INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE INGENIERÍA MECÁNICA Y ELÉCTRICA

SECCIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACIÓN

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

Implementación de un medidor inteligente con tecnología ARM

T E S I S

QUE PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS

EN INGENIERÍA ELÉCTRICA

PRESENTA:

RICARDO CANO CARBAJAL

MÉXICO, D. F. JULIO 2014

SIP-14

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

ACTA DE REVISIÓN DE TESIS

En la Ciudad de **México, D. F.** siendo las **17:00** horas del día **27** del mes de **Mayo** del **2014** se reunieron los miembros de la Comisión Revisora de la Tesis, designada por el Colegio de Profesores de Estudios de Posgrado e Investigación de **ESIME-Zacatenco** para examinar la tesis titulada:

IMPLEMENTACIÓN DE UN MEDIDOR INTELIGENTE CON TECNOLOGÍA ARM

Presentada por el alumno:

CANO	**CARBAJAL**	**RICARDO**
Apellido paterno	Apellido materno	Nombre(s)

Con registro:

B	1	2	0	8	8	5

aspirante de:

MAESTRO EN CIENCIAS EN INGENIERÍA ELÉCTRICA

Después de intercambiar opiniones, los miembros de la Comisión manifestaron ***APROBAR LA TESIS***, en virtud de que satisface los requisitos señalados por las disposiciones reglamentarias vigentes.

LA COMISIÓN REVISORA

Director(a) de tesis

DR. RAÚL ÁNGEL CORTÉS MATEOS

PRESIDENTE

DR. RICARDO OCTAVIO MOTA PALOMINO

SEGUNDO VOCAL

DR. DOMITILO LIBREROS

TERCER VOCAL

DR. JAIME ROBLES GARCÍA

SECRETARIO

DR. DAVID SEBASTIAN BALTAZAR

PRESIDENTE DEL COLEGIO DE PROFESORES

DR. MAURO ALBERTO ENCISO AGUILAR

ESCUELA SUPERIOR DE INGENIERIA MECANICA Y ELECTRICA
ESTADOS UNIDOS MEXICANOS
I. P. N.
SECCIÓN DE ESTUDIOS DE POSGRADO E INVESTIGACION

INSTITUTO POLITÉCNICO NACIONAL

SECRETARÍA DE INVESTIGACIÓN Y POSGRADO

CARTA CESIÓN DE DERECHOS

En la Ciudad de México, D.F. el día 4 del mes de Junio del año 2014, el que suscribe **Ing. Ricardo Cano Carbajal** alumno del Programa de Maestría en Ciencias en Ingeniería Eléctrica, con número de registro B120885, adscrito a la Sección de Estudios de Posgrado e Investigación de la ESIME Unidad Zacatenco, manifiesto que es el autor intelectual del presente trabajo de Tesis bajo la dirección del **Dr. Raúl Ángel Cortes Mateos** y cede los derechos del trabajo titulado **Implementación de un Medidor Inteligente con tecnología ARM**, al Instituto Politécnico Nacional para su difusión, con fines académicos y de investigación.

Los usuarios de la información no deben reproducir el contenido textual, gráficas o datos del trabajo sin el permiso expreso del autor y/o director del trabajo. Este puede ser obtenido escribiendo a las siguientes direcciones: richarcc_05@hotmail.com, rcortes@ipn.mx.

Si el permiso se otorga, el usuario deberá dar el agradecimiento correspondiente y citar la fuente del mismo.

Ricardo Cano Carbajal

Nombre y firma del alumno.

Agradecimientos:

Quiero Agradecer en primer lugar al ser que me ha permitido vivir y llegar hasta en donde estoy, al ser que sin su voluntad nada es posible, al ser que todo lo puede: Gracias a DIOS.

Por supuesto agradezco al apoyo incondicional de mi familia en los momentos difíciles y que gracias a ellos he podido superar, a mis Padres, mis Hermanos y a mi Abuelita quien siempre está al pendiente de mí.

Gracias también al Dr. Raúl Cortes Mateos quien ha sido el asesor de este trabajo y que me brindó todo su apoyo en el desarrollo y culminación de este trabajo.

Agradezco al Dr. Ricardo Mota Palomino, Dr. David Sebastián Baltazar, Dr. Domitilo Libreros, Dr. Jaime Robles García y Dr. Jaime Rodríguez Rivas, por sus valiosas observaciones, comentarios y sugerencias, para el desarrollo de un mejor trabajo de tesis.

A mis compañeros y amigos que gracias a ellos ha sido muy agradable la estancia en esta sección de posgrado.

Gracias también a la institución que me ha formado académicamente desde el Medio Superior, Licenciatura y ahora Nivel Posgrado, gracias al Instituto Politécnico Nacional y a la Escuela Superior de Ingeniería Mecánica y Eléctrica Unidad Zacatenco.

Por último y no menos importante agradezco a la institución que gracias al apoyo económico se hizo posible la culminación de mis estudios. Gracias a CONACYT.

Dedicatoria

A mi Papá que siempre ha estado al pendiente de mí y quien me ha enseñado a trabajar. A mi Mamá que con su apoyo incondicional he podido llegar a concluir mis estudios y a mis hermanos quienes siempre han depositado su confianza en todo lo que hago.

RESUMEN

En este trabajo se hace uso de un sistema embebido con un microcontrolador Kinetis KL25Z el cual está diseñado bajo la tecnología ARM Cortex M0+.

Este microcontrolador es de tipo general, lo que significa que puede ser usado para aplicaciones de medición a pesar de contar solo con un convertidor analógico/digital de 16 bits.

Los objetivos principales de este trabajo son demostrar que es posible construir un medidor de energía electrica con microcontroladores ARM, mediante la construcción de un prototipo para aplicaciones de uso doméstico e industrial monofásico y algoritmos que permitan conocer el consumo de energía en kWh y el costo de la misma.

El prototipo presentado cuenta con una pantalla táctil como interfaz hombre máquina, Bitácora de consumo histórico en memoria SD, e interfaces para comunicación remota.

Los algoritmos desarrollados permiten calcular el consumo de energía en kWh y el costo de la energía de acuerdo a las tarifas actuales, que se encuentran almacenadas en la memoria del microcontrolador.

El programa desarrollado en este prototipo se implementó en lenguaje C y el sistema operativo en tiempo real MQX lite. Para la depuración se utilizó la herramienta FreeMaster que permite visualizar en tiempo real los datos analógicos adquiridos por el Convertidor Analógico Digital así como monitorear los cálculos internos realizados por el microcontrolador.

En esta tesis se utilizan algoritmos de medición en el dominio del tiempo para el cálculo de las variables eléctricas tales como: tensión, corriente, potencias, factor de potencia, frecuencia, energía. Los resultados de dichas variables son desplegados en una interfaz hombre-máquina compuesta por una pantalla táctil.

ABSTRACT

This work uses an embedded system with the microcontroller KL25Z which is designed under the ARM Cortex M0+ technology. The embedded system is a general purpose microcontroller, that means it can be used for measurement applications, despite it has only one 16 bit analogic to digital converter.

The main objectives of this work are to demonstrate that it is possible to build an electric energy meter with ARM microcontrollers, by building a prototype for domestic and single-phase industrial applications, and the application of algorithms that reveal energy consumption in kWh and the cost of the same.

The prototype presented has a touch screen as human machine interface, historical log of consumption in SD memory, and interfaces for remote communication.

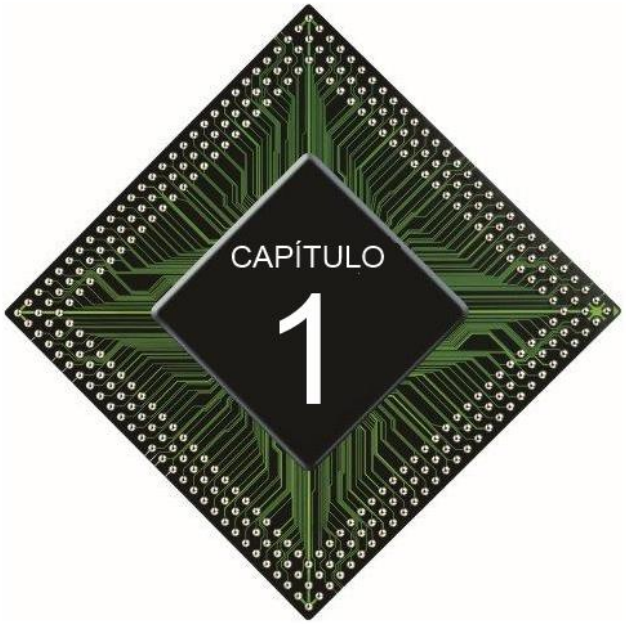
The algorithms developed allow calculating the energy consumption in kWh and energy cost according to the current rates, which are stored in the microcontroller.

The program developed in this prototype was implemented in C language and using the real-time operating system MQX lite. For debugging the FreeMaster tool was used to visualize real-time data acquired by Analog Digital Converter and monitor internal calculations performed by the microcontroller.

In this thesis measurement algorithms in the time domain are used to calculate the electrical variables such as voltage, current, power, power factor, frequency, energy. The results of these variables are displayed in a touch screen used as a Human-Machine Interface.

Índice

Índice de Figuras



Capítulo 1. Introducción

1.1 Introducción

Una parte fundamental del sistema eléctrico son los medidores eléctricos ya que son dispositivos que miden la cantidad de energía eléctrica consumida por un usuario y vigilan la transferencia de energía presente en un punto específico del sistema. Existen distintos tipos de medidores:

- Medidores Electromecánicos
- Medidores Digitales

1.1.1 Medidores Electromecánicos

Los medidores electromecánicos han sido usados para medir el consumo eléctrico doméstico e industrial por décadas [1], estos medidores están basados en el principio de inducción y está formado por partes mecánicas y eléctricas (estator, bobina de potencial, bobina de corriente, rotor etc) [2]. Sin embargo en la actualidad es necesario cumplir con ciertos estándares de calidad y precisión [3] que un medidor electromecánico sería incapaz de cumplir. En la figura 1.1 se muestra el diagrama a bloques de un medidor convencional, donde se puede observar que el flujo de información es unidireccional, además cabe mencionar de que la forma de recolectar información

consiste en el desplazamiento de personal de la compañía suministradora lo cual implica grandes gastos para ésta.

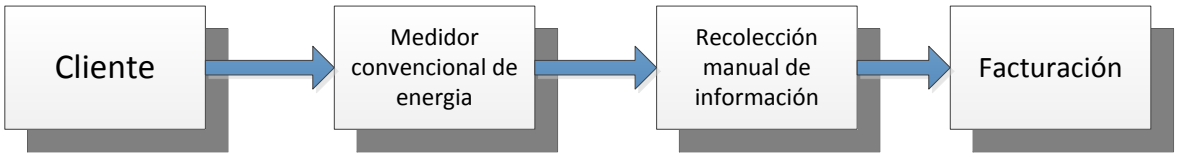


Figura 1. 1 Diagrama de bloques de un medidor convencional

1.1.2 Medidor Digital

En la actualidad con el desarrollo de microcontroladores de bajo costo es posible construir medidores digitales que proporcionan los mismos servicios que un medidor electromecánico, y que además ofrecen distintos beneficios como [4]:

- Mayor precisión.
- No presenta desgaste ya que no tiene partes móviles.
- Soporta diferentes tipos de mediciones.
- Actualización del software.

Un medidor digital típico contiene un circuito de acondicionamiento de señales, cuenta con un convertidor analógico digital el cual normalmente se encuentra embebido en un sistema digital como un microcontrolador.

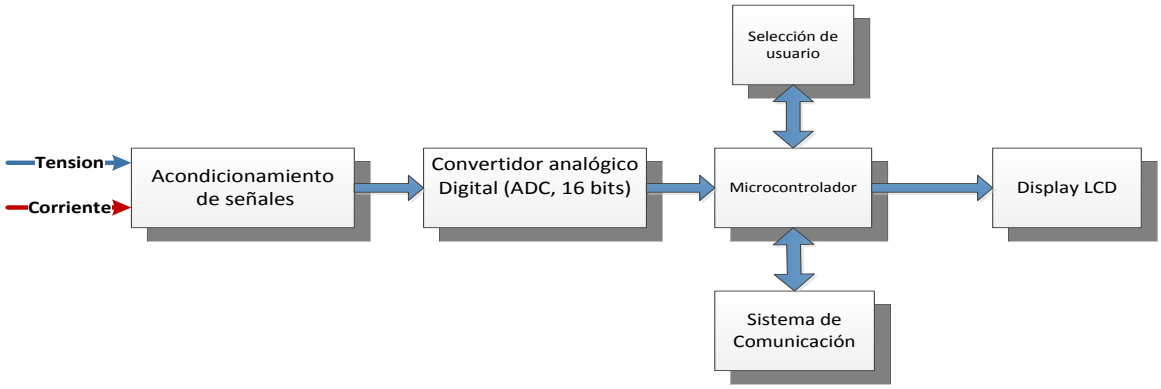


Figura 1. 2 Diagrama a Bloques de medidor digital **[5]**.

El circuito de acondicionamiento es usado para adaptar la alta amplitud de las señales de la red a amplitudes más bajas, las cuales sumadas con una tensión de desplazamiento se pueden introducir al ADC del microcontrolador con el objetivo de ser digitalizadas.

