UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INGENIERÍA

MAESTRÍA EN CIENCIAS MECATRÓNICA

Implementación de algoritmos de medición para calidad de la energía eléctrica basado en FPGA con la especificación CFE G0000-48

Opción de titulación

Tesis

Que como parte de los requisitos para obtener el Grado de Maestro en Ciencias Mecatrónica

Presenta:

Ing. Leonardo Esteban Moreno Suárez

Dirigido por:

Dr. Luis Morales Velázquez

San Juan del Río, Querétaro, Noviembre de 2020.

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Maestría en Ciencias Mecatrónica

Implementación de algoritmos de medición para calidad de la energía eléctrica basado en FPGA con la especificación CFE G0000-48

Opción de titulación
Tesis

Que como parte de los requisitos para obtener el Grado de Maestro en Ciencias Mecatrónica

Presenta:
Ing. Leonardo Esteban Moreno Suárez

Dirigido por:
Dr. Luis Morales Velázquez

Dr. Luis Morales Velázquez
Presidente

Dr. Arturo Yosimar Jaen Cuellar
Secretario

Dr. Roque Alfredo Osornio Ríos
Vocal

Dra. Adriana del Carmen Téllez Anguiano
Suplente

Dr. Martín Valtierra Rodríguez
Suplente

Centro Universitario Querétaro, Qro.
Noviembre 2020.
México

Resumen

La calidad de la energía es indispensable en las mediciones cuantificables y comparables de disturbios y eventos que ocurren en un sistema de red eléctrica regidas por estándares y normas internacionales. En México, la Comisión Federal de Electricidad (CFE) establece una especificación que indica la incertidumbre, margen de error y tiempo en la parametrización de disturbios en los medidores. Los dispositivos en hardware descritos en arreglo de compuertas programables en campo (FPGA, de sus iniciales en inglés *Field Programmable Gate Array)* y los dispositivos en software conocidos como ARM (*Advanced RISC Machine*) son dispositivos programables en ejecución individual dentro de sus plataformas de desarrollo. Se pretende realizar la comparación de ambas tecnologías para verificar la correcta funcionalidad de algoritmos de calidad de la energía y su tiempo de ejecución, cumpliendo con las incertidumbres como medidores de Clase S. La implementación de los algoritmos de medición de calidad de la energía eléctrica con base en los estándares de las especificaciones CFE G0000-48 entre ambos dispositivos en la tarjeta de adquisición PQ-UAQ brindó una mejora en la eficiencia computacional en procesamiento mediante la programación VHDL, síntesis y la comparación de ambos dispositivos con instrumentos de medición basados en su tiempo de ejecución, los recursos consumidos, la memoria utilizada y su margen de error; además de obtener la relación costo-beneficio. Dichos algoritmos fueron específicamente seleccionados y validados para posteriormente conocer los recursos existentes del dispositivo para su correcta implementación. Con los algoritmos de medición, se generaron señales sintéticas y se validaron con señales reales muestreadas de una red eléctrica trifásica definiendo bancos de pruebas. Posteriormente, se implementaron los algoritmos en los dispositivos y se validó su funcionalidad con los bancos de pruebas y se obtuvo su margen de error. Los dispositivos se probaron en la red eléctrica trifásica y se analizó su sensibilidad con el entorno. El dispositivo FPGA mostró un menor consumo de recursos, un tiempo de ejecución seis veces más rápido y un de margen de error mil veces mayor en comparación a los resultados entregados por el dispositivo ARM, pero contemplando una posible reducción del margen de error y a la vez un incremento de recursos aproximadamente el doble. A pesar de haberse empleado el 75% de la capacidad total de la memoria, la comparación de ambos dispositivos permitió garantizar la implementación de los algoritmos en ellos.

(**Palabras clave**: Algoritmos, implementación, dispositivos, comparación.)

Summary

Power quality is indispensable in quantifiable and comparable measurements of disturbances and events that occur in a power grid system governed by international standards and norms. In Mexico, the Comisión Federal de Electricidad (CFE) establishes a specification that indicates the uncertainty, margin of error and time in the parameterization of disturbances in the meters. The hardware devices described in the field programmable gate array (FPGA) and the software devices known as ARM (Advanced RISC Machine) are programmable devices in individual execution within their development platforms. It is intended to compare both technologies to verify the correct functionality of power quality algorithms and their execution time, complying with the uncertainties such as class S meters. The implementation of electrical power quality measurement algorithms based on the CFE G0000-48 specification standards between both devices on the PQ-UAQ acquisition card provided an improvement in computational efficiency in processing through VHDL programming, synthesis and the comparison of both devices with measurement instruments based on their execution time, the resources consumed, the memory used and their margin of error; in addition to optimizing the cost-benefit ratio. Such algorithms were specifically selected and validated to later know the existing resources of the device for its correct implementation. With the measurement algorithms, synthetic signals were generated and validated with real signals sampled from a three-phase electrical network defining test benches. Subsequently, the algorithms were implemented in the devices and their functionality was validated with the test benches and their margin of error was obtained. The devices were tested on the three-phase electrical network and their sensitivity to the environment was analyzed. The FPGA device showed lower resource consumption, six times faster execution time and a margin of error considerably at a scale of a thousand times greater compared to the ARM device, but considering a possible reduction of the margin of error and at the same time an increase of resources approximately double. Despite having used 75% of the total memory capacity, the comparison of both devices allowed to guarantee the implementation of the algorithms in them.

(**Keywords:** Algorithms, implementation, devices, comparison.)

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por la beca de Maestría con número de CVU 931510.

A la Universidad Autónoma de Querétaro por la beca de posgrado con número de expediente 281382.

A mi director de tesis Dr. Luis Morales Velázquez por brindarme la oportunidad de participar en este proyecto, por su apoyo y tiempo dedicado para culminar una etapa más de mi vida.

A mi hermosa madre que siempre ha sido la primera en brindarme su apoyo en cada etapa que me he propuesto siendo mi fuente de motivación, a mi hermano que constantemente coincidimos en nuestros logros apoyándonos mutuamente, y familiares quienes me animaron en todo momento, motivaron a seguir adelante y dieron consejos, a pesar de la distancia en la que nos encontrábamos.

A mi profesora y amiga la Dra. Adriana del Carmen Téllez Anguiano por brindarme confianza y apoyo en todo momento.

A mis sinodales el Dr. Arturo Yosimar Jaen Cuellar, Dr. Roque Alfredo Osornio Ríos y el Dr. Martín Valtierra Rodríguez quienes me brindaron sus consejos en el proyecto, apoyo y amistad en el transcurso de mi estadía.

A mis compañeros de la Facultad de ingeniería por su compañía, ayuda y los momentos pasados dentro de los cubículos.

Índice general

Índice de figuras

Índice de cuadros

I. INTRODUCCIÓN

La energía eléctrica es una fuente popular de consumo que es usada en actividades residenciales, comerciales e industriales y su demanda crece de un 6 al 8% al año. La implementación de calidad de energía eléctrica tiene un fuerte impacto en la realización de medidores inteligentes de la señal eléctrica, que a su vez presentan perturbaciones en la señal, regidas bajo normas internacionales y/o estándares como los mostrados por el instituto de ingenieros eléctricos y electrónicos *(Institute of Electrical and Electronics Engineers*, IEEE). El desarrollo continuo de nuevas tecnologías y su aporte en la energía eléctrica se ha reflejado en el alto incremento de controladores, dispositivos eléctricos y electrónicos de ámbito industrial tales como variadores de velocidad con controlador de estado sólido, hornos o soladoras de arco eléctrico, sistemas de tracción eléctrica, transformadores, entre otros. Estos aparatos de consumo eléctrico producen perturbaciones en los ciclos de la señal de tensión y la corriente del sistema de energía eléctrica nacional, por lo que es importante la optimización en la exactitud de la medición en perturbaciones bajo estándares y normativas, en su eficiencia y la disminución en sus costos en consumos de recursos, por lo que se requiere un medidor de calidad de energía que sea capaz de caracterizar los parámetros de los disturbios y eventos en la red eléctrica.

La solución de esta problemática en el presente trabajo se rigió con la especificación CFE G0000-48, la cual es un documento mexicano fundamentado por normas internacionales y estándares que es establecida para medidores de señales eléctricas de 60 Hz. Con la especificación se realizó una comparación al implementarlos en dos tecnologías diferentes en programación en software y descripción en hardware con el uso de la tarjeta de adquisición PQ-UAQ, pretendiendo obtener una mejora en la eficiencia computacional en procesamiento mediante la programación en lenguaje C y lenguaje descriptivo de circuitos integrados de muy alta velocidad (*Very high-speed integrated circuit Hardware Description Language*, VHDL), síntesis y la comparación de ambos dispositivos con instrumentos de medición, basándose en las variables de tiempo de ejecución, recursos consumidos, memoria utilizada

y el margen de error, optimizando la relación costo-beneficio. Los algoritmos implementados fueron específicamente seleccionados y validados, y se conocieron los recursos existentes del dispositivo para tener una correcta implementación. Las señales sintéticas generadas por los algoritmos se validaron con señales reales muestreadas de una red eléctrica trifásica con la que se definieron bancos de pruebas.

Se obtuvo la validación de la funcionalidad en las implementaciones de los algoritmos en los dispositivos con los bancos de pruebas y su margen de error, probando los dispositivos en la red eléctrica trifásica y analizando su sensibilidad con el entorno.

Los resultados fueron prometedores y mostraron una mejora en las variables de estudio, permitiendo garantizar la implementación de los algoritmos en los dispositivos. Esto permitirá la futura integración como medidor en residencias domésticas e industriales, ya que se podrá aplicar en la medición de cualquier señal eléctrica, en sistemas de generación de energías renovables, análisis de generadores, además del monitoreo de equipos como motores para detección de fallas.

1.1. Antecedentes

La calidad de la energía eléctrica es un tema de importancia actualmente, las normas internacionales y/o estándares más utilizados para la medición de parámetros de calidad son: IEEE 519, IEC 61000, EN 50160, IEEE 1156, entre otros. El consumo de energía eléctrica se medía por medio del uso de medidores mecánicos que no son funcionales con respecto a la técnica de calidad de la energía eléctrica actual. Los medidores electromecánicos operan por medio de bobinas de potencia y corriente, provocando el movimiento de un disco de aluminio (contador de inducción) y así accionar un contador mecánico que indica el consumo (Comisión Federal de Electricidad, 2006).

En la actualidad estos medidores se han mejorado, utilizando componentes de estado solidos (digitales), donde el programa de control es almacenado en una memoria digital, para así mostrar en una pantalla los registros de medición como son: potencia kilo-Watt-hora (kWh) (mediciones obtenidas por sensores de voltaje y corriente), interrupciones de tensión

y parámetros de programación del medidor. Otra de las mejoras de este tipo de medidores, es la capacidad de integrar energías en forma bidireccional (CFE, 2006).

El desarrollo en medición de energía eléctrica ha evolucionado a nivel mundial desde el año 2000. La prioridad y el primer paso a nivel mundial es la implementación de medidores inteligentes. Cada pieza de la red inteligente se despliega a diferente velocidad (Ruiz & García, 2015).

Una manera de medir los parámetros eléctricos para tener un control es denominado calidad de la energía (*Power Quality*, PQ), y se rige bajo ciertas normas (IEC 61000-4-30, 2008; IEC 61000-4-7, 2008) y/o estándares (IEEE Std 1159, 2009; IEEE Std 519, 2014), el cual es una manera de obtener resultados fiables, repetibles y comparables de las mediciones en la red eléctrica (Villablanca, 2009). Los algoritmos junto con la reconfiguración de la red eléctrica han servido en la minimización de la pérdida en la distribución del sistema eléctrico (Bashardoust et al., 2016).

La especificación G0000-48 (2010) contempla los parámetros caracterizados con base en lo establecido en la norma internacional IEC 61000-4-30 (2008) para medidores Clase S, donde el margen de error debe estar en el rango ±0.2%, para detectar valores fuera de rango, mínimos, máximos, promedio e intervalo de tiempo. Esta especificación considera la medición de calidad de energía en los parámetros de frecuencia del sistema trifásico de tensiones, magnitud (variaciones de tensión) por fase tomando como referencia el cruce por cero, depresiones conocidas como huecos de tensión (*sag*), incrementos (*swell*), interrupciones, todos estos por fase, desbalance de sistema trifásico de tensiones, armónicos de tensión y corriente (individuales y totales) hasta la armónica 25 (CFE, 2010; Jaramillo, 2013; Rens et al., 2014). La norma IEC 61000-4-30 especifica cómo medir los eventos de calidad de la energía, que son mediciones derivadas medio ciclo, y sincronizadas independientemente de cada canal. La existencia de medidores clasificado en tres maneras: Clase A, B y S (Neumann, 2007). Por otro lado, la norma EN 50160 establece que los armónicos de voltaje deben de ser hasta el 40vo y el valor máximo es de 8% (Rohrig et al., 2012).

El equipo de medición encargado de monitorear el sistema eléctrico ha cumplido con los méritos en la solución de un software de fácil acceso, capacidad de almacenamiento de datos, disponibilidad en la modificación del algoritmo por parte del usuario (Tarasiuk et al., 2011) y los equipos registran las perturbaciones de voltaje a medida en que ocurrieron (Romero et al., 2011). Se han implementado sensores basados en arreglo de compuertas programables en campo (*Field Programmable Grid Array*, FPGA) para observar disturbios de calidad de la energía, que son capaces de ejecutar tareas en tiempo real debido a sus capacidades de procesamiento de alta velocidad y configurabilidad (Granados-Lieberman et al., 2013). También se ha realizado la identificación de fenómenos transitorios bajo el estándar IEEE 1159-209 (Guillén-García et al., 2019) utilizando como método la transformada discreta wavelet y la transformada rápida de Fourier (*Fast Fourier Transform*, FFT). El diseño de medidor Clase A bajo la norma IEC 61000-4-30 basado en plataforma hardware CompacRIO-9024 y FPGA como medio de sincronización, los métodos utilizados son ventanas de tensión y frecuencia, detección por salto de fase y detección de umbral de distorsión armónica total (*Total Harmonic Distorsion*, THD) (Real-Calvo et al., 2017).

Por otra parte, se han implementado medidores de calidad de la energía basados en microcontroladores ARM, en la medición de frecuencia, valores de raíz cuadrada promedio (*Root-Mean-Square*, RMS) de voltaje y corriente, potencia activa y aparente, y distorsión armónica total de voltaje (Cardona et al., 2013) y corriente basada en el estándar EN 50160 (Gallo et al., 2010). Se han realizado medidores de calidad basados en procesadores digitales de señal (*Digital Siganl Processor*, DSP), que se caracterizaba por medir la potencia instantánea y las potencias (activa, reactiva y aparente), donde el dispositivo era el encargado de realizar las operaciones con la información obtenida por un convertidor (Casellas et al., 2010).

Algunos de los métodos para la disminución de carga computacional son la FFT radix-2 en la transformación del dominio de tiempo a frecuencia y la computadora digital para rotación de coordenadas (*Coordinate Rotation Digital Computer*, CORDIC) para la obtención de ángulo de fase. Estos métodos se han implementado en FPGA para el monitoreo de señales monofásicas de 60 Hz (Ria et al., 2012). Los métodos de igual manera han servido

para obtener el desbalance de tensión, que es causado por conexiones de cargas monofásicas en un sistema trifásico, fallas en el aislamiento del conductor, desconexión de una base en el banco de condensadores, transformadores delta abiertos, etc. (Jaramillo, 2013).

1.2. Descripción del problema

Los medidores diseñados utilizando el PQ-UAQ no son capaces de caracterizar los parámetros de calidad de la energía bajo la especificación G0000-48. La implementación de algoritmos en dispositivos de arreglo de compuertas programables en campo (Hardware) y procesador basado en arquitectura RISC (Software) con base en los estándares muestra una variación en el tiempo de ejecución, cantidad de memoria utilizada, recursos consumidos y margen de error mayor a 0.2%, que son evaluados mediante instrumentos de medición. Esto conlleva un alto costo y baja eficiencia operacional en el equipo debido a un elevado consumo de recursos en el proceso de ejecución.

1.3. Justificación

La implementación de los algoritmos de calidad de la energía en dos tecnologías diferentes (hardware FPGA y software ARM), permite contrastar cuál de estos dispositivos cumple con los requisitos que establece la especificación CFE G0000-48. Se realizará la comparación de los dispositivos con base en el tiempo de ejecución, la memoria utilizada, el consumo de recursos y el margen de error. Dicha comparación garantizará una mejoría en la eficiencia computacional en el procesamiento y en la relación costo-beneficio.

1.4. Hipótesis y objetivos

1.4.1. Hipótesis

Mediante técnicas de lenguaje en descripción de hardware y la programación en lenguaje C de los algoritmos de medición de calidad de la energía eléctrica basados en las especificaciones CFE G0000-48, realizar una comparación entre dispositivos digitales, donde se obtiene una mejora en la relación costo-beneficio de la implementación de FPGA con respecto a la del microcontrolador.

1.4.2. Objetivo general

Implementar un conjunto de algoritmos para la medición de índices de calidad de la energía eléctrica bajo las normas especificadas en el documento CFE G0000-48, mediante la programación VHDL, síntesis y comparación en dispositivos FPGA y microcontrolador, para obtener una mejora en la eficiencia computacional en procesamiento.

1.4.3. Objetivos particulares

Con finalidad de dividir el problema en partes más específicas se tienen los siguientes objetivos particulares:

- Seleccionar algoritmos basados en la norma IEC 61000-4-30 Clase S en frecuencia (armónicos individuales y totales) y variaciones de tensión RMS (sag, swell e interrupciones) en el sistema trifásico por fase, para calcular los parámetros de calidad de energía eléctrica.
- Validar los algoritmos seleccionados mediante simulaciones en MATLAB, código C y VHDL, para determinar los parámetros de operación del dispositivo (Frecuencia máxima de operación, cantidad de memoria y el tiempo de procesamiento).

- Diseñar los RTL (Register-Transfer Level) de los bloques de procesamiento en frecuencia (armónicos individuales y totales) con FFT y variaciones de tensión RMS (sag, swell e interrupciones) en el sistema trifásico por fase, mediante los parámetros y ecuaciones establecidas en la norma IEC 61000-4-30.
- Implementar los algoritmos de calidad de energía de procesamiento en frecuencia (armónicos individuales y totales) y variaciones de tensión RMS (sag, swell e interrupciones) en sistema trifásico por fase y la FFT; en el FPGA y el microcontrolador, para obtener tiempo de operación y recursos utilizados.

1.5. Planteamiento general

El desarrollo del proyecto conlleva varias etapas, que consiste en seis bloques esenciales que se mencionan a continuación:

1. Sistema de red trifásica. Seleccionar los algoritmos de medición de la calidad de la energía eléctrica basado en la especificación de la CFE G0000-48 para un sistema de red trifásica. Desarrollar los diagramas de flujo y estructuras digitales traduciendo los algoritmos matemáticos especificados en el documento necesarios para implementar la técnica de procesamiento de las señales seleccionadas, mediante lenguaje de descripción en hardware y programación en lenguaje en C.

2. Adquisición de datos. Generar señales sintéticas y almacenar bancos de señales reales de corriente-voltaje que serán empleados en el funcionamiento de los algoritmos en las dos tecnologías diferentes en el procesamiento de datos basados en el sistema de adquisición. Establecer la comunicación física con el sistema de adquisición mediante conectores para verificar el funcionamiento de los algoritmos de acuerdo a las características de la trasferencia de datos que maneja.

3. Procesamiento de datos. Implementación de los módulos descritos en VHDL en el dispositivo FPGA *Spartan 6* y los algoritmos en lenguaje C en el dispositivo *Hercules*

para detección de índices de calidad de la energía como son interrupción, swell, sag, THD y desequilibrio; además de verificar la correcta funcionalidad en los dispositivos. Establecer la comunicación serial con el FPGA en el sistema de adquisición de datos, y a su vez la comunicación en paralelo con el dispositivo *Hercules*.

4. Bitácoras de eventos. Los resultados obtenidos de la validación de los algoritmos con los bancos de pruebas son almacenados en las bitácoras de eventos de cada dispositivo, además de registrar los datos del sistema de adquisición conectado a una red eléctrica.

5. Validación de mediciones. Empleando la bitácora de eventos, se realiza la comparación de cada dispositivo contemplando el margen de error e incertidumbre de cada índice de calidad de la energía para verificar la correcta implementación en ambas tecnologías.

6. Carga computacional. Realizar la comparación de ambas tecnologías, donde se obtiene la cantidad de memoria y uso de recursos a partir de la implementación en el procesamiento de datos, y donde se obtiene el margen de error de la bitácora de eventos de cada dispositivo.

La Figura I.1 muestra la implementación del proyecto propuesto de manera general.

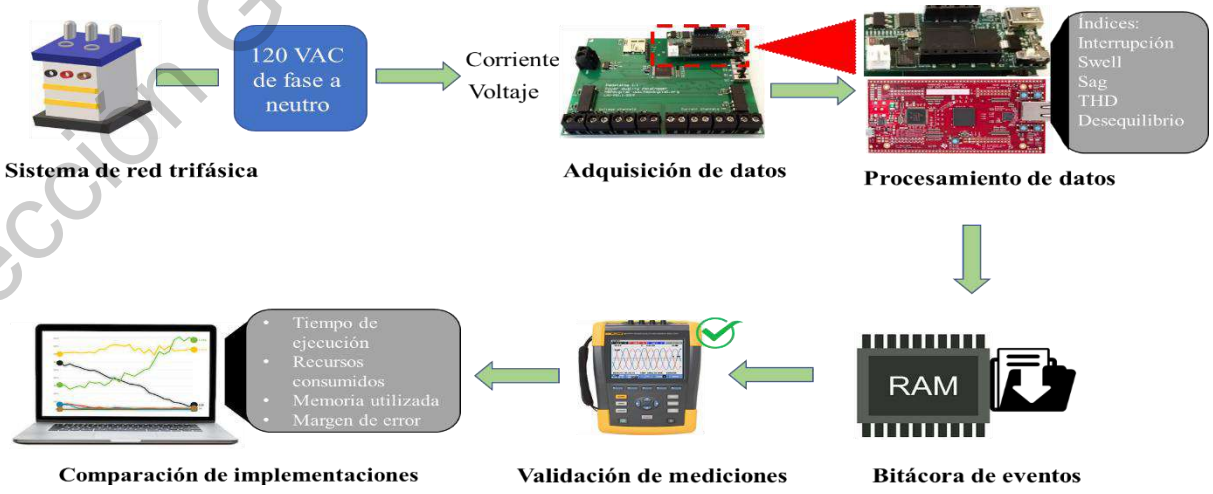


Figura I.1. Planteamiento general del proyecto.

II. FUNDAMENTACIÓN TEÓRICA

En este apartado se mencionan las mediciones necesarias para la implementación de los algoritmos de calidad de la energía eléctrica en un sistema monofásico o trifásico basado en las especificaciones de la CFE. Se manejan términos y acrónimos, por ejemplo: pu (por unidad), que es un término adimensional referente, siendo la cantidad 1.0 pu que corresponde al 100%; la condición nominal de una señal eléctrica sin perturbaciones o también conocida como la amplitud pico de la señal sobre la raíz cuadrada de dos. Además, se mencionan los armónicos, componentes senoidales de una onda periódica con una frecuencia que es un múltiplo entero de la frecuencia fundamental y los inter-armónicos, que es un componente de frecuencia de una cantidad periódica que no es un múltiplo entero de la frecuencia a la que opera el sistema de suministro, las diferentes variaciones de tensiones RMS y la forma de cómo medirla bajo lo especificado en las normas y/o estándares, que también lo es en los armónicos, y la distorsión armónica total. Todos estos términos se integran fundamentalmente para el desarrollo de algoritmos para FPGA mediante VHDL y microcontrolador en código C.

2.1. Variaciones de corta duración

Las variaciones de corta duración son disturbios en el nivel de voltaje de la señal eléctrica que son provocadas por fallas en la alimentación de equipos que requieren alta cantidad de corriente o desconexiones en el cableado del sistema eléctrico.

La manera de medir las variaciones es por medio del valor eficaz, cuando se suman las señales de voltaje o corriente de diferentes frecuencias para obtener su resultante (Casaravilla & Echinope, 2010).

2.1.1. Valor eficaz

El valor eficaz del voltaje de la señal eléctrica es obtenido por medio del RMS, mostrada en la ecuación (1):

$$V_{rms} = \sqrt{\frac{1}{N}\sum_{i=1}^{N} V_i^2} \qquad (1)$$

Donde V_{rms} es el voltaje eficaz, N es la cantidad de datos, V_i es la amplitud del voltaje en la señal. El I_{rms} es la corriente eficaz que puede ser obtenida de la misma manera.

Las fallas de corta duración, como las interrupciones, pueden estar precedidas por una caída de voltaje; si se presenta una interrupción es debido a fallas en el sistema de alimentación (fuente). El voltaje sag se produce entre el momento en que se inicia una falla del sistema, cambio de cargas pesadas o arrancadores de motores grandes. Los voltajes swell son menos comunes que los sag, puede ser causado al apagado, cambio de cargas grandes del sistema o a la conexión de bancos de capacitores (IEEE Std 1159, 2009).

A continuación, se presentarán detalladamente las variaciones de voltaje (interrupciones, sag y swell), y las categorías donde pueden presentarse.

2.1.2. Interrupción momentánea y temporal

Una interrupción se produce cuando la tensión de alimentación o la corriente de carga disminuye a menos de 0,1 pu durante un período de tiempo no mayor a un minuto. Las interrupciones son el resultado de fallas en el sistema de energía, en los equipos y en el control.

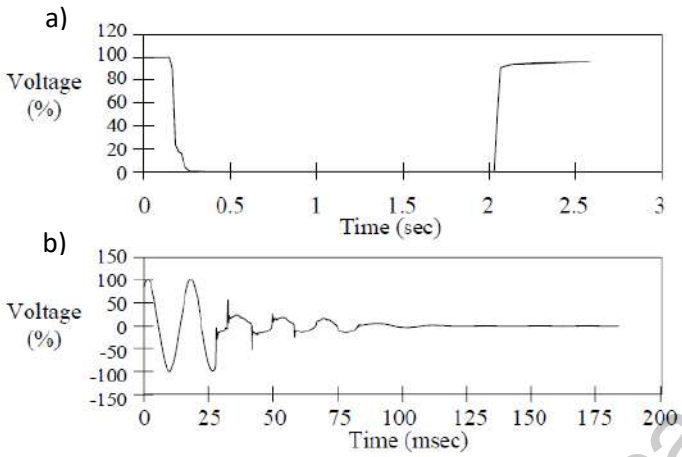


Figura II.1. Interrupción momentánea a) valor eficaz y b) forma de onda debido a una falla, (Std 1159, 2009).

En la Figura II.1 se muestran dos gráficas del momento en que ocurre una falla de interrupción. En a) se muestra el porcentaje del valor eficaz de la señal eléctrica de b) que es obtenida por cada medio ciclo. Se puede observar que el valor eficaz decae hasta cero en un tiempo de 0.25 segundos, siendo constante hasta llegar a 1.75 segundos. Dicho disturbio entra en la categoría momentánea.

2.1.3. Caída de voltaje

Caída de voltaje es una reducción momentánea del valor eficaz de la tensión al orden de 0.1 a 0.9 pu como se muestra en la Figura II.2, con una duración entre 0.5 ciclos a un minuto. Generalmente está asociada a fallas del sistema, pero también es producida por la entrada de grandes bloques de carga o arranque de motores grandes. La caída de voltaje puede provocar el paro de equipos eléctricos o electrónicos y la interrupción de los procesos productivos (Peña et al., 2010). De acuerdo a la IEC 61000-4-11, este fenómeno se define como *dip*, y en Estados Unidos de América se define como sag. De las principales características de los voltajes sag se clasifican de acuerdo a su duración en instantáneos, momentáneos y temporales.

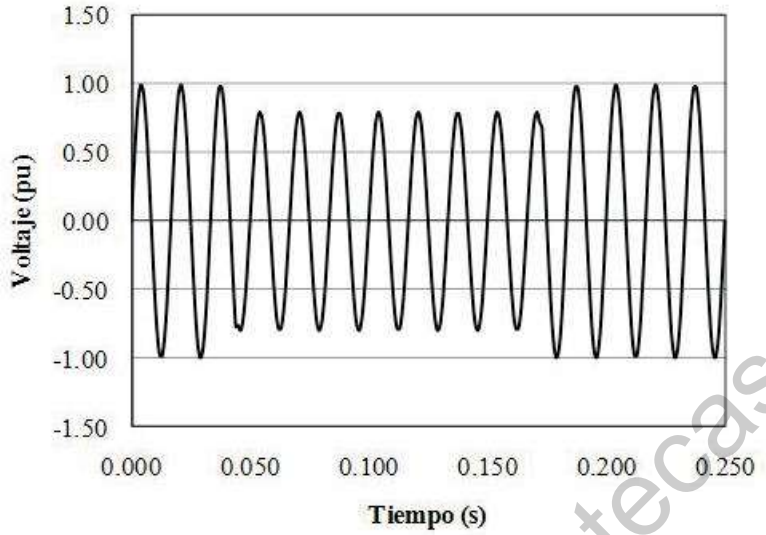


Figura II.2. Forma de la onda de tensión de un voltaje sag (Sanchez & Caicedo, 2008).

Las caídas de voltaje también son causadas por grandes cambios de carga o el arranque de motores grandes. Esta gran corriente provoca una caída de voltaje a través de la impedancia del sistema como se muestra el efecto de arranque de un motor grande en la Figura II.3 (IEEE Std 1159, 2009).

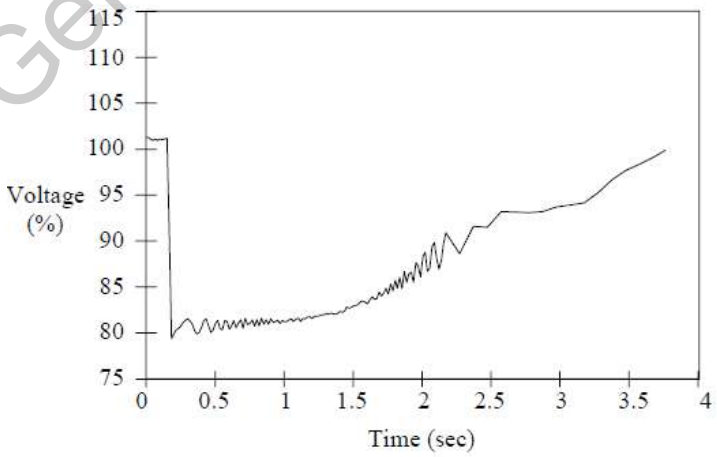


Figura II.3. Disturbio sag temporal causado por el arranque de un motor (IEEE Std 1159, 2009).

2.1.4. Salto de Tensión

El salto de tensión es caracterizado por el incremento del valor eficaz de la tensión en el orden de 1.1 a 1.8 pu como se muestra en la Figura II.4, con una duración entre 0.5 ciclos a un minuto. El salto de tensión es generalmente asociado a condiciones de falla desequilibrada en el sistema, salida de grandes bloques de carga y entrada de bancos de capacitores, y puede causar degradación y falla inmediata del aislamiento de los equipos y fuentes electrónicas, quema de varistores y de diodos Zener (Holguin & Gomezcoello, 2010).

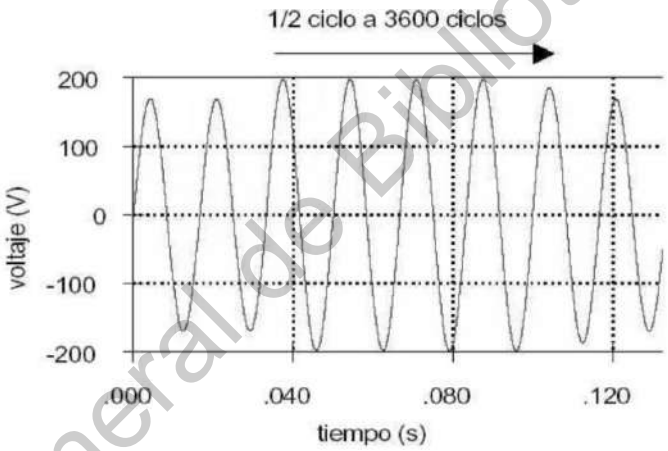


Figura II.4. Curva de espectro de swell (Holguin & Gomezcoello, 2010).

2.1.5. Características de duración y magnitud de voltaje

En el Cuadro II.1 se muestran las características de los disturbios de voltaje dadas por el estándar IEEE 1159 que es similar a la norma internacional IEC 61000-4-30 donde es basada la especificación de la CFE. La caracterización depende de la duración del evento, la magnitud de pu obtenida con el RMS para clasificar los fenómenos eléctricos sag, interrupción y swell en la red eléctrica, este último su rango cambia conforme a la categoría.

Cuadro II.1. Características de duración y magnitud de voltaje.

Categoría	Duración Típica	Magnitud de Voltaje Típico
Instantáneo	0.5 - 30 ciclos	0.1 - 0.9 pu (sag) 1.1 – 1.8 pu (swell)
Momentáneo	30 ciclos – 3 s 30 ciclos – 3 s 0.5 ciclos – 3 s	0.1 – 0.9 pu (sag) 1.1 – 1.4 pu (swell) < 0.1 pu (interrupción)
Temporal	>3 s – 1 min	0.1 – 0.9 pu (sag) 1.1 – 1.2 pu (swell) < 0.1 pu (interrupción)

*Valores obtenidos del estándar IEEE 1159 (IEEE Std 1159, 2009)

2.2. Distorsión Armónica

Se conoce como distorsión armónica a la deformación de la onda de su característica sinusoidal pura, originado por múltiplos enteros de la frecuencia. Sin embargo, otras componentes se encuentran en la distorsión de los voltajes o corrientes que no son múltiplos enteros de la frecuencia a la cual el sistema de suministro está diseñado para funcionar son conocidos como inter-armónicos.

La THD es definida como la raíz cuadrada de la suma de los cuadrados de las magnitudes de las componentes armónicas individuales dividido por la magnitud de la componente fundamental, como se muestra en la ecuación (2):

$$THD = \frac{\sqrt{\sum_{i=1}^{H} C_i^2}}{C_f} \qquad (2)$$

Donde C_i es la componente armónica individual, C_f la componente fundamental y H el número del armónico.

En las siguientes secciones se hace mención de las características e importancia en la obtención de los armónicos e inter-armónicos y la manera en que se generan.

Un análisis matemático (Fourier) de ondas distorsionadas por cargas no lineales muestra que las ondas están compuestas de la onda seno fundamental junto con una o más ondas con una frecuencia múltiplo entero de la frecuencia fundamental. Por ejemplo, la suma de una onda fundamental de 60 Hz, una de 180 Hz y otra de 300 Hz resulta una onda distorsionada (Rojas et al., 2014). Estos múltiplos de la frecuencia fundamental son llamados armónicos.

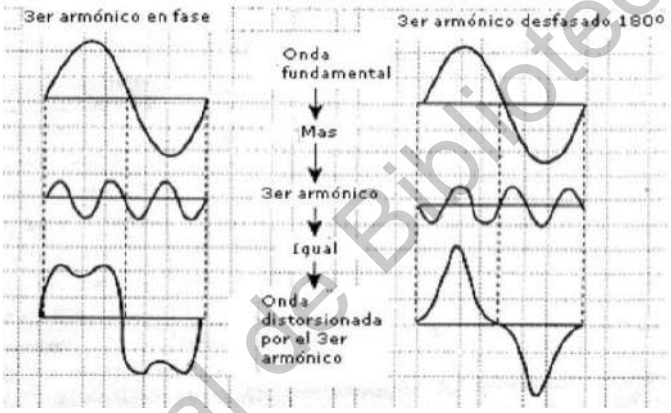


Figura II.5. Distorsión de una onda fundamental por armónicos (Holguin & Gomezcoello, 2010).

Las formas de onda no senoidales consisten y son descompuestas en un número finito de ondas seno puras de diferentes frecuencias. En la Figura II.5, se muestra la combinación de una forma de onda de voltaje senoidal y una forma de onda de tercer armónico que crea una forma de onda armónicamente distorsionada la cual dependerá del desplazamiento de la fase del armónico (Holguin & Gomezcoello, 2010).

2.3. Desequilibrio

El desequilibrio en un sistema trifásico se define como la relación de la magnitud del componente de secuencia negativa a la magnitud del componente de secuencia positiva,

expresada en porcentaje. Esta definición se aplica para voltaje como para corriente (Std 1159, 2009).

De acuerdo a la norma IEC 61000-4-30, el porcentaje de desequilibrio u_2 se expresa usualmente como (3), donde U_2 corresponde a la componente de secuencia negativa de la tensión de línea y U_1 corresponde a la componente de secuencia positiva de la tensión de línea (Casaravilla & Echinope, 2010).

$$u_2 = 100 * \frac{U_2}{U_1} \quad (3)$$

Las componentes de secuencia se derivan de las componentes de fase a través de la transformación de Fortescue (4).

$$\begin{bmatrix} \overline{U_1} \\ \overline{U_2} \\ \overline{U_0} \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 1 & a & a^2 \\ 1 & a^2 & a \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} \overline{U_{ab}} \\ \overline{U_{bc}} \\ \overline{U_{ca}} \end{bmatrix} \quad (4)$$

donde $a = e^{\frac{j2\pi}{3}}$.

Para determinar las componentes de secuencia de las tensiones de un sistema trifásico es necesario conocer las tres componentes de fase de las tensiones, así como los ángulos de desfase entre ellas. Esto implica que en una zona suministrada por un sistema trifásico, donde se deba controlar el desbalance, debería instalarse un equipo que sea competente de registrar los parámetros (Sánchez et al., 2012). A los efectos de simplificar este conjunto de medidas, la norma IEC 61000-4-30 y el estándar IEEE 1159 proponen fórmulas de cálculo alternativas. Ambos documentos tienden a la mínima de parámetros para estimar el índice de desbalance, lo cual se logra con distintos grados de exactitud en cada caso.

Existe una fórmula alternativa según la IEC 61000-4-30 la cual propone que el índice de desbalance definido en (3) puede calcularse como (5) para un sistema trifásico (Casaravilla & Echinope, 2010):

$$u_2 = 100 * \sqrt{\frac{1-\sqrt{3-6\beta}}{1+\sqrt{3-6\beta}}} \quad (5)$$

Donde β se calcula según (6)

$$\beta = \frac{U_{ab_{fund}}^4 + U_{bc_{fund}}^4 + U_{ca_{fund}}^4}{\left(U_{ab_{fund}}^2 + U_{bc_{fund}}^2 + U_{ca_{fund}}^2\right)^2} \quad (6)$$

Donde las tensiones $U_{ij_{fund}}$ corresponden a la componente fundamental de la tensión entre la fase i y la fase j.

Se propone el uso de la Transformada de Fortescue (4), desarrollando la secuencia positiva y negativa considerándola (3) para desbalance, eliminamos la multiplicación de $\frac{1}{3}$ considerado en la división, y factorizando obtenemos las componentes de frecuencia negativa y positiva (7) y (8).

$$\overline{U_1} = \overline{U_{ab}} + \overline{U_{bc}} \cdot a + \overline{U_{ca}} \cdot a^2 \quad (7)$$

$$\overline{U_2} = \overline{U_{ab}} + \overline{U_{bc}} \cdot a^2 + \overline{U_{ca}} \cdot a \quad (8)$$

Utilizando ecuaciones de Euler sustituyendo en a y separando de $\overline{U_{xy}}$ la parte real e imaginaria como se puede observar en las siguientes ecuaciones:

$$\overline{U_{xy}} = U_{xr} - U_{yr} + U_{xi} - U_{yi} \quad (9)$$

$$a = e^{\frac{j2\pi}{3}} = \cos\left(\frac{2}{3}\pi\right) + j \cdot \sin\left(\frac{2}{3}\pi\right) \;\; \therefore \;\; a^2 = \cos\left(\frac{4}{3}\pi\right) + j \cdot \sin\left(\frac{4}{3}\pi\right) \quad (10)$$

Donde $U_{(x,y)r}$ es la parte real y $U_{(x,y)i}$ la parte imaginaria. Sustituyendo (9) y (10) en (7) y (8), y factorizando obtenemos la componente de secuencia positiva parte real e

imaginaria mostradas en (11) y (12) y la componente de secuencia negativa mostradas en (13) y (14):

$$U_{1r} = U_{ar} - U_{br} - \frac{1}{\sqrt{3}}(U_{ai} + U_{bi} - 2 * U_{ci}) \quad (11)$$

$$U_{1i} = U_{ai} - U_{bi} + \frac{1}{\sqrt{3}}(U_{ar} + U_{br} - 2 * U_{cr}) \quad (12)$$

$$U_{2r} = U_{ar} - U_{br} + \frac{1}{\sqrt{3}}(U_{ai} + U_{bi} - 2 * U_{ci}) \quad (13)$$

$$U_{2i} = U_{ai} - U_{bi} - \frac{1}{\sqrt{3}}(U_{ar} + U_{br} - 2 * U_{cr}) \quad (14)$$

Del porcentaje de desequilibrio de acuerdo a la norma IEC 61000-4-30 (3) se sustituye la secuencia negativa y positiva con las ecuaciones reales e imaginarias, además, queda recordar que la separación permite realizar la factorización en números complejos, por lo tanto, se tiene que aplicar el valor absoluto de estas ecuaciones que es la raíz cuadrada de la suma del cuadrado de la parte real e imaginaria. Como resultado final para el cálculo del porcentaje de desequilibrio obtiene la siguiente ecuación:

$$u_2 = 100 * \sqrt{\frac{U_{2r}^2 + U_{2i}^2}{U_{1r}^2 + U_{1i}^2}} \quad (15)$$

2.5. Especificación G0000-48

El instrumento de medición, debe de realizar la medición de calidad de energía de por lo menos los siguientes parámetros (João et al., 2007):

a) Frecuencia del sistema trifásico de tensiones.

b) Magnitud de la tensión (variaciones de tensión) por fase.

c) Depresiones de tensión sag por fase.

d) Incrementos de tensión (swell) por fase.

e) Interrupciones de tensión por fase.

f) Desbalance de sistema trifásico de tensiones.

g) Armónicos de tensión y corriente (individuales y totales) por fase hasta armónica 25.

