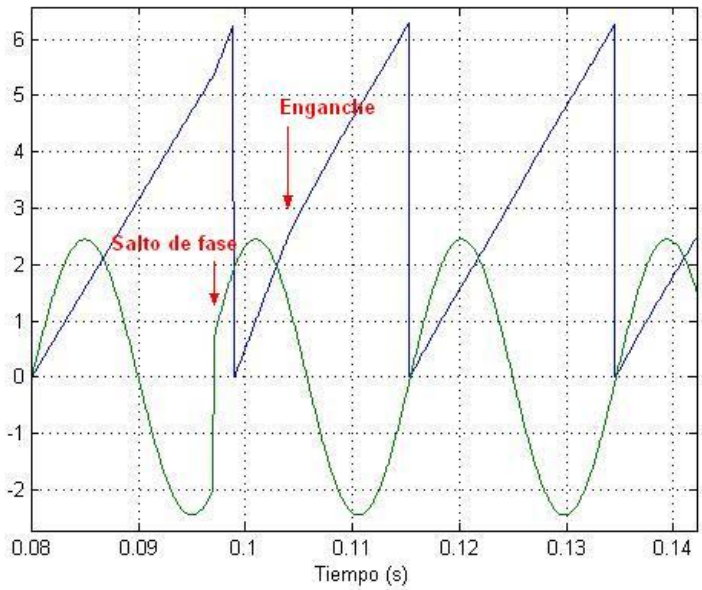


Figura 4.4.8 – Ensayo de salto de fase.

Como conclusión general sobre el SSI-PLL, podemos afirmar que es muy robusto frente a los cambios a los cuales se puede enfrentar en la realidad. La respuesta a los cambios en todos los casos fue medianamente rápida, con lo cual se puede prever que durante el filtrado, no se van a tener grandes transitorios por cambios en la respuesta del PLL.

4.5. Control del VSI

Una vez hallada la corriente que debe tomar el filtro, se debe elegir alguna estrategia para que el VSI tome la corriente deseada. Al conmutar las llaves del inversor se obtienen distintas tensiones de salida, variando así la corriente que este toma.

Control Bang Bang

Este tipo de control maneja independientemente cada una de las 3 fases. Consiste en observar la corriente por el filtro real en cada instante de tiempo y conmutar las llaves correspondientes a esa fase para lograr la tensión deseada. Si la corriente que mide es mayor a la deseada se conmutan las llaves de manera de que la tensión quede en su máximo (V_F=+E/2) y si la corriente es menor a la deseada se conmuta para que aumente (V_F= -E/2). Este control se hace instante a instante en los tres juegos de llave de forma independiente, lo que puede ser contraproducente para el inversor.

Control Vectorial

El bloque que implementaba el control vectorial de corriente fue proporcionado por otro grupo ya referido [3]. Este método fue propuesto originalmente por Akagi [15]. La idea del mismo es separar en tres zonas según que tan lejos estemos de la corriente de referencia: una zona en la cual no se actúa sobre las llaves, otra en la que se realizan pocas conmutaciones para la reducción de armónicos y la otra en la cual se actúa de la forma más rápida que se pueda. Estas zonas se determinan con 3 entradas que tiene el bloque: una de corriente eficaz, otra de porcentaje inferior de la corriente eficaz (limite inferior) y otra de porcentaje superior de la corriente eficaz (límite superior). Por lo mencionado, las señales de entrada a este bloque son la corriente real que entra al filtro (al VSI) y la corriente de referencia.

Ejemplo (Simulación incorporando el Control Vectorial)

Se tiene una corriente I_C con armónicos -5 y +19. Se filtró muestreando 256 veces por ciclo de red. Se usó control vectorial trabajando a 4 veces esa frecuencia (bastante exigente, lo cual da una frecuencia de 51.2 KHz) y con bandas de histéresis bastante ajustadas (h = 50mA, δ = 25mA, ver anexo). El VSI no está conectado a un condensador del lado de continua, sino por una batería de tensión E=500V. El resultado se observa en las figuras 4.5.1 y 4.5.2.

En la figura 4.5.1 se observa el ripple que añade la conmutación de las llaves, que si se observa el espectro 4.5.2, se concluye que no está formado por una frecuencia en particular sino por varias, por lo que se lo podría clasificar como ruido (aunque a nuestro tutor no le guste el término).

También se observa que la corriente real del filtro también tiene una componente fundamental (de 50Hz). De esto se hablará al final de la siguiente sección.

Por último, es claro que hay una gran dificultad para filtrar el armónico 19, mientras que el 5 parece inalterado respecto de las partes anteriores (recordar el desfasaje).

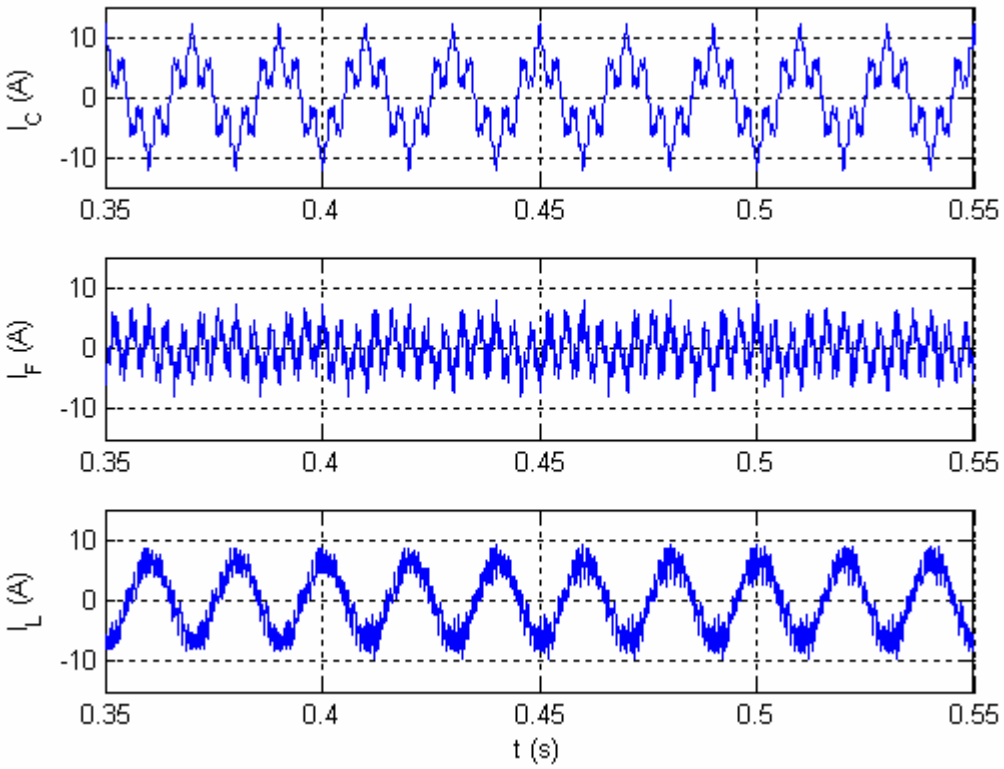


Figura 4.5.1 – Ejemplo de filtrado, armónicos -5 y 19.

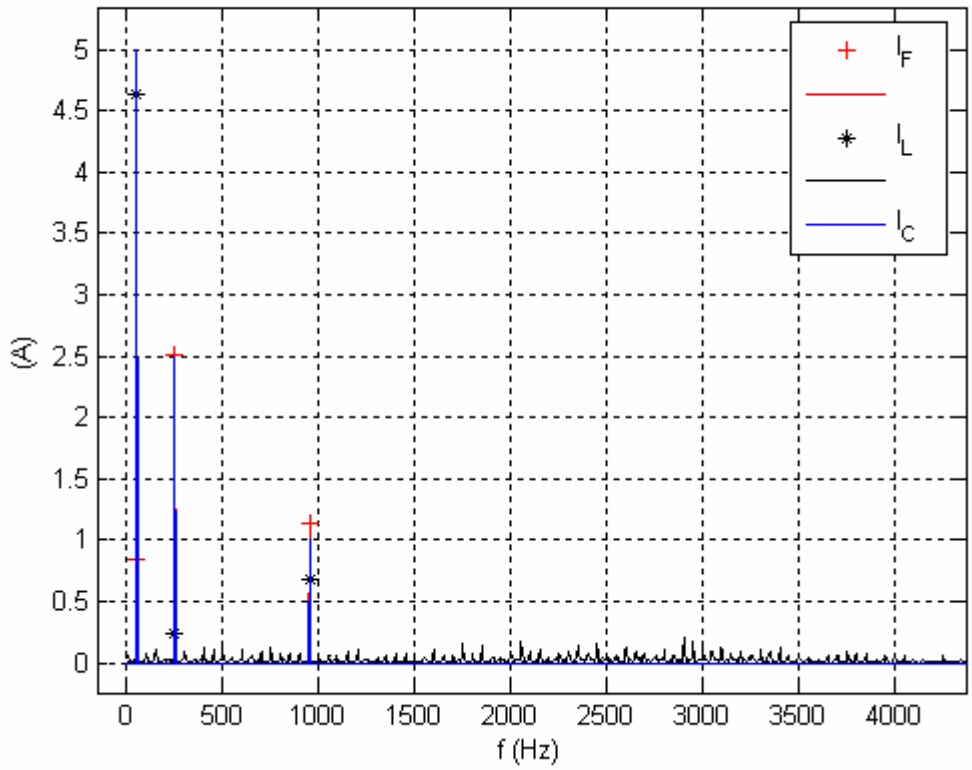


Figura 4.5.2 – Espectro de la figura 4.5.1

4.6. Control de tensión en el condensador

Debido a que el VSI no es ideal, ya que existen perdidas por conmutación por las llaves y que el condensador tiene su resistencia interna, se le debe suministrar potencia al condensador para que este mantenga su tensión relativamente constante. Esto es necesario ya que el control vectorial (estrategia para la conmutación de llaves) lo supone. Para ello, se debe utilizar algún tipo de control en cuanto a la potencia a entregar al condensador, a través de la propia conmutación de las llaves.

Primero que nada se debe elegir el condensador a utilizar se pueden usar varios criterios. Se utilizó el criterio de que la energía máxima oscilante en el condensador sea menor que un cierto porcentaje por encima de la energía promedio [7].

Como controlador de esta tensión se optó por un controlador PI (considerando como entrada la diferencia entre la tensión real y la tensión de referencia). De esta manera se obtiene una reacción rápida debida a la ganancia proporcional y una más lenta a través del integrador.

Esta potencia hallada a la salida del controlador PI, es agregada previa a la demodulación del último bloque del filtro (celda de filtrado residual de secuencia positiva). De esta manera la corriente que tomará el filtro ya incluye la necesaria para compensar las perdidas.

En todos los casos para hallar los valores óptimos de Kp y Ki (ganancias del bloque proporcional e integrador respectivamente) se fue iterando hasta obtener una respuesta adecuada, de manera de no obtener sobrepicos muy altos ni tiempos de establecimiento muy grandes.

Aunque no se realizó un estudio analítico, en primer lugar se simuló el comportamiento del sistema a partir de las ecuaciones del sistema de potencia (tensión de continua en función de la potencia entrante al filtro). En este caso las pérdidas en el inversor se supusieron constantes y el filtro se modeló como un retardo multiplicado por una ganancia.

En la figura 4.6.1 se observa el circuito simulado y en la figura 4.6.2 se observa como responden el controlador PI y la tensión en bornes del condensador para Ki = 1 y Kp =0,02. Se supuso perdidas constantes de 10 W, tensión inicial 325V ($\sqrt{2}U$) de salida y tensión de referencia de 500V. En condiciones normales el valor inicial de E es $\sqrt{2}U$ porque el VSI con las llaves apagadas funciona como un rectificador trifásico de diodos debido a los diodos en antiparalelo de las llaves. En la figura 4.6.2 se observa que se llega al régimen en menos de 1 segundo, con un sobretiro de menos de 20V (4% del valor en régimen) con respecto al valor final.

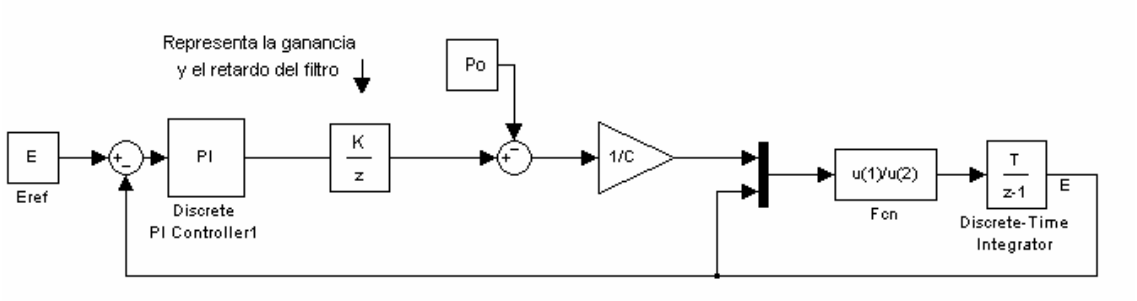


Figura 4.6.1 – Lazo de control simplificado

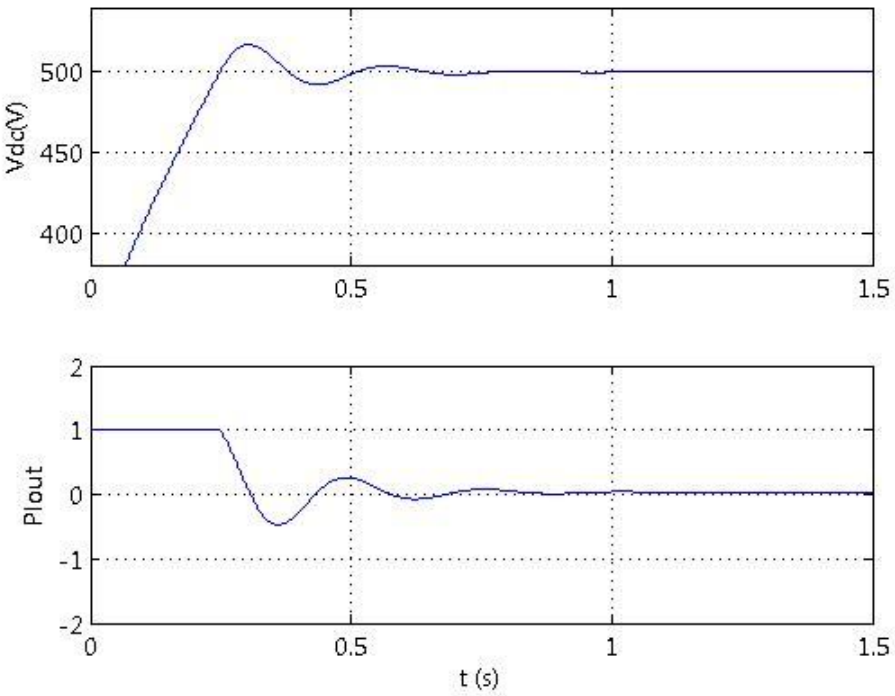


Figura 4.6.2 – Tensión en el condensador y salida del controlador PI, modelo simplificado.

A continuación se simuló en Simulink el sistema simulado por completo, incluyendo el circuito de potencia, y los algoritmos de filtrado. El circuito del VSI es el de la figura 4.6.3, donde la conmutación de llaves resulta del control vectorial. En la figura 4.6.4 se muestran las respuestas de la tensión y del controlador PI.

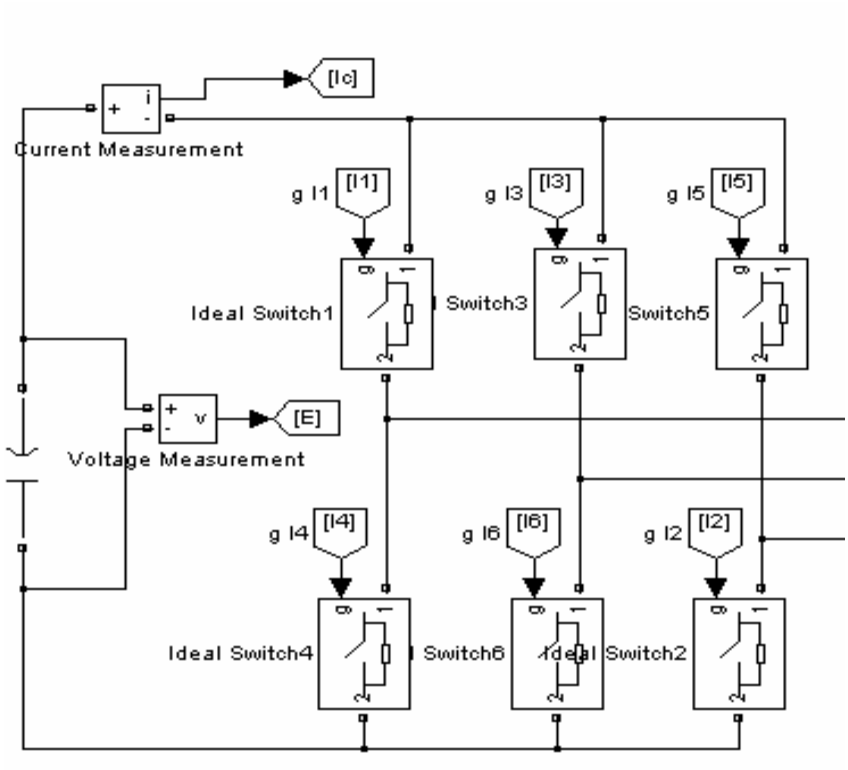


Figura 4.6.3 – VSI

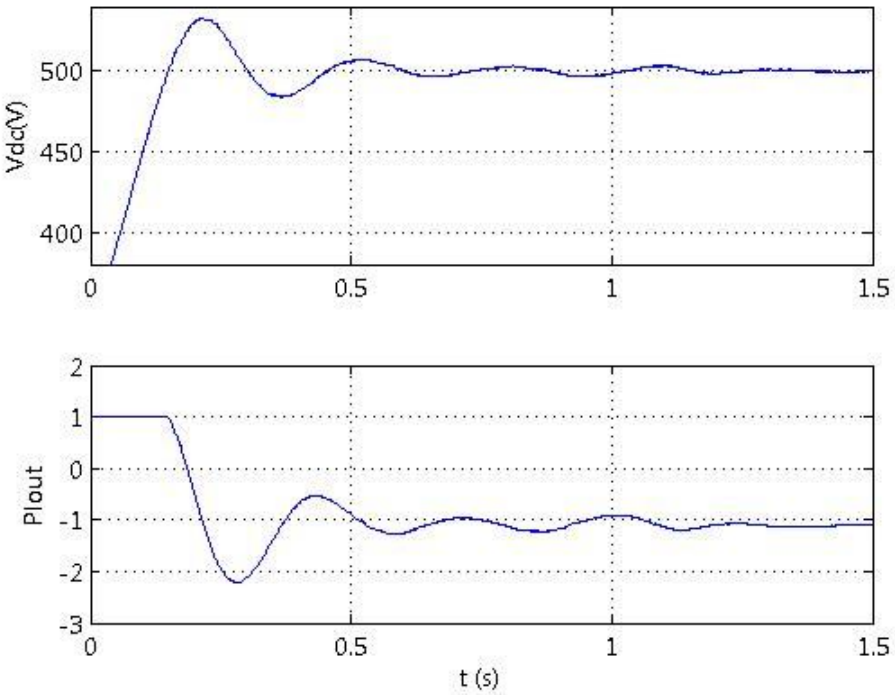


Figura 4.6.4 – Tensión en el condensador y salida del controlador PI, simulación del circuito de potencia.

En todos los casos se pudo observar que las constantes que mejor se ajustan al controlador PI para la tensión del condensador son aprox. Kp=0.02 y Ki=1. De los cuales se obtuvieron los resultados observados. También se limitó la salida del controlador entre 1 y -5, para no tener corrientes excesivas en el filtro.

Como primera observación se puede apreciar que el controlador tiene como valor de régimen un valor negativo, lo que significa que el condensador tiende a cargarse. Este efecto es causado por la conmutación de las llaves y se explicará en breve.

El hecho de que el valor de régimen no sea el mismo en ambos ejemplos causa que haya un sobretiro mayor cuando se simula el circuito de potencia, por lo tanto, el tiempo de establecimiento es mayor. Sin embargo, se observa que la frecuencia de las oscilaciones es similar, lo que significa que el modelo simplificado no se aparta tanto del modelo complejo del filtro.

Por otro lado, también se ve que el comportamiento en la figura 4.6.4 no es tan “lineal” como el de la figura 4.6.5 (nótese que ninguno de los lazos de control es lineal), puesto que las oscilaciones no decrecen de forma monótona.

En esta parte se intentó mostrar que se puede tomar un modelo simplificado del lazo de control de la tensión del condensador. Este modelo podría ser linealizado más fácilmente para un futuro estudio analítico. Sin embargo, como este estudio escapa al alcance de este proyecto, el ajuste de parámetros se realizó simplemente iterando. Tampoco se intentó que el ajuste fuese el óptimo.

Efecto de la conmutación

Como se introdujo en la parte anterior, un efecto interesante que se observó es que cuando se realiza la conmutación de las llaves, esta introduce una corriente fundamental. Recordar que el VSI está conectado a la red a través de inductancias, y que la derivada de corriente del filtro es proporcional a la diferencia entre la tensión a la salida del VSI y la de la red. Esta última se supone sin armónicos. En la siguiente figura (4.6.6) se puede observar que si se desea aumentar la corriente en una rama, esta se debe conectar a $+V_{DC}$, por lo que la diferencia entre tensiones es menor a que si decidimos disminuir la corriente (considerando 0V a la salida del inversor), por lo que se puede ver que la corriente va a disminuir mas rápido de lo que podría aumentar (debido a la diferencia de tensiones). Esto se repite a 50Hz, es por ello que se genera la fundamental de corriente además de la componente de alta frecuencia.

La corriente resultante resulta tener una componente de 50Hz que además está en fase con la tensión en esa fase, como consecuencia (como estamos tomando la corriente I_F entrante al filtro) entra potencia al filtro y por lo tanto al condensador. En oposición a lo que uno pensaría normalmente, el condensador debería tender a descargarse debido a las pérdidas, pero por el efecto mencionado, este se cargaría indefinidamente.

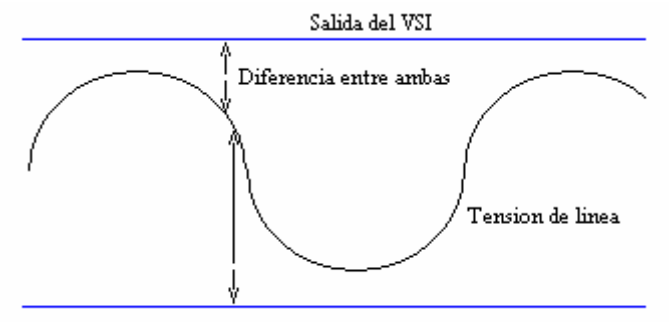


Figura 4.6.6 – Efecto de la conmutación. Diferencia de tensiones

Este problema es solucionado por el controlador PI que se agregó para controlar la tensión en el condensador. Dicho controlador maneja la señal del canal p de la celda residual correspondiente a la secuencia +1 de la corriente del filtro, lo que significa que controla el valor de la fundamental en fase con la secuencia +1 de la tensión red. Ajustando esta componente de corriente el controlador es capaz de controlar la potencia media entrante al filtro aún cuando la corriente real no es igual a la corriente de referencia. En la figura 4.6.7 se observa como ahora la corriente I_F casi no tiene fundamental. La fundamental que aparece es la necesaria para compensar las pérdidas en las llaves y cables.

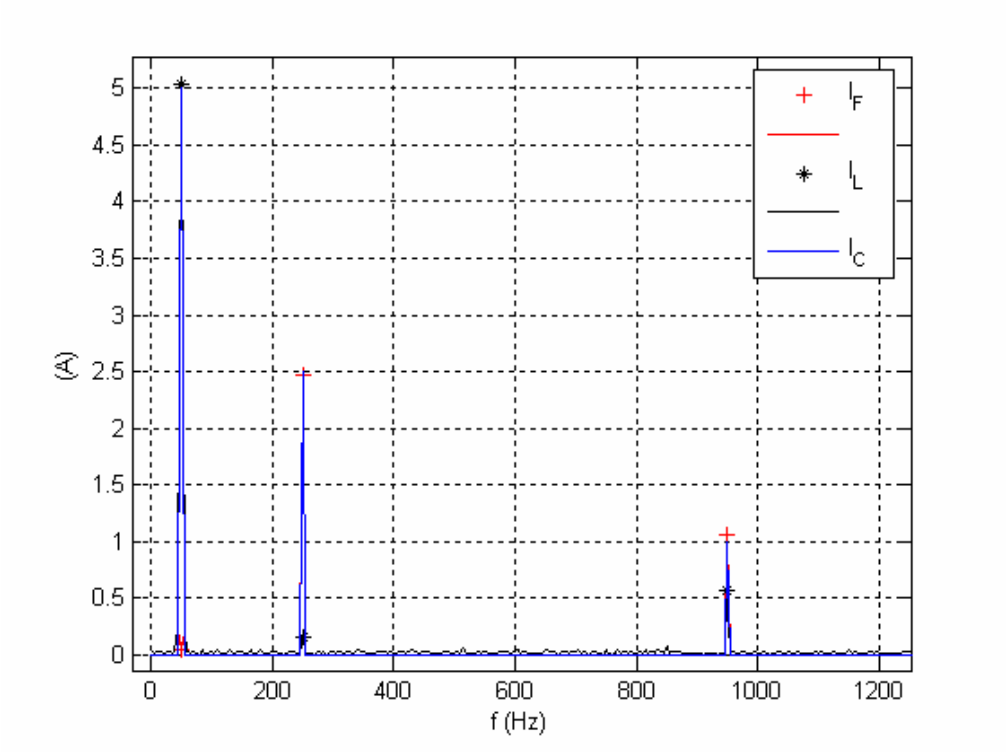


Figura 4.6.7

4.7. Conclusiones

En esta sección se observaron las simulaciones de distintos aspectos del sistema que luego se implementaría en el DSP. Se distinguieron claramente sub-bloques, los cuales fueron ensayados teniendo en cuenta distintas variables. Estas variables pueden ser clasificadas en: externas y propias. Las variables externas en su mayoría están dadas y no son modificables, por lo que se debió ajustar nuestro sistema (mediante las variables propias) para tener la mejor respuesta posible. Fue así que se llegaron a acotar ciertos grados de libertad que inicialmente se tuvieron, más allá de que ciertos criterios ya estaban fijados de antemano por el tutor.

Mediante las simulaciones, se dieron pautas para los siguientes parámetros:

- frecuencia de muestreo del filtro (fs)
- constantes del controlador PI para el control en la tensión del condensador del VSI (Kp y Ki)
- tensión en el condensador del VSI (E)
- frecuencia del bloque de manejo del VSI (f_{VSI})

Hay otros parámetros que ya habían sido optimizados en otras investigaciones (especialmente por Casaravilla o grupos a su cargo), por ejemplo, el ancho de los filtros IIR, parámetros del control vectorial del VSI, valores del condensador y de las inductancias, entre otros.

Es claro que para llegar a los valores mencionados se tuvo que estudiar y/o desarrollar distintos algoritmos. Es así que el grupo adquirió experiencia en el tema, lo cual fue fundamental a la hora de la implementación en el DSP. Hay que notar que el sistema de potencia que se utilizó en la última etapa no fue exactamente el que se simuló, por lo que finalmente algunos parámetros debieron ser cambiados.

5. Hardware in the loop

5.1. Introducción

El objetivo de esta parte del proyecto fue diseñar un sistema para simular filtros activos de una topología y método de cálculo dado. Se utilizó Simulink para la simulación del sistema eléctrico y el DSP TMS320F2812 para las operaciones del filtro. En el anexo C se trata el kit DSP utilizado.

En general el DSP realiza todas las operaciones necesarias para implementar el filtro, esto implica:

- PLL
- Cálculo de corrientes del filtro
- Control del inversor
- Control de la tensión en el condensador del inversor.
- Comunicación RS232

En Simulink se implementan los modelos de:

- La red eléctrica
- La carga no lineal
- El inversor
- Las impedancias de conexión
- Comunicación RS232

Las señales necesarias para la implementación del filtro desde el punto de vista del DSP son:

- Tensiones de la red eléctrica en el punto de conexión. Es utilizada por el PLL para sincronizarse a la frecuencia de la red.
- Corrientes de las tres fases de la carga. A partir de la cual se calcula la corriente del filtro según qué armónicos y cuánto se quieran filtrar.
- Corrientes de las tres fases del filtro. Es la realimentación necesaria para el control del inversor.
- Tensión en bornes del condensador del inversor. Es la realimentación para el control de la misma.
- Estado de las llaves. Es con lo que el filtro controla la corriente que está suministrando el inversor.

En la figura 5.1.1 se muestra un esquema general del sistema diseñado.

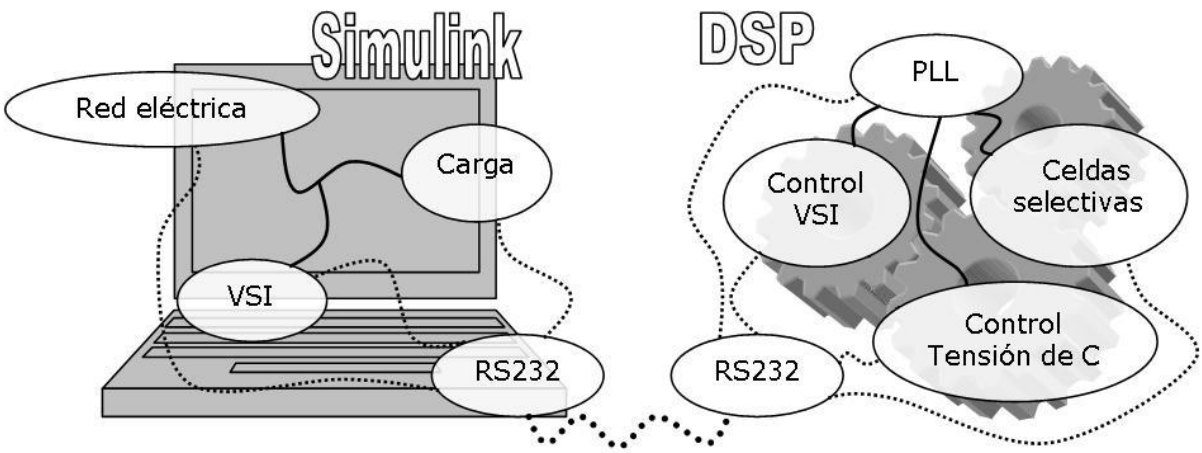


Figura 5.1.1 – Sistema a diseñar.

5.2. Interfaz Simulink-DSP

5.2.1. Software

El primer “hardware-in-the-loop” [1] realizado en este proyecto fue la simulación en Simulink de un inversor trifásico de tensión con fuente de continua ideal alimentando una carga pasiva. En este caso, el DSP controla las llaves del VSI, mediante ondas cuadradas, y “lee” variables desde la simulación, todo a través del puerto serie. Este primer “hardware-in-the-loop” se explica en el anexo A. Más allá de que luego se depuraron algunos aspectos, la técnica utilizada aquí fue la que se realizó luego para probar los distintos bloques del filtro y luego todo el filtro en sus conjunto.

Aunque es cierto que se realizó un protocolo para cada prueba de cada bloque implementado en el DSP, explicaremos a continuación la secuencia en la interacción en la última versión del filtro activo hasta el momento.

La simulación en Simulink consta básicamente de tres bloques, tal como se muestra en la figura 5.2.1.: un llamado a una función de Matlab de lectura de las posiciones de la llaves calculadas por DSP (leer_DSP), un bloque donde se simula el circuito eléctrico y finalmente otro llamado a una función Matlab para la escritura de las señales al DSP (escribir_DSP). En el anexo A se explica detalladamente del funcionamiento de este método para interactuar con el DSP.

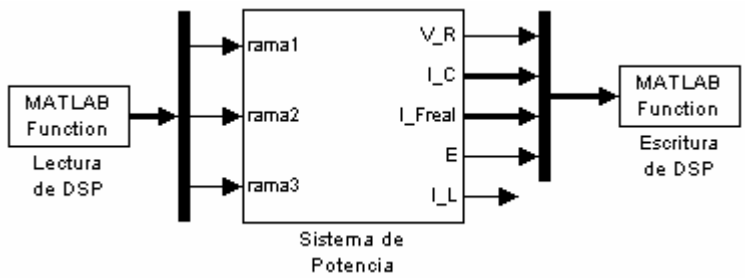


Figura 5.2.1 – Esquema de bloques implementados en Simulink

En el otro extremo, el DSP, quien recibe todas las señales para luego hacer los cálculos que determinarán la posiciones de las llaves a enviar.

En la figura 5.2.2 se muestra la interacción entre el Simulink y el DSP para el “hardware-in-the-loop” del filtro activo. En dicho esquema se observa cómo el Simulink envía las señales de tensiones y corrientes necesarias para el filtrado, luego el DSP hace las operaciones y responde con un byte con el estado de las llaves. Todas las señales que envía el Simulink, se envían en formato de punto flotante “single precision” (32 bits). El DSP luego se encarga de traducir este formato de datos al formato IQ del DSP.

Aquí la frecuencia de muestreo la fija el Simulink. El DSP solo espera recibir todos los datos para después responder con el estado de las llaves y volver a la inactividad hasta tener nuevamente todos los datos.

Lo que se buscó es evitar la necesidad de realizar muchas modificaciones en el código ya implementado en el DSP para poder interactuar con el VSI real. Una diferencia será que la frecuencia de muestreo será dada por un timer. Por otro lado, las señales entrantes y salientes no se manejarán a través del puerto serie sino a través de: 1) puertos de E/S digital para las posiciones de las llaves y 2) entradas analógicas para las corrientes y tensiones a leer.