Una vez digitalizadas las señales están son procesadas por un Microcontrolador para calcular los parámetros de la red, tales como valores eficaces, potencias, factor de potencia etc. Con los parámetros calculados la sección de comunicación tiene la responsabilidad de enviar la información usando distintos protocolos e interfaces con otros módulos conectados como esclavos.

Uno de los aspectos más importantes en los medidores electrónicos modernos es la tecnología relacionada con la recolección del consumo de energía, diagnóstico y estado del dispositivo, con el objetivo de facturación, resolución de problemas y análisis de calidad de la energía. Los medidores con tecnología AMR pueden leer y enviar información con mecanismos como:

- Radio frecuencia.
- PLC (Power line carrier).
- Puerto serial.

Adicionalmente, es posible agregar sistemas de comunicación al dispositivo, lo que le permite ser supervisado, controlado e incluso reprogramado en forma remota. Se considera como ***medidor inteligente*** a aquellos medidores con tecnología digital que son capaces de enlazarse a un sistema de comunicación bidireccional y vincular al usuario con la empresa suministradora del servicio, utilizando redes inalámbricas u otros medios de comunicación convencional. Estos medidores poseen todas las virtudes del medidor digital, más las que le proporciona un sistema de comunicación de estas características. La tabla 2.2 menciona las principales ventajas de la instalación medidores inteligentes sobre medidores digitales sin comunicación.

Tabla 1. 1. Ventajas de los medidores digitales.

Consumidores	➢ Información para gestión en el uso de energía. ➢ Facturación precisa y oportuna. ➢ Información sobre el consumo de la energía.
Operaciones de campo	➢ Menor costo en lectura de medidores. ➢ Reducción de toma de lecturas extemporáneas. ➢ Elimina el uso de aparatos portátiles para toma de lecturas.
Facturación, contabilidad y protección de ingresos	➢ Detección temprana de alteración de medidores o robo de energía. ➢ Disminución de estimación en facturación. ➢ Reducción de ajustes de cuenta por errores de facturación
Empresa suministradora del servicio	➢ Disminución de quejas. ➢ Mejor perfil de riesgo. ➢ Reducción en accidentes de operadores.

1.2 Objetivo

Implementar un medidor de energía utilizando tecnología ARM, para permitir que los clientes conozcan su consumo de energía.

Objetivos Particulares.

- Implementar un prototipo de medidor de energía con un microcontrolador ARM de última generación de bajo costo.
- Implementar un prototipo el cual sea capaz de medir la energía consumida o la energía generada por el usuario.
- Mostrar la utilización de la medición diferencial del voltaje.
- Implementar el programa de medición mediante un sistema operativo que permita sincronizar las tareas definidas por el prototipo.
- Mostrar los conceptos fundamentales del desarrollo de software para sistemas embebidos.
- Implementar una interfaz Hombre-Máquina la cual permita al usuario relacionarse con las tarifas actuales de CFE.

1.3 Justificación

En una época de aumento del precio de la energía, muchos países exigen que las empresas de servicios públicos ofrezcan más opciones para el uso eficiente de la energía, estos proyectos tiene como objetivo afrontar el hecho de que los consumidores en la actualidad no tienen contacto con el precio de la electricidad ni información relacionada con este de manera oportuna que permita la toma de decisiones y su consumo no se diferencia por período, es decir, no se tiene información del consumo en cada hora del día.

Pero por medio de medidores digitales que realizan una medición del consumo horario se logrará aumentar la susceptibilidad de los mismos ante los cambios en el precio del mercado.

Los Medidores Digitales animarán a los clientes a pensar más en cómo y cuándo se consume electricidad, lo cual, a su vez, les ayudará a satisfacer las necesidades energéticas actuales y futuras. Es por eso que este trabajo se implementa un prototipo el cual permita que el cliente conozca el consumo de energía y el costo de ésta en el momento que el usuario la requiera.

1.4 Marco Teórico

Este trabajo está basado en documentos extraídos de la base de datos de la IEEE, notas de aplicación de compañías como Freescale, Texas Instruments, así como trabajos desarrollados en la Sección de Estudios de Posgrado e Investigación (SEPI-ELECTRICA).

A continuación se describen algunos de los trabajos utilizados en el desarrollo de este trabajo.

Notas de Aplicación y artículos de la IEEE:

[6] Implementation of a Three-Phase Electronic Watt-Hour Meter using the MSP430F47. En esta nota de aplicación de Texas Instruments, se propone el hardware para la medición de tensión de forma diferencial de un medidor trifásico.

[7] Design of a Smart Meter Techno-Economic Model for Electric Utilities in Ontario. Muestra los retos que deben satisfacer las redes de medidores, se hace un análisis de las ventajas que ofrecen tales redes, además de mostrar el estado de desarrollo de dichas redes.

[8] Medidor de Potencia Activa Monofásico y Trifásico. Describe la medición de la energía activa en aplicaciones monofásicas y trifásicas utilizando un microcontrolador Motorola MC908GP32.

[9] Single-Phase Electricity Meter. En esta nota de aplicación se muestra el diseño de un medidor de energía utilizando un microcontrolador Kinetis M.

Algunos trabajos desarrollados en el IPN son:

[10] *Diseño de un medidor electrico digital de prepago*. En esta tesis se realiza el diseño y la implementación de un medidor de prepago. Aquí se muestran las ventajas que ofrecen los medidores digitales sobre los medidores electromecánicos.

[11] *Diseño de medidor inteligente e implementacion de sistema de comunicación bidireccional*. Se describe la implementación de una red de medidores inteligentes. Aquí se da una descripción detallada de los algoritmos de medición tanto en el dominio del tiempo como en el dominio de la frecuencia, se muestran las comparaciones de los resultados de la aplicación de ambos algoritmos.

[12] *Diseño e implementacion de un medidor fasorial sincrono normalizado con el estandar IEEE C37.118.* En esta tesis se diseña e implementa un medidor fasorial síncrono capaz de proporcionar mediciones fasoriales de tensión y corriente sincronizado vía satélite mediante un módulo receptor GPS y transmitir la información de los fasores a un centro de control.

1.5 Aportaciones.

En la Sección de Estudios de Posgrados e investigación se han desarrollado trabajos sobre la medición de variables eléctricas, y con base en estos se puede concluir que las aportaciones de este trabajo son las siguientes:

- Utilización de microcontroladores de última generación de 32 bits, conocidos como Kinetis.
- Implementación de un medidor bidireccional, con lo que es posible medir si el usuario consume o genera energía.
- La utilización de un sistema operativo en tiempo real, el cual permite la sincronización de las distintas tareas que debe cumplir el prototipo.

- Implementación de la medición de tensión en forma diferencial, el cual permite la eliminación del ruido, y la necesidad de restar por software el valor de tensión de desplazamiento el cual se traduce en un menor uso del CPU del sistema.

1.6 Estructura de la tesis.

A continuación se muestra la organización de este trabajo. Se describe de forma general el contenido por capítulo y por apéndice.

Capítulos.

En el **capítulo 1**, se da una descripción general del contenido de este trabajo. Se define el objetivo de este trabajo, así como la justificación, alcance, aportaciones y se da un panorama general del contenido de la tesis.

En el **capítulo 2,** en este capítulo se muestran los conceptos generales sobre los algoritmos de medición en el dominio del tiempo,

En **el capítulo 3**, se muestra el desarrollo tanto del hardware como del software. Se inicia con la descripción del sistema de desarrollo empleado, continua con la descripción de los circuitos de acondicionamiento de señales de tensión y corriente. Posteriormente se describe el software implementado en el microcontrolador, se describe de forma detallada como fue que se configuró cada elemento del que consta el programa.

En **el capítulo 4** se muestran las pruebas que se le hicieron al prototipo así como los resultados que arroja el prototipo de medición.

Capítulo 5. Aquí se dan las conclusiones del trabajo, aportaciones y recomendaciones para trabajos a futuro.

Apéndices.

Apéndice A. Se da una descripción general acerca del entorno de desarrollo utilizado en esta tesis: CodeWarrior 10.5

Apéndice B. Se definen los conceptos fundamentales acerca del sistema operativo MQX lite. Se muestran un conjunto de instrucciones propias de MQX Lite.

Apéndice C. Aquí se muestran los esquemáticos utilizados en la tarjeta de adquisición de datos.

Apéndice D. Se da una descripción del software utilizado para el manejo de la pantalla.

Apéndice E. Se muestran las características de los microcontroladores ARM

Apéndice F. Se muestra la configuración de la herramienta de Freemaster.

Apéndice G. En esta sección se muestra el programa en lenguaje C del prototipo implementado.

Capítulo 2. Algoritmos de Medición de Energía.

2.1 Introducción.

El desarrollo de proyectos en donde se requiere la adquisición de información utilizando dispositivos como Convertidores Analógicos Digitales (ADC, por sus siglas en inglés), hace necesario el conocimiento de conceptos sobre el muestreo y tratamiento de señales, es por eso que en la primera parte de este capítulo se muestran dichos conceptos.

Posteriormente se definen los algoritmos de medición utilizados en este trabajo, se dan las ecuaciones tanto en tiempo continuo como en tiempo discreto.

Por último se muestra la forma en que la compañía suministradora realiza la facturación de un servicio monofásico domiciliario.

2.2 Muestreo de Señales.

Para implementar cualquier dispositivo capaz de recopilar información a través de un microcontrolador es necesario conocer los conceptos básicos del procesamiento de señales digitales: muestreo digital y el teorema del muestreo.

2.2.1 Muestreo Digital.

El muestreo digital es un proceso de adquisición de datos, a intervalos de tiempo equidistantes, consistente en la obtención del valor que toma la señal original en un momento dado [13]. El parámetro fundamental del muestreo digital es el intervalo de muestreo Δs, o su equivalente en frecuencia, 1/Δs, cuanto menor sea Δs, mayor número de datos se tendrá de la señal. En la figura 2.1 se observa el muestreo de una señal obtenida de algún fenómeno físico.

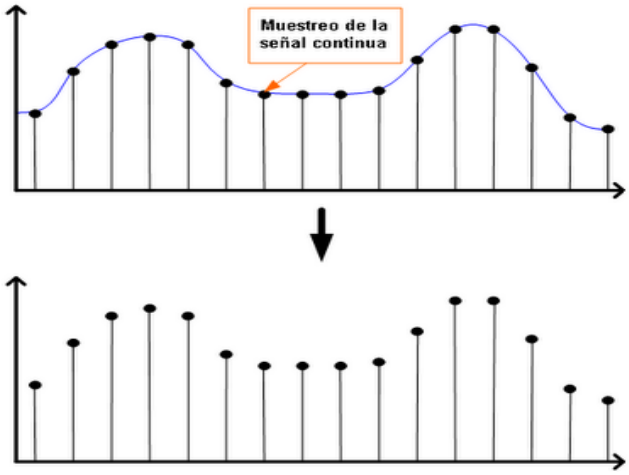


Figura 2. 1 Muestreo de una señal analógica.

El resultado de dicho muestreo es la obtención de una serie discreta ordenada x[n] = {x[0], x[1], x[2],…,x[N] }, en la que el índice n indica la posición de orden temporal del valor x[n]. En este trabajo este arreglo es la estructura de datos con el que los algoritmos de medición trabajan.

2.2.2 Teorema del muestreo.

Según el teorema del muestreo de Nyquist, para poder reconstruir con exactitud la forma de una onda es necesario que la frecuencia de muestreo f_m, sea como mínimo el doble de la máxima frecuencia a muestrear [14]. Por lo tanto para que el muestreo de una señal continua sea correcto, se debe escoger la frecuencia de muestreo de tal forma que:

$$f_m = 2\, f_{max} \qquad \textbf{2. 1}$$

Dónde:

f_m Representa la frecuencia de muestreo.

f_{max} Representa frecuencia máxima de la señal a muestrear.

En la figura 2.2 se presentan dos formas de onda con distintas frecuencias. La señal con mayor número de ciclos representa una señal continua obtenida de algún fenómeno físico mientras la otra señal representa la forma de onda construida a partir del muestreo de la primera. Se puede observar que la forma de onda construida a partir de las muestras M1-M10 no representa de forma correcta la forma de onda original, y esto es debido a que la frecuencia de muestreo no cumple con la ecuación 2.1, este fenómeno es conocido como Aliasing [15].

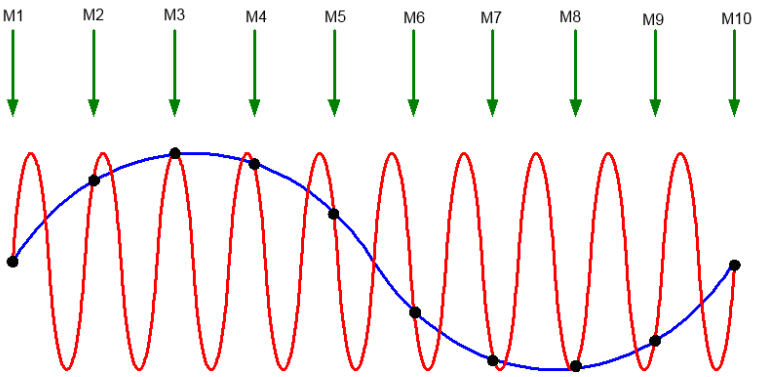


Figura 2. 2 Fenómeno de Aliasing.

2.3 Algoritmos de medición en el dominio del tiempo

Los algoritmos de medición en el dominio del tiempo, son métodos para el cálculo de las variables eléctricas más comunes en ingeniería eléctrica, a continuación se destacan algunas características del empleo de estos algoritmos [11].