Se debe suministrar la opción de Parpadeo (flicker) por fase, cuando se indiquen características particulares.

El cálculo se debe realizar con base en lo establecido en la norma IEC 61000-4-30 para medidores Clase S y NMX-J-610-3-6 ANCE.

En el Cuadro II.2 se muestran los intervalos de ajuste para detectar valores fuera de rango y los valores a registrar, ya sean mínimos, máximos, promedio y de tiempo indicados en el documento de la CFE.

Cuadro II.2. Parámetros de Calidad de Energía (eventos) (CFE, 2010).

Parámetro de Calidad (Eventos)	**Tipo**	**Magnitud**	**Duración**
Interrupción	Momentánea	< 0.1 pu	1 ciclo – 3 s
	Temporal	< 0.1 pu	3s – 1 min
	Sostenida	0%	> 1 min.
Depresiones de tensión (Sag)	Instantánea	0.1 – 0.9 pu	3 – 30 ciclos
	Momentánea	0.1 – 0.9 pu	30 ciclos – 3s
	Temporal	0.1 – 0.9 pu	3 s – 1 min.
Incremento de tensión (Swell)	Instantánea	1.1 – 1.8 pu	3 – 30 ciclos
	Momentánea	1.1 – 1.4 pu	30 ciclos – 3 s
	Temporal	1.1 – 1.2 pu	3 s – 1 min.

*Valores tomados en el documento G0000-48 (CFE, 2010).

Para el sistema de Baja California Sur, el valor será +/- 0.8 %.

El medidor multifunción debe almacenar el promedio de registros históricos de cada 10 minutos, durante un mínimo de 35 días con estampa de tiempo de los siguientes parámetros:

a) Frecuencia del sistema trifásico de tensiones.

b) Magnitud de tensión por fase.

c) Parpadeo (flicker) por fase, indicando valores fuera de rango (Aplicable a Clase S IEC 61000-4-30).

d) Desbalance de tensión. El valor de desbalance debe ser registrado cuando exceda el umbral configurado.

e) Distorsión armónica total de tensión por fase.

f) Distorsión armónica total de corriente por fase.

Los parámetros de calidad de la energía en estado estable se muestran en el Cuadro II.3, para variaciones de larga duración, desbalance, distorsión armónica total, variaciones de frecuencia y flicker, este último con una ventana de medición de 2 horas.

Cuadro II.3. Parámetros de Calidad de Energía (estado estable) (CFE, 2010).

Parámetro de Calidad (Estado Estable)	**Tipo**	**Ventana de Medición**	**Variación**
Variación de tensión de larga duración	Incremento de Tensión	10 min.	+ 10 % Vnom
	Decremento de Tensión	10 min.	- 10 % Vnom
Desbalance	Tensión	10 min.	2%
Distorsión Armónica Total	Tensión	10 min.	6.5 %
Variación de Frecuencia (1)	Alta Frecuencia	10 min.	+ 0.5 %
	Baja Frecuencia	10 min.	- 0.5 %
Parpadeo (Flicker)	< 25 Hz	2 horas	1

*Valores tomados en el documento G0000-48 (CFE, 2010).

Debe tener capacidad de almacenamiento para registrar al menos 200 eventos de calidad de la energía en memoria circular (CFE, 2010).

Parámetros en estado transitorio, indicando magnitud y duración:

a) Depresiones de tensión Sag por fase. El ajuste del umbral debe estar dentro de un intervalo del 10 % al 90 % del valor nominal y duración de 3 a 3,600 ciclos en un sistema de 60 Hz.

b) Incrementos de tensión (swell) por fase. El ajuste del umbral debe estar dentro de un intervalo del 110 % al 180 % del valor nominal y duración de 3 a 3,600 ciclos en un sistema de 60 Hz.

c) Interrupciones de tensión por fase.

2.6. Norma IEC 61000-4-30

La norma IEC 61000-4-30, define los métodos para las mediciones e interpretación de los resultados en los parámetros de calidad de energía, dentro de los 50/60 Hz en ca (corriente alterna) en los sistemas de suministros de energía. Los parámetros de calidad de energía considerados por la norma son en frecuencia, magnitud del suministro de voltaje, flicker o parpadeo de sistemas eléctricos, suministro de voltaje sag y swell, interrupciones de voltaje, voltajes transitorios, desbalance en suministros de voltaje, armónicos e inter-armónicos, principalmente señales en el suministro de voltaje y cambios rápidos de voltaje.

Entre los parámetros de calidad de la energía, se debe de tomar en cuenta para la valorización de los armónicos e inter-armónicos, los requerimientos para estas mediciones está definido por el estándar IEC 61000-4-7 donde especifica lo siguiente:

1) La estructura general de los instrumentos de medición.

2) La exactitud de instrumentos.

3) Las características de las señales para ser medidas.

4) Los tipos de mediciones.

El estándar o norma define la frecuencia del armónico como una integral múltiple de la frecuencia fundamental (Ramírez & Cano, 2006), en el trabajo se han hecho referencias con el uso de la FFT con un tiempo de ventana de 200 ms equivalente a 10 veces (50 Hz) o 12 veces (60 Hz) del periodo fundamental, cada espectro deberá componer su medición con un rango de resolución superior e inferior de 5 Hz. Las líneas de los espectros tienen que ser procesadas adecuadamente en el orden para adquirir grupos/subgrupos de armónicos e inter-armónicos. Las tomas provienen de valores, los cuales incluyen los efectos de fluctuación de los componentes armónicos. Se hace mención que los armónicos de tensión y corriente tanto individuales como totales por fase debe de ser hasta la armónica 25 en este caso.

III. METODOLOGÍA

En este capítulo se expone la manera en que se desarrolla el proyecto y la metodología que se sigue.

La metodología está esencialmente constituida en tres partes principales:

- Arquitectura general.
- Implementación.
- Comparación.

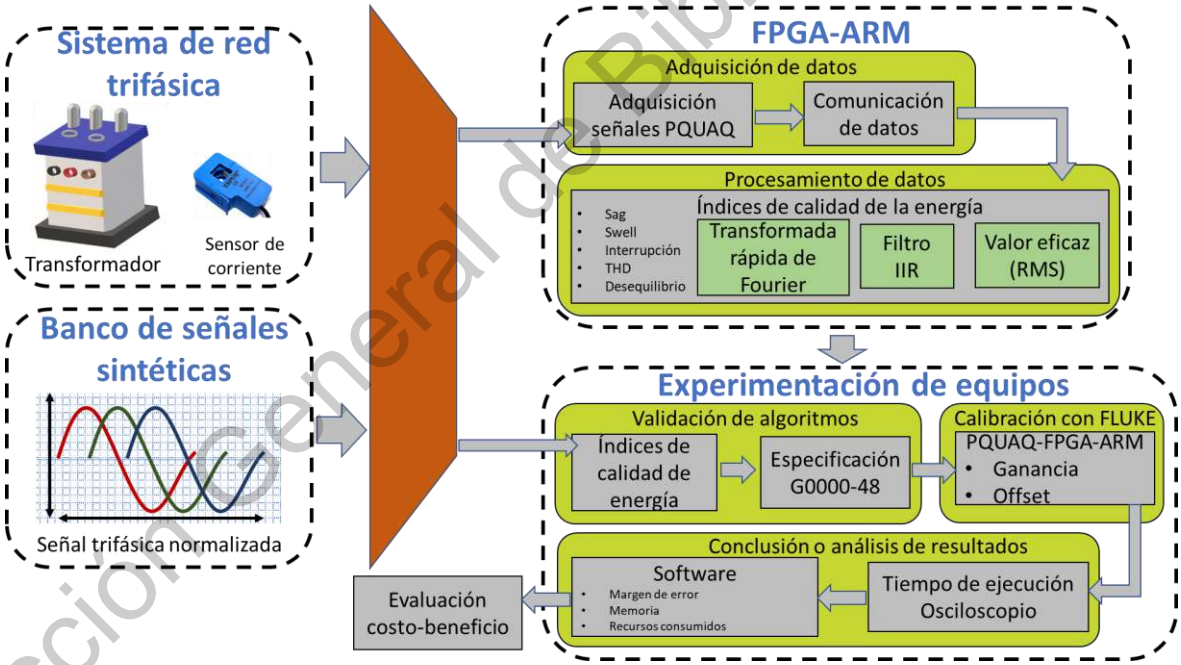


Figura III.1. Metodología general del proyecto.

La metodología propuesta en el proyecto como se puede observar en la Figura III.1, el análisis es dado por medio de datos almacenados que validan los algoritmos de calidad de la energía, con la especificación G0000-48 para los parámetros: variaciones de corta duración

(sag, swell e interrupción) para un sistema trifásico, distorsión armónica total por fase de corriente y voltaje, y desequilibrio en fases de voltaje. Con la validación de los algoritmos lo posterior es la implementación en dos tecnologías diferentes, una de ellas es basada en hardware y la otra en software, con ello se realizan las comparaciones con ayuda de diversos sistemas de medición como son: osciloscopios, simulaciones, bitácoras de eventos y medidores de calidad de la energía eléctrica comerciales. Finalmente, con los dispositivos desarrollados se pondrán a prueba en un sistema de red trifásica con ayuda de un sistema de adquisición de datos, verificando que los dispositivos puedan operar en el monitoreo continuo, con esta prueba y los resultados dados de cada dispositivo obtendremos cuál de ellos tiene un mejor costo-beneficio en la implementación de un medidor de calidad de la energía.

3.1. Arquitectura general

El primer bloque, consiste en la implementación de equipos de medición de calidad de la energía eléctrica, mediante software y hardware que están conformados por tres fases o etapas. En la primera etapa, se obtienen las señales de voltaje y corriente de un sistema de red trifásica, mediante un sistema de adquisición de datos y tres sensores de corriente no invasivos. Dichas señales contienen información de la tensión, frecuencia y desplazamiento de fase de la señal eléctrica. La señal de voltaje es conectada directamente a las fases y al neutro de la red trifásica, los sensores de corriente son conectados en la carga de cada fase para obtener la corriente por medio de inducción electromagnética. Posteriormente, como segunda fase, se realiza la configuración en el muestreo de las señales eléctricas y la comunicación de la tarjeta de adquisición con los dos dispositivos, siendo así posible el procesamiento de la información en los dispositivos cuando esta sea adquirida.

Obtenida la información de las señales de voltaje y corriente, como tercera fase, se realiza el procesamiento individual de cada canal aplicando diversas técnicas de procesamiento como son: valor eficaz, filtrado digital de las señales por medio de un filtro

IIR Butterworth y técnica de descomposición de señales del dominio de tiempo a dominio de la frecuencia, como lo es la transformada rápida de Fourier. De tal manera, con estás técnicas de procesamientos de la información, se desarrollan las estructuras digitales y diagramas de flujo que conforman los índices de calidad de la energía eléctrica propuestos para ser implementados en un FPGA y un procesador basado en la arquitectura RISC (ARM). Las estructuras digitales anteriormente mencionadas se describen mediante lenguaje de descripción en hardware (HDL) y con los diagramas de flujo la programación para el ARM mediante el lenguaje de programación C.

Una vez integrada la metodología del bloque anterior, el segundo bloque las implementaciones de cada dispositivo se tendrán tres fases o etapas las cuales son: pruebas de validación, calibración, y por último realizar la comparación de desempeño y/o carga computacional. En la primera etapa, se pondrá a prueba los algoritmos implementados en los dispositivos mediante bancos de datos de señales sintéticas almacenadas para la validación de los índices de calidad de energía eléctrica y el margen de error. Ya validados los algoritmos, como segunda fase se tiene la calibración de cada dispositivo mediante los resultados entregados de cada uno, y tomando como referencia el equipo de calidad de la energía se realiza el ajuste correspondiente. Finalmente, se contrastan las dos diferentes tecnologías en tiempo de ejecución, recursos consumidos, memorias y margen de error, mediante el uso de osciloscopio, software de programación y bitácoras de eventos.

3.1.1. Sistema de red trifásica

Los dispositivos se pondrán a prueba por medio de bancos de pruebas de señales reales, que han sido obtenidas de la base de datos de la IEEE (IEEE, 2017), estos bancos fueron adquiridos de una red trifásica de 60 Hz, donde partes de las señales presentan perturbaciones de diferente clasificación. Con los bancos de prueba de las señales reales, se verifica si los dispositivos implementados pueden realizar la identificación de los disturbios establecidos.

Por medio de un sistema de adquisición de datos, se realiza la conexión de los dispositivos en tiempo real con el sistema de red trifásica, donde se pretende hacer el monitoreo en línea y la parametrización de la red eléctrica.

3.1.2. Bancos de señales sintéticas

Se maneja un conjunto de señales sintéticas de una red trifásica, que nos permita tener una referencia de los índices de calidad de la energía que se quieren validar en la implementación de los algoritmos. Estos datos sintéticos serán normalizados al formato 2.14, quiere decir, 2 bits para la parte entera y 14 bits en la parte fraccionaria. Otras de las características que tendrán nuestras señales sintéticas es la frecuencia de oscilación de 60 Hz simulando la frecuencia de la red eléctrica suministrada en el país, y la tasa del muestreo de generado es la misma que maneja el sistema de adquisición, el cuál es de 8000 muestras por segundo.

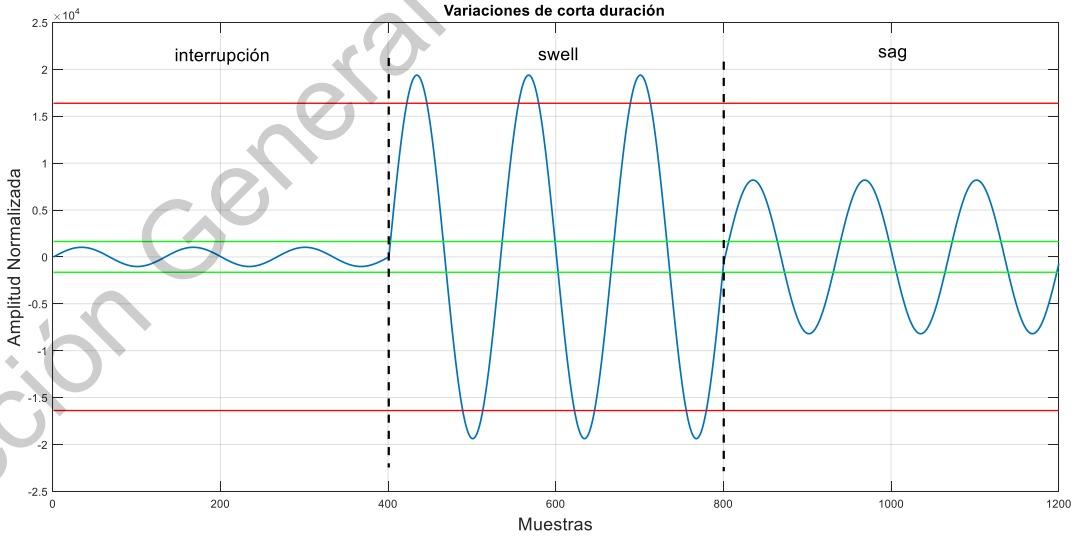


Figura III.2. Señales sintéticas sag, swell e interrupción.

Con las señales sintéticas se realizó la validación del bloque de variaciones de corta duración, se pueden observar en la Figura III.2, se marcaron límites de una señal ideal de

color rojo y una con interrupción de color verde, se dividieron fragmentos de la señal para poder distinguir las características principales simulada de los disturbios sag, swell e interrupción. El tamaño de la ventana total de datos es de 6144, donde cada disturbio de calidad de la energía eléctrica tiene una longitud de 2048 muestras, se utiliza una señal senoidal pura, pero con amplitud variada de pico, definida de la siguiente manera:

- Señal 1: 6.25% Interrupción (0.0625 amplitud normalizada)
- Señal 2: 118.31% Swell (1.1831 amplitud normalizada)
- Señal 3: 50% Sag (0.5 amplitud normalizada)

La señal sintetizada se aplicó para todos los canales tanto de voltaje y corriente del algoritmo. Aunque la especificación en las variaciones de corta duración es solo aplicada en las fases de voltaje, pero de igual manera el boque está encargado de entregar los resultados RMS de todos los canales.

Por otro lado, en el diseño de la señal sintética para distorsión armónica total, se propuso que los datos contengan dos armónicos de la frecuencia fundamental del sistema al igual que el bloque de variaciones de corta duración se utilizó una frecuencia de 60 Hz, con amplitud variada de pico tanto en la frecuencia fundamental como en los armónicos, sin sobrepasar los rangos de medición del bloque. Los armónicos empleados son el cuarto y décimo de la señal fundamental, y están diseñados por las siguientes ecuaciones:

$$Frecuencia\ fundametal = 0.9 * \sin(w * t + \emptyset) \quad (16)$$

$$Cuarto\ armónico = 0.12 * \sin(5 * w * t + \emptyset) \quad (17)$$

$$Décimo\ armónico = 0.03 * \sin(11 * w * t + \emptyset) \quad (18)$$

Donde:

w: $2 * \pi * f$

f: Frecuencia fundamental (60 Hz)

t: La cantidad de datos sobre la tasa de muestreo (ventana).

∅: Ángulo de fase.

Como se puede observar en las ecuaciones (17) y (18) para los armónicos, siempre se considera el número del armónico más uno, debido a que la frecuencia fundamental no se considera como armónico, por ejemplo, en el caso de los armónicos cuarto es cinco y el décimo es once el coeficiente que multiplica *w*. En el caso de las señales sintetizadas utilizadas en este bloque el ángulo de desfase en todos los canales es igual a cero.

En la Figura III.3, se muestra las señales armónicas y fundamental por separado, se puede observar que la señal de frecuencia fundamental es de color azul, el cuarto armónico de color rojo y el décimo armónico de color amarillo. Además, la señal resultante que es utilizada de la sumatoria de las tres señales sintéticas es representada de color morado.

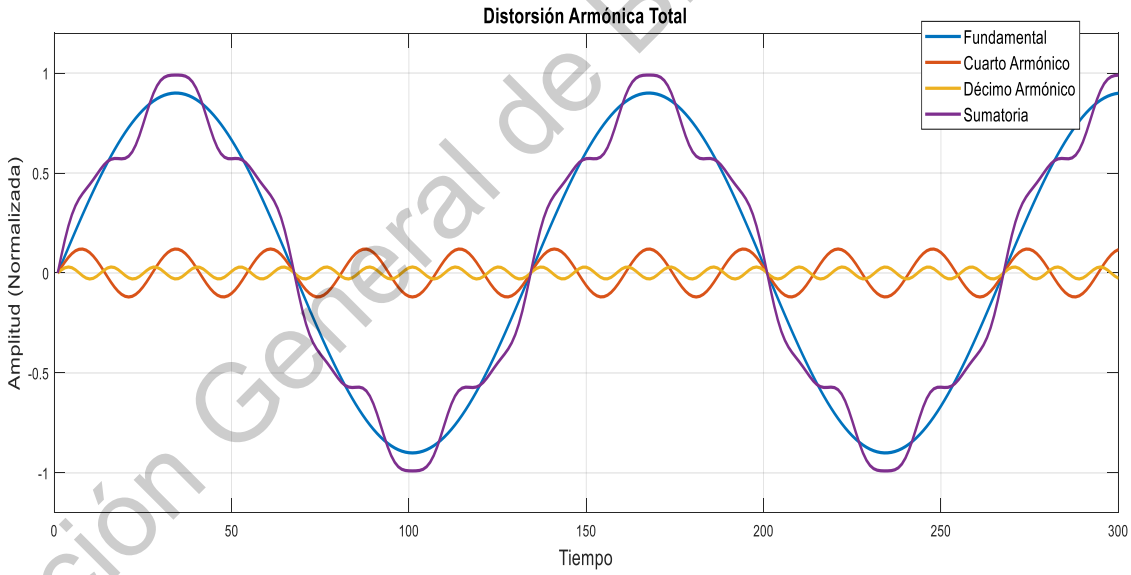


Figura III.3. Señal sintética con cuarto y décimo armónico.

El tamaño de la ventana que se manejó para la medición del THD, es de 8192 datos (muestras) para todos los canales de voltaje y corriente. Con la finalidad de verificar que el

resultado sea el mismo en cada canal. El resultado esperado del THD en cada canal es de 17.98% utilizando la señal sintética diseñada.

En el caso del bloque desbalance en fases, se continua con el mismo criterio manejado en el bloque de THD, pero con un cambio significativo en el ángulo de desfase. Para los canales de voltaje y corriente de la fase, el valor del ángulo es el siguiente:

- Va-Ia. El ángulo desfase es 0°.
- Vb-Ib. El ángulo desfase es -120° $\left(-\frac{1}{3}\pi\right)$.
- Vc-Ic. El ángulo desfase es 120° $\left(\frac{2}{3}\pi\right)$.

Con ello tendremos la simulación de 3 señales con la misma cantidad de armónicos, pero con un desfase entre una con otra de 120° como se espera de una red trifásica. El resultado en el bloque es de 0.36% de desbalance entre las fases.

Por último, en el diseño del banco de pruebas se utilizaron las señales sintéticas de la manera mostrada en la Cuadro III.1, las cuales están basadas en los criterios de medidores de calidad de la energía Clase S para pruebas, dichos criterios se pueden encontrar en el documento de las especificaciones de CFE G0000-48 en la tabla 15 (CFE, 2010). Las características que deben de tener las señales tienen un seguimiento para el cumplimiento del documento.

Con las particularidades de las señales mostradas en el cuadro, verificar que los algoritmos implementados en los dispositivos cumplan con la incertidumbre que establece los medidores de Clase S, además se utiliza como punto de comparación en los resultados entregados de cada dispositivo y sea apto como medidor de calidad de la energía.

A continuación, se hace una breve explicación del contenido en el Cuadro III.1. En los intervalos de medición el porcentaje representa la amplitud pico normalizada, los parámetros swell, sag e interrupción la cantidad de ciclo va de 3 a 3600 y la incertidumbre que se consideran es una variación máxima de un ciclo y 0.5% en la amplitud como resultado.

Cuadro III.1. Parámetros de señales de prueba.

Parámetro	**Método de medición 610004-30**	**Incertidumbre Clase S**	**Intervalo de medición**
Depresiones de Tensión (sag)	5.4.1	± 0.5% Vn ± 1 ciclos	10 % Vn a 3, 6, 1800 y 3600 ciclos 50 % Vn a 3, 6, 1800 y 3600 ciclos 90 % Vn a 3, 6, 1800 y 3600 ciclos
Incremento de tensión (swell)	5.4.1	± 0.5% Vn ± 1 ciclos	110 % Vn a 3, 6, 1800 y 3600 ciclos 150 % Vn a 3, 6, 1800 y 3600 ciclos 180 % Vn a 3, 6, 1800 y 3600 ciclos
Interrupción	5.5.1	± 1 ciclos	9 % Vn a 3, 6 y 3600 ciclos
Desbalance	NA	± 0.3 %	73 % ± 0.5 % Vn fase A 80 % ± 0.5 % Vn fase B 87 % ± 0.5 % Vn fase C Todos los ángulos de fase a 120° V= 5.05 % 152 % ± 0.5 % Vn fase A 140 % ± 0.5 % Vn fase B 128 % ± 0.5 % Vn fase C Todos los ángulos de fase a 120° V= 4.95 %

*Valores tomados en el documento G0000-48 (CFE, 2010).

En la parte de desequilibrio (desbalance) en fases, las señales sintéticas de voltaje que se ponen a prueba en los dispositivos, en conjunto con el algoritmo de THD, el valor pico es como se muestra a continuación:

Señal 1:

- Fase A (73 %)
- Fase B (80%)
- Fase C (87%)

Señal 2:

- Fase A (152 %)
- Fase B (140%)
- Fase C (128%)

Todos los ángulos de fase de la señal sintética están desfasados a ±120° de la fase de referencia, por lo tanto, el resultado que se espera de la señal 1 en desbalance es de 5.05% y la señal 2 de 4.95%. La incertidumbre o error límite del resultado dado por cada dispositivo tendrá que ser de ±0.3% al valor ideal de cada señal. En la Figura III.4, se muestran como ejemplo la señal 2, con las tres fases de amplitud indicada, la señal solo será aplicada a las fases de tensión y no a las de corriente. Los limites mostrados es el valor ideal de una señal normalizada de color rojo y de color verde el rango donde se considera interrupción.

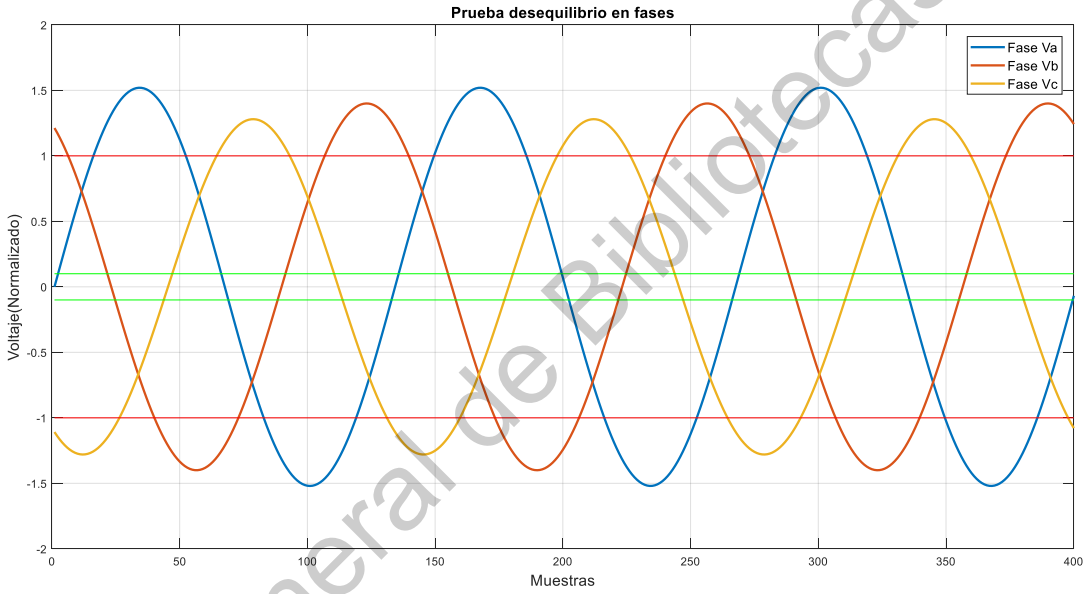


Figura III.4. Señal sintética desequilibrio de fases.

La dimensión de la ventana que deben de tener las señales sintéticas para el cálculo de desbalance por fase es igual para distorsión armónica total, el tamaño por fase es de 8192 muestra o datos. Es debido a la cantidad de datos necesaria por ventana en el procesamiento con la FFT, este rango de ventana tiene como finalidad mantener el límite de incertidumbre o el margen de error en el resultado del desbalance en fases, establecido en la especificación G0000-48.

3.2. Implementación

En esta sección, se muestra con mayor detalle el método utilizado para la implementación de los algoritmos de calidad de la energía, en dos diferentes tecnologías. Se explicará el diagrama a flujo general, que toma parte directa en la programación del dispositivo ARM (dispositivo en software), así como la descripción en los diseños propuestos para el FPGA (dispositivo en hardware) en cada módulo y sus máquinas de estados. Los diagramas y diseños están basados a la especificación y al sistema de adquisición en las muestras por segundo, orden de transferencia de información por canal y tamaño de resolución en los datos.

3.2.1. ARM

En la implementación de los algoritmos en el dispositivo ARM, se utilizó una tarjeta de desarrollo *Hercules*. Cuenta con la capacidad requerida en el almacenamiento de datos y procesamiento de las instrucciones para la implementación en conjunto de los algoritmos de calidad de la energía y además que el dispositivo sea de grado industrial (aplicaciones de control y automatización industrial). En la programación del dispositivo, se requiere como herramienta adicional el manejo de plataformas de programación, donde se desarrollan los diagramas a flujo en lenguaje C y la carga de los periféricos necesarios para el funcionamiento del dispositivo.

3.2.1.1 Diagramas de flujo

A continuación, se presentan los diagramas de flujo a implementar en el dispositivo ARM y en el dispositivo FPGA.

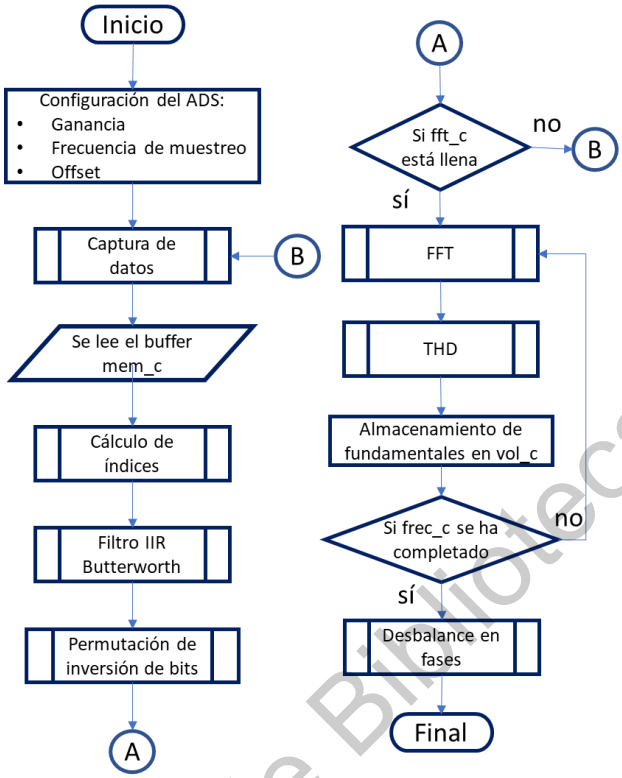


Figura III.5. Diagrama de flujo principal.

En la Figura III.5, se muestra el diagrama de flujo principal en conjunto con los algoritmos para medir los parámetros requeridos en el medidor de calidad de la energía eléctrica. La configuración del ADS en el diagrama de flujo está determinado por el dispositivo utilizado en la captura de datos de la señal eléctrica.

La implementación del diagrama de flujo principal será de la siguiente forma:

- *Configuración del ADS*. Antes de realizar la manipulación de los datos entregados por el dispositivo es necesario verificar los siguientes aspectos del mismo. Ganancia y *offset*, la mayoría de los ADS cuentan con estas configuraciones para las ganancias y *offset* de cada canal de muestreo, se utiliza como herramienta un generador de señales en los canales de voltaje y así poder ajustar la ganancia de los canales, el ajuste tiene importancia en la medición del desbalance y variaciones de corta duración (Cálculo de índices) para que el resultado sea correcto. Por otra parte, el *offset* solo

es necesario realizar el aterrizaje a tierra para eliminarlo, con ello se asegura que la medición de medio ciclo de la señal sea adecuada. La frecuencia de muestreo en algunos dispositivos puede ser modificada, si es el caso lo recomendado es que la frecuencia de muestreo sean múltiplo de la frecuencia de la señal eléctrica medida y así disminuir el error de medición, la frecuencia de muestreo es un dato utilizado para la medición del THD y el desequilibrio de la señal eléctrica que se menciona posteriormente.

- *Captura de datos*. Es requerido el almacenamiento de los datos medidos en cada canal, debido al tiempo de ejecución que les toma a los algoritmos en el procesamiento de la señal, con ello se evita la pérdida de información en la medición o valores erróneos en los resultados. En la Cuadro III.2, se recomienda un buffer para cada canal con el tamaño de la palabra del muestreo con una capacidad de 4096 datos, llamado "mem_c". En las siguientes secciones se puede observar el diagrama de flujo la manera en que se almacenan los datos.

Cuadro III.2. Buffers recomendados en el diagrama a flujo.

Buffer	**Descripción**
mem_c	Buffer [6][4096], almacenamiento individual de los 6 canales.
sum_c	Buffer [6], acumulador del producto de cada canal.
xn, yn	Buffer [6][n], almacenamiento del producto de coeficientes del filtro digital de orden "n".
fft_c	Buffer [12][8192], almacenamiento de los canales filtrados y con permutación de inversión de bits.
frec_c	Buffer [3][2], valores reales e imaginarias de las 3 fases de voltaje.

- *Leer buffer "mem_c".* A la vez que se realiza el almacenamiento de datos en el buffer es recomendado ir leyendo el buffer cuando se tenga por lo menos un dato en todos los canales.

- *Cálculo de índices.* Se refiere a los índices de variaciones de corta duración, en esta parte es necesario un buffer más del doble del tamaño de la palabra del muestreo para cada canal debido a la multiplicación y acumulación de éste, ya que si no es así es muy posible que ocurra un desbordamiento en los bits. El cálculo de los índices se encarga de analizar el resultado RMS de medio ciclo en los canales de voltaje y si entra en algunos de los parámetros swell, sag o interrupción. El algoritmo a utilizar se explica en el diagrama a flujo de cálculo de índices.

- *Filtro IIR Butterworth.* Con los datos almacenados en el buffer "mem_c" se hará el procesamiento de la información con el filtro IIR Butterworth en cada señal. El filtro digital tiene como características un filtro pasa-bajas de orden 6, con frecuencia de corte de 2400 Hz, por si se requiere analizar hasta el armónico 40. Ya que el filtro IIR es una retroalimentación del mismo, se recomienda el almacenamiento de los datos actuales con el anterior de cada canal con una longitud *n* (cantidad de coeficientes del orden del filtro) de tipo flotante. Se pueden adquirir los valores del filtro por medio de la función *butter* de MATLAB, donde simplemente se requiere la frecuencia de corte, orden y tipo de filtro (pasa-bajas, pasa-altas y pasa-banda).

- *Permutación de inversión de bits.* Como se vio en la fundamentación teórica, la FFT en el procesamiento realiza una operación de multiplicación en mariposa o Radix-2 en este caso, por lo tanto antes de realizar la ejecución de la FFT el resultado del filtro digital en cada canal se maneja una permutación de inversión de bits en el almacenamiento del buffer para la FFT (fft_c), la inversión de bits consta en cambiar el orden de los bits, por ejemplo, teniendo solo el tamaño de cuatro bits "1010" al realizar la permutación de los bits nos da "0101", esto quiere decir que los bits menos significativos ahora son los más significativos. Con ello, al terminar el procesamiento la FFT el resultado será directo y no se necesitará reacomodar el orden de los datos entregados en el buffer. El buffer "fft_c" es de 12 columnas considerando la parte entera e imaginaria de cada canal y 8192 filas el tamaño requerido en el

procesamiento de la FFT. El tamaño de la ventana se considera así para el cumplimiento de la especificación G0000-48.

- *fft_c llena*. Se verifica que el buffer para la FFT este completo antes de realizar el procesamiento, si no es así se siguen leyendo los datos de los canales para filtrarlos y reacomodarlos.

- *FFT, THD y almacenamiento de frecuencias fundamentales*. Se realiza el procesamiento de la FFT por canal con el método Radix-2. En el procesamiento de THD, para conocer en qué índice del buffer "fft_c" se encuentran los armónicos de la señal y la frecuencia fundamental, se utiliza la siguiente ecuación:

$$H(x) = (F * W * x)/Fs \quad (19)$$

Donde *H(x)* si x toma el valor de uno es la frecuencia fundamental y si es diferente de uno son los armónicos, *F* frecuencia de la señal eléctrica analizada (60 o 50), *W* tamaño de la ventana que se utiliza en la FFT y *Fs* es la frecuencia de muestreo en la que se adquirieron los datos.

Teniendo las frecuencias fundamentales se almacenan en el buffer "frec_c" de los canales de voltaje, la parte real y la parte imaginaria. Para la obtención del THD en cada canal se realiza la acumulación de los productos obtenidos de la multiplicación en sí misma de la parte imaginaria y la parte real hasta llegar al armónico 25, como lo indica la especificación, una vez llegado al armónico se divide la suma con la frecuencia fundamental de la misma manera, multiplicando partes reales e imaginarias y sumándolo.

- *frec_m completa*. Si se ha completado el almacenamiento de la parte real e imaginaria de las frecuencias fundamentales de los canales de voltaje se calcula el desbalance de las tres fases, si no es así, se continúa el procesamiento con la FFT.

- *Desbalance en fases*. Teniendo por separado la parte real e imaginaria de la fundamental de los canales de voltaje. Se aplica directamente con la ecuación (15) desarrollada en la fundamentación teórica para desbalance en fases.

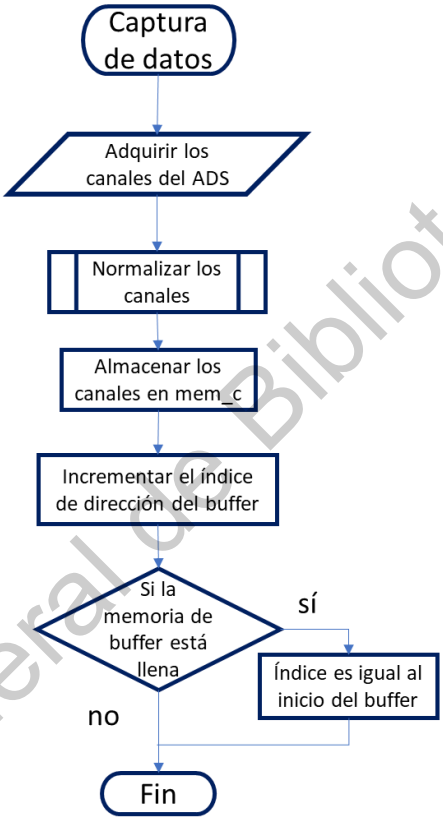


Figura III.6. Diagrama de flujo captura de datos.

En la captura de datos del sistema de adquisición, se siguen los pasos mostrados en el diagrama de flujo en la Figura III.6. Los datos obtenidos del sistema de adquisición se manipulan según la secuencia configurada en el dispositivo o en la tarjeta, además el tamaño de la palabra. En el caso del sistema de adquisición en la implementación de los algoritmos, la secuencia de la información muestreada de los canales es seis, los datos de cada canal tienen un tamaño de la palabra de 16 bits, donde los tres primeros son de corriente y los tres

restantes de voltaje. En la normalización de los canales, como se mencionó con anterioridad la información del canal tiene un tamaño de 16 bits, se consideran los primeros bits más significativos de izquierda a derecha como la parte entera y los 14 bits restantes la parte fraccionaria, es decir, nuestro formato a manejar es 2.14. en la normalización en este caso se realiza la división donde el canal es el numerador y 2^{14} es el divisor.

Al almacenar los datos, la cantidad de canales son los reglones del buffer "mem_m" y en las columnas la ventana de datos de 4096. Se almacenan los canales y, una vez completada la tarea, se incrementa el índice de columnas hasta llegar al tamaño de la ventana, el índice comienza desde cero para no tener pérdida de la información obtenida del sistema de adquisición.

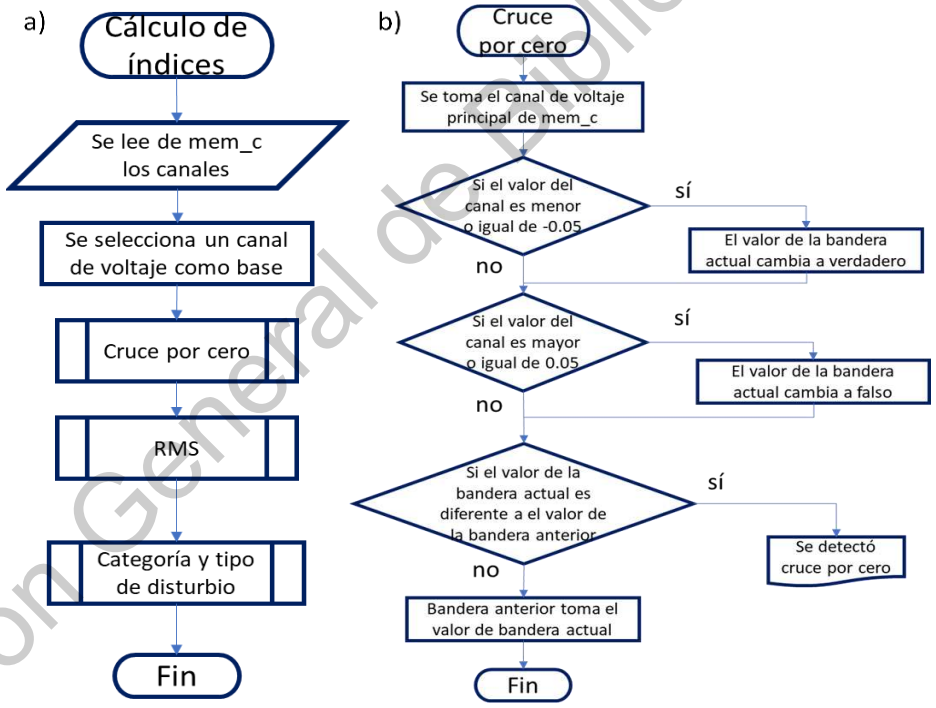


Figura III.7. Diagramas de flujo a) Cálculo de índices y b) Cruce por cero.