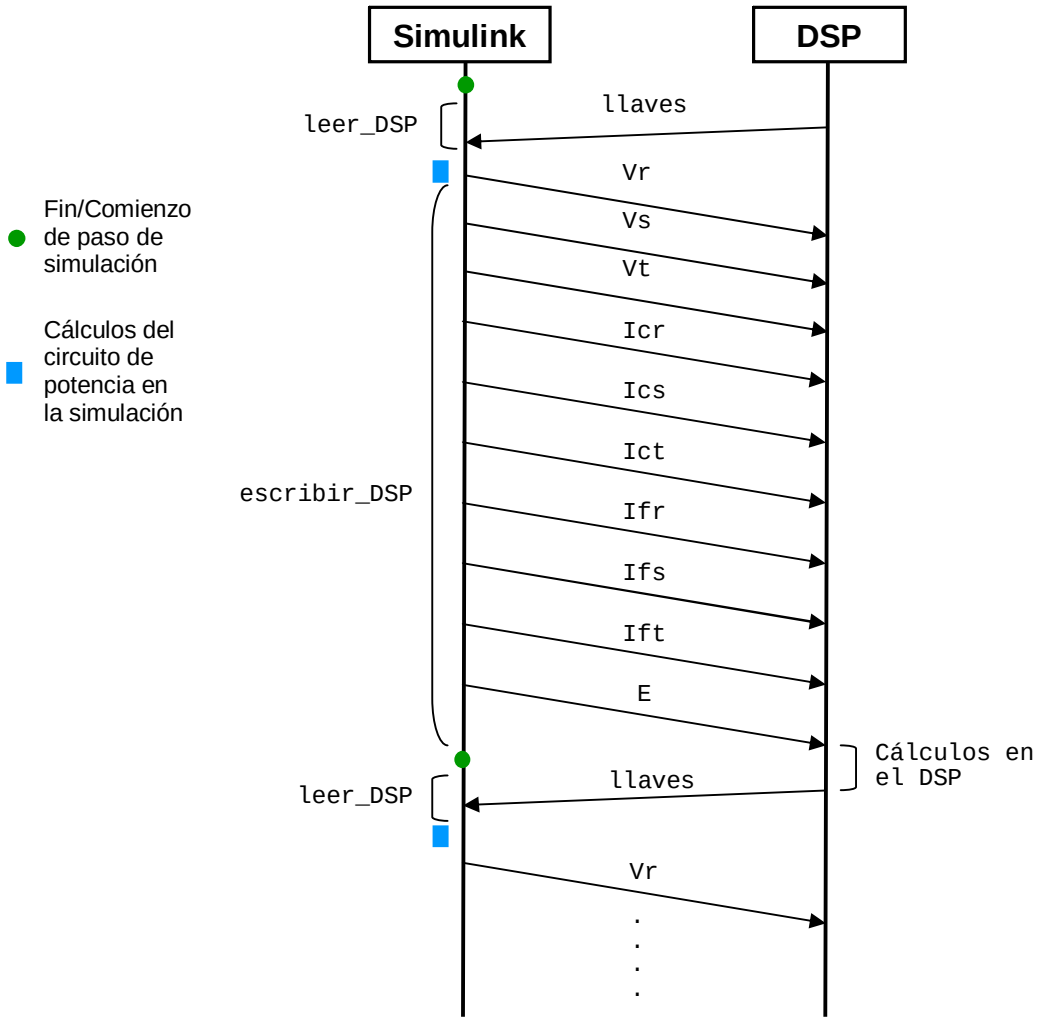


Figura 5.2.2 – Interacción Simulink-DSP (filtro activo completo)

5.2.2. Hardware

Para lograr la conexión entre el kit DSP y el puerto serie de la PC fue necesario implementar una tarjeta para la adaptación de niveles de tensión. Esta tarjeta además debía tener los conectores necesarios: DB9 hembra para conectar a la PC y conectores para los pines en el kit. El integrado utilizado para la adaptación de niveles fue el MAX232 [6]. En el anexo F se da una descripción algo más detallada de esta placa.

5.3 Bloques implementados en el DSP

5.3.1. Consideraciones generales

El código del DSP se realizó por completo en lenguaje C. Para realizar las operaciones matemáticas de punto fijo se utilizó la librería IQmath, suministrada por Texas Instruments [2]. Esta librería permite realizar todas las operaciones necesarias para implementar cualquier tipo de sistema de procesamiento de señales. Fue de gran utilidad, pero no fue inmediato su uso. Un ejemplo de ello fue que se necesitó configurar la memoria (mediante el archivo de asignación de memoria, extensión cmd del proyecto en CCS) de una forma dada, para poder cargar correctamente las tablas de la librería (como la tabla para las funciones seno y coseno).

5.3.2. Clarke

Las trasformadas de Clarke se implementaron en funciones independientes, una para la transformada y otra para la antitransformada. Se utiliza la transformada al principio del filtrado para pasar de tres coordenadas a dos, y la antitransformada se utiliza al final para pasar nuevamente a tres coordenadas.

En general, para el filtrado se utilizan variables globales, para así lograr una buena abstracción de la celda selectiva. Con esta técnica se logran velocidades de ejecución considerablemente mayores.

Para la transformada de Clarke, las entradas son las tres corrientes de carga, y las salidas son las corrientes i_α e i_β. Estas variables están definidas como globales, por lo cual lo único que hace la función Clarke, es multiplicar el vector de corrientes de entrada por la matriz de transformación (ecuación 5.1) y así obtener los valores de i_α e i_β [7].

$$\begin{bmatrix} i_o \\ i_\alpha \\ i_\beta \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} i_R \\ i_S \\ i_T \end{bmatrix} \tag{5.1}$$

La corriente homopolar i_0 es ignorada.

Como ejemplo, en este caso el código utilizado es:

```
Ialfa1 = _IQ21mpy(sqrt2div3,Ir1) + _IQ21mpy(menos_inv_sqrt6, (Is1 +
It1));
Ibeta1 = _IQ21mpy(inv_sqrt2, (Is1 - It1));
```

donde

`Ir1, Is1, It1:` corrientes de carga muestreadas
`Ialfa1 e Ibeta1 :` variables de entrada de la celda selectiva
`_IQ21mpy :` función IQmath [2] que multiplica dos variables de tipo IQ21
`sqrt2div3, menos_inv_sqrt6 inv_sqrt2 :` constantes de tipo IQ21

Para la antitransformada, es todo muy similar, a diferencia que aquí las entradas son las variables i_α e i_β y las salidas son las tres corrientes del filtro. La corriente i_0 es considerada nula.

$$\begin{bmatrix} i_R \\ i_S \\ i_T \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \frac{1}{\sqrt{2}} & 1 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{2} & \frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} i_o \\ i_\alpha \\ i_\beta \end{bmatrix} \tag{5.2}$$

Los coeficientes de las matrices son declarados como constantes y se utilizan las mismas para las dos funciones.

5.3.3. PLL

En un principio se había utilizado un PLL que lo único que hacia era mirar los cruces por cero para determinar la fase de la tensión R del punto de conexión. Como ya fue referido, finalmente se utilizó un PLL mucho más robusto, el SSI-PLL, descrito en la sección 4.4 de este documento.

El PLL esta desarrollado en una única función, llamada *void fpll(void)*. Se puede dividir en tres partes: filtro SSI, transformada [T] y control PI, y por ultimo saturación del controlador y formación del diente de sierra.

1) Filtro SSI

Transformada de Clarke de las tensiones:

```
valfa = _IQmpy(vr,sqrtDeDosTercios) + IQmpy(vs,menos_inv_sqrtDeSeis) +
_IQmpy(vt,menos_inv_sqrtDeSeis);
```

`vr, vs y vt:` Son las tensiones de la red en el punto de conexión.
`sqrtDeDosTercios, menos_inv_sqrtDeSeis`: Son coeficientes de la matriz de transformación de Clarke, ver ecuación 5.1.
`valfa` es el resultado de la transformada. Aquí no se usa la componente beta.

Filtro SSI propiamente dicho:

```
suma1=_IQmpy(_IQ(1000),valfa-valfafiltrada) + IQmpy(vbeta,masW);

valfafiltrada +=_IQmpy(_IQmpy(suma1ant+suma1,tsamp),_IQ(0.5));
```

`valfafiltrada, vbeta`: Son las salidas del filtro SSI.

```
suma1ant = suma1;

int2 += _IQmpy(_IQmpy(valfafiltrada+valfafiltradaant,tsampMenosW),
_IQ(0.5));

vbeta = int2;

valfafiltradaant = valfafiltrada;
```

`Int2` es un de las integrales que utiliza el filtro SSI, en particular aquí hubo que introducir una nueva constante, `tsampMenosW`, el motivo de esta es reducir el ruido numérico que genera multiplicar por un numero pequeño como lo es `tsamp.`

2) Transformada [T] y control PI

```
xpi = _IQmpy(valfafiltrada,_IQcos(int4)) - _IQmpy(vbeta,_IQsin(int4));
```

`xpi` es la entrada del controlador PI, y es determinada utilizando la transformada [T], que se puede ver en la ecuación 4.1. Aquí `int4` es lo que sería θ en la ecuación 4.1.

```
int3 += _IQmpy(_IQmpy(xpi+xpiant,tsampKi),_IQ(0.5));

xpiant = xpi;
```

`int3` es la integral para la parte integral del controlador PI. Aquí `tsampKi` se define por el mismo motivo que `tsampMenosW`. Solo que en este caso se tenía el problema que `Ki`, era demasiado grande, la forma de contrarrestar este problema fue multiplicar por una constate chica como lo es `tsamp.`

```
ypi = _IQmpy(Kp_pll,xpi) + int3;
```

`ypi` es la salida del controlador PI. `Kp_pll` es la constante proporcional del PI.

3) Saturación del controlador y formación del diente de sierra

```
if (ypi>_IQ(500)){
      ypi = _IQ(500);
}
if (ypi<_IQ(-100)){
      ypi = _IQ(-100);
}
```

Dado que el tiempo de ejecución del hardware in the loop resultaron muy grandes, se optó por imponer un máximo para la salida del controlador el valor 500 y como mínimo -100. Con estos valores enormes se podía llegar rápidamente al régimen, a costas de un pico de corriente muy elevado en el arranque. En el control del VSI real estos valores fueron de alrededor de 3 y -3. Esto no implica un arranque aprox. 100 veces más lento, puesto que, como se verá más adelante, el sistema de potencia finalmente fue distinto al simulado (condensador más grande, filtro conectado a la red a través de un transformador). Es decir, debieron ser hallados nuevamente parámetros aceptables para el control de la tensión en el condensador.

```
int4 += _IQmpy(_IQmpy(ypi+ypiant,tsamp),_IQ(0.5));
ypiant = ypi;
```

`int4` es la fase que genera el PLL. Esta es la fase de la salida del filtro SSI.

```
if(int4>=_IQ(6.28318530717959)){ // 6.28318530717959 = 2π
      int4 = _IQ(0);
      ypiant = _IQ(0);
}
```

`int4` se baja a 0 cuando llega a 2π, es así que se genera el diente de sierra entre 0 y 2π. Esta variable se utiliza en la modulación y demodulación en las celdas selectivas.

En la implementación de este PLL se tuvo problemas numéricos. El filtro SSI utiliza constantes muy altas al igual que el controlador PI. Las medidas tomadas para solucionarlos fueron:

- Reducir la amplitud de las tensiones de entrada, de aproximadamente 220 a 2 como valor de pico.
- Jugar con los algoritmos de modo tal que los productos entre las constantes se simplificaran a una sola constante. Es el caso de `tsampKi` y `tsampMenosW`. `tsamp` es del orden de los microsegundos, `Ki` esta en el entrono de 10^4 y `MenosW` es $-2\pi 50$, o sea del orden de 600.
- Cambiar el formato de algunas variables IQ21 a IQN con N menor a 21 (N global). De esta manera se pueden manejar números con mayor valor absoluto [2].

Este tipo de medida solucionaron por completo todos los problemas numéricos.

5.3.4. Celdas Selectivas

La celda selectiva se trató de hacer lo mas genérica posible, para así independizarnos del armónico que se quiera filtrar. Para esto se implementó un arreglo en memoria (mediante una estructura) que es el que va a contener los datos necesarios para filtrar una secuencia armónica. Estos datos son:

- Numero de armónico.
- Ganancia del filtrado.
- Estados de los filtros IIR de *p* y *q* [7].

Las operaciones que implementa esta celda son:

1. Modulación de las corrientes i_α e i_β, para obtener *p* y *q*.
2. Filtrado IIR de *p* y *q*.
3. Demodulación de *p* y *q*, para obtener las nuevas i_α e i_β.

Ampliamos cada punto a continuación:

1. Modulación de las corrientes i_α e i_β

Las potencias activa y reactiva instantáneas [7] se hayan según la ecuación 5.3:

$$\begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} sin(\omega_c t) & -cos(\omega_c t) \\ -cos(\omega_c t) & -sin(\omega_c t) \end{pmatrix} \begin{pmatrix} i_\alpha \\ i_\beta \end{pmatrix} \quad (5.3)$$

donde $\omega_c = n\omega$, siendo ω la velocidad angular de la red y n el valor de la secuencia, ya sea positiva o negativa.

La biblioteca IQmath tiene implementadas las funciones seno y coseno. Estas funciones utilizan una tabla en ROM que contiene los valores de un período y medio de una sinusoide, muestreada a 512 muestras por ciclo.

Veamos como queda el código para la modulación:

```
sin = _IQ21sinPU(nwt);
cos = _IQ21cosPU(nwt);
p =  _IQ21mpy(sin,Ialfa1) - _IQ21mpy(cos,Ibeta1);
q = -_IQ21mpy(sin,Ibeta1) - _IQ21mpy(cos,Ialfa1);
```

donde vale aclarar

`nwt` : fase de la secuencia a filtrar ($\omega_c t$) en por unidad

`_IQ21sinPU` : función seno de IQmath en por unidad [2]

`_IQ21cosPU` : función coseno de IQmath en por unidad [2]

2. Filtrado IIR de *p* y *q*.

El filtro IIR es simplemente la ecuación 5.4:

$$A(1)y(n) = B(1)\cdot x(n) + B(2)\cdot x(n-1) + B(3)\cdot x(n-2) - A(2)\cdot y(n-1) - A(3)\cdot y(n-2) \quad (5.4)$$

donde x e y son las señales de entrada y salida del filtro, mientras que los parámetros *A(i)* y *B(i)* son constantes.

El filtro utilizado en las celdas selectivas es un filtro pasa-bajos. Sin embargo, los parámetros *A(i)* y *B(i)* se hallan en Matlab de forma de obtener un pasa-altos butterworth de segundo orden. Luego, en la ejecución del DSP, se le resta el resultado del filtro a la variable de entrada del filtro, como muestra la figura 4.C.vi.1, para obtener finalmente el filtro pasa-altos (figura 5.3.1).

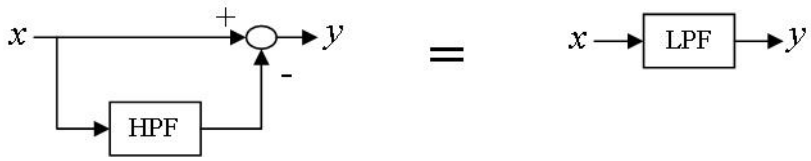


Figura 5.3.1 – Filtro pasa bajos.

El motivo de este algoritmo es la mejora de fase que proporciona, dado que el filtro pasa bajos directo produce un desfasaje que dificulta el filtrado selectivo [14].

En la versión que presentamos para esta parte del proyecto se utilizan los mismos coeficientes para todas las secuencias armónicas. El hecho de que estos coeficientes sean enviados desde el Matlab al DSP en la inicialización hace que estos se puedan cambiar fácilmente desde el PC.

3. Demodulación de *p* y *q*

En este caso se modificó la demodulación presentada en [7], introduciéndole una nueva variable: φ. Esta es configurada al inicio del programa y su finalidad es el ajuste de la fase en la demodulación. Con un valor apropiado de φ se puede mejorar mucho el filtrado, porque se puede lograr que la corriente por el filtro tenga la fase que realmente se necesita para lograr el objetivo de filtrado, especialmente al compensar los retardos introducidos por los filtros IIR y por el control de corriente. La ecuación modificada es entonces es la ec. 5.5.

$$\begin{pmatrix} \tilde{i}_\alpha \\ \tilde{i}_\beta \end{pmatrix} = \begin{pmatrix} sin(n(\omega t+\varphi)) & -cos(n(\omega t+\varphi)) \\ -cos(n(\omega t+\varphi)) & -sin(n(\omega t+\varphi)) \end{pmatrix} \begin{pmatrix} \tilde{p} \\ \tilde{q} \end{pmatrix} \quad (5.5)$$

Se observó con simulaciones que con un único φ se podía mejorar en buena manera el filtrado de todas las celdas selectivas. Sin embargo para la versión final, en la que se manejó el VSI real, se asignó un desfasaje para cada una de las celdas.

Todas las operaciones se realizan en el cuerpo de la función de la celda, para así reducir el tiempo de ejecución, dado que de lo contrario se necesitarían operaciones innecesarias para el pasaje de parámetros.

5.3.6. Control de tensión en el condensador

La tensión en el condensador, a la cual llamamos *E*, se controla mediante el control de la potencia (continua) que entra al filtro. Esto se traduce en la variación del módulo de la fundamental de la corriente del filtro en fase con la tensión de la red.

Ya fue visto en secciones anteriores que, como se explica en la tesis de Gonzalo Casaravilla [7], la corriente de referencia que sale del bloque de filtrado selectivo puede tener algo de fundamental. Esta la anulamos de la manera propuesta en dicha fuente, mediante el uso de un filtrado residual, solo que nosotros le sumamos al canal *p* [7], a la salida del filtro IIR, una continua para controlar la potencia que entra al filtro. El valor de esta continua la obtenemos mediante un control PI. En régimen debe ser igual a las perdidas del dispositivo (llaves, condensador e inductancias) para que no haya una variación en la tensión del condensador.

La implementación del bloque de filtrado residual es análoga al del filtrado selectivo. Más precisamente, éste consiste en una celda residual para filtrar la secuencia -1 en cascada con otra correspondiente a la secuencia +1. Estas celdas difieren de la selectivas en que el filtro IIR es un pasa-altos. Luego está el agregado del controlador PI que se suma al canal *p* de la celda "+1".

El control PI está implementado en C de la siguiente manera:

```
aux = xPI;        // se guarda valor anterior de xPI
xPI = Eref - E;   // se calcula nuevo valor de xPI
yPI = yPI + _IQ21mpy(PIa1,xPI) + _IQ21mpy(PIa2,aux); // Control PI
```

donde *PIa1* e *PIa2* son constantes cargadas desde Matlab en la inicialización. Estas constantes se calculan en dicho programa a partir de las constantes de un controlador PI: K_P y K_I. La relación entre los dos pares de constantes se presenta en las ecuaciones 5.6 y 5.7:

$$\mathrm{PIa1} = \mathrm{K_P} + \frac{\mathrm{K_I}}{2\mathrm{fs}} \quad (5.6)$$

$$\mathrm{PIa1} = \mathrm{K_P} - \frac{\mathrm{K_I}}{2\mathrm{fs}} \quad (5.7)$$

donde *fs* es la frecuencia nominal de muestreo del filtro. En el hardware in the loop se utilizó una *fs* de 25,6 kHz, pero en el control del VSI real esta fue de 12,8 kHz.

En la sección que trató las simulaciones se realizó un análisis para hallar los valores óptimos de K_P y K_I para una situación dada (en particular, *E* dado).

Otra diferencia con en filtrado selectivo, es que no realizamos un ajuste de fase en la demodulación de las celdas residuales. Más adelante se mostrará que en el control del VSI real se decidió también tener un ajuste de la fase que halla el PLL.

5.3.7. Control del VSI

La idea de esta parte es generar las corrientes de fase halladas en la etapa de filtrado. Para ello se conmutarán las llaves del inversor de forma tal que las corrientes que tome este

sean igual o sigan muy de cerca a las corrientes de referencia (las halladas en el filtrado). En la tabla 5.1 se muestran todas las combinaciones de llaves del VSI.

V(k)	k=0	k=1	k=2	k=3	k=4	k=5	k=6	k=7
(Su,Sv,Sw)	000	100	110	010	011	001	101	111
Llaves	0	4	6	2	3	1	5	7

Tabla 5.1 – Combinaciones de llaves.

Donde:

-V(k) es el estado de las llaves.

-Su, Sv, Sw son las llaves de la rama R, S y T respectivamente.

-Llaves es el numero entero resultante de considerar en binario el vector (Su, Sv, Sw).

Esta parte se basa en la teoría de Control Vectorial (estudiada por Biardo, Giacosa, Rivoir [3]). Ella se basa en medir las corrientes de fase, luego calcular un vector error de corriente *delta_i* (como la diferencia entre las corrientes de fase actuales y las de referencia) y optar entre 3 casos posibles de acción: una respuesta rápida, una de minimización de armónicos y otra de inacción. Para esta elección es necesario conocer el estado actual, las corrientes actuales que está tomando el filtro y las corrientes que se desea que tome. El vector error de corriente *delta_i* es hallado como el mayor valor absoluto de la diferencia entre las corrientes de fase y las de referencia. A este valor se lo compara con δ y *h*. Estos son los límites entre las 3 zonas antes descriptas:

1) $|\Delta l| = \delta$ Inacción

2) $\delta < |\Delta l| < h$ Reducción de armónicos

3) $|\Delta l| > h$ Respuesta rápida

En el caso 1) no se realiza ninguna conmutación entre las llaves.

Una mejora que se realizó a los algoritmos presentados en [3] es la eliminación de la variable intermedia V(k). En esta nueva implementación se trabaja directamente con las posiciones de las llaves, logrando un cálculo más directo, es decir más rápido, lo cual tiene gran prioridad en este proyecto.

En el caso de la respuesta rápida se observa en que zona se encuentra el vector *delta_i* y se actúa inmediatamente.

Esto se puede ver a continuación

```
llaves=zonasrr[IQ21sign(Delta_Ir)][IQ21sign(Delta_Is)][IQ21sign(Delta_
It)];
```

Donde *zonasrr* es una tabla (de 2x2x2) que contiene la nueva posición de las llaves y tiene como elementos directamente el signo del vector error. La función *sign(a)* devuelve 1 si *a*>0 y 0 en otro caso.

La tabla que resulta de la eliminación de V(k) se implementó de la siguiente manera:

```
const Uint16 zonasrr[2][2][2] = { { {0,1},{2,3} } , { {4,5},{6,7} } };
```

Por ejemplo:

```
zonasrr[1][0][0]=4;
```

En el caso de reducción de armónicos se deben tener en cuenta más parámetros. En primer lugar debemos calcular el signo de la derivada del error de corrientes, para ello se necesita el anterior valor del vector error, por ejemplo el signo de ΔIx':

```
signo_delta_i_x_prima = IQ21sign(Delta_Ir-Delta_Ir_anterior);
```

Entonces, con los tres signos de las derivadas y el estado anterior de las llaves, obtenemos un vector *e* que es la tensión que deberíamos tener en bornes del inversor para obtener la corriente deseada.

Los autores de este proyecto consideraron que no se estaba prestando atención a un aspecto importante: en el caso de tener que ir al estado nulo (0 o 7) se debe tener en cuenta a cuál de los dos conviene ir, considerando la menor conmutación de llaves posibles. Para ello se generó un nuevo *e*, que difiere del *e* utilizado en la referencia [3] según se indica en la tabla 5.2.

Llaves	**2 o más en 0**						**2 o más en 1**					
e clásico	1	2	3	4	5	6	1	2	3	4	5	6
Nuevo e	0	1	2	3	4	5	6	7	8	9	10	11

Tabla 5.2 – Combinaciones de llaves.

El nuevo valor de *e* será un índice de un array en C, por lo tanto es mejor que el primer valor sea 0. Habiendo realizado esa aclaración se construye una nueva tabla 3, que se implementó de la siguiente manera:

```
const Uint16 Tabla3[8][2][2][2] = {
{ { { 0,10},{ 8, 9} } , { { 6,11},{ 7, 0} } },
{ { { 1, 1},{ 2, 2} } , { { 1, 1},{ 2, 2} } },
{ { { 3, 3},{ 3, 3} } , { { 4, 4},{ 4, 4} } },
{ { { 8, 9},{ 8, 9} } , { { 8, 9},{ 8, 9} } },
{ { { 5, 0},{ 5, 0} } , { { 5, 0},{ 5, 0} } },
{ { {10,10},{10,10} } , { {11,11},{11,11} } },
{ { { 6, 6},{ 7, 7} } , { { 6, 6},{ 7, 7} } },
{ { { 0, 1},{ 3, 2} } , { { 5, 0},{ 4, 0} } },
};
```

donde

```
e=tabla3[llaves][signo_delta_i_x_prima][signo_delta_i_y_prima][signo_d
elta_i_z_prima];
```

Por ejemplo:

```
Tabla3[7][0][0][1]=1;
Tabla3[7][0][1][1]=2;
```

No deberían suceder los casos

```
Tabla3[0 o 7][0][0][0]
Tabla3[0 o 7][1][1][1]
```

Recordar que este valor de *e* ya es el modificado.

Ahora con este valor de *e* y los signos del vector de corriente hallamos el nuevo estado de las llaves. Para tal fin se utiliza la tabla 4, la cuya implementación se presenta a continuación.

```
llaves=
    tabla4[e][IQ21sign(Delta_Ir)][IQ21sign(Delta_Is)][IQ21sign(Delta_I
t)];


const Uint16 Tabla4[12][2][2][2] = {
```

```
{ { {0,5},{0,0} } , { {4,5},{4,0} } },
{ { {0,1},{0,1} } , { {5,5},{0,0} } },
{ { {0,1},{3,3} } , { {0,1},{0,0} } },
{ { {0,3},{2,3} } , { {0,0},{2,0} } },
{ { {0,0},{6,0} } , { {4,4},{6,0} } },
{ { {0,0},{6,0} } , { {4,4},{6,0} } },
{ { {0,7},{6,7} } , { {4,4},{6,0} } },
{ { {0,7},{2,2} } , { {6,7},{2,0} } },
{ { {0,3},{2,3} } , { {7,7},{2,0} } },
{ { {0,1},{3,3} } , { {7,1},{7,0} } },
{ { {0,1},{7,1} } , { {5,5},{7,0} } },
{ { {0,5},{7,7} } , { {4,5},{4,0} } },
};
```

Por ejemplo:

```
Tabla4[0][0][0][1]=6;
Tabla4[0][0][1][0]=0;
```

No deberían suceder los casos

```
Tabla4[x][0][0][0]
Tabla4[x][1][1][1]
```

Antes de retornar al programa principal debemos dejar todo pronto para la próxima invocación. Esto es adjudicar al vector error anterior el nuevo valor del vector error:

```
Delta_Ir_anterior=Delta_Ir;
```

5.3.7. Sistema general del DSP

En el apartado 5.2.1 de este mismo documento, está detallado el protocolo que se sigue para llevar adelante la comunicación entre Simulink y el DSP. A continuación se describirá la secuencia de los algoritmos implementados en el DSP.

El DSP trabaja en un loop infinito en el cual espera los datos, ejecuta todas las funciones, y envía los resultados. Las funciones antes mencionadas son PLL, transformada de Clarke, celdas selectivas, filtrado residual, control de tensión en el condensador, antitransformada de Clarke, y por último, control vectorial de las corrientes del filtro.

En la inicialización se actualizan las variables generales de la simulación, como son:

- Inicialización del puerto serie.

Desde Simulink se inicializa:

- Frecuencia de simulación.
- Cantidad de secuencias armónicas.
- Secuencias armónicas a filtrar y ganancias.
- Ajuste de fase genérico.
- Coeficientes del filtro IIR.
- Tensión de referencia para el control de la tensión del condensador del VSI.
- Coeficientes para el controlador de la tensión del condensador del VSI.
- Coeficientes para el control vectorial.

5.4. Resultados

5.4.1. Tiempos

Utilizando el osciloscopio, y mediante el uso de un puerto de entrada salida del DSP [5], se midieron los tiempos de ejecución de los distintos bloques al ejecutar el hardware-in-the-loop. Ver tablas 5.3 y 5.4.

Bloque	Tiempo (μs)
PLL	Min: 0,30 – Max: 0,36
Clarke	0,36
Residual	7,60
Anti-Clarke	0,30
Control Vectorial	Min: 0,96 Max: 1,40
Incrementar paso	0,10

Tabla 5.3 – Tiempos de ejecución de distintos bloques.

Cantidad de secuencias a filtrar	Tiempo Celdas Selectivas (μs)
0	0,17
1	3,20
2	6,20
3	9,20
8	24,50
9	27,50

Tabla 5.4 – Tiempos de bloque de filtrado selectivo para varios armónicos.

El objetivo en esta parte fue muestrear los datos y actuar en el VSI (simulado) a 512 muestras por ciclo de red. Esto significa muestrear a 25,6 kHz, o bien, hacer que el VSI funcione a lo sumo a 12,8 kHz. Ésta es la frecuencia a la que se podía llegar con el inversor disponible en la facultad. En definitiva, la suma de todos los retardos no debía llegar a 39 μs (finalmente se verá que la frecuencia utilizada con el sistema real fue la mitad de la esperada en esta parte; por falta de tiempo no se pudo intentar subir la frecuencia al doble).

En la tabla 5.3 se observan los retardos de los bloques que no dependen de cuántos armónicos se quieren filtrar. Algunos de ellos (PLL y Control Vectorial) normalmente varían entre un valor mínimo y un máximo. Para analizar cuantos armónicos se pueden filtrar actuando sobre el VSI a la frecuencia deseada debemos tener en cuenta los valores máximos.

Por otro lado, en la tabla 5.4 vemos como varía el tiempo de demora del bloque de filtrado selectivo en función de las cantidad de secuencias armónicas que se desean filtrar (con la ganancia deseada). Se deduce de dicha tabla que el tiempo de este bloque es aproximadamente:

$$t_{sel} = Arm \times 3{,}0\mu s + 0{,}2\mu s$$

donde Arm es la cantidad de secuencias a filtrar.

Sumando los valores máximos de los otros bloques queda:

$$t_{total}^{max} = Arm \times 3{,}0\mu s + 10{,}6\mu s < 39\mu s$$

resultando que podemos filtrar hasta 9 secuencias armónicas con los programas realizados.

Esta aseveración se puede verificar sumando los valores de la tabla 5.3 y el último valor de la tabla 5.4. Más aún, el retardo total medido de todos los bloques filtrando 9 armónicos resultó ser $t_{total}^{9arm} = 37{,}9\mu s < 39\mu s$. Es así que resulta un tiempo de inactividad de $1{,}7\mu s$**.** No hay que olvidar que para el caso "real" faltan los tiempos de lectura y escritura de puertos, por lo que tenemos que agregar unas decenas de ciclos de reloj más en el cálculo. Teniendo en cuenta que el tiempo que le toma a los ADC tomar una muestra es de $80ns$, y que la cantidad de muestras que hay que tomar es 8, tenemos unos $640ns$ para tomar las muestras, y se va a necesitar aproximadamente 16 instrucciones, por lo cual el tiempo de toma de muestras va a rondar los $740ns$. Ahora, el DSP tiene 2 ADC que pueden trabajar en paralelo, lo cual reduce este tiempo a la mitad. En esta parte se concluyó entonces que el tiempo iba a ser suficiente para filtrar 9 secuencias.

Teniendo en cuenta los tiempos observados, se le hizo un ajuste más a la simulación, de modo que esta tenga en cuenta el retardo del DSP entre que recibe los datos y actúa. Esto

se logró introduciendo un bloque que retarda una cantidad entera de pasos de simulación la señal a su entrada. Esta señal, o señales, son las posiciones de las llaves que salen del bloque de lectura del DSP.

La cantidad de retardos de las llaves en la simulación depende de la frecuencia de simulación (*fsim*). No podemos subir mucho esta frecuencia, porque eso conllevaría a demasiados datos enviados y recibidos (específicamente *Vr* y *muestrear*) entre muestreo y muestreo a través de la comunicación serie, enlenteciendo la simulación de una manera insostenible. Es por eso que usamos una *fsim* de a lo sumo 102,4kHz que es igual a 4 veces la frecuencia de muestreo del filtro. La tabla 5.5 muestra la cantidad de retardos que más se aproxima a la realidad en función de la cantidad de armónicos a filtrar, para dos valores de *fsim*. Esta se basa en los tiempos máximos medidos con el osciloscopio.

Hay que observar que en esta parte no se tuvo en cuenta el tiempo de ejecución del código de protección del VSI y del código asociado a la comunicación serie con la interfaz gráfica. En la sección que trata el control del sistema real se mostrarán los resultados obtenidos.

Cantidad de armónicos	**Retardos para *fsim* = 102,4kHz**	**Retardos para *fsim* = 51,2kHz**
0	1	1
1	1	1
2	2	1
3	2	1
4	2	1
5	3	1
6	3	1
7	3	2
8	4	2
9	4	2

Tabla 5.5 – Retardos en las llaves para simular retardos reales del DSP

5.4.2. Simulaciones

Como ejemplo de las simulaciones realizadas, a continuación se presenta un caso particular, en el cual se tiene una carga con secuencias -5, 7 y -17. La frecuencia de muestreo del filtro fue de 25,6 kHz (512 muestras por período de red). El objetivo del filtrado en este caso fue eliminar las secuencias mencionadas al 100%.

En la figura 5.4.1 se pueden observar las corrientes obtenidas una vez llegado al régimen. Con esto último nos referimos en especial al régimen en la tensión del condensador, ya que ésta presenta un transitorio asociado al controlador PI (figura 5.4.4). Esto no significa que el comportamiento del filtro en este transitorio haya estado alejado del comportamiento en régimen.

El espectro de estas corrientes aparece en la figura 5.4.2. Se observa un buen filtrado de las secuencias correspondientes a armónicos de frecuencia baja, no siendo tan así para la secuencia -17. Como los valores de las corrientes I_F e I_L son iguales en módulo, se concluye que el problema está en el ajuste de fase. Este tema será tratado más adelante.