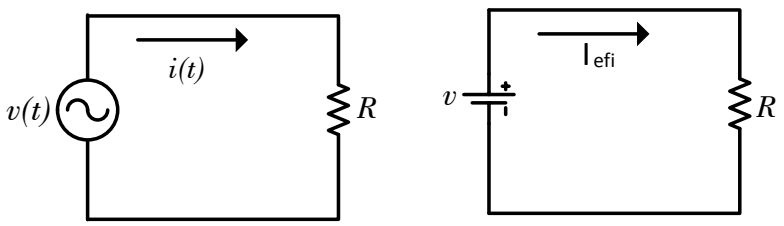
- La precisión de los algoritmos en el dominio del tiempo comparada con algoritmos en el dominio de frecuencia es idéntica.
- Utilizan directamente las muestras de las señales de interés para sus cálculos.
- Son deseables para aplicaciones donde no es necesario conocer el contenido armónico de las señales.

- La cantidad de operaciones implementadas en estos algoritmos requeridas aumenta en forma lineal.

A continuación se describen los algoritmos de medición utilizados en este trabajo, éstos se describen tanto en sus expresiones continuas como en su expresión discreta, para la implementación en el microcontrolador.

2.3.1 Valor Eficaz

La idea de valor eficaz surge de la necesidad de medir la eficacia de una fuente de tensión o de corriente, variante en el tiempo, en el suministro de potencia a una carga resistiva. Para calcular la corriente eficaz proporcionada por la fuente variante en el tiempo *v(t)* considere los circuitos de la figura 2.3 [16].



a) Circuito básico de Corriente Alterna

b) Circuito básico de Corriente Directa

Figura 2. 3 Determinación de la corriente eficaz.

La potencia promedio absorbida por el resistor en el circuito de corriente alterna está expresada por la ecuación:

$$P = \frac{1}{T}\int_0^T i(t)^2\, R\, dt = \frac{R}{T}\int_0^T i(t)^2\, R\, dt \qquad \textbf{2. 2}$$

En tanto que la potencia absorbida por el resistor de la figura 2.3b, se define por:

$$P = {I_{efi}}^2 R \qquad \textbf{2. 3}$$

Al igualar ambas expresiones y despejar I_{efi} se obtiene:

$$I_{efi} = \sqrt{\frac{1}{T}\int_0^T i(t)^2 dt} \qquad \textbf{2. 4}$$

Es decir, **el valor eficaz de una corriente periódica es la corriente de corriente directa que suministra la misma potencia promedio a una resistencia que la corriente periódica**.

La ecuación equivalente en tiempo discreto es:

$$X_{rms} = \sqrt{\frac{1}{N}\sum_{n=0}^{N-1} x[n]^2} \qquad \textbf{2. 5}$$

Dónde:

X_{rms} : es el valor eficaz de la señal

x[n] función periódica en tiempo discreto

N representa el número de muestras en un ciclo.

2.3.2 Potencia Instantánea, y energía activa

La potencia instantánea ***p(t)*** absorbida por un elemento es el producto de la tensión instantánea ***v(t),*** en las terminales del elemento y la corriente instantánea ***i(t)*** a través de él. La potencia instantánea está definida por:

$$p(t) = v(t)\, i(t) \qquad \textbf{2. 6}$$

La potencia instantánea (en watts) es la potencia en cualquier instante. Sin embargo a la compañía suministradora le interesa la energía consumida por el usuario, y está definida por:

$$E_{activa} = \int_{t_1}^{t_2} p(t)dt = \int_{t_1}^{t_2} v(t)\, i(t)dt \qquad \textbf{2. 7}$$

Dónde:

v(t), i(t), tensión y corriente eléctrica presentes en el elemento.

t_1 Representa el tiempo en el que se inicia la transferencia de energía

t_2 Representa el tiempo en el que finaliza la transferencia de energía.

La energía activa en tiempo discreto está definida por la siguiente ecuación:

$$E_{activa} = \sum_{n=0}^{N-1} v[n]i[n] \qquad \textbf{2. 8}$$

Dónde:

E_{activa} Es la energía que los dispositivos eléctricos transforman en trabajo útil.

N número de muestras.

2.3.3 Potencia Activa

La potencia activa, medida en watts, es el promedio de la potencia instantánea a lo largo de un periodo, está definida por:

$$P = \frac{1}{T}\int_0^T p(t)dt \qquad \textbf{2. 9}$$

Y para señales discretas está definida por:

$$P = \frac{1}{N}\sum_{n=0}^{N-1} v[n]\, i[n] \qquad \textbf{2. 10}$$

2.3.4 Potencia reactiva

La potencia reactiva no es una potencia realmente consumida en la instalación, ya que no produce trabajo útil debido a que su valor medio es nulo. Aparece en una instalación eléctrica en la que

existen bobinas o capacitores, y es necesaria para crear campos magnéticos y eléctricos en dichos componentes. Se representa por Q y se mide en volt-amperes reactivos (VAr). Está definida por la siguiente ecuación:

$$Q = \sqrt{S^2 - P^2} \qquad \textbf{2. 11}$$

2.3.5 Potencia aparente

La potencia aparente está definida por:

$$S = V_{rms} I_{rms} \qquad \textbf{2. 12}$$

Se le llama así porque aparentemente la potencia debería ser el producto de la tensión por la corriente, por analogía con los circuitos resistivos alimentados con corriente directa. Esta potencia se mide en volt-amperes o VA para distinguirla de la potencia promedio, que se mide en watts. En la figura se observa la relación entre S, P y Q.

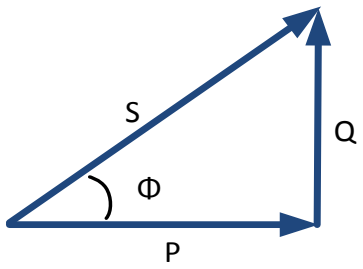


Figura 2. 4 Triángulo de potencias.

2.3.6 Factor de potencia

El factor de potencia es el coseno de la diferencia de fase entre la tensión y la corriente. También es igual al coseno del Angulo de la impedancia. El factor de potencia se puede calcular con las siguientes expresiones:

$$fp = \frac{P}{S} = \cos(\theta_v - \theta_i) \qquad \textbf{2. 13}$$

2.3.7 Frecuencia: Algoritmo de cruce por cero

El algoritmo para calcular la frecuencia de las señales, es el algoritmo de cruce por cero, éste consiste en detectar un cruce por cero con pendiente positiva, de la señal a analizar, y a partir de ese

instante contabilizar el número de muestras trascurridas hasta que suceda otro cruce por cero con las mismas características.

En la figura 2.5 se observa como el número de muestras que caben en un ciclo no es un número entero, hay que determinar las fracciones de muestras que se obtienen antes y después del cruce por cero, para conocer el número fraccionario de muestras que caen en un ciclo.

Figura 2. 5 Algoritmo de cruce por cero.

Una vez conocido el número exacto de muestras entre ceros, se puede calcular la frecuencia de la señal con la siguiente ecuación:

$$f_{señal} = \frac{f_m}{N_{ceros}} \qquad \textbf{2. 14}$$

Dónde:

$f_{señal}$ es la frecuencia de la señal bajo análisis.

f_m Frecuencia de muestreo

N_{ceros} Número de muestras entre ceros.

2.4 Tarifas eléctricas en México

La Comisión Federal de Electricidad es la empresa gubernamental encargada de la prestación de servicio público de la energía eléctrica en México [17]. De acuerdo con el artículo 27 de la Constitución Política de los Estados Unidos Mexicanos y el primero de la Ley del Servicio Público

de Energía Eléctrica, la nación tiene el ejercicio exclusivo de generar, transportar, modificar, distribuir y abastecer el recurso eléctrico.

La paraestatal aplica diferentes tarifas para los hogares y negocios además de determinar el costo del consumo en diferentes apartados. En la siguiente sección se describen las tarifas aplicadas al uso de este servicio así como el sistema de cobro en México.

En primer lugar la CFE divide sus tarifas en cinco grupos de clientes:

- Domestico
- Agrícola
- Industrial
- Comercial
- Servicio

En donde la unidad de medida es el kWh.

En el caso de las tarifas destinadas a los hogares, estas se clasifican por el nivel de consumo en ocho rangos: (1, 1A, 1B, 1C, 1D, 1E, 1F y DAC), de los cuales los siete primeros están relacionados con la temperatura media de cada región; es decir los precios son diferentes en las distintas entidades federativas debido a los subsidios aplicables en las regiones más cálidas del país (debido a la demanda de energía en refrigeración, por ejemplo), además existe una variación en los precios dependiendo de la época de verano o invierno. Por otra parte la tarifa DAC (Doméstica de Alto Consumo) tiene relación con la demanda de utilización de electricidad, por lo que este cobro "aplica cuando el cliente excede el límite establecido para la localidad del usuario en el consumo mensual promedio de los últimos 12 meses. La tabla muestra la relación entre el tipo de tarifa y el promedio de temperatura anual [18].

Tabla 2. 1 Relación entre el tipo de tarifa y el promedio de temperatura anual.

Tarifa	Temperatura media Mínima
1	Menor a 25 °C
1A	25 °C
1B	28 °C
1C	30 °C
1D	31 °C
1E	32 °C
1F	33 °C
DAC	Aplica cuando el cliente excede el límite establecido.

Estas tarifas están subsidiadas mediante escalones, por lo que el costo de la electricidad es menor que el de otras tarifas.

A continuación se muestra la forma en que CFE factura a sus clientes residenciales. Se considera la figura 2.6 como ejemplo de facturación con los siguientes datos [19]:

Periodo de Consumo: 05 Dic 2013 – 06 Febrero 2014

Cargos por energía consumida, para consumos mayores a 150 (ciento cuarenta) kilowatts-hora.

Consumo básico	$ 0.792	Por cada uno de los primeros 75 (setenta y cinco) kilowatts-hora.
Consumo intermedio	$ 0.963	Por cada uno de los siguientes 65 (cincuenta) kilowatts-hora.
Consumo excedente	$ 2.817	Por cada kilowatt-hora adicional a los anteriores.

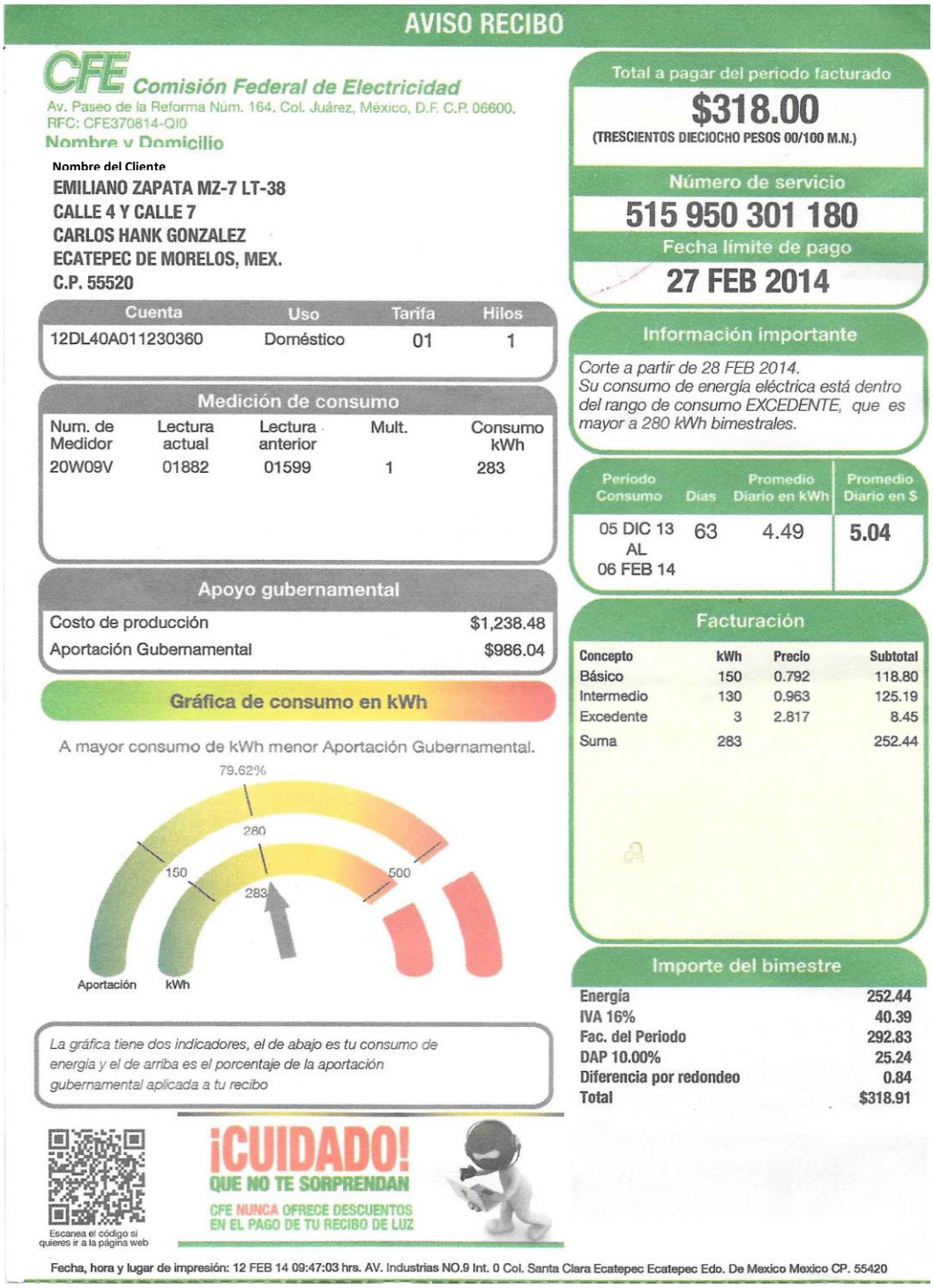
Del recibo podemos observar que el consumo del cliente fue de 283 kWh en un periodo de dos meses, por lo que al cliente se le factura lo siguiente:

Concepto	**kWh**	**Precio**	**Subtotal**
Básico	150	$0.792	$ 118.80
Intermedio	130	$0.963	$ 125.19
Excedente	3	$2.817	$ 8.45
Suma	283		$ 252.44

Este importa no incluye IVA por lo tanto:

Costo de la energía Consumida	**$252.44**
IVA 16%	$40.39
Facturación Total del Periodo	$292.83
DAP 10%	$25.24
Diferencia Por redondeo	$0.84
Total	$318.91

El DAP se refiere al Derecho al Alumbrado Publicado y en este recibo representa el 10% del costo de la energía consumida.