Las variaciones de corta duración son aplicadas únicamente en los canales de voltaje; sin embargo, la obtención del RMS es en todos los canales del ADS. La Figura III.7 muestra los diagramas de flujo en la manipulación de información para la obtención de los índices y

la funcionalidad del cruce por cero. El diagrama de flujo tiene como nombre cálculo de índices, debido ya que se pretende hacer el monitoreo de categoría (instantánea, momentánea y temporal) y tipo de disturbio (swell, sag e interrupción) con el valor RMS de los canales de voltaje.

De los valores previamente almacenados en le buffer “mem_c” se seleccionan los datos del reglón donde se encuentra el canal de voltaje Va considerado base para el detector de cruce por cero. El detector de cruce por cero por medio de una bandera que alerta al RMS donde éste termina de realizar la acumulación de todos los canales individuales almacenados en el buffer “sum_m” para entregar el resultado de medio ciclo de la señal eléctrica. Los valores RMS de cada canal de voltaje se analizan para determinar la categoría y tipo de disturbio, si es el caso de que ocurra alguno de estos.

En la implementación del cruce por cero se elige un canal de voltaje como base en la comparación del dato y el signo, marcando un rango de histéresis ±0.05 (valor normalizado). El valor inicial de la “bandera actual” es falso y permanece con este valor si el dato es positivo y mayor que el rango de histéresis; solo cambia a falso si el signo del dato es negativo y menor que el rango de histéresis. Con el valor actualizado de la” bandera actual” se hace la comparación con el valor de la “bandera anterior”, el cual tiene un valor falso inicial. Si los valores entre estas dos banderas son diferentes nos indica que ocurrió un cruce por cero y la “bandera anterior” toma el valor de la “bandera actual” debido a que los valores de ambas banderas inicialmente son iguales. El primer valor RMS no se considera ya que habría un error y posiblemente no sea el valor de medio ciclo completo de la señal eléctrica.

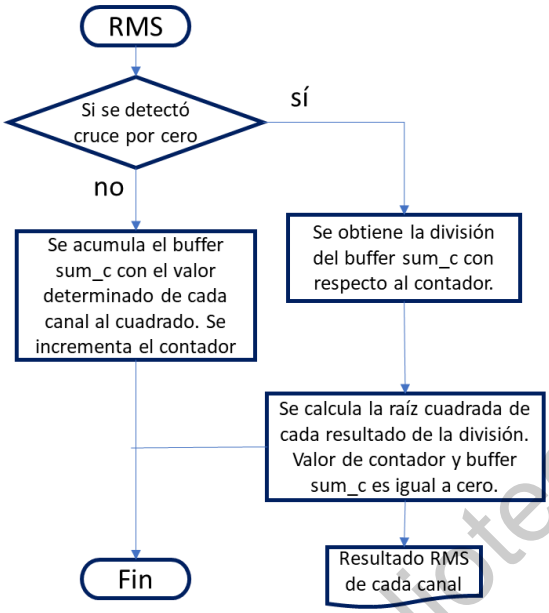


Figura III.8. Diagrama de flujo RMS.

Con el valor de cruce por cero para el RMS, como se observa en la Figura III.8, si no se detecta el cruce por cero, se acumula el producto dato por dato con el valor del buffer "sum_c", donde este buffer tendrá como valores iniciales cero y deberá tener de tamaño más del doble de rango de los datos obtenidos por el sistema de adquisición para evitar el desbordamiento de bits por la acumulación de datos. También se hace el incremento de un "contador" para tener como referencia el número de datos se han acumulado en el buffer, de igual manera con valor inicial de cero. Si se detecta un cruce por cero se realiza la división del "sum_c" como numerador con el "contador" de divisor, con el valor que se tiene de la división, se calcula la raíz cuadrada y así tener el valor RMS de todos los canales. Una vez calculado el RMS, tanto el "contador" y el buffer "sum_c" su valor será igual a cero, para obtener una nueva medición de medio ciclo.

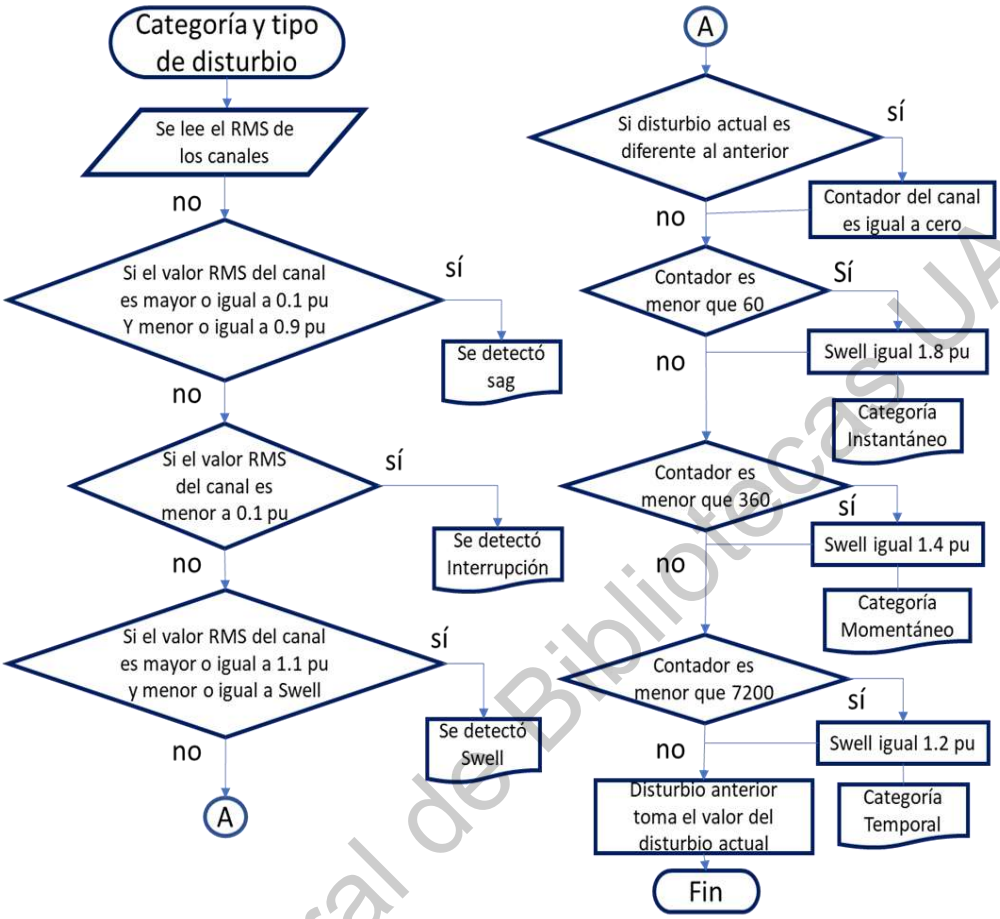


Figura III.9. Diagrama de flujo categoría y tipo de disturbio.

El diagrama de flujo de la Figura III.9 muestra los valores utilizados en la comparación seleccionados del Cuadro II.2 para definir la categoría y el tipo de disturbio por medio del valor eficaz. En la parte de disturbios, se hace la comparación con el valor RMS, por lo tanto, es un sag si el valor es mayor o igual a 0.1 pu y menor o igual a 0.9, solo será una interrupción si el valor es menor a 0.1 pu y si el evento (disturbio) tiene una duración mayor o igual a 60 (30 ciclos). La duración del evento es llevada por un contador que se mostrará más adelante. El disturbio swell tiene una particular diferente con respecto a los dos anteriores, es debido a que una de las comparaciones es modificada con respecto a la duración de este evento, si el valor RMS es mayor o igual a 1.1 pu y menor o igual a “Swell”, donde

el valor es de 1.8 pu cuando el evento sea menor o igual a 60, 1.4 pu si es menor o igual a 360 (3 segundos) y 1.2 pu si el contador del evento es menor a 7200 (1 minuto).

A continuación, con la información que vimos en la parte anterior de los disturbios, se verifica si ocurrió un cambio de evento por medio de la comparación "disturbio actual" con el "disturbio anterior", si es diferente, el contador del evento regresa a cero. El contador siempre será incrementado, evitando de esa manera divisiones entre cero. La comparación con el contador es la siguiente:

- Si el contador es menor o igual a 60. La categoría en la que se encuentra el disturbio es instantánea sin considerar el disturbio de interrupción ya que este no interfiere en la categoría, y el valor "swell" será de 1.8 pu.
- Si el contador es menor o igual a 360. La categoría en la que se encuentra el disturbio es momentánea y el valor "swell" será de 1.4 pu.
- Si el contador es menor o igual a 7200. La categoría en la que se encuentra el disturbio es temporal y el valor "swell" será de 1.2 pu.

Al finalizar la comparación de los contadores, se actualiza el valor del disturbio del actual con el anterior. El valor swell cambia respecto a la categoría en la que se encuentra mostrada en el Cuadro II.2.

3.2.2. FPGA

En esta sección se exponen los módulos descritos en la implementación del FPGA, su funcionalidad, la cantidad de datos que manejan, la descripción de las máquinas de estados, las entradas y salidas de señales digitales y la secuencia de datos seleccionados para su descripción. Se menciona como bloques a los módulos encargados de realizar los algoritmos de calidad de la energía, como se vio en los diagramas de flujo en la sección anterior.

3.2.2.1. Módulo principal FPGA

El módulo consiste en tres partes esenciales como se muestra en la Figura III.10, la primera de ellas es la comunicación del sistema de adquisición de datos con los dispositivos en software (ARM) y hardware (FPGA), donde la transferencia de datos en el ARM es en forma paralela con un tamaño de 16 bits y la utilización de un bit de interrupción, el cual tiene como tarea principal indicar la actualización en los datos de los canales capturados por el sistema de adquisición, por lo tanto, el dispositivo FPGA es una herramienta que nos permite establecer la comunicación del sistema de adquisición con el ARM. La segunda parte, el dispositivo FPGA será el único encargado en la configuración del sistema de adquisición manipulando las ganancias, la frecuencia de muestreo (si es posible modificarse) y el *offset*. Esta última configuración se verifica mediante pruebas con el sistema de adquisición para eliminar errores en la medición.

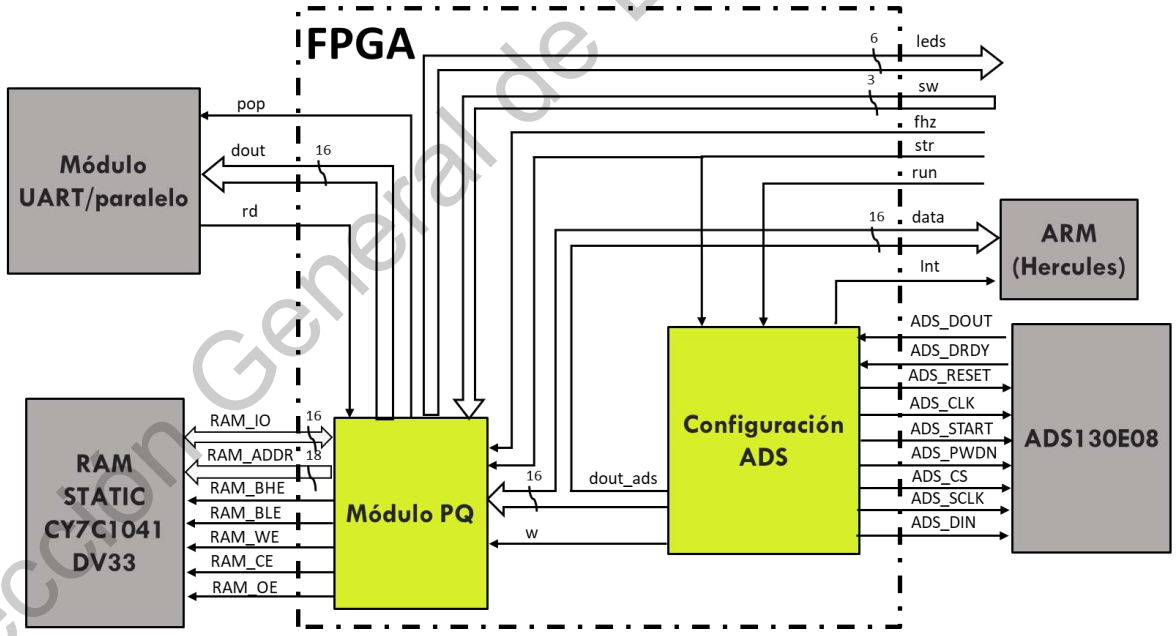


Figura III.10. Módulo principal FPGA.

Por último, en el FPGA se integra un submódulo llamado ‘módulo PQ’ encargado de hacer las mediciones de calidad de energía establecidas por la especificación G0000-48,

además se hace uso de una memoria RAM estática (SRAM), debido a que la capacidad de memoria del FPGA no cuenta con los recursos suficientes en el almacenamiento de información necesaria para realizar todos los cálculos establecidos en los algoritmos de calidad de la energía. El módulo PQ cuenta con una comunicación para los datos almacenados de los resultados de cada algoritmo de calidad de la energía, que se obtienen en paralelo o en serie si se requiere una comunicación con el módulo UART. La información es actualizada de manera continua y es notificada con la habilitación de una bandera o bit de salida, también cuenta con un bit de entrada que permite la lectura a la información almacenada en la memoria interna del FPGA. El orden y el tamaño de la memoria lo indica el módulo PQ.

La memoria SRAM cuenta con almacenamiento suficiente de dos ventanas para cada canal y tendrá un tamaño de la palabra de 16 bits, facilitando así el almacenamiento directo de los datos adquiridos del convertidor. El control con la memoria SRAM se da a través de las señales de salida digital RAM_OE, RAM_CE, RAM_WE, RAM_BLE, RAM_BHE y RAM_ADDR, las cuales operan la habilitación de la memoria, dirección y selección del modo de lectura o escritura. Por medio de la señal de salida-entrada RAM_IO es posible almacenar o leer la información en la memoria SRAM.

En la captura de los canales de voltaje y corriente, como se ha mencionado antes se emplea un convertidor de analógico-digital que es controlado y configurado mediante el submódulo CONFI_ADS, a través de señales de salida digital: ADS_RESET, ADS_CLK, ADS_START, ADS_PWDN, ADS_CS, ADS_SCLK y ADS_DIN, encargadas de configurar la operación del convertidor. También, a través de señales de entrada digital: ADS_DOUT y ADS_DRDY, que opera la señal discretizada, la señal de lectura y de sincronización respectivamente.

Cuando se obtiene una muestra completa *ads_dout* se dirige a la entrada PQ y a señales de salida digital DATA_ARM de un tamaño de 16 pines. La sincronización es habilitada por dos banderas como es *w* con una duración de un ciclo de reloj para el submódulo PQ e INT con una duración de cinco veces el ciclo de reloj para el vector de interrupción del ARM. La secuencia que se maneja en los datos de los canales es la siguiente:

los primeros tres datos son los canales de la corriente (Ia, Ib y Ic) y los tres siguientes son del voltaje (Va, Vb y Vc), es de suma importancia conocer la secuencia debido a que todos los módulos y bloques son diseñados con ese orden de datos.

Las señales de entrada y salida faltantes del módulo principal, cumple con las siguientes funciones:

- RUN. Inicializa la comunicación y carga la configuración en el convertidor.
- STR. Empieza la captura de los datos obtenidos de los canales de manera continua e inicializa el submódulo PQ.
- FHZ. Selecciona la frecuencia de la red que se está analizando (60 o 50 Hz).
- SW. Selecciona la fase de voltaje que se quiere observar de las variaciones de corta duración en los LEDS.
- RD. Habilita la lectura de los resultados almacenados en el submódulo.
- LEDS. Muestra la categoría y el tipo de disturbio detectado en la fase de voltaje.
- POP. Bandera que indica el momento en que existe por lo menos un dato almacenado en la memoria del submódulo.
- DOUT. Datos de salida del submódulo.

3.2.2.2. Submódulo PQ

Las operaciones de calidad de la energía basado con las especificaciones G0000-48 propuestas en la metodología, se encuentran diseñadas o descritas en la arquitectura digital PQ, la cual contiene los siguientes bloques:

- RMS. El bloque es encargado de obtener las variaciones de corta duración para los canales de voltaje y resultados RMS de medio ciclo de una señal eléctrica en todos los canales; para las fases de voltaje y las de corriente. Las señales que componen al bloque: *str_rms* habilita el funcionamiento del bloque, *w* comparte la señal del submódulo, ya que los valores procesados en el bloque son rápidos y no es necesario algún retardo, *dsqx* representación visual del tipo y categoría del disturbio en los 3

canales de voltaje, *yx* se representa como un conjunto de las señales RMS de los seis canales debido al tamaño para representarlas, y *rdy_rms* indica cuando el bloque finalizo el cálculo.

- FIFO_ext. Es el encargado de escribir en la memoria externa para almacenar los datos de los canales del convertido directamente (buffer de datos), de tal manera la información almacenada pueda ser adquirida por el bloque de distorsión armónica total. El bloque lo componen las señales digitales de la memoria SRAM mencionadas en el módulo principal, además cuenta con señales internas: *nr* es el encargado de ubicar el buffer de almacenamiento de un canal individual, *r_fifo* habilita el modo de lectura y *w_fifo* el modo de escritura de la memoria SRAM externa, *data* son los datos de escritura, *din_thd* los datos de lectura de la memoria y *fl* que indica si en el bloque se encuentra datos disponibles para leer. El almacenamiento y lectura de los datos en la memoria es de manera circular, por lo tanto, los datos son secuenciales permitiendo reemplazar el dato nuevo con el más antiguo con un tamaño de 16384 datos por canal.
- THD. La tarea principal del bloque es calcular la distorsión armónica total en cada canal *dout*_thd, y proporcionar la parte real (*fre*) e imaginaria (*fim*) que componen la frecuencia fundamental de la fase, que se está analizando en el bloque. El tamaño de buffer de almacenamiento de datos se considera por el tamaño de la ventana para el THD y adicionalmente el retardo ocasionado por el procesamiento de los mismos debido a la carga computacional. Las señales digitales: *str_thd* da la inicialización al bloque, *fhz* define la frecuencia de la red eléctrica que se está procesando, *w_thd* carga el dato al bloque, y el conjunto de señales *load/busy* sincroniza e indica cuando el bloque está listo para un dato nuevo.
- DSQ. Las señales de salida de información de la frecuencia fundamental del módulo THD son requeridas en el procesamiento del módulo de desbalance o desequilibrio por fases (DSQ), el cual recibirá los valores de la parte real (*fre*) e imaginaria (*fim*) de la salida del módulo THD de los canales de voltaje. De esa manera es posible calcular el desequilibrio que existe entre las tres fases de voltaje. Las señales digitales del bloque: *str_dsq* inicializa el bloque, *w_dsq* indica al bloque que ya puede tomar los valores del bloque THD y *rdy_dsq* avisa cuando la operación ha sido completada.

- FIFO_int. El bloque diseñado es una memoria interna del FPGA encargada de almacenar los resultados de los bloques, para llevar de esta manera una bitácora de eventos. Las señales digitales que componen el bloque son: *clr* encargada de limpiar la memoria, *wr* y *rd* habilita la escritura y lectura de la memoria, *pop* indicador de que existe por lo menos un dato sin leer de la memoria, *x* datos de entrada y *dout* datos de salida de la memoria.

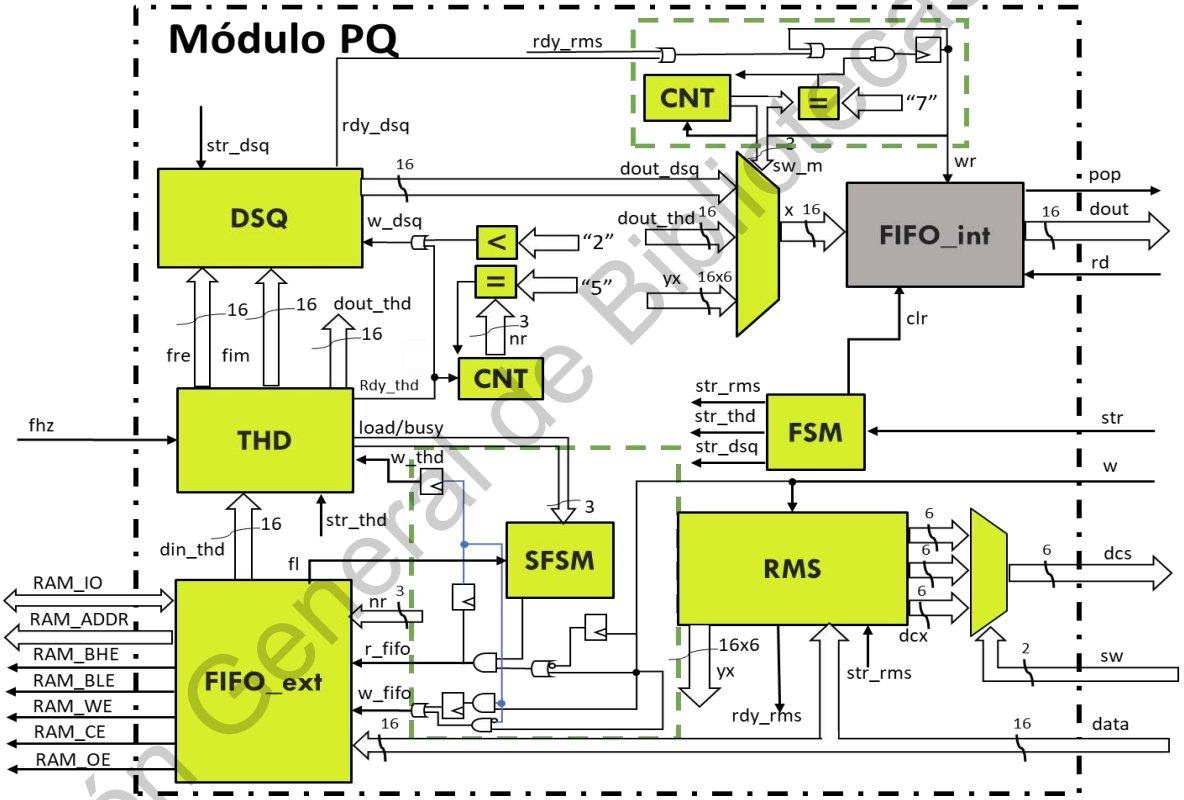


Figura III.11. Submódulo PQ.

El submódulo como se muestra en la Figura III.11, cuenta con dos máquinas de estados independientes, donde la máquina de estados *FSM* es la encargada de inicializar los bloques de calidad de la energía y habilitar la memoria interna en el FPGA. Por otro lado, la máquina de estados *SFSM* se encarga de sincronizar la carga de datos en el bloque THD esperando la habilitación de las señales *load/busy*, debido a que puede ocurrir una colisión en la lectura y

escritura en la SRAM por los tiempos de retardo en el procesamiento en el bloque THD y el dato actualizado por el convertidor. También se hizo uso de compuertas lógicas y registros, para dar mayor prioridad a la señal digital *w_fifo* (escritura de datos en la memoria) y una vez acabada la tarea junto a la señal digital *fl* se toma lectura de la memoria con la señal digital *r_fifo.* El tiempo mínimo de actualización de los datos es de 4 us, siendo el rango de retardo provocado por la comunicación SPI con el convertidor ADS. La señal *w_thd* es un retraso de la señal *r_fifo*, que es considerado por el tiempo necesario en la lectura o escritura en la SRAM. En el diseño del submódulo se utiliza un cristal de reloj de 48 MHz; es suficiente el aplazamiento de dos ciclos de reloj que permite el tiempo necesario para el correcto almacenamiento y lectura en la memoria SRAM. En caso que se requiera un cristal mayor al utilizado en este proyecto se recomienda agregar un registro adicional, si es menor no es requerido eliminar o agregar registros.

Se utiliza un contador (CNT) para seleccionar el bloque de lectura en la memoria SRAM con la señal digital *nr* teniéndose dos comparaciones: cuando el contador sea igual a cinco, inicializa en cero, y si el contador es mayor que dos indica que los resultados al término del bloque THD son los valores de los canales de voltaje. El contador que maneja la señal *sw_m* manipula un selector para almacenar los resultados en la memoria interna de FPGA, como son: *yx* que son las salidas RMS de los 6 canales, *dout_thd* y *dout_dsq*. El contador es habilitado tanto con la señal *rdy_rms* o *rdy_dsq,* ya que son los bloques que tienen menor tiempo de procesamiento.

La comparación que se utiliza en el contador es para deshabilitar la escritura, por medio de compuertas lógicas y un registro que mantiene la señal digital *wr* hasta que se termine de escribir todos los resultados teniendo así la bitácora de eventos. Por último, la señal de entrada digital *sw* por medio del multiplexor selecciona cual disturbio y categoría del canal se quiere visualizar.

3.2.2.3. Bloque RMS (variaciones de corta duración).

El bloque RMS o variaciones de corta duración como se muestra en la Figura III.12, se encarga de calcular el RMS con una longitud de medio ciclo de una señal eléctrica, ya que es el valor más pequeño de comparación que establece la especificación G0000-48.

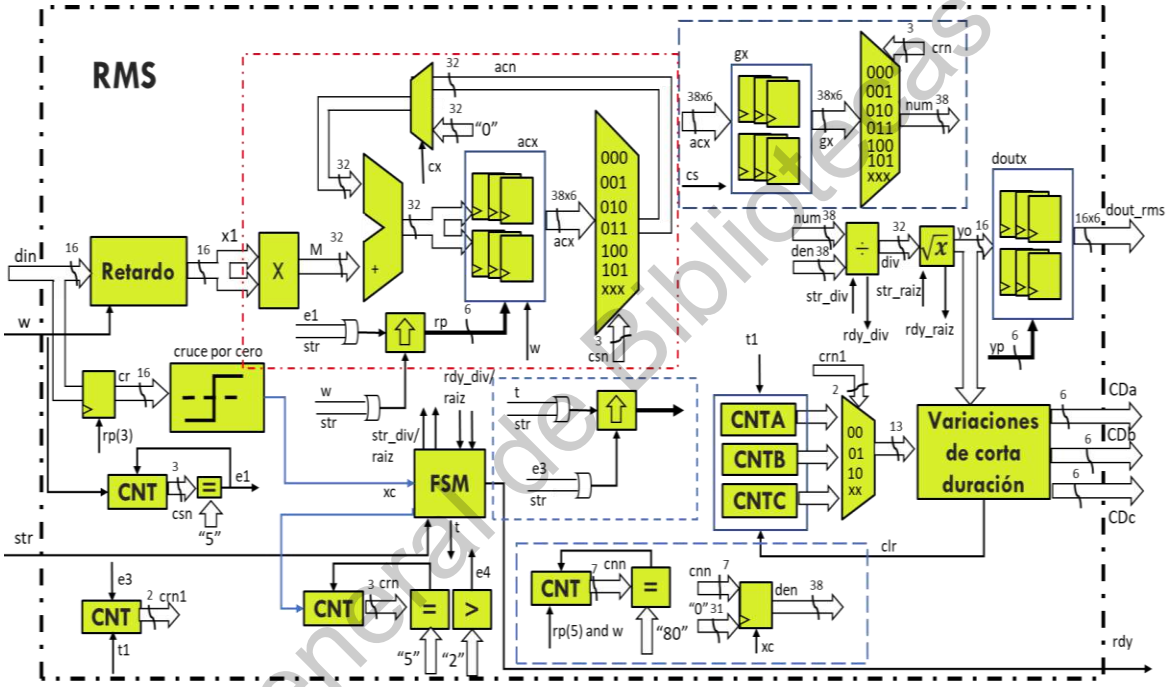


Figura III.12. Bloque RMS.

El cálculo se realiza con el conjunto de diferentes sub bloques contenidos dentro del bloque RMS. La señal digital de entrada *din* y *w* del bloque pasa por el sub bloque de Retardo, el cual es un conjunto de seis registros en serie que son habilitados con la señal *w*, este aplazamiento es recomendado debido a la ubicación de la fase Va del sistema de adquisición y el detector de cruce por cero, evitando pérdida de información.

De la salida de la señal *x1* del sub bloque de retardo se realiza la acumulación individual del producto de los canales utilizando un multiplicador y un sumador. El multiplexor controlado por la señal *csn* selecciona del conjunto de registros llamados *acx*

correspondiente al canal que se tienen en *x1*, cuando se detecta cruce por cero mediante otro multiplexor controlado por la señal *cx* borra la acumulación de cada registro, pero antes es almacenada en el registro correspondiente del conjunto gx habilitado con la señal *cs*. La máquina de estados es la encargada de sincronizar las operaciones de división y raíz cuadrada; el numerador para la división es seleccionado por el multiplexor controlado por la señal *crn*, y para el denominador es mediante el resultado del contador, el incremento de la cuenta está dado por dos señales *rp(5)* y *w* que indica la cantidad de datos que pasaron en cada canal. Como requerimiento para realizar la división, la señal de entrada en el denominador debe de ser el mismo tamaño que el numerador, por lo tanto, a la señal *cnn* del contador se concatena una señal de ceros para tener el tamaño faltante en los bits menos significativos.

La raíz cuadrada es aplicada a la señal *div*, que es el resultado de la división. Al finalizar la raíz se obtiene el resultado RMS del canal que es mantenido por el registro correspondiente del conjunto doutx. Si el valor RMS es de algún canal de voltaje, se realiza la comparación en el sub bloque variaciones de corta duración, que indica si ocurrió algún disturbio, y mediante la comparación de los contadores A, B o C, que son multiplexados con la señal *crn1*, determina el tipo de categoría en que se encuentra. Si el disturbio es diferente al anterior con su respectivo canal, el contador escogido por el multiplexor es borrado y reiniciado en cero.

El sub bloque de cruce por cero, realiza la operación de comparación de histéresis del dato que tiene de entrada, la comparación se realiza por dos límites del ±5% considerando la normalización 2.14 ya mencionado anteriormente. Cuando el dato es mayor al límite superior la bandera actual toma el valor de cero, si el dato es menor que el límite inferior la bandera actual tendrá de valor uno. Se utiliza un registro para mantener la bandera anterior y actualizada con la actual, mediante una compuerta lógica XOR, se obtiene el cambio de la bandera actual con la anterior indicando de esta manera que ocurrió un cruce por cero.

La máquina de estados es la encargada de sincronizar todos los sub bloques, cuando la señal *str* es habilitada el bloque empieza a trabajar y permanece en espera de un cruce por cero (*xc*) del canal de voltaje base, cuando esto sucede activa la señal de *str_div* comenzando

la división y está a la espera de la señal *rdy_div* que indica que el resultado es completo, lo mismo sucede para la raíz cuadrada que inicia con la señal *str_raiz* y termina con la señal *rdy_raiz*, esto se repite la cantidad de canales a analizar. Una vez terminado el RMS de todos los canales, se habilita la señal *rdy* y la máquina de estados vuelve a la espera de otro cruce por cero.

3.2.2.4. Bloque THD

El bloque de la Figura III.13, es encargado de medir la distorsión armónica total de los canales de voltaje y corriente, además proporciona la amplitud y la fase en formato real e imaginaria de los canales para el bloque de desequilibrio.

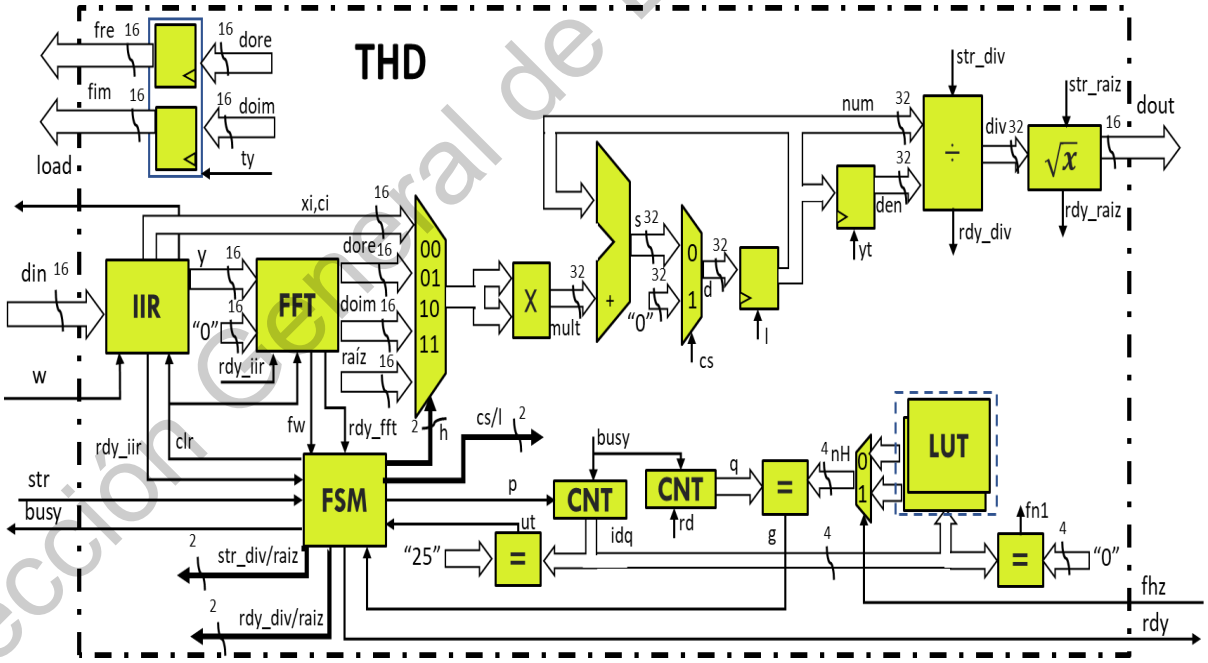


Figura III.13. Bloque THD.

El bloque está constituido por dos LUTs, los cuales indican los índices donde se encuentra la frecuencia fundamental y todos los armónicos hasta el 25vo. Las LUTs son

específicas para la red de 60 o 50 Hz y es electa con un multiplexor por medio de la señal digital *fhz*. La señal *nH* que contiene el indicie de la LUT seleccionada, es comparado con el contador con la señal *q* que indica el índice leído de la FFT, cuando la señal *q* y *nH* son iguales con la señal *g* en la máquina de estados para sincronizar el incremento del contador *idq* con la señal *p*, *idq* es encargado de la dirección de los datos de la LUT y la cantidad de armónicos encontrados, mediante las comparaciones *idq indica* que se ha completado hasta el 25vo armónico para realizar posteriormente las operaciones de división y raíz cuadrada, de la misma manera como se mostró en el bloque RMS, y si se tiene la frecuencia fundamental; es decir, cuando *idq* sea igual a cero.

En la frecuencia fundamental por medio de las señales *fn1* y *p* con una compuerta lógica AND, se genera la señal *ty* encargada de mantener los valores de las señales *dore* y *doim* en los registros de salida para la frecuencia fundamenta real (*fre*) e imaginaria (*fim*) del canal, además en el registro *den* mantiene el producto y la suma de la componente real e imaginaria de la fundamental como resultado de la señal *num*. Con la señal *ut* le indica a la máquina de estados que el producto y la acumulación de los armónicos se encuentra concluida para realizar posteriormente la división, con la señal *den* y el resultado se aplica la raíz cuadrada dando como resultado la distorsión armónica total en la señal *dout*.

El sub bloque IIR, se diseñó con las características planteadas en trabajos anteriores mostradas en los antecedentes. El sub bloque es un filtro digital tipo IIR Butterworth pasa-bajas de sexto orden, los coeficientes de los filtros se obtuvieron mediante la función de MATLAB *butter*, los cuales están determinados por la frecuencia de muestreo del convertidor y el doble de la frecuencia de corte. La frecuencia de corte que se utiliza es la recomendada para el 40vo armónico por mayor estabilidad (Ferrigno & Landi, 2007), aunque la especificación indica hasta el 25vo armónico. Este diseño no afectará las mediciones. El dato a filtrar *din* es sincronizado con la señal de entrada digital *w*, que le ordena al bloque el comienzo del filtrado. Terminado el filtrado, el resultado se encuentra en la señal *y* notificándose por la señal *rdy_iir* que a la vez funciona como indicador de escritura en la FFT.

El diseño del sub bloque FFT se ha utilizado en trabajos anteriores, siendo un proceso primordial en la metodología de este trabajo. Las características que tiene la FFT es tipo Radix-2 con una ventana de 8192 datos cumpliendo de esta manera las incertidumbres de THD y desbalance en fases en el Cuadro III.1, especificación del documento de CFE. Las señales digitales del sub bloque como es *fw* indica cuando la ventana de datos está completa para realizar el funcionamiento de la FFT y *rdy_fft* avisa que se ha completado el procesamiento, la señal *dore* contiene la información real del índice leído en la FFT y *doim* la parte imaginaria.

En el diseño del bloque THD se considera el mínimo uso de los recursos del FPGA posibles, por lo tanto, se compartió un multiplicador del FPGA en las operaciones de raíz cuadrada por el método de aproximaciones sucesivas (Successive Approximation Register, SAR), la multiplicación del acumulador y del filtro digital IIR, que es seleccionado por el multiplexor y controlado por la señal *h*, que es manipulada por la máquina de estados.

A continuación, se muestra a detalle el proceso de sincronización que desempeña la máquina de estados (FSM) en el bloque THD. Primeramente, el bloque limpia los registros con la señal *clr* de los sub bloque IIR y FFT hasta la habilitación de la señal de entrada *str*. Después de que la información de la señal *din* es filtrada, se almacena en la memoria de la FFT hasta completar la ventana de 8192 datos, que se indica con la señal *fw* hacia la FSM que habilita la señal *str_fft*, comenzando el procesamiento y en la espera de la señal *rdy_fft*, que notifica la terminación de la FFT; además, se mantiene el multiplexor en “00” con la señal *h* para la multiplicación requerida en el filtro digital IIR.

Completada la transformación del dominio del tiempo a frecuencia, se realiza la acumulación de la frecuencia fundamental en primera estancia, seguida de la acumulación de los armónicos hasta el 25vo. Al efectuarse la acumulación, la señal *h* cambia a “01” y “10” para obtener el producto de la multiplicación de sí misma en la parte real (*dore*) y en la parte imaginaria (*doim*), respectivamente, de las señales de salida de la FFT. La FSM es encargada de controlar la acumulación con la señal *l* y la limpia con el multiplexor con la señal *cs*.

Terminada la acumulación de todos los armónicos y notificada con la señal *ut*, que viene de la comparación con el contador, la FSM se encarga de sincronizar el comienzo de la división y la raíz cuadrada con las señales *str_div* y *srt_raiz*, respectivamente, cambiando la señal *h* a "11" para el proceso del SAR de la raíz cuadrada, esperando la finalización de las operaciones con las señales *rdy_div* y *rdy_raiz*. Con las operaciones terminadas, el resultado es dirigido al registro de la señal de salida *dout*, borrando los registros de los sub bloques IIR y FFT en la espera de otra ventana.

3.2.2.5. Bloque DSQ (desbalance)

El desbalance por fase es calculado por el submódulo de la Figura III.14, el diseño está basado en la transformada de Fortescue, pero descrita como se muestra en la ecuación (15) debido a que los valores de amplitud y fase del canal de voltaje son entregados en forma de número complejo, recibiendo las señales de entrada *fre* parte real y *fim* parte imaginaria y su procesamiento por canal es mediante la señal digital de entrada *w*.