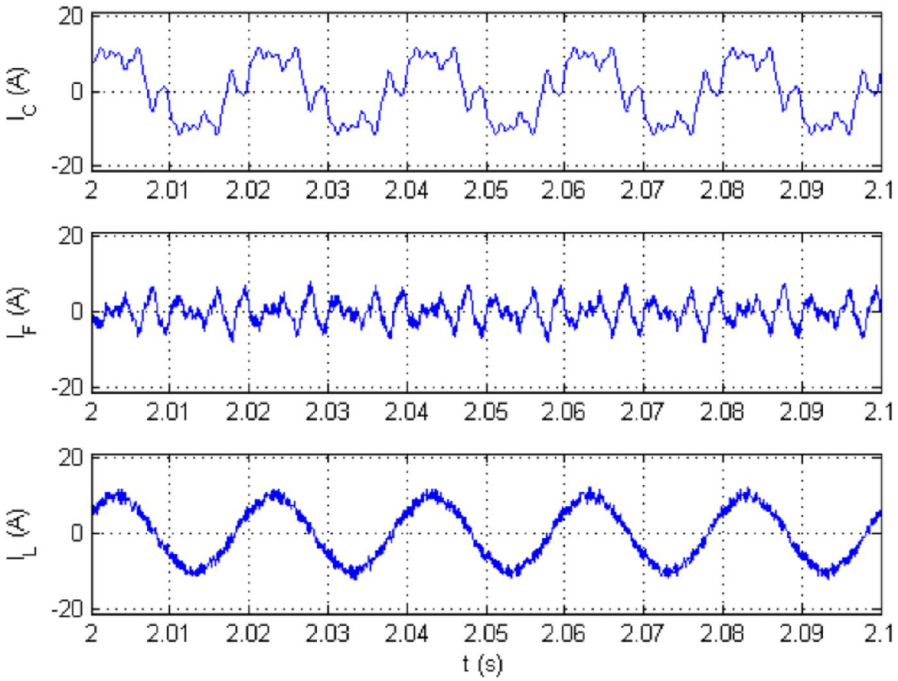


Figura 5.4.1 – Corrientes simuladas de la fase R en régimen

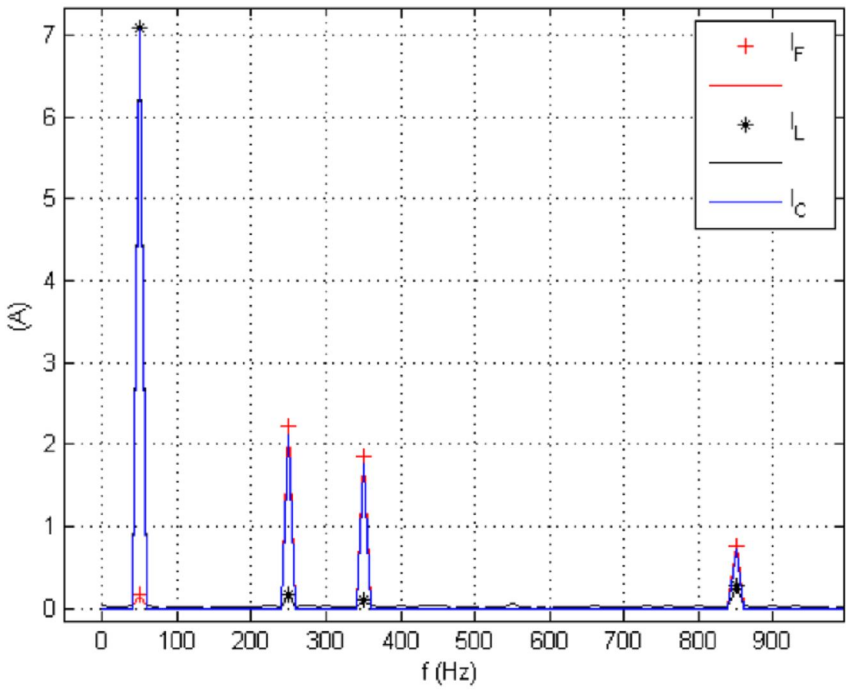


Figura 5.4.2 – Espectro de las corrientes simuladas

Como ya se vio en el ejemplo de la simulación del control vectorial de corriente, se observa que en el espectro de la corriente por el filtro no existe una componente de alta frecuencia debido al control del inversor. Lo que en cambio sí aparece es que en la banda de frecuencias de hasta 6kHz (aprox.) existe lo que se podría dar a llamar "ruido". Analizamos cuantitativamente esta situación.

La corriente eficaz por el filtro esperada sería la debida a los armónicos a filtrar: $I_F^{teo} = 2{,}14A$. Sin embargo, la corriente real es $I_F^{real} = 3{,}08A$, de la cual podemos separar una corriente eficaz debida a los armónicos que se desean filtrar y otra debida al ruido introducido

por la conmutación de las llaves del inversor. Estas son $I_F^{arm} = 2{,}38A$ y $I_F^{ruido} = 1{,}96A$ respectivamente.

En otras palabras, si bien los valores máximos de la corriente instantánea generada por el control de corriente son similares a las corrientes ideales, el ripple introducido se traduce en una corriente eficaz mayor a la estrictamente necesaria para el filtrado. Esto debe ser tenida en cuenta a la hora de diseñar un sistema de potencia de estas características, puesto que una corriente eficaz mayor implica pérdidas de energía mayores. Una frecuencia de conmutación mayor se traduce en un ripple menor, pero a la vez en pérdidas mayores en las llaves. Es así que encontrar la frecuencia óptima para tener pérdidas mínimas se vuelve una tarea difícil, que llevará a resultados diferentes dependiendo del control de corriente y de los componentes del sistema utilizados.

En cuanto al control de la tensión en el condensador del bus de continua, en la figura 5.4.4 se muestra el arranque del sistema. Se puede observar que se llegaba al régimen en aproximadamente 2 segundos. Sin embargo, se tuvo un comportamiento no deseado al comienzo: aparecía un pico en la corriente del filtro que podría dañar los equipos. Simulando varias veces, con distintas cargas (pero de mismos valores eficaces) se pudo apreciar que este comportamiento no siempre se presentaba. Esto fue luego analizado y corregido. Un defecto en la implementación del PLL que detectaba los cruces por cero fue el causante del problema.

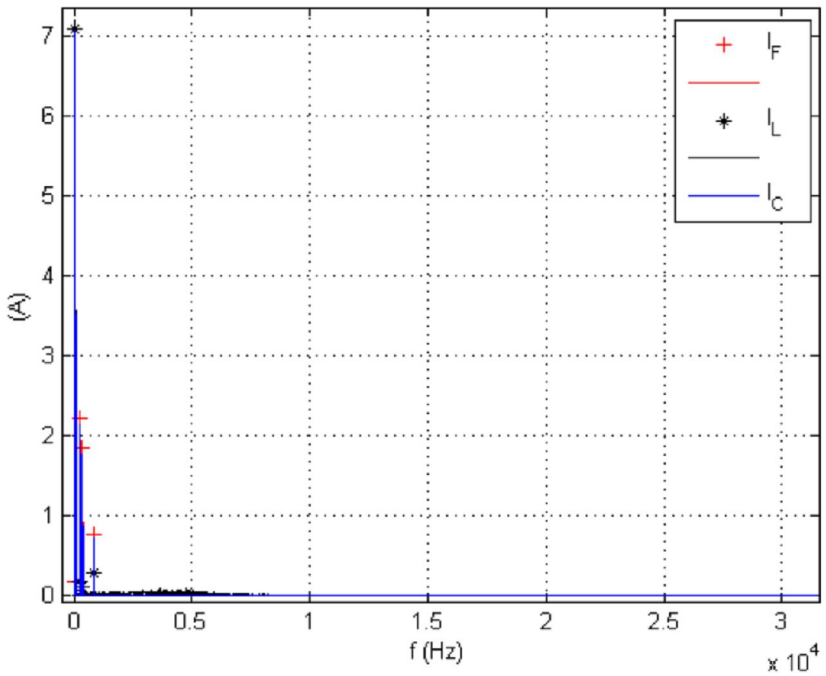


Figura 5.4.3 – Espectro de las corrientes simuladas

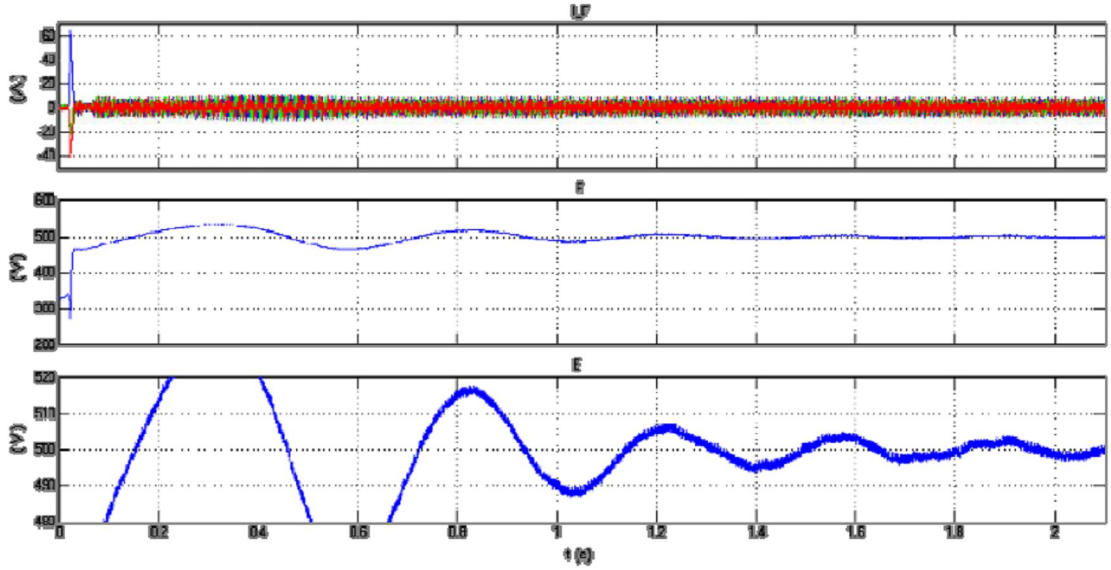


Figura 5.4.4 – Corrientes del filtro, tensión en le condensador y tensión en el condensador (ampliado)

5.5. Conclusiones

En la segunda etapa del proyecto logramos realizar todos los objetivos planteados al inicio. A partir de esto refinamos la planificación para la siguiente etapa, así como también incorporamos nuevas metas que pueden enriquecer el producto final.

Concluimos que el método de hardware-in-the-loop resultó muy útil en el desarrollo del firmware del DSP. Este tiene la virtud de ser una herramienta potente, siendo a la vez económica y segura. Como aspecto negativo resaltamos la lentitud en la ejecución, debido principalmente a la comunicación serie y al tiempo requerido por la PC para realizar las operaciones. Por lo tanto, aunque este método de trabajo permitió determinar el comportamiento del sistema, impidió el ajuste fino de los parámetros. De todos modos, para esto último se podía recurrir a las simulaciones del Simulink, en caso de ser inviable la prueba con el sistema real.

En cuanto al filtrado selectivo, observamos que este era bueno, pero podríamos realizar una optimización mediante un ajuste de la fase. No valió la pena realizar este ajuste fino dado que el sistema real iba a ser diferente la simulado (a ese nivel de detalle).

Con el estudio del espectro de las corrientes simuladas concluimos que se debe tener muy en cuenta que la conmutación de las llaves causa que las corrientes sean mayores a las necesarias. Esto depende del método de control utilizado, así como la frecuencia del inversor, entre otros factores.

Con respecto a los tiempos medidos en el osciloscopio, las noticias fueron buenas. De ellas concluimos que no sería necesaria una optimización en assembler de los programas realizados en C. También vimos que se podría estudiar el uso de un PLL más complejo (que no solo detectara los cruces por 0 de la fase R).

6. Sistema real

6.1. Introducción

Luego de numerosas simulaciones y pruebas de los algoritmos, tanto en el PC como en el DSP, llegó el momento de llevar todo lo desarrollado a la práctica. Esto es el DSP conectado al VSI real filtrando armónicos de una carga no lineal. Igualmente se realizaron varias pruebas con el VSI antes de efectivamente filtrar las secuencias armónicas. Estas pruebas consistieron básicamente en chequear cada bloque del filtro interactuando con el sistema real y poner en marcha la adquisición de señales, la cual no se utilizaba en las etapas anteriores.

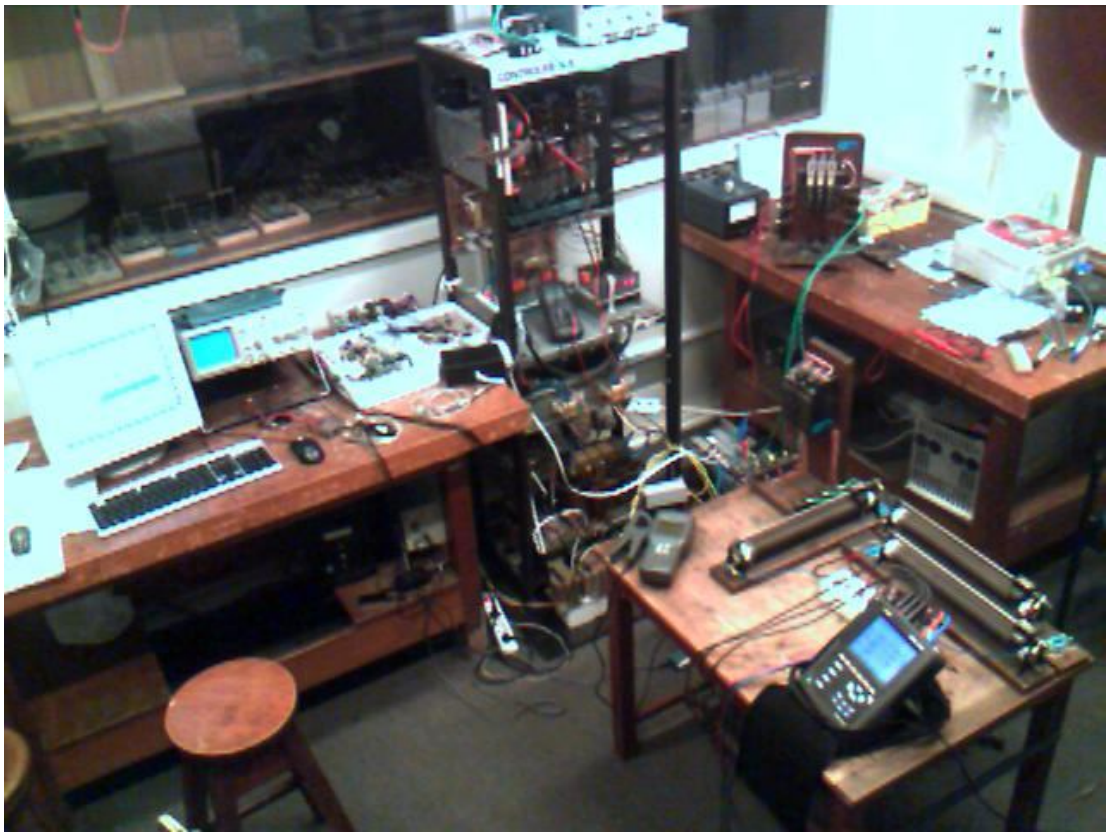


Figura 6.1.1. – Sistema completo en el laboratorio de electrónica de potencia (IIE).

6.2. El sistema completo

En el apartado 5 de este documento se describe el Hardware in the loop, es en ese sistema que se desarrollaron todos los algoritmos del DSP probándolos con el sistema de potencia simulado. Ahora este sistema de potencia es real. En esta etapa el DSP debe adquirir todas las señales que involucran el filtrado, a diferencia del hardware in the loop en el cual las señales llegaban al DSP por el puerto serie, pero en si los algoritmos son los mismos, por lo cual no se describen aquí.

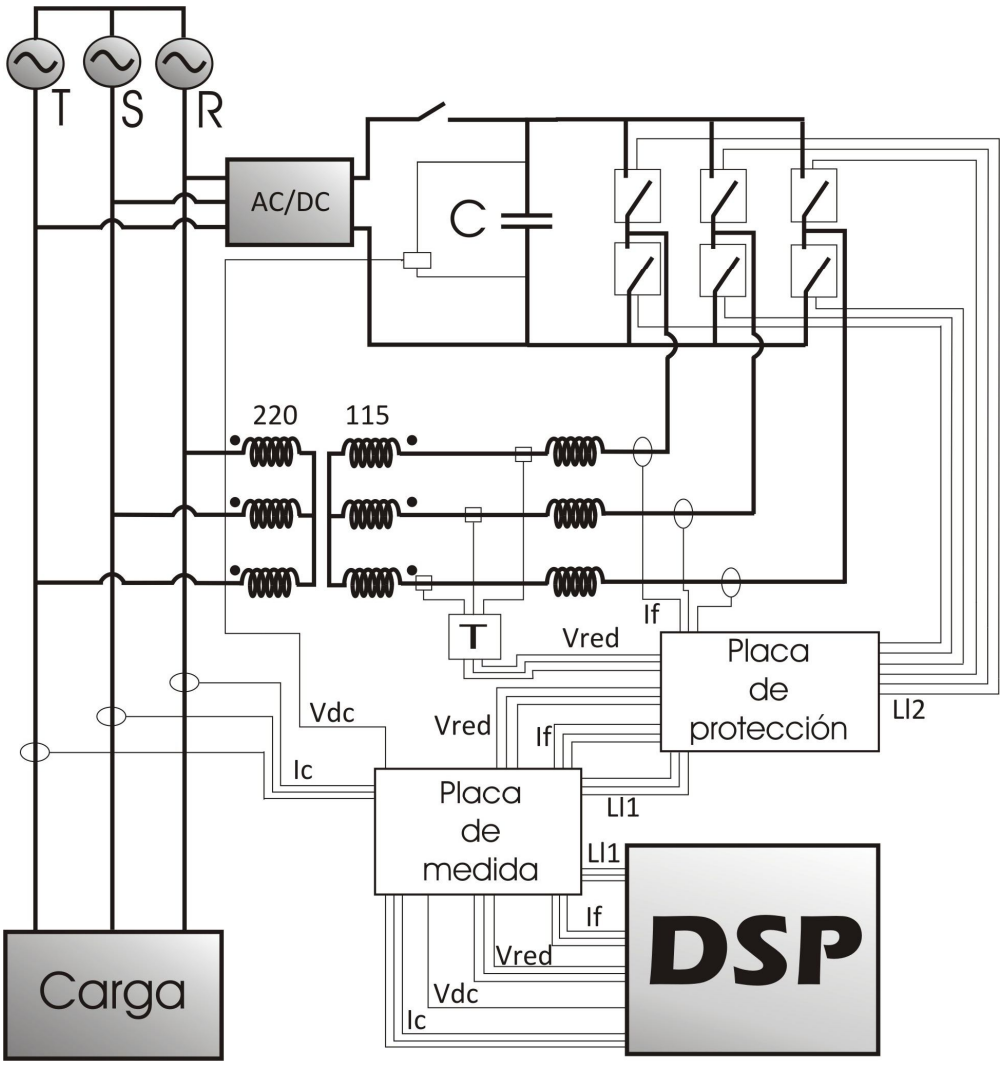


Figura 6.2.1. – Sistema completo.

En la figura 6.2.1 se muestra un esquema simplificado del sistema completo. En esta se puede ver la red conectada a la carga y el VSI conectado en paralelo por medio de un transformador 115/220. También se muestra el convertidor AC/DC, que se utilizó para cargar inicialmente el condensador del VSI, pero en el sistema final, la idea es que el condensador sea cargado por el control de tensión. También se pueden ver las placas de acondicionamiento de señal y protección para la medida de todas las variables que conciernen al filtrado. La placa de medida fue diseñada y fabricada en el transcurso de este proyecto, la placa de protección es producto de otros proyectos de fin de carrera ([10] y [3]). Ésta última ya manejaba algunas de las señales a adquirir, como lo son las tensiones de la red, las corrientes del filtro, los estados de las llaves a leer y a escribir. Las señales restantes, corrientes de carga y tensión del condensador, son acondicionadas en la placa de medida, igualmente todas las señales pasan por la placa de medida. Solo se muestran las señales de control del las llaves, *LI1* y *LI2*, pero también se utilizan, los estados que retornan de las placas driver de los IGBT, para ver que verdaderamente esté funcionando correctamente todo el sistema de manejo de los IGBT. Las señales *LI1* son una para cada rama del VSI, en el PLD de la placa de protección, por medio de una lógica, se generan las seis señales de control de los IGBT, *LI2*, de forma tal que se tienen en cuenta los retardos de los IGBT para que no se generen cortocircuitos en las ramas, o sea que los dos IGBT de una rama queden prendidos a la vez. Las tensiones en el punto de conexión son medidas por medio de tres transformadores, en la figura es llamado **T**.

6.3. Modificaciones en el DSP

En general el código del DSP es el mismo que el del hardware in the loop, sólo se adiciona en esta parte la adquisición de señales, la comunicación con una interfaz gráfica

realizada en Matlab (GUI) y las interrupciones con el timer. Pero esto no hace que esta parte sea menos importante, todo lo contrario, es muy importante, dado el riesgo que implica el manejar sistemas de potencia, mas que nada porque se pueden generar perdidas materiales.

6.3.1. Módulo ADC

El módulo ADC del TMS320F2812, posee 16 entradas analógicas, 2 samples and holds, un conversor AD de 12 bits de resolución y otros bloques para manejar la lógica [11]. Al igual que se hizo con los demás periféricos, se utilizo los Header Files and Peripheral Examples [5], referidos al manejo de los ADC, para la familiarización con este periférico. Tiene varios modos de trabajo, modo on/off, y varios modos secuenciales. En particular se utilizó un modo secuencial de muestreo continuo con override, en software lo único que se hace es inicializar el periférico en el modo adecuado y luego darle comienzo a la secuencia, esta toma muestras continuamente de los canales deseados, guardando las muestras en registros específicos, pero todo esto es en hardware sin ningún tipo de intervención software, luego en el programa principal y desde software se lee los registros a la frecuencia de muestreo utilizada.

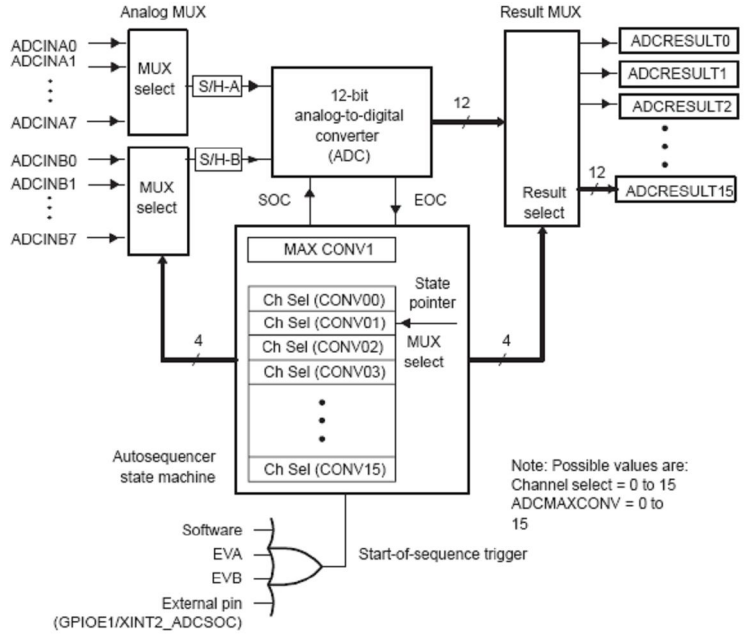


Figura 6.3.1 – Esquema del periférico de los ADC

En la figura 6.3.1 se muestra un esquema de los bloques que componen el periférico de los ADC. Aquí se pueden ver los MUX de las entradas analógicas, los samples and holds, el conversor AD, los registros donde se guardan las muestras y la maquina de estados que controla la secuencia de muestreo. Los CONVxx que se encuentran en la maquina de estados contienen 4 bits con los cuales se direccionan las entradas, o sea si por ejemplo el CONV00 vale 0000b, el canal ADCINA0 se va a guardar en el registro ADCRESULT0. El modo de trabajo seleccionado y la configuración es tal que el modulo esta funcionando continuamente, tomando muestras del ADCINA0-7 y del ADCINB0-7 simultáneamente. En la figura 6.3.2 se ilustra un diagrama de tiempos sobre el modo de funcionamiento. Luego de un ADC SOC Trigger, comienza la secuencia, el State pointer que se puede ver en la figura 6.3.1 arranca en 0, con esto se tiene seleccionado el canal analógico que se encuentra en el registro CONV00, nuestra configuración es tal que se corresponden las entradas con los registros, ver tabla 6.1, se da el pulso de adquisición en la señal SH Clock, la cual es una división de el reloj del DSP, durante este pulso los samples and holds almacenan las tensiones que se encuentran en los pines ADCINA1 y ADCINB1, luego durante los tiempos C1 y C2 se hace la conversión de ambos samples and holds, luego los datos son guardados en ADCRESULT0 y ADCRESULT8 respectivamente. Luego se pasa al siguiente paso con un nuevo pulso en SH Clock, así se continúa hasta que se llega a la última señal a muestrear, es aquí que se vuelve al comienzo y se actualizan los datos. Si se observa la figura 6.3.2 se puede apreciar que los tiempos de la primera conversión se superponen con los de la segunda, esto es porque el AD trabaja como un pipeline.

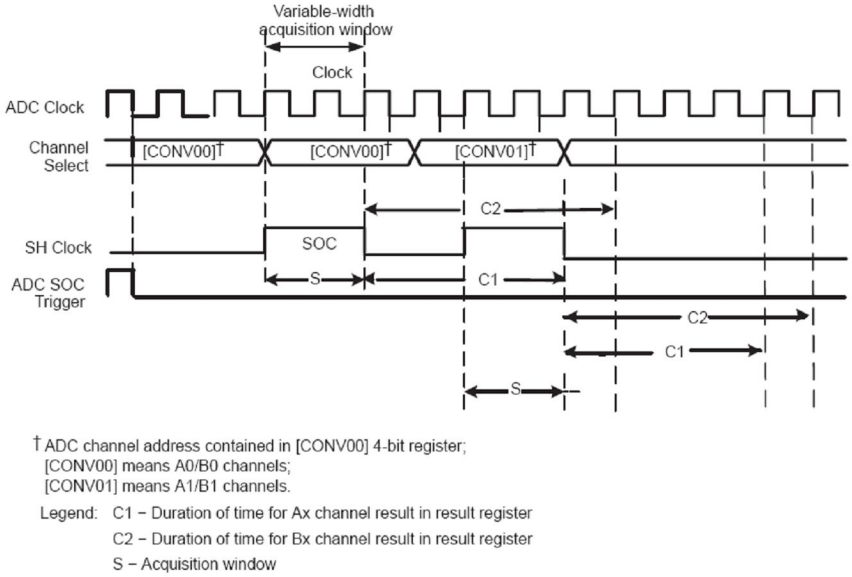


Figura 6.3.2 – Modo de muestreo simultaneo

Pin	ADCRESULT	Puerto placa	# pin del puerto	Señal asociada
ADCINA0	ADCRESULT0	P9	2	IFr
ADCINA1	ADCRESULT1	P9	4	IFs
ADCINA2	ADCRESULT2	P9	6	IFt
ADCINA3	ADCRESULT3	P9	8	ICr
ADCINA4	ADCRESULT4	P9	10	ICs
ADCINA5	ADCRESULT5	P9	12	ICt
ADCINA6	ADCRESULT6	P9	14	-
ADCINA7	ADCRESULT7	P9	16	-
ADCINB0	ADCRESULT8	P5	1	Vr
ADCINB1	ADCRESULT9	P5	2	Vs
ADCINB2	ADCRESULT10	P5	3	Vt
ADCINB3	ADCRESULT11	P5	4	-
ADCINB4	ADCRESULT12	P5	5	-
ADCINB5	ADCRESULT13	P5	6	-
ADCINB6	ADCRESULT14	P5	7	-
ADCINB7	ADCRESULT15	P5	8	-

Tabla 6.1 – Asignación de pines y mapeo

En cuanto a la resolución del ADC, las entradas se encuentran entre 0V y 3V, la conversión es de 12 bits, esto implica que a 3V le corresponden FFFh o 4095d, en otras palabras, si se estuviera adquiriendo una señal que varia entre 0V y 3V, se tendría en el DSP una señal variando entre 0 y 4095 si se hablara en números enteros. La idea es trabajar con números IQ21, por lo cual luego en la calibración de las señales hay que tener en cuenta esto.

6.3.2. Protocolo de comunicación GUI – DSP

En cuanto al GUI, se realizo una función que implementa el protocolo de comunicación con Matlab por medio del puerto serie. Para esta comunicación se puede decir que el DSP actúa en modo esclavo y el PC en modo maestro, esto es el PC siempre inicia las comunicaciones y el DSP se limita a responder. La comunicación se inicia por un byte que envía el PC, llamado *offset*, este puede significar cinco cosas:

- Arrancar el filtrado (*offset* = 45)
- Detener el filtrado (*offset* = 50)
- Leer estado del filtro, si está corriendo o no (*offset* = 44)
- Modificar un parámetro (*offset* < 128 y *offset* ≠ 45 y *offset* ≠ 50 y *offset* ≠ 44)
- Leer un parámetro (*offset* ≥128)

Para acceder a las variables se cuenta con dos arrays de punteros, *punts*, y *puntsInt*. El array *punts* es para almacenar punteros a las variables de tipo IQ, y *puntsInt* para los de tipo entero. Para acceder a cada variable se utiliza *offset* como índice en estos arrays.

En la figura 6.3.3 se muestra un ejemplo de lo que podría ser una comunicación entre el PC y el DSP, en este caso el PC primero envía offset con el valor 45, esto es la orden de comenzar el filtrado. En la primera comunicación (1ª Com) lee el contenido de la variable *Var_n* enviando el valor de *offset + 128*, correspondiente a la variable *Var_n*, luego el DSP responde con el valor *Y*. La adición de 128 a *offset* es para estar en la parte superior del rango de 1 byte. En la segunda comunicación (2ª Com) el PC escribe en el DSP en la misma variable el valor *X*, pero esta vez envía solo *offset*. Y en la tercera comunicación (3ª Com) se verifica el valor de *Var_n*. Por ultimo se envía *offset* con el valor 50 para detener el filtrado.

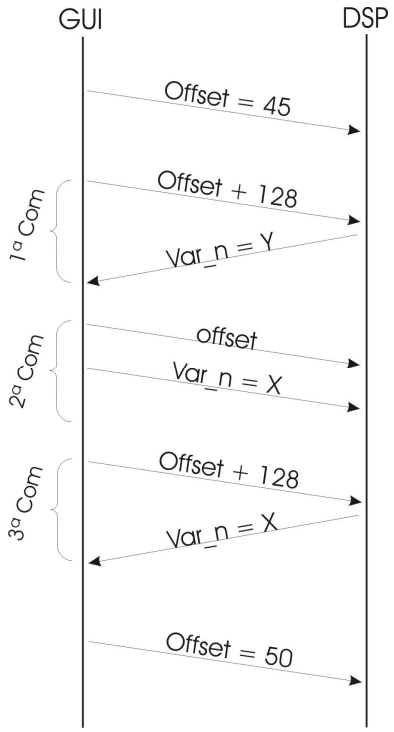


Figura 6.3.3 – Ejemplo de comunicación entre el PC (GUI) y el DSP

Las variables pueden ser cambiadas sin necesidad de estar filtrando.

En la figura 6.3.4 se ilustra el diagrama de flujo de la función GUI del DSP que implementa la comunicación entre el DSP y el PC. Es llamada desde el programa principal en cada instante de muestreo. Esta función tiene la característica de no quedar en lazos infinitos de espera que interrumpen los programas, esto se logra por medio de la flag *wait*. El propósito de esta flag es determinar si se están esperando datos o no. Solo cambia de valor si se recibe un *offset* menor que 128, distinto de 44, de 45, de 50 y no se recibieron los 4 bytes desde el PC. La condición anterior implica que el PC va a enviar 4 bytes con el contenido de la variable a modificar. Es necesario utilizar el flag *wait* porque no se sabe cuanto tiempo puede demorar el PC en enviar los datos.

Para explicar como se comporta este programa, se presentan un par de ejemplos. Primer ejemplo, el PC quiere modificar una variable. O sea primero el PC va a enviar un byte con el valor de *offset* correspondiente a la variable deseada, y luego enviara 4 bytes con el nuevo valor de la variable en formato *float* de 4 bytes (tipo *single* en Matlab). En el DSP se va a recibir el primer byte el cual corresponde a *offset*, se verifica en que rango se encuentra este byte y si tiene alguno de los valores de las ordenes de arranque o parado o lectura de estado. Como el PC quiere modificar una variable, este va a ser menor que 128. Ahora se pregunta si hay 4 bytes en el buffer, esto es imposible, dado que la velocidad del puerto serie es muy lenta comparada con la velocidad del DSP. Enviar 5 bytes al puerto serie le lleva aproximadamente 430us (a 115200bps), y cada instante de muestreo para el DSP son 78.125us, por lo cual para recibir los 5 bytes implican mas de 5 instantes de muestro o 5 llamados a GUI. Por esto último

es importante utilizar la flag *wait.* Como no hay 4 byes se pone *wait* en 1 y se retorna al programa principal. En el siguiente llamado de GUI, se verifica el valor de *wait,* va a estar en 1, por lo cual, se pasa a verificar si ya llegaron los 4 bytes. Como ya se explicó tienen que pasar mas de 5 instantes de muestreo para que lleguen, por lo tanto, se vuelve a poner *wait* en 1 y se retorna al programa principal. Esto se repite hasta que llegan los 4 bytes. En el momento que se constata que están los 4 bytes en el buffer de entrada, se leen, se genera un *long int* con el contenido de los 4 primeros bytes del buffer. Aquí se verifica el valor de *offset*, si es menor que 8, se guarda en **puntsInt[offset]*, en caso contrario, se pasa este entero a *float* y luego a formato IQ, por ultimo este IQ es asignado a lo apuntado por **punts[offset-8]*. Es aquí que se da por finalizada la comunicación.