AVISO RECIBO

CFE Comisión Federal de Electricidad
Av. Paseo de la Reforma Núm. 164. Col. Juárez, México, D.F. C.P. 06600.
RFC: CFE370814-QI0

Nombre y Domicilio

Nombre del Cliente
EMILIANO ZAPATA MZ-7 LT-38
CALLE 4 Y CALLE 7
CARLOS HANK GONZALEZ
ECATEPEC DE MORELOS, MEX.
C.P. 55520

Cuenta	Uso	Tarifa	Hilos
12DL40A011230360	Doméstico	01	1

Medición de consumo

Num. de Medidor	Lectura actual	Lectura anterior	Mult.	Consumo kWh
20W09V	01882	01599	1	283

Apoyo gubernamental

Costo de producción	$1,238.48
Aportación Gubernamental	$986.04

Gráfica de consumo en kWh

A mayor consumo de kWh menor Aportación Gubernamental.

La gráfica tiene dos indicadores, el de abajo es tu consumo de energía y el de arriba es el porcentaje de la aportación gubernamental aplicada a tu recibo

Escanea el código si quieres ir a la página web

¡CUIDADO!
QUE NO TE SORPRENDAN
CFE NUNCA OFRECE DESCUENTOS EN EL PAGO DE TU RECIBO DE LUZ

Total a pagar del periodo facturado
$318.00
(TRESCIENTOS DIECIOCHO PESOS 00/100 M.N.)

Número de servicio
515 950 301 180

Fecha límite de pago
27 FEB 2014

Información importante
Corte a partir de 28 FEB 2014.
Su consumo de energía eléctrica está dentro del rango de consumo EXCEDENTE, que es mayor a 280 kWh bimestrales.

Periodo Consumo	Dias	Promedio Diario en kWh	Promedio Diario en $
05 DIC 13 AL 06 FEB 14	63	4.49	5.04

Facturación

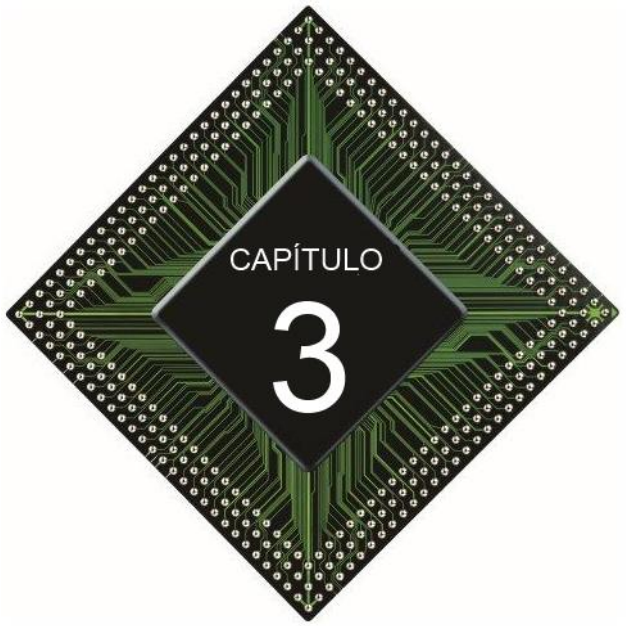
Concepto	kWh	Precio	Subtotal
Básico	150	0.792	118.80
Intermedio	130	0.963	125.19
Excedente	3	2.817	8.45
Suma	283		252.44

Importe del bimestre

Energía	252.44
IVA 16%	40.39
Fac. del Periodo	292.83
DAP 10.00%	25.24
Diferencia por redondeo	0.84
Total	$318.91

Fecha, hora y lugar de impresión: 12 FEB 14 09:47:03 hrs. AV. Industrias NO.9 Int. 0 Col. Santa Clara Ecatepec Ecatepec Edo. De Mexico Mexico CP. 55420

Figura 2. 6 Ejemplo de recibo de CFE



Capítulo 3. Descripción del Hardware y Software del sistema

3.1 Introducción

En este capítulo se describen los elementos de hardware y software utilizados en este trabajo. En primer lugar se muestra el diagrama a bloques del prototipo propuesto, en este diagrama se puede observar de forma general los componentes que forman el medidor.

Posteriormente se describen las características del sistema de desarrollo, las configuraciones del CPU, del bean MQX lite, del ADC, UART y demás. Al final se revisan las características del circuito utilizado para la adquisición de datos.

Posteriormente se describe el programa embebido en el microcontrolador, así como los diagramas de flujo de cada uno de los algoritmos de medición. Se muestra de forma detallada el uso de las herramientas de MQX Lite utilizadas en el proyecto.

3.1.1 Descripción del medidor.

El medidor cuanta con las siguientes características:

- 110/127 VCA, 60Hz
- Rango de medición de corriente de 0 - 45 A.
- Medición de:
  - ✓ Tensión y corriente.
  - ✓ Medición de energía activa, aparente.
  - ✓ Factor de potencia.
  - ✓ Frecuencia.
  - ✓ Energía consumida o generada.
- Transformador de Corriente como sensor de corriente.
- Divisor de tensión diferencial como sensor de tensión.
- Pantalla Táctil como interfaz Hombre-Maquina.
- Memoria SD para registro de historial de consumo de energía.

3.1.2 Diagrama a bloques del prototipo

En la figura 3.1 se muestra el diagrama a bloques del prototipo propuesto en esta tesis.

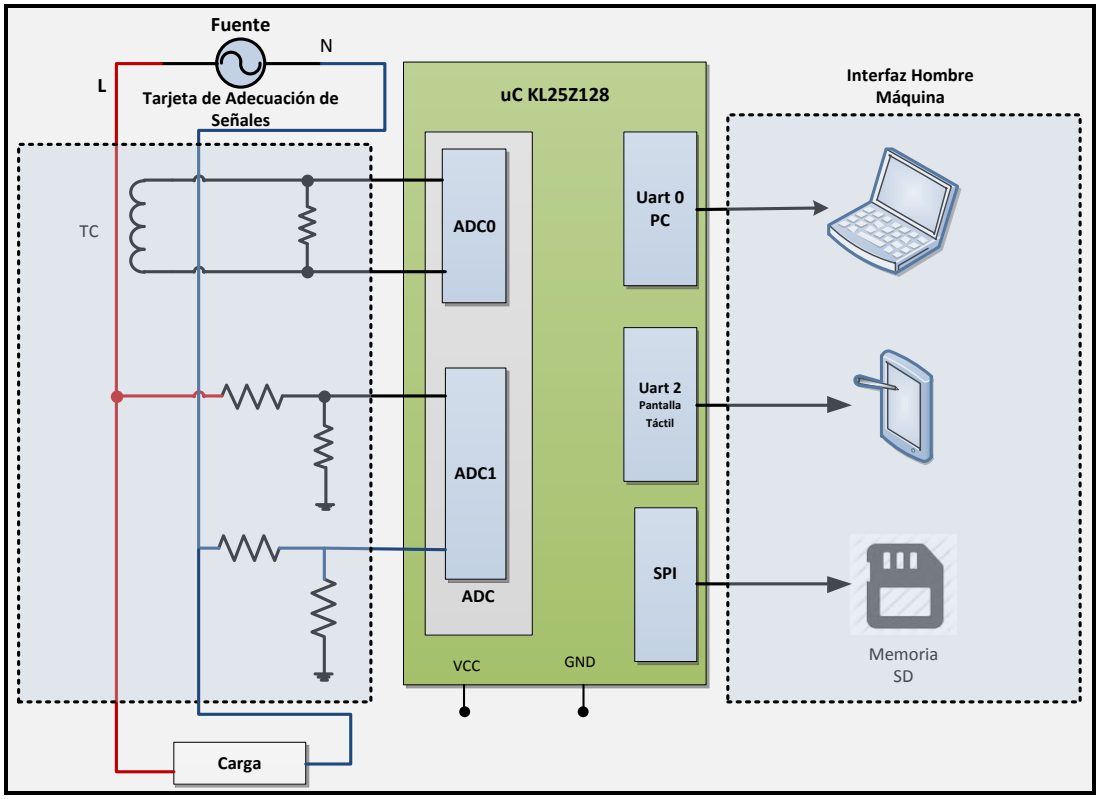


Figura 3. 1 Diagrama a bloques del prototipo.

3. 2 Descripción del Hardware del sistema

Con el desarrollo de dispositivos de estado sólido, hoy en día es posible construir dispositivos de bajo costo capaces de adquirir, procesar, enviar y comunicar información en tiempo real. El microcontrolador MKL25Z128, desarrollado por la compañía de FREESCALE SEMICONDUCTOR, es un dispositivo de bajo costo y consumo de energía, el cual se encuentra integrado en una tarjeta de desarrollo que tiene las siguientes características [20]:

- Procesador MKL25Z128VLK4.
- Touch Slider Capacitivo.
- Tensión de operación 3.3VCD
- Cristal de 8 MHz.
- Interfaz OpenSDA.
- Acelerómetro MMA8451Q, led RGB.
- Compartimiento para pila CR2032.

En la figura 3.2 se muestra el diagrama de bloques de la FRDM-KL25Z, y en la figura 3.3 se muestra la tarjeta de desarrollo empleado en este trabajo.

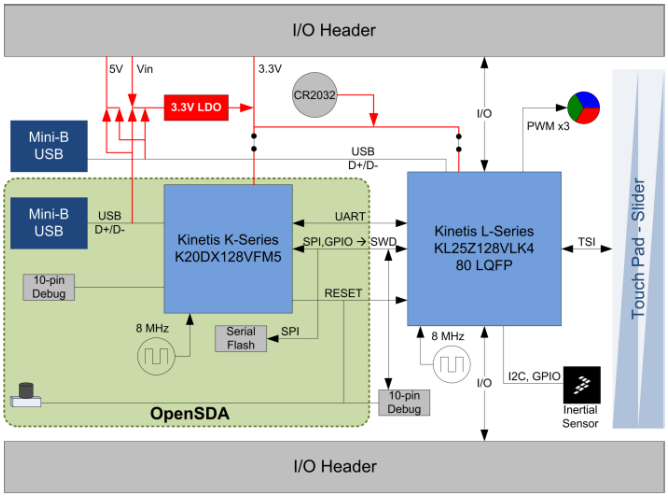


Figura 3. 2. Diagrama de bloques de la FRDM – KL25Z [21].

3.2.1 Microcontrolador KL25Z

El microcontrolador integrado en la FRDM es el KL25Z128VLK4, y pertenece a la familia Kinetis L[15]. Sus principales características son [21]:

- CPU: ARM Cortex M0+ de 32 bits.
  - ✓ Frecuencia de operación de hasta 48 MHz
  - ✓ Rápido acceso a los puertos de entrada y salida.
- Memorias
  - ✓ 128 Kb de Flash

- ✓ 16 Kb de SRAM

- Relojes:
  - ✓ Reloj interno de 32KHz y 4MHz
  - ✓ Oscilador de 1 KHz para el RTC y watchdog.
  - ✓ Modulo generador para altas frecuencias de operación.
- Periféricos analógicos:
  - ✓ ADC de 16 bits.
  - ✓ DAC de 12 bits.
  - ✓ Comparador de alta velocidad.
- Puertos de comunicación:
  - ✓ 1 UART de bajo consumo, 2 UART estándar
  - ✓ Dos módulos I2C
  - ✓ Dos puertos SPI
- Timers
  - ✓ Un módulo Timer/PWM de seis canales
  - ✓ Dos módulos Timer/PWM de dos canales
  - ✓ Real time Clock.
- HMI
  - ✓ Controlador de propósito general para entradas y salidas.
  - ✓ Sensor capacitivo.

Figura 3. 3 Sistema de desarrollo FRDM KL25Z.[21]

3.2.2 Interfaz OpenSDA

OpenSDA (Serial and Debug Adapter) es un software libre que se utiliza para la depuración de programas [22]. El circuito está basado en el µC Kinetis K20, el cual tiene la característica de contar con un controlador USB, lo que le proporciona un mecanismo para la ejecución de Aplicaciones OpenSDA, como programadores de memoria flash, interfaces de depuración, convertidores serial-usb etc. En la figura 3.4 se muestra el diagrama de bloques de esta interfaz.