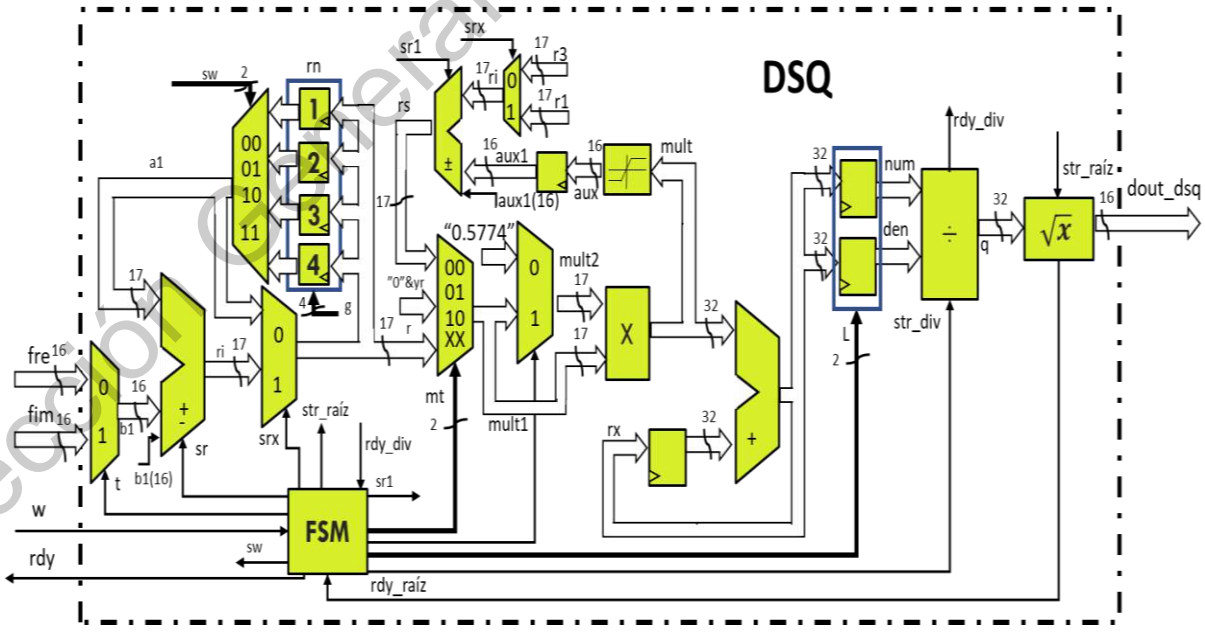


Figura III.14. Bloque DSQ.

La secuencia que sigue el bloque está definida por los canales de voltaje manejados por el convertidor, en este caso el primer valor corresponde Va, el segundo Vb y el tercero Vc. Es requerido el orden de entrada de estos canales, en el cumplimiento de la ecuación (15), de no ser así el cálculo resultante en la señal *dout_dsq* será erróneo y posiblemente se mantenga en saturación. El DSQ es constituido solo por tres sub bloques: limitador que mantiene el formato 2.14, división y raíz cuadrada. La secuencia de las operaciones es realizada con la máquina de estados FSM, con ello se disminuye el consumo de recursos y multiplicadores del FPGA compartiendo la operación de la raíz cuadrada y del acumulador.

A continuación, se hace la explicación del proceso llevado en el bloque con la FSM. Con la señal *w*, FSM suma la señal *fre* con el registro *r1* y la resta con *r2*, la señal *fim* se suma con el registro *r3* y se resta con el registro *r4*, todo esto es con los valores de Va. Al recibir *w* con la información de Vb, se hace la resta de *fre* con los registros *r1* y *r2*, y también la resta de *fim* con los registros *r3* y *r4*. Con los valores Vc y notificado con *w*, se realiza dos veces la suma de *fre* con lo almacenado en el registro *r2*, después la multiplicación con la constante "0.5774" (valor en decimal de $\frac{1}{\sqrt{3}}$), el resultado es restado con el registro *r1* utilizando la señal *srx* en el multiplexor y cambiando la señal *mt*, se almacena el producto del resultado en el registro *num*, *fim* es restado con el registro *r4* y el producto del resultado queda en el registro *den*. De igual manera se suma dos veces. Obtenida la información de todos los canales de voltaje, se hacen las operaciones faltantes, el registro *r4* es acumulado con el producto del *num*, el registro *r2* es multiplicado y acumulado en el registro *den*. Posterior a ello, se realiza las operaciones división y con el resultado se calcula la raíz cuadrada con las señales correspondientes de cada sub bloque, obteniendo así el desbalance entre las fases.

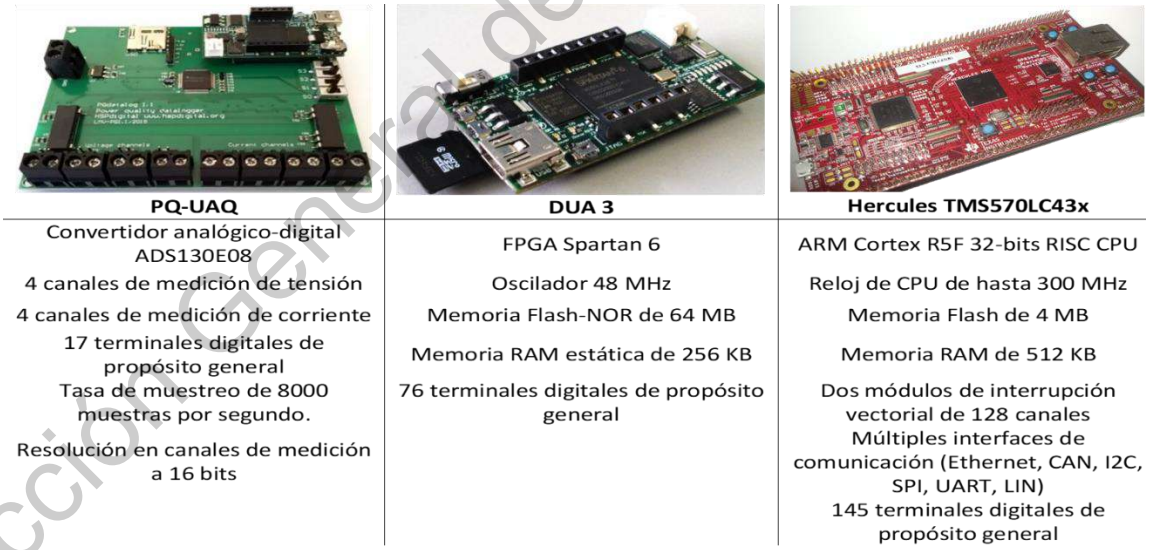
3.3. Puesta del experimento

En este trabajo, se propuso una sección adicional donde se muestran los dispositivos, sistemas y herramientas utilizadas en el proyecto. Además, se presentan las características

particulares de cada dispositivo como son FPGA y ARM usados en la implementación de los algoritmos de calidad de la energía, que sirve como base para determinar el consumo de recursos junto con los resultados cuál de ellos se obtiene un mejor costo-beneficio.

3.3.1. Dispositivos y tarjeta de adquisición

En la prueba de los algoritmos desarrollados en dos diferentes tecnologías para ponerlos en comparación se implementaron en las tarjetas que se observan en la Figura III.15, siendo el sistema de adquisición de datos PQ-UAQ que es la base de referencia de los algoritmos en el procesamiento de datos debido al orden, resolución y frecuencia de muestreo de las señales eléctricas. El prototipo DUA 3 que es una tarjeta embebidos que contiene como principal el dispositivo FPGA y la tarjeta de desarrollo *Hercules* con la tecnología ARM.



PQ-UAQ	DUA 3	Hercules TMS570LC43x
Convertidor analógico-digital ADS130E08	FPGA Spartan 6	ARM Cortex R5F 32-bits RISC CPU
4 canales de medición de tensión	Oscilador 48 MHz	Reloj de CPU de hasta 300 MHz
4 canales de medición de corriente	Memoria Flash-NOR de 64 MB	Memoria Flash de 4 MB
17 terminales digitales de propósito general	Memoria RAM estática de 256 KB	Memoria RAM de 512 KB
Tasa de muestreo de 8000 muestras por segundo.	76 terminales digitales de propósito general	Dos módulos de interrupción vectorial de 128 canales
Resolución en canales de medición a 16 bits		Múltiples interfaces de comunicación (Ethernet, CAN, I2C, SPI, UART, LIN)
		145 terminales digitales de propósito general

Figura III.15. Tarjetas utilizadas para experimentación.

El PQ-UAQ es un sistema de adquisición de señales de voltaje y corriente para un sistema de red trifásica que puede usarse también en una red monofásica, el integrado

principal de la tarjeta es el ADS13OE08 que es el convertidor de analógico-digital tipo delta-sigma que es constituido por integrador, cuantificador y convertidor de retroalimentación, además cuenta con un filtro *sinc* de tercer orden pasa-bajas y la forma de comunicación de la información es por medio del protocolo SPI (*Serial Peripheral Interface*). Cuenta con 8 canales de medición, con tasa de muestreo de 8000 muestras por segundo con una resolución de 16 bits, el diseño de la tarjeta está basado a los conectores de la DUA 3.

La DUA 3 es un prototipo del departamento de Ingeniería de la Universidad Autónoma de Querétaro, Campus San Juan del Río al igual que el PQ-UAQ. La tarjeta incluye como integrado principal el XC6SLX16 que es un FPGA de la familia *Spartan 6*, también contiene otros dispositivos como son: sensores (acelerómetros, giroscopio, magnetómetro, temperatura y presión barométrica), RAM estática de 256 K y FLASH-NOR 64 Mb. La tarjeta tiene disponibles 76 entradas o salidas de propósito general, oscilador para el FPGA de 48 MHz, 8 indicadores luminosos y socket para microSD.

El *Hercules* es una tarjeta de desarrollo de la familia de Texas Instruments con el integrado TMS570LC4357 que es un ARM con un CPU de 32 bits, quiere decir que el tamaño de bus de instrucciones es de esa cantidad y la unidad de punto flotante. Las características principales que sobresalen de este dispositivo son tres: la frecuencia de operación que es de 300 MHz, la memoria RAM de 512 Kb y FLASH de 4 Mb. Otras características son los protocolos de comunicación, vectores de interrupción y pines digitales de propósito general. En el proyecto se tiene conocido la cantidad de memoria que ocupa en el almacenamiento de la información de la señal muestreada, pero se desconoce la cantidad de memoria necesaria para las instrucciones en el procesamiento de estos datos, por ello al seleccionar el dispositivo se considera una cantidad grande en memoria FLASH.

Otras características que cuenta la tarjeta de desarrollo *Hercules* son los tipos de protocolos de comunicación que maneja, entre los más conocidos se encuentran ethernet, CAN (*Controller Area Network*), I2C (*Inter-Integrated Circuit*), SPI y UART (*Universal Asynchronous Receiver-Transmitter*), y pocos usados como lo es el protocolo LIN (*Local Interconnect Network*).

En los dispositivos queda mencionar que son de propósito industrial, varias de sus aplicaciones se encuentran mencionadas en las hojas de datos del integrado, su forma de programación es similar a otras marcas y las características en recursos son genéricas encontradas en diferentes dispositivos comerciales.

3.3.2. Sensores

En la medición de corriente y voltaje en la fuente de energía que es suministrada por la CFE por un transformador trifásico de voltaje a pico de 120 VAC a 60 Hz de fase a neutro del departamento de Ingeniería de la Universidad Autónoma de Querétaro, campus San Juan del Río, que proporciona energía a salones, edificios y equipos de uso industrial encontrados en laboratorios será tomado por la tarjeta de adquisición PQ-UAQ de la siguiente manera: Se conectarán tres canales de voltaje tomando las fases por separado con referencia a neutro de tal forma que las mediciones de tensión de calidad de la energía sean individuales. En la parte de corriente, se utilizaron los tres canales del PQ-UAQ que son diseñados para el uso de sensores de corriente.

Figura III.16. Sensor de corriente SCT013.

En la Figura III.16, se puede observar el sensor que se usará en la medición de corriente siendo el SCT013 un sensor no invasivo, esto quiere decir que no es necesario

detener o desconectar la carga para la medición de corriente. La característica de medición del sensor es por medio de un transformador de corriente, donde la corriente que pasa en el bobinado primario es transformada por el bobinado secundario, esta relación de corriente es de 100A/1V, quiere decir, si se obtiene una corriente en nuestra carga de 100A como resultado de salida nuestro sensor será de 1V, así el voltaje es proporcional a la corriente que pasa por la bobina.

IV. RESULTADOS Y DISCUSIÓN

En la validación de la metodología descrita en el apartado anterior y comprobar el funcionamiento, se realiza la programación de los diagramas de flujo de los algoritmos para obtener la dimensión de cantidad de memoria utilizada, en el procesamiento de las instrucciones y resultados esperados con los bancos de pruebas de señales sintéticas como reales. Por otro lado, para demostrar el funcionamiento de las arquitecturas digitales propuestas, comparar eficiencia y tiempo de ejecución se realizó la simulación de hardware de estas, y los resultados son comparados con los obtenidos por un computador personal (PC) mediante software de programación MATLAB. Finalmente, se realiza la implementación de las arquitecturas digitales en un sistema embebido con un FPGA y los diagramas de flujo en una tarjeta de desarrollo ARM interconectadas a la tarjeta de adquisición de datos PQ-UAQ, verificando su funcionalidad con los bancos de pruebas.

4.1. Banco de pruebas de señales sintéticas y reales

En la validación de los algoritmos de calidad de la energía, se realizaron tres tipos de pruebas. La primera prueba fue mediante señales sintéticas con características propuestas, que sirvieron para verificar que los algoritmos sean capaces de identificar los disturbios y analizar el margen de error debido a la tasa de muestreo. La segunda prueba se realiza bajo los intervalos que se muestran en el Cuadro III.1, con el cumplimiento de la especificación G0000-48 para medidores de energía con el margen de incertidumbre de Clase S, asegurando que los dos dispositivos sean capaces de cumplir con lo establecido en el documento. Como prueba final, con bancos de datos de señales reales muestreadas de una red trifásica, que fueron adquiridas de los bancos de prueba de la IEEE (IEEE, 2017), se valida el funcionamiento del dispositivo con los algoritmos y si el margen de error no aumenta al obtenido con las señales sintéticas.

4.1.1. Validación del algoritmo RMS

En esta sección, se muestran las pruebas realizadas en la validación de los algoritmos encargados de analizar la señal eléctrica, se obtuvieron con las pruebas márgenes de error en cada dispositivo, detección correcta de disturbios y el cumplimiento de las incertidumbres especificado en el documento de la CFE. Las señales sintéticas fueron generadas en MATLAB, la ventana de datos que se manejó en los bancos de prueba para cada disturbio fue de 1,048,576 (2^{20}) datos con sus respectivos porcentajes, esto es debido a que la cantidad de ciclos que indica el Cuadro III.1, en la parametrización del medidor es de 3,600 ciclos como prueba máxima, y la frecuencia de muestreo manejada es de 8000 muestras por segundo, cada 150 muestras representa un ciclo completo de una señal con frecuencia de 60 Hz, por lo tanto, se requieren 540000 muestras por cada disturbio en particular.

La memoria de los dispositivos es recomendable utilizarla en base 2, debido a ello se utilizó la cantidad de datos mencionado anteriormente. El tamaño de la palabra de los datos es de 16 bits donde se manejó la normalización 2.14, dos bits para la parte entera con signo y los 14 bits restantes para la parte fraccionaria.

En la categorización y detección del disturbio, para mostrarlo en cada dispositivo se manejó de la siguiente manera: se utilizó un número entero de dos dígitos, donde 10 es la categoría de instantáneo, 20 es momentáneo y 30 temporal. Así mismo para el disturbio 1 es interrupción, 2 se refiere al sag y por último 3 a un swell. Una vez teniendo la categoría y tipo de disturbio, se hace una suma y posteriormente ser enviada o visualizada.

En la Figura IV.1, se puede observar un fragmento de la señal utilizada para verificar la detección del disturbio de interrupción en los dispositivos. La señal a) se simuló como una señal sin perturbaciones a lo largo de la ventana, excepto 300 muestras que es igual a 4 semiciclos de una señal de 60 Hz, donde fue sustituida con un disturbio de 6.25%, que es por debajo del 10% dando como resultado una interrupción. La señal b) tiene una amplitud de 9% simulando un disturbio de interrupción sin cambio, fue usado para medir la incertidumbre de un rango de ±1 ciclo marcada en la especificación.

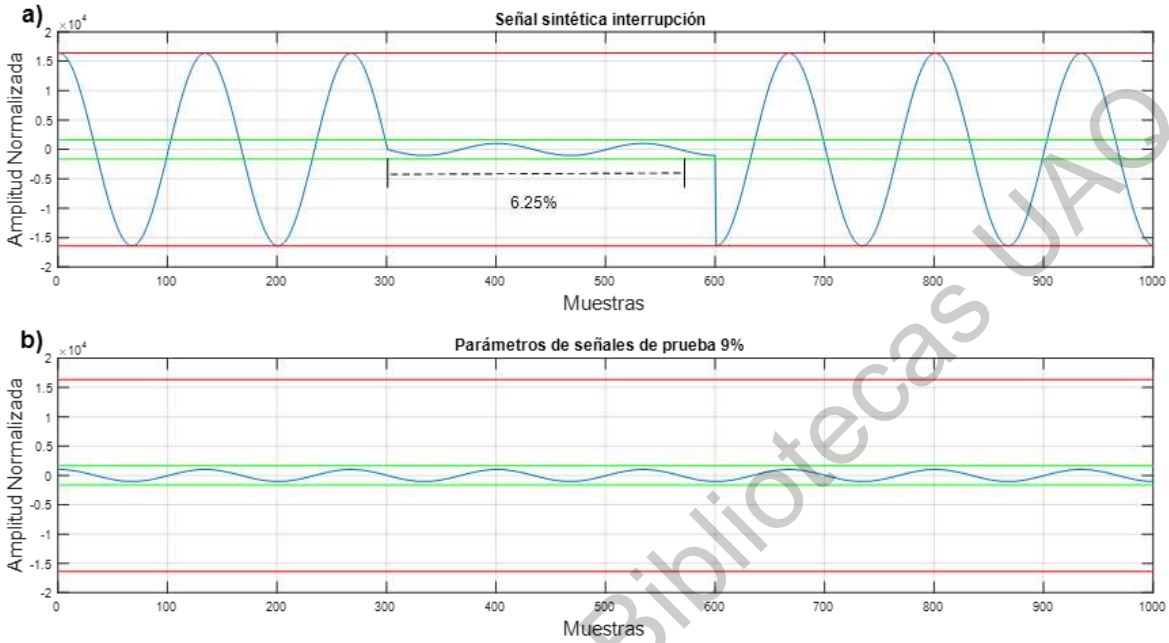


Figura IV.1. Señales sintéticas disturbio de interrupción.

Además, se puede observar líneas continuas que marcan los límites, de color rojo siendo el rango de 100% (amplitud positiva y negativa) de una señal sana y de color verde de 10% donde por debajo de este rango ya se considera una interrupción.

En las Figuras IV.2, se puede observar cómo el dispositivo ARM es capaz de identificar la señal sintética propuesta por la CFE para validación del equipo. En la señal el disturbio fue integrado como se explicó anteriormente en 3, 6 y 3600 ciclos, en el caso de interrupción no se pide la integración de 1800 ciclos, en las figuras mencionadas se puede observar en la interrupción de 3600 ciclos, como los dispositivos identifican las tres categorías del disturbio en la cantidad de ciclos manejado en el estándar de la IEEE (IEEE Std 1159, 2009).

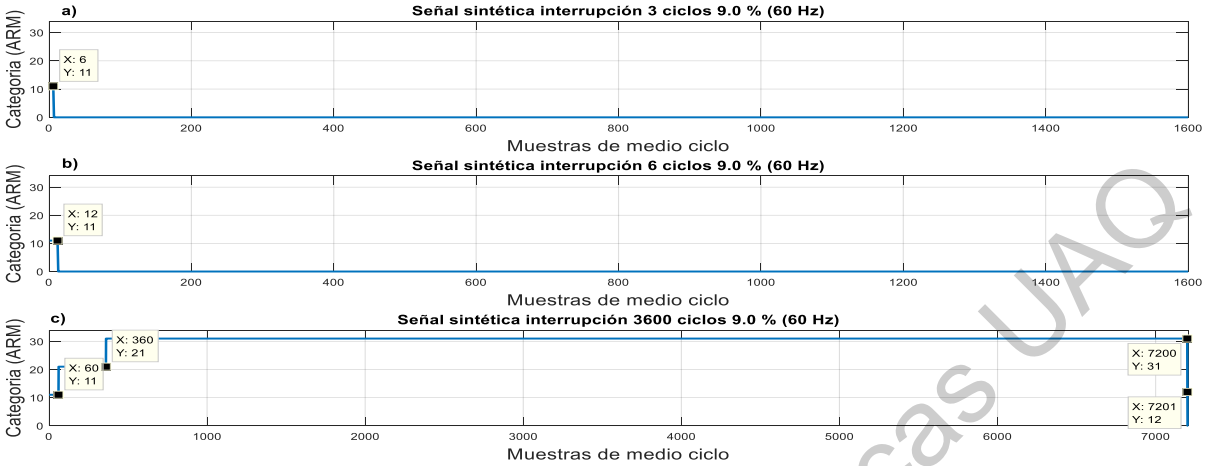


Figura IV.2. Categoría y disturbios obtenidos por el ARM.

En la Figura IV.3 se muestra la respuesta del FPGA con la señal de interrupción. Con referencia a la figuras y pruebas realizadas en los dispositivos ARM y FPGA, para las siguientes señales con disturbio sag y swell, se omite colocar en este trabajo la categoría y disturbio de cada dispositivo, debido a que la respuesta es igual en los dos, de aquí en adelante solo se manejará una gráfica para las dos tecnologías.

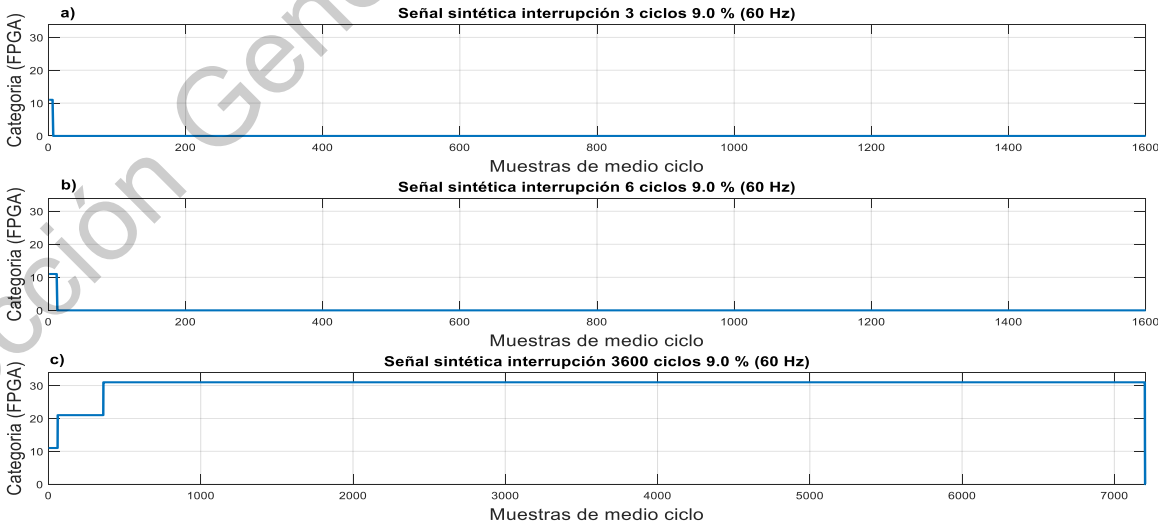


Figura IV.3. Categoría y disturbio obtenidos por el FPGA.

Los resultados obtenidos de los ciclos con la perturbación se muestran en el Cuadro IV.1, es el porcentaje de margen de error obtenido en los dispositivos con referencia al computador, la incertidumbre en los ciclos es el cambio que se tuvo en la comparación del evento, al no ocurrir un cambio con referente a la señal utilizada será cero y en el porcentaje promedio de *Vn*, se refiere que tan alejado es el resultado de cada dispositivo con el 9% que se utilizó en la señal sintética para simular una interrupción.

Cuadro IV.1. Porcentajes promedios de margen de error e incertidumbre.

9% interrupción (ciclos)	**% error promedio**		**Incertidumbre**			
	FPGA	ARM	Ciclos		% promedio Vn	
			FPGA	ARM	FPGA	ARM
3	$37.3177\ e^{-3}$	$13.792\ e^{-6}$	0	0	0.04312	0.0465
6	$37.3177\ e^{-3}$	$14.9721\ e^{-6}$	0	0	0.021	0.0232
3600	$37.3177\ e^{-3}$	$16.1466\ e^{-6}$	0	0	$37.3177\ e^{-3}$	$89.108\ e^{-3}$

Como se puede observar en el Cuadro IV.1, los dos dispositivos cumplieron con las incertidumbres indicadas en el Cuadro III.1, pero el FPGA a pesar de que su margen de error promedio sea más de mil veces al ARM, el porcentaje de *Vn* tiene menos desviación en milésimas, esto es debido a que la acumulación de la parte fraccionaria, como se hizo mención el FPGA las operaciones son manejadas en punto fijo siguiendo la normalización, en cambio el ARM tiene una unidad de punto flotante de 32 bits, por lo que al realizar el promedio se adquiere más error acumulado que el FPGA. En el caso de la señal de interrupción en el dispositivo ARM el margen de error fue incrementando en comparación con el FPGA donde este se mantuvo fijo, aunque este aumento no es notable por la escala en el resultado.

Se realizó la prueba con señales reales, donde se manejó bancos de pruebas en cada dispositivo al mismo tiempo, y se estableció la comunicación individual de ellos con el computador, en el momento que se identifica un disturbio de interrupción en medio ciclo de

la señal, se detuvo la transferencia de datos y se analizó en el computador para visualizar la parte de la señal donde se identificó y realizar la comparación con los resultados de los dispositivos. En la Figura IV.4 se muestra una parte de la señal donde se identificó la interrupción en ambas tecnologías.

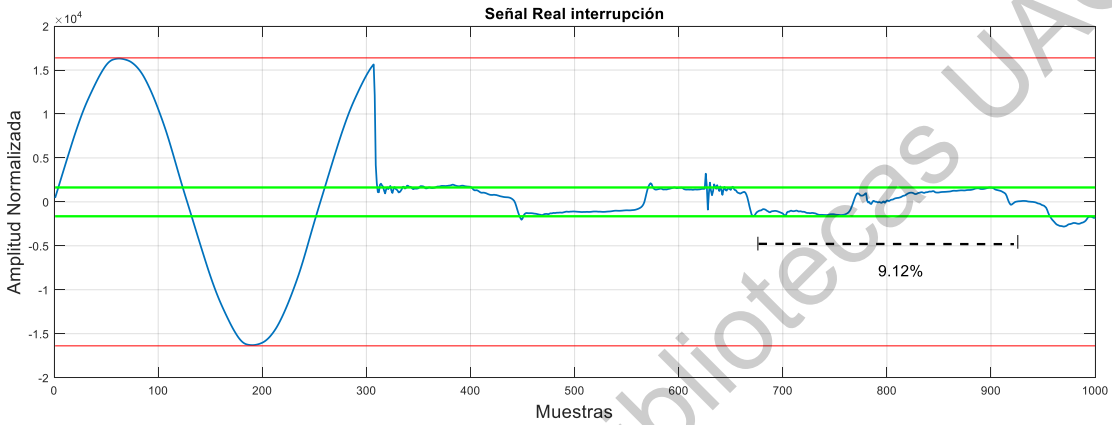


Figura IV.4. Señal real de interrupción en bancos de pruebas IEEE (IEEE, 2017).

En la gráfica de la Figura IV.4, se puede observar también los límites que se manejaron en las señales sintéticas, de esta manera es fácil identificar a simple vista el disturbio. El porcentaje de disturbio presentado fue calculado en el computador con un procesador Intel® Core™ i7-8750H CPU 2.21 GHz con una resolución de punto flotante en procesamiento de 64 bits, posteriormente en referencia con este valor se adquirió el margen de error con lo obtenido en cada dispositivo.

El resultado de los dispositivos es mostrado en el Cuadro IV.2, donde el porcentaje de *Vn*, son similares en la detección del disturbio, pero con un margen de error diferente, en el ARM el margen de error es 39 mil veces menor que el resultado en el dispositivo FPGA. Falta mencionar que el disturbio fue detectado en la misma muestra enviada, por lo que los dos dispositivos fueron capaces de identificarla a tiempo.

Cuadro IV.2. Margen de error señal real.

Dispositivo	% Vn	% margen de error
FPGA	9.1107	$59.8759\ e^{-3}$
ARM	9.1162	$1.5275\ e^{-6}$

Se llevaron a cabo pruebas con el disturbio sag, siguiendo los mismos pasos elaborados en la verificación de la interrupción, pero agregando dos señales más en caracterización de las incertidumbres. En la Figura IV.5, se muestran las señales sintéticas utilizadas como propuesta en la identificación correcta en a) se manejó un sag de 50.06% con una duración de 2 ciclos empezando por la secuencia negativa y en b) la señal representada en la gráfica es dividida en tres partes, que representan la simulación con un porcentaje de 90, 50 y 10, indicados en la especificación con duración de 3, 6, 1800 y 3600 ciclos.

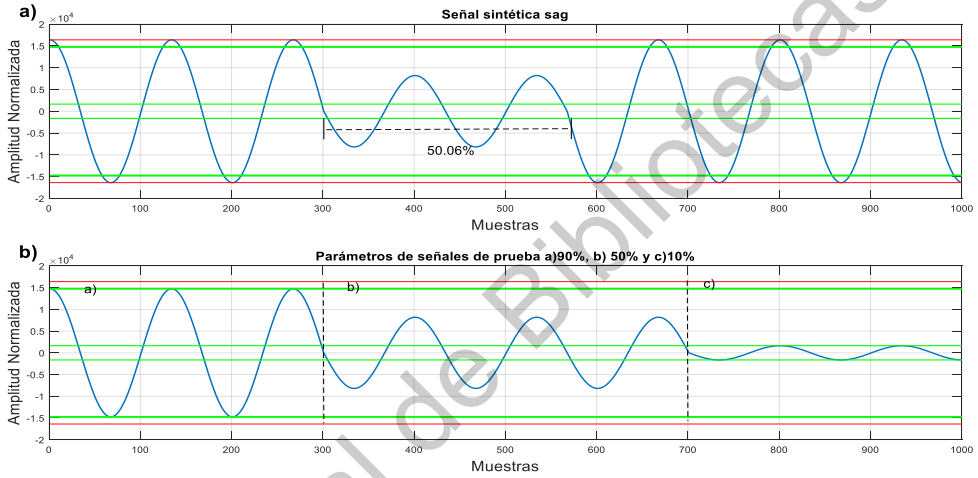


Figura IV.5. Señales sintéticas disturbio sag.

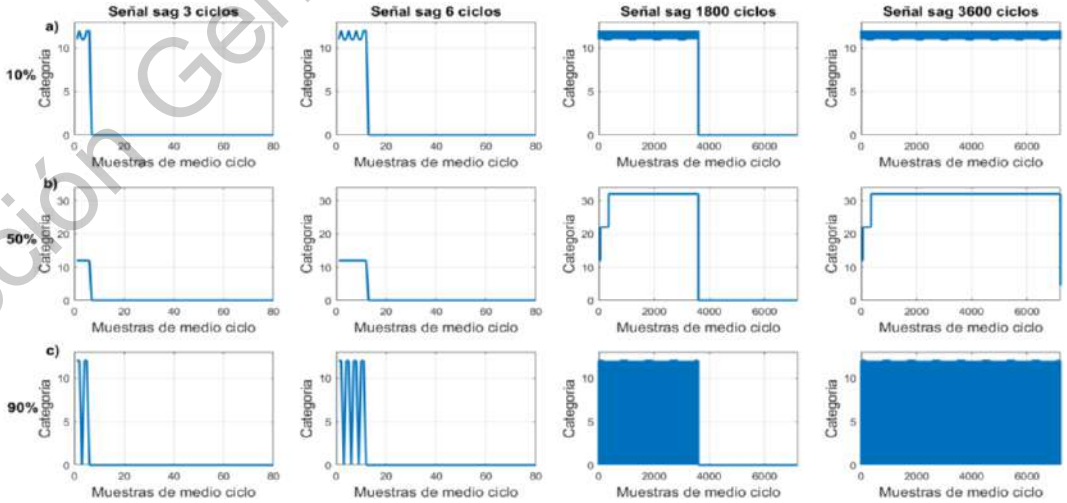


Figura IV.6. Categoría y disturbio en señal de a) 10%, b) 50% y c) 90%.

Por lo tanto, en los dos dispositivos en el momento de identificar el disturbio fueron idénticos como se muestra en la Figura IV.6, la representación del disturbio identificado varia por el porcentaje y la frecuencia de muestreo utilizada.

La gráfica a) con porcentaje de 10, tuvo una variación de un ciclo entre con el disturbio de interrupción, por lo tanto, la categoría se mantiene en instantánea, en la cantidad de 1800 ciclos se puede observar que la categoría se mantiene fija pero el disturbio oscila entre sag e interrupción hasta que la señal vuelve a estar sana. La gráfica b) de 50% en las muestras con todos los ciclos se mantiene estable sin variación de ciclos, mientras que la gráfica c) la variación es de medio ciclo con el porcentaje de 90, detectando como una señal sana y con sag ya que se encuentra en el límite de margen superior.

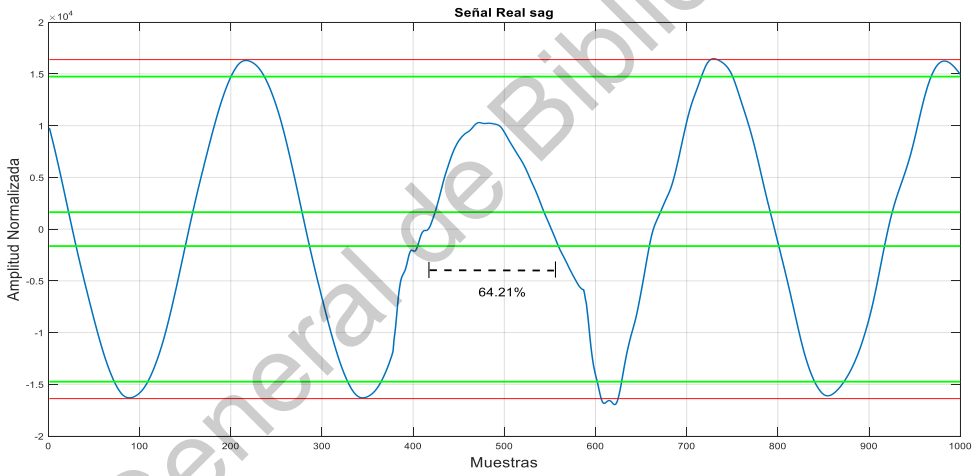


Figura IV.7. Señal real sag en bancos de pruebas (IEEE, 2017).

El resultado de los dispositivos como muestra en el Cuadro IV.3, el porcentaje de *Vn*, redondeado es idéntico en la detección del disturbio con el computador, el margen de error se diferencia en el ARM, este es mil veces menor que el resultado en el dispositivo FPGA.

Cuadro IV.3. Margen de error señal real sag.

Dispositivo	% Vn	% margen de error
FPGA	64.2024	$10.9103\ e^{-3}$
ARM	64.2094	$1.0109\ e^{-6}$

En margen de error aquí expuesto, no es acumulativo ya que solo se presentó en medio ciclo la perturbación.

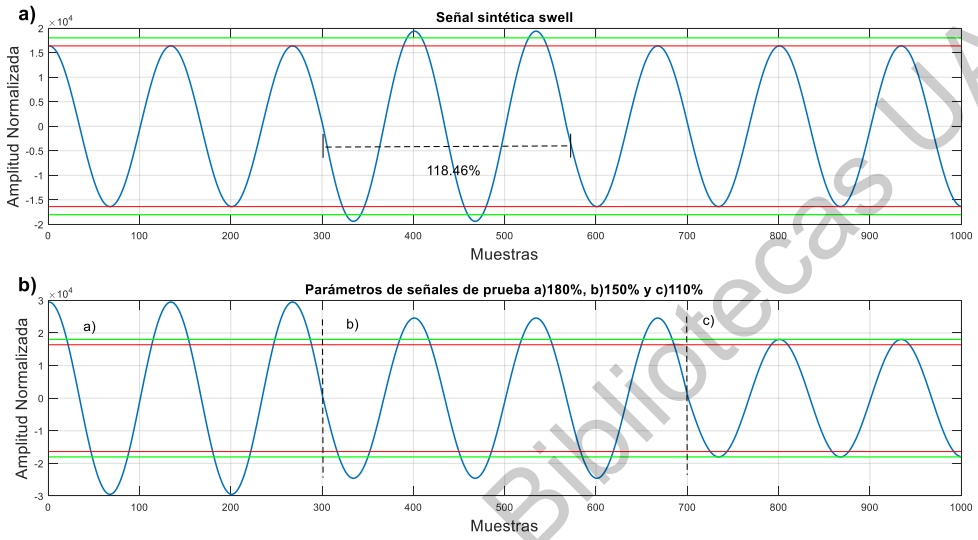


Figura IV.8. Señales sintéticas disturbio swell.

Se realizaron pruebas con el disturbio swell, siguiendo los mismos pasos elaborados en las verificaciones anteriores. En la Figura IV.8, se muestran las señales sintéticas utilizadas como propuesta en la identificación del disturbio, en a) se manejó un swell de 118.46% con una duración de 2 ciclos empezando por la secuencia negativa y en b) la señal representada en la gráfica es dividida en tres partes, que representan la simulación con un porcentaje de 180, 150 y 110, indicados en la especificación con duración correspondientes al anterior.

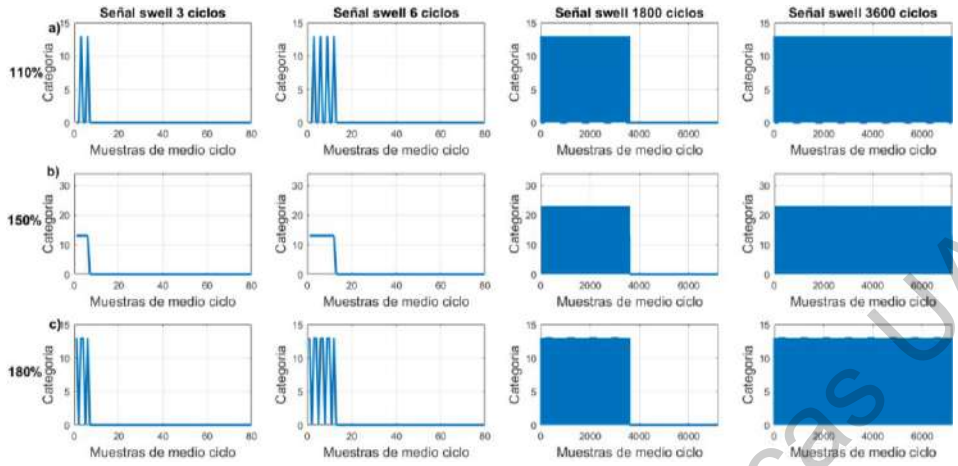


Figura IV.9. Categoría y disturbio en a) 110%, b) 150% y c) 180%.

Las gráficas en la Figura IV.9, En a) el porcentaje de 110, tuvo una variación de un ciclo entre el disturbio y una señal sana, por lo tanto, la categoría se mantiene en instantánea, manteniendo este resultado hasta la cantidad de ciclos donde se encuentra la perturbación. La b) de 150% en las muestras con todos los ciclos se mantiene estable sin variación en los primeros 3 y 6 ciclos, cuando la categoría pasa de instantánea a momentánea, cambia también el límite superior del disturbio swell a un porcentaje de 140% por lo tanto, la categoría regresa a instantánea y se ve las fluctuaciones en las gráficas de 1800 y 3600 ciclos. Por último, la gráfica c) la variación es de medio ciclo con el porcentaje de 180, detectando como una señal sin perturbaciones y con swell, ya que se encuentra en el límite de margen superior.

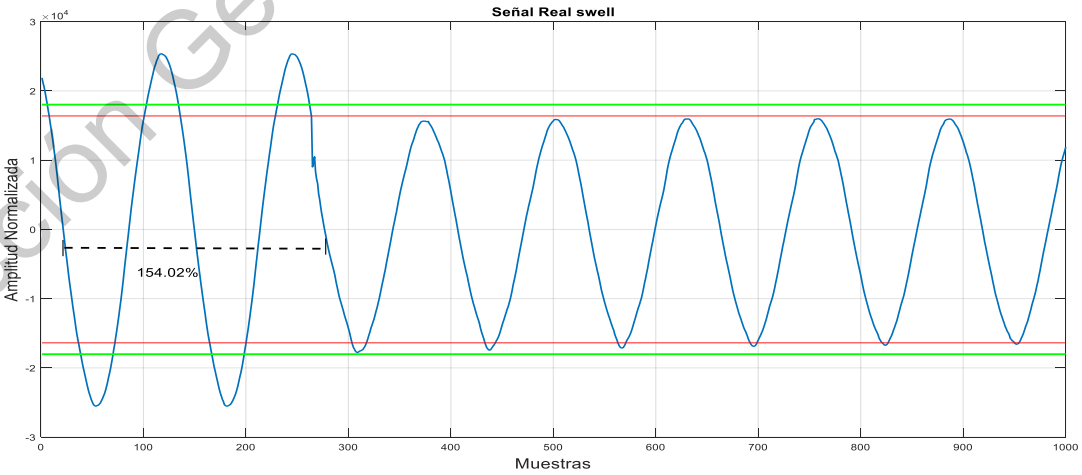


Figura IV.10. Señal real swell en bancos de pruebas (IEEE, 2017).

Cuadro IV.4. Margen de error señal real swell.