Segundo ejemplo, el PC quiere leer una variable. Para esto envía un byte con el valor de *offset* de la variable deseada más 128. Y luego el PC espera recibir 4 bytes con el contenido de la variable a leer. Aquí el DSP hace todo en un solo llamado de GUI. Primero verifica el valor de *wait*, si el PC no a violado el protocolo, *wait* va a valer 0, luego lee el byte con *offset*, verifica que sea menor que 128, como no es así, se verifica si *offset-128* es mayor a 7, si es así se quiere leer un *float*, por lo cual se coloca en el buffer de salida los 4 bytes correspondientes al formato float de lo apuntado por **punts[offset-128-8]*. En caso contrario se coloca en le buffer **puntsInt[offset-128]*. Y aquí finaliza la comunicación.

Este protocolo de comunicación puede tener problemas si el PC envía bytes sin sentido, llevando a la función GUI a un estado fuera de control. Pero igualmente como ya se menciono, esta función al no generar lazos de espera, se hace independiente del filtrado. La programación del GUI en el PC esta echa de forma tal que se respeten todos los tiempos y se siga el protocolo al pie de la letra, asegurando así el correcto funcionamiento.

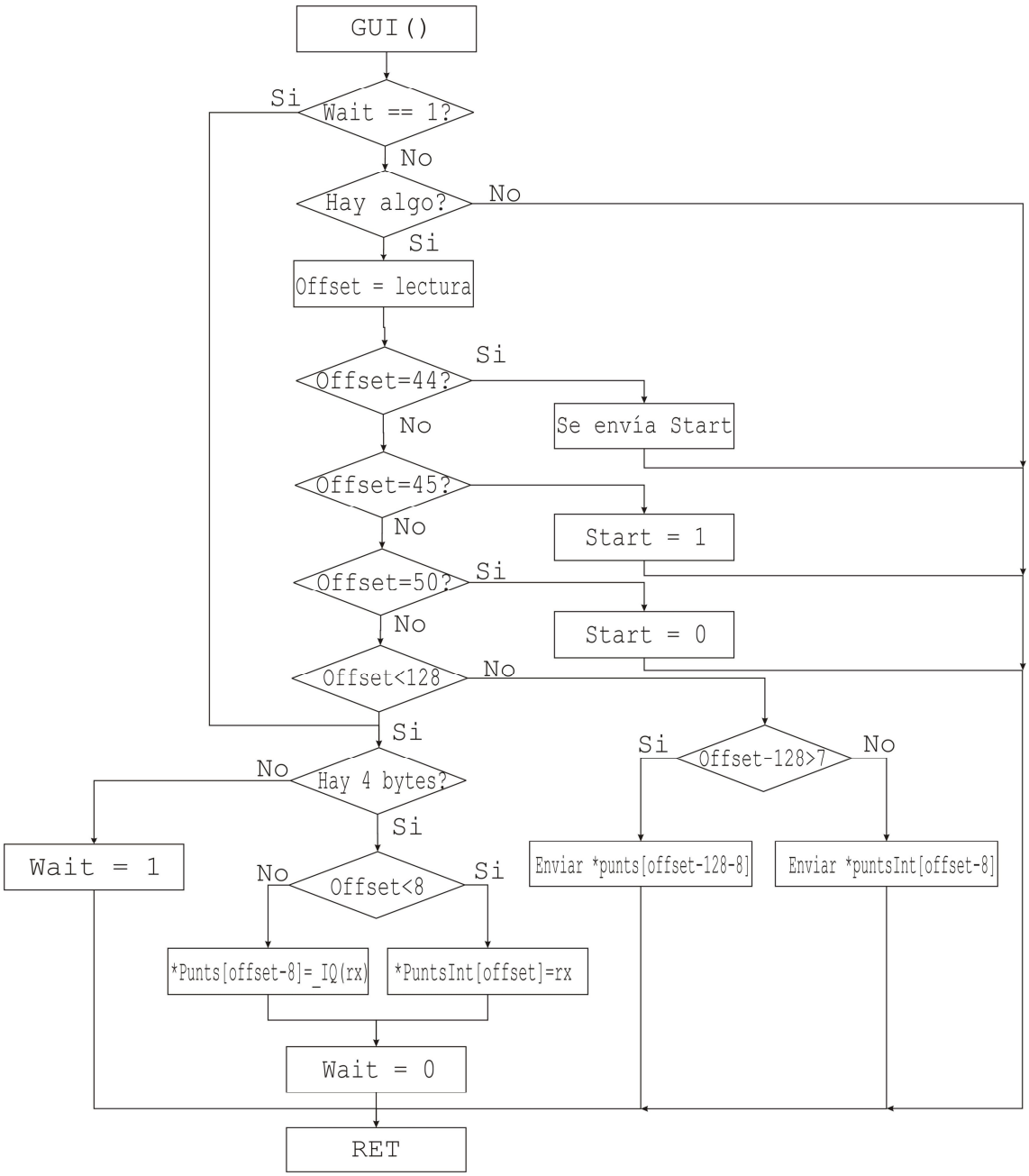


Figura 6.3.4 – Diagrama de flujo de la función GUI del DSP

6.3.3. Interrupciones con el CPU–Timer

El TMS320F2812 posee 3 CPU–timers [12], llamados TIMER0/1/2. Estos son tres contadores descendientes de 32 bits. Los TIMER1 y TIMER2 están reservados para real-time OS (Sistemas operativos de tiempo real), el TIMER0 esta destinado a aplicaciones de usuario, como la desarrollada en este proyecto. En la figura 6.3.5 se muestra un esquema general del los CPU–timers.

El registro contador TIMH:TIM es cargado con el valor del registro de periodo PRDH:PRD. El contador de registro se decrementa con cada TDDRH:TDDR +1 ciclo de SYSCLKOUT. Cuando el contador llega a 0, se genera un pulso en la señal TINT\ el cual genera una interrupción, y TIMH:TIM se recarga con el contenido de PRDH:PRD.

Hilando un poco más fino, en cada ciclo de SYSCLKOUT, PSCH:PSC se decrementa en 1, hasta alcanzar 0, aquí se recarga con el contenido de TDDRH:TDDR, y le genera un pulso al contador para que se decremente.

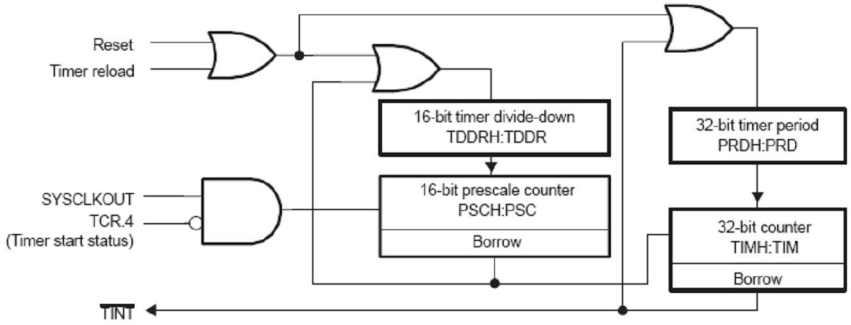


Figura 6.3.5 – Esquema de los CPU–timers

En esta aplicación su utilizó el TIMER0, interrumpiendo cada 78.125us, eso es 12800kHz. En la rutina de atención a la interrupción se genera un pulso en una salida digital y se le da el valor 1, a una flag llamada *muestrear*. El pulso es para el watchdog de la placa de protección [10]. En el programa principal se hace un polling de la flag *muestrear*, si esta es 0, no se hace nada y sigue el loop del programa principal. Si *muestrear* es 1, se corren todas las funciones del programa principal y se retorna la flag a 0. Es de esta manera que se maneja la frecuencia de muestreo.

Al igual que con los demás periféricos se utilizaron los Header Files and Peripheral Examples [5], para la familiarización con los CPU–timers. Se utilizaron las funciones que proveen estos archivos para llevar a cabo la inicialización del timer, y la programación de este. En el anexo D se muestra el archivo donde se definen estas funciones.

6.3.4. Señales digitales

A parte de las señales analógicas, otro de los medios por donde se interactúa con el mundo es por medio de las señales digitales, como entradas o como salidas. Estas son utilizadas para:

- Mantener dormido el watchdog de la placa de protección
- Habilitar y deshabilitar la palca de protección
- Conmutar las llaves
- Leer los estados de las llaves
- Manejar los LEDs de la placa de medida

Las señales digitales pertenecen al periférico GPIO (entradas y salidas de propósito general [5]). El manejo del periférico se hizo por medio de los Header Files and Peripheral Examples. Este periférico administra la interfaz digital del DSP. Todas las señales que maneja deben ser configuradas para que trabajen del modo deseado, dado que pueden ser manejadas por el usuario o por otros periféricos. En las aplicaciones que aquí se detallan son utilizadas como puertos de entrada salida simples.

Para configurar las señales solo hay que hacer tres cosas, definir la señal como de propósito general, definir la dirección, o sea como entrada o salida y por último definir si se quiere que sea una señal síncrona. A continuación se muestra un ejemplo:

```
GpioMuxRegs.GPAMUX.bit.PWM1_GPIOA0 = 0;  // de propósito general
GpioMuxRegs.GPADIR.bit.GPIOA0 = 1;        // dirección = salida
GpioMuxRegs.GPAQUAL.all = 0x0000;         // asíncrona
```

El Header File referido al GPIO proporciona una serie de métodos para manejar las señales. Si hablamos de una salida, se tienen cuatro modos de escritura.

Escribir un 0 en el puerto:

```
GpioDataRegs.GPACLEAR.bit.GPIOA6 = 1;
```

Escribir un 1 en el puerto:

```
GpioDataRegs.GPASET.bit.GPIOA5 = 1;
```

Escribir el estado inverso al que posee, hacer un “toggle”:

```
GpioDataRegs.GPATOGGLE.bit.GPIOA3 = 1;
```

Escribir cualquier valor:

```
GpioDataRegs.GPADAT.bit.GPIOA0 = (llaves>>2) & 0x0001;
```

Para mantener inactivo el watchdog, se le debe mandar pulsos a una frecuencia mayor a 2kHz. En nuestro caso generamos una onda cuadrad de frecuencia 6,4kHz. Esta es generada haciendo un toggle en GPIOA6 durante la interrupción.

Para habilitar y deshabilitar la salida de la placa de protección, esto es comandar o no las llaves, se maneja la habilitación de un buffer de la placa antes mencionada. Esto se hace poniendo a cero (deshabilitar) o a uno (habilitar) un puerto, en este caso es el GPIOB8.
Para deshabilitar:

```
GpioDataRegs.GPBCLEAR.bit.GPIOB8 = 1;
```

Para habilitar:

```
GpioDataRegs.GPBSET.bit.GPIOB8 = 1;
```

La conmutación de las llaves se hace escribiendo el estado de una variable entera, *llaves*, en tres puertos simples. El estado de la variable *llaves* es manejado por el control vectorial. La manera de manejar las señales se muestra a continuación:

```
GpioDataRegs.GPADAT.bit.GPIOA0 = (llaves>>2) & 0x0001;
GpioDataRegs.GPADAT.bit.GPIOA1 = (llaves>>1) & 0x0001;
GpioDataRegs.GPADAT.bit.GPIOA2 = llaves & 0x0001;
```

Cada señal se corresponde con un bit del puerto, por lo cual se debe hacer un shift de *llaves* y un *and* con 0x0001, para así dar el valor correcto al puerto.

Como protección se leen los estados de las llaves, y se verifica que posean el valor dado en el instante de muestreo anterior, por lo tanto esto es lo primero que se hace en cada instante de muestreo. A continuación se muestra el código que implementa esta protección.

```
if ((cont_start == 1) &
((GpioDataRegs.GPBDAT.bit.GPIOB0 != ((llaves>>2) & 0x0001)) ||
 (GpioDataRegs.GPBDAT.bit.GPIOB1 == ((llaves>>2) & 0x0001)) ||
 (GpioDataRegs.GPBDAT.bit.GPIOB2 != ((llaves>>1) & 0x0001)) ||
 (GpioDataRegs.GPBDAT.bit.GPIOB3 == ((llaves>>1) & 0x0001)) ||
 (GpioDataRegs.GPBDAT.bit.GPIOB4 != (llaves & 0x0001)) ||
 (GpioDataRegs.GPBDAT.bit.GPIOB5 == (llaves & 0x0001))){
      fin();
} else {
      if ( cont_start == 2) {
            // Habilito la salida
            GpioDataRegs.GPBSET.bit.GPIOB8 = 1;
      }
}
```

Si el estado de las llaves no es el correcto, se invoca la función *fin*, la cual lleva las llaves a 0 y deshabita la salida de la palca de protección. Las protecciones se describen en la sección 6.3.5. La variable *cont_start*, se utiliza para arrancar el PLL antes que el filtrado, y para leer el estado de las llaves en el instante de muestreo posterior al arranque del filtro.

Los leds se manejan de forma similar que la señal de habilitación de la placa de protección. Se poseen dos leds, uno verde y otro rojo. El led verde indica que la placa de medida esta energizada y conectada. El led rojo indica que se invocó la función *fin*, o sea se activo alguna de las protecciones implementadas en el DSP.

6.3.5. Protecciones software

Una adición muy importante que se realizo en esta parte, fue la implementación de protecciones. Estas se hicieron en esta parte dado que hasta el hardware in the loop parecían carecer de sentido, y no fueron tomadas en cuenta en esa etapa. Pero vale destacar que hubiera sido positivo haber simulado estas protecciones para así ver su respuesta, y el comportamiento del sistema frente a éstas. Desde otro punto de vista, se podría haber estimado la cantidad de operaciones necesarias para las protecciones, para luego así estimar el tiempo de respuesta real, etc.

Se tienen básicamente tres protecciones, contra sobrecorriente en el filtro, contra sobretension en el condensador y por estado erróneo de las llaves, esta última protección se

describe en la sección 6.3.4. Todas tienen el mismo efecto, invocan la función *fin*, que se muestra a continuación:

```
void fin(){
      // se apagan todas las llaves
      GpioDataRegs.GPBCLEAR.bit.GPIOB8 = 1;

      // se dejan las llaves en 0 (por las dudas)
      GpioDataRegs.GPACLEAR.bit.GPIOA0 = 1;
      GpioDataRegs.GPACLEAR.bit.GPIOA1 = 1;
      GpioDataRegs.GPACLEAR.bit.GPIOA2 = 1;

      //  se prende el LED rojo
      GpioDataRegs.GPASET.bit.GPIOA4 = 1;
}
```

Esta función lo primero que hace es deshabilitar la salida de la placa de protección, luego lleva a cero las señales correspondientes a las llaves, esto último llega sólo hasta la entrada del buffer, dado que este se apago anteriormente. Por último se prende el led rojo de la placa de medida (ver sección 6.6.3). En definitiva esta función lo que hace es deshabilitar la salida hacia el VSI, e indicar con el led rojo que hubo un problema.

La protección contra sobre corriente verifica que el modulo de las corrientes del filtro adquiridas no sean superiores a 14A, a continuación se muestra la implementación:

```
if (_IQabs(Ifr)>_IQ(14) | _IQabs(Ifs)>_IQ(14) | _IQabs(Ift)>_IQ(14)){
      fin();
}
```

La protección de sobre tensión verifica que el valor de la tensión en el condensador adquirida no se mayor que 365V durante dos instante de muestreo consecutivos, si esto sucede se invoca a la función *fin*. A continuación se muestra la implementación:

```
if (E > _IQ(365)) {
      if (conteoSobreTension < 1)  {
            conteoSobreTension++;
      } else {
            fin();
      }
} else {
      conteoSobreTension = 0;
}
```

La variable *conteoSobreTension* es utilizada para que se verifique en dos instantes de muestreo que la tensión este sobre 365V, esto es así, por que puede haber una medida errónea causada por ruido.

6.4. Pruebas

Las pruebas consistieron en testear cada bloque que corre en el DSP interactuando con el sistema real. El sistema de potencia era dado y no se podía modificar demasiado. En general el único cambio que sufrió fue bajar la tensión de la red de 220V a 110V. Esto se debe a que se observó en las simulaciones que la tensión en el bus de continua debe ser de más de 450V para tener buenos resultados con una red de 230V. Sin embargo, el VSI que controlamos posee una protección para no sobrepasar los 360V (módulo de frenado disipativo independiente). Una primera solución a este problema es la inserción de un transformador estrella-estrella entre las inductancias del filtro y la red, de forma que la red vista por el filtro fuese de una tensión menor. Esto se probó con un transformador de 220V/115V, pero se observó que las corrientes generadas eran desfasadas debido a la inductancia de vacío. Esto

complica el filtrado, dado que es esencial que las corrientes del filtro estén en fase con la corriente de carga (o dicho de otra forma, en contrafase con sus armónicos). Esto en principio podría ser solucionado mediante el ajuste de la fase hallada por el PLL, además de la posible corrección para cada celda selectiva. Sin embargo, debido a la falta de tiempo se optó por tomar como tensión de red los "110V", que en realidad son 120V nominales. Es decir que la carga también se conectó a esta tensión.

Una de las partes mas importantes de esta etapa, es tener en el DSP una imagen lo mas cercana posible a la realidad, esto significa que lo que midiéramos del sistema real se aproximara muy bien a las mediadas reales. Por tal motivo la calibración de la placa de medida es un punto crucial de esta etapa.

6.4.1. Calibración de entradas analógicas

A través de la placa de mediada pasan todas las señales del sistema. Algunas de éstas se acondicionan y otras no. Para ver ésto con más detalle véase sección 7 Hardware adicional.

Las señales a calibrar son:

- Corrientes del filtro
- Corrientes de carga
- Tensión en bornes del condensador
- Tensiones de la red

Para la calibración se utilizó un programa de Matlab (*medida.m*) y otro del DSP (*calibración.c*). En el programa del DSP se adquiere una señal real y se envían 1024 muestras por puerto serie. El programa de Matlab recibe estas muestras, hace el promedio de todas y grafica la señal.

La más simple de todas y que no implico muchos problemas fueron las tensiones de la red, dado que solo se usan en el PLL y no es necesario tener precisión en cuanto a amplitud, puesto que lo único que importa de estas señales es la fase. Por otro lado esta es unas de las señales mas grandes en modulo que se manejan, lo cual trae aparejado problemas de saturación en los algoritmos del PLL, por tal motivo estas señales fueron fuertemente atenuadas numéricamente. Como ya se mencionó en la sección 6.3.1, el valor máximo entero que podemos adquirir es 4095, por lo cual para hacer una calibración lo que hay que hacer es una regla de tres. En definitiva a las señales de tensiones de la red se las atenúa y se les resta el offset que genera la adquisición:

```
vr = _IQmpyI32(_IQ(0.2*0.00146555935515),(long int)(AdcRegs.ADCRESULT8>>4))-_IQ(0.57243795776367);
vs = _IQmpyI32(_IQ(0.2*0.00146555935515),(long int)(AdcRegs.ADCRESULT9>>4))-_IQ(0.57072766113281);
vt = _IQmpyI32(_IQ(0.2*0.00146555935515),(long int)(AdcRegs.ADCRESULT10>>4))-_IQ(0.57017230224609);
```

Más adelante se verá que esta calibración debió ser modificada para mejorar la respuesta transitoria del PLL.

Para calibrar las señales alternas, observando la grafica generada, se determina la amplitud pico a pico de la señal adquirida. Si por ejemplo se sabe que la señal tiene amplitud X, y de la grafica se determinó una amplitud Y, entonces se modifica la ganancia a X/Y. Luego se compila y corre nuevamente el programa del DSP y de Matlab, se verifica la amplitud pico a pico, si es distinta se hace el mismo procedimiento y se itera, en caso contrario se mide el offset y se le reta en el programa, se corre nuevamente y se verifica, si todo esta bien se puede seguir con otra señal. Esta calibración resulto para las corrientes de carga:

```
Ir1 = _IQmpyI32(_IQ(0.00729483282675),(long int)(AdcRegs.ADCRESULT3>>4))-_IQ(15.19179904549523);
Is1 = _IQmpyI32(_IQ(0.00729483282675),(long int)(AdcRegs.ADCRESULT4>>4))-_IQ(15.24605328361064);
It1 = _IQmpyI32(_IQ(0.00729483282675),(long int)(AdcRegs.ADCRESULT5>>4))-_IQ(15.12497264978117);
```

Para las corrientes del filtro:

```
Ifr = _IQ(16.53168190473989)-_IQmpyI32(_IQ(0.00793381322012),(long int)(AdcRegs.ADCRESULT0>>4));
Ifs = _IQ(14.94341020249683)-_IQmpyI32(_IQ(0.00723497993130),(long int)(AdcRegs.ADCRESULT1>>4));
Ift = _IQ(14.89854100857263)-_IQmpyI32(_IQ(0.00719768622031),(long int)(AdcRegs.ADCRESULT2>>4));
```

Para calibrar la tensión en bornes del condensador, se cortocircuito el condensador y se midió el cero, efectivamente este promedio dio cero. Luego se cargo el condensador con tensión de trabajo, y se corrió el programa nuevamente, luego con el promedio que arrojo Matlab se determino la ganancia necesaria como el cociente entre lo medido con multimetro, y lo medido con Matlab. Esta calibración resulto:

```
E=_IQmpyI32(_IQ(0.12490605165941),(long int)(AdcRegs.ADCRESULT11>>4));
```

En general no fue necesario realizar demasiadas iteraciones, en el entorno de 2 o 3.

En cuanto a la resolución utilizada en las ganancias y offsets, se puede ver que cuentan con varias cifras significativas, esto es para tratar de tener la mayor precisión posible en las medidas. A estos números los toma el compilador, los convierte en formato IQ y luego los guarda en los lugares que él determine. Al pasar a formato IQ, estos números son aproximados al valor mas cercano en la resolución, por lo cual, si se utilizan varias cifras significativas, se obtiene una precisión aceptable, y esto no implica ningún overhead para el DSP.

6.4.2. Prueba del SSI-PLL

Luego de tener la calibración de todas las señales, el paso natural a seguir es comenzar las pruebas de los bloques. El primero fue el SSI-PLL, la prueba consistió en adquirir las señales de tensión y utilizar estas como entradas del PLL, todo esto a una frecuencia de muestreo de 12.8kHz. Para observar los resultados, se mandan por el puerto serie, muestras de la fase detectada por el PLL y la tensión de la fase R.

Antes de realizar estas pruebas, se realizaron programas para simular el sistema de potencia con Simulik, o sea se probó con el hardware in the loop, obteniendo resultados satisfactorios. Al aceptar los resultados obtenidos con el hardware in the loop, se procedió a llevar el PLL al sistema real. Allí se tuvieron varios problemas, entre ellos, problemas de saturación, se tuvo que atenuar las señales de tensión, esto trajo aparejado el enlentecimiento de la respuesta del PLL. Por tal motivo se replantearon los algoritmos y se llego a una respuesta aceptable.

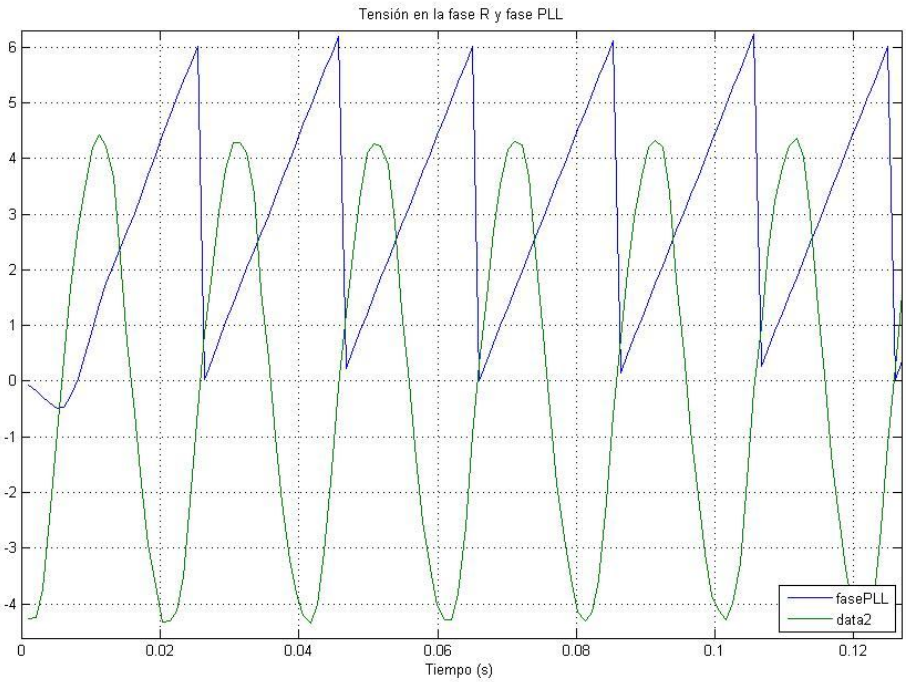


Figura 6.4.1 – Arranque del nuevo PLL

Desde un principio se habían atenuado las tensiones de la red dadas las grandes ganancias que utiliza el filtro SSI. Cuando se llevó a cabo la prueba real se constató un transitorio excesivamente largo, el cual era causado por la pobre amplitud de las señales. La solución adoptada fue modificar la forma de llevar a cabo las operaciones del los algoritmos, con lo cual se solucionó el problema. En la figura 6.4.1 se puede observar la respuesta transitoria del PLL, se ve que en un poco menos de 1 ciclo esta en régimen, esto se ve en la pendiente de la fase, la cual antes de llegar a finalizar el primer ciclo esta estable. Este resultado es coherente con lo observado en las simulaciones del PLL, en las cuales en menos de un ciclo se llegaba al régimen.

Lo que trajo aparejado los cambios en el PLL, fue modificación en la calibración de las tensiones de la red, la cual se muestra a continuación:

```
vr = _IQmpyI32(_IQ(3*0.00146555935515),(long int)(AdcRegs.ADCRESULT8>>4)) -_IQ(9.19906415939331);
vs = _IQmpyI32(_IQ(3*0.00146555935515),(long int)(AdcRegs.ADCRESULT9>>4)) -_IQ(9.14737083435059);
vt = _IQmpyI32(_IQ(3*0.00146555935515),(long int)(AdcRegs.ADCRESULT10>>4)) -_IQ(9.17059282302856);
```

6.4.3. Prueba de control de corrientes

Para probar el control de corriente del filtro se diferenció en dos casos, sin PLL y con PLL. Estas pruebas consisten en generar distintas corrientes de referencia y luego el control vectorial debe tratar de imponerlas en una carga pasiva. La tensión en el condensador es impuesta por el convertidor AC/DC en ambos casos.

Control de corriente sin PLL

El método aquí utilizado para generar la referencia, fue hacer en Matlab, señales de 256 muestras, las cuales después eran guardadas en archivos .asm, para así luego cargarlas en los proyectos del DSP. La forma de generar la onda es recorrer la señal guardada en memoria una muestra por periodo de muestreo. La señal es recorrida por un puntero, el cual se incrementa considerando las 256 muestras como un buffer circular. Para generar las tres fases, se definen dos punteros más, los cuales se encuentran desplazados en el buffer, el número de muestras correspondiente a 120 grados. En la figura 6.4.2 se puede observar un ejemplo de las señales de referencia generadas en Matlab, la de la figura en particular tiene 8A (100%) de fundamental, 2A (25%) de 5º armónico secuencia positiva y 1A (12.5%) de 17º armónico de secuencia positiva.

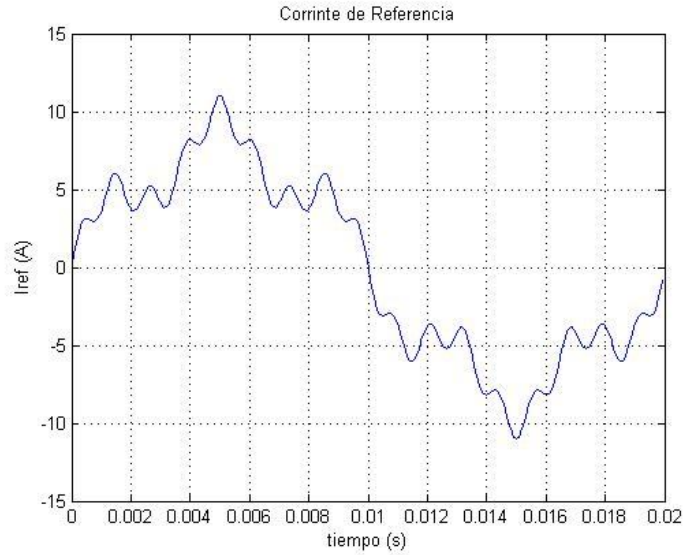


Figura 6.4.2 – Ejemplo de corriente de referencia.

En la figura 6.4.3 se tienen las señales adquiridas en la prueba de control del corriente sin PLL utilizando como corriente de referencia la que se ilustra en la figura 6.4.2. Se puede ver que hay una similitud, pero se puede apreciar el ruido que genera las conmutaciones del VSI. Ahora sí observamos el espectro del las señales adquiridas, el cual se encuentra en la figura 6.4.4, y el porcentaje de armónicos en la figura 6.4.5, vemos que el contenido armónico coincide con el de la corriente de referencia.

El 5º armónico en la referencia era el 25% del fundamental y en la generada fue del 26%. Y en el 17º armónico, en la referencia era el 12,5%, y en la generada fue del 12,7%. Por lo cual se puede concluir que se obtuvieron muy buenos resultados dada la similitud entre la referencia y lo generado. El resto de los armónicos que aparecen en el espectro son debidos a la conmutación de las llaves y es imposible de eliminar. Por otro lado tienen un comportamiento un tanto aleatorio, dada su variabilidad, la cual solo se puede ver en la práctica, no en una figura.

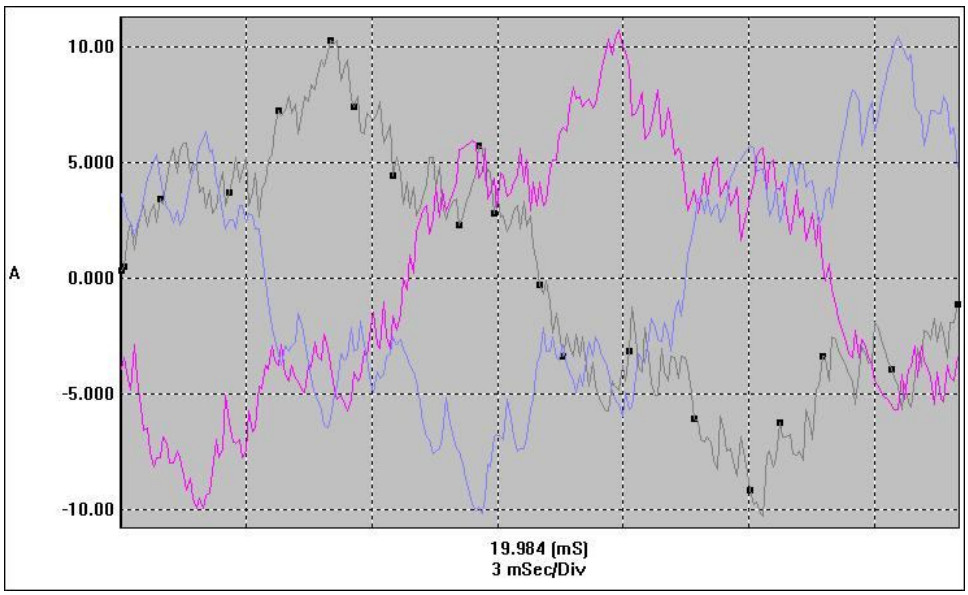


Figura 6.4.3 – Ejemplo de corrientes generadas por el VSI, sin PLL

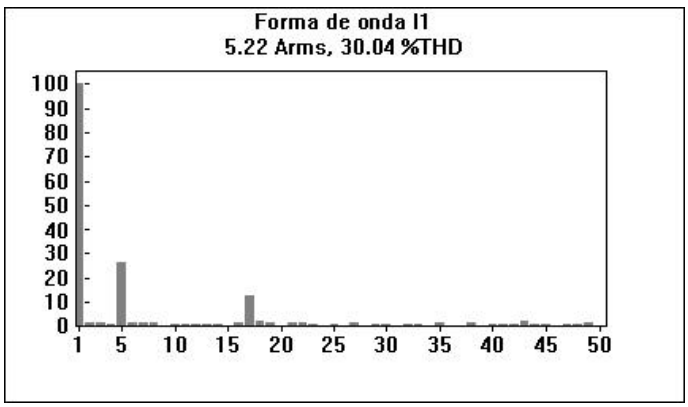


Figura 6.4.4 – Espectro de las señales generadas por el VSI, sin PLL, fase R

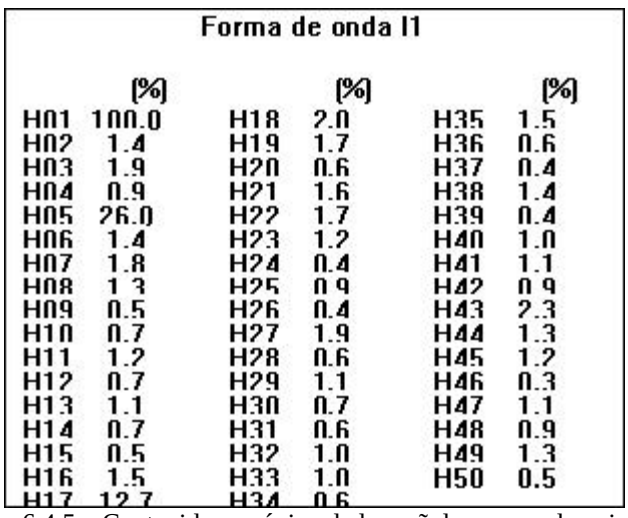


Figura 6.4.5 – Contenido armónico de las señales generadas, sin PLL

Control de corriente con PLL

La prueba realizada fue tomar como corriente de referencia el seno de la fase generada por el PLL. Esto además de ser una prueba del control vectorial, sirve de prueba del PLL. Ya que si la señal de corriente generada con el VSI esta en fase con la tensión de la red, implicaría que el PLL esta funcionando correctamente.