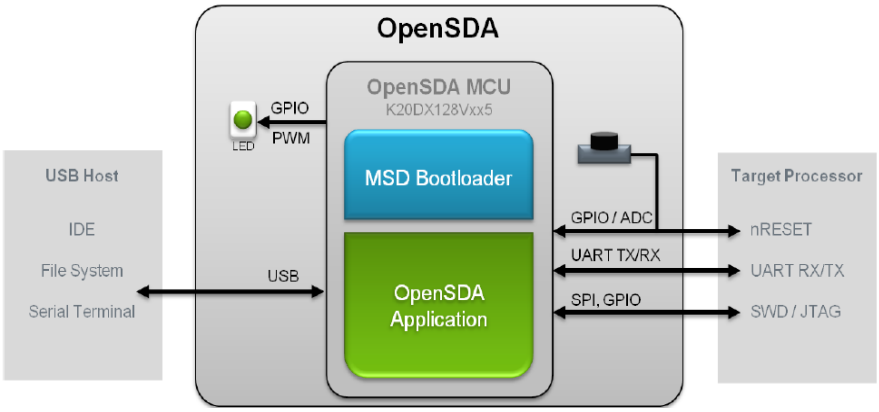


Figura 3. 4. Interfaz OpenSDA [22].

La interfaz OpenSDA, es una interfaz poderosa al depurar un programa ya que ésta permite:

- Colocar puntos de ruptura en cualquier parte del programa.
- Ejecutar el programa paso a paso.
- Visualizar variables en tiempo de ejecución, y con actualización de datos automática.
- Modificar variables cuando el programa se está ejecutando paso a paso.

3.2.3 Circuitos de acondicionamiento de señal.

Debido a que las señales de tensión y corriente de un sistema de energía residencial no son señales con las que los dispositivos digitales puedan trabajar, es necesario que éstas pasen a través de una etapa de acondicionamiento. En esta sección se describe el circuito que cumple con esta función.

El primer objetivo del circuito de acondicionamiento es atenuar la señal a un valor considerablemente menor, posteriormente se debe agregar una tensión de CD para convertir esta señal alterna en una tensión directa pulsante, la cual el ADC del microcontrolador pueda manejar. En la figura 3.5 se observa el proceso de adecuación de una señal alterna.

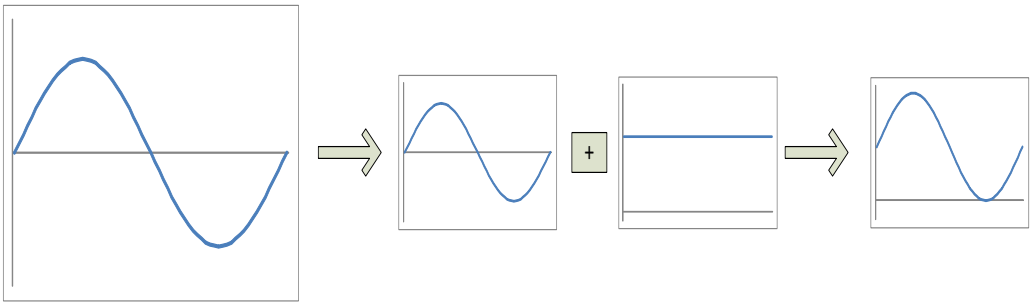


Figura 3. 5. Proceso de adecuación de una señal de corriente alterna.

Por último se agrega una etapa de filtraje y una etapa de protección al circuito, con lo cual se asegura que los niveles de tensión no sobrepasen los límites de operación del convertidor analógico digital. Esta etapa está compuesta por diodos Shotkky los cuales protegen al ADC de sobrepasar tensiones de 3.3 VCD o tensiones negativas. En el apéndice C se pueden ver los esquemáticos.

3.2.3.1 Circuito de acondicionamiento de tensión.

Se toma como de tensión máxima a medir de $200V_{RMS}$, y el circuito se basa en el principio del divisor de tensión. A continuación se muestra el circuito utilizado en este trabajo.

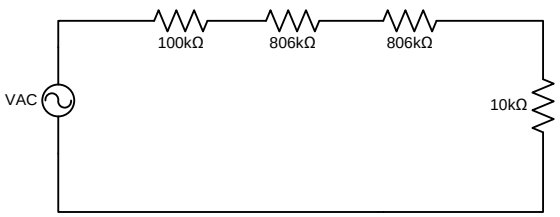


Figura 3. 6. Circuito de acondicionamiento de tensión.

Suponiendo que se aplica la tensión máxima al circuito de 200 V_{RMS}, se tiene en la resistencia una tensión de

$$V_{10k} = 200\ \frac{10k\Omega}{1722k\Omega} = 1.1614\ V_{RMS}$$

Y una tensión pico de

$$V_P = \sqrt{2} * 1.1614 = 1.6424\ V$$

Por lo que cuando se aplican 200 V_{RMS} se tiene una tensión pico a pico de 3.2848 VCA.

La tensión obtenida de la resistencia de 10K aún no se puede introducir al µC ya que esta sigue siendo alterna y como los microcontroladores trabajan únicamente con tensiones directas positivas, es necesario transformar ésta tensión alterna a una señal continua, esto se logra sumando una tensión directa positiva, llamada tensión de desplazamiento.

Debido a que el ADC del µC funciona con tensiones desde 0 hasta 3.3 V, se escoge una tensión de offset de 1.65 VCD. Por lo tanto si se suman 1.65VCD a la señal alterna se obtienen valores máximos y mínimos de:

$$V_{max} = 1.6424 + 1.65 = 3.2924\ VCD$$

$$V_{min} = -1.6424 + 1.65 = .0076\ VCD$$

Donde

V_{max} y V_{min} Representan la tensión máxima y mínima que el convertidor analógico digital tendrá en sus terminales, cuando $200V_{RMS}$ se encuentren en el circuito de la figura 3.7.

Finalmente se agrega un filtro pasabajas, un varistor con el objetivo de proteger al circuito contra voltajes superiores a 200 Vrms.

En la figura 3.7 se muestra el circuito de eléctrico utilizado en este trabajo.

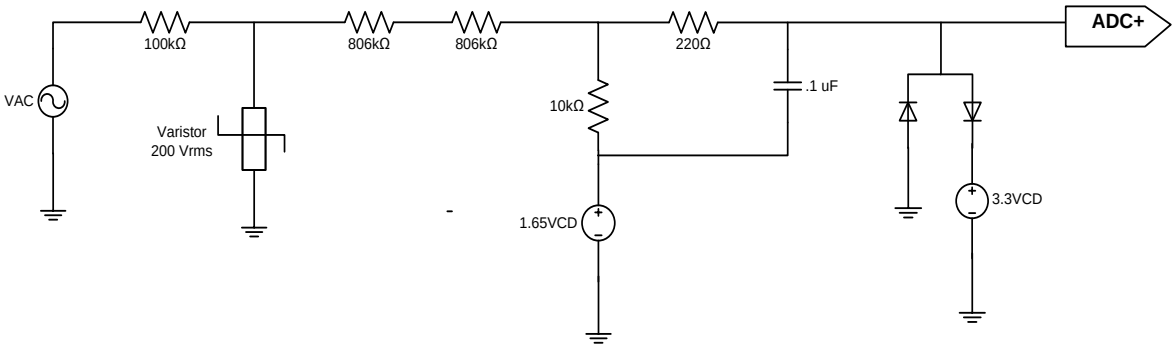


Figura 3. 7. Circuito de acondicionamiento de Tensión, línea viva.

Cabe mencionar que este circuito es únicamente para el acondicionamiento de la línea viva, se implementó el mismo circuito para el acondicionamiento del neutro, figura 3.8.

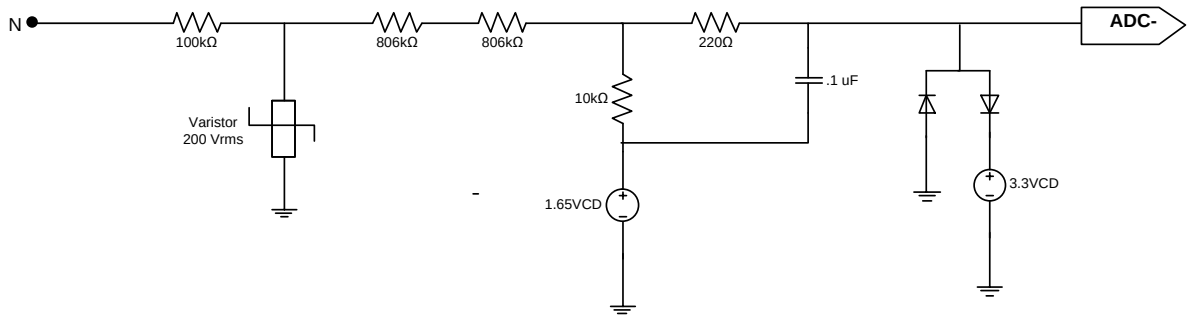


Figura 3. 8. Circuito de acondicionamiento de Tensión, hilo neutro.

Los dos circuitos forman el acondicionamiento de la señal tensión y se conectan a dos canales del ADC, los cuales están configurados como entradas diferenciales.

3.2.3.2 Circuito de acondicionamiento de corriente.

Este circuito pasa por el mismo proceso que el circuito de acondicionamiento de señal de tensión y hace uso de un transformador de corriente como instrumento de medición[23]. El transformador es capaz de medir una corriente máxima de 50 Amperes y tiene una relación de transformación de 1000:1, es decir que cuando circulan 50 Amperes por el primario por el secundario circularan 50 mA los cuales al pasar por una resistencia (llamada resistencia de Burden) produce una caída de tensión, la cual es proporcional a la corriente en el primario.

El circuito al igual que el circuito de tensión está configurado para que funcione en modo diferencial y se puede observar el esquemático en el apéndice C. En la figura 3.9 se muestra es circuito de acondicionamiento de corriente.

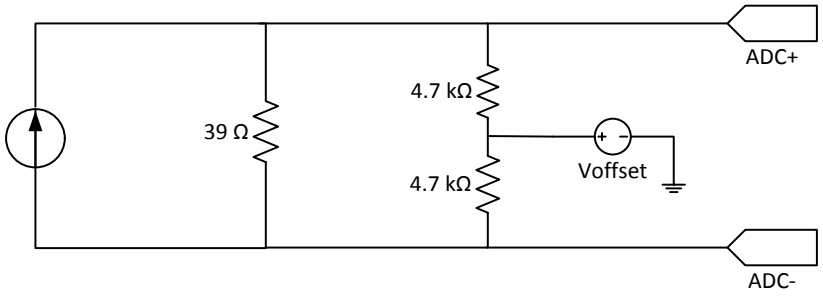


Figura 3. 9 Circuito de acondicionamiento de corriente.

Este circuito se implementó con dispositivos de protección capaces de recortar la señal cuando ésta alcance valores superiores a 3.3VCD o valores negativos.

3.2.3.3 Tarjeta de acondicionamiento de señales

En este PCB se encuentran integrados los circuitos de acondicionamiento tanto de tensión como de corriente, además de contar con las fuentes de alimentación que generan la tensión de desplazamiento, cuenta con comunicación serial: una para la comunicación con la pantalla, y cuenta con interfaz SPI para comunicación memorias SD.

En la figura 3.10 se puede observar el diseño final de dicha tarjeta.

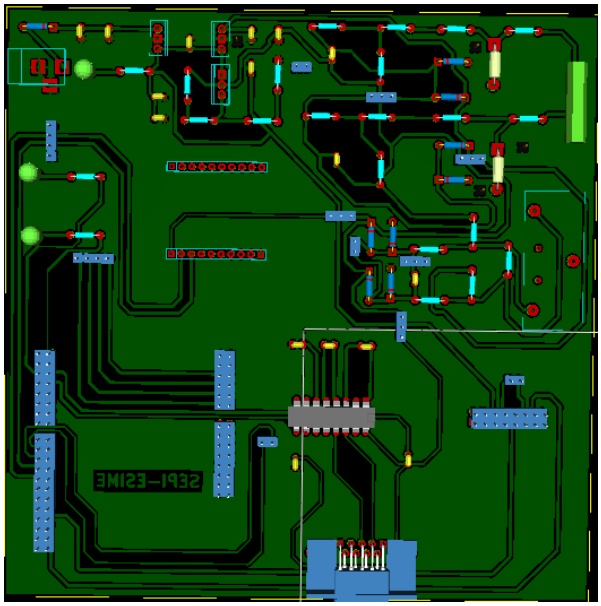


Figura 3. 10. Tarjeta de adquisición *de datos.*

3.2.4 Fuente Patrón.

En la realización de este trabajo, se utilizó la fuente "doble F2251", debido a que esta es capaz de entregar tensiones de CA variables desde 0 V hasta 300 V a una frecuencia de 60 Hz (Figura 3.11), y corrientes variables desde 0 A hasta 30 A de forma independiente, con resolución de mA por lo que fue utilizada para hacer el escalamiento de las variables así como para realizar pruebas de funcionamiento. Además con esta fuente es posible realizar desfasamientos ya sea de tensión o de corriente.

Figura 3. 11 Fuente Patrón.

3.2.5 Interfaz HMI.

La interfaz utilizada en este trabajo es una pantalla Touch Resistiva, figura 3.12, de la marca Dwin Technology [24]. En la Tabla 3.1 se describen sus principales características.