Dispositivo	% Vn	% margen de error
FPGA	154.0128	$23.3138\ e^{-3}$
ARM	154.0164	$1.3329\ e^{-6}$

En la Figura IV.10 en conjunto con el Cuadro IV.4, se observan los márgenes de error obtenidos de una señal real, inicial en el arranque de un grupo de capacitores de un variador, el disturbio que se detecta es un swell con una duración de dos ciclos y un porcentaje 154.32%, la identificación en los dispositivos fue correcta, pero con la diferencia en el margen de error del FPGA y el ARM, la incertidumbre es por debajo del establecido. Cumpliendo así con la especificación para ambas tecnologías.

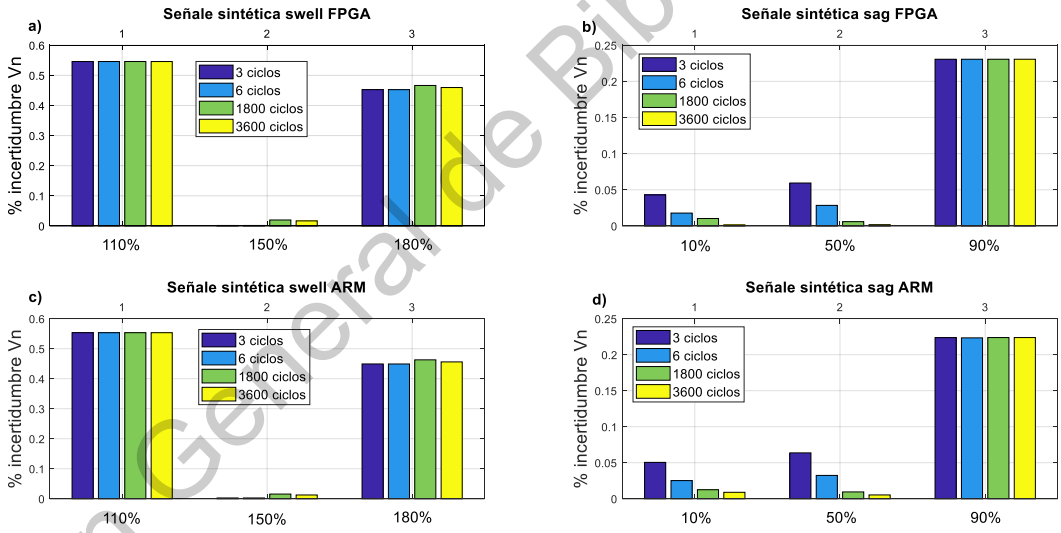


Figura IV.11. Incertidumbre en voltaje Vn en disturbio sag y swell.

En la Figura IV.11, la gráfica de barras muestra la incertidumbre que se obtuvo en los disturbios sintéticos swell y sag, con sus respectivos porcentajes de voltaje, las barras de color azul oscuro son 3, azul claro de 6, verde 1800 y 3600 ciclos del disturbio. En el comportamiento de incertidumbre tanto en el FPGA y el ARM fueron similares, mostrando menor incertidumbre en un swell de 150% y sags con 10 y 50 %.

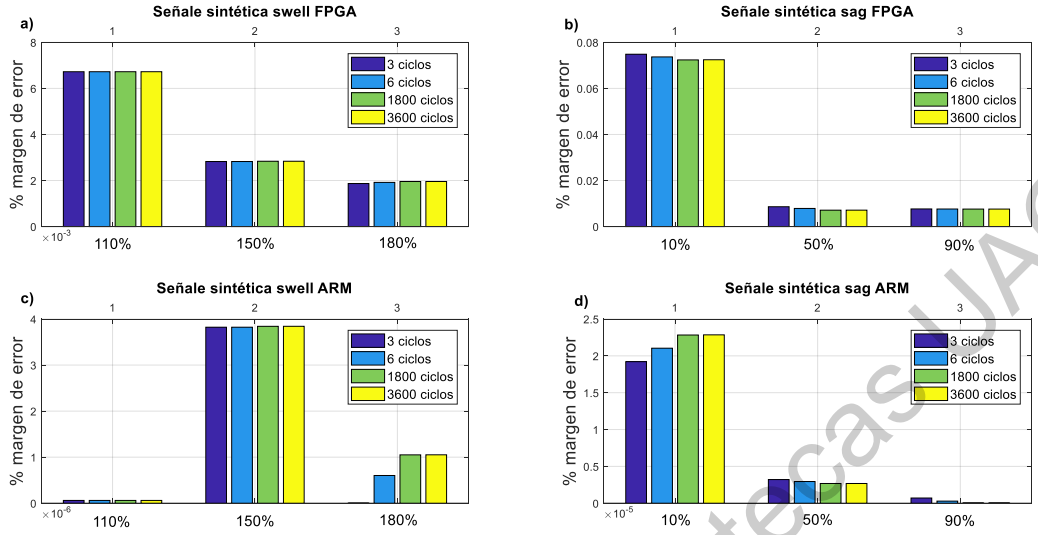


Figura IV.12. Margen de error en disturbio sag y swell.

Por otro lado, en la Figura IV.12, en el FPGA conforme aumentaba el porcentaje del disturbio swell, el margen de error disminuye en comparación al ARM, el mayor incremento de error se obtuvo en el porcentaje de 150. En comparación, el disturbio sag entre más aumenta el porcentaje de voltaje el margen de error se hace menor.

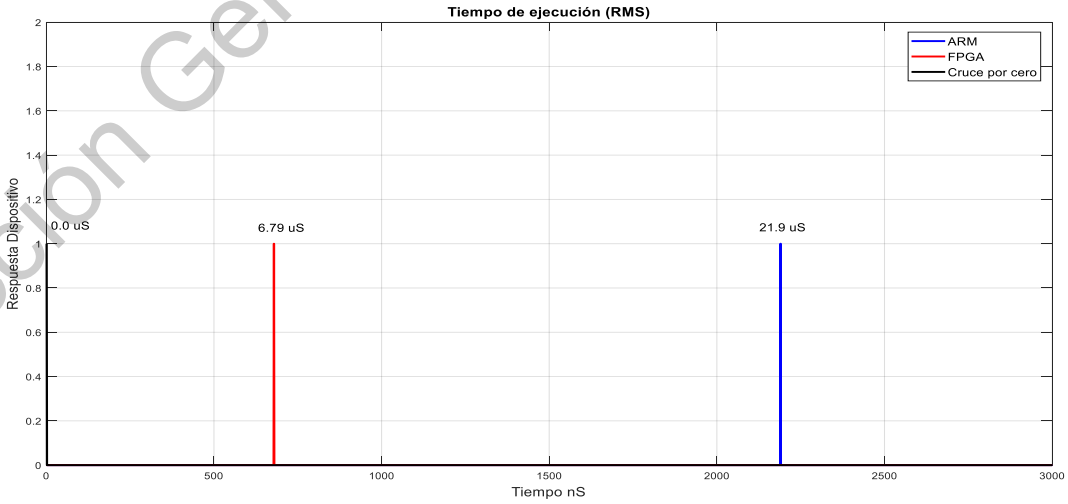


Figura IV.13. Tiempo de ejecución en ARM y FPGA.

En la Figura IV.13, se muestra por medio de un impulso el tiempo de ejecución que le toma al dispositivo realizar el cálculo del RMS y la identificación del disturbio, después de que ocurriera un cruce por cero. El FPGA su tiempo de ejecución es óptimo, ya que el dispositivo ARM tarda aproximadamente 4 veces más. A pesar de estos resultados, las dos tecnologías en el tiempo de ejecución entregado se encuentran dentro del margen a tiempo real, debido a que la frecuencia de muestreo del sistema de adquisición le toma 125 us adquirir un dato nuevo. La frecuencia de muestreo empleada en este trabajo generó inexactitudes en el cálculo de los algoritmos de calidad de la energía. El error fue aproximadamente de 0.1% observado en las variaciones de pu dentro de los límites en los disturbios.

4.1.2. Validación del algoritmo THD

A continuación, en esta sección de muestran los resultados que se obtuvieron en las pruebas realizadas en el módulo THD, diseñado en el FPGA y ARM. El tamaño de la ventana utilizado en el procesamiento de la FFT, es de 8192 datos por canal. El documento de la CFE pide que la medición de THD sea tanto en las fases de voltaje como las de corriente, el cálculo del THD tiene que ser hasta el 25vo armónico, el resultado en tensión la variación no debe sobrepasar al 6.5% y que la incertidumbre de los armónicos individuales obtenidos con la transformación del dominio de tiempo al dominio de la frecuencia, se mantenga dentro del rango ± 3%.

En la metodología se propuso una señal sintética de 60 Hz que está compuesta del cuarto y décimo armónico para validar solo el funcionamiento del módulo como se mostró en la Figura III.3, y el resultado entregado por el módulo debe de ser de 17.98%. Al implementar el algoritmo en el FPGA se observó que la respuesta proporcionada por la FFT está dividida en dos debido a esto el bit menos significativo se pierde y baja su resolución; con ello, el error que siempre se obtiene en la medición será de un bit, lo que indica que el error acumulativo será de $1.5259e^{-3}$ %. El error acumulativo se maneja como tal puesto que en el algoritmo de THD se suman los productos de los armónicos hasta llegar al requerido.

El resultado en los dispositivos en el módulo THD fue correcto manejado en porcentaje manteniéndose dentro del rango de incertidumbre establecido en la especificación. Aunque en el FPGA como se mencionó con anterioridad el error del bit no afecto en la respuesta final, esto es considerando los primeros cuatro dígitos de la parte decimal. Se presentan más pruebas realizadas sobre el módulo para medir las variaciones en los dispositivos.

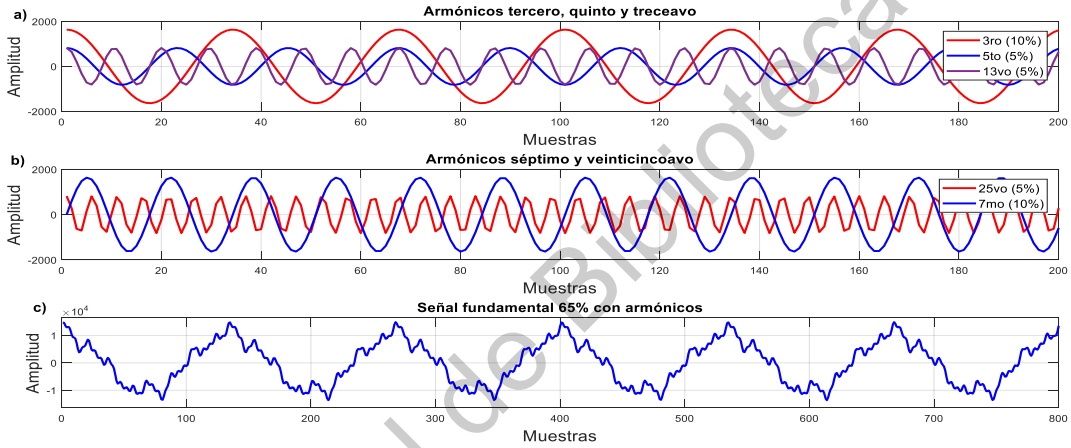


Figura IV.14. Gráficas con armónicos a) 3ro, 5to, 13vo, b) 7mo y 25vo, y señal sintética resultante.

En la Figura IV.14, se observan los armónicos usados para el diseño de la señal sintética utilizada en las variaciones de incertidumbres en los armónicos y margen de error del módulo en cada dispositivo. Se manejaron 512 ventanas de 8192 datos en cada canal con la señal de la gráfica c), pero el resultado fue el mismo, por lo tanto, solo se presenta una ventana por dispositivo que representa los canales de voltaje y corriente. El módulo THD cuenta con un filtro IIR Butterworth de 6 orden, en el caso del ARM se manipula un buffer de memoria por canal, permitiendo que el filtrado de la señal sea continuo una vez que inicializa, por otro lado, en el FPGA solo tiene un bloque de filtrado así que la retroalimentación de las variables *x* y *y* son borradas al finalizar una ventana de datos por canal.

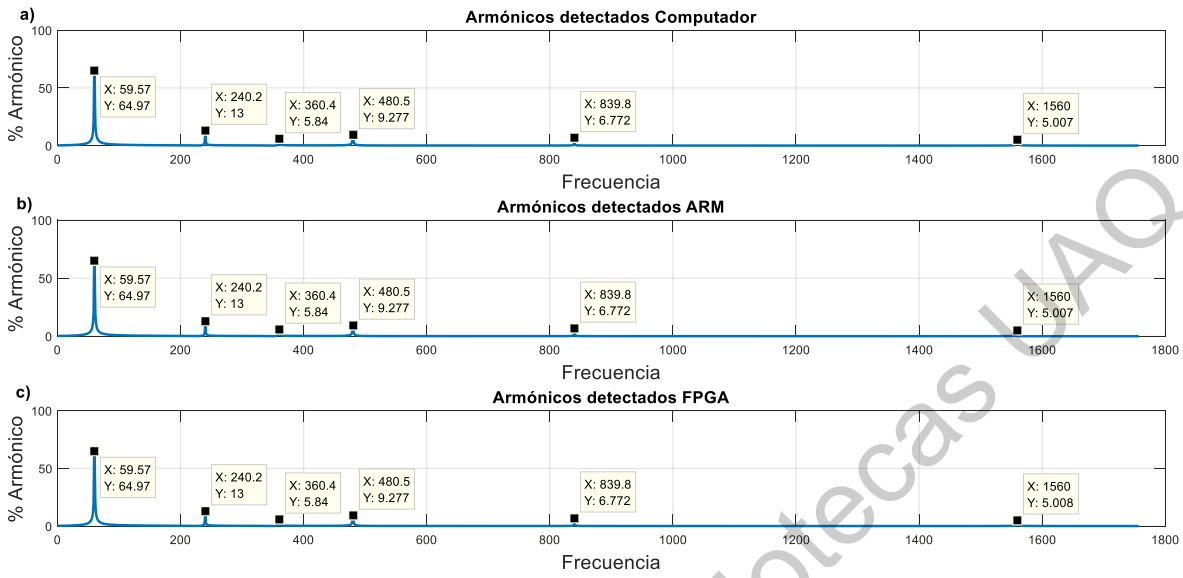


Figura IV.15. Gráficas de armónicos detectados en a) computador, b) ARM y c) FPGA.

Como se muestra en la Figura IV.15, las gráficas representan los armónicos detectados por las diferentes tecnologías como son: un computador (punto de referencia), el ARM y el FPGA. La amplitud de los armónicos en el eje Y es representada en porcentaje y en el eje X la frecuencia, está ultima es definida por la resolución de la FFT así que la frecuencia no son múltiplos enteros como se puede observar. En el Cuadro IV.5, contiene los armónicos y su representación en frecuencia con la resolución de la FFT, el porcentaje de margen de error e incertidumbre por cada uno de ellos y su promedio.

Cuadro IV.5. Margen de error e incertidumbre en armónicos.

Armónicos	% error		% Incertidumbre	
	FPGA	ARM	FPGA	ARM
3ro (240.2)	$3.7886\ e^{-6}$	$3.5895\ e^{-8}$	-2.9993	-2.9994
5to (360.4)	$20.0927\ e^{-6}$	$4.0516\ e^{-8}$	-0.8374	-0.8375
7mo (480.5)	$54.2405\ e^{-6}$	$1.7959\ e^{-8}$	0.7265	0.7265
13vo (839.8)	$0.7965\ e^{-6}$	$3.1099\ e^{-8}$	-1.7691	-1.7695
25vo (1560)	$11.6062\ e^{-6}$	$34.3025\ e^{-8}$	-0.0054	-0.0054
Promedio	$18.1049\ e^{-6}$	$9.3699\ e^{-8}$	1.2676	1.2676

El dispositivo FPGA tiene aproximadamente doscientas veces más error que el ARM, pero las dos tecnologías se encuentran dentro del margen de incertidumbre indicada en la especificación en la parte de armónicos en tensión y corriente encontrada en la tabla 15 del documento. En el Cuadro IV.6, se muestra el porcentaje del THD y el margen de error en cada dispositivo con referencia al computador. En este caso el margen de error en el FPGA es aproximadamente la consideración que se explicó en la pérdida del bit menos significativo y a la vez es mil veces mayor al margen de error del ARM. Los resultados del cuadro son el promedio de las 512 ventanas utilizadas en la validación del módulo.

Cuadro IV.6. Porcentaje de THD y margen de error.

Dispositivo	% THD	% margen de error
FPGA	29.2372	$1.168\ e^{-6}$
ARM	29.2376	$0.6981\ e^{-8}$

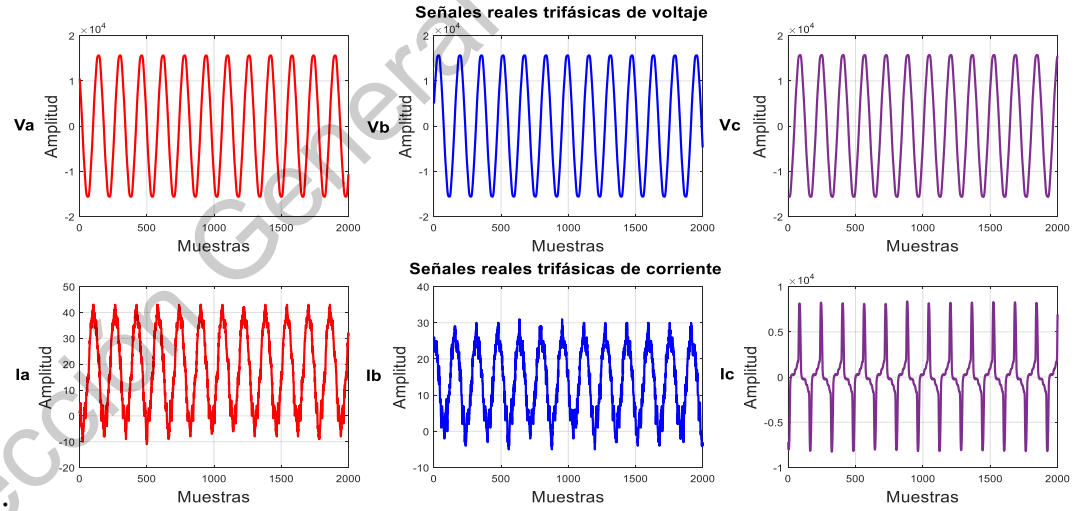


Figura IV.16. Señales reales de un sistema trifásico.

Con los bancos de pruebas de señales reales con frecuencia de 50 Hz y muestreadas a 8000 Hz, se validó la funcionalidad del módulo en el cálculo del THD para cada fase de

corriente y voltaje. En la Figura IV.16, se muestra la señal eléctrica utilizada, donde el consumo de corriente es menor en las fases a y b, representando una desconexión en la carga, mientras que en la fase c la cantidad de corriente es considerable y la forma de onda está distorsionada por lo que la cantidad de armónicos es notable.

Cuadro IV.7. Distorsión Armónica en señales reales.

Tipo de señal	FPGA				ARM			
	a	b	c	error	a	b	c	error
Corriente	10.4885	14.4935	75.9881	$20.0276e^{-3}$	10.4916	14.498	75.9881	$0.9324e^{-8}$
Voltaje	1.5974	1.3666	1.6331	$1.1263e^{-6}$	1.5974	1.3666	1.6331	$0.1906e^{-8}$

En el Cuadro IV.7 se muestran el porcentaje promedio del THD obtenido en cada canal de voltaje y corriente en cada dispositivo respecto a 358 ventanas, el resultado en corriente en las fases a y b es diferente en el FPGA en comparación con el ARM es debido a error que se mencionó con respecto a la FFT, también por el formato manejado donde los armónicos más pequeños de $0.1221e^{-3}$ son redondeados a cero. El porcentaje promedio de error es notorio también en las señales de corriente en el FPGA, tiene un crecimiento aproximadamente de 20 veces al margen obtenido de las señales sintéticas, en comparación del ARM el margen de error se mantiene estable. Con las pruebas realizadas en las dos tecnologías se comprueba que el error en el ARM tiene una oscilación entre $0.1\text{-}1.0e^{-8}$ y en el FPGA de $0.5\text{-}20e^{-6}$ que es error acumulado por el método mencionado para detección de armónicos, a pesar de estos resultados la incertidumbre se encuentra dentro de lo establecido en la especificación.

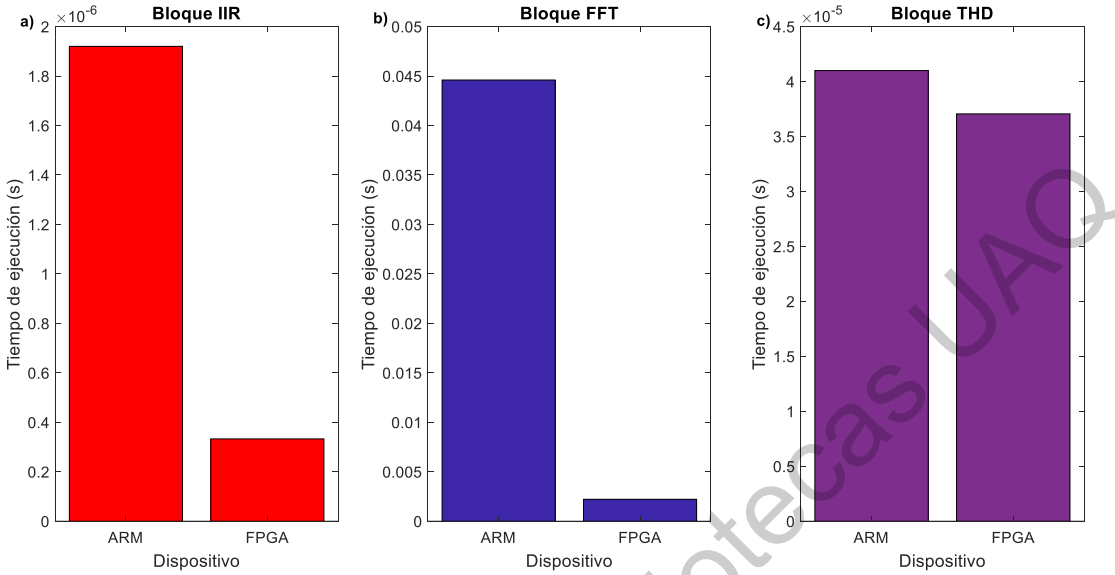


Figura IV.17. Tiempo de ejecución en bloques a) IIR, b) FFT y c) THD.

El tiempo de ejecución del módulo está dividido en tres partes esenciales como en el tiempo que le toma al bloque IIR filtrar los datos de canal de entrada, el procesamiento de la FFT y el cálculo del THD conforme a la ecuación. En la Figura IV.17 se muestra el tiempo de ejecución en el dispositivo FPGA y el ARM, estas mediciones obtuvieron mediante un osciloscopio digital con un ancho de banda de 100 MHz. En el bloque IIR se refleja el tiempo que tarda en realizar el filtrado de solo un dato, debido a que actualización de información está dada por la frecuencia de muestreo, lo que quiere decir que el tiempo total de ejecución es 8192 veces al tiempo mostrado. Cuanto se ha realizado el filtrado del tamaño de la ventana, comienza la FFT hasta terminar el procesamiento de todos los datos, esta operación es la parte principal del módulo THD y del DSQ obteniendo la frecuencia fundamenta y los armónicos en formato complejo, siendo el bloque que le toma mayor tiempo en ejecutarse en el ARM que es aproximadamente 20 veces al FPGA. Por último, el bloque THD de tiempo de ejecución la diferencia no es de gran proporción debido que en el ARM se tuvo control directo de las direcciones donde se encontraban localizados los armónicos facilitando la operación, en cambio en el FPGA el diseño del bloque FFT permite la lectura en orden como se fue almacenando en la memoria interna del dispositivo hasta encontrar la dirección en donde se encuentra el armónico, generando un retardo en la operación.

4.1.3. Validación del algoritmo DSQ (desequilibrio)

En las pruebas de funcionalidad en el módulo DSQ, solo se utilizaron las propuestas en la parametrización de la CFE en el Cuadro III.1. Se generaron dos señales sintéticas como se muestra en la Figura IV.18, las señales presentan desbalance solo en la amplitud del voltaje donde las líneas verdes en la parte superior e inferior de cada señal es el rango de amplitud de una señal sana, el desfase de una señal de voltaje con respecto a otra es de 120°, las fases de voltaje están representadas en diferente color donde *Va* es roja, *Vb* azul y *Vc* morada. El manejo de ventanas es igual al que se utilizó en la validación del módulo THD, debido a que los valores reales e imaginarios de la frecuencia fundamental son proporcionados por este módulo.

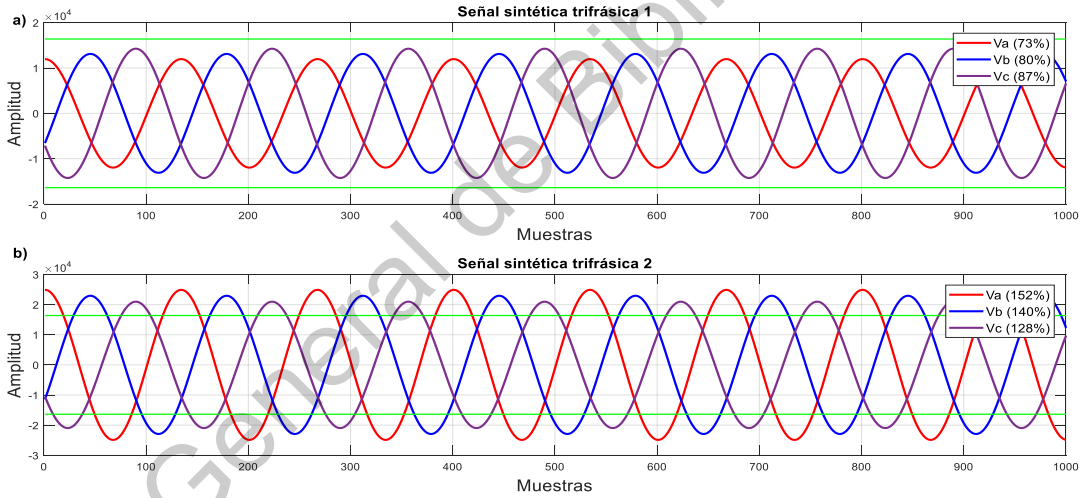


Figura IV.18. Señales sintéticas trifásicas.

El desequilibro esperado en la señal uno es de 5.05% y en la señal dos de 4.95%, este resultado esperado es el ideal, pero puede tener una variación ±0.3% mostrado en la incertidumbre del Cuadro III.1. En la Figura IV.19, se observan las variaciones que se tuvieron como respuesta en los dispositivos por ventana, realizando de esta manera lo establecido para medidores de Clase S. La ecuación (15) que se utilizó para la implementación de este módulo realiza las operaciones directas del número complejo de la

frecuencia fundamental, evitando el uso innecesario de otras operaciones como son el valor absoluto y el arco tangente para obtener el ángulo.

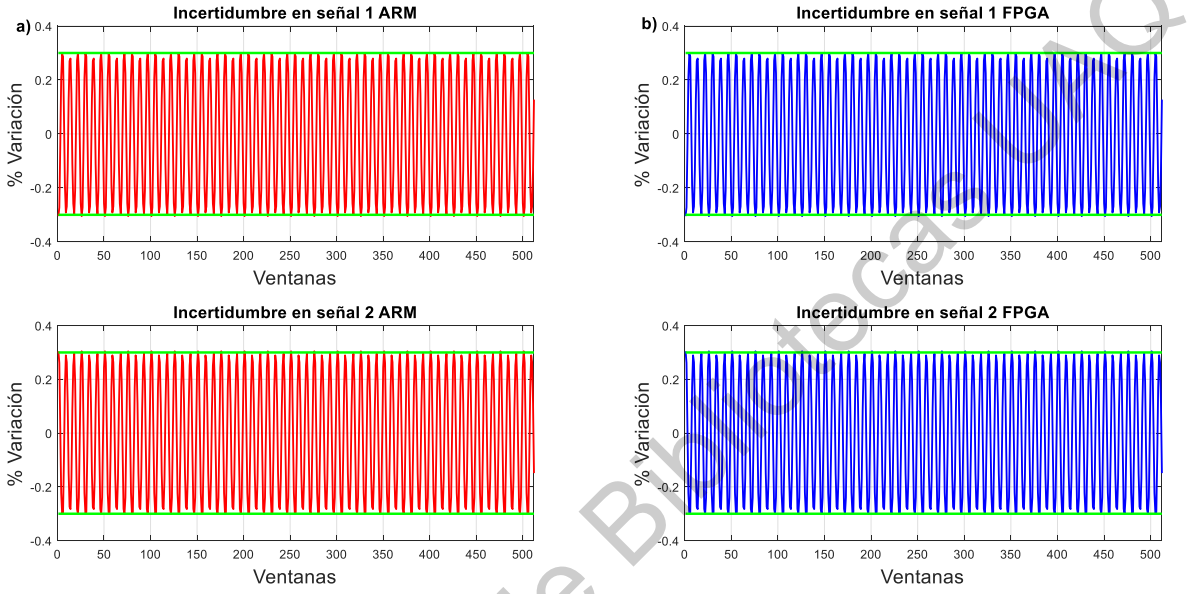


Figura IV.19. Porcentaje de Variación en desequilibrio en a) ARM y b) FPGA.

Cuadro IV.8. Variación y margen de error obtenido de señales.

Dispositivo	señal	% Variación	% margen de error
FPGA	1	0.2264	$0.9055\ e^{-3}$
	2	0.2264	$0.2277\ e^{-3}$
ARM	1	0.2264	$0.3551\ e^{-8}$
	2	0.2264	$0.3241\ e^{-8}$

En el Cuadro IV.8, se muestra el porcentaje de variación RMS que se tiene de las 512 ventanas usadas en cada dispositivo para la validación del módulo de desequilibrio, siendo el resultado igual en las dos tecnologías. El margen de error en el FPGA en comparación al análisis de los módulos anteriores es mayor al resultado de mil veces la diferencia con el dispositivo ARM, como se mencionó en la validación del módulo THD, con el uso de la FFT se tiene una pérdida de un bit o corrimiento hacia el menos significativo, como el algoritmo de desequilibrio hace uso del resultado real e imaginario de la fundamental proporcionado por la FFT perdiendo información y aumentando el margen de error.

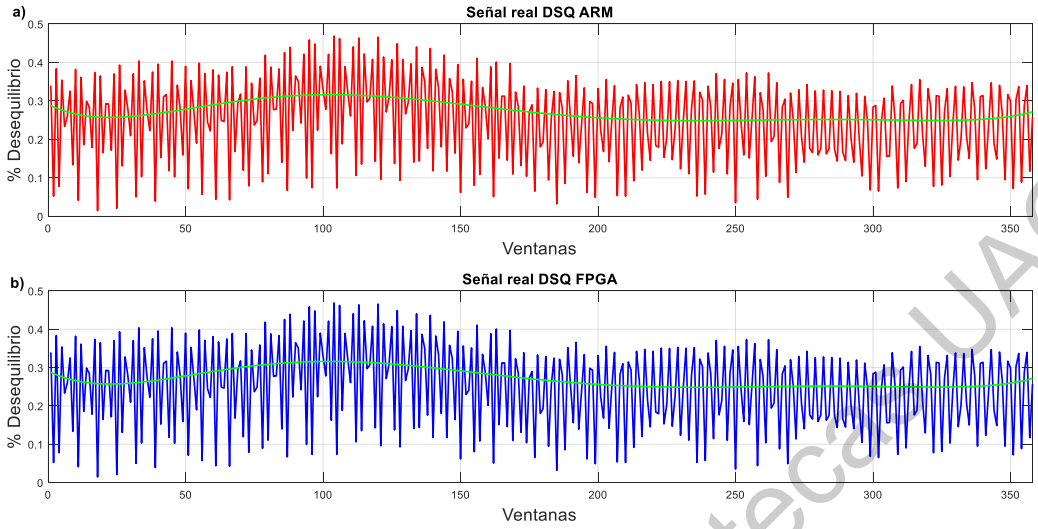


Figura IV.20. Desequilibrio en señal real.

Al realizar el procesamiento con las señales reales del sistema trifásico de 50 Hz utilizado en la sección de THD, se obtienen variaciones por ventana en el resultado del porcentaje en desequilibrio de fases de voltaje, por lo tanto, se generó una aproximación del comportamiento que debería de tener el módulo DSQ mediante estos resultados, permitiendo de esta manera medir el porcentaje de variación. La forma que se adquirió el porcentaje de variación fue por medio de la diferencia entre el resultado del módulo con el valor idealizado de nuestra función. En la Figura IV.20 se muestran los resultados en porcentaje de desequilibrio en cada dispositivo, además de color verde es la función generada del valor ideal esperado en la medición.

Debido a que nuestra función de resultados idealizados no pasa por la mitad de todos los puntos de la medición adquirida en el módulo DSQ, se hace el RMS de todas las soluciones para obtener el porcentaje de variación en las 358 ventanas. En el Cuadro IV.9 se observa el porcentaje obtenido de variación en el conjunto de todas las ventanas utilizadas de señales reales de voltaje y porcentaje promedio de margen de error. El porcentaje de variación del Cuadro IV.8 con respecto al que se está mencionando es mayor, se debe a que la frecuencia de muestreo que se maneja es múltiplo entero de la frecuencia de la red real,

quiere decir que se tiene 0.12 % de error en la medición de un sistema que trabaja a 60 Hz, pero se obtuvo un incremento en el margen de error.

Cuadro IV.9. Porcentaje de error y variación con la señal real.

Dispositivo	% Variación	% margen de error
FPGA	0.0963	$7.0017\ e^{-3}$
ARM	0.0963	$3.9613\ e^{-6}$

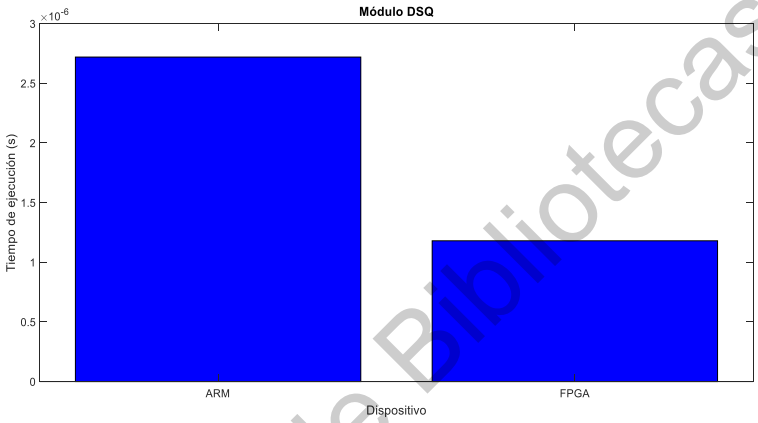


Figura IV.21. Tiempo de ejecución en módulo DSQ.

El tiempo de ejecución del módulo en los dispositivos está representado en la Figura IV.21, el ARM las operaciones son directas y se evitó el incremento de instrucciones, aunque el consumo de tiempo es un poco más del doble que en el FPGA, en el diseño del módulo no es requerimiento tener los valores de las frecuencias fundamentales de las tres fases completas, debido a que comienza a realizar las operaciones en el momento que se obtiene una fase.

4.2. Implementación de algoritmos

En la sección se expondré la implementación de los algoritmos de calidad de la energía y la comunicación con el sistema de adquisición de datos PQ-UAQ en los dispositivos FPGA marca *Spartan 6* y el ARM de la familia *Hercules*, también abarca la

cantidad de recursos y memoria utilizada en la aplicación de los algoritmos. El modelo de los dispositivos y sus características principales se presentó en la metodología en la subsección puesta en experimento.

Figura IV.22. Implementación de dispositivos.

En la Figura IV.22, se observan las dos diferentes tecnologías comunicadas con la tarjeta PQ-UAQ, el dispositivo FPGA es el encargado de configurar y obtener la información de las mediciones de cada canal, además es el encargado de realizar la comunicación con el ARM. La manera en que se transfiere los datos del PQ-UAQ es una comunicación serial SPI, que tarda 4 us en adquirir el dato completo de un canal, que son actualizados cada 125 us, la cantidad de canales que se usaron fueron 6 donde el orden de muestras es los primeros tres datos la parte de corriente y los últimos tres los de voltaje.

A continuación, se muestran el consumo obtenido en las implementaciones de los algoritmos de calidad de la energía en el dispositivo FPGA y ARM, teniendo la cantidad de recursos utilizados y cantidad de memoria requerida. En el Cuadro IV.10, se observa la cantidad de recursos usados en el FPGA y la cantidad de memoria que ocupa las instrucciones en el ARM con la implementación del RMS, THD, DSQ, PQ y el módulo principal, este último es solo aplicado en el dispositivo FPGA para establecer la comunicación del sistema de adquisición con el ARM.

Cuadro IV.10. Recursos y memoria consumida en los dispositivos.

Algoritmo/ Módulo	Dispositivo						% Prom. Utilizado	
	ARM		FPGA				ARM | FPGA	
	FLASH (bytes)	RAM (bytes)	No. Slices	No. LUTs	No. bloques RAM	No. DSP48A1s	Proceso	Memoria
RMS	22,732 (1%)	1,205 (0%)	943 (5%)	536 (5%)	N/A	2 (6%)	1 | 5.33	0 | 0
THD	415K (19%)	98,488 (18%)	890 (4%)	1086 (11%)	24 (75%)	5 (15%)	19 | 10	18 | 75
DSQ	21,124 (1%)	28 (0%)	355 (1%)	396 (4%)	N/A	1 (3%)	1 | 2.66	0 | 0
PQ	418K (19%)	98,673 (18%)	2,281 (12%)	2,303 (25%)	25 (78%)	8 (25%)	19 | 20.66	18 | 78
Principal	418K (19%)	98,673 (18%)	2,418 (13%)	2,540 (27%)	26 (81%)	8 (25%)	19 | 21.66	18 | 81

En el microcontrolador ARM, los únicos parámetros medibles son la cantidad de memoria ocupada, la cual es dividida en dos partes, memoria FLASH donde es almacenado las instrucciones del programa y memoria RAM que almacena las variables o datos manipulados en el programa. En la implementación del RMS la cantidad de memoria FLASH alcanza el 1% de su capacidad y el 5% de almacenamiento de variables en RAM, por lo tanto, solo se utiliza el 6% de las memorias total del dispositivo. Mientras que el dispositivo FPGA se ocupó 1362 Slices que representada en porcentaje es el 7% consumido en el dispositivo, la cantidad de LUTs fueron 790 que equivale al 8% y se usaron 2 multiplicadores (DSP48A1s) que equivalen al 6%, estos fueron utilizados en el proceso de acumulación y raíz cuadrada del bloque. En este caso se consideró la cantidad de Slices utilizados como prioridad en el diseño del RMS siendo así el total de los recursos consumidos en el dispositivo.

La implementación del THD maneja la mayoría de la memoria de los dispositivos, debido a que realiza la transformación de los datos en dominio del tiempo al dominio de la frecuencia mediante la FFT. En el funcionamiento de esta transformada la ventana de datos

utilizada es de 8192, permitiendo que el margen de error se mantuviera dentro de la especificación, en el FPGA los bloques de memoria tienen un tamaño de 16 bits mientras que en el ARM es de 32 bits, por el formato de punto flotante que maneja el dispositivo. Las instrucciones a realizar en el ARM aumentan debido al método de Radix-2 utilizado en el proyecto, en el FPGA el uso de los multiplicadores crece, no solo por el método también es por la implementación del filtro IIR y raíz cuadrada, aunque estos dos comparten un mismo multiplicador. El ARM utiliza el doble de recursos que el FPGA en los procesos.

El módulo que utilizó menos recursos en las dos tecnologías es DSQ, ya que la cantidad de operaciones fueron limitadas por el desarrollo de la ecuación (15), que utiliza de manera directa los números complejos de las frecuencias fundamentales, separando operaciones reales con operaciones imaginarias. En el caso del FPGA requiere de un multiplicador para realizar una sola operación de raíz cuadrada, por lo tanto, el tiempo de ejecución es también óptimo para los dos dispositivos, debido a que el valor no se requiere almacenamiento para posteriores operaciones, no se utiliza memoria en el FPGA ya que solo se utilizan registros para mantener los valores entregados y procesados en el bloque, por otro lado, el ARM la memoria que utiliza es para conservar el valor y poder realizar la operación con ellos.

4.3. Comparativa dispositivos

Con base en los resultados que se obtuvieron en los apartados anteriores se muestra en esta sección el contraste en costo-beneficio entre las dos tecnologías diferentes, comparando el cumplimiento de la especificación de la CFE, tiempo de ejecución, margen de error medidos del dispositivo FPGA y ARM, y el precio de estas tecnologías en el mercado.

Las dos tecnologías son adecuadas como medidores de Clase S, cumpliendo con el porcentaje de incertidumbre indicado en la especificación de la CFE sin consumir toda la capacidad de memoria y recursos del dispositivo que se utilizó en la implementación de los algoritmos de calidad de la energía. El tiempo de ejecución de los módulos en el FPGA fue menor a comparación del ARM cabe mencionar que el dispositivo en hardware la frecuencia

de operación es aproximadamente 6 veces menor que en el ARM usando un cristal de reloj de 48 MHz.