En la figura 6.4.6 se muestran las tres corrientes generadas por el filtro, a simple vista parecen ser sinusoidales, con el mismo tipo de ruido que introduce la conmutación de las llaves, visto en la prueba sin PLL. Si miramos las figuras 6.4.7 y 6.4.8, llegamos a la misma conclusión, se puede ver que todos los armónicos tienen amplitudes muy bajas comparadas con la del fundamental. Estos armónicos son causados por la conmutación de las llaves. El peso de estos armónicos esta relacionado con la amplitud de la señal. Al aumentar la amplitud de la señal que se quiere generar, se reduce la proporción fundamental – armónicos, esto se podría ver con la THD.

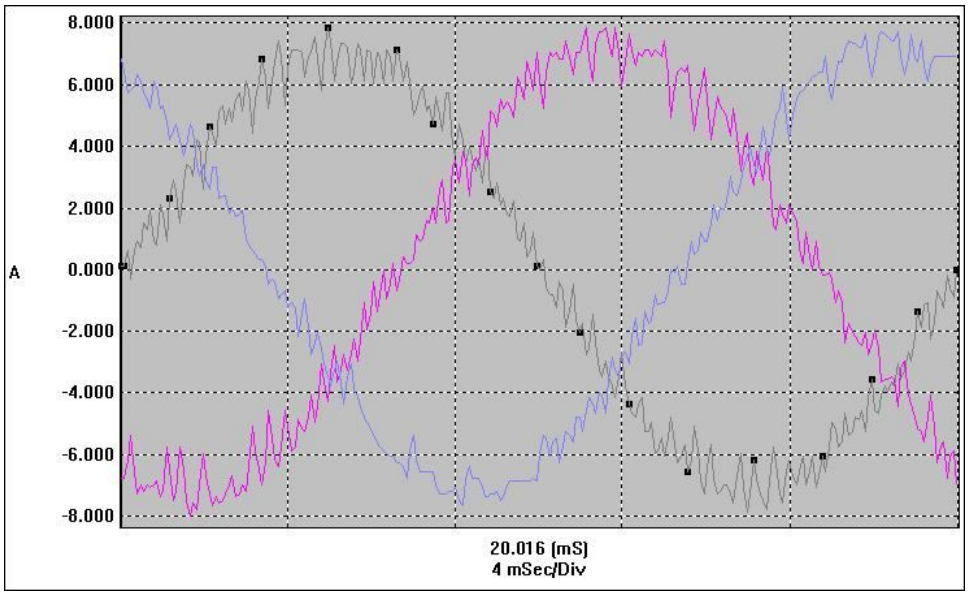


Figura 6.4.6 – Señales de corriente generadas por el VSI, con PLL

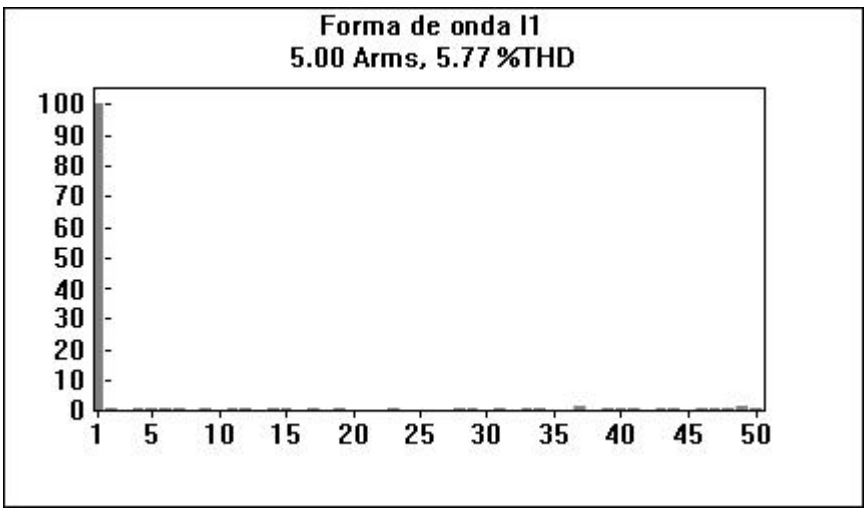


Figura 6.4.7 – Espectro de las señales generadas con el VSI, con PLL, fase R

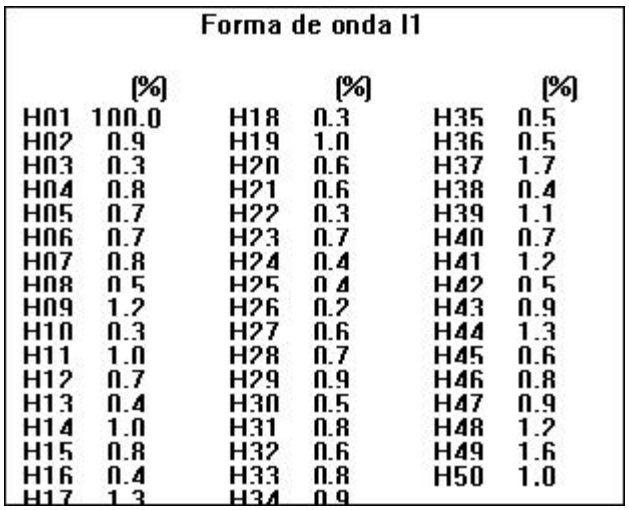


Figura 6.4.8 – Contenido armónico de las señales generadas, con PLL

6.4.5. Prueba de control de tensión en el condensador

Hasta ahora todas las pruebas se realizaron cargando el condensador del inversor con el conversor AC/DC visto en la figura 6.2.1. Pero el sistema real debe usar como fuente de tensión continua la carga en el condensador, y esta debe mantenerse en el valor deseado. Para esto se usa un controlador PI. En la figura 6.4.9 se puede observar el transitorio de la carga del condensador. Se puede ver que la respuesta es un tanto lenta, esto es así dado que si se tratara de ir más rápido con este tipo de controladores, se tendría un sobretiro excesivo. En la figura se puede ver que la señal posee un sobretiro un poco menor de 20V, pero éste es aceptable.

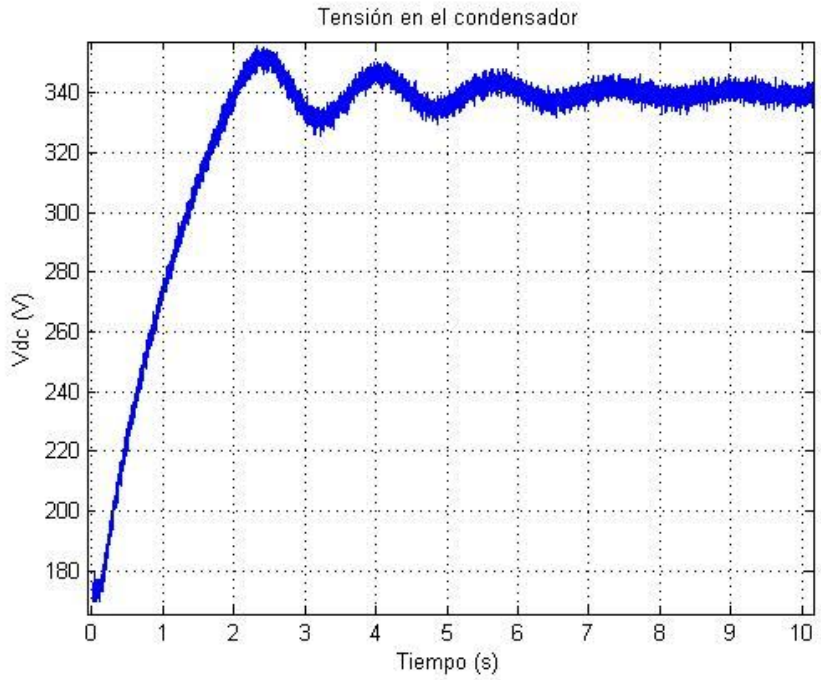


Figura 6.4.9 – Transitorio medido de la tensión en el condensador

La salida del controlador tiene una banda de trabajo, esto es que presenta saturación superior e inferior. Se hizo de esta manera para evitar sobrecorrientes en el filtro. El límite superior es 2 y el inferior -3. En la figura 6.4.10 se puede observar la salida del controlador, en esta se puede ver que en el arranque la señal está saturada a 2. También se puede ver que le toma en el entorno de 15 segundos llegar al régimen.

Otra cosa interesante para ver es la corriente del inversor cuando el controlador se encuentra en régimen. Estas se pueden observar en la figura 6.4.11, se puede ver que es muy

pequeña en amplitud. Al igual que las señales vistas en el control de corriente, esta corriente presenta el mismo ruido causado por la conmutación de las llaves.

Observar esta corriente sirve de prueba de lo mencionado en la sección 6.4.3 sobre el peso de los armónicos cuando disminuye la amplitud de la señal. En la figura 6.4.12 se muestra el espectro de estas corrientes, y en la figura 6.4.13 el contenido armónico de estas señales. Se puede ver que, en porcentaje, la amplitud de los armónicos no es despreciable. Estos rondan el 10% de la fundamental, y algunos casi llegan al 20%. Pero igualmente esto no es gran problema dado que estamos hablando de corrientes muy pequeñas en valor absoluto.

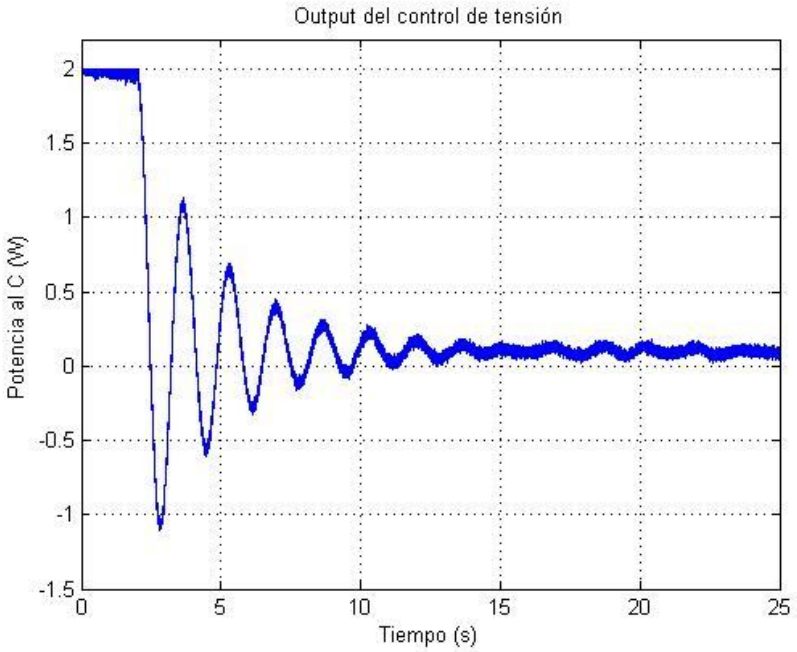


Figura 6.4.10 – Transitorio de la salida del controlador de tensión

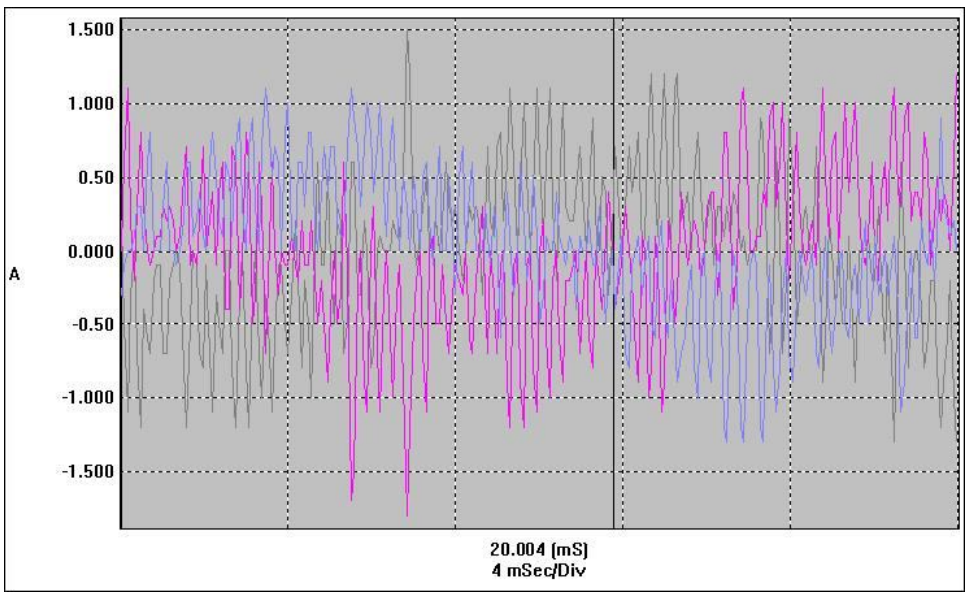


Figura 6.4.11 – Corriente del filtro en régimen, sin filtrar

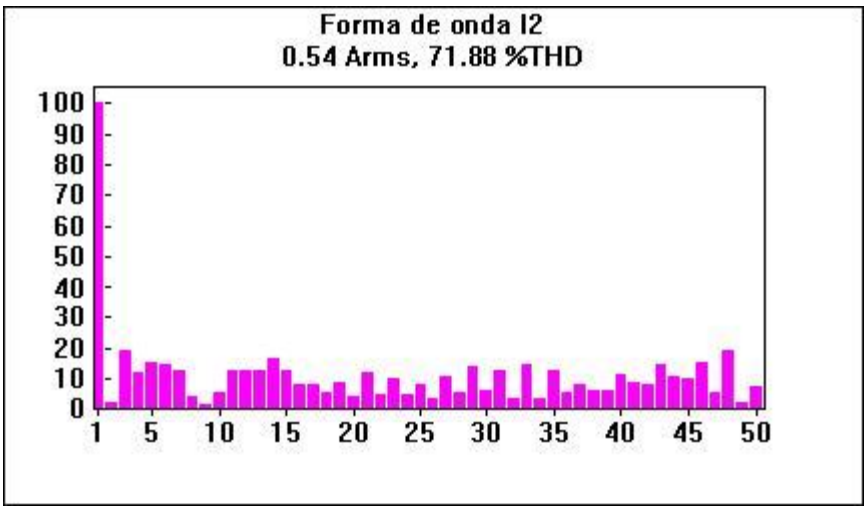
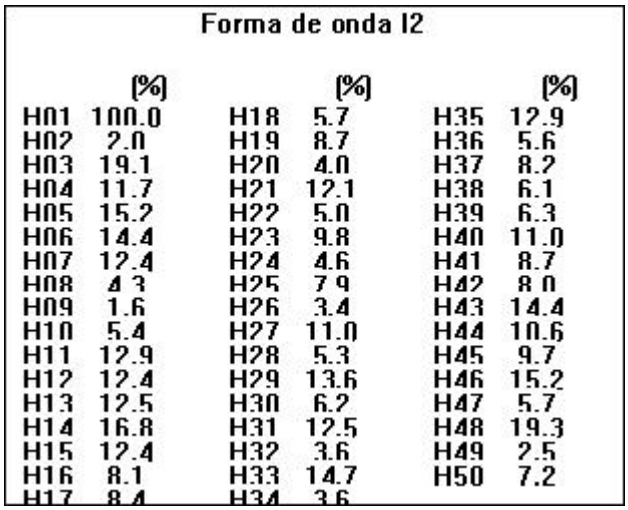


Figura 6.4.12 – Espectro de la corriente del filtro en régimen, sin filtrar



	(%)		(%)		(%)
H01	100.0	H18	5.7	H35	12.9
H02	2.0	H19	8.7	H36	5.6
H03	19.1	H20	4.0	H37	8.2
H04	11.7	H21	12.1	H38	6.1
H05	15.2	H22	5.0	H39	6.3
H06	14.4	H23	9.8	H40	11.0
H07	12.4	H24	4.6	H41	8.7
H08	4.3	H25	7.9	H42	8.0
H09	1.6	H26	3.4	H43	14.4
H10	5.4	H27	11.0	H44	10.6
H11	12.9	H28	5.3	H45	9.7
H12	12.4	H29	13.6	H46	15.2
H13	12.5	H30	6.2	H47	5.7
H14	16.8	H31	12.5	H48	19.3
H15	12.4	H32	3.6	H49	2.5
H16	8.1	H33	14.7	H50	7.2
H17	8.4	H34	3.6		

Figura 6.4.13 – Contenido armónico de la corriente del filtro en régimen, sin filtrar

Vale destacar que en estas condiciones, o sea en régimen, el VSI consume una corriente con fundamental de 0,54A rms (ver figura 6.4.12). Esto es aceptable dado que nuestro sistema es capaz de controlar corrientes de 15A de pico con el VSI (pues esto es lo que es posible medir). Es importante aclarar que los valores de corriente del filtro propiamente dicho (conjunto VSI-transformador) son aproximadamente la mitad de los anteriormente mencionados, debido al factor del transformador (220/115).

6.4.6. Prueba de filtrado

La última prueba realizada y la más importante de todas, dado que es el objetivo principal de este proyecto, fue la del filtrado de armónicos.

Como carga no lineal, generadora de armónicos, se utilizó un triangulo constituido en cada rama por una resistencia y un dimmer. Las distintas pruebas se hicieron sobre la misma carga. Se probó la mejora que brinda el ajuste de fase para los armónicos. Como así también se probó el filtrado con ganancias intermedias.

Corrientes de carga

En la figura 6.4.14 se muestra la corriente de carga generada. En la figura 6.4.15 se puede ver a la derecha los espectros de cada fase de la corriente de carga y a la izquierda la amplitud de cada armónico medida en Ampere RMS. Se puede ver que las señales están desbalanceadas en cuanto a la fundamental y al contenido armónico, dado que la amplitud de estos varía sustancialmente de una fase a otra, pero igualmente estamos hablando de amplitudes muy pequeñas.

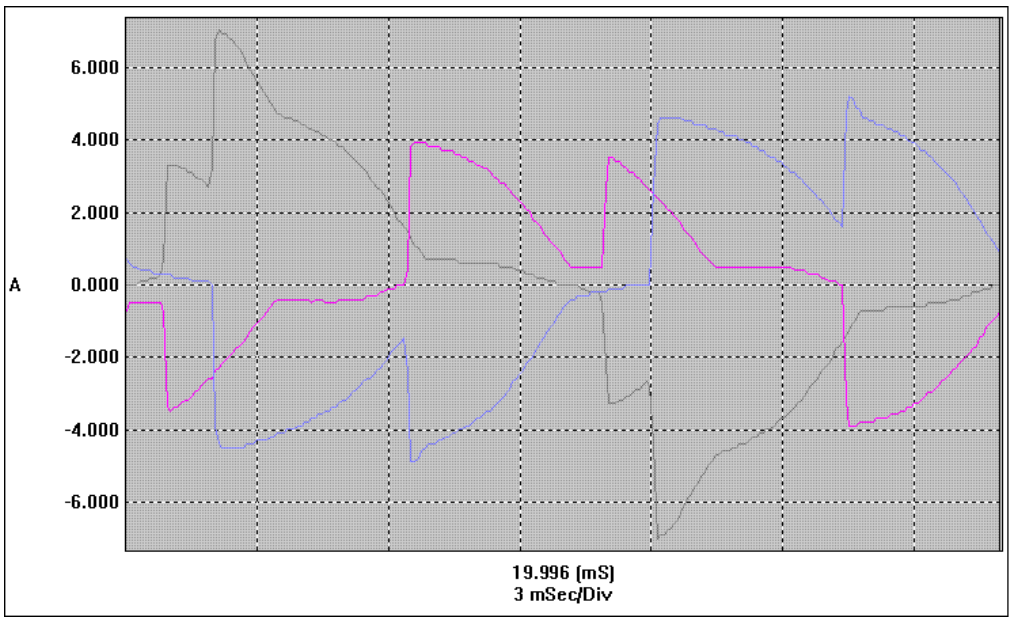


Figura 6.4.14 – Corrientes de carga

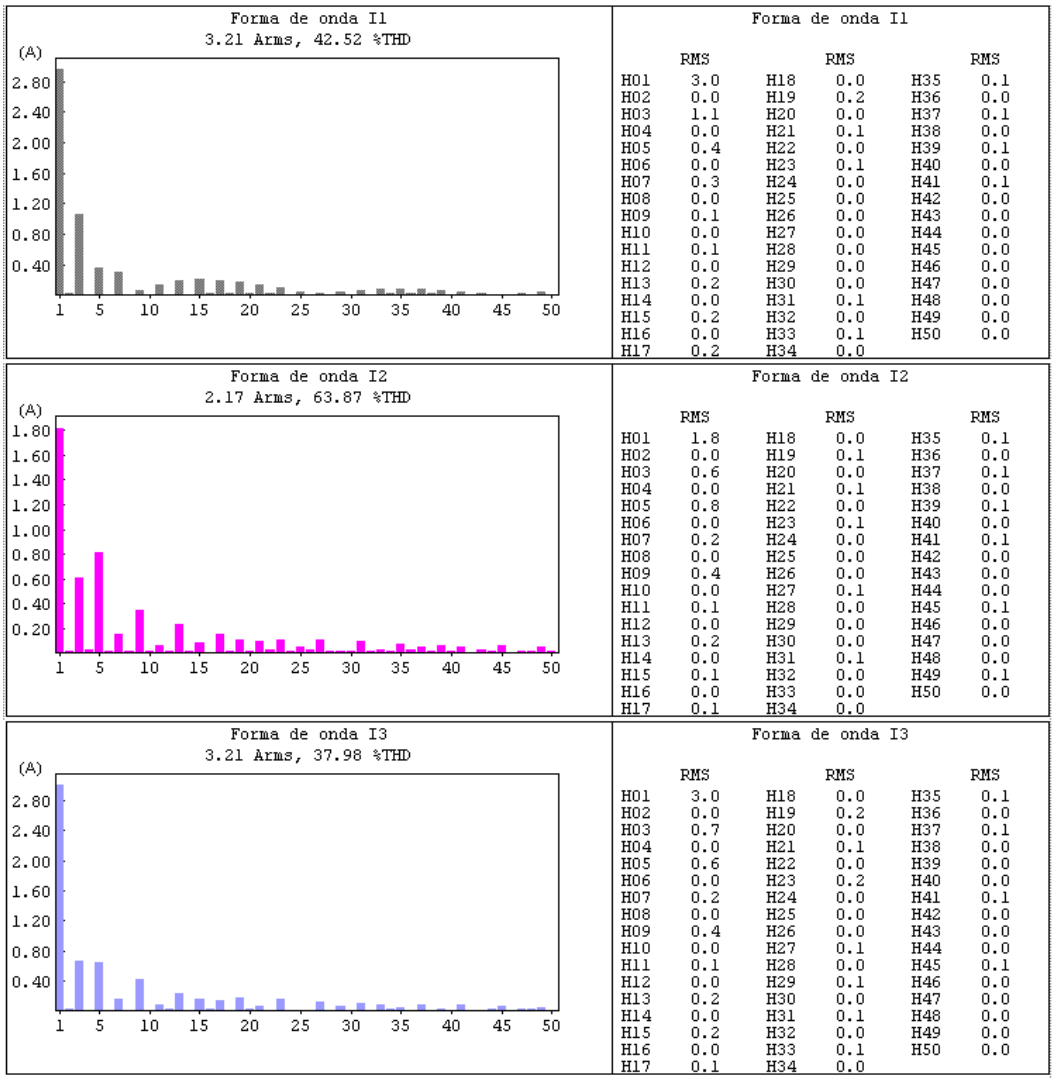


Figura 6.4.15 – Espectros de las corrientes de carga, con medida de armónicos, en orden R S T

Filtrado de secuencias sin ajuste de fase

La primera prueba de filtrado fue tratar de filtrar los armónicos 3º, 5º, 7º y 13º, en todos los casos secuencias positivas y negativas, todas las ganancias en 1 y sin ajuste de fase de los armónicos, en total 8 secuencias armónicas. En la figura 6.4.16 se puede observar la corriente de línea resultante de esta prueba. Se puede ver una mejora en cuanto a la fundamental frente a los armónicos, pero los saltos bruscos que presenta al corriente de carga aún se pueden notar en la corriente de línea. El ruido que se adiciona es debido a la conmutación de las llaves, y es inevitable dado que es un ruido que se presenta en una banda amplia de frecuencia, esto se vio ya en las simulaciones (sección 5.4.2 Simulaciones, ver figura 5.4.3), y también en las pruebas antes realizadas.

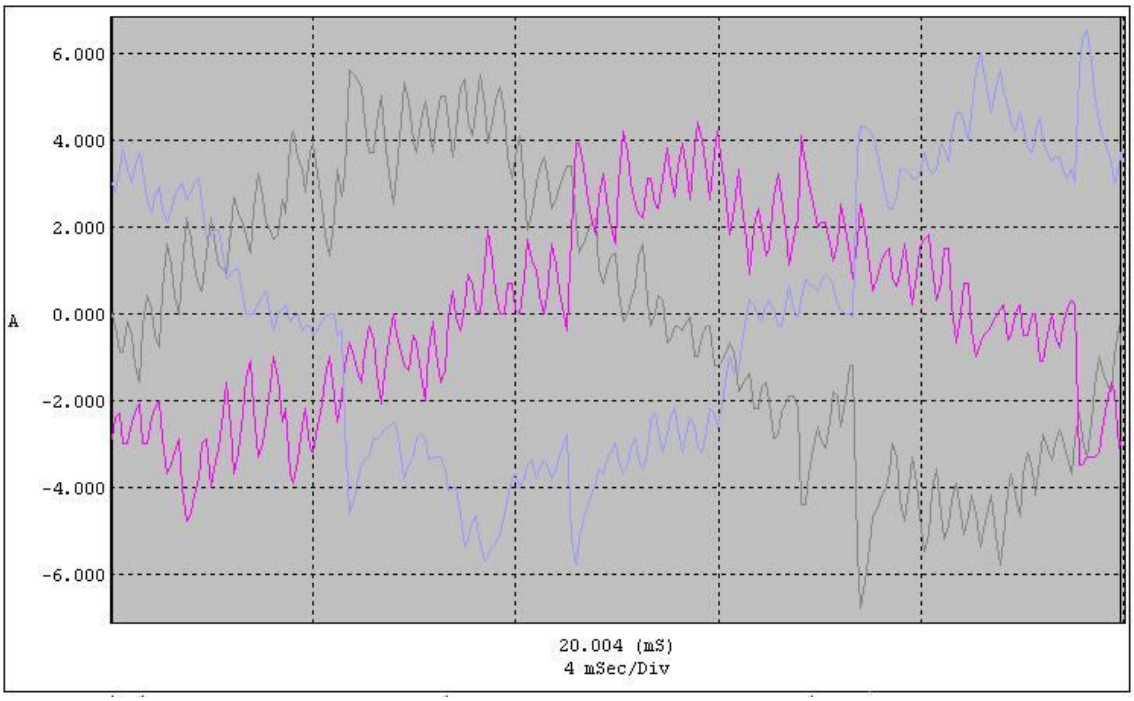


Figura 6.4.16 – Corrientes de línea, sin ajuste

En la figura 6.4.17 se presenta los espectros del las corrientes de línea, y la medida de los armónicos. Se nota un aumento en la amplitud de la corriente fundamental, este es debido al consumo de potencia del filtro para mantener cargado el condensador. Se puede observar la disminución de la amplitud de las secuencias filtradas. En la tabla 6.4.1 se muestran las corrientes de las secuencias filtradas tanto del lado de la carga como del lado de la red, se muestra el porcentaje de variación. Se puede apreciar que hay una notoria reducción de las amplitudes de los armónicos, aunque el filtrado es con ganancia 1 igualmente no se logra filtrar al 100%, pero igualmente el resultado sin ajuste es bueno de todas formas. Vale aclarar que la resolución del instrumento no es la mejor para esta prueba (0,1A rms), y que las amplitudes de los armónicos son muy pequeñas, por tal motivo es que en los armónicos 7 y 13 se ven reducciones al 0% y otras al 50%. La resolución del instrumento de medida es del orden de las señales medidas, las posibles soluciones para este problema son, o cambiar el instrumento o cambiar la carga para así manejar corrientes más grandes. De todas formas se asume que los resultados obtenidos son aceptables.

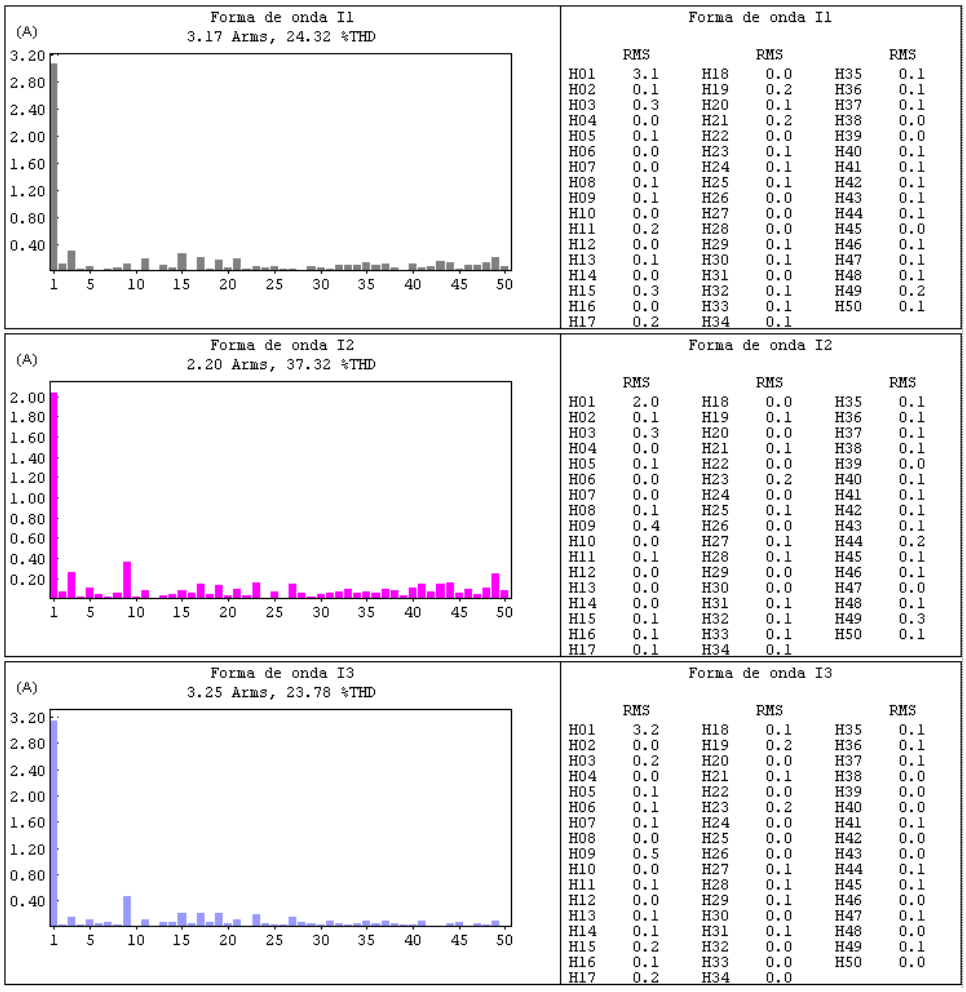


Figura 6.4.17 - Espectros de las corrientes de línea, con medida de armónicos, sin ajuste, en orden R S T

	1º	3º	5º	7º	13º
Fase R carga	3,0	1,1	0,4	0,3	0,2
Fase S carga	1,8	0,6	0,8	0,2	0,2
Fase T carga	3,0	0,7	0,6	0,2	0,2
Fase R línea	3,1	0,3	0,1	0,0	0,1
Fase S línea	2,0	0,3	0,1	0,0	0,0
Fase T línea	3,2	0,2	0,1	0,1	0,1
Variación R	103%	27%	25%	0%	50%
Variación S	111%	50%	12%	0%	0%
Variación T	106%	28%	16%	50%	50%

Tabla 6.4.1 – Comparación de corrientes de carga y de línea

Filtrado de secuencias con ajuste de fase de 3er armónico

Luego de esta prueba, se procedió a tratar de ajustar la fase del filtrado de la 3ª secuencia, positiva y negativa. En la figura 6.4.18 se pueden ver los espectros de las corrientes de línea luego del ajuste de fase. Se puede notar que hay una disminución del 3er armónico, es más notoria en la fase R, en la cual la amplitud del armónico se redujo de 0,3A a 0,1A, lo cual es un buen resultado dado que se hizo simplemente corriendo -5 grados la fase del PLL para ambas secuencias. En cuanto a las otras fases, la fase S cayó de 0,3A a 0,2A, y la fase T permaneció en el mismo valor de 0,2A. En teoría con el ajuste de fase se podrían lograr filtrar las secuencias al 100%, pero en la práctica solo se puede llegar a un máximo de filtrado, el cual es debido al error en las medidas, las conmutaciones de las llaves, cosas que hacen que el sistema se aleje de la idealidad. En definitiva se considera que se puede llegar hacer un

ajuste en el cual se reduzcan más los armónicos, pero la carga que poseemos no es la adecuada como para hilar tan fino.