Tabla 3. 1 Características dela pantalla [25]

Tipo de LCD	TFT Resistiva	4 - cables
Resolución	800x600	Con rotación de 90°
Color	65K	16 bits de color 5R6G5B
Área Activa	162 mm x 121.5 mm	
Sd Slot	Formato Fat32	
RTC	1 s/24 H	
Tensión de Alimentación	6 – 42 VCD	Con protección contra polaridad inversa
Iluminación	LED	64 Niveles de ajuste
Consumo	5.4 W	12 V, 500 mA
Protocolo de Comunicación	RS232 y/o RS485	115200 baudios
Software de programación	Dwin DGUS	Última versión 4.9
Memoria para imágenes	224 imágenes	Únicamente imágenes .bmp
Memoria de Registros	256 Bytes	
Memoria RAM	56 Kb	

Las pantallas de esta marca se caracterizan por la facilidad y rapidez con la que es posible desarrollar un proyecto, ya que está basada en un protocolo de comunicación sencillo (comunicación serial), que combinado con un reducido conjunto de comandos y con el software Dwin DGUS evita el desarrollo de una gran cantidad de software, que de otra forma tendría que ser ejecutado por el microcontrolador.

Figura 3. 12 Pantalla Dwin.

3.3 Configuración de los beans del proyecto

En esta sección se hace la descripción del programa embebido en el KL25Z. En primer lugar se muestra la configuración del CPU, ya que de esta configuración dependen los demás periféricos del microcontrolador, también se muestran las configuraciones de los siguientes beans:

- Configuración del CPU.
- Convertidor Analógico Digital.
- Bean MQX lite.
- Comunicación Serial.
- Comunicación SPI.
- FAT32
- SDCARD

Además se muestra la descripción de las siguientes herramientas de MQX lite:

- Lightweight timers
- Lightweight events

3.3.1 Configuración del reloj del CPU

Este componente se configuró de tal forma que el microcontrolador trabaje a la máxima velocidad. A continuación se muestran los pasos necesarios para configurar el CPU a 48 MHz [26].

1. Dar doble clic en ProcessorExpert.pe en la pestaña de CodeWarrior Project. Seleccionar el componente CPU: MKL25Z128VLK4.
2. En la ventana Component Inspector se realizan las configuraciones marcadas en la figura 3.13. De esta manera el núcleo del microcontrolador trabaja a una velocidad de 48 Mhz y el Bus Clock trabaja a la mitad, es decir, a 24 MHz, que es la fuente de reloj de los demás periféricos.
3. Habilitar System Oscilator.
4. Seleccionar PEE en la opción de MCG Mode
5. Configurar a 96 MHz el PLL.
6. Configurar a 48 MHz el Core Clock.
7. Configurar a 24 MHz el Bus Clock.

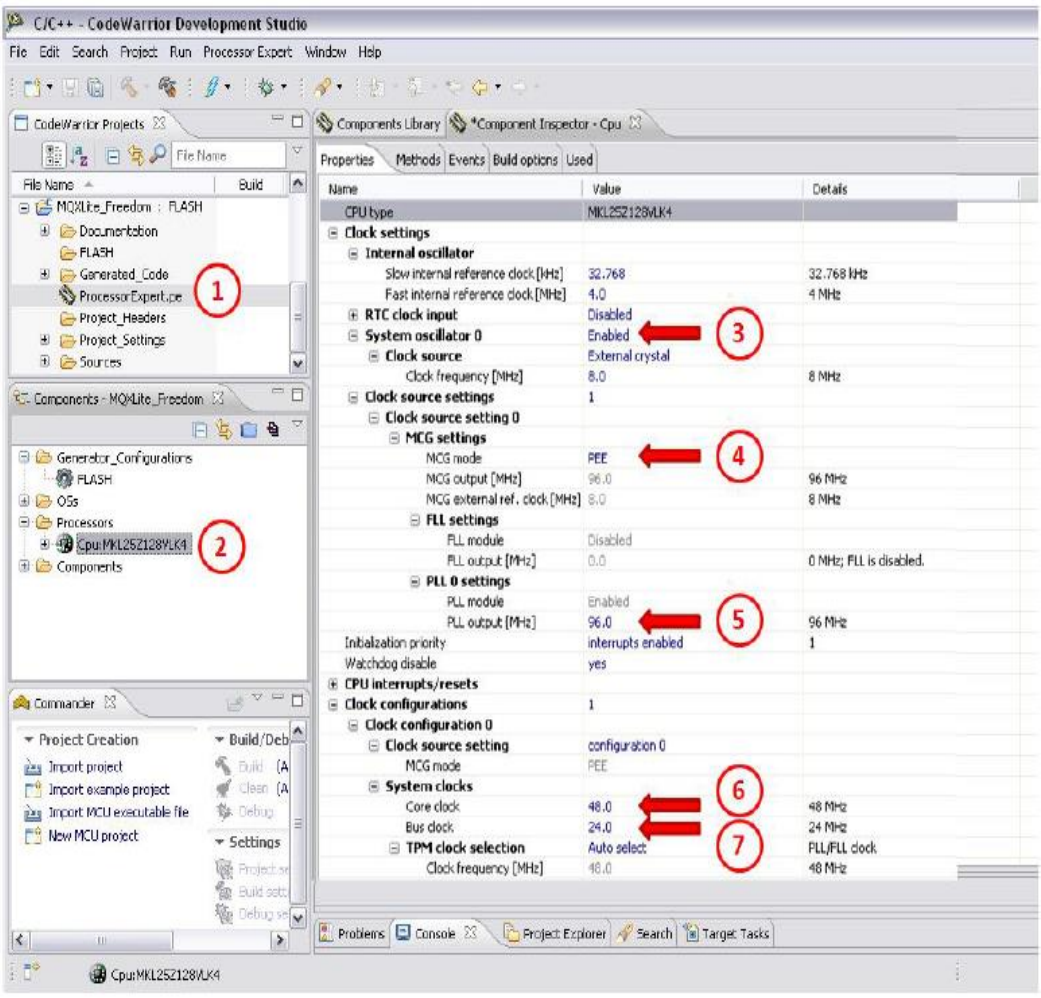


Figura 3. 13 Configuración del reloj.

8. Después de los cambios, es necesario generar el código usando el botón de la figura 3.14.

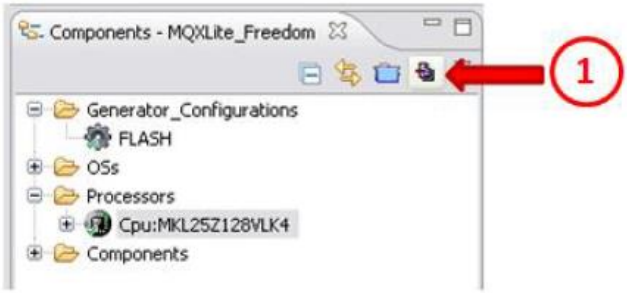


Figura 3. 14 Generación de código Processor Expert.

3.3.2 Configuración del Bean MQX Lite

Este bean está relacionado con el sistema operativo MQX Lite (figura 3.15) y es aquí donde es posible configurar:

1. El número de tareas.
2. Los componentes y herramientas que estarán disponibles en el proyecto.
   - ✓ Semáforos, eventos, timers, etc.
3. El espacio de memoria en bytes que utilizaran las interrupciones.

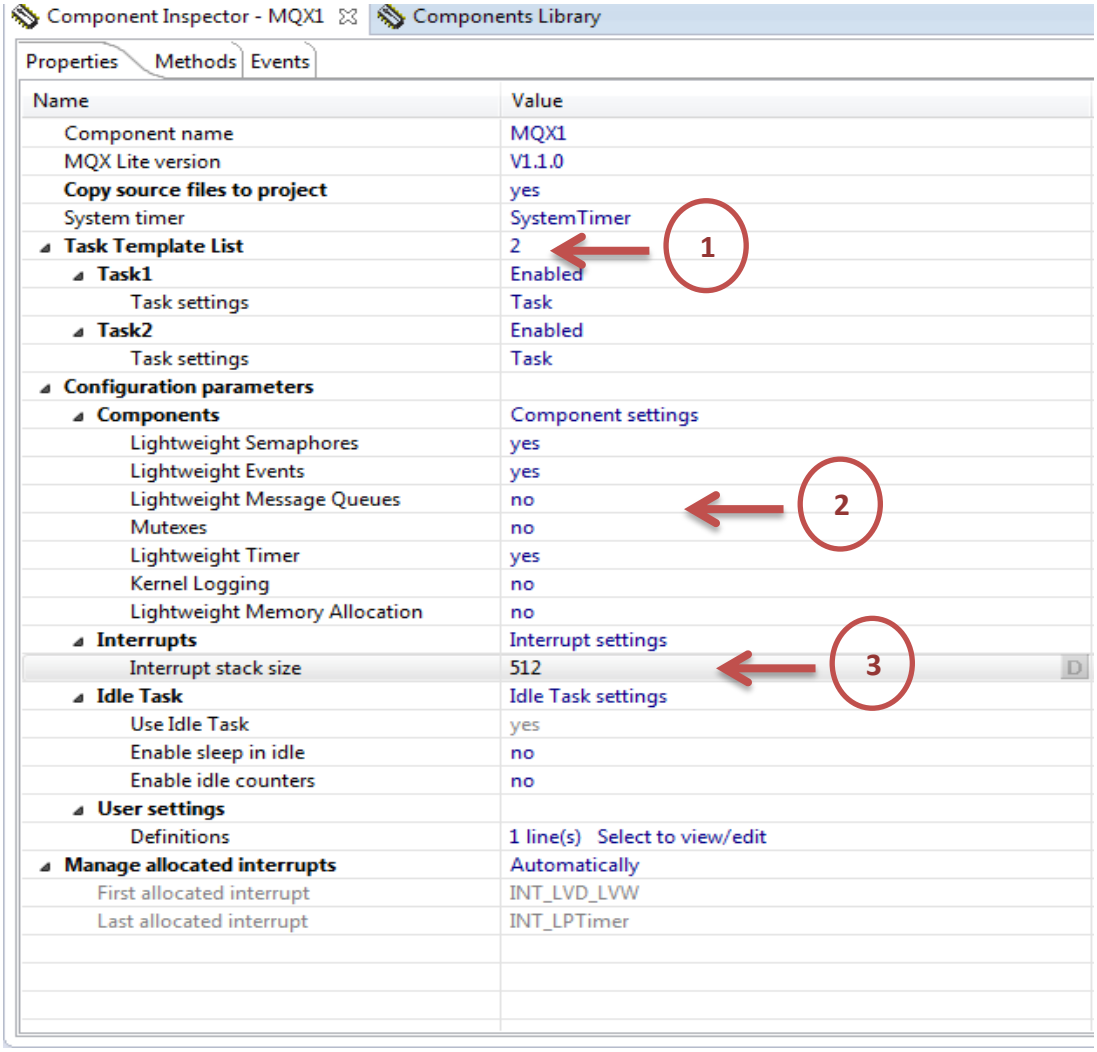


Figura 3. 15 Configuración del bean MQX lite.

3.3.3 Configuración del ADC.

El ADC es un periférico esencial, ya que este es el dispositivo encargado de recopilar información de los circuitos de adecuación de tensión y de corriente. Se utilizaron los beans LDD (Logical Device Driver) ya que estos son totalmente compatibles con MQX Lite, y donde es posible configurar:

1. Las interrupciones habilitadas.
2. El número de Canales del ADC.
3. El modo de trabajo: Modo Simple, o Modo Diferencial, en este caso Modo Diferencial.
4. Selección de los pines para medir.

En la siguiente imagen se muestran los pines utilizados por el ADC.

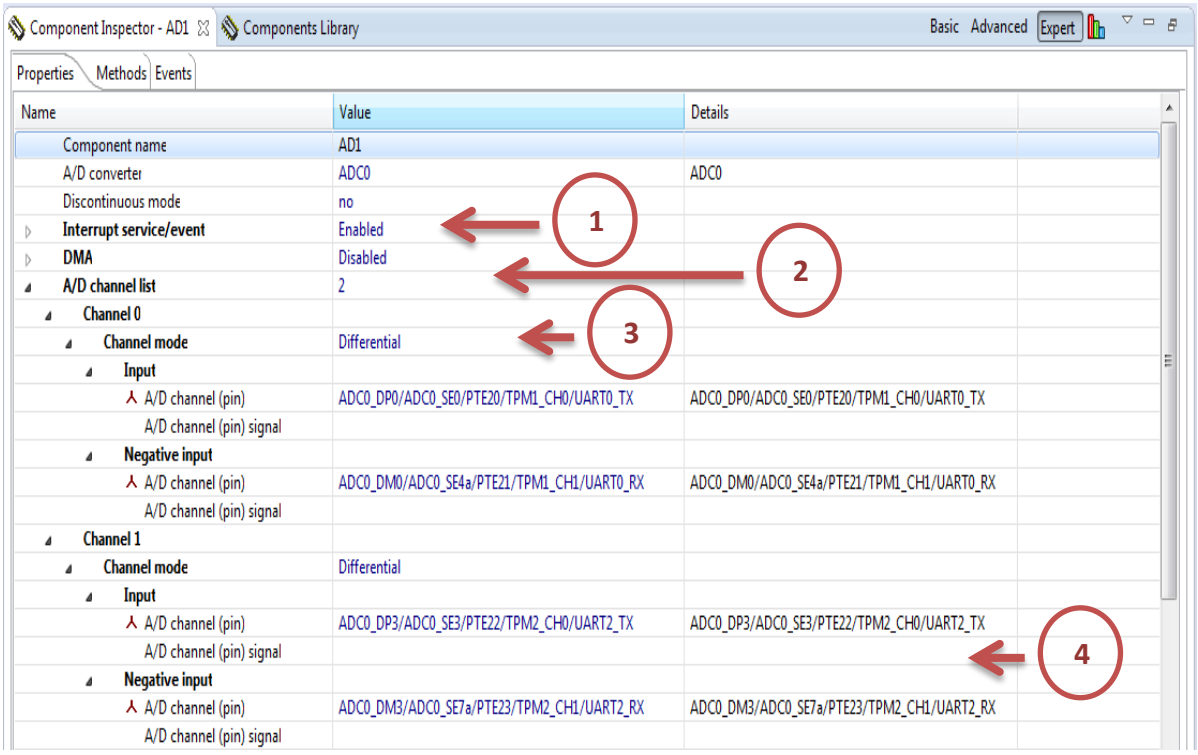


Figura 3. 16 Configuración del ADC

Como se puede observar en la figura 3.16, se escogieron dos canales, uno para la medición de tensión y otro para la medición de corriente. Los canales se encuentran configurados en forma diferencial, es decir, el ADC realiza la conversión de ambos canales, resta los valores obtenidos y guarda el valor obtenido en el registro de resultados correspondiente.