En el FPGA utilizando una resolución 2.16 existiría una disminución en el margen de error manteniendo la cantidad de bits máxima por multiplicador generando un pequeño incremento en el uso de Slices y LUTs. En el ARM el margen de error puede ser cero si el tipo de dato manejado es *double* que es el mismo tipo que maneja el computador que se tomó como referencia, al realizar este cambio la cantidad de consumo en memoria FLASH y RAM será el doble, ya que el dispositivo ARM el bus de instrucciones que maneja tiene un tamaño de 32 bits y el tipo *double* es de 64 bits para el almacenamiento de la información, sin embargo, tendrá un aumento del 50% en el tiempo de ejecución.

El ARM que se manejó en el trabajo tiene un costo de $59.99 dólares montado en la tarjeta de desarrollo *Hercules*, el precio del dispositivo es elevado debido a que en la actualidad su estado comercial se encuentra en su obsolescencia, dejando de producirse y solo están en venta los que se encuentran aún en stock en el mercado, mientras en el FPGA que se usó el integrado tiene un precio de $26.53 dólares siguiendo en producción por una cantidad de tiempo mayor que el ARM, la tarjeta de desarrollo DUA3 tuvo un costo en su diseño de prototipo de $270.00, que contiene entre otros integrados sensores.

Debido a que los dispositivos ARM se encuentra en constante cambio la duración en el mercado es de menor tiempo como se expuso con el *Hercules*, en el Cuadro IV.11 se muestran los dispositivos de diferentes marcas que están comercializados actualmente, pero con las características esenciales en frecuencia de operación, los recursos y memoria utilizados en la implementación de los algoritmos, de igual manera los dispositivos FPGA. Se puede observar que el costo de las dos tecnologías es similar y la variación es debido a la marca.

Cuadro IV.11. Dispositivos FPGA y ARM comerciales.

FPGA							
Modelo	**Marca**	**Slice**	**Slice LUTs**	**DSP [3]**	**RAM [2]**	**Frecuencia de operación (Máx. MHz)**	**Precio (dólar)**
T13F256C3	Efinix, Inc.	12,828 [1]		24	40	310	5.79
10CL006	Intel	6,272 [1]		15	30	200	11.23
XC6SLX9	Xilinx Inc.	11,440	5,720 [1]	16	32	250	17.08
ARM							
Modelo	**Marca**	**CPU (bits)**	**FLASH (bytes)**	**RAM (bytes)**	**Frecuencia de operación (Máx. MHz)**		**Precio (dólar)**
ATSAMS70	Microchip	32	512 K	256 K	300		5.05
STM32H723	STMicroelectronics	32	512 K	176 K	550		9.69
MIMXRT1064	NXP USA INC.	32	4 M	1 M	600		11.79

Notas:

1) Cantidad de Slice que contienen 4 LUTs.

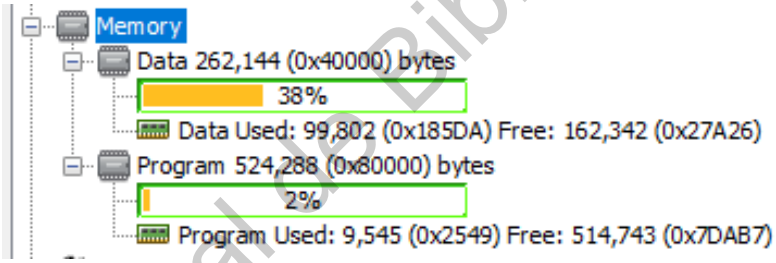
2) Bloques de RAM de 18 K bytes o dobles de 9 K bytes.

3) Multiplicadores con tamaño de 18 bits.

En los FPGA mostrados en el cuadro solo requieren un software de programación dependiendo de la marca del dispositivo y el compilador es el único encargado de optimizar la descripción diseñada en los RTL para los bloques de procesamiento, donde se pretende que no existe ningún cambio. Dentro de las propuestas se encuentra una versión anterior al dispositivo usado de la familia *Spartan 6*, que contiene los requerimientos indispensables para la implementación de los algoritmos de calidad de la energía, pero con un costo menor donde los recursos en la implementación fueron igual. Por otro lado, el ARM para su programación se utilizan dos softwares diferentes, uno es encargado de generar los registros necesarios para el funcionamiento dependiendo del modelo del integrado y el otro es encargado de compilar el algoritmo en lenguaje C específicamente para el dispositivo que se esté utilizando, donde el uso de memoria FLASH tiene variaciones conforme a la optimización del compilador y el tamaño de bus de instrucciones que se maneje, en cambio la memoria RAM debe permanecer dentro de los rangos utilizados al igual que el *Hercules* o la variación debe ser pequeña.

En la Figura IV.23 se muestra el consumo de memoria en la implementación de los algoritmos de calidad de la energía eléctrica en dos diferentes marcas de ARM, pero con la capacidad de memoria y frecuencia de operación necesaria que se utilizó en la implementación con el *Hercules*, evitando el aumento en tiempo de ejecución y margen de error. Los ARM que se usaron son las marcas conocidas como es *Microchip* y *STMicroelectronics* del Cuadro V.11, en a) el ARM Microchip la cantidad de memoria FLAH es menor que la que se usó en el *Hercules*, pero la cantidad de memoria es similar como pasó con b) *STMicroelectronics*, comparando los resultados en memoria el dispositivo *Microchip* tiene mejor compilador, los recursos en memoria son suficientes al igual que la frecuencia de operación y con un precio bajo.

a)



b)

Memory Regions | Memory Details

Region	Size	Used	Usage (%)
DTCMRAM	128 KB	0 B	0.00%
ITCMRAM	64 KB	0 B	0.00%
RAM_D1	320 KB	98.93 KB	30.91%
RAM_D2	32 KB	0 B	0.00%
RAM_D3	16 KB	0 B	0.00%
FLASH	512 KB	18.19 KB	3.55%

Figura IV.23. Implementación en a) Microchip y b) STMicroelectronics.

V. CONCLUSIONES Y PROSPECTIVA

5.1. Conclusión

Los algoritmos de calidad de la energía eléctrica implementados en el dispositivo FPGA y ARM cumplieron con las incertidumbres establecidas en la especificación de la CFE para medidores de Clase S, con un error aproximadamente 0.2%. El FPGA el tiempo de ejecución y la cantidad de recursos en el procesamiento son mejores en comparación con el ARM, además, una eficiencia mayor en la compilación de los algoritmos. El costo de estos dispositivos es parecido tomando la cantidad de memoria, recursos y frecuencia de operación necesarias en este trabajo.

En la implementación de los algoritmos fue necesario el uso de memoria externa para el almacenamiento de los datos de medición en el FPGA ya que se excedían los recursos de memoria RAM por la cantidad de ventanas para cada canal. Sin considerar la parte de memoria en el FPGA, los recursos utilizados en conjunto a los bloques no son mayores al 22% de la capacidad del dispositivo para el procesamiento y el 78% para almacenamiento de datos de la FFT e información de bitácoras de evento. Por otra parte, en el ARM el consumo es de aproximadamente el 18% de su capacidad de memoria de almacenamiento sin el uso de memoria externa y el 19% en la ejecución de instrucciones.

El tiempo de ejecución del FPGA en los módulos es menor debido a que su proceso es de manera concurrente siendo entre 4-6 veces más rápido que las instrucciones en secuencias del ARM. El algoritmo que requirió una cantidad de tiempo considerable fue el cálculo de la FFT en los dos dispositivos; además, es posible reducir el tiempo de ejecución en el FPGA utilizando un reloj no mayor a 76 MHz. Por otro lado, no es posible aumentar el tiempo de ejecución en el ARM ya que se trabajó a la máxima frecuencia que el dispositivo opera.

El margen de error en los dos dispositivos dio resultados por debajo del 0.2% sin considerar la frecuencia de muestreo cumpliendo lo establecido en la especificación. Se observó que es posible disminuir el margen de error utilizando de memorias externas para las dos tecnologías, modificando el almacenamiento interno de las operaciones de la FFT; de esta manera se aumenta la resolución.

Comparando ambas tecnologías se obtuvo que el ARM otorga un error en escala de 10^{-6} que es mil veces menor al FPGA ya que este su margen está por los 10^{-3}. El error del FPGA se puede reducir, pero se necesitaría el doble o un poco más de los recursos ya que los multiplicadores son de 18 bits (respecto al FPGA utilizado en el proyecto); por ello, el margen de error entraría en consideración principalmente en este recurso.

Finalmente, en la parte de costo-beneficio, los dispositivos en hardware permanecen activos por mayor tiempo en el mercado como lo es el *Spartan 6* (FPGA), y los dispositivos en software la curva de obsolescencia tiene mayor inclinación por las continuas actualizaciones como en el caso de los dispositivos *Hercules* (ARM) que en la actualidad dejaron de fabricarse y los existentes se encuentran a un alto costo en comparación a otras marcas con características similares. En las tarjetas de desarrollo donde se colocan los integrados, en caso del FPGA, son un prototipo que tuvieron un costo elevado debido a que se producen en cantidades pequeñas y su fabricación no está comercializada, además por los sensores que tiene integrados en la tarjeta.

Con los recursos utilizados en cada dispositivo, es posible indicar que el consumo monetario es del 50% del valor del dispositivo FPGA. Es factible implementar el algoritmo en una versión anterior existente al integrado con el mismo compilador donde la cantidad de recursos es similar pero el costo del dispositivo es menor. En el caso del ARM el consumo monetario es de 18% de su costo, sin embargo, no existen mejoras en el integrado siendo obsoleto al igual que el compilador y menos eficaz en comparación a las otras marcas de ARM donde se realizó la implementación.

Uno de los beneficios de la implementación de los algoritmos es la factibilidad para ejecutarse en otros dispositivos, ya que los algoritmos implementados tuvieron un desarrollo

genérico que facilitó manipular su estructura adaptándola al sistema de adquisición de datos y tamaño de ventana que se quiera procesar, facilitando al usuario ponerlo en marcha pero contemplando el límite de sus recursos, memoria y frecuencia de operación; si alguno de estos factores no se cumple, no se garantiza un funcionamiento correcto. La descripción de los módulos en el FPGA fue mediante VHDL, que es manejado en cualquier marca donde la optimización se lleva a cabo por el compilador, teniendo mejor beneficio ya que no requiere direccionamientos de periféricos de memoria, vectores de interrupción, frecuencia de operación y comunicaciones seriales, mientras que, en los ARM, a pesar de que el lenguaje de programación en C sea universal, en este tipo de dispositivo será necesario conocer las características de sus periféricos. En cuestión de tiempo de ejecución en el FPGA, la implementación de los algoritmos en otra marca no es afectado. En el caso del ARM, el compilador generó instrucciones en manejo de memoria de 32 bits, siendo la duración específica en este tipo de modelo. En caso que se quiera implementar en un ARM diferente, pero con características similares, el tiempo de ejecución no será el mismo que se reportó en este proyecto a pesar de que los algoritmos sean genéricos ya que las instrucciones están limitadas a la cantidad de bits de memoria.

5.2. Prospectiva

Una recomendación para el uso del equipo desarrollado en las dos tecnologías, es la utilización de un sistema de adquisición donde la frecuencia de muestreo sea múltiplo entero de la frecuencia de la red eléctrica que se quiera monitorear. En el trabajo se trabajó a 8000 muestras por segundo para una frecuencia en la red de 60 Hz, generando error en la medición donde se notó en el cálculo del RMS.

Desarrollar la comunicación del dispositivo ARM con el sistema de adquisición PQ-UAQ sin la intervención del FPGA, para utilizar los dispositivos independientes uno del otro con su propio convertidor. Demostrando así que la transferencia de datos no es afectada entre la comunicación del dispositivo, ya que si falla el FPGA a la vez terminaría sin conexión con

el muestreo de la señal el ARM de esta manera no se podría hacer la comparación de funcionalidad en campo.

Realizar pruebas del equipo en laboratorios en México para analizar la susceptibilidad al ruido o campos magnéticos, realizar la comparación de la bitácora de eventos que entregan los dispositivos con el equipo de medición de calidad de la energía FLUKE, finalmente hacer pruebas y validación en campo como son: residencias e industrias.

A pesar de que este proyecto la idea principal es orientada hacia el monitoreo en línea de una red convencional, el trabajo implementado no está limitado, ya que se puede utilizar para monitorear cualquier señal eléctrica. Con respecto a esto, una de las aplicaciones donde es posible utilizar los equipos implementados en sistemas de generación de energías renovables, en el análisis de generadores, en la parametrización de la norma en paneles solares entre otros, puede ser aplicado el proyecto realizado. Además del monitoreo de señales, el equipo es capaz de usarse en la detección de fallas en motores.

Tomando los recursos utilizados como referencia, la integración de los algoritmos pueden ser aplicadas en marcas diferentes y de menor precio en el dispositivo, lo cual dependerá de las necesidades del usuario. El margen de error en los ARM disminuiría si se emplea un tipo *double* y la ventana del FFT al doble, donde el error máximo se da por la frecuencia de muestreo no múltiplo de la señal eléctrica de 60 Hz, duplicando el consumo en las memorias FLASH y RAM manteniéndose dentro de los rangos de la capacidad del dispositivo. En caso del FPGA, en el módulo FFT se deberá emplear una memoria externa para el procesamiento si se requiere aumentar la ventana y mantener la cantidad de recursos en el procesamiento.

VI. Referencias

Balcells, J. (2001). Calidad de la red eléctrica:¿ como medirla? Online: Http://Www. Jcee. Upc. Es/JCEE2001/PDFs, 22–26. Retrieved from http://www.jcee.upc.es/JCEE2001/PDFs 2000/5BALCELLS.pdf.

Bilik, P. (2009). Measurement of voltage and current harmonics for frequencies up to 9 kHz according to IEC61000-4-7. 2009 10th International Conference on Electrical Power Quality and Utilisation, EPQU'09, 1–5. https://doi.org/10.1109/EPQU.2009.5318832.

Broshi, A. (2007). MONITORING POWER QUALITY BEYOND EN 50160 AND IEC 61000-4-30. Power Quality, 155–187. https://doi.org/10.1201/9781420064827.ch8.

Bashardoust, A., Farrokhifar, M., Fard, A. Y., Safari, A., & Mokhtarpour, E. (2016). Optimum network reconfiguration to improve power quality and reliability in distribution system. *International Journal of Grid and Distributed Computing*, *9*(4), 101–110. https://doi.org/10.14257/ijgdc.2016.9.4.10

Cardona, D., López, D., & Hernández Suárez, C. A. (2013). Diseño y construcción de un prototipo para la medición de distorsión armónica en redes eléctricas. *Revista Tecnura*, *17*(35), 107. https://doi.org/10.14483/udistrital.jour.tecnura.2013.1.a09

Casaravilla, G., & Echinope, V. (2010). *Desbalances - Estudio de alternativas para su estimacion*. Retrieved from http://epim2005.fing.edu.uy/trabajos/p12.pdf

Casellas, F., Velasco, G., & Guinjoan. (2010). El concepto de Smart Metering en el nuevo escenario de distribución eléctrica. *Universitat Politècnica de Catalunya. Departament d'Enginyeria Electrònica*, 752–757. Retrieved from https://upcommons.upc.edu/bitstream/handle/2117/9066/5025.pdf

CFE. (2006). *Watthorímetros monofásicos y polifásicos electrónicos, clase de exactitud 0,5*.

CFE. (2010). *Medidores multifunción para sistemas eléctricos*.

Ferrigno, L., & Landi, C. (2007). *A Study on the Feasibility and Effectiveness of Digital Filters Approach for Power Quality Monitoring in Compliance with IEC 61000-4-7*.

Gallo, D., Ianniello, G., Landi, C., & Luiso, M. (2010). An advanced energy/power meter based on ARM microcontroller for smart grid applications. *17th Symposium IMEKO TC4 - Measurement of Electrical Quantities, 15th International Workshop on ADC Modelling and Testing, and 3rd Symposium IMEKO TC19 - Environmental*

Measurements, (February 2015), 310–315.

Granados-Lieberman, D., Valtierra-Rodriguez, M., Morales-Hernandez, L. A., Romero-Troncoso, R. J., & Osornio-Rios, R. A. (2013). A Hilbert transform-based smart sensor for detection, classification, and quantification of power quality disturbances. *Sensors (Switzerland)*, *13*(5), 5507–5527. https://doi.org/10.3390/s130505507

Guillén-García, E., Morales-Velazquez, L., Zorita-Lamadrid, A. L., Duque-Perez, O., Osornio-Rios, R. A., & Romero-Troncoso, R. D. J. (2019). Accurate identification and characterisation of transient phenomena using wavelet transform and mathematical morphology. *IET Generation, Transmission and Distribution*, *13*(18), 4021–4028. https://doi.org/10.1049/iet-gtd.2019.0101

IEC 61000-4-30. (2008). *IECc 61000-4-30. Electromagnetic compatibility (EMC) Part 4-30:Testing and measurement techniques* (p. 134). p. 134.

IEC 61000-4-7. (2008). *IEC 61000-4-7*. EMC Part4-7: Testing and measurement techniques - General guide on harmonics and intherharmonics measurements and instrumentation, for power supply systems and equipement connected thereto.

IEEE. (2017). online. Retrieved from www.grouper.ieee.org/groups/1159/2/testwave.html.

IEEE Std 1159. (2009). *IEEE Standard 1159-2009: IEEE Recommended Practice for Monitoring Electric Power Quality* (Vol. 2009). https://doi.org/10.1109/IEEESTD.2009.5154067

IEEE Std 519. (2014). IEEE Recommended Practice and Requirements for Harmonic Control in Electric Power Systems IEEE Power and Energy Society. *ANSI/IEEE Std. 519, 2014*. https://doi.org/10.1109/IEEESTD.2014.6826459

Jaramillo, A. (2013). Efecto de las perturbaciones: huecos de tensión, desequilibrios de tensión y armónicos, en los motores de inducción con rotor Jaula de Ardilla. *INGE@ UAN-Tendencias En La Ingeniería*, 36–51. Retrieved from http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Efecto+de+las+perturbaciones+:+huecos+de+tensión+,+desequilibrios+de+tensión+y+armónicos+,+en+los+motores+de+inducción+con+rotor+Jaula+de+A+rdilla#0%5Cnhttp://scholar.google.com/scholar?hl=en&b

João, L. A., Batista, J., & Sepúlveda, M. J. (2007). *Sistema Digital de Bajo Coste para la Monitorización de la Calidad de Energía Eléctrica Low-Cost Digital System for Power Quality Monitoring*. *18*, 15–23.

Neumann, R. (2007). The importance of IEC 61000-4-30 Class A for the coordination of power quality levels: Is it important? *2007 9th International Conference on Electrical Power Quality and Utilisation, EPQU*. https://doi.org/10.1109/EPQU.2007.4424226

Peña, R. A., Gómez, V. A., & Hernámndez, C. (2010). Diseño y construcción de un prototipo de medición de los indicadores de calidad del servicio de energía eléctrica (DES y FES) para usuario residencial. *CISCI 2010 - Novena Conferencia*

Iberoamericana En Sistemas, Cibernetica e Informatica, 7to Simposium Iberoamericano En Educacion, Cibernetica e Informatica, SIECI 2010 - Memorias, *7*(2), 48–53.

Ramírez, S., & Cano, E. (2006). Calidad del Servicio de Energía Eléctrica. In *Universidad Nacional de Colombia* (Primera). Colombia.

Real-Calvo, R., Moreno-Munoz, A., Pallares-Lopez, V., Gonzalez-Redondo, M. J., Moreno-Garcia, I. M., & Palacios-Garcia, E. J. (2017). Sistema Electrónico Inteligente para el Control de la Interconexión entre Equipamiento de Generación Distribuida y la Red Eléctrica. *RIAI - Revista Iberoamericana de Automatica e Informatica Industrial*, *14*(1), 56–69. https://doi.org/10.1016/j.riai.2016.11.002

Rens, J., De Kock, J., Van Wyk, W., & Van Zyl, J. (2014). The effect of real network phase disturbances on the calculation of IEC 61000-4-30 parameters. *Proceedings of International Conference on Harmonics and Quality of Power, ICHQP*, 303–306. https://doi.org/10.1109/ICHQP.2014.6842886

Ria, I., Ladino, E., & Mart, F. H. (2012). Implementación de la transformada FFT sobre una FPGA orientada a su aplicación en convertidores electrónicos de potencia. *Tekhne (Universidad Distrital Francisco Jose de Caldas)*, *9*, 21–32.

Rohrig, C., Rudion, K., Styczynski, Z. A., & Nehrkorn, H. J. (2012). Fulfilling the standard en 50160 in distribution networks with a high penetration of renewable energy system. *IEEE PES Innovative Smart Grid Technologies Conference Europe*, 1–6. https://doi.org/10.1109/ISGTEurope.2012.6465766

Rojas, H. E., Rivas, E., & Jaramillo, A. A. (2014). *Aspectos técnicos y normativos para el monitoreo y medición de armónicos Technical and Regulatory Aspects for Monitoring and Measurement of Harmonics on de arm ´*. 6–11.

Romero, M., Pardo, R., & Gallego, L. (2011). Developing a PQ monitoring system for assessing power quality and critical areas detection. *Ingeniería e Investigación*, *31*(2), 102–109. Retrieved from http://www.scielo.org.co/scielo.php?pid=S0120-56092011000500016&script=sci_arttext&tlng=pt

Ruiz, M. G., & García, M. (2015). Interoperabilidad entre medidores inteligentes de energía eléctrica reutilizando redes celulares. *Prim. Congr. Int. y Expo Científica*, (January 2014). Retrieved from https://www.researchgate.net/publication/259869925

Sánchez, M. R., Salmerón Revuelta, P., Pérez Litrán, S., & Pérez Vallés, A. (2012). Análisis de las definiciones de desequilibrio de tensión en los sistemas de potencia. *Dyna (Spain)*, *87*(2), 198–203. https://doi.org/10.6036/4375

Tarasiuk, T., Szweda, M., & Tarasiuk, M. (2011). Estimator-analyzer of power quality: Part II - Hardware and research results. *Measurement: Journal of the International Measurement Confederation*, *44*(1), 248–258. https://doi.org/10.1016/j.measurement.2010.09.048

Villablanca, M. (2009). Medidores Del Mañana. *Ingeniare. Revista Chilena de Ingeniería*, *16*(3), 392–393. https://doi.org/10.4067/s0718-33052008000300001

VII. Anexos

7.1. Artículo publicado

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO
FACULTAD DE INGENIERÍA
DIPFI POSGRADO
13° COLOQUIO

Se otorga la presente

CONSTANCIA a:

LEONARDO ESTEBAN MORENO SUÁREZ

Por su participación como **Asistente** y **Presentador Oral** en el evento:

13° Coloquio de Posgrado de la Facultad de Ingenería

20, 21 y 22 de Noviembre de 2019
Facultad de Ingeniería

Dr. Manuel Toledano Ayala
DIRECTOR
FACULTAD DE INGENIERÍA

Dr. Juan Carlos Jáuregui Correa
Jefe de la Divisón de Investigación y Posgrado
FACULTAD DE INGENIERÍA

“Comparación en el desempeño entre FPGA y procesador ARM, para la detección de armónicos e inter-armónicos basado en la norma IEC61000-4-7.”

“Performance comparison between FPGA and ARM processor, for harmonics and interharmonics detection based on the IEC61000-4-7 standard.”

Leonardo E., Moreno-Suárez.[1] Luis, Morales-Velázquez.[2] Roque A., Osornio-Ríos.[1*]
[1]Facultad de Ingeniería, Universidad Autónoma de Querétaro, Campus San Juan de Río, San Juan del Río 76807, México.
[2]HSPdigital-CA Mecatrónica, Facultad de Ingeniería, Universidad Autónoma de Querétaro, Campus San Juan de Río, San Juan del Río 76807, México.
*leo.12121134@hotmail.com

Resumen

Se utilizó la norma IEC61000-4-7 que establece la manera en cómo deben ser medidos los armónicos e inter-armónicos, así como su correcta interpretación. Por lo tanto, se realizó una comparativa en desempeño en carga computacional y resultados de dos dispositivos diferentes, para un instrumento de PQ (calidad de la energía) como lo son: FPGA (arreglo de compuertas programables en campo) y procesador ARM (microprocesador). Se desarrolló la descripción en hardware y programación software de los algoritmos para: un filtro digital IIR (respuesta infinita al impulso) Butterworth de orden 6, FFT (transformada rápida de Fourier) radix 2, la descripción de la arquitectura de un divisor y raíz cuadrada en FPGA, mediante aproximaciones sucesivas, y con esto se obtuvo el THD (distorsión armónica total) de diferentes ventanas de una señal eléctrica normalizada. Los resultados en tiempo de ejecución del FPGA es seis veces más favorables que el Procesador ARM. La resolución es estable en el cálculo de los procesos en punto fijo del FPGA y puede ser aumentada, pero a un costo elevado.

Palabras Clave: *armónicos, inter-armónico, FPGA, ARM, IEC61000-4-7.*

Abstract

The IEC61000-4-7 standard was used, which establishes the way in which harmonics and inter-harmonics should be measured, as well as their correct interpretation. Therefore, a comparison was made in computational load performance and results of two different devices, for a PQ instrument (power quality) such as: FPGA (field programmable gate array) and ARM processor (Advanced RISC Machine). The description in hardware and software programming of the algorithms was developed by: a digital filter IIR (infinite impulse response) Butterworth of order 6, FFT (fast Fourier transform) radix 2, the description of the architecture of a divisor and square root in FPGA, by successive approximations, and with this the THD (total harmonic distortion) of different windows of a normalized electrical signal was obtained, the results at FPGA runtime are six times more favorable than the ARM Processor. The resolution is stable in the calculation of FPGA fixed point processes and can be increased but a high cost.

Keywords: *MEC, harmonics, interharmonics, FPGA, ARM, IEC61000-4-7.*

1. Introducción

La calidad en el suministro eléctrico en la red pública es de importancia como fuente popular de energía, es usada en actividades residenciales, comerciales e industriales. Por ello, en el monitoreo de calidad de la energía (Power Quality, PQ) deben de satisfacer lo establecido en ciertas normas o estándares. La norma internacional IEC 61000-4-7 [1], cuenta con los últimos cambios en el área de instrumentos de medición de armónicos e inter-armónicos. Se conoce como distorsión armónica a la deformación de la onda de su característica sinusoidal pura original de múltiplos enteros, por su parte los inter-armónicos están definidos como los voltajes o corrientes que tienen componentes de frecuencia que no son múltiplos enteros de la frecuencia a la cual el sistema de suministro está diseñado para funcionar (frecuencia 50 Hz o 60 Hz). Pueden aparecer como frecuencias

discretas o como un espectro de banda ancha. Los efectos de los inter-armónicos no son bien conocidos, pero se ha demostrado que afectan la señalización de la portadora de la línea de alimentación e inducen parpadeo (flicker) visual en dispositivos ópticos como los tubos de rayos catódicos [2].

Un análisis matemático para obtener los componentes armónicos e inter-armónicos es utilizando la Transformada Rápida de Fourier (Fast Fourier Transform, FFT), que es la técnica más común utilizada para el análisis de los espectros armónicos. La FFT transforma una señal discreta en el dominio de tiempo a su representación en el dominio de la frecuencia, de una manera rápida y eficiente, la cual la convierte en un método sencillo en el análisis de las perturbaciones que presenta la red eléctrica [3].

En la señal eléctrica para evitar armónicos innecesarios a los que se quieran analizar se utiliza un filtrado. Existen diferentes tipos de filtros que se pueden considerar en la implementación para el monitoreo de PQ, por ejemplo, los filtros de repuesta de impulso finita (Finite Impulse Response, FIR), por mencionar algunos como: Equiriple, Interpolated Fir, etc., y los filtros de respuesta de impulso infinita (Infinte Impulse Response, IIR) que son: los filtros Butterworth, Chebyshev, Elliptic, Maximally Flat, etc. La selección del tipo y arquitectura del filtro dependerá de los disturbios estimados en el diseño.

Algunos trabajos que han sido implementados en el desarrollo de un instrumento de monitoreo de PQ, donde para realizar la monitorización de la calidad de energía eléctrica se basa en la utilización de una tarjeta de adquisición de datos (Data Acquisition, DAQ) y un ordenador, mostrando los resultados en un software con entorno de programación grafica (Lab-VIEW) [4], dejando de lado el cálculo de los parámetros de PQ en el ordenador y fuera del dispositivo DAQ con la finalidad de solo adquirir la señal y enviarla al ordenador encargado de realizar las operaciones necesarias y mostrar los resultados obtenidos. De otra manera en la función de control de calidad de la energía eléctrica se han elaborado instrumentos basados en un microprocesador [5] encargado de solo enviar los datos al ordenador de la señal eléctrica muestreada por el convertidor analógico a digital (Analog to Digital Converter, ADC) de tal manera que el ordenador realice los cálculos y graficar los resultados. Sin embargo, el instrumento de monitoreo de PQ llamado PQ1000 el cual consta de un procesador digital de señales (Digital Signal Processor, DSP) [6], los algoritmos de mediciones de calidad de la energía son desarrollados por el dispositivo con una arquitectura de punto flotante y enviando solo los resultados obtenidos. Por otro lado, el instrumento de medición para el monitoreo de PQ basado en FPGA [7], el cual consta de bloques definidos para el filtrado digital de la señal eléctrica capturada por el ADC, para su procesamiento con la norma IEC 61000-4 y al final un bloque de memoria para almacenar los resultados.

En este artículo se realizará la comparativa de un instrumento de PQ implementado en los dispositivos basados: en un arreglo de compuertas programable en campo (Field Programmable Gate Array, FPGA) y en un Procesador ARM (Advanced RISC Machine) que es un procesador basado en la arquitectura RISC (Reduced Instruction Set Computer), los cuales tendrán el mismo diagrama a flujo de operaciones. El resultado entregado de cada dispositivo será comparado con los obtenidos en la simulación con el software MATLAB, las comparaciones que serán realizadas en los dispositivos es en desempeño, carga computacional (recursos y/o memoria del dispositivo) y margen de error (punto fijo y punto flotante) respectivamente de cada dispositivo. Así brindarle al lector, información de las variaciones que se pueden tener en los resultados, la capacidad al escoger el dispositivo (memoria, frecuencia de operación, multiplicadores, etc.) sea suficiente y tiempos de operación.

2. Marco Teórico

En la sección se mostrará las definiciones que nos otorga la norma internacional para la medición e interpretación de las ecuaciones requeridas en el instrumento PQ, en la parte de armónicos e inter-armónicos. También se hablará de la utilidad y diseño del filtro digital en el acondicionamiento, el tipo de filtro y el orden que se recomienda que se ha utilizado en trabajos anteriores.

2.1. Norma Internacional

La norma IEC 61000-4-30 [8] define el método para la medición e interpretación de los resultados obtenidos en el instrumento PQ del suministro de energía a frecuencias de 50/60 Hz a.c (corriente alterna).

Los requisitos para la medición de los armónicos e inter-armónicos están definidos en la norma, que especifica [7]:

La estructura general del instrumento de medición.

La precisión del instrumento.

Las características de las señales a medir.

Los tipos de medición.

Los intervalos de medición de los armónicos se realizan por medio de ventanas, 10 ciclos para una frecuencia de 50 Hz y de 12 para la frecuencia de 60 Hz, así obtener componentes espectrales con paso de 5 Hz, pero no es indispensable que las componentes espectrales tengan una magnitud de separación de este rango, se hace de esta manera para tener mediciones más prácticas para una mejor facilidad en su manejo en los cálculos [4].

La distorsión armónica total (Total Harmonic Distortion, THD) es definida como la raíz cuadrada de la suma de los cuadrados de las magnitudes de las componentes armónicas individuales y dividido por la magnitud de la componente fundamental Ec. (1). La componente fundamental es la frecuencia de mayor interés, además de ser la componente de mayor magnitud en el espectro del dominio de la frecuencia. Para sistemas de potencia, la frecuencia fundamental es la frecuencia natural del sistema [5].

$$THD = \frac{1}{I_f}\sqrt{\sum_{n=2}^{H}(I_n)^2} \quad (1)$$

Donde:

I_f: Componente fundamental.

I_n: Componente armónica.

H: Número de armónico.

Las amplitudes de los n-th armónicos e inter-armónicos en los grupos Ecs. (2) y (3), y los subgrupos Ecs. (4) y (5) para una frecuencia de 50Hz con una ventana de 200 ms, pueden ser evaluadas como [7]:

$$G_{sg,n}^2 = \sum_{k=-1}^{1} C_{10n+k}^2 \quad (2)$$

$$C_{isg,n}^2 = \sum_{k=2}^{8} C_{10n+k}^2 \quad (3)$$

$$G_{g,n}^2 = \frac{C_{10n-5}^2}{2} + \sum_{k=-4}^{4} C_{10n+k}^2 + \frac{C_{10n+5}^2}{2} \quad (4)$$

$$C_{ig,n}^2 = \sum_{k=1}^{9} C_{10n+k}^2 \quad (5)$$

Donde:

n: número de armónico.

C: componente espectral obtenida.

Enfocando el uso de grupos, los índices de calidad de energía están definidos en grupo (THDG) Ec. (6) y subgrupo (THDS) Ec. (7) de la distorsión total de armónicos. Puede ser evaluada respectivamente como [7]:

$$THDG = \frac{1}{G_{g1}} \sqrt{\sum_{n=2}^{H} \left(G_{gn}\right)^2} \quad (6)$$

$$THDS = \frac{1}{G_{gs1}} \sqrt{\sum_{n=2}^{H} \left(G_{gsn}\right)^2} \quad (7)$$

Donde:

H: orden especificado de los componentes armónicos.

G_n: valor eficaz del armónico *H.*

G_1: componente armónica fundamental.

Donde H especifica el orden de armónico grupo/subgrupo definido en la norma [8].

2.2. Filtro digital

La utilización de filtros digitales optimizados en la evaluación de PQ la carga computacional y el tiempo de ejecución tiene importancia significativa en el dispositivo donde se requiera implementar, por ello se debe seleccionar la arquitectura, el tipo y el orden del filtro digital adecuadamente al proceso a realizar. Los filtros digitales pueden desarrollar diferentes propósitos ya sea como pasa-altas, pasa-bajas, pasa-banda, etc., En la selección del filtro digital en el instrumento PQ se considera la evaluación del ultimo armónico de la señal eléctrica para el cálculo de THD establecido en la norma, siendo así la frecuencia de corte y otro dato necesario es la frecuencia a la que se está muestreando la señal eléctrica. El filtro IIR es un sistema cuya salida depende de las anteriores, el tamaño de las operaciones dependerá del número de orden del filtro, por lo que la Ec. (8) en diferencias general es de la siguiente manera:

$$y(n) = \sum_{k=0}^{N} b_k * x(n-k) - \sum_{k=1}^{N} a_k * y(n-k) \quad (8)$$

Donde:

N: es el número de orden del filtro.

a: coeficiente del filtro.

b: coeficiente del filtro.

3. Metodología / Materiales y Métodos

La metodología que se planteó en este trabajo se puede observar en las Fig. 1 y 3, los datos son previamente adquiridos de las mediciones realizadas por el dispositivo PQUAQ, es una tarjeta desarrollada en la Universidad Autónoma de Querétaro (UAQ), cuenta con un ADC especial para mediciones de calidad de energía eléctrica y almacena los datos en memoria extraíble. Los datos obtenidos serán almacenados en cantidades y ancho de la palabra (bits) de mismo tamaño en memoria ROM (Read Only Memory) en los dispositivos FPGA y procesador ARM, al igual que en el ordenador con el software MATLAB. Realizar los bloques y/o arquitecturas de las operaciones necesarias dentro de las instrucciones del módulo y/o función, para posteriormente comparar

los recursos o memoria utilizada, y medir el desempeño entre el Procesador ARM (procesamiento en software) y el FPGA (procesamiento en hardware). Utilizando MATLAB, se extraerán los resultados de los datos almacenados en memoria RAM (Random Access Memory) en cada dispositivo por medio de comunicación serial UART (RS232), de tal manera poder verificar los resultados entregados de cada dispositivo con los obtenidos en MATLAB.

Las especificaciones que se deben de considerar con los dispositivos FPGA y Procesador ARM son: la programación en el Procesador ARM ejecuta instrucciones precisas en secuencia que comprende el algoritmo, mientras en el FPGA la arquitectura es concurrente donde el diseño puede ejecutar 2 o más instrucciones en forma paralela. El Procesador ARM cuenta con una unidad de punto flotante de 32 bits, en los FPGA la descripción del algoritmo debe de manejarse en punto fijo, donde el usuario puede seleccionar la cantidad de bits para representar los números enteros y los números fraccionarios.

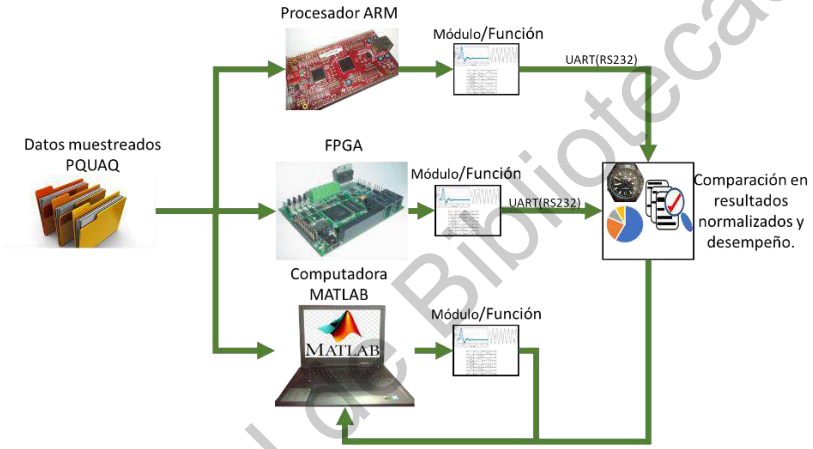


Figura 1. Modelado de los dispositivos en operación para comparación de desempeño y resultados.

La arquitectura mostrada en la Fig. 2, son los bloques de operaciones que contiene el módulo/función que serán realizadas por los dispositivos. Los datos previamente guardados en memoria ROM de las mediciones obtenidas del PQUAQ, son datos muestreados por el ADC de la tarjeta a una frecuencia de 8 kHz de una red eléctrica de 60 Hz en formato de 16 bits con signo normalizados, donde el rango de voltaje es 110 Vrms. Haciendo uso de las funciones ya establecidas por el software MATLAB, como lo es la FFT y THD, se realizará las comparaciones de los resultados dados con estas herramientas con los resultados extraídos de cada dispositivo.

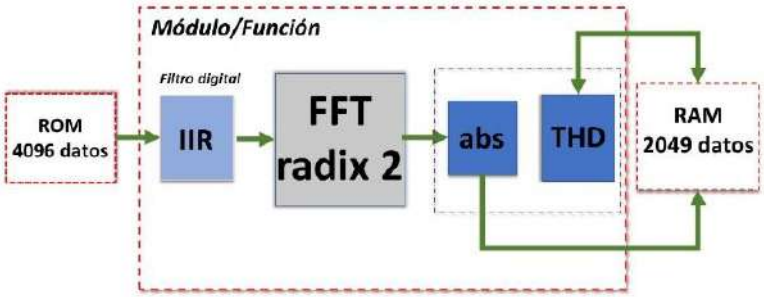


Figura 2. Arquitectura del módulo/función

Los datos almacenados en bloques de memoria ROM en el FPGA y en el Procesador ARM se generaron de MATLAB con la descripción o lenguaje de cada dispositivo de los valores guardados del PQUAQ, siendo completamente iguales, con un tamaño de 16 bits y con una resolución en formato 2.14 de punto fijo, es decir,

dos bits para la parte entera con signo y catorce bits para la parte fraccionaria, mostrada en la Fig. 3. En el diseño del filtro digital, se utilizó la herramienta que proporciona MATLAB para el cálculo de los coeficientes de un filtro IIR Butterworth de orden 6, donde la herramienta requiere el tipo de filtro siendo un pasa-bajas con una frecuencia de corte de 2400 Hz y la frecuencia de muestreo que es de 8 kHz. La frecuencia de corte tiene ese valor debido a que se tomara como limite el armónico 40. Los coeficientes en el diseño del FPGA es con un formato 2.14 debido a que las operaciones son en punto fijo y los coeficientes del filtro para el procesador ARM son valores directos generados por MATLAB, definidos como constantes en punto flotante.

Figura 3. Resolución de datos almacenados.