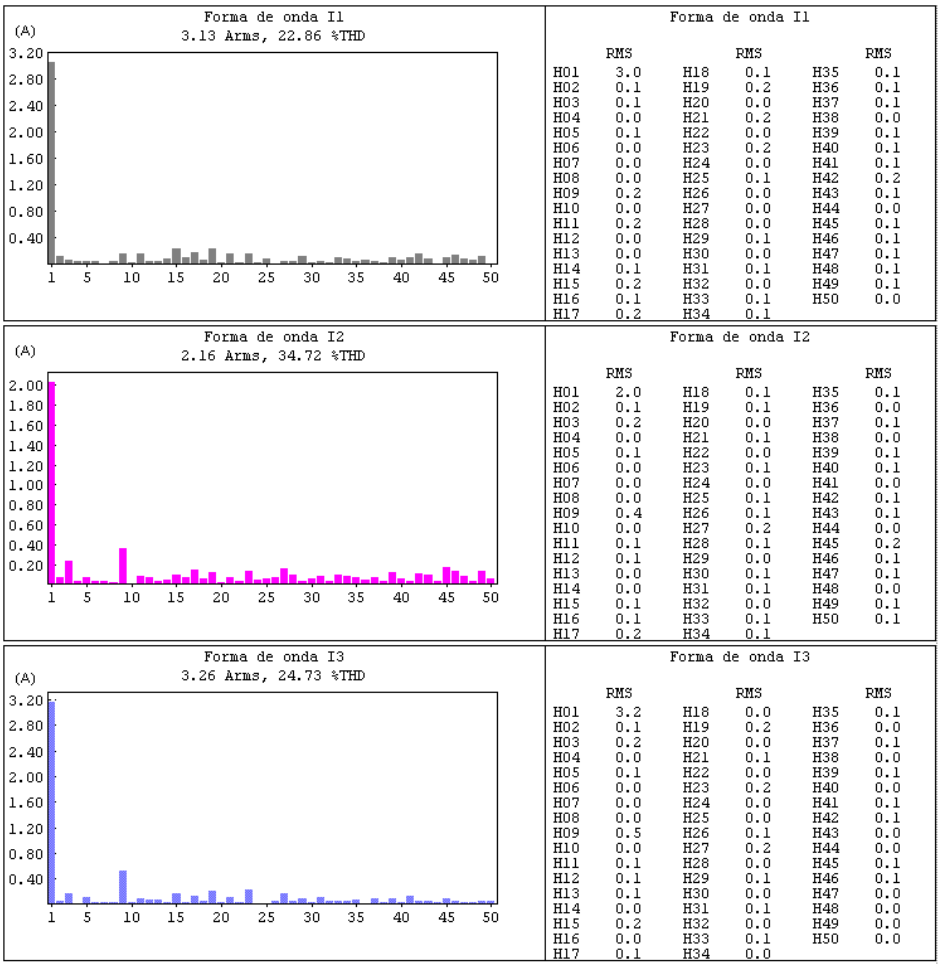


Figura 6.4.18 – Espectros de las corrientes de línea, con medida de armónicos, con ajuste, en orden R S T

Filtrado con ganancia intermedia del 5º armónico

Por ultimo, otra prueba que se llevo a cabo fue la de filtrar un armónico con una ganancia intermedia, en este caso 50%. El sentido de esto se encuentra entre los objetivos de este proyecto, la idea no es filtrar los armónicos al 100%, sino cumplir las normativas existentes sobre los límites de generación de armónicos.

Con la misma carga, se bajó la ganancia de filtrado de las secuencias positiva y negativa del 5º armónico, obteniendo como resultado el espectro que se muestra en la figura 6.4.19. En la tabla 6.4.2 se presenta la comparación entre la corriente de línea y de carga. Nuevamente si observamos los números, se puede ver que hay atenuación, pero en si la medida tiene muy poca resolución como para afirmar que esta funcionando adecuadamente. En si el resultado es bueno, en la fase R se ve una diferencia con lo esperado, pero en las fases S y T, el resultado es el ideal.

	5º
Fase R carga	0,4
Fase S carga	0,8
Fase T carga	0,6
Fase R línea	0,3
Fase S línea	0,4
Fase T línea	0,3

Tabla 6.4.2 – Resultados de filtrado con ganancia intermedia

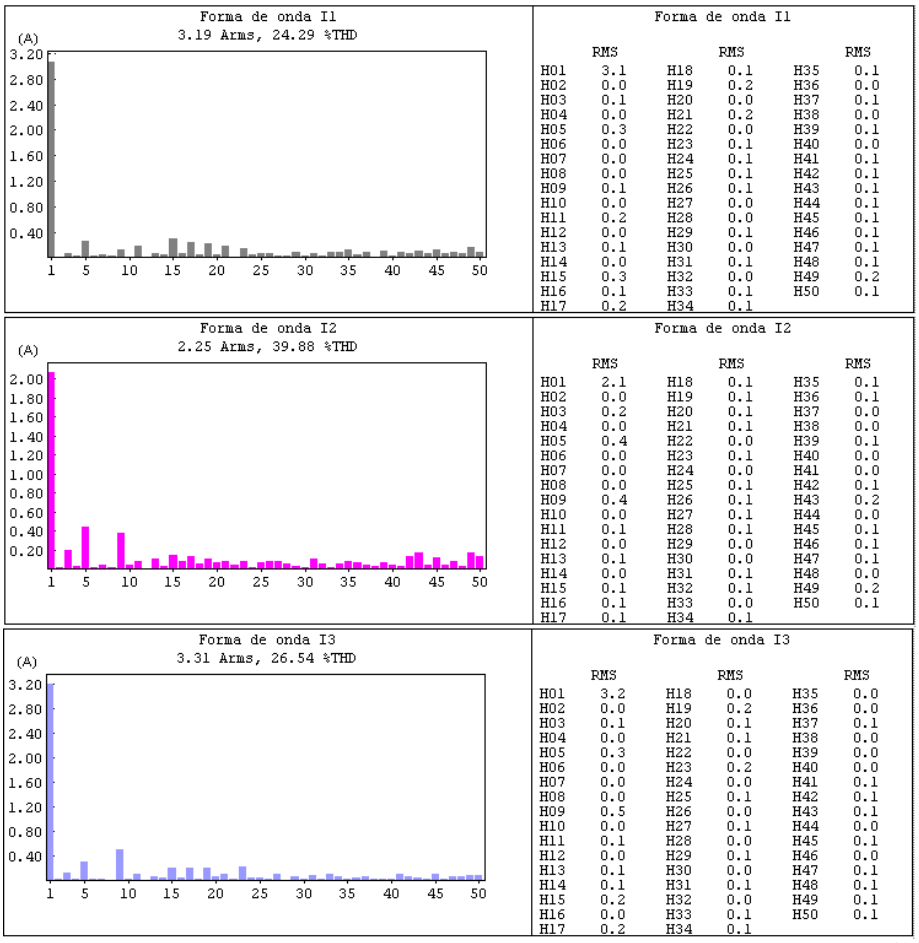


Figura 6.4.19 - Espectros de las corrientes de línea, con medida de armónicos, con ajuste en 3er y ganancia intermedia en 5º armónico, en orden R S T

Observaciones generales

Es importante mencionar que el transformador que utilizamos consumía una corriente no despreciable en vacío (0,8A, 0,5A y 0,7A en las fases R, S y T respectivamente), que además presentaba armónicos. Específicamente, en el tercer armónico se tienen 0,2 y 0,1A en las fases S y T, y se tienen 0,1A de 5º armónico en las fases R y T.

En cuanto a la potencia consumida por todo el sistema (es decir, la asociada a I_L) obtuvimos los resultados que se observan en la tabla 6.4.3. para distintas situaciones (la carga no es la que se vio en las pruebas anteriores)

	P (W)	Q (VAr)
Sólo transformador en vacío	40,6	258,2
Sólo carga y transformador	649	910
Con filtro sólo controlando Vdc	735	914
Con filtro filtrando	757	820

Tabla 6.4.3 – Potencias activa y reactiva

Se observa que el VSI consume unos 735W - 649W = 86W solo para controlar la tensión en el condensador. Así mismo, cuando se comienza a filtrar secuencias armónicas, las pérdidas ascienden a 108W, debido a que se deben manejar corrientes mayores. Éstas producen un aumento en las pérdidas en llaves y conductores.

La reactiva no se incrementa sustancialmente cuando el filtro empieza a controlar la tensión DC, mientras que gracias al filtrado, esta se reduce. Las conclusiones de esto dependen de cómo mide la reactiva el instrumento utilizado (dada la presencia de armónicos). Es de suponer que la Q que éste aparato mide debe depender de la THD, por lo que al reducirse el contenido armónico mediante el filtrado, la reactiva también disminuye.

En cuanto a los tiempos de ejecución en la tabla 6.4.4 se observa que se tiene un tiempo de ejecución mínimo de 15,6 μs, en el cual se ejecutan el PLL, el control de tensión (junto al filtrado residual) y control vectorial. Por cada secuencia que se quiere filtrar, la ejecución demora 3 μs más, aprox. La función de atención a la interfaz gráfica puede durar hasta 2,3 μs en cada período de muestreo.

Los tiempos se midieron de la misma forma que en la sección 5.4.1.

Cantidad de sec. armónicas a filtrar	**Tiempo de ejecución (μs)**
0	15,6
1	18,5
2	22,0
3	24,5
8	40,0

Tabla 6.4.4 – Tiempos de ejecución

6.5. Descripción del Firmware

Se describe a continuación el funcionamiento del firmware del DSP. Al igual que en el firmware del hardware in the loop, todo el código del DSP se implementó en lenguaje C. No se vio necesario hacer ninguna optimización en assembler.

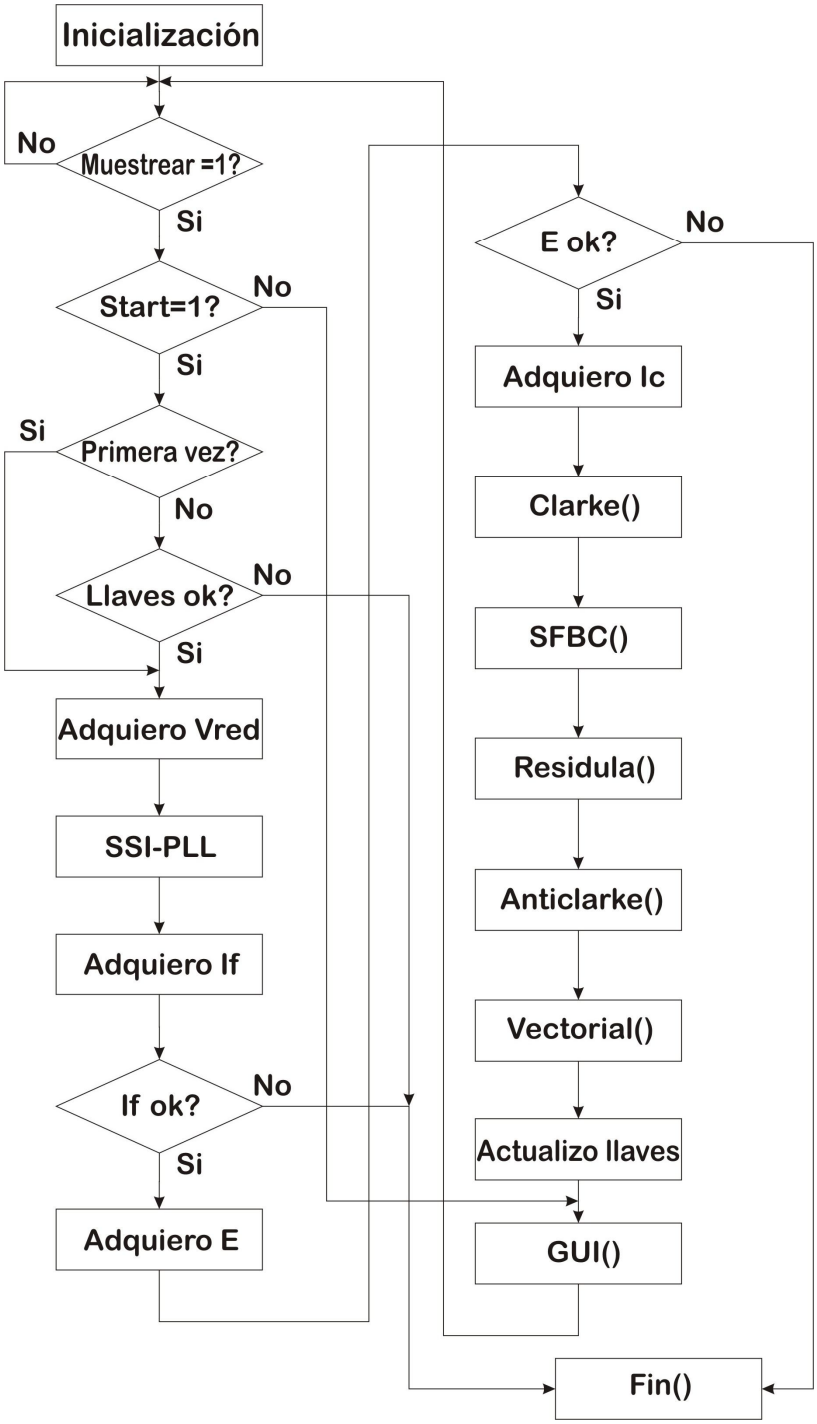


Figura 6.5.1 – Diagrama de flujo del firmware del DSP

En la figura 6.5.1 se muestra un diagrama simplificado del firmware del DSP. Antes de entrar al loop principal, se inicializan todas las variables y periféricos. La variable *muestrear* arranca en 0, por lo cual no se entra inmediatamente al loop. Esta variable es llevada a 1 en la rutina de atención a la interrupción del timer. Luego se hace un polling de *muestrear* para determinar si hay que correr el programa o no. Cuando se chequea *muestrear* si vale 0 se

vuelve a chequear, si vale 1 se sigue adelante, y se le da el valor 0. Tenemos la variable *start*, con la cual se habilita el filtro. Esta es modificada desde el GUI. Si *start* vale 0 se invoca la función de atención a la comunicación, *GUI*. En caso contrario se verifica si esta es la primera vez que se entra al filtro, si es así, se salta la protección de verificación del estado de las llaves, si no es la primera vez se verifica el estado de las llaves. Si el estado de las llaves es el correcto se sigue adelante en caso contrario, se invoca la función *fin*, y se desactiva la salida del buffer hacia el VSI. Si todo va bien se arranca lo que sería el filtro. En el diagrama de flujo no se muestra, pero antes de arrancar realmente con el filtro se arranca con la salida de la placa de protección deshabilitada, por lo tanto, todo va a correr normalmente pero las señales de control no llegan al VSI.

En definitiva después de la verificación del estado de las llaves se prosigue a adquirir las tensiones de la red, para así luego correr el PLL. Se adquieren las corrientes del filtro y se verifica que el módulo de estas no sea mayor a 14A. Se adquiere la tensión en el condensador y se verifica que no sea mayor a 365V, pero la protección se activa sólo si se verifica esta condición en dos instantes de muestreo consecutivos. Si todo va bien, se adquiere la corriente de carga, y se prosigue a correr todas las funciones del filtro, transformada de Clarke, luego las celdas selectivas, sólo si alguna de las ganancias asociadas a los arreglos de los armónicos es distinta de cero. Después se hace el filtrado residual y el control de tensión del condensador, luego la antitransformada para generar las entradas del control vectorial, con la salida de esta última, se actualiza el estado de las llaves y por ultimo se invoca a la función de atención al GUI. Se pasa nuevamente al loop de espera por la interrupción, cuando ésta llega y modifica *muestrear* se comienza el mismo ciclo.

Todas las protecciones llevan a invocar a la función *fin*, para así luego desactivar la salida del buffer que alimenta los drivers de las llaves. Si esta situación se da, se recomienda, no volver a tratar de correr el filtro, sin una verificación exhaustiva de lo ocurrido.

6.6. Tarjeta de adaptación de señales

6.6.1. Introducción

El inversor del laboratorio de Electrónica de Potencia ha sido controlado para distintos propósitos en los últimos años mediante una PC. La conexión entre ambos se lograba a través de una tarjeta, que aquí llamamos **tarjeta de protección**. Desde esta tarjeta la PC podía, mediante una tarjeta de adquisición interna (PC-LPM-16), tomar medidas de distintas tensiones y corrientes, además de enviar y recibir señales digitales. La mayor importancia de esta radica en el PLD que posee, que básicamente asegura que nunca se cierren dos llaves de una misma rama del inversor a la vez. Otro elemento importante es el watchdog, implementado con otros integrados, independientes del PLD. La documentación de esta placa se encuentra en [10].

A efectos de poder utilizar el inversor como filtro activo de 3 ramas, fue necesario fabricar una tarjeta de interfaz, que permita comandar el inversor a través del kit de desarrollo del DSP. De esta forma se suplanta la PC existente por el DSP.

Dado que en el transcurso del proyecto se plantearon nuevos objetivos (específicamente la implementación del SSI-PLL y la interfaz gráfica), el tutor del proyecto aceptó que ayudara el Ing. Fernando Chiaramello (participante del proyecto PDT 47/14) en la implementación del hardware faltante para el control del inversor con el kit DSP. Su ayuda se centró en el diseño y fabricación del impreso de la nueva tarjeta. Sin embargo él también asistió en el diseño de algunas partes de ella. Además llegó a realizar una documentación de una primera versión de la placa. Por otro lado, el montaje y soldadura de los componentes, así como el testeo y ajuste corrió por cuenta de los integrantes de este proyecto. Luego, también fue necesario realizar una placa para poder utilizar un transductor de tensión de efecto hall, la cual fue implementada por Chiaramello.

6.6.2. Requerimientos

La nueva tarjeta de interconexión, que aquí se llama **tarjeta de medidas**, debe:

- Adaptar señales analógicas que llegan a la tarjeta existente de interconexión con el VSI.
- Adaptar señales analógicas provenientes de otros transductores.
- Adaptar señales digitales que van del DSP al PLD y viceversa.
- Proporcionar la alimentación a la tarjeta de interconexión, proveniente de una fuente externa.

Las medidas analógicas que requiere el DSP para controlar el filtro activo son:

- Corrientes de filtro, que llegan a la tarjeta de protección desde celdas de efecto hall (IFR, IFS y IFT)
- Corrientes de la carga, desde pinzas de corriente (ICR, ICS y ICT)
- Tensiones de red, que llegan a la tarjeta de protección desde transformadores (VR, VS, y VT)
- Tensión del condensador del bus de continua, desde un transductor de efecto Hall, LV-20P, marca LEM.

En cuanto a señales digitales, por un lado el DSP debe enviar:

- comandos de las 3 ramas
- señal para el watchdog
- una señal de habilitación

Por otro lado, debe recibir el estado de las 6 llaves a través de 6 entradas digitales.

Las señales que la tarjeta de protección intercambiaba con la computadora en el proyecto [10], pero no son utilizadas por el DSP, se dejan en un conector aparte (señales auxiliares). Finalmente se terminó usando una de ellas, como se verá más adelante.

La tarjeta de medida de tensión continua requiere alimentación externa de +12V y -12V, mientras que la tarjeta de protección requiere además +5V. La tarjeta de medida les brinda esa alimentación a partir de una fuente externa.

Por último incorporamos a la tarjeta de medidas dos LEDs, los cuales son comandados por el DSP.

6.6.3. Detalles de la tarjeta de medidas

A continuación se repasa la implementación de los distintos bloques dentro de la placa, la cual se muestra a la derecha en la figura 6.6.1. Las señales entran y/o salen de la placa a través de los conectores, explicados en el anexo E.

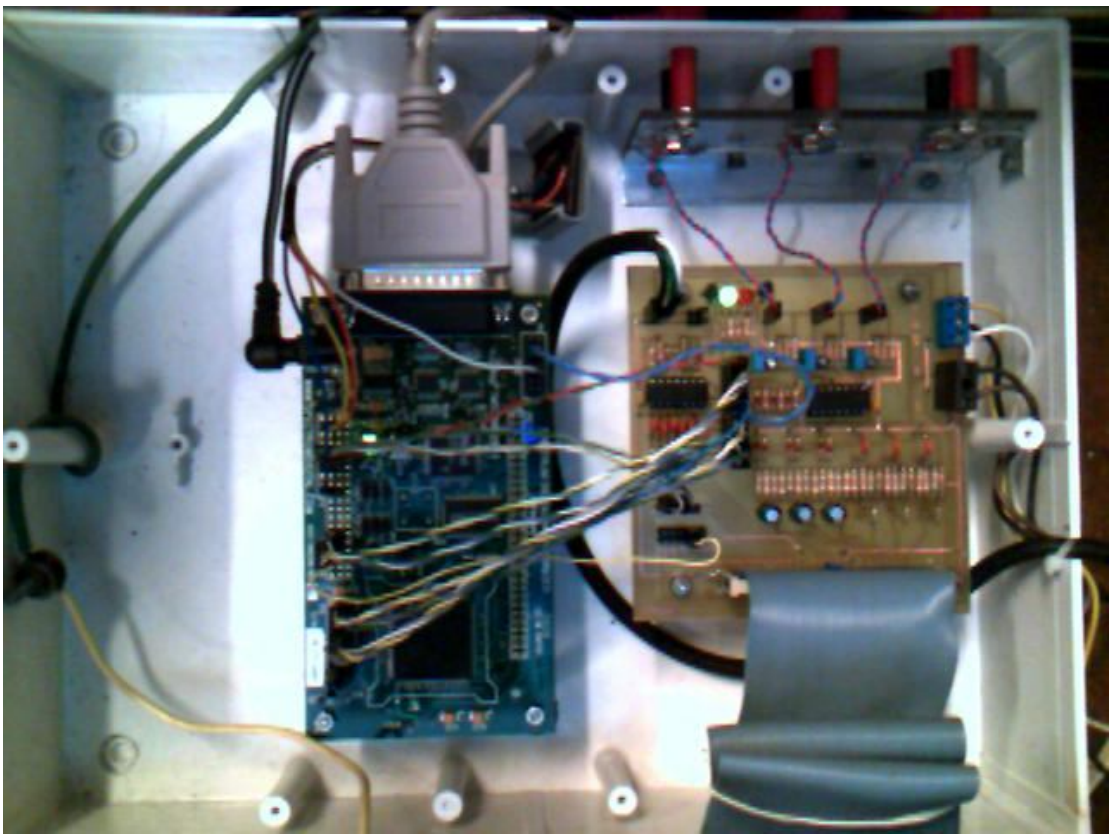


Figura 6.6.1 – Foto del kit DSP (izquierda) y la placa de medidas (derecha)

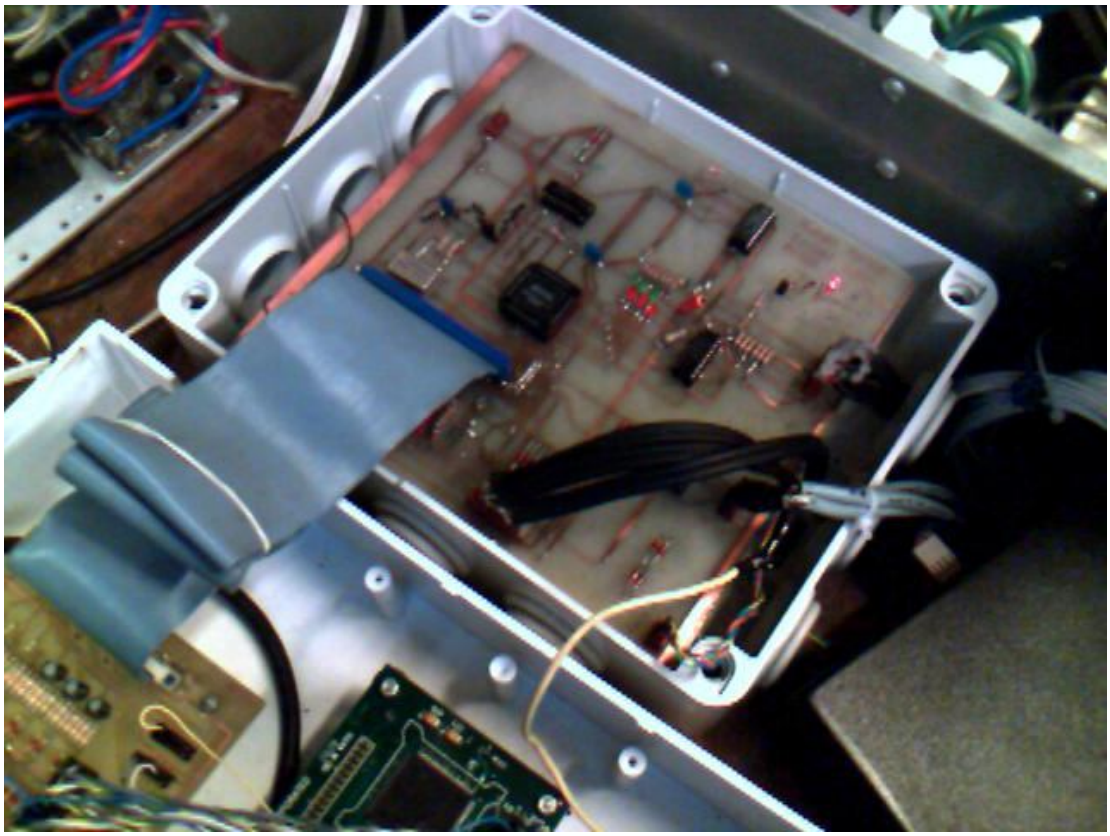


Figura 6.6.2 – Foto de la tarjeta de protección

Entradas digitales:

Las entradas digitales desde la tarjeta de protección corresponden a la realimentación de los estados de las 6 llaves del inversor. El DSP maneja niveles digitales de 0 – 3,3V, mientras que se esperaba que las señales entrantes a la tarjeta de medidas fueran de 5V. Nuestra tarjeta tiene un circuito como el de la figura 6.6.3 para cada una las señales mencionadas. Mediante diodos zener 1N5226, cuyo voltaje es de 3,3V, se adaptan los niveles de tensión a la vez que se protege el DSP si la tensión de la señal entrante sube mucho debido a alguna falla.

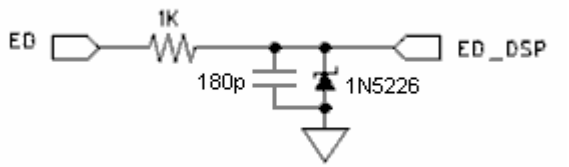


Figura 6.6.3 – Manejo de entradas digitales

Cuando se probó la señal real de entrada, se vio que ésta no fue de 5V como se esperaba, sino de 2,5V aprox., por lo que el zener nunca entra en conducción inversa. El DSP detecta (por poco) los niveles altos porque supera los 2V necesarios para ello, pero debido a que picos de interferencia podían producir un error en la lectura, se decidió colocar el condensador que se observa en la figura.

A pesar de los esfuerzos por mejorar la realimentación de los estados de las llaves, a veces se detectan errores, fundamentalmente debido a errores en las señales generadas en la electrónica del inversor.

Salidas digitales:

Los niveles de tensión del DSP y el PLD permiten conectar directamente estas señales. Esto es así porque aunque el PLD está alimentado a 5V, reconoce un "1" lógico cuando la tensión es mayor a 2V, la cual es menor a los 3,3V que genera el DSP. No existen problemas con el "0" lógico.

Las salidas digitales hacia el PLD son los estados de las tres ramas del inversor y las que denominamos señales auxiliares, de las cuales solo usamos la señal "pedido de inhabilitación". Además está el manejo del watchdog de la placa de protección, que consta de un monoestable (HEF4538B) que comanda un buffer triestado (DM74366N). Tampoco hubieron problemas al comandarlo directamente con un puerto digital del DSP.

Entradas analógicas:
Tensiones de red

Las medidas analógicas desde la tarjeta de protección están entre ±5V, los cuales deben ser transformados al rango 0 - 3V.

Las medidas de tensión son de AC. Debido a esto se implementó el circuito de la figura 6.6.4 para cada una de las tres tensiones de fase.

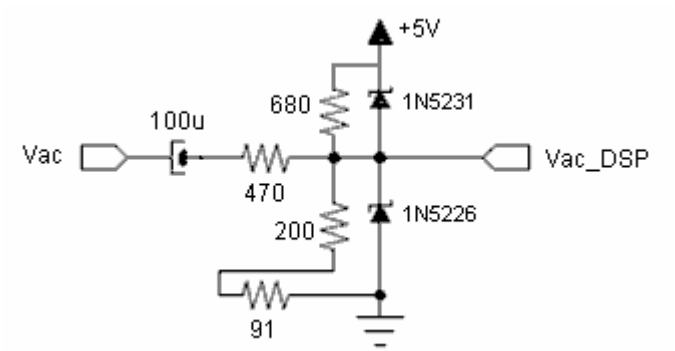


Figura 6.6.4 – Tratamiento de las señales de tensiones de la red

El condensador de desacople de continua se calculó para que el polo del pasa-altos quede por debajo de 0,5 Hz.

Con las resistencias se logró tanto la atenuación, como el agregado de un offset de 1,5V.

No fue necesario agregar condensadores para el filtrado de alta frecuencia debido a que estas señales son filtradas digitalmente en los algoritmos del DSP.

Los zeners protegen al DSP de sobretensiones, mediante la saturación de la señal cuando esta intenta salirse del intervalo -0,1V a 3,3V. El zener 1N5231 es de 5,1V mientras, el 1N5226 es de 3,3V, como se mencionó anteriormente.

Corrientes del filtro

Las corrientes del filtro se miden con celdas de efecto hall (IHA150). Las señales de medida llegan incambiadas a la tarjeta de medidas, pudiendo estar entre ±5V y tener componente continua. Por ésta última razón el circuito de adaptación es el que se muestra en la figura 6.6.5.

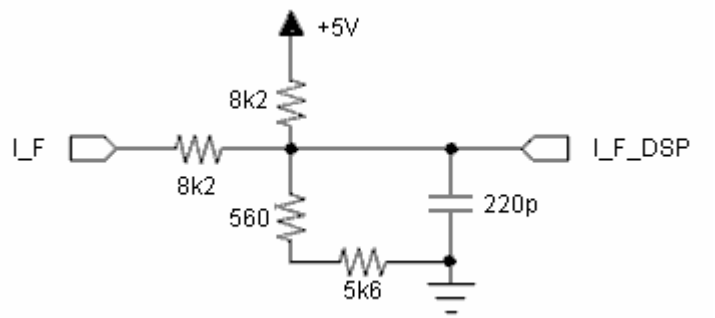


Figura 6.6.5 – Circuito de adaptación de corrientes del filtro

A diferencia del circuito anterior, no hay un desacople de continua, lo cual hace que los valores de las resistencias también sean diferentes. Aquí no hay zeners de protección, pero el DSP está protegido por zeners en la placa de protección que limitan las señales entre ±5V. En el futuro será conveniente integrar estas dos placas en una.

Se agregaron condensadores para el filtrado de altas frecuencias, debido a interferencias en la medida. Esto fue necesario dado que el DSP no realiza un filtrado digital al manejar estas señales.

Corrientes de carga

Las medidas de corriente de carga provenientes de las pinzas de corriente (AEMC MN114), las cuales generan una señal alterna con relación 10A:1V, con corriente nominal de 10A. Es así que solo se debió adaptar niveles de tensión continua para poder ser adquirida por el DSP. En la figura 6.6.6 se muestra el circuito que se repite para cada una de las tres fases.

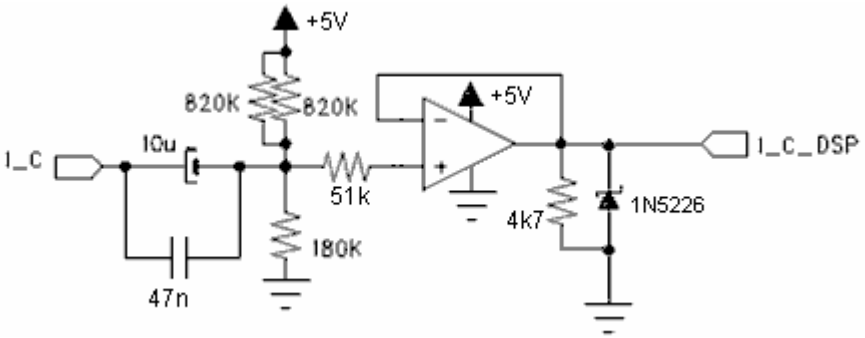


Figura 6.6.6 – Circuito de adaptación de las corrientes de carga

Debido a la impedancia mínima de entrada que debía tener el circuito (100k), se decidió utilizar el amplificador operacional LM324 como seguidor.
Las resistencias para el ajuste de la continua están a la entrada del operacional debido a que éste está alimentado entre +5V y 0V.

Fue necesario introducir resistencias para el correcto funcionamiento del integrado mencionado. Por un lado se colocó una resistencia a la entrada, debido a que se observó un comportamiento no deseado cuando la señal bajaba de -0.7V. Esto no es un defecto del integrado puesto que aquí nos estamos saliendo de las especificaciones del mismo. Con la resistencia a la entrada se evita que cuando la señal de entrada se vuelve negativa (lo que significa una amplitud muy grande en la medida), la entrada no inversora sea forzada a menos de 0V. Por otro lado, fue necesario colocar una resistencia entre la salida del operacional y tierra, puesto que este integrado se vuelve muy susceptible a ruidos cuando no está saliendo una corriente positiva a través de su salida.

La protección para que la señal hacia el DSP no se vuelva negativa la brinda la propia saturación del operacional (con la ayuda de la resistencia a la entrada), mientras que el zener 1N5226 impide una tensión mayor a los 3,3V.

Tensiones de continua

Para este proyecto se necesita la medida de la tensión en el condensador para su control. Sin embargo, como este proyecto es parte de un proyecto PDT que también abarcará el filtrado de corrientes homopolares, se previó que esta placa pueda manejar dos medidas de tensión continua y no solo una.

La medida de las tensiones continuas se realiza con un transductor de efecto Hall LV-20P (relación de transformación 25mA:10mA), los cuales miden la corriente a través de una resistencia en bornes del condensador. Esta medida, también en corriente, finalmente la pasamos a una medida en tensión que puede ir de 0 a 3V para tensiones en el rango de trabajo. Esto se logró mediante resistencia adecuadas en el primario y secundario del transductor. En la figura 6.6.7 se muestra el circuito que se colocó en el rack del inversor para la medida de una tensión continua.