3.3.4. Configuración de la SD CARD

Para guardar información se hizo uso de una memoria SD Card de 2 Gigabytes de capacidad de almacenamiento. El medio de comunicación entre el microcontrolador y la memoria es el protocolo SPI, además de la utilización del sistema de archivos FAT32 [27].

El protocolo SPI es un estándar de comunicaciones usado principalmente para la transferencia de información entre un dispositivo digital llamado maestro, y un esclavo. Es decir es un protocolo maestro-esclavo diseñado para controlar varios periféricos utilizando el mismo bus de datos, figura 3.17.

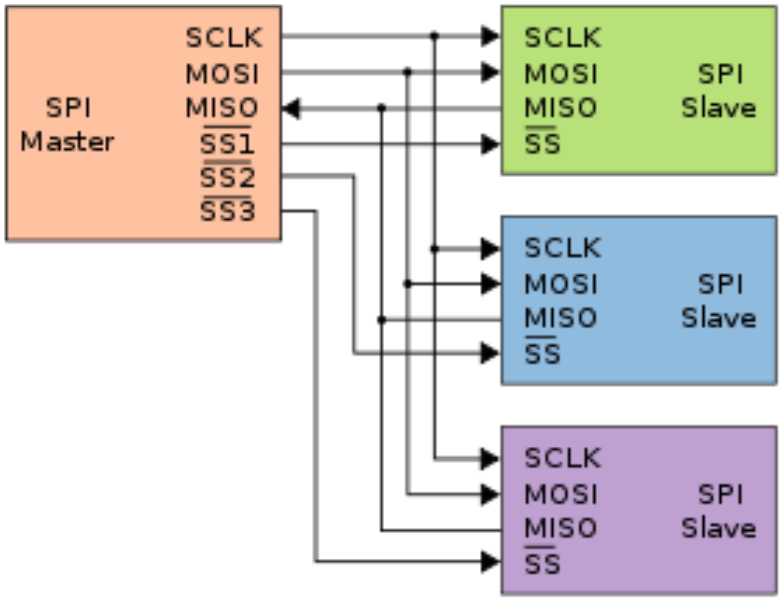


Figura 3. 17 Comunicación SPI. Maestro - Esclavos

Éste estándar consiste en un bus de cuatro líneas, sobre el cual se transmiten paquetes de información de 8 bits. Cada dispositivo conectado al bus puede actuar como transmisor y receptor al mismo tiempo por lo que este tipo de comunicación serial es full dúplex.

Las líneas de transmisión son:

- MISO. Master in, slave out. Bus de entrada de datos.
- MOSI. Master out, slave in. Bus de salida de datos.
- SCK. Señal de reloj.
- SS. Slave Select, señal de habilitación del esclavo.

Funcionamiento

Como ya se mencionó el protocolo de comunicación SPI consta de un dispositivo maestro y uno o más dispositivos esclavos. El maestro es el encargado de proveer la señal de reloj necesaria para realizar cualquier transferencia de información; el esclavo por lo tanto es dependiente del maestro y no puede comenzar una transferencia de información por sí mismo.

Al inicio de una comunicación, el maestro activa el reloj, a una frecuencia que comúnmente se establece en el rango de 1 MHz a 70MHz. Después el maestro pone a nivel bajo el Slave Select del dispositivo esclavo deseado y durante cada ciclo de reloj subsecuente ocurre una transmisión full dúplex de información:

1. El maestro envía un bit por la línea MOSI; el esclavo lee dicho bit por la misma línea, figura 3.18.

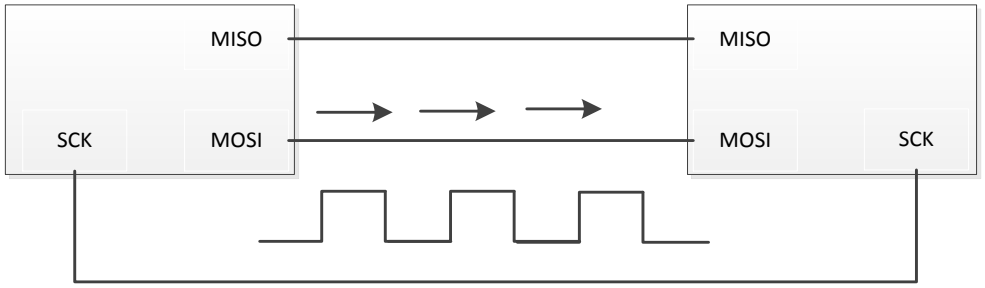


Figura 3. 18 Comunicación Maestro-Esclavo

2. El esclavo envía un bit por la línea MISO; el maestro lee dicho bit por la misma línea, figura 3.19.

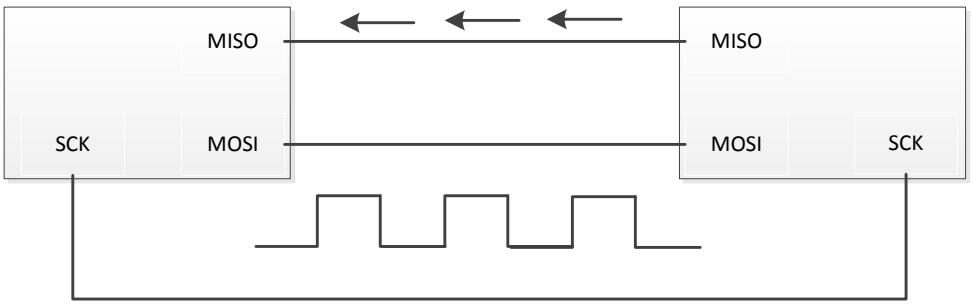


Figura 3. 19 Respuesta Esclavo-Maestro

Esta operación se repite a cada ciclo de reloj mientras exista información por transmitir, cuando la información se termina el maestro detiene la señal de reloj y normalmente deselecciona al esclavo con el cual se estaba comunicando. Es importante hacer notar que solo un esclavo puede estar activo.

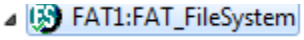
Sistema de Archivos FAT32

Tabla de asignación de archivos, comúnmente conocido como **FAT** (del inglés file allocation table), es un sistema de archivos desarrollado para MS-DOS, así como el sistema de archivos principal de las ediciones no empresariales de Microsoft Windows hasta Windows Me[28].

El sistema de archivos FAT32 es un formato popular para disquetes admitido prácticamente por todos los sistemas operativos existentes para computadora personal. Se utiliza como mecanismo de intercambio de datos entre sistemas operativos distintos que coexisten en la misma computadora, lo que se conoce como entorno multiarranque. También se utiliza en tarjetas de memoria y dispositivos similares.

Este trabajo utiliza el sistema de archivos FAT32 para crear archivos en la tarjeta SD, y en estos archivos guardar la información deseada. Se utiliza el componente de distribución libre: **FatFS file system**, con el cual es posible montar el sistema de archivos en el microcontrolador.

A continuación se muestran los componentes utilizados además de las configuraciones de dichos componentes [29].

Componente. 

Implementa el sistema de archivos FAT32 en el microcontrolador. Este componente es de licencia libre por lo que puede ser usado sin ninguna restricción. En la figura 3.18 se observa tanto los métodos como la configuración utilizada en el trabajo. Cabe resaltar las siguientes configuraciones:

1. El tamaño de cada sector : 512 Bytes
2. La habilitación de la opción de Long File Name, la cual permite asignar nombres a los archivos de hasta 255 caracteres.
3. La utilización del reloj TmDt1 como reloj de tiempo real.
4. La habilitación de una memoria como memoria de almacenamiento masivo.

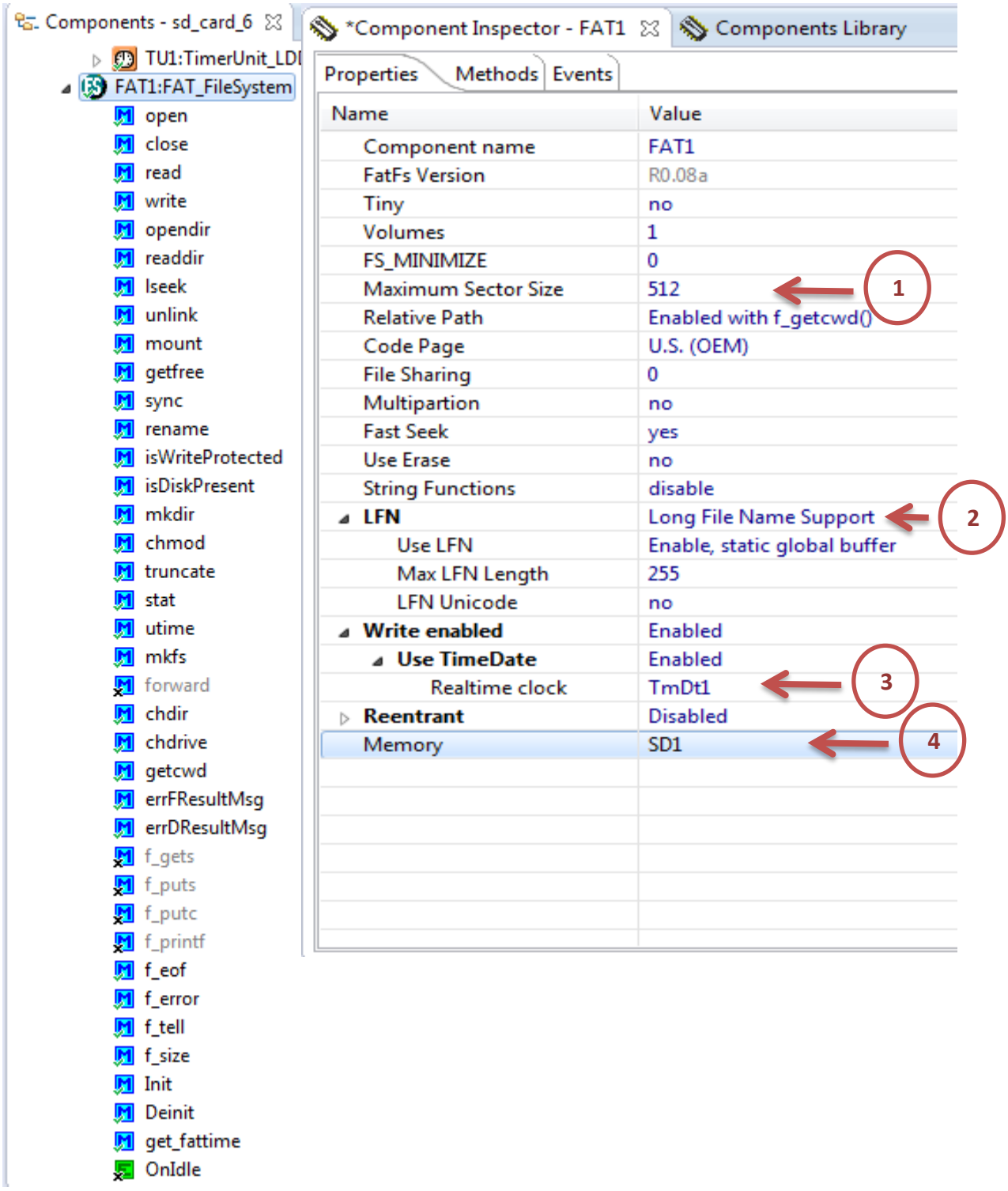


Figura 3. 20 Métodos y configuración del componente FAT_File_System

Con respecto a los métodos, a continuación se muestra la API de algunos de éstos métodos.

FAT1_mount(0, file_system); /* mount file system */

Esta función lo que hace es montar el Sistema de archivos en el microcontrolador, es decir, crear en el micro la capacidad de poder crear archivos en la memoria. Como parámetros esta función recibe:

1. EL primer archivo es el volumen que tendrá el sistema de archivos en el micro. En este caso el volumen es cero.
2. El segundo parámetro es un puntero al tipo de dato: **FATFS**

FAT1_open(file,name_file, FA_OPEN_EXISTING | FA_WRITE | FA_READ)

Este método como su nombre lo indica abre un archivo. Los parámetros que recibe esta función son:

1. Un puntero al tipo de dato FIL. Este tipo de dato (FIL) está definido en el archivo ff.h.
2. Una cadena con el nombre del archivo, esta cadena debe contener la extensión del archivo a crear.
3. El tercer parámetro es un conjunto de macros, los cuales especifica la forma en que se va a crear el archivo, los privilegios que tendrá el usuario sobre el archivo, es decir si el archivo será solo de lectura, si se puede escribir en el etc.