Los 4096 datos de la ROM pasan por el filtro digital IIR Butterworth pasa-bajas Fig. 2, se seleccionó este tipo de filtro ya que se caracteriza por un rendimiento aceptable, incluso si el orden del filtro es bastante bajo siendo de sexto orden que permite una buena estabilidad, rendimiento y un reducido tiempo computacional [9]. Al terminar de filtrase dato por dato de la ROM son reordenados hasta obtener una ventana de los 4096 datos, este cambio de ubicación de los vectores es necesario para la FFT. Una vez que se termina de reordenar los datos de la ventana se realiza la FFT la cual se implementó con el método de Radix 2, este bloque entrega como resultados de la operación números complejos y una ventana equivalente en espejo, por lo tanto, se realiza un bloque para el valor absoluto del resultado de la FFT, pero solo tomando la mitad de los datos procesados, es decir, 2048 datos, almacenándola en la memoria RAM. La memoria RAM tiene un rango de 2049 datos esto es debido a que se tendrá en cuenta un espacio para el resultado dado en el bloque THD. En el procesador ARM se hizo uso de la librería math.h, para calcular y almacenar previamente en vectores los valores de las funciones senoidales y cosenoidales, también la función sqrt() que se encarga de calcular la raíz cuadrada y obtener así resultado absoluto de los números complejos como resultado de la FFT. Por otra parte, en el dispositivo FPGA se describió la raíz cuadrada mediante aproximaciones sucesivas, antes de la operación del bloque se requiere de la multiplicación de la parte real y la parte imaginaria de los números complejos, se tendrá como resultado un ancho de la palabra al doble en bits. Por esa razón la entrada para el bloque de la raíz cuadrada es un registro de 32 bits, regresando como salida el resultado en 16 bits manteniendo el formato 2.14.

Al final, para el bloque de THD como se mostraron en las ecuaciones anteriores, se requiere la multiplicación y sumatoria de armónicos e inter-armónicos, y después ser dividido por la componente fundamental de la señal eléctrica (60Hz). Por ello los valores absolutos de la FFT una vez terminada la operación de cada dato se almacenaron en la memoria RAM, hasta tener los 2048 datos para poder ejecutar el bloque THD el cual se encargará de manejar los datos de la memoria RAM, llevando a cabo la sumatoria y multiplicación de los armónicos e inter-armónicos dando como resultado no solo el doble del ancho de la palabra, si no cinco bits más debido a la acumulación de bits de la sumatoria. Al igual que la raíz cuadrada, el bloque de división en el FPGA se utilizó el método de aproximaciones sucesivas, manteniendo el formato de salida de 2.14. Una vez que el bloque THD termina de hacer las operaciones el resultado es almacenado en la memoria RAM, teniendo como resultado una cantidad de datos de 2049 en la memoria del FPGA y Procesador ARM. Al final, para poder extraer los datos resultantes de cada dispositivo y enviarlos por comunicación UART (RS232) a la computadora para ser comparados con los obtenidos en MATLAB como se mostró en la Fig.1. El bloque de comunicación UART que se utilizó en el FPGA ya está implementado por lo cual no se hace mención explícita, al igual del Procesador ARM.

El dispositivo utilizado fue un FPGA Spartan 3 XC3S1600E, sus principales características son: 33,192 celdas lógicas, 648 kbits en bloques de RAM, 36 multiplicadores y un cristal de reloj de 48 MHz. El integrado se encuentra montado en una plataforma para sistemas embebidos llamada DUA1, orientada al control de servosistemas, implementación de IMU's y sistemas de control distribuido. El costo del dispositivo es casi el doble al procesador ARM.

El otro dispositivo es un procesador ARM Cortex-R5F el nombre del integrado es TMS570LC43357, que contiene: 4MB de Flash, 512 kB de SRAM, 128kB de datos en la Flash, cuenta con las interfaces seriales más comunes, por ejemplo, comunicación SPI, I2C, CAN, UART, etc., su frecuencia de operación es de 300 MHz utilizando el PLL (Phase Locked Loop). El integrado se encuentra montado en una plataforma de desarrollo de la familia de Texas Instruments (TI) llamado Hercules TMS570LC43x LaunchPad.

4. Resultados y discusión

En esta sección veremos la comparación de los resultados obtenidos en la implementación en el FPGA y Procesador ARM con los cálculos en MATLAB, mostrándolo en gráfica, de misma forma se mostrarán en tablas los resultados de tiempo de ejecución total y por bloque de cada dispositivo, y los recursos o memoria que se necesitaron para las operaciones del módulo/función.

En la Fig. 4, se puede observar el resultado de los errores máximo obtenidos en el FPGA y Procesador ARM con una resolución de 2^-14, comparando con los resultados calculados en MATLAB. Se realizaron pruebas de 20 ventanas de la FFT con diferentes datos, donde las barras de color azul muestran el error máximo del FPGA que tiene una variación de 1^-4, pero como se ha mencionado, en el procedimiento el resultado está en una resolución de 2^-14 punto fijo. En el procesador ARM el error obtenido es el doble de pequeño que en el FPGA con una variación de 1.25^-8 en las ventanas de la FFT obtenidas, es debido a que el resultado de las operaciones es de punto flotante de 32 bits, a pesar de que los valores de entrada tienen una resolución de 14 bits. El error que se tiene de cada dispositivo con los resultados del software MATLAB se puede notar ya que el software maneja los valores en DOUBLES (64 bits del ordenador), obteniendo estos resultados en la comparación dando como perdida de bits en el FPGA aproximadamente de uno en el bit menos significativo.

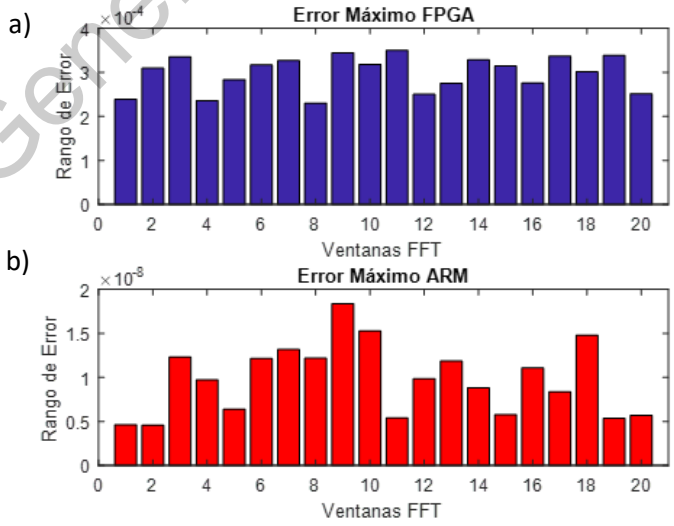


Figura 4. Gráfica de errores máximos en a) FPGA y b) procesador ARM.

En la Fig. 5, se obtuvieron los errores promedio de los 2048 datos de cada ventana del resultado de la FFT, donde el error promedio comparando con el error máximo obtenido en el procesador ARM, es pequeño por la

cantidad de bits del punto flotante a comparación con el FPGA y la variación del promedio de cada ventana se puede notar.

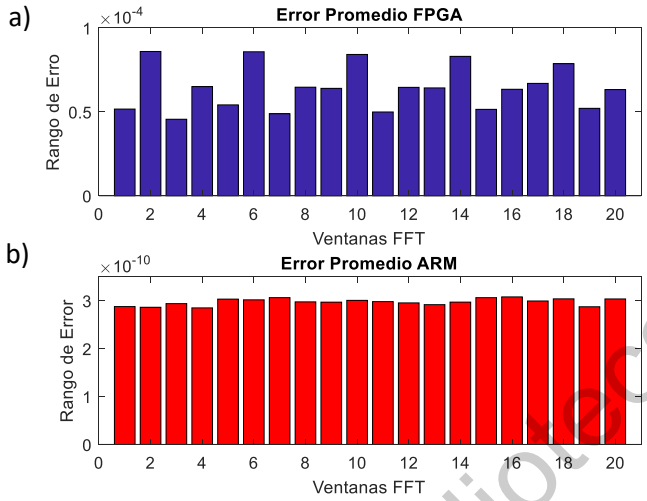


Figura 5. Gráfica de errores promedio en a) FPGA y b) procesador ARM.

En la Fig. 6, el THD obtenido por cada ventana de los diferentes datos de la FFT, mostradas en las figuras anteriores realizadas en los dispositivos, la comparación con los resultados obtenidos en MATLAB dando rangos más visibles del error tanto en el FPGA como en el Procesador ARM, debido a la acumulación del error por la sumatoria del bloque THD.

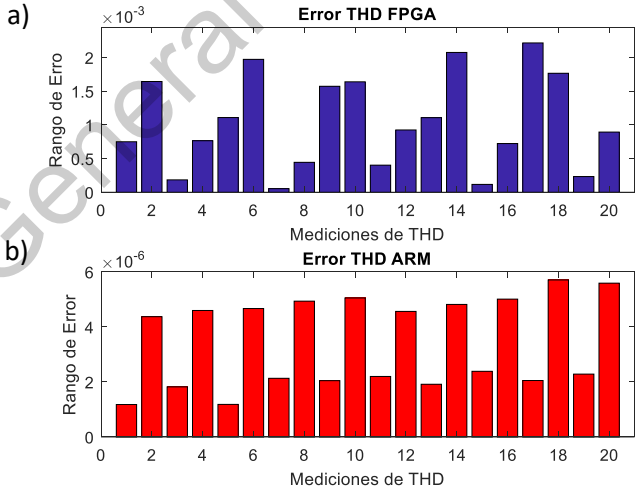


Figura 6. Gráfica de error THD en a) FPGA y b) procesador ARM.

En la Tabla 1, se muestra el resultado promedio de la gráfica de la Fig. 5.

Tabla 1. Valor promedio THD.

DISPOSITIVO	ERROR PROMEDIO
FPGA	0.001024120404628
Procesador ARM	3.425494784576721e-06

El procesador ARM otorga resultados cercanos a los esperados en MATLAB, debido por la unidad de punto flotante que cuenta es de 32 bits y MATLAB cuenta con valores dobles de 64 bits. El FPGA con operaciones de punto fijo de 16 bits mantiene un error a considerarse aun pequeño. En la Tabla 2, se tomaron mediciones de los tiempos que dura cada bloque de la arquitectura del módulo/función Fig. 2, los valores se capturaron con el uso de un osciloscopio digital, tomando como referencia un pin de entrada/salida del FPGA y Procesador ARM, donde el cambio de alto (3.3v) a bajo (GND), el tiempo que permanezca el pin en bajo es el tiempo que tarda en ejecutar el bloque el dispositivo. En el caso del procesador ARM se juntaron los bloques de valor absoluto con el THD para si el resultado obtenido directamente se realice el cálculo para el THD.

Tabla 2. Tiempo de cómputo en los bloques.

DISPOSITIVO	TIEMPO DE OPERACIÓN EN BLOQUES (MILISEGUNDOS)				TIEMPO TOTAL
	IIR	FFT	abs	THD	
FPGA	1.44	1.02	0.84	0.02	3.32
Procesador ARM	5.5	13.4	1.4		20.3

El FPGA el tiempo que tarda en desarrolla las operaciones o instrucciones del módulo/función es aproximadamente 6 veces más rápido que el procesador ARM, considerando que el dispositivo FPGA trabaja a una frecuencia de 48 MHz mientras que el procesador ARM la frecuencia de operación es de 300 MHz, es debido a que las intrucciones o tareas realizadas por el FPGA son concurrentes, y las operaciones por software como lo es el Procesador ARM son secuenciales. En el FPGA donde las instrucciones y operaciones concurrentes se realizan en hardware.

Los recursos utilizados en el FPGA, fue posible obtenerlos de la información proporcionada por la herramienta ISE de Xilinx, la capacidad de memoria consumida en el procesador ARM se obtuvo del compilador Code Composer Studios de Texas Instruments. Los recursos y memoria utilizada se muestran en la Tabla 3, cabe destacar que gran parte de la memoria y recursos consumidos en los dos dispositivos es debido a la memoria ROM de los datos capturados por el PQUAQ, y la memoria RAM, necesaria para almacenar los datos y así poder realizar las comparaciones.

Tabla 3. Recursos FPGA y memoria en procesador ARM.

RESUMEN DE UTILIZACIÓN DEL DISPOSITIVO FPGA (VALORES ESTIMADOS)			
Utilización Lógica	Usadas	Disponibles	Porcentaje utilizado
Número de Slices	4,967	14,752	33%
Número de Slices Flip Flops	1,063	29,504	3%
Número de LUTs de 4 entradas	9,280	29,504	31%
Número de BRAMs	12	36	33%
Número de MULT18x18SIOs	14	36	38%
LOCALIZACIÓN DE MEMORIA (BYTES) PROCESADOR ARM			
Memoria	Cantidad Usada	Cantidad Disponible	Porcentaje utilizado
FLASH0	44,988	2,097 k	2%
FLASH1	0	2,097 k	0%
STACKS	0	38,144	0%
RAM	49,352	486 k	10%

Los MULT18x18SIOs son multiplicadores de 18 bits, para el bloque de filtro digital se utilizó 1 multiplicador, en la FFT 4 multiplicadores, abs 4 multiplicadores y en THD 5. Los multiplicadores utilizados en el bloque abs y THD es debido a que el ancho de la palabra es de 19 bits, por lo cual se necesita un multiplicador extra para poder realizar la operación indicada.

5. Conclusiones

Se logró demostrar los rangos de tiempo de ejecución que tarda cada dispositivo, en realizar las operaciones de cada bloque del módulo/función. Esta información es útil ya que nos da una idea aproximada de los requerimientos de recursos o memoria que se necesitan en el dispositivo, ya sea un FPGA o un Procesador ARM para la implementación de un instrumento PQ. Los tiempos de ejecución tanto en el FPGA y Procesador ARM, están dentro del rango, evitando perdidas de actualizaciones de datos en un sistema de monitoreo continuo.

El manejo de los datos en punto fijo en el FPGA, su resultado en el error manteniendo la resolución es aproximadamente de 1 bit, el cual puede ser menor aumentando la resolución en las operaciones de cada bloque

y el resultado en la salida del FPGA, lo que conlleva a consumir más recursos del dispositivo haciéndolo de igual manera más costoso. Los resultados que se obtuvieron del Procesador ARM, es menor tomando en cuenta su unidad de punto flotante de 32 bits, pero el tiempo de ejecución que tarda en realizar las operaciones es mayor, a pesar que la frecuencia de operación con la que se trabajo es máxima frecuencia que puede operar el dispositivo, y si se requiere disminuir el tiempo de ejecución se necesita un Procesador ARM con una frecuencia mayor a lo cual su costo aumentara. En cambio, el FPGA su frecuencia de operación a pesar de ser 6 veces menor al procesador ARM, los cálculos y procesos son realizados a un tiempo aproximadamente 6 veces más rápido, y en cambio al Procesador ARM, la frecuencia de operación puede ser aumentada en el FPGA. El costo del FPGA es aproximadamente el doble al precio del procesador ARM. Por lo tanto, si se requiere un instrumento de PQ precisión, económico, pero con tiempo de ejecución de cada operación no sobrepase a lo necesitado se puede utilizar el procesador ARM con una frecuencia de operación mayor a 100 MHz, por otra parte, si se requiere mayor velocidad en el tiempo de ejecución de las operaciones, con una frecuencia de operación no muy elevada, pero teniendo limitado el margen de error al considerar los resultados, el FPGA es la mejor opción.

Agradecimientos

Agradecemos al Consejo Nacional de Ciencia y Tecnología por su apoyo durante el tiempo que se llevó a cabo para esta investigación.

Fuentes de financiamiento

Esta investigación fue parcialmente financiada por el proyecto FOFIUAQ-FIN201819.

Referencias

[1] Electromagnetic Compatibility - Testing and Measurement Techniques- General Guide on Harmonics and Interharmonics Measurements and Instrumentation, for Power Supply Systems and Equipment Connected thereto, International standard IEC 61000-4-7, 2002.

[2] IEEE Std 1159, *IEEE Standard 1159-2009: IEEE Recommended Practice for Monitoring Electric Power Quality*, vol. 2009, no. June. 2009.

[3] R. Peña, “Diseño y Construccion de un prototipo de medicion de los indicadores de calidad del servicio de energía eléctrica (DES y FES) para usuario residencial,” *Univ. Dist. Fr. Jose Caldas Bogota D.C. Colomb.*, vol. I Volumen, no. Medicion de los Indicadores de Calidad, p. 6, 2011.

[4] L. A. João, J. Batista, and M. J. Sepúlveda, “Sistema Digital de Bajo Coste para la Monitorización de la Calidad de Energía Eléctrica Low-Cost Digital System for Power Quality Monitoring,” vol. 18, pp. 15–23, 2007.

[5] J. Balcells, “Calidad de la red eléctrica:¿ como medirla?,” *Online http//www. jcee. upc. es/JCEE2001/PDFs*, pp. 22–26, 2001.

[6] A. E. Legarreta, J. H. Figueroa, and J. A. Bortolin, “An IEC 61000-4-30 class a - Power quality monitor: Development and performance analysis,” *Proceeding Int. Conf. Electr. Power Qual. Util. EPQU*, pp. 459–464, 2011.

[7] L. Ferrigno, C. Landi, and M. Laracca, “FPGA-based measurement instrument for power quality monitoring according to IEC standards,” *Conf. Rec. - IEEE Instrum. Meas. Technol. Conf.*, pp. 906–911, 2008.

[8] Electromagnetic Compatibility - Testing and Measurement Techniques - Power Quality Measurement Methods, International standard IEC 61000-4-30, 2003.

[9] L. Ferrigno and C. Landi, “A Study on the Feasibility and Effectiveness of Digital Filters Approach for Power Quality Monitoring in Compliance with IEC 61000-4-7,” 2007.

7.2. Código en C

```
#include "HL_sys_common.h"
#include "math.h"
#include "HL_system.h"
#include "HL_gio.h"
#include "HL_het.h"

//FFT//////////////////////
#define Hz 60
#define Fs 8000.0
#define HM 26
#define PI 3.14159265358979323846
#define DATOS 8192
#define DAT DATOS-1
#define S 13
#define DT DATOS>>1
#define N 1.0/(DT)
#define H DATOS*1.0/Fs
//Filtro IIR Butterworth
#define A1   1.187600680175614
#define A2   1.305213349288550
#define A3   0.674327525297998
#define A4   0.263469348280138
#define A5   0.051753033879641
#define A6   0.005022526595088

#define B0   0.070115413492454
#define B1   0.420692480954721
#define B2   1.051731202386804
#define B3   1.402308269849071
#define B4   1.051731202386804
#define B5   0.420692480954721
#define B6   0.070115413492454

////////////////////////////////
#define Q     1.0/sqrt(3.0)
////////////////////////////////
#define limp 819
#define limn -819

#define canales 6
#define canal 4
#define cruze canal-1
#define mem  canal-2

#define sag_i (0.1*(1<<14))/sqrt(2.0)
#define sag_s (0.9*(1<<14))/sqrt(2.0)
#define swell_i (1.1*(1<<14))/sqrt(2.0)
#define swell_s1 (1.8*(1<<14))/sqrt(2.0)
#define swell_s2 (1.4*(1<<14))/sqrt(2.0)
```

```
#define swell_s3 (1.2*(1<<14))/sqrt(2.0)
#define MEM  (DT) - 1

uint8 ind=0,idx=0;
uint16 a=0,st=0,dfft=0,drms=0;
short mem_c[canales][DT]={0};
float rms[canales];
//float iir[DATOS/2]={0.0};
//float ab[canales][DATOS>>1]={0.0};
float ba[canales];
uint8 dist[3];
uint32 cct[3]={0};
bool std=0,f1=0,b=0;
uint8 cnt=1,c=1,rt=0,act[3]={0},ant[3]={0},ctg[3]={0};
uint64 sum_c[canales][2]={0};
int g[canales][2]={0};


static float swell_s[4];//={swell_s1,swell_s2,swell_s3};
void gioHighLevelInterrupt(void);
/* USER CODE END */

int main(void)
{
	static float cost[DATOS/2 +1],sint[DATOS/2 +1];
	static uint16 rev[DATOS];
	////IIR y FFT
	static uint16 thdv[43]={0};
	float thd[canales],dsq;
	float ab;
	float thds[6];
	uint8 ch,h;
	uint8 init_fft=0;
	uint16 et=1,pot=2,n,pw,pos=0,id,ps=0,dst=0;
	float x1[canales][6]={0},y[canales][7]={0},aw[2],bw[2];
	float fft_c[2*canales][DATOS]={0.0};

	swell_s[0]=swell_s1;
	swell_s[1]=swell_s2;
	swell_s[2]=swell_s3;
	swell_s[3]=swell_s3;
	h=0;
	for(st=1;st<=HM;st++){
		thdv[h]=(uint16)round(st*Hz*H);
		h++;
	}
	for(st=0;st<=DATOS>>1;st++){
		cost[st]=cos(-(2*PI*(1.0*st/DATOS)));
		if(cost[st]*cost[st]<1e-15)
			cost[st]=0.0;
		sint[st]=sin(-(2*PI*(1.0*st/DATOS)));
		if(sint[st]*sint[st]<1e-15)
			sint[st]=0.0;
```

```
	}

	for(st=0;st<DATOS;st++){
		a=st;
		for(n=0;n<=S;n++){
			if(a==0)
				break;
			rev[st]<<=1;
			rev[st]|=a%2;
			a>>=1;
		}
		rev[st]<<=S-n;
	}
	a=0;
	st=0;
	gioInit();
	hetInit();
	gioPORTA->DIR|=0x02;

	_enable_IRQ_interrupt_();
	gioPORTB->DOUT &= ~0x40;
	gioPORTA->DOUT &= ~0x02;

	while(1){
		gioPORTB->DOUT |= 0x40;
		//////////IIR
		switch(init_fft){
		case 0:
			if(dfft>0){

				h=pos<<1;
				ch=h+1;
				y[pos][6]=1.0*mem_c[pos][dst]*B0 +
B1*x1[pos][5]+B2*x1[pos][4]+B3*x1[pos][3]+B4*x1[pos][2]+B5*x1[pos][1]+B6*x1[pos]
[0]-A1*y[pos][5]-A2*y[pos][4]-A3*y[pos][3]-A4*y[pos][2]-A5*y[pos][1]-
A6*y[pos][0];

				x1[pos][0]=x1[pos][1];
				x1[pos][1]=x1[pos][2];
				x1[pos][2]=x1[pos][3];
				x1[pos][3]=x1[pos][4];
				x1[pos][4]=x1[pos][5];
				x1[pos][5]=1.0*mem_c[pos][dst];
				y[pos][0]=y[pos][1];
				y[pos][1]=y[pos][2];
				y[pos][2]=y[pos][3];
				y[pos][3]=y[pos][4];
				y[pos][4]=y[pos][5];
				y[pos][5]=y[pos][6];
				//iir[st]=y[0][6];
				fft_c[h][rev[st]]=y[pos][6];
				fft_c[ch][st]=0;
				pos++;
				if(pos==canales){
```

```
                                        pos=0;
                                        dfft--;
                                        st++;
                                        dst++;
                                        if(dst==DT)///////////////
                                              dst=0;
                                        if(st==DATOS){
                                              init_fft=1;
                                              n=0;
                                              ps=0;
                                              et=1;
                                              pot=2;
                                              ch=0;
                                              st=1;
                                              id=0;
                                              h=0;
                                        }
                                  }
                            }

                            break;
                      case 1://FFT
                            ps=n+id;
                            pw=(1<<(S-st))*n;
                            aw[0]=fft_c[h][ps]+(fft_c[h][ps+et]*cost[pw]-
fft_c[h+1][ps+et]*sint[pw]);

      aw[1]=fft_c[h+1][ps]+(fft_c[h][ps+et]*sint[pw]+fft_c[h+1][ps+et]*cost[pw]
);
                            bw[0]=fft_c[h][ps]-(fft_c[h][ps+et]*cost[pw]-
fft_c[h+1][ps+et]*sint[pw]);
                            bw[1]=fft_c[h+1][ps]-
(fft_c[h][ps+et]*sint[pw]+fft_c[h+1][ps+et]*cost[pw]);
                            fft_c[h][ps]=aw[0];
                            fft_c[h+1][ps]=aw[1];
                            fft_c[h][ps+et]=bw[0];
                            fft_c[h+1][ps+et]=bw[1];
                            n++;
                            if(n==et){
                                  n=0;
                                  id+=pot;
                                  if(id>=DATOS){
                                        id=0;
                                        pot<<=1;
                                        et<<=1;
                                        st++;
                                        if(st>S){
                                              et=1;
                                              pot=2;
                                              thds[ch]=0.0;
                                              ch++;
                                              h=ch<<1;
                                              st=1;
```

```
                                        if(ch==canales){
                                                ch=0;
                                                init_fft=2;
                                                st=0;
                                        }                                       }
                                }
                        }
                        break;
                case 2:
                        gioPORTA->DOUT |= 0x02;
                        h=ch<<1;
                        et=thdv[st];

        //ab[ch][st]=sqrt(f[h][st]*f[h][st]+f[h+1][st]*f[h+1][st])*N;

        ab=sqrt(fft_c[h][et]*fft_c[h][et]+fft_c[h+1][et]*fft_c[h+1][et]);
                        if(st==0)
                                ba[ch]=ab;
                        else
                                thds[ch]+=ab*ab;
                        st++;
                        if(st==HM){
                                gioPORTA->DOUT &= ~0x02;
                                st=0;
                                et=1;
                                thd[ch]=sqrt(thds[ch]/ba[ch]);
                                ch++;
                                if(ch==canales){
                                        thds[0]=fft_c[6][thdv[0]]-fft_c[8][thdv[0]]-
Q*(fft_c[7][thdv[0]]+fft_c[9][thdv[0]]-2*fft_c[11][thdv[0]]);
                                        thds[2]=fft_c[6][thdv[0]]-
fft_c[8][thdv[0]]+Q*(fft_c[7][thdv[0]]+fft_c[9][thdv[0]]-2*fft_c[11][thdv[0]]);
                                        thds[1]=fft_c[7][thdv[0]]-
fft_c[9][thdv[0]]+Q*(fft_c[6][thdv[0]]+fft_c[8][thdv[0]]-2*fft_c[10][thdv[0]]);
                                        thds[3]=fft_c[7][thdv[0]]-fft_c[9][thdv[0]]-
Q*(fft_c[6][thdv[0]]+fft_c[8][thdv[0]]-2*fft_c[10][thdv[0]]);

        dsq=sqrt((thds[0]*thds[0]+thds[1]*thds[1])/(thds[2]*thds[2]+thds[3]*thds[
3]));

                                        ch=0;
                                        init_fft=0;
                                }
                        }

                        break;
                default:
                        init_fft=0;
                        break;

                }
        }
}
```

```
/* USER CODE BEGIN (4) */
#pragma CODE_STATE(gioHighLevelInterrupt, 32)
#pragma INTERRUPT(gioHighLevelInterrupt, IRQ)

void gioHighLevelInterrupt(void)
{
	uint32 offset = gioREG->OFF1;
	mem_c[ind][a]=hetREG2->DIN;
	g[ind][1]=mem_c[ind][a];
	if(ind==cruze){
		if(g[ind][1]>=limp)
			std=0;
		else
			if(g[ind][1]<=limn)
				std=1;
	}
	if(std!=f1){
		b=1;
		c++;
		rt=cnt;
		cnt=0;
	}
	sum_c[ind][1]=sum_c[ind][0]+g[ind][0]*g[ind][0];
	sum_c[ind][0]=sum_c[ind][1];
	if(c==2){
		//gioPORTB->DOUT |= 0x01;
		rms[ind]=sqrt(sum_c[ind][1]*1.0/rt);
		if(ind>2){
			idx=ind-3;
			if(rms[ind]>=sag_i && rms[ind]<=sag_s)
				act[idx]=2;
			else
				if(rms[ind]>=swell_i && rms[ind]<=swell_s[ctg[idx]])
					act[ind]=3;
				else
					if(rms[ind]<sag_i)
						act[idx]=1;
					else
						act[idx]=0;
			if(act[idx]!=ant[idx])
				cct[idx]=0;
			else
				cct[idx]+=1;
			ant[idx]=act[idx];
			if(ant[idx]!=0){
				if(cct[idx]<=60)
					ctg[idx]=1;
				else
					if(cct[idx]<=360)
						ctg[idx]=2;
					else
```

```
                                        ctg[idx]=3;
                        }
                        else
                                ctg[idx]=0;
                        if(ant[idx]==1 && ctg[idx]==1)
                                dist[idx]=0;
                        else
                                dist[idx]=ctg[idx]*10+ant[idx];

                }
                //gioPORTB->DOUT &= ~0x01;
        }
        g[0][0]=g[0][1];
        g[1][0]=g[1][1];
        g[2][0]=g[2][1];
        g[3][0]=g[3][1];
        g[4][0]=g[4][1];
        g[5][0]=g[5][1];
        f1=std;
        if(b==1)
                sum_c[ind][0]=0;
        if(ind==mem){
                b=0;
                if(c==2){
                        c=1;
                }
        }
        if(ind==5){
                dfft++;
                ind=0;
                if(a==MEM)
                        a=0;
                else
                        a++;
                if (cnt>85){
                        c=2;
                        rt=cnt;
                        cnt=1;
                }
                cnt++;
        }else
                ind++;


}
```

7.3. Descripción en hardware

7.3.1. Módulo principal

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity Modulo_principal is
    Port (
                    clk : in  STD_LOGIC;
              rst : in  STD_LOGIC;
                    str : in std_logic;
                    stp : in std_logic;
                    int : out std_logic;
                    sw : in std_logic_vector(2 downto 0);
                    led : out std_logic_vector(7 downto 0);
                    ---ADS-------------------
                    data : out std_logic_vector(15 downto 0);
                    ADS_RESET : out STD_LOGIC;
                    ADS_CLK   : out STD_LOGIC;
                    ADS_START : out STD_LOGIC;
                    ADS_DRDY  : in  STD_LOGIC;
                    ADS_PWDN  : out STD_LOGIC;
                    ADS_CS    : out STD_LOGIC;
                    ADS_SCLK  : out STD_LOGIC;
                    ADS_DIN   : out STD_LOGIC;
                    ADS_DOUT  : in  STD_LOGIC;
                    ---SRAM-------------------
                    A : out  STD_LOGIC_VECTOR (17 downto 0);
          I_O : inout  STD_LOGIC_VECTOR (15 downto 0);
          BHE : out  STD_LOGIC;
          WE : out  STD_LOGIC;
          CE : out  STD_LOGIC;
          OE : out  STD_LOGIC;
          BLE : out  STD_LOGIC
                    );
end Modulo_principal;

architecture Behavioral of Modulo_principal is

component PQ_m is
      generic(
            ec : integer :=2;
            fc : integer :=14;
            fm : real :=8.0e3
      );
```

```
    Port (
                        clk : in  STD_LOGIC;
                  rst : in  STD_LOGIC;
                        str : in  STD_LOGIC;
                        stp : in  STD_LOGIC;
                        w : in std_logic;
                        data : in std_logic_vector(ec+fc-1 downto 0);
                        dst : out std_logic_vector(17 downto 0);
                        r : in std_logic;
                        rdy : out std_logic;
                        dout : out std_logic_vector(ec+fc-1 downto 0);
              A : out  STD_LOGIC_VECTOR (17 downto 0);
              I_O : inout  STD_LOGIC_VECTOR (15 downto 0);
              BHE : out  STD_LOGIC;
              WE : out  STD_LOGIC;
              CE : out  STD_LOGIC;
              OE : out  STD_LOGIC;
              BLE : out  STD_LOGIC
                        );
end component;

component CONFI_ADS
    Port (
                        clk : in  STD_LOGIC;
              rst : in  STD_LOGIC;
                        str : in std_logic;
                        run : in std_logic;
                        rw : out std_logic;
                        int : out std_logic;
                        dout : out std_logic_vector(15 downto 0);
                        ADS_RESET : out STD_LOGIC;
                        ADS_CLK   : out STD_LOGIC;
                        ADS_START : out STD_LOGIC;
                        ADS_DRDY  : in  STD_LOGIC;
                        ADS_PWDN  : out STD_LOGIC;
                        ADS_CS    : out STD_LOGIC;
                        ADS_SCLK  : out STD_LOGIC;
                        ADS_DIN   : out STD_LOGIC;
                        ADS_DOUT  : in  STD_LOGIC
                        );
end component;

signal dst : std_logic_vector(17 downto 0);
signal dout,dmem : std_logic_vector(15 downto 0);
signal qn,qp : std_logic_vector(1 downto 0);
signal w,r,run,rdy,str_m : std_logic;

begin
```

```
data<=dout;
r<='0';
run<=stp;

	process(sw,dst,dmem,rdy)
	begin
	led<=rdy & '0' & dst(5 downto 0);
		case sw is
			when "000"=>led<=rdy & '0' &  dst(17 downto 12);
			when "001"=>led<=rdy & '0' &  dst(11 downto 6);
			when "010"=>led<=rdy & '0' &  dst(5 downto 0);
			when "011"=>led<=rdy & '0' &  dst(5 downto 0);
			when "100"=>led<=rdy & '0' &  dst(5 downto 0);
			when "101"=>led<=rdy & '0' &  dst(5 downto 0);
			when "110"=>led<=dmem(7 downto 0);
			when "111"=>led<=dmem(15 downto 8);
			when others=>null;
		end case;
	end process;

	process(qn,str)
	begin
	qp<=qn;
	str_m<='0';
	case qn is
		when "00"=>
			if str='1' then
				qp<="01";
			end if;

		when "01"=>
			qp<="10";
			str_m<='1';

		when "10"=>
		when "11"=>
		when others=>null;
	end case;
	end process;

	qn<=(others=>'0') when rst='1' else qp when rising_edge(clk);

ADS : confi_ads port map(clk,rst,str,run,w,int,dout,
                                ADS_RESET,ADS_CLK,ADS_START,ADS_DRDY,
                          ADS_PWDN,ADS_CS,ADS_SCLK,ADS_DIN,ADS_DOUT);

PQ : pq_m generic map(2,14,8.0e3) port
map(clk,rst,str_m,'0',w,dout,dst,r,rdy,dmem,A,I_O,BHE,WE,CE,OE,BLE);
end Behavioral;
```

7.3.2. Submódulo PQ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;


entity PQ_m is
	generic(
		ec : integer :=2;
		fc : integer :=14;
		fm : real :=8.0e3
	);
    Port (
			clk : in  STD_LOGIC;
		rst : in  STD_LOGIC;
			str : in  STD_LOGIC;
			stp : in  STD_LOGIC;
			w : in std_logic;
			data : in std_logic_vector(ec+fc-1 downto 0);
			dst : out std_logic_vector(17 downto 0);
			r : in std_logic;
			rdy : out std_logic;
			tmp : out std_logic;
			dout : out std_logic_vector(ec+fc-1 downto 0);
		A : out  STD_LOGIC_VECTOR (17 downto 0);
		I_O : inout  STD_LOGIC_VECTOR (15 downto 0);
		BHE : out  STD_LOGIC;
		WE : out  STD_LOGIC;
		CE : out  STD_LOGIC;
		OE : out  STD_LOGIC;
		BLE : out  STD_LOGIC
			);
end PQ_m;

architecture Behavioral of PQ_m is

component rms_m
	generic(
		frs : real := 8.0e3;
		ec : integer :=2;
		fc : integer :=14
		);
    Port ( clk : in  STD_LOGIC;
		rst : in  STD_LOGIC;
			str : in   std_logic;
			x   : in std_logic_vector(ec+fc-1 downto 0);
			w   : in   std_logic;
```

```
                    rdy  :      out    std_logic;
                    y1   :      out    std_logic_vector(ec+fc-1 downto 0);
                    y2   :      out    std_logic_vector(ec+fc-1 downto 0);
                    y3   :      out    std_logic_vector(ec+fc-1 downto 0);
                    y4   :      out    std_logic_vector(ec+fc-1 downto 0);
                    y5   :      out    std_logic_vector(ec+fc-1 downto 0);
                    y6   :      out    std_logic_vector(ec+fc-1 downto 0);
                    dst1 :      out    std_logic_vector(5 downto 0);
                    dst2 :      out    std_logic_vector(5 downto 0);
                    dst3 :      out    std_logic_vector(5 downto 0)
                    );
end component;

component Fifo_r
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
                     w : in std_logic;
                     r : in std_logic;
                     nr : in std_logic_vector(2 downto 0);
                     din : in std_logic_vector(15 downto 0);
                     dout : out std_logic_vector(15 downto 0);
                     fl : out std_logic;
           A : out  STD_LOGIC_VECTOR (17 downto 0);
           I_O : inout  STD_LOGIC_VECTOR (15 downto 0);
           BHE : out  STD_LOGIC;
           WE : out  STD_LOGIC;
           CE : out  STD_LOGIC;
           OE : out  STD_LOGIC;
           BLE : out  STD_LOGIC
                     );
end component;

component THD_m
      generic(
            np : integer := 8192;
             ec     : integer := 2;
             fc     : integer := 14
      );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
                     str  : in   std_logic;
                     w    : in   std_logic;
                     din : in  std_logic_vector(ec+fc-1 downto 0);
                     load : out std_logic;
                     busy : out std_logic;
                     rdy  : out std_logic;
                     fre : out std_logic_vector(ec+fc-1 downto 0);
                     fim : out std_logic_vector(ec+fc-1 downto 0);
                     dout : out std_logic_vector(ec+fc-1 downto 0)
```

```
                    );
end component;

component dsq_m
       generic(
              ec : integer := 2;
              fc : integer :=14
       );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           w : in  STD_LOGIC;
           xr : in  STD_LOGIC_VECTOR (ec+fc-1 downto 0);
           xi : in  STD_LOGIC_VECTOR (ec+fc-1 downto 0);
                     dout : out STD_LOGIC_VECTOR (ec+fc-1 downto 0);
                     bsy : out std_logic;
                     z : out std_logic;
           rdy : out  STD_LOGIC
                     );
end component;

component fifo_in
       generic (n : integer := 8;
                           w : integer := 4);
        Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
                     WR : in  STD_LOGIC;
           RD : in  STD_LOGIC;
           FLUSH : in  STD_LOGIC;
           E : out  STD_LOGIC;
           Di : in  STD_LOGIC_VECTOR(n-1 downto 0);
           Do : out  STD_LOGIC_VECTOR(n-1 downto 0)
                     );
end component;

signal y1,y2,y3,y4,y5,y6 : std_logic_vector(15 downto 0);
signal dst1,dst2,dst3 : std_logic_vector(5 downto 0);
signal rdy_rms,str_rms : std_logic;

signal qn,qp : std_logic_vector(1 downto 0);

signal wr,fl,e,u,clr,ws,wg : std_logic;

signal din_thd,fre,fim,dout_thd : std_logic_vector(ec+fc-1 downto 0);
signal cp,cn : std_logic_vector(1 downto 0);
signal nr,nn : std_logic_vector(2 downto 0);
signal busy,str_thd,rdy_thd,load,w_thd,r_fifo : std_logic;

signal ft,rm,rw,ps_dsq,rg,w_fifo : std_logic;
```

```
signal dout_dsq,x : std_logic_vector(ec+fc-1 downto 0);
signal w_dsq,bsy,rdy_dsq,rdy1 : std_logic;


signal sw,sn : std_logic_vector(2 downto 0);
signal x1 : std_logic_vector(15 downto 0);

signal gs,tiempo,z : std_logic;

begin

	dst<=dst1 & dst2 & dst3;

	u<='1' when sw="111" else '0';
	sw<=(others=>'0') when rst='1' else sn when rising_edge(clk);
	sn<=(others=>'0') when wr='0' else std_logic_vector(unsigned(sw)+1);

	wr<='0' when rst='1' else (wr or rdy_rms) and (not u) when
rising_edge(clk);

	--gs<='0' when rst='1' else gs or rdy_rms when rising_edge(clk);

	process(sw,y1,y2,y3,y4,y5,y6,dst1,dst2,dst3,dout_thd,dout_dsq)
	begin
		case sw is
			when "111"=>x<=y1;
			when "001"=>x<=y2;
			when "010"=>x<=y3;
			when "011"=>x<=y4;
			when "100"=>x<=y5;
			when "101"=>x<=y6;
			when "000"=>x<=dout_thd;
			when "110"=>x<=dout_dsq;
			when others=>null;
		end case;
	end process;

	process(qn,str,rdy_dsq,stp)
	begin
	qp<=qn;
	str_rms<='0';
	str_thd<='0';
	clr<='0';
	case qn is
		when "00"=>
			if str='1' then
				qp<="01";
			end if;
			clr<='1';
```

```
	when "01"=>
		str_rms<='1';
		str_thd<='1';
		qp<="10";

	when "10"=>
		if stp='1' then
			qp<="11";
		end if;

	when "11"=>
		if rdy_dsq='1' then
			qp<="00";
		end if;
		clr<='1';

	when others=>null;

end case;
end process;
qn<=(others=>'0') when rst='1' else qp when rising_edge(clk);

ft<='1' when nr="101" else '0';
ps_dsq<='1' when nr>"010" else '0';
nr<=(others=>'0') when rst='1' else nn when rising_edge(clk) and
rdy_thd='1';
nn<=(others=>'0') when ft='1' else std_logic_vector(unsigned(nr)+1);

w_dsq<=ps_dsq and rdy_thd;

w_thd<=rg when rising_edge(clk);

w_fifo<=(not rg and w) or wg ;

ws<=w when rising_edge(clk);
rg<=r_fifo when rising_edge(clk);
wg<=w and rg when rising_edge(clk);
r_fifo<=(not (ws or w)) and rm;

process(cn,load,r_fifo,busy,fl)
begin
cp<=cn;
rm<='0';
	case cn is
		when "00"=>
			if fl='1' then--w=0
				cp<="01";
			end if;
```

```
                    when "01"=>
                           if r_fifo='1' then
                                  cp<="10";
                           end if;
                           rm<='1';

                    when "10"=>
                           if load='1' then
                                  cp<="11";
                           end if;

                    when "11"=>
                           if busy='0' then
                                  cp<="00";
                           end if;

                    when others=>null;
             end case;
      end process;
      cn<=(others=>'0') when rst='1' else cp when rising_edge(clk);

      tmp<=tiempo;
      tiempo<='0' when rst='1' else (tiempo or z) and (not rdy_dsq) when
rising_edge(clk);

dsq : dsq_m generic map(ec,fc) port
map(clk,rst,w_dsq,fre,fim,dout_dsq,bsy,z,rdy_dsq);
rms : rms_m generic map(fm,ec,fc) port
map(clk,rst,str_rms,data,w,rdy_rms,y1,y2,y3,y4,y5,y6,dst1,dst2,dst3);
fifo : fifo_r port
map(clk,rst,w_fifo,r_fifo,nr,data,din_thd,fl,A,I_O,bhe,we,ce,oe,ble);
THD : thd_m generic map(8192,ec,fc) port
map(clk,rst,str_thd,w_thd,din_thd,load,busy,rdy_thd,fre,fim,dout_thd);
fifoin :  FIFO_in generic map(ec+fc,10) port map(clk,rst,wr,r,clr,rdy1,x,dout);
rdy<=rdy_dsq;
end Behavioral;
```