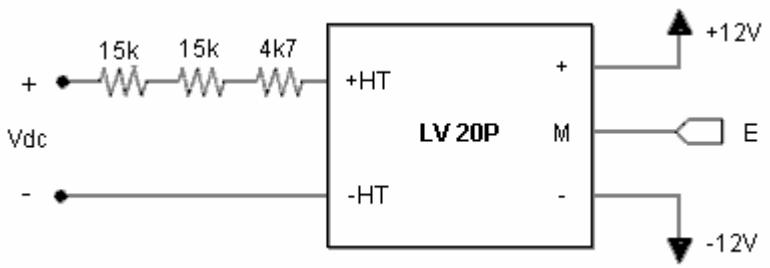


Figura 6.6.7 – Circuito del transductor de voltaje

El circuito dentro de la placa de medidas se ilustra en la figura 6.6.8. El seguidor de tensión está a efectos de no cargar la medida.

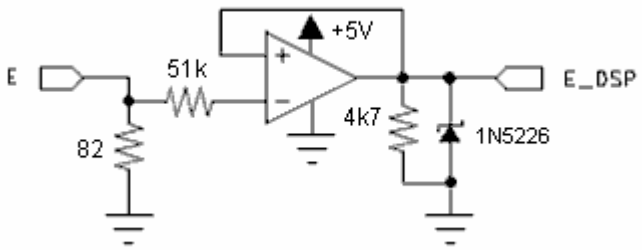


Figura 6.6.8 – Circuito para medida de tensión en la placa de medida

También fue necesario colocar aquí resistencias a la entrada y a la salida del seguidor, para su correcto funcionamiento.

LEDs

El DSP comanda dos transistores BJT para el encendido de un LED rojo y otro verde. Estos sirven para indicar el estado del filtro. En la figura 6.6.9 se observa uno de los dos circuitos.

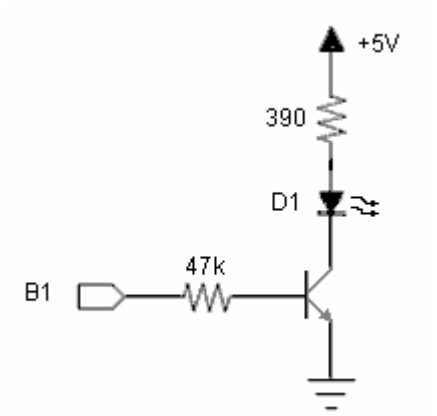


Figura 6.6.9 – Circuito de manejo de un LED

Inmunidad al ruido

El diseño del impreso trata de minimizar los ruidos en las medidas, puesto que son esenciales para el correcto funcionamiento del filtro. Se tuvo cuidado en que la tierra analógica y la tierra digital se unan, en lo posible, solo en un punto cercano a la entrada de la alimentación en la placa. La tierra digital se conecta al GND del kit DSP, pero la tierra analógica se conecta a la pata llamada V_{LO}, el cual es el cero analógico del ADC.

7. Interfaz gráfica

La interfaz grafica es una herramienta que simplifica mucho la comunicación con el DSP, dado que por medio de un display grafico se puede interactuar de manera amigable con un sistema de complejidad no despreciable.

Esta interfaz fue desarrollada en la herramienta GUIDE de Matlab. Como fuente de información para familiarizarnos con esta herramienta, se utilizó la ayuda y las demostraciones que proporciona Matlab.

Antes de ejecutar el programa de interfaz, el DSP ya debe estar corriendo y a la espera de alguna orden del GUI, dichas órdenes se enviaran por intermedio de este programa.

Cuando ejecutamos el programa aparecerá la siguiente pantalla (Figura 7.1):

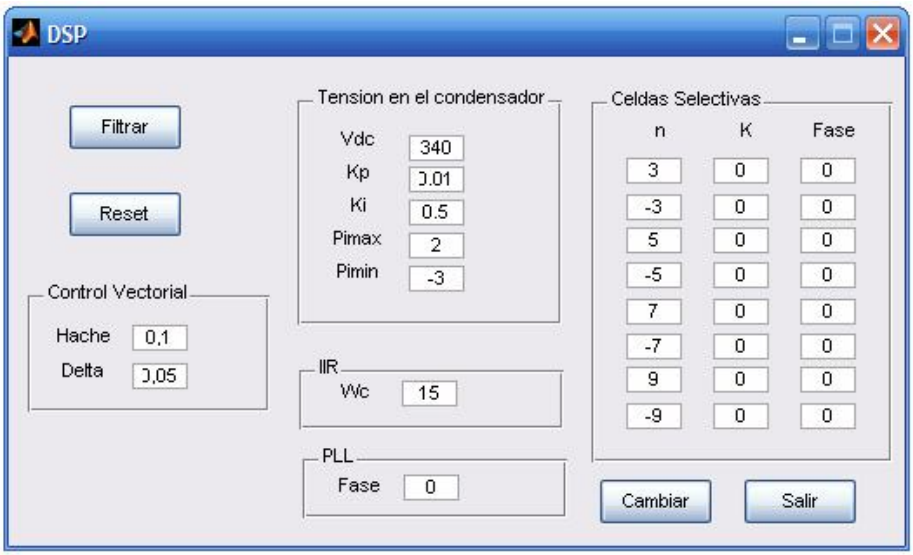


Figura 7.1 – Pantalla inicial.

Tal como se explicó en la comunicación DSP-GUI, el GUI puede cambiar los parámetros que el DSP tiene en su memoria. Dichos parámetros son los que se observan en la pantalla anterior. Los valores que se muestran son los que inicialmente tiene el DSP. Estos valores están agrupados en distintos paneles, para así poder apreciar los bloques implementados en el DSP y cuales son los parámetros que se involucran en ellos. Ellos son: Control Vectorial, Tensión en el condensador, fase de PLL y variables de las celdas selectivas, mas el ancho de banda del filtro IIR.

Cabe destacar que todos los valores cambiados van a ser verificados por el GUI, debiendo ser estos valores coherentes a enviar al DSP. Al introducir un valor en la pantalla, el GUI chequea dicho valor y si se sobrepasa de los límites admisibles de las variables se desplegará un mensaje de error en pantalla. A continuación se ve un ejemplo de cuando tratamos de imponer una tensión mayor a las especificadas. Intentamos tener una tensión en el condensador de 390 V, y los márgenes admitidos para esta son entre 200 y 350V, tal como lo muestra la figura 7.2

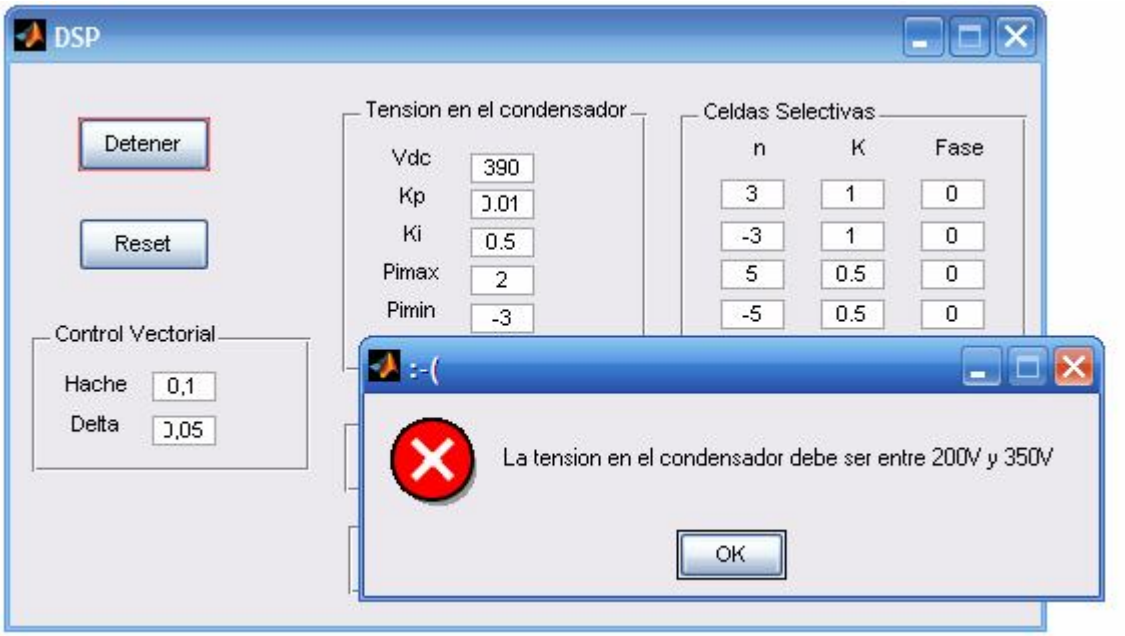


Figura 7.2 – Mensaje de error.

Lo mismo sucede en todos los parámetros, por ejemplo si se introduce en "n" un valor no entero, o una ganancia de filtrado muy grande (mayor a 2), si no respeta que el control vectorial el hache sea mayor que el delta, etc. En la tabla 8.1 se muestran todos los límites.

Parámetro	Mínimo	Máximo
Vdc	200V	350V
Kp	0	0.1
Ki	0	2
Pimax	0	3
Pimin	-4	0
Hache	0A	5A
Delta	0A	5A
Wc	0Hz	-
Fase PLL	-70º	70º
K	0	2

Tabla 7.1 – Rango de valores permitidos en el GUI

En la tabla 8.1 no se presenta ni *n* ni *fase.* En particular *n* tiene que ser un entero distinto de -1, 0 y 1, pero no tiene límite superior ni inferior. En cuanto a *fase,* no se le impusieron límites, pero debería tener un valor razonable.

Una vez cambiados los parámetros de interés, debemos enviárselos al DSP, esto se realiza con el botón "Cambiar". Cuando se presiona dicho botón, el GUI enviará al DSP solo la información de aquellos valores que han sido modificados y aceptados por el GUI, es decir que han respetado los limites establecidos. La modificación del ancho de banda del filtro debe ser realizada con todas las ganancias de las celdas en 0, dado que si se realiza la modificación con el sistema filtrando, en el transitorio luego del cambio se generan corrientes de referencia elevadas, llevando al sistema a activar una protección por sobre corriente del filtro. Por tal motivo si se trata de cambiar el ancho de banda del filtro IIR con alguna ganancia distinta de 0, se despliega un mensaje de error y el cambio no se lleva a cabo.

Vale aclarar que en todos los mensajes de error se especifica la causa que lo generó.

Cada vez que desde el GUI se cambia un parámetro en el DSP, inmediatamente después se chequea que el valor que recibió el DSP fue el enviado, esto se hace por medio de

una consulta que se activa después de enviado el parámetro. En caso de obtener algún tipo de error, el GUI avisara con otro mensaje de error, indicando dicho problema.

Hasta ahora se han guardado en el DSP todos los valores modificados, entonces ya es hora de filtrar, esto se hace presionando el botón “Filtrar”. Aquí el GUI envía al DSP un código, el cual el DSP interpreta que es hora de filtrar. Inmediatamente el botón que antes decía “Filtrar” ha pasado a decir “Detener” tal como se puede observar en la siguiente pantalla (Figura 7.3).

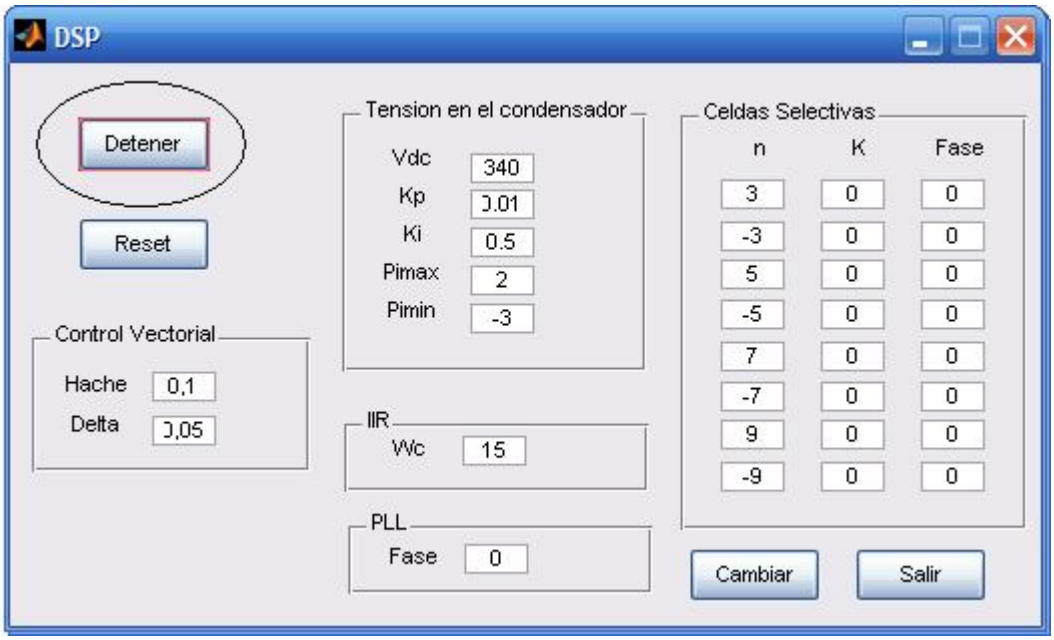


Figura 7.3 – Botón Detener

Si se presiona este botón se detendrá al DSP, quedando guardados todos los valores que ya se tenían.

Debajo de este botón se observa el botón “Reset”. Cuando se presiona, se envía al DSP la información original que se tenía en pantalla. Estos valores son actualizados en la pantalla también. Este botón se puede presionar solo si el sistema no esta filtrando, si se trata de hacer un reset con el sistema filtrando se despliega un mensaje de error y no se hace el reset.

Para salir de la aplicación se presiona el botón “Salir”, el cual cierra la comunicación con el DSP, quedando este con los últimos valores que se enviaron. El DSP puede quedar filtrando, o sea se puede terminar la comunicación pero el DSP puede seguir filtrando igualmente. El caso opuesto también está contemplado, este es cuando el DSP está filtrando y se abre la aplicación, aquí el GUI cuando carga consulta si el DSP está filtrando, en caso de estar en esta condición, ya se pone el botón “Filtrar/Detener” en “Detener”.

8. Conclusiones

En la primera etapa del proyecto se le dedicó un buen tiempo a la comprensión de la teoría pq, así como al manejo de la herramienta de simulación Simulink. Gracias a que se desarrolló la simulación del filtro activo, se logró una familiarización aún mayor con la teoría y con algunos aspectos prácticos que escapan a esta.

En la etapa del Hardware-in-the-Loop se tuvo el primer contacto con el DSP, pero finalmente se desarrolló y probó prácticamente todo el firmware que luego se utilizaría para controlar el sistema real. Este método de trabajo fue fundamental para un desarrollo seguro y flexible de los algoritmos, dado que se iba a manejar un costoso equipo de electrónica de potencia.

Ya en la última etapa, los mayores problemas se encontraron en la interconexión del kit DSP con el inversor. En particular, el correcto manejo de las señales analógicas fue algo muy importante, pero no se había tenido en cuenta en etapas anteriores. En cuanto a los algoritmos de los distintos bloques del filtro, se comprobó que éstos funcionaban adecuadamente. Esta etapa tuvo los agregados de un nuevo PLL, el cual se implementó sin mayores inconvenientes, y la interfaz gráfica, que aunque era un objetivo opcional, mostró tener una gran utilidad práctica a la hora de cambiar parámetros en tiempo real.

Respecto a la verificación experimental de la teoría de filtrado selectivo, se logró superar con éxito todos los problemas de implementación real, considerando las necesarias protecciones y limitaciones practicas de las diferentes tecnologías asociadas.

Aunque este proyecto se da por terminado aquí, las pruebas para este sistema son inacabables, en particular variando los parámetros y manejando otros tipos de carga. Es más, en el Instituto queda una herramienta de desarrollo muy potente, donde se podrán probar (y se probarán) otros algoritmos de filtrado, de control de corriente, etc., con diferentes topologías en el circuito de potencia, lo cual no es un aporte menor.

Si se hace una abstracción y se observa este proyecto, se puede ver que la tarea que más se aprecia es la programación en un DSP de ciertos algoritmos bien conocidos. Pero si se va al detalle, se puede observar que se abarcaron distintas áreas de la ingeniería eléctrica:

- Procesamiento de señales: es evidente
- Potencia: el área de aplicación del proyecto
- Control: hay un par de lazos de control (PLL y control de Vdc)
- Electrónica: se desarrolló una placa y se estudió el funcionamiento de otra
- Instrumentación: se tuvo especial cuidado en el manejo adecuado de las medidas, protegiendo a su vez al DSP
- Informática: se desarrolló tanto software (interfaz gráfica) como firmware (DSP).
- Telecomunicaciones: se desarrolló un protocolo de conmutación entre el DSP y la interfaz gráfica

Para poder abordar este proyecto en toda su extensión fue de gran importancia la etapa de planificación, pero además esto se logró mediante el trabajo integrado e interdisciplinario, realizándose una adecuada subdivisión del proyecto para así alcanzar las metas planteadas.

9. Bibliografía

[1] Hardware-in-the-loop.
http://en.wikipedia.org/wiki/Hardware-in-the-loop. Extraído en octubre de 2007.

[2] Librería IQmath, de Texas Instrument.
http://focus.ti.com/docs/toolsw/folders/print/sprc087.html. Versión de marzo de 2003.

[3] Proyecto de fin de carrera: "Generación Distribuida - Conexión a la red de un Inversor" – Gastón Rivoir, Diego Giacosa, Renzo Biardo. Junio 2007.

[4] Serial Communication Interface (SCI) Reference Guide
http://focus.ti.com/lit/ug/spru051b/spru051b.pdf. Noviembre 2004.

[5] C281x C/C++ Header Files and Peripheral Examples
http://focus.ti.com/docs/toolsw/folders/print/sprc097.html. Septiembre 2007.

[6] Hoja de datos del integrado MAX232.
http://rocky.digikey.com/WebLib/Texas%20Instruments/Web%20data/MAX232,232I.pdf.
Febrero 2007.

[7] Tesis doctoral: "Filtros activos selectivos de corrientes armónicas" – Gonzalo Casaravilla (Agosto 2003)

[8] Detección de tensiones de frecuencia fundamental y secuencia positiva
http://www.tdx.cbuc.es/TESIS_UPC/AVAILABLE/TDX-0303105-152111//05Prc05de09.pdf.
Octubre 2007.

[9] Voltage sags (dips) and swells
http://www.powerstandards.com/tutorials/sagsandswells.htm. Octubre 2007.

[10] Proyecto de fin de carrera: "Control Vectorial de un Motor de Inducción (II)" – Ruben Méndez, Ariel del Pino Daniel Cohn. Febrero 1999.

[11] TMS320F28x Analog-to-Digital Converter (ADC) Peripheral Reference Guide
http://focus.ti.com/lit/ug/spru060d/spru060d.pdf. Noviembre 2004.

[12] TMS321x281x DSP System Control and Interrupts Reference Guide
http://focus.ti.com/lit/ug/spru078d/spru078d.pdf. Octubre 2004.

[13] "Analysis and Comparison of Phase Locked Loop Techniques for Grid Utility Applications" – L. R. Limongi, R. Boji, C. Pica, F. Profumo and A. Tenconi. Power Conversion Conference – Nagoya 2007.

[14] Curso "Filtros Activos de Corrientes Armónicas" – Docente responsable: Dr. Gonzalo Casaravilla (Facultad de Ingeniería, Universidad de la República)

[15] "Instantaneous reactive power compensator comprising switching devices without energy storage components" - Akagi, H., Kanazawa, Y. and Nabae, A. (1984).

A. Programa de prueba del Hardware-in-the-loop

Para probar la interacción entre el Simulink y el DSP a través del puerto serie se hizo un programa principal en Matlab (.m). Al correrlo, lo primero que hace es la inicialización del puerto serie de la computadora y de los parámetros de la simulación. Luego, éste llama a la simulación en Simulink. Cuando esta termina siguen unas líneas de código para terminar la comunicación (en particular cerrar el puerto).

La inicialización del puerto consta de las instrucciones:

```
global s1
```

Se crea una variable global, pues va a ser usada por otras funciones.

```
instrreset;
```

Se desconectan y borran todos las variables asociadas a instrumentos. Esta instrucción la usamos para resetear el puerto serie, para que luego podamos asociar la variable s1 al puerto serie sin importar si había otra variable "ocupándolo" antes.

```
s1 = serial('COM1','BaudRate',19200,'Timeout',1)
```

La variable s1 pasa a ser una estructura asociada al puerto serie 'COM1'. Se configuran explícitamente la velocidad de transmisión y el tiempo de espera en cada lectura. Las otras propiedades se dejan configuradas por defecto. Si se escribe esta sentencia sin punto y coma aparece un listado de las distintas propiedades del puerto COM1 (que no son otra cosa que campos de la estructura s1).

Nota: para tener más información sobre s1 se puede escribir: get(s1)

```
fopen(s1)
```

Se "abre" el puerto, de forma que ahora se puede recibir y mandar datos a través del puerto. El buffer de entrada está vacío (BytesAvailable = 0).

Luego de cargar otras variables propias de la simulación el programa está listo para llamar a la simulación, la cual está en un archivo de Simulink (.mdl). El llamado se realiza mediante la función `sim`.

A.1. Simulink

Antes que nada hay que entender como funciona el Simulink. Para simplificar, vayamos a nuestro caso, donde la simulación en Simulink está configurada en tiempo discreto, a paso fijo. Cuando empieza a correr la simulación, el Simulink establece el tiempo inicial y algunos valores iniciales para las variables. Con esos datos empieza a realizar las operaciones en orden, es decir, partiendo de la fuentes de datos ("sources"). Cuando se calculó todas las variables, se pasa al siguiente paso de simulación, incrementando el tiempo de simulación en un cierto valor constante (por ser paso fijo). De esta forma se vuelven a realizar las operaciones partiendo de las fuentes, las cuales pueden haber cambiado para este nuevo paso. De la misma manera, también se utilizan algunos de los valores de las variables en el paso anterior (por ejemplo al integrar o al existir "retardos"). Cuando todas las variables quedan calculadas se aumenta el tiempo en un paso y así sucesivamente. La simulación termina cuando se llega al tiempo máximo de simulación. Si la simulación fue llamada por un programa de Matlab, es recién en este punto que éste sigue ejecutando las siguientes líneas de código.

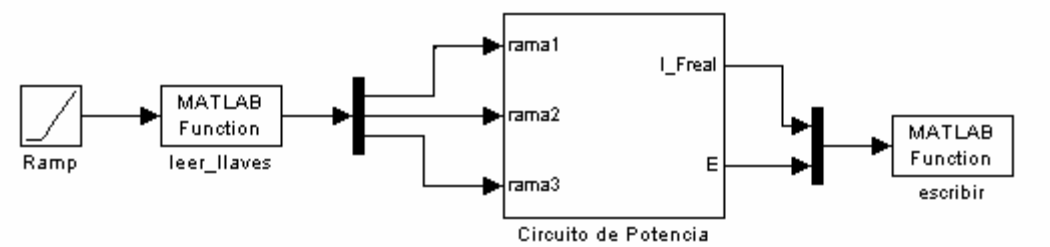


Figura A.1.1 – Esquema de simulación en Simulink (simplificado) en primera prueba.

En este caso, nuestra simulación consta de un inversor de tensión con fuente de tensión ideal alimentando una carga pasiva. El control de las llaves las realiza el DSP, pasándole las posiciones a la simulación a través del puerto serie. El Simulink calculas las corrientes y la tensión de continua (que es constante) y se las envía al DSP. El DSP lee estos valores pero no hace nada con ellos.

La simulación se divide en tres bloques. El primer y el último bloque son los bloques "Matlab Fcn", los cuales realizan un llamado a una función de Matlab cada uno. Estas funciones están en archivos ".m", realizados por nosotros, y se encargan de la comunicación con el DSP. El bloque intermedio es el sistema de potencia antes descrito.

El primer bloque llama a una función que realiza la lectura de un 'uint8' (entero de 8 bits sin signo) del COM1. Como salida de esta función están las posiciones de las tres llaves. Estas valen 1 o 0 y son los tres bits menos significativos del byte que leyó del puerto. Para leer el puerto se usa la función `fread`. A esta función se le pasan como parámetros: la variable asociada al puerto (s1), la cantidad de valores a leer y el tipo de dato que se espera leer (ej, uint8, int16, char, etc). `fread` saca del buffer del puerto serie la cantidad de valores deseada, pero si no hay tal cantidad, espera un cierto tiempo ('Timeout'). Mientras `fread` espera, toda la simulación queda detenida, pues la simulación espera a tener algún valor en la salida del primer bloque para continuar operando. En definitiva, no se avanza al siguiente paso de simulación, por lo que al circuito de potencia simulado le es indiferente el tiempo que se demore en leer datos del puerto. El único efecto es una mayor duración del tiempo de ejecución de la simulación.

Nota: en el primer paso de simulación no se realiza ninguna lectura, sino que la salida del bloque es una posición cualquiera de las llaves.

Luego de que se tienen las posiciones de las llaves en ese paso de simulación, el Simulink procede a calcular las variables del circuito eléctrico, el cual está discretizado (a la frecuencia de simulación) gracias al bloque "PowerGUI" de la librería SimPower Systems de Simulink.

Las variables calculadas que se eligieron (corrientes y tensión de continua) pasan después al bloque de escritura, que como se mencionó llama a una función escrita en Matlab. Esta función se encarga de ordenar los datos que entran en paralelo al bloque para luego transmitir por el puerto serie los datos con la función `fwrite`. En cuanto a "ordenar" los datos nos referimos a definir el tipo de los datos (en este caso usamos entero de 16 bits con signo, int16) y el orden en el que se van a escribir, valga la redundancia (quién va primero, segundo, etc.).

En resumen, para cada paso de simulación el Simulink recibe datos del DSP, procesa la información, envía los nuevos datos al DSP y espera a que éste también reciba, procese y "conteste".

A.2. DSP

El programa que corre en el DSP fue escrito en C en el ambiente de desarrollo Code Composer Studio (CCS) que vino con el kit de desarrollo. En CCS el desarrollo sea realiza mediante proyectos (.pjt). Al abrir uno de estos archivos aparecen listados todos los archivos

asociados al programa que se quiere grabar y correr en el DSP. Los archivos de interés son: el código del programa (archivo .c), archivos de definición de cabezales (.h), archivos para la organización de la memoria (.cmd) y librerías (.lib).

En el código del programa, escrito en el lenguaje C, inicialmente debe configurarse el DSP, mediante la escritura a registros. Para esto simplemente utilizamos un programa de ejemplo. Algunas variables a setear: System Control registers, PLL, WatchDog, Clocks, PIE vector table, CPU and PIE interrupts, SCI, etc. Casi toda la configuración que venía en el ejemplo no necesitaba ser cambiada

Luego de la inicialización el programa entra en un loop infinito. Al igual que en Simulink, en cada iteración, el DSP espera a que le manden datos, luego maneja los datos y enseguida los envía. Aquí no hay manejo de timers ni de interrupciones.

Veamos parte del código:

```
llaves = 1;       // Estado de las llaves (tres bits menos
significativos)
contador = -1;    // Se inicializa un contador para que no se lea en el
                  // primer paso
for(;;)     // Se entra al loop infinito
{
      Ir = rx_int16();  // Se llama varias veces a una función que
      Is = rx_int16();  // lee dos bytes y lo guarda en una variable
      It = rx_int16();  // de tipo int16. Se deben cargar los datos
      E = rx_int16();   // en el orden que el Simulink los envió

      if (contador == 32)     // Al pasar 32 pasos de simulación, es
                              // decir, veces que se leyeron datos,
      {                       // hay un cambio de posición
                  switch (llaves) {
                  case 1: llaves = 5; break;    // Acá se cambia el
                  case 5: llaves = 4; break;    // estado de las
                  case 4: llaves = 6; break;    // llaves teniendo en
                  case 6: llaves = 2; break;    // cuenta la posición
                  case 2: llaves = 3; break;    // actual
                  case 3: llaves = 1; break;
                  }
                  contador = 0;
      }

      contador++; // Solo se incrementa el contador luego de que se
                  // realizó una lectura todas las variables
      SciaRegs.SCITXBUF = llaves;   // Se escribe en el buffer de
salida
      SciaRegs.SCITXBUF = contador; // la posición de las llaves y el
                                    // contador. Inmediatamente el DSP
                                    // envía los datos.
}
```

Hay que aclarar que lo que se lee del buffer de entrada son bytes, por lo tanto en la función de lectura se tienen que acomodar los datos para que sean int16, Uint16, IQN, etc. El formato IQN es el que usamos para leer señales continuas (fue estudiado más tarde).

Para leer un solo byte:

```
while(SciaRegs.SCIFFRX.bit.RXFIFST !=1) { }     // wait for XRDY =1
                                                // for empty state
ByteRecibido = SciaRegs.SCIRXBUF.all;
```

B. Elección del DSP

B.1. Introducción

Existe actualmente en el mercado un abanico muy amplio de fabricantes, líneas y modelos de DSP's. Para desarrollo se presentan distintas alternativas, Starter Kits y placas de desarrollo, estas últimas un tanto costosas. Por lo cual se hace muy compleja la correcta elección del mismo.

Entre todos los fabricantes se destacan en ventas y desarrollo Texas Instruments, Analog Devices y Freescale (ex Motorola).

Este estudio se basará en un DSP específico, el TMS320F2812 de Texas Instruments. Para este DSP se escogerá una placa de desarrollo o un Starter Kit.

Se comprobará que este DSP se ajusta a las necesidades del proyecto. Desde el punto de vista hardware, o sea las operaciones a ejecutar, frecuencia de muestreo. También la aplicación del DSP en proyectos del estilo, cantidad de información disponible, etc.

B.2. Elección del DSP

El tutor de este proyecto está en contacto con un grupo de investigación brasileño, dedicado a las aplicaciones de electrónica de potencia en los Sistemas Eléctricos de Potencia (SEP). Este grupo ha desarrollado filtros utilizando el DSP TMS320F2812, por lo cual han obtenido una gran experiencia y mucha documentación de la misma. Este último es el argumento más fuerte que se manejo para la elección.

B.3. Tareas del DSP

El estudio que sigue se presenta tal como fue documentado en el primer informe intermedio de este proyecto, en el cual buscó estimar tiempos de ejecución. Como se veía y tal como se verá en el capítulo 5 fueron consideraciones conservadoras pero consistentes con el resultado final.

El DSP trabajará en dos modalidades:

- Conectado a un PC para realizar simulaciones haciendo una interfaz con Simulink.
- Conectado a un inversor de tensión para implementar el filtro real.

En la primera modalidad, el PC envía datos al DSP, el DSP opera y retorna la información al PC. En este caso no hay restricciones de tiempo, ya que no se estará trabajando en tiempo real.

En la segunda modalidad, se estará trabajando en tiempo real. El DSP tomará muestras a una cierta frecuencia y operará entre muestras.

En la figura B.1. se ilustran ambas modalidades. Los pulsos indican quien esta operado en ese momento, TX es la transmisión de datos del PC al DSP y RX la recepción por parte del PC. El tiempo en que opera el PC puede ser variable, en cambio el del DSP es fijo. En la segunda modalidad T representa el tiempo de operación del DSP y T_s el tiempo entre muestras.

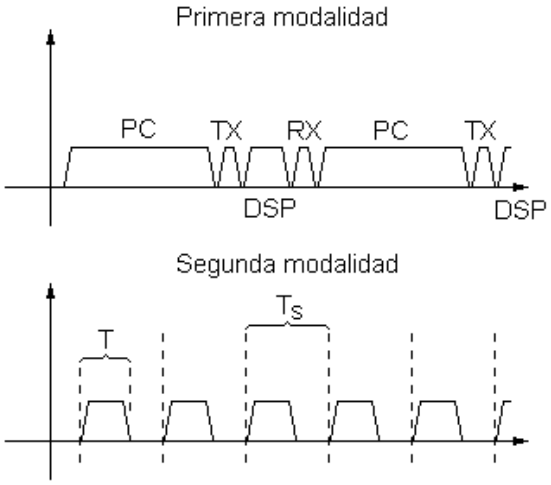


Figura B.1. – Modalidades de trabajo

Claramente, es más restrictiva la segunda opción, dado que esta en juego la variable tiempo. Lo que tenemos que estimar es que el tiempo de operación con los datos es menor al tiempo entre muestras, en la figura B.1 estos tiempos son T y T_s, respectivamente.

B.4. Estimación de la cantidad de operaciones

El firmware a implementar va a tener cinco grandes bloques: programa principal, celda selectiva, PLL, control del inversor y control de la tensión del condensador.

Programa principal.

Este bloque no es muy critico desde el punto de vista del tiempo de ejecución. Las funciones principales que desarrollará son la de inicializar variables y la de invocar los demás bloques.

Se desarrollará en lenguaje C, dado que no parece necesario optimizarla en assembler.

Este bloque debe llamar a las celdas de filtrado, al controlador del inversor y al controlador del condensador.

A las celdas selectivas se le pasaran dos parámetros, y la celda devolverá un puntero a los resultados. Luego con los resultados tendrá que hacer un par de sumas. En total tenemos seis *moves* y dos sumas por celda.

Al controlador del inversor solo le tendrá que pasar las tres muestras de corriente del filtro, lo cual se realiza por medio de un puntero, entonces son dos *moves*.

Para el controlador del condensador solo se pasa la muestra de la tensión del condensador.

Bloque	***Operaciones***	***Comentarios***
Celda selectiva	• 2 sumas • 6 moves Aprox. 8 operaciones	Esto es para una sola celda, y considerando que las transformadas de Clarke se hacen en la celda.
Control del inversor	• 2 moves Aprox. 2 operaciones	El controlador del inversor no retorna nada.
Control del condensador	• 2 moves Aprox. 2 operaciones	El controlador del condensador no retorna nada.

Tabla B.1. – Operaciones estimadas para el programa principal

En condiciones normales, para filtrar una secuencia armónica se estarían llamando a tres celdas, la celda del armónico y dos celdas de filtrado residual. Entonces, en total tenemos 28 operaciones para una celda.

Dentro del programa principal también podemos considerar la adquisición de las muestras de corriente y de tensión. En total son 5 muestras, el tiempo por cada una es de 80ns, entonces el tiempo de adquisición de muestras rondara en los 400ns, pero será menor ya que el DSP posee 2 ADC con 16 entradas analógicas.