7.3.3. Bloque RMS

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.MATH_REAL.all;

entity rms_m is
      generic(
```

```
              frs : real := 8.0e3;
              ec : integer :=2;
              fc : integer :=14
              );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
                    str : in    std_logic;
                     x    : in std_logic_vector(ec+fc-1 downto 0);
                     w   : in   std_logic;
                     rdy  :     out    std_logic;
                     y1   :     out    std_logic_vector(ec+fc-1 downto 0);
                     y2   :     out    std_logic_vector(ec+fc-1 downto 0);
                     y3   :     out    std_logic_vector(ec+fc-1 downto 0);
                     y4   :     out    std_logic_vector(ec+fc-1 downto 0);
                     y5   :     out    std_logic_vector(ec+fc-1 downto 0);
                     y6   :     out    std_logic_vector(ec+fc-1 downto 0);
                     dst1 :     out    std_logic_vector(5 downto 0);
                     dst2 :     out    std_logic_vector(5 downto 0);
                     dst3 :     out    std_logic_vector(5 downto 0)
                     );
end rms_m;

architecture Behavioral of rms_m is

component raiz_b
      generic(
            nbits : integer :=8;
            e : integer :=2;
            f : integer :=14
      );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           x : in  STD_LOGIC_VECTOR (nbits+2*(e+f)-1 downto 0);
           str : in  STD_LOGIC;
           rdy : out  STD_LOGIC;
           y : out  STD_LOGIC_VECTOR (e+f-1 downto 0)
                     );
end component;

component Div1
      generic(
            e : integer := 8;
            f : integer := 8
      );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
                      str : in std_logic;
           num : in  STD_LOGIC_VECTOR (e+f-1 downto 0);
           den : in  STD_LOGIC_VECTOR (e+f-1 downto 0);
```

```
           q : out  STD_LOGIC_VECTOR (e+f-1 downto 0);
                     rdy : out std_logic
                     );
end component;

component lim_cero
      generic(
             ec : integer := 2;
             fc : integer := 14
      );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           datos : in  STD_LOGIC_VECTOR (ec+fc-1 downto 0);
           x : out  STD_LOGIC);
end component;

constant hz50 : integer :=  integer(round(0.5*real(frs/50.0)));
constant bg : integer := integer(ceil(log2(real(hz50))));

constant divr : real := sqrt(2.0);
constant acy : signed :=
to_signed(integer(round(1.8*real(2.0**fc)/divr)),ec+fc-1);
constant acx : signed :=
to_signed(integer(round(1.4*real(2.0**fc)/divr)),ec+fc-1);
constant acz : signed :=
to_signed(integer(round(1.2*real(2.0**fc)/divr)),ec+fc-1);

constant linf_sag : signed :=
to_signed(integer(round(0.1*real(2.0**fc)/divr)),ec+fc);
constant linf_swell : signed :=
to_signed(integer(round(1.1*real(2.0**fc)/divr)),ec+fc);
constant l_int : signed :=
to_signed(integer(round(0.1*real(2.0**fc)/divr)),ec+fc);

constant lsup_sag : signed :=
to_signed(integer(round(0.9*real(2.0**fc)/divr)),ec+fc);

signal s,acn,d,mx,ac1,ac2,ac3,ac4,ac5,ac6,g1,g2,g3,g4,g5,g6,ks,yd,a :
std_logic_vector(bg+2*(ec+fc)-1 downto 0);
signal M : std_logic_vector(2*(ec+fc)-1 downto 0);
signal r1,r2,r3,r4,r5,r6,x1,x2,cr,yo : std_logic_vector(ec+fc-1 downto 0);
signal cnn,cnq : std_Logic_vector(6 downto 0);
signal rp,rp1,yp : std_logic_vector(5 downto 0);
signal csn,csq,crq,crn,qn,qp: std_Logic_vector(2 downto 0);
signal cs,xc,e,cx,e1,e2,e3,str_div,str_raiz,rdy_div,rdy_raiz,t,at,bps :
std_logic;
----------------------
signal cnt1,cnt2,cnt3,cnt4,cnt5,cnt6,ccn,cct: std_logic_vector(18 downto 0);
signal clr,inter : std_logic;
```

```
signal lsup_swell,lsup : std_logic_vector(ec+fc-2 downto 0);
signal dst,iec : std_logic_vector(5 downto 0);
signal categoria,act,ant,ant1,ant2,ant3,ant4,ant5,ant6 : std_logic_vector(2
downto 0);
signal ctg : std_logic_vector(1 downto 0);

signal crn1,crq1 : std_logic_vector(1 downto 0);
signal e4,t1 : std_logic;

begin

      rp <= (others=>'0') when rst='1'  else rp(4 downto 0)&(e1 or str) when
rising_edge(clk) and (w='1' or str='1');
      cr <= (others=>'0') when rst='1' else x when rising_edge(clk) and
(rp(3)='1' and w='1');
      --cr <= (others=>'0') when rst='1' else x when rising_edge(clk) and
rp(3)='1';
      --rp <= rp1 when w='1' else (others=>'0');
      M<= std_logic_vector(signed(x1)*signed(x1));--4.28
      mx(2*(ec+fc)-1 downto 0)<=M;
      mx(2*(ec+fc)+bg-1 downto 2*(ec+fc))<=(others=>M(2*(ec+fc)-1));

      cruce : lim_cero generic map(ec,fc) port map(clk,rst,cr,e);
      xc<=(e or (e2 and rp(3)));
      ------------------------------
      r1 <= (others=>'0') when rst='1' else x when rising_edge(clk) and w='1';
      r2 <= (others=>'0') when rst='1' else r1 when rising_edge(clk) and w='1';
      r3 <= (others=>'0') when rst='1' else r2 when rising_edge(clk) and w='1';
      r4 <= (others=>'0') when rst='1' else r3 when rising_edge(clk) and w='1';
      r5 <= (others=>'0') when rst='1' else r4 when rising_edge(clk) and w='1';
      ------------------------
      x1 <= (others=>'0') when rst='1' else r5 when rising_edge(clk) and w='1';

      ---------------------------------------
      s<= std_logic_vector(unsigned(mx)+unsigned(d));
      d<= (others=>'0') when cx='1' else  acn;

      cx<='0' when rst='1' else (w and e1 and cs) or (cx and (not (w and e1)))
when rising_edge(clk);
      cs<='0' when rst='1' else (cs or xc)and(not rp(0)) when rising_edge(clk);
      at<=xc when rising_edge(clk);
      ac1<=(others=>'0') when rst='1' else s when rising_edge(clk) and
(rp(0)='1' and w='1');
      ac2<=(others=>'0') when rst='1' else s when rising_edge(clk) and
(rp(1)='1' and w='1');
      ac3<=(others=>'0') when rst='1' else s when rising_edge(clk) and
(rp(2)='1' and w='1');
      ac4<=(others=>'0') when rst='1' else s when rising_edge(clk) and
(rp(3)='1' and w='1');---
```

```
	ac5<=(others=>'0') when rst='1' else s when rising_edge(clk) and
(rp(4)='1' and w='1');
	ac6<=(others=>'0') when rst='1' else s when rising_edge(clk) and
(rp(5)='1' and (w='1' or bps='1'));

	bps<=(w and cs ) when rising_edge(clk);
	g1<=ac1 when rising_edge(clk) and cs='1'; g2<=ac2 when rising_edge(clk)
and cs='1';
	g3<=ac3 when rising_edge(clk) and cs='1'; g4<=ac4 when rising_edge(clk)
and cs='1';
	g5<=ac5 when rising_edge(clk) and cs='1'; g6<=ac6 when rising_edge(clk)
and cs='1';

	process(csn,ac1,ac2,ac3,ac4,ac5,ac6)
	begin
	case csn is
		when "000"=>
			acn<=ac1;
		when "001"=>
			acn<=ac2;
		when "010"=>
			acn<=ac3;
		when "011"=>
			acn<=ac4;
		when "100"=>
			acn<=ac5;
		when "101"=>
			acn<=ac6;
		when "110"=>
			acn<=ac1;
		when others=>
			acn<=ac1;
	end case;
	end process;
	-------------------------------------

	e1<='1' when csn="101" else '0';
	csn<=(others=>'0') when rst='1' else csq when rising_edge(clk) and w='1';
	csq<=(others=>'0') when e1='1' else std_logic_vector(unsigned(csn)+1);

	-----------------------------------------------
	--ex
	e2<='1' when cnn=std_logic_vector(to_unsigned(hz50+5,bg)) else '0';

	cnn<=(others=>'0') when rst='1' else cnq when rising_edge(clk) and
((rp(5)='1' and w='1') or xc='1');--e1
	cnq<=(others=>'0') when xc = '1' else std_logic_vector(unsigned(cnn)+1);
	-----------------------------------------------------------------------
	process(qn,e3,rdy_div,cs,rdy_raiz,str,at)
```

```
begin
qp<=qn;
t<='0';
str_div<='0';
str_raiz<='0';
rdy<='0';
case qn is
	when "000"=>
		if str='1' then
			qp<="001";
		end if;

	when "001"=>
		if at='1' then
			qp<="010";
		end if;

	when "010"=>
		str_div<='1';
		qp<="011";

	when "011"=>
		if rdy_div='1' then
			qp<="100";
		end if;

	when "100"=>
		str_raiz<='1';
		qp<="101";

	when "101"=>
		if rdy_raiz='1' then
			qp<="110";
		end if;

	when "110"=>
		if e3='1' then
			qp<="111";
		else
			qp<="010";
		end if ;
		t<='1';

	when "111"=>
		rdy<='1';
		qp<="001";

	when others=>null;
end case;
```

```
end process;

qn<=(others=>'0') when rst='1' else qp when rising_edge(clk);
-------------------------------------------------------------------
process(crn,g1,g2,g3,g4,g5,g6)
begin
	case crn is
		when "000"=>
			a<=g1;
		when "001"=>
			a<=g2;
		when "010"=>
			a<=g3;
		when "011"=>
			a<=g4;
		when "100"=>
			a<=g5;
		when "101"=>
			a<=g6;
		when "110"=>
			a<=g6;
		when "111"=>
			a<=g6;
		when others=>null;
	end case;
end process;

process(crn1,cnt1,cnt2,cnt3,cnt4,cnt5,cnt6,ant1,ant2,ant3,ant4,ant5,ant6)
begin
	case crn1 is
		when "00"=>
			ccn<=cnt1;
			ant<=ant1;
		when "01"=>
			ccn<=cnt2;
			ant<=ant2;
		when "10"=>
			ccn<=cnt3;
			ant<=ant3;
		when "11"=>
			ccn<=cnt3;
			ant<=ant3;
		when others=>null;
	end case;
end process;

e3<= '1' when crn="101" else '0';
e4<='1' when crn>"010" else '0';
crn<=(others=>'0') when rst='1' else crq when rising_edge(clk) and t='1';
```

```
	crq<=(others=>'0') when e3='1' else std_logic_vector(unsigned(crn)+1);

	t1<=t and e4;
	crn1<=(others=>'0') when rst='1' else crq1 when rising_edge(clk) and
t1='1';
	crq1<=(others=>'0') when e3='1' else std_logic_vector(unsigned(crn1)+1);

	--División
	ks(2*(ec+fc)-1 downto 0)<=(others=>'0');
	ks(bg+2*(ec+fc)-1 downto 2*(ec+fc))<=cnn when rising_edge(clk) and
xc='1';
	div : div1 generic map(bg+2*ec,2*fc) port
map(clk,rst,str_div,a,ks,yd,rdy_div);

	---raiz
	raiz : raiz_b generic map(bg,ec,fc) port
map(clk,rst,yd,str_raiz,rdy_raiz,yo);

	yp<=(others=>'0') when rst='1' else yp(4 downto 0)&(str or e3) when
rising_edge(clk) and (t='1' or str='1');

	-------------------------------------------------
	process(crn1,clk,t1,act,rst,cct)
	begin
	if rst='1' then
			cnt1<=(others=>'0');
			cnt2<=(others=>'0');
			cnt3<=(others=>'0');
			ant1<=(others=>'0');
			ant2<=(others=>'0');
			ant3<=(others=>'0');
	elsif rising_edge(clk) and t1='1' then
		case crn1 is
			when "00"=>
				cnt1<=cct;
				ant1<=act;
			when "01"=>
				cnt2<=cct;
				ant2<=act;
			when "10"=>
				cnt3<=cct;
				ant3<=act;
			when "11"=>
				cnt1<=cct;
				ant1<=act;
			when others=>null;
			end case;
	end if;
	end process;
```

```
	process(yo,clk,ant,lsup,inter)
	begin
		if rising_edge(clk) then
		if yo < std_logic_vector(l_int) or yo(ec+fc-1)='1' then
			act<="001";
		elsif yo >= std_logic_vector(linf_sag) and yo<=
std_logic_vector(lsup_sag) then
			act<="010";
		elsif yo>= std_logic_vector(linf_swell) and yo <=
std_logic_vector('0'&lsup) then
			act<="100";
		else
			act<="000";
		end if;
		end if;
	end process;

	process(ctg)
	begin
	case ctg is
		when "00"=>
			categoria<="000";
			lsup<= std_logic_vector(acx);
		when "01"=>
			lsup<= std_logic_vector(acx);
			categoria<="001";
		when "10"=>
			lsup<= std_logic_vector(acy);
			categoria<="010";
		when "11"=>
			lsup<= std_logic_vector(acz);
			categoria<="100";
			when others=>null;
	end case;
	end process;

	process(ccn,act)
	begin
	if act>"000" then
			if ccn <= ("000" & x"003c") then
				ctg<="01";
			elsif ccn <= ("000" & x"0168") then
				ctg<="10";
			else
				ctg<="11";
			end if;
	else
```

```
            ctg<="00";
      end if;
      end process;

      IEC<="000000" when (categoria(0)='1' and act(0)='1') else categoria &
act;
      dst<=(others=>'0') when rst='1' else IEC when rising_edge(clk);
      cct<= (others=>'0') when clr='1' else std_logic_vector(unsigned(ccn)+1);
      clr <= '0' when act=ant else '1';
      ----------------------------------------------------
      y1<=yo when rising_edge(clk) and (yp(0)='1' and t='1');y2<=yo when
rising_edge(clk) and (yp(1)='1' and t='1');
      y3<=yo when rising_edge(clk) and (yp(2)='1' and t='1');y4<=yo when
rising_edge(clk) and (yp(3)='1' and t='1');
      y5<=yo when rising_edge(clk) and (yp(4)='1' and t='1');y6<=yo when
rising_edge(clk) and (yp(5)='1' and t='1');

      dst1<=dst when rising_edge(clk) and (yp(3)='1' and t1='1');
      dst2<=dst when rising_edge(clk) and (yp(4)='1' and t1='1');
      dst3<=dst when rising_edge(clk) and (yp(5)='1' and t1='1');
end Behavioral;
```

7.3.4. Bloque THD

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.MATH_REAL.all;

entity THD_m is
      generic(
            np : integer := 8192;
            ec     : integer := 2;
            fc     : integer := 14
      );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
                     str  : in   std_logic;
                     w    : in   std_logic;
                     din : in  std_logic_vector(ec+fc-1 downto 0);
                     load : out std_logic;
                     busy : out std_logic;
                     rdy  : out std_logic;
                     fre : out std_logic_vector(ec+fc-1 downto 0);
                     fim : out std_logic_vector(ec+fc-1 downto 0);
                     dout : out std_logic_vector(ec+fc-1 downto 0)
```

```
                    );
end THD_m;

architecture Behavioral of THD_m is

component fftFFTr2dif
       generic(np : integer := 256;
                   e : integer := 2;
                   f : integer := 10;
                   ifft : boolean := false);
       port(
             RST : in STD_LOGIC;
             CLK : in STD_LOGIC;
             STR : in STD_LOGIC;
             RDY : out STD_LOGIC;
             BUSY : out STD_LOGIC;
             FR : out STD_LOGIC;
             FW : out STD_LOGIC;
             RD : in STD_LOGIC;
             WR : in STD_LOGIC;--Leer y escribr de memoria
             CLR : in STD_LOGIC;--Vaciar FIFOs
             DiRe : in STD_LOGIC_VECTOR(e+f-1 downto 0);
             DiIm : in STD_LOGIC_VECTOR(e+f-1 downto 0);
             DoRe : out STD_LOGIC_VECTOR(e+f-1 downto 0);
             DoIm : out STD_LOGIC_VECTOR(e+f-1 downto 0)
          );
end component;

component iirIIR
      generic(
      ex : integer := 16;
      fx : integer := 0;
      ec : integer := 2;
      fc : integer := 14
      );
      port(
      rst : in  std_logic;
      clk : in  std_logic;
      clr : in  std_logic;
      str : in  std_logic;
      rdy : out std_logic;
      x   : in  std_logic_vector(ex+fx-1 downto 0);
      a0, a1, a2, a3, a4, a5, a6 : in  std_logic_vector(ec+fc-1 downto 0);
      b1, b2, b3, b4, b5, b6     : in  std_logic_vector(ec+fc-1 downto 0);
      y   : out std_logic_vector(ex+fx-1 downto 0)
      );
end component;

component Div_f
```

```
      generic(
            e : integer := 8;
            f : integer := 8
      );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
                      str : in std_logic;
           num : in  STD_LOGIC_VECTOR (e+f-1 downto 0);
           den : in  STD_LOGIC_VECTOR (e+f-1 downto 0);
           q : out  STD_LOGIC_VECTOR (e+f-1 downto 0);
           --r : out  STD_LOGIC_VECTOR (e+f-1 downto 0);
                     rdy : out std_logic
                     );
end component;

component macMAC
      generic(
      ex : integer := 12;
      fx : integer := 0;
      ec : integer := 1;
      fc : integer := 15;
      g  : integer := 2
      );
      port(
      rst : in  std_logic;
      clk : in  std_logic;
      clr : in  std_logic;
      opa : in  std_logic_vector(1 downto 0);
      x   : in  std_logic_vector(ex+fx-1 downto 0);
      c   : in  std_logic_vector(ec+fc-1 downto 0);
      y   : out std_logic_vector(ex+fx-1 downto 0)
      );
end component;

component iirGCounter
      generic(
      n : integer := 4
      );
      port(
      rst : in  std_logic;
      clk : in  std_logic;
      opc : in  std_logic_vector(1 downto 0);
      k   : in  std_logic_vector(n-1 downto 0);
      q     : out std_logic_vector(n-1 downto 0);
      e   : out std_logic
      );
end component;

component iirFSM
```

```
	port(
	rst : in  std_logic;
	clk : in  std_logic;
	str : in  std_logic;
	equ : in  std_logic;
	ldr : out std_logic;
	opa : out std_logic_vector(1 downto 0);
	opc : out std_logic_vector(1 downto 0);
	rdy : out std_logic
	);
end component;

component macGLimiter
	generic(
	er : integer := 10;
	fr : integer := 16;
	ey : integer := 7;
	fy : integer := 11
	);
	port(
	RAW : in  std_logic_vector(er+fr-1 downto fr-fy-1);
	LIM : out std_logic_vector(ey+fy-1 downto 0)
	);
end component;


--THD
constant Hz : integer := 60;--frecuencia de la red electrica
constant Fms : real := 8000.0;--frecuencia de muestreo
constant Harmonic : natural := 26;-- Armónicos
constant nhc  : integer := integer(ceil(log2(real(Harmonic+1))));
constant nb  : integer := integer(log2(real(hz*(Harmonic+1)*np)/FMS));
constant nc  : integer := integer(real(np)/2.0)-1;

type lut_type is array (natural range <>) of std_logic_vector(nb-1 downto 0);

	-- Sine function
	function Harm_rom( n : natural ) return lut_type is
		variable lut : lut_type( 0 to n-1 );
		variable kH : integer;
	begin
		for i in 1 to lut'length loop
			kH := integer(round((real(i)*real(Hz)*real(np))/Fms));
			lut(i-1) := std_logic_vector(to_signed(kH-1,nb));--nb
		end loop;
		return lut;
	end function;

constant Hlut : lut_type(0 to Harmonic) := Harm_rom(harmonic+1);
```

```
signal idq,idn : std_logic_vector(nhc-1 downto 0);
signal x : std_logic_vector(2*(ec+fc)-1 downto 0);
signal a,s,d,hf,r_div : std_logic_vector(2*(ec+fc)-1 downto 0);
signal a0,a1,a2,a3,a4,a5,a6,b1,b2,b3,b4,b5,b6 : std_logic_vector(ec+fc-1 downto
0);
signal y, dore,doim,xo : std_logic_vector(ec + fc -1 downto 0);
signal mult1,mult2 : std_logic_vector(ec + fc-1 downto 0);
signal qn,qp : std_logic_vector(3 downto 0);
signal h : std_logic_vector(1 downto 0);
signal q,n,nH : std_logic_vector(nb-1 downto 0);----nb
signal str_raiz,l,cs,
str_fft,clr,fw,fr,rdy_fft,rdy_raiz,rdy_iir,busy_fft,bsy,rd,p,g,fnl,ty,yh,str_div
,rdy_div,ut : std_logic;
--raiz
signal yp,yn,yr,yq,yqx : std_logic_vector(ec+fc-1 downto 0);
--IIR
type std_logic_matrix is array(13 downto 0) of std_logic_vector(ec+fc-1 downto
0);
signal xp : std_logic_matrix;
signal ci : std_logic_vector(ec+fc-1 downto 0);
signal idx : std_logic_vector(3 downto 0);
signal xi, yi : std_logic_vector(ec+fc-1 downto 0);
signal e, ldr : std_logic;
signal opa, opc : std_logic_vector(1 downto 0);
signal xmul,xmulx, rmul, s_iir, acc_iir,qn_iir : std_logic_vector(2+2*(ec+fc)-1
downto 0);


begin

        nH <= Hlut(to_integer(unsigned(idq))) when rising_edge(CLK);
        ut<='1' when to_integer(unsigned(idq))=Harmonic else '0';
        idq<=(others=>'0') when rst='1' else idn when rising_edge(clk) and p='1';
        idn<=(others=>'0') when bsy='0' else std_logic_vector(unsigned(idq)+1);
        fnl<='1' when to_integer(unsigned(idq))= 0 else '0';
        g<='1' when q= nH else '0';
        q<=(others=>'0') when rst='1' else n when rising_edge(clk) and rd='1';
        n<=(others=>'0') when bsy='0' else std_logic_vector(unsigned(q)+1);

        ty<=(p and fnl);

        fim<=doim when rising_edge(clk) and ty='1';
        fre<=dore when rising_edge(clk) and ty='1';

        busy<=bsy or busy_fft;
        process(qn,str,rdy_fft,rdy_raiz,fw,fr,rdy_iir,fnl,g,rdy_div,ut,ty)
        begin
        bsy<='1';
```

```
qp<=qn;
clr<='0';
str_fft<='0';
str_div<='0';
str_raiz<='0';
rd<='0';
rdy<='0';
h<="00";
cs<='0';
l<='0';
p<='0';
       case qn is
              when "0000"=>
                     if str='1' then
                            qp<="0001";
                     end if;
                     clr<='1';
                     bsy<='0';

              when "0001"=>
                     if fw='1' and rdy_iir='1' then
                            qp<="0010";
                     end if;
                     bsy<='0';

              when "0010"=>
                     str_fft<='1';
                     qp<="0011";

              when "0011"=>
                     if rdy_fft='1' then
                            qp<="0100";
                     end if;

              when "0100"=>
                     h<="01";
                     cs<='1';
                     l<='1';
                     qp<="0101";

              when "0101"=>
                     if g='1' then
                            qp<="0110";
                     end if;
                     h<="01";
                     rd<='1';


              when "0110"=>
```

```
        h<="01";
        qp<="0111";

when "0111"=>
        h<="01";
        l<='1';
        qp<="1000";

when "1000"=>
        h<="10";
        l<='1';
        qp<="1001";

when "1001"=>
        if ty ='0' then
                qp<="1010";
        else
                qp<="0100";
        end if;
        p<='1';


when "1010"=>
        if ut='1' then
                qp<="1011";
        else
                qp<="0101";
        end if;

when "1011"=>
        str_div<='1';
        qp<="1100";

when "1100"=>
        if rdy_div='1' then
                qp<="1101";
        end if;
        h<="11";

when "1101"=>
        str_raiz<='1';
        h<="11";
        qp<="1110";

when "1110"=>
        if rdy_raiz='1' then
                qp<="1111";
        end if;
        h<="11";
```

```
                when "1111"=>
                        p<='1';
                        cs<='1';
                        l<='1';
                        rd<='1';
                        clr<='1';
                        qp<="0001";
                        bsy<='0';
                        rdy<='1';

                when others=>null;
        end case;
    end process;

    load<=(rdy_iir and (not bsy));

    qn<=(others=>'0') when rst='1' else qp when rising_edge(clk);

    ftt : fftfftr2dif generic map(np,ec,fc,false) port
map(rst,clk,str_fft,rdy_fft,busy_fft,fr,fw,rd,rdy_iir,clr,y,x"0000",dore,doim);
    div : div_f generic map(2*ec,2*fc) port
map(clk,rst,str_div,a,hf,r_div,rdy_div);

    process(h,doim,dore,yr,xi,ci)
    begin
        case h is
            when"00"=>
                mult1<= xi;
                mult2<= ci;
            when"01"=>
                mult1<=dore;--(ec+fc-2 downto 0) & '0';
                mult2<=dore;--(ec+fc-2 downto 0) & '0';
            when"10"=>
                mult1<=doim;--(ec+fc-2 downto 0) & '0';
                mult2<=doim;--(ec+fc-2 downto 0) & '0';
            when others=>
                mult1<= yr;
                mult2<= yr;
        end case;
    end process;
    x<=std_logic_vector(signed(mult1)*signed(mult2));

    s<= std_logic_vector(unsigned(x)+unsigned(a));
    a<= (others=>'0') when rst='1' else d when rising_edge(clk) and l='1';--
12.28--2
    d<= (others=>'0') when cs='1' else  s;
    ---------------------------
    dout<= xo when rising_edge(clk);
```

```
      hf<= a when rising_edge(clk) and ty='1';

      --raiz
      yp<=(others=>'0') when rst='1' else str_raiz & yp(ec+fc-1 downto 1) when
rising_edge(clk);
      yn<=(others=>'0') when rst='1' else yqx when rising_edge(clk);
      yqx<=(others=>'0') when str_raiz='1' else yq;
      yr<=yp or yn;
      yq<=yn when x>r_div else yr;
      rdy_raiz<=yp(0);
      xo<=yn;
      --------------------------------

      --IIR

      xp(0)<=din;

    xp(1) <= (others => '0') when (rst = '1' or clr = '1') else xp(0) when w =
'1' and rising_edge(clk);
      xp(2) <= (others => '0') when (rst = '1' or clr = '1') else xp(1) when w
= '1' and rising_edge(clk);
    xp(3) <= (others => '0') when (rst = '1' or clr = '1') else xp(2) when w =
'1' and rising_edge(clk);
      xp(4) <= (others => '0') when (rst = '1' or clr = '1') else xp(3) when w
= '1' and rising_edge(clk);
      ---
       xp(5) <= (others => '0') when (rst = '1' or clr = '1') else xp(4) when w
= '1' and rising_edge(clk);
      xp(6) <= (others => '0') when (rst = '1' or clr = '1') else xp(5) when w
= '1' and rising_edge(clk);
       xp(7) <= (others => '0') when (rst = '1' or clr = '1') else xp(6) when w
= '1' and rising_edge(clk);

      xp(8) <= (others => '0') when (rst = '1' or clr = '1') else Y    when w =
'1' and rising_edge(clk);
       xp(9) <= (others => '0') when (rst = '1' or clr = '1') else xp(8) when w
= '1' and rising_edge(clk);
      xp(10) <= (others => '0') when (rst = '1' or clr = '1') else xp(9) when w
= '1' and rising_edge(clk);
       xp(11) <= (others => '0') when (rst = '1' or clr = '1') else xp(10) when
w = '1' and rising_edge(clk);
      xp(12) <= (others => '0') when (rst = '1' or clr = '1') else xp(11) when
w = '1' and rising_edge(clk);
       xp(13) <= (others => '0') when (rst = '1' or clr = '1') else xp(12) when
w = '1' and rising_edge(clk);

       mux : process(idx,xp,a0,a1,a2,a3,a4,a5,a6,b1,b2,b3,b4,b5,b6)
      begin
            case idx is
```

```
                    when "0000"    => ci <= a0; xi <= xp(1);
                when "0001"    => ci <= a1; xi <= xp(2);
                    when "0010"    => ci <= a2; xi <= xp(3);
                    when "0011"    => ci <= a3; xi <= xp(4);--
                    when "0100"    => ci <= a4; xi <= xp(5);
                    when "0101"    => ci <= a5; xi <= xp(6);
                    when "0110"    => ci <= a6; xi <= xp(7);
                    when "0111"    => ci <= b1; xi <= xp(8);--
                    when "1000"    => ci <= b2; xi <= xp(9);
                    when "1001"    => ci <= b3; xi <= xp(10);
                    when "1010"    => ci <= b4; xi <= xp(11);
                    when "1011"    => ci <= b5; xi <= xp(12);
                    when "1100"    => ci <= b6; xi <= xp(13);
                    when others    => ci <= (others => '0'); xi <= (others =>
'0');
            end case;
      end process mux;

      xmul(2+2*(ec+fc)-1 downto 2*(ec+fc)) <= (others => x(2*(ec+fc)-1));
      xmul(2*(ec+fc)-1 downto 0) <= x;
      rmul <= (others => '0') when rst = '1' else xmulx when rising_edge(clk);
      xmulx<=(others=>'0') when clr='1' else xmul;
      s_iir <= std_logic_vector( unsigned(rmul) + unsigned(acc_iir) );
      process(acc_iir,opa,s_iir)
      begin
            case opa is
                  when "00"   => qn_iir <= acc_iir;
                  when "01"   => qn_iir <= s_iir;
                  when others => qn_iir <=(others => '0');
            end case;
      end process;

      acc_iir <= (others => '0') when rst = '1' else qn_iir when
rising_edge(clk);
      limit : macGLimiter generic map(2+fc+2*ec,fc,ec+fc,0) port
map(acc_iir(2+2*(ec+fc)-1 downto fc-1),yi);
      Count : iirGCounter generic map(4) port map(rst, clk, opc, "1100", idx,
e);
      FSM     : iirFSM            port map(rst, clk, w, e, ldr, opa, opc,
rdy_iir);

      y <= (others => '0') when (rst = '1' or clr='1') else yi when ldr = '1'
and rising_edge(clk);

      ------------------------

   a0 <= "0000010001111101"; --
   a1 <= "0001101011101101"; --
   a2 <= "0100001101010000"; --
```

```
   a3 <= "0101100110111111"; --
   a4 <= "0100001101010000"; --
   a5 <= "0001101011101101"; --
   a6 <= "0000010001111101"; --
   b1 <= "1011001111111110"; --
   b2 <= "1010110001110111"; --
   b3 <= "1101010011011000"; --
   b4 <= "1110111100100011"; --
   b5 <= "1111110010110000"; --
   b6 <= "1111111110101110"; --


end Behavioral;
```

7.3.5. Bloque DSQ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity dsq_m is
      generic(
            ec : integer := 2;
            fc : integer :=14
      );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           w : in  STD_LOGIC;
           xr : in  STD_LOGIC_VECTOR (ec+fc-1 downto 0);
           xi : in  STD_LOGIC_VECTOR (ec+fc-1 downto 0);
          dout : out STD_LOGIC_VECTOR (ec+fc-1 downto 0);
          bsy : out std_logic;
           rdy : out  STD_LOGIC
                 );
end dsq_m;

architecture Behavioral of dsq_m is

component macGLimiter
      generic(
      er : integer := 10;
      fr : integer := 16;
      ey : integer := 7;
      fy : integer := 11
      );
      port(
      RAW : in  std_logic_vector(er+fr-1 downto fr-fy-1);
```

```
      LIM : out std_logic_vector(ey+fy-1 downto 0)
      );
end component;

component Division1
      generic(
            e : integer := 8;
            f : integer := 8
      );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
                      str : in std_logic;
           num : in  STD_LOGIC_VECTOR (e+f-1 downto 0);
           den : in  STD_LOGIC_VECTOR (e+f-1 downto 0);
           q : out  STD_LOGIC_VECTOR (e+f-1 downto 0);
           --r : out  STD_LOGIC_VECTOR (e+f-1 downto 0);
                     rdy : out std_logic
                     );
end component;

component div_f
generic(
            e : integer := 2;
            f : integer := 14
      );
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
                      str : in std_logic;
           num : in  STD_LOGIC_VECTOR (e+f-1 downto 0);
           den : in  STD_LOGIC_VECTOR (e+f-1 downto 0);
           q : out  STD_LOGIC_VECTOR (e+f-1 downto 0);
                     rdy : out std_logic
                     );

end component;


signal mult : std_logic_vector(2*(ec+fc)+1 downto 0);
signal num,den,rx,ax,raux,q,rxa : std_logic_vector(2*(ec+fc)-1 downto 0);
signal a1,rs,mult1,mult2,r,r1,r2,r3,r4,ri,rxb :std_logic_vector(ec+fc downto 0);
signal yp,yn,ynx,yr,yq : std_logic_vector(ec+fc-1 downto 0);
signal b1,aux,aux1 : std_logic_vector(ec+fc-1 downto 0);
signal qn,qp : std_logic_vector(4 downto 0);
signal g : std_logic_vector(3 downto 0);
signal sw,mt,l : std_logic_vector(1 downto 0);
signal sr,t,srx,tx,sr1,k,str_d,rdy_d,str_r,rdy_r,clr : std_logic;


begin
```

```
process(sw,r1,r2,r3,r4)
begin
	case sw is
		when "00"=>
			a1<=r1;
		when "01"=>
			a1<=r4;
		when "10"=>
			a1<=r3;
		when others=>
			a1<=r2;
	end case;
end process;

process(qn,w,rdy_d,rdy_r)
begin
qp<=qn;
g<="0000";
sw<="00";
t<='1';
srx<='0';
sr<='0';
rdy<='0';
l<="00";
mt<="00";
tx<='0';
sr1<='0';
k<='0';
str_d<='0';
str_r<='0';
clr<='0';
bsy<='1';
	case qn is
		when "00000"=>
		 if w='1' then
			qp<="00001";
		 end if;
		 bsy<='0';

		when "00001"=>
			g<="0001";
			qp<="00010";

		when "00010"=>
			g<="0010";
			sw<="01";
			sr<='1';
			qp<="00011";
```

```
when "00011"=>
       t<='0';
       g<="0100";
       sw<="10";
       qp<="00100";

when "00100"=>
       t<='0';
       g<="1000";
       sw<="11";
       sr<='1';
       qp<="00101";

when "00101"=>
       if w='1' then
              qp<="00110";
       end if;
bsy<='0';
when "00110"=>
       sr<='1';
       g<="0001";
       qp<="00111";

when "00111"=>
       sr<='1';
       sw<="01";
       g<="0010";
       qp<="01000";

when "01000"=>
       sr<='1';
       t<='0';
       sw<="10";
       g<="0100";
       qp<="01001";

when "01001"=>
       sr<='1';
       t<='0';
       sw<="11";
       g<="1000";
       qp<="01010";

when "01010"=>
       if w='1' then
              qp<="01011";
       end if;
       sw<="01";
```

```
bsy<='0';

when "01011"=>
       sw<="01";
       g<="0010";
       qp<="01100";

when "01100"=>----
       sw<="01";
       k<='1';
       g<="0010";
       qp<="01101";

when "01101"=>
       t<='0';
       mt<="01";
       sr1<='1';
       l<="01";--------num
       sw<="11";
       g<="1000";
       qp<="01110";

when "01110"=>--den
       mt<="01";
       l<="10";
       tx<='1';
       t<='0';
       sw<="11";
       --g<="1000";
       qp<="01111";

when "01111"=>
       sw<="11";
       t<='0';
       k<='1';
       qp<="10000";

when "10000"=>--num
       mt<="01";
       srx<='1';
       l<="01";
       qp<="10001";

when "10001"=>--den
       mt<="01";
       srx<='1';
       sr1<='1';
       tx<='1';
       l<="10";
```

```
                        qp<="10010";

                    when "10010"=>
                        str_d<='1';
                        qp<="10011";

                    when "10011"=>
                        if rdy_d='1' then
                            qp<="10100";
                        end if;
                        mt<="10";

                    when "10100"=>
                        str_r<='1';
                        mt<="10";
                        qp<="10101";

                    when "10101"=>
                        if rdy_r='1' then
                            qp<="10110";
                        end if;
                        mt<="10";

                    when "10110"=>
                        qp<="10111";
                        g<="1111";
                        l<="11";
                        clr<='1';

                    when "10111"=>
                        rdy<='1';
                        --clr<='1';
                        qp<="00000";
                    when others=>null;
                end case;
        end process;

        qn<=(others=>'0') when rst='1' else qp when rising_edge(clk);

        b1<= xr when t='1' else xi;

        r<=std_logic_vector(unsigned(a1)-unsigned(b1(ec+fc-1) & b1)) when sr='1'
else std_logic_vector(unsigned(a1)+unsigned(b1(ec+fc-1) & b1));
        mult<=std_logic_vector(signed(mult1)*signed(mult2));

        process(mt,yr,rs,r)
        begin
            case mt is
                when "00"=>
```

```
                        mult1<= r;
                        mult2<='0' & x"24f3";
                  when "01"=>
                        mult1<=rs;
                        mult2<=rs;
                  when "10"=>
                        mult1<='0' & yr;
                        mult2<='0' & yr;
                  when others=>
                        mult1<='0' & yr;
                        mult2<='0' & yr;
            end case;
      end process;


      rx<=std_logic_vector(unsigned(ax)+unsigned(mult(2*(ec+fc)-1 downto 0)));
      rs<=std_logic_vector(unsigned(ri)-unsigned(aux1(ec+fc-1) & aux1)) when
sr1='1' else std_logic_vector(unsigned(ri)+unsigned(aux1(ec+fc-1) & aux1));
      ri<=r1 when srx='1' else r3;
      ax<= den when tx='1' else num;

      num<=(others=>'0') when rst='1' else rxa when rising_edge(clk) and
l(0)='1';
      den<=(others=>'0') when rst='1' else rxa when rising_edge(clk) and
l(1)='1';
      rxa<=(others=>'0') when clr='1' else rx;

      --raux<= (others=>'0') when rst='1' else mult when rising_edge(clk);

      lim : macglimiter generic map(2*ec+fc,fc,ec+fc,0) port
map(mult(2*(ec+fc)-1 downto fc-1),aux);
      aux1<=aux when rising_edge(clk) and k='1';

      --div
      div : div_f generic map(2*ec,2*fc) port
map(clk,rst,str_d,num,den,q,rdy_d);
      ----
      --raiz
      yp<=(others=>'0') when rst='1' else str_r & yp(ec+fc-1 downto 1) when
rising_edge(clk);
      yn<=(others=>'0') when rst='1' else ynx when rising_edge(clk);
      ynx<= (others=>'0') when str_r='1' else yq;
      yr<=yp or yn;
      yq<=yn when mult(2*(ec+fc)-1 downto 0)>q else yr;
      rdy_r<=yp(0);
      ------------------
      dout<=yn when rising_edge(clk);
```

```
	r1<=(others=>'0') when rst='1' else rxb when rising_edge(clk) and
g(0)='1';
	r2<=(others=>'0') when rst='1' else rxb when rising_edge(clk) and
g(3)='1';
	r3<=(others=>'0') when rst='1' else rxb when rising_edge(clk) and
g(2)='1';
	r4<=(others=>'0') when rst='1' else rxb when rising_edge(clk) and
g(1)='1';
	rxb<=(others=>'0') when clr='1' else r;

end Behavioral;
```