Celda selectiva.

La celda es la que realiza todo el filtrado. Es uno de los bloques mas críticos, en primera instancia será desarrolla en C pero para la aplicación de tiempo real se tendrá que optimizar en assembler.

Este bloque lleva a cabo varias operaciones:

- Transformada de clarke
- Modulación, calculo de P y Q
- Filtrado IIR
- Cambio de signo
- Demodulación
- Antitransformada de clarke

En la tabla B.2. se muestran las estimaciones del orden de operaciones.

Tarea	***Operación***	***Estimación***	***Comentarios***
Transformada de clarke	$\begin{pmatrix} i_0 \\ i_\alpha \\ i_\beta \end{pmatrix} = \sqrt{\frac{2}{3}} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix} \begin{pmatrix} i_R \\ i_S \\ i_T \end{pmatrix}$	• 5 multiplicaciones • 3 sumas • aprox. 14 move Aprox 22 operaciones	No tomamos en cuenta la homopolar. Entonces solo se utilizan la fila 2 y 3.
Modulación	$\begin{pmatrix} p \\ q \end{pmatrix} = \begin{pmatrix} sin(\omega_c t) & -cos(\omega_c t) \\ -cos(\omega_c t) & -sin(\omega_c t) \end{pmatrix} \begin{pmatrix} i_\alpha \\ i_\beta \end{pmatrix}$	• 4 multiplicaciones • 2 sumas • aprox. 12 move Para el seno y coseno: • 6 multiplicaciones • 10 sumas • aprox. 20 move Aprox. 54 operaciones	Para el cálculo de las señales trigonométricas se utilizara una tabla e interpolación lineal.
Filtro IIR	$A(1)y(n) = B(1)x(n) + B(2)x(n-1) + B(3)x(n-2) - A(2)y(n-1) - A(3)y(n-2)$	• 5 multiplicaciones • 5 sumas • 17 move Aprox. 27 operaciones	Se realizan dos filtrados por celda. El filtro que se utiliza es un pasa altos, luego se hace 1 menos lo obtenido para lograr el filtrado pasa bajo.
Cambio de signo	$\tilde{p}(n) = -p(n)$ $\tilde{q}(n) = -q(n)$	• 2 multiplicaciones • 8 move Aprox. 10 operaciones	
Demodulación	$\begin{pmatrix} \tilde{i}_\alpha \\ \tilde{i}_\beta \end{pmatrix} = \begin{pmatrix} \sin(\omega_c t) & -\cos(\omega_c t) \\ -\cos(\omega_c t) & -\sin(\omega_c t) \end{pmatrix} \begin{pmatrix} \tilde{p} \\ \tilde{q} \end{pmatrix}$	• 4 multiplicaciones • 2 sumas • aprox. 12 move Aprox. 18 operaciones	Aquí se utiliza el seno y coseno calculado en la modulación.

Tarea	*Operación*	*Estimación*	*Comentarios*
Anti-transformada de clarke	$\begin{pmatrix} \tilde{i}_R \\ \tilde{i}_S \\ \tilde{i}_T \end{pmatrix} = \sqrt{\frac{2}{3}} \begin{pmatrix} \frac{1}{\sqrt{2}} & 1 & 0 \\ \frac{1}{\sqrt{2}} & -\frac{1}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & -\frac{1}{2} & -\frac{\sqrt{3}}{2} \end{pmatrix} \begin{pmatrix} 0 \\ \tilde{i}_\alpha \\ \tilde{i}_\beta \end{pmatrix}$	• 5 multiplicaciones • 2 sumas • 13 move Aprox. 20 operaciones	La homopolar vale 0.

Tabla B.2. – Operaciones estimadas para las celdas selectivas

En total se tienen 178 operaciones. Esto es haciendo una estimación aproximada, seguramente en la práctica el número de operaciones sea mayor. Los factores que alteraran esta estimación son, por ejemplo, el pasaje de parámetros a las subrutinas, operaciones con el stack, etc. Igualmente estas operaciones se desarrollaran en assembler para lograr el mejor aprovechamiento posible del tiempo.

Las operaciones en el DSP demoran un ciclo de reloj en ejecutarse. La frecuencia del reloj es de 150MHz, o sea 6,67ns de periodo, lo que hace un total de unos 1,2us por celda, aproximadamente. La frecuencia de muestreo que se piensa utilizar es de 12,8kHz (78us), esto implica que se podrán filtrar un buen número de secuencias armónicas.

Observaciones:

- No se tomo en cuanta que la implementación en assembler utilizará paralelismo. Lo cual aumenta la eficiencia del DSP, pudiendo realizar varias operaciones a la misma vez.
- La transformada y antitransformada de clarke se hace solo una vez por muestra. Cuando se filtran varias secuencias armónicas no es necesario realizar las transformadas entre secuencias, solo se hace la transformada al principio y la antitransfromada al final del filtrado de todas las secuencias. Entonces para una celda normal estaríamos hablando de unas 136 operaciones.

PLL

El PLL es utilizado para determinar la frecuencia de la red. Con múltiplos de la frecuencia obtenida, se calcula la frecuencia de las señales necesarias para la modulación y la demodulación, en una celda, para un armónico dado.

El algoritmo es muy simple y no posee un gran número de operaciones. Básicamente hace un filtrado IIR de la señal, para disminuir el ruido, y luego con los cruces por cero determina la frecuencia de la misma.

Las operaciones que realiza son las que corresponden al filtro, comparaciones para determinar los cruces por cero, el incremento de una variable (para generar un diente de sierra).

Tarea	*Operación*	*Estimación*	*Comentarios*
Filtro IIR	$A(1)y(n) = B(1)x(n) + B(2)x(n-1) + B(3)x(n-2) - A(2)y(n-1) - A(3)y(n-2)$	• 5 multiplicaciones • 5 sumas • 17 move Aprox. 27 operaciones	Se realizara un filtrado de segundo orden, con frecuencia de corte del orden de los 70Hz.
Algoritmo	Case state: 0: if v(i)*v(i-1)<=0	Peor caso: • 4 jumps • 5	Se realizara un filtrado de segundo orden, con frecuencia de corte

	... 1: if v(i)*v(i-1)<=0 ... if t >= ts + 1/(N*f) ... 2: if v(i)*v(i-1)<=0 ... if t >= ts + 1/(N*f) ...	comparaciones • 6 multiplicaciones • 7 sumas • 40 moves aprox. Aprox. 62 operaciones	del orden de los 70Hz. El algoritmo se implementa con una sentencia *case*, lo cual, a priori, hace muy difícil estimar las operaciones que se necesiten para esta acción.

Tabla B.3 – Operaciones estimadas para el PLL

A simple vista, en la tabla B.3., parece que este bloque no va a implicar un gran número de operaciones. Pero puede que en el desarrollo del proyecto presente modificaciones para optimizarlo desde el punto de vista de la estabilidad, de la velocidad de respuesta, etc, lo cual incrementara el numero de operaciones a ejecutar. Supondremos que el orden de operaciones será el mismo. En este caso tenemos aproximadamente 90 operaciones, unos 600ns, entonces esperamos que este algoritmo ronde el micro segundo de ejecución.

Control del Inversor

El inversor a manejar es del tipo VSI (o inversor de tensión). El control del inversor consiste en generar una tensión de salida, que conectada a la red por medio de una impedancia, genere la corriente del filtro deseada. Esto significa que a parir de la corriente del filtro, la impedancia de conexión y la tensión de la red, vamos a calcular la tensión de salida del inversor.

Tarea	***Operación***	***Estimación***	***Comentarios***
Calculo de Tensión	$V_F = V_L + L \cdot f_s \cdot \left(i_F(n) - i_F(n-1)\right)$	• 2 suma • 2 multiplicaciones • 12 moves Aprox. 16 operaciones	Esto para una sola fase. Entonces son 48 operaciones.
Conmutación	-Comparación de corrientes -Conmutación	• 1 suma • 1 comparación • 6 moves Aprox. 8 operaciones	Esto para una sola fase. Entonces son 24 operaciones.

Tabla B.4. – Operaciones estimadas para el control del inversor

Estas estimaciones se refieren control Bang Bang, para control vectorial se necesitan más operaciones. A partir de la tabla B.4. concluimos que se necesitan unas 72 operaciones, o sea unos 480ns aproximadamente, para realizar el control del inverso. Aquí no se tuvo en cuenta que se puede hacer una verificación del estado de las llaves.

Control de la tensión del condensador

El control de la tensión del condensador se hace por medio de un consolador PI. La entrada del controlador es la diferencia entre un set point y la tensión en bornes del

condensador. La salida del controlador se suma con la potencia de la celda de filtrado residual de secuencia positiva.

Tarea	*Operación*	*Estimación*	*Comentarios*
Calculo de error.	$e[n] = E - V_C[n]$	• 1 suma • 6 moves Aprox. 7 operaciones	
Controlador PI	$\text{int}[n] = \text{int}[n-1] + K_I\left(\frac{e[n]+e[n-1]}{2f_s}\right)$ $V_{Opi}[n] = K_P e[n] + \text{int}[n]$	• 3 sumas • 4 multiplicaciones • 18 move Aporx. 25 operaciones	Se usa un integrador por trapecios.
Compensación	$p^*[n] = \tilde{p}[n] + V_{Opi}[n]$	• 1 suma • 6 moves Aprox. 7 operaciones	

Tabla B.5. – Operaciones estimadas para el control del condensador

En total se tienen aproximadamente 40 operaciones, lo cual implica menos de 300ns como tiempo de ejecución.

Operaciones totales

Tomando en consideración todas las operaciones estimadas para un ciclo de filtrado para cada bloque, se muestran en la tabla B.6 las operaciones estimadas y los tiempos de ejecución de cada bloque.

Bloque	*Orden de operaciones*	*Tiempo estimado*	*Comentarios*
Programa principal	28 operaciones	T_{pp} = 590ns	Para el tiempo se tomo en cuenta la adquisición de muestras
Celda selectiva	Con Clarke: 178 operaciones. Sin Clarke 136 operaciones.	Con Clarke: T_{cc} = 1.2us Sin Clarke T_{sc} = 900ns	La distinción se hace para la estimación por cantidad de celdas.
PLL	90 operaciones	T_{PLL}= 600ns	Como ya se dijo, pueden ser más.
Control del inversor		T_I = 480ns	Si se usa control vectorial se necesitan más operaciones.
Control del Condensador	40 operaciones	T_C = 300ns	

Tabla B.6. – Estimación de las operaciones totales

A partir de los tiempos que se muestran en la tabla B.6. podemos construir una expresión para el tiempo de ejecución en función de la cantidad de secuencias armónicas a filtrar. Esta expresión es la ecuación B.1.

$$T_T(n) = T_{pp} + T_{PLL} + T_I + T_C + \underbrace{T_{CC} + T_{SC}}_{residual} + \underbrace{nT_{SC}}_{selectivo} \quad \text{(B.1)}$$

Utilizando la expresión B.1, se deduce que para filtrar una secuencia armónica se necesitan aproximadamente 680 operaciones o 4,5 μs, lo cual es un tiempo muy bajo.

Se supuso en su momento que este numero en realidad iba a ser un tanto mayor, sin embargo el tiempo de ejecución de cada celda selectiva fue de 3 μs.

C. El Starter Kit

Luego de seleccionado el DSP, se comenzó la búsqueda de un kit de desarrollo o un Starter Kit. Se encontraron varios tipos, de diferentes precios y distintas prestaciones para desarrollar aplicaciones. Para nuestra aplicación particular escogimos el eZdsp F2812. Esta placa es un Starter Kit que posee:

- Un DSP TMS320F2812
- Dos buses de expansión, uno digital y uno analógico.
- Una memoria externa de 64K por 16 bits
- Interfaz con el PC por puerto paralelo (para programación)
- Conector JTAG y un controlador JTAG
- Software Code Composer Studio (CCS)

En la figura

C.1 se ilustra un diagrama de bloques de la placa.

.

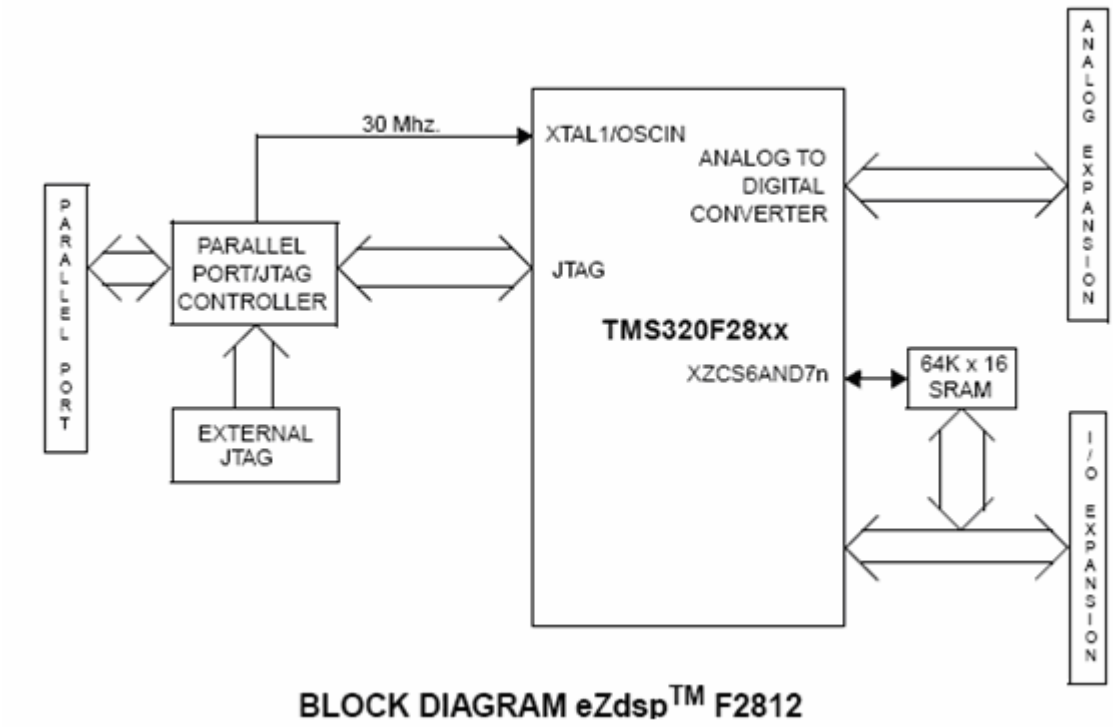


Figura C.1

Los conectores de expansión nos van a dar la posibilidad de interacción del DSP con el mundo. El conector analógico tiene 30 pines de entrada, 16 entradas analógicas, 10 GNDs y 4 señales de referencia para trabajar con los conversores ADC. Solo necesitamos 8 de las entradas analógicas.

El conector digital tiene 40 pines. En este conector encontramos las señales de los puertos serie que posee el DSP, de los cuales vamos a usar uno para la interfaz con Simulink

Otra característica importante que tiene esta placa, es el costo. En comparación con otras soluciones para desarrollo, esta es muy económica. Placas para desarrollo se encuentran en el entorno de los 2000 dólares, mientras que el eZdsp ronda los 400 dólares.

El Starter Kit trae con sigo el CCS, es un software muy potente, cuenta con herramientas como editor, simulador, linker, compilador de C y C++, etc. Este software fue esencial en el desarrollo de nuestro proyecto.

Los motivos por los cuales seleccionamos este DSP son muy claros, el lazo de comunicación que posee nuestro tutor con gente que tiene experiencia con este mismo DSP. Esto puede significar intercambio de información, como puede ser experiencias y código, ambas muy valiosas.

Al observar las características del DSP, como son, conversores ADC, memoria, velocidad, etc., se pudo ver que este DSP, a simple vista, es suficiente para nuestra aplicación. Luego del análisis del orden de la cantidad de operaciones a realizar, llegamos a la misma conclusión. Pero hay que tener en cuenta que en la práctica puede que el análisis de operaciones haya sido muy escueto. Como ya se menciono, por el tema de las operaciones con el stack y otras, que a priori no son difíciles de evaluar. Si bien el número de operaciones se va a ver incrementado, el DSP será suficiente para nuestra aplicación, por experiencias previas con otros DSPs.

En cuanto al Starter Kit, se optó por uno relativamente económico, que cumple los requerimientos básicos necesarios, como lo son los puertos analógicos y digitales, la posibilidad de comunicación por puerto paralelo para la programación y puerto serie como interfaz con Simulink y otros. Otra característica muy importante de esta placa, es que trae el software CCS, herramienta que resultó básica para el desarrollo de la aplicación.

D. Funciones del CPU–Timer

A continuación se muestra el archivo DSP281x_CpuTimers.c, en el cual se definen las funciones de inicialización del timer y de programación del mismo.

```
// TI File $Revision: /main/2 $
// Checkin $Date: April 29, 2005   11:11:22 $
//###########################################################################
//
// FILE:       DSP281x_CpuTimers.c
//
// TITLE:      DSP281x CPU 32-bit Timers Initialization & Support Functions.
//
// NOTES:   CpuTimer1 and CpuTimer2 are reserved for use with DSP BIOS and
//          other realtime operating systems.
//
//          Do not use these two timers in your application if you ever plan
//          on integrating DSP-BIOS or another realtime OS.
//
//          For this reason, the code to manipulate these two timers is
//          commented out and not used in these examples.
//
//###########################################################################
// $TI Release:$
// $Release Date:$
//###########################################################################

#include "DSP281x_Device.h"     // DSP281x Headerfile Include File
#include "DSP281x_Examples.h"   // DSP281x Examples Include File

struct CPUTIMER_VARS CpuTimer0;

// CpuTimer 1 and CpuTimer2 are reserved for DSP BIOS & other RTOS
//struct CPUTIMER_VARS CpuTimer1;
//struct CPUTIMER_VARS CpuTimer2;

//---------------------------------------------------------------------------
// InitCpuTimers:
//---------------------------------------------------------------------------
// This function initializes all three CPU timers to a known state.
//
void InitCpuTimers(void)
{
    // CPU Timer 0
        // Initialize address pointers to respective timer registers:
        CpuTimer0.RegsAddr = &CpuTimer0Regs;
        // Initialize timer period to maximum:
        CpuTimer0Regs.PRD.all  = 0xFFFFFFFF;
        // Initialize pre-scale counter to divide by 1 (SYSCLKOUT):
        CpuTimer0Regs.TPR.all  = 0;
        CpuTimer0Regs.TPRH.all = 0;
        // Make sure timer is stopped:
        CpuTimer0Regs.TCR.bit.TSS = 1;
        // Reload all counter register with period value:
        CpuTimer0Regs.TCR.bit.TRB = 1;
        // Reset interrupt counters:
        CpuTimer0.InterruptCount = 0;


// CpuTimer 1 and CpuTimer2 are reserved for DSP BIOS & other RTOS
// Do not use these two timers if you ever plan on integrating
// DSP-BIOS or another realtime OS.

}

//---------------------------------------------------------------------------
// ConfigCpuTimer:
//---------------------------------------------------------------------------
// This function initializes the selected timer to the period specified
// by the "Freq" and "Period" parameters. The "Freq" is entered as "MHz"
// and the period in "uSeconds". The timer is held in the stopped state
// after configuration.
//
void ConfigCpuTimer(struct CPUTIMER_VARS *Timer, float Freq, float Period)
```

```
{
	Uint32  temp;

	// Initialize timer period:
	Timer->CPUFreqInMHz = Freq;
	Timer->PeriodInUSec = Period;
	temp = (long) (Freq * Period);
	Timer->RegsAddr->PRD.all = temp;

	// Set pre-scale counter to divide by 1 (SYSCLKOUT):
	Timer->RegsAddr->TPR.all  = 0;
	Timer->RegsAddr->TPRH.all  = 0;

	// Initialize timer control register:
	Timer->RegsAddr->TCR.bit.TSS = 1;      // 1 = Stop timer, 0 = Start/Restart Timer
	Timer->RegsAddr->TCR.bit.TRB = 1;      // 1 = reload timer
	Timer->RegsAddr->TCR.bit.SOFT = 1;
	Timer->RegsAddr->TCR.bit.FREE = 1;     // Timer Free Run
	Timer->RegsAddr->TCR.bit.TIE = 1;      // 0 = Disable/ 1 = Enable Timer Interrupt

	// Reset interrupt counter:
	Timer->InterruptCount = 0;
}
```

E. Conectores

En la figura E.1 se observan las posiciones de los distintos conectores de la placa implementada. Las señales de cada conector se detallan a continuación.

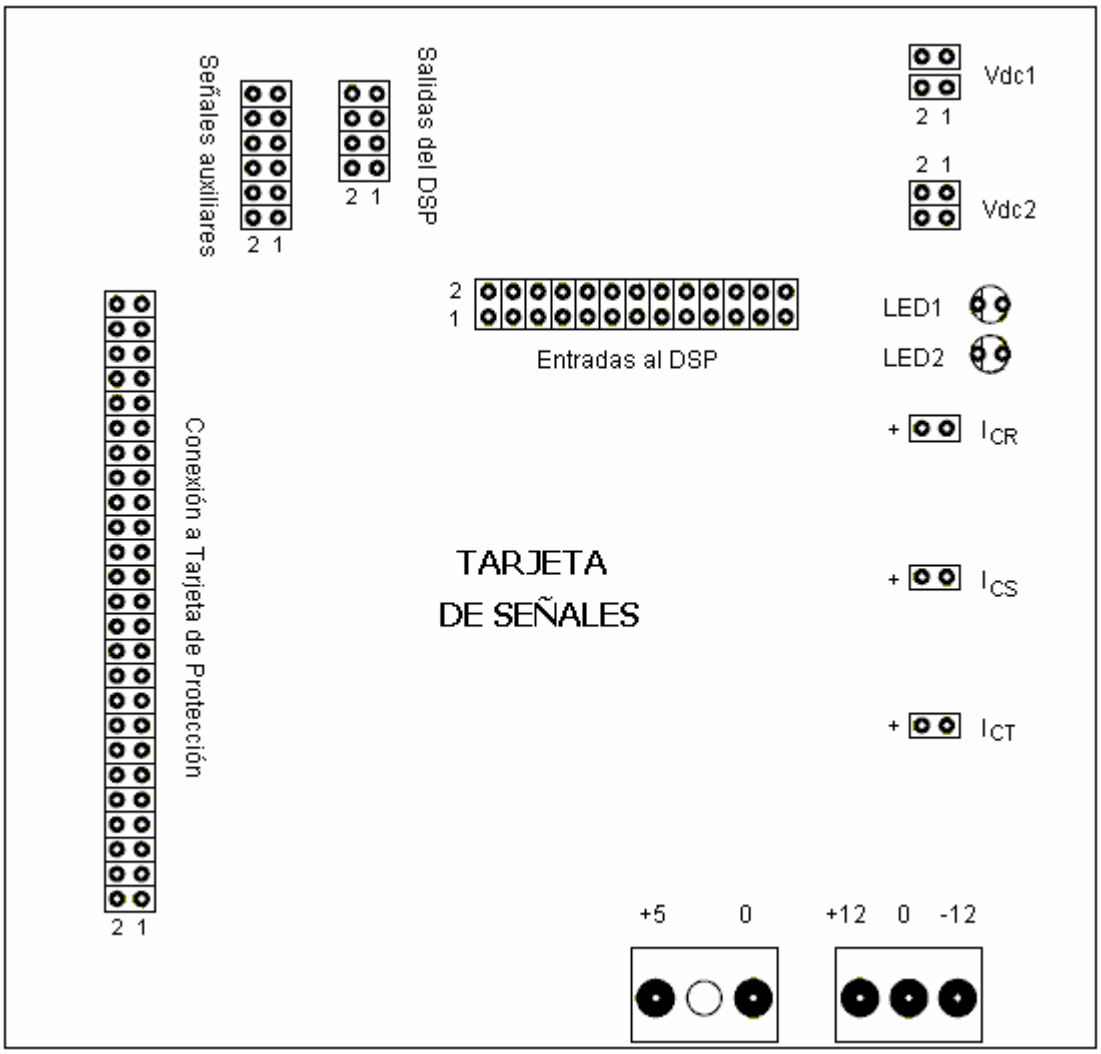


Figura E.1 – Conectores en la placa

Conectores hacia el DSP:

Entradas al DSP:

Las señales de cada pin del conector de entradas analógicas al DSP son:

1. Voltaje de filtro fase T,
2. Entrada digital 2
3. Voltaje de filtro fase S
4. Entrada digital 4
5. Voltaje de filtro fase R
6. Entrada digital 6
7. Corriente de filtro fase T
8. Entrada digital 1 ED_1
9. Corriente de filtro fase S
10. Entrada digital 3
11. Corriente de filtro fase R
12. Entrada digital 5
13. Tierra digital
14. Tierra digital
15. Tierra analógica
16. Tierra analógica
17. No se conecta
18. Tensión del condensador 2
19. Corriente de carga fase R
20. Tensión del condensador 1
21. Corriente de carga fase S
22. No se conecta
23. Corriente de carga fase T
24. No se conecta
25. Led 2
26. Led 1

Salidas del DSP:

1. No se conecta
2. Control rama 1
3. No se conecta
4. Control rama 3
5. No se conecta
6. Watchdog
7. No se conecta
8. Control rama 2

Señales auxiliares:

Solo usamos la señal pedido de inhabilitación, la cual habilita el encendido de las llaves con un 1 lógico. Por otro lado la señal habilitación de salida debe estar siempre alta, para lo cual existe un jumper en la placa de protección.

1. No se conecta
2. Sentido de giro
3. No se conecta
4. Habilitación salida
5. No se conecta
6. Pedido inhabilitación
7. No se conecta
8. Pin 48
9. No se conecta
10. Encoder
11. No se conecta
12. Pin 41

Conector a tarjeta de protección

La siguiente tabla (nueva versión de la extraída de [10]) describe las señales que antiguamente intercambiaban la tarjeta de protección y la computadora mediante un conector de 50 pines. Nuestra placa tiene un conector compatible. En negrilla marcamos cuáles son las señales de nuestro interés y cuáles son nuestras observaciones.

NOMBRE	PIN	USO ASIGNADO
AIGND	**1**	**Tierra analógica**
AIGND	**2**	**Tierra analógica**
AIN0	3	Tensión del bus de continua
AIN8	4	No se conecta
AIN1	**5**	**Corriente de fase 1**
AIN9	6	No se conecta
AIN2	**7**	**Corriente de fase 2**
AIN10	8	No se conecta
AIN3	**9**	**Corriente de fase 2**
AIN11	10	No se conecta
AIN4	**11**	**Tensión T de la red (antes U)**
AIN12	12	No se conecta
AIN5	**13**	**Tensión S de la red (antes V)**
AIN13	14	No se conecta
AIN6	**15**	**Tensión R de la red (antes W)**
AIN14	16	No se conecta
AIN7	17	No se conecta
AIN15	18	No se conecta
DGND	19	Tierra digital
-12V	**20**	**Fuente de -12V (corregido, pues figuraba como -5V)**
+12V	**21**	**Fuente de +12V**
DIN0	22	No se conecta
DIN1	**23**	**Entrada del estado de la llave S1 U**
DIN2	**24**	**Entrada del estado de la llave S1 D**
DIN3	**25**	**Entrada del estado de la llave S2 U**
DIN4	**26**	**Entrada del estado de la llave S2 D**
DIN5	**27**	**Entrada del estado de la llave S3 U**
DIN6	**28**	**Entrada del estado de la llave S3 D**
DIN7	29	Entrada del sentido de giro
DOUT0	30	No se conecta
DOUT1	**31**	**Salida del estado de Rama 1**
DOUT2	**32**	**Salida del estado de Rama 2**
DOUT3	**33**	**Salida del estado de Rama 3**
DOUT4	**34**	**Pulso al watch dog**
DOUT5	35	Señal de habilitación de salida
DOUT6	36	No se conecta
DOUT7	**37**	**Señal pedido de inhabilitación**
OUT1*	38	No se conecta
EXTINT*	39	No se conecta
EXTCONV*	40	No se conecta
OUT0	41	Conectado con el pin 48
GATE0	42	Conectado a +5V
OUT1	43	No se conecta
GATE1	44	Conectado a +5V
CLK1	45	Entrada de los pulsos de velocidad provenientes del encoder
OUT2	46	No se conecta (genera el pedido de interrupción para la adquisición)
GATE2	47	Conectado a +5V
CLK2	48	Conectado con el pin 41
+5V	**49**	**Fuente de +5V**
AIGND	**50**	**Tierra digital (conectado con el pin 19)**

F. Placa para adaptación de niveles en la comunicación serie

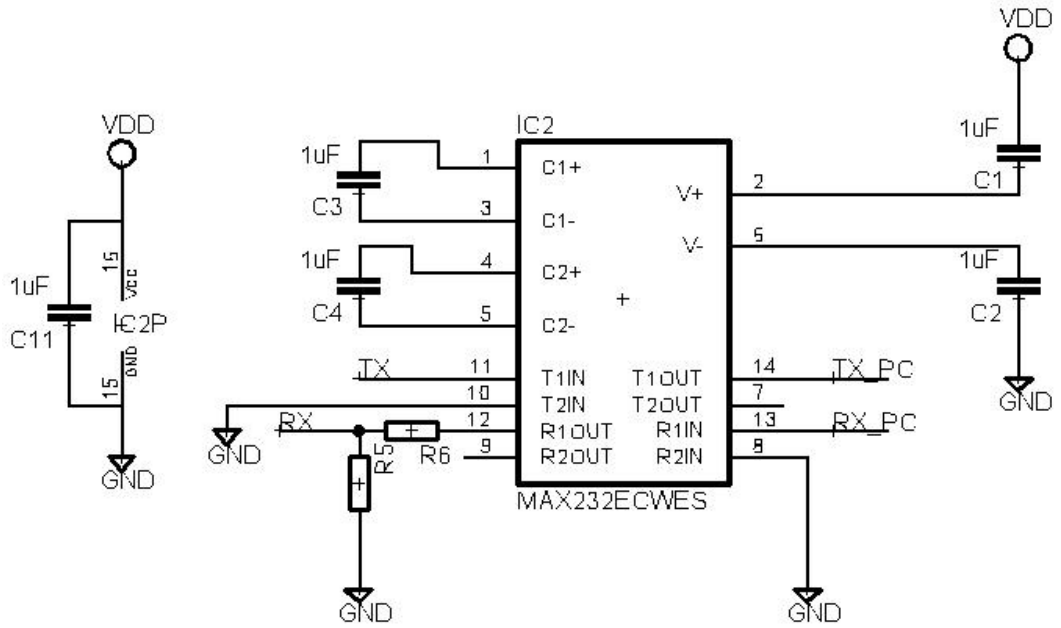


Figura F.1. – Esquemático placa de ajuste de niveles para el RS232.

El DSP maneja niveles de entre 0V y 3.3V, el estándar de RS232 maneja niveles entre 12V y -12V, por tal motivo surge la necesidad de ajustar los niveles de tensión. El MAX232 realiza la conversión de niveles TTL a niveles RS232, o sea convierte 0V TTL en 12V, y 5V TTL en -12V, y viceversa. El circuito del MAX232 es alimentado con 5V proporcionados por la placa del DSP, esto implica que la señal RX para un '1' lógico vale 5V, por lo cual se coloco un divisor resistivo en la señal RX del MAX232 para así obtener 3.3V en la pata RX del DSP. Para la transmisión no hubo problemas al utilizar 3.3V (en vez de 5V) en la señal de TX, por lo cual se conecto directamente.

Esta placa no trajo ningún problema de implementación ni de funcionamiento, y aparte de ser necesaria para el objetivo del proyecto, ha sido muy importante para el debug de todos los bloques.

G. Material contenido en el CD

Documentos
En la carpeta Documentación se encuentra en formato pdf (excepto el poster):

- Documentación final
  Documentación_FiltroActivo.pdf

- Entregables intermedios
  Tarea 1.pdf
  Tarea 2.pdf
  Tarea 3.pdf
  Tarea 4 (costos).pdf
  Plan de Proyecto.pdf

- Paper
  Paper Filtro Activo.pdf

- Poster
  Paper Filtro Activo.cdr

Simulaciones
Requerimientos:
- Matlab 7.1
- Se recomienda Intel Core Duo

1) Filtro Activo 1 – Simulación inicial del filtro activo (resultados documentados)

- *Main_FiltroActivo.m* – programa principal

- *FiltroActivo.mdl* – archivo de Simulink

2) Filtro Activo con transformador – Simulación del sistema implementado en el instituto (resultados no documentados).

- *Main_FiltroActivoFacultad.m* – programa principal

- *FiltroVSIfacultad.mdl* – archivo de Simulink

Hardware-in-the-Loop
Requerimientos:
- Matlab 7.1
- Code Componer Studio 3.1
- Starter Kit ezDSP con conexión al puerto serie de la PC.
- Se recomienda Intel Core Duo

1) Control Filtro con PLL simple (resultados documentados)

- Proyecto CCS:
  FiltroActivo.pjt

- Archivos Matlab-Simulink:
  main_FiltroActivo.m

2) Control Filtro con SSI-PLL (resultados no documentados)

- Proyecto CCS:
  FiltroActivo.pjt

- Archivos Matlab-Simulink:
  main_FiltroActivo.m

Control del Filtro Real
- Proyecto CCS:
  FiltroActivoConGUI.pjt

- Archivos Matlab-GUIDE:
  DSP.m
  DSP.fig

- GUI (ejecutable, no necesita Matlab)
  DSP.exe

Archivos auxiliares DSP
Archivos comunes a varios de los programas implementados en el DSP: código de programas (.c), archivos de encabezados (.h), archivos de asignación de memoria (.cmd) y librerías (.lib).

Bibliografía
Extracción del control vectorial de corriente:

- Autores: Biardo, Giacosa, Rivoir (2007)

Manuales DSP (Texas Instruments):

- *IQmath.pdf* – IQmath Library, Module user's Guide.

- *SCI.pdf* – Serial Communication Interface (SCI) Reference Guide

- *ADC.pdf* – Serial Communication Interface (SCI) Reference Guide

- *Interrupts.pdf* – TMS321x281x DSP System Control and Interrupts Reference Guide