FAT1_read(&fp,(void*)data,size[i],&br);

Esta función lee un archivo. Los argumentos que recibe esta función son los siguientes:

1. Un puntero al tipo de dato FIL. Este tipo de dato (FIL) está definido en el archivo ff.h.
2. Direccion de memoria donde se guardara la información leída, Es decir la dirección de memoria del buffer donde se guardara la información leída del archivo.
3. El número de datos que se van a leer del archivo.
4. Puntero a un int. La función guardará en este puntero el número de bytes leidos por la función. Si éste número coincide con el parámetro anterior, significa que la función ha podido leer todos los bytes que el programador solicito.

f_close_file(&fp);

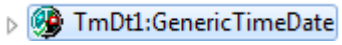
Esta función cierra el archivo abierto y el único parámetro que recibe es un puntero al tipo de dato FIL.

FAT1_write(&fp, data, sizeof data - 1 , &bw)

Esta función escribe en el archivo, y los parametros que recibe son los siguientes:

1. Un puntero al tipo de dato FIL. Este tipo de dato (FIL) está definido en el archivo ff.h.
2. La dirección de memoria donde se encuentra la información a escribir.
3. El número de datos a escribir.

4. Puntero a un int. La función guardará en este puntero el número de bytes escritos por la función. Si éste número coincide con el parámetro anterior, significa que la función ha podido escribir todos los bytes que el programador solicito.

Componente: 

. Este Componente permite establecer la hora y fecha del sistema. Cabe resaltar las siguientes configuraciones:

1. Tick time (ms). Define la resolución del reloj del sistema.
2. Permite configurar fecha y hora del sistema.

La principal función de este componente es incrementar el reloj del sistema cada 10 ms.

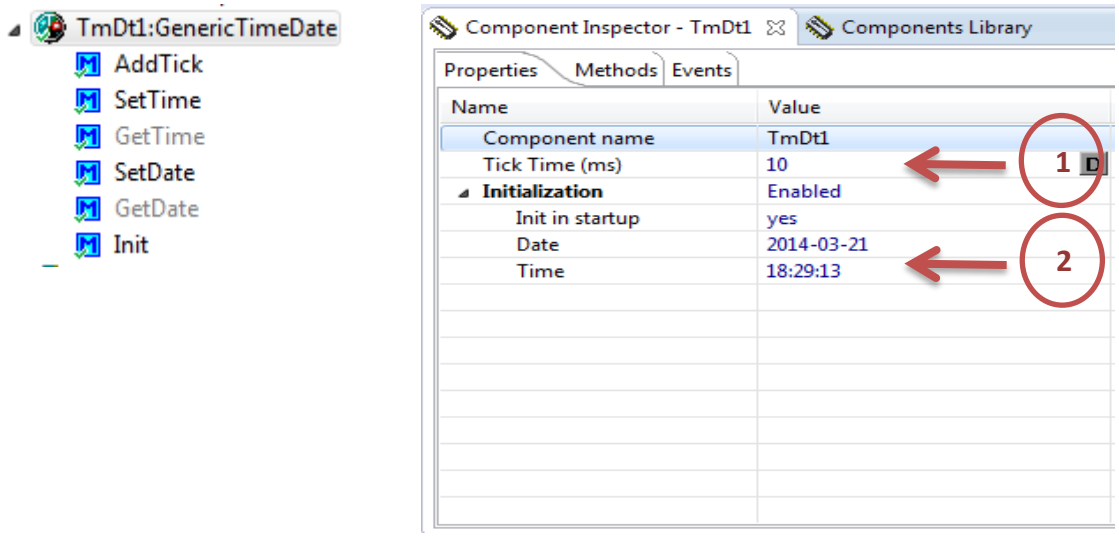
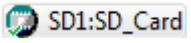


Figura 3. 21 Configuración del componente TmDt1.

Componente 

. Este componente es un driver para la memoria SD CARD. A continuación se enlistan las configuraciones hechas en este componente.

1. Habilitar HW SPI.
2. Seleccionar el componente SM1.
3. Habilitar el pin Slave Select.
4. Seleccionar los componentes: Wait y Timeout.

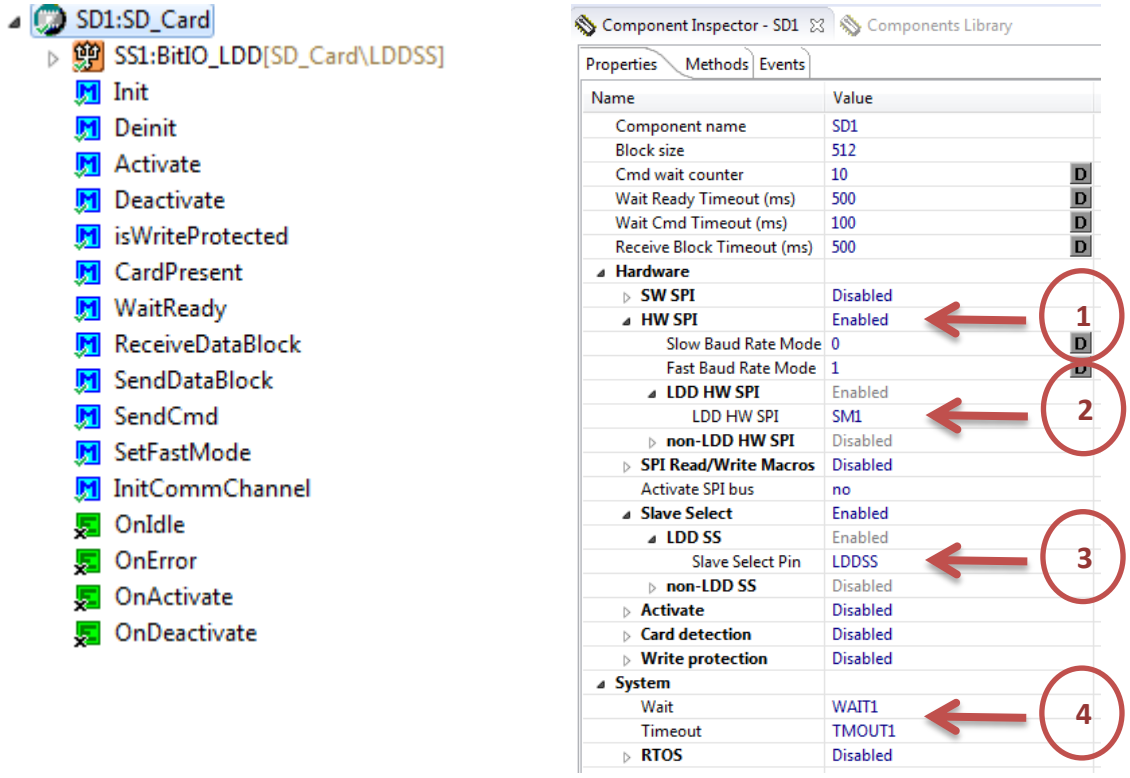


Figura 3. 22 Configuración del componente SD_Card.

Componente SM1:SPIMaster_LDD . Este componente para configurar el protocolo de comunicación SPI. A continuación se enlistan las configuraciones hechas en este componente:

1. Habilitar las interrupciones.
2. Configurar el pin MISO.
3. Configurar el pin MOSI.
4. Seleccionar el reloj
5. Configurar dos velocidades de trabajo: 375 kHz y 12MHz.
6. Configurar la opción Chip Select Toogle.

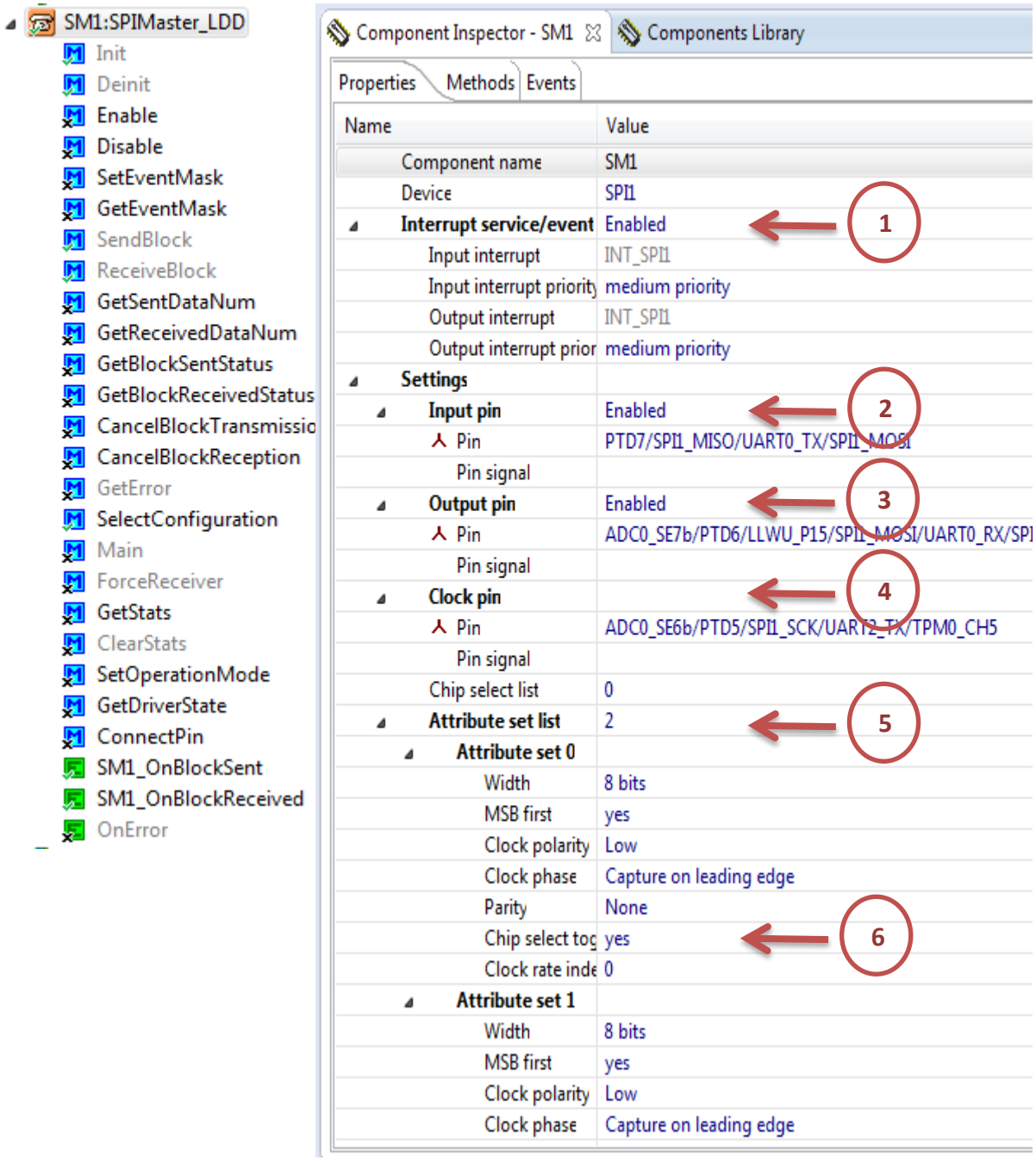


Figura 3. 23 Configuración del componente SPI_Master_LDD.

Componente TMOUT1:Timeout[Timeout] . Debido a que las operaciones sobre la memoria puede tomar cientos de milisegundos, este componente permite esperar por el tiempo especificado.

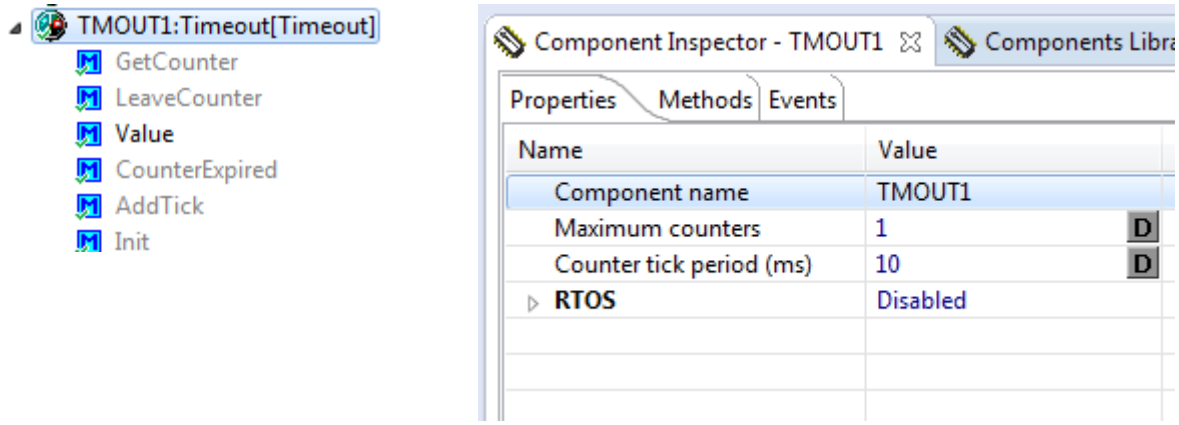
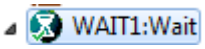


Figura 3. 24 Configuración del Componente Timeout.

Componente  Este driver implementa retardos de tiempo necesarios para la inicialización de la memoria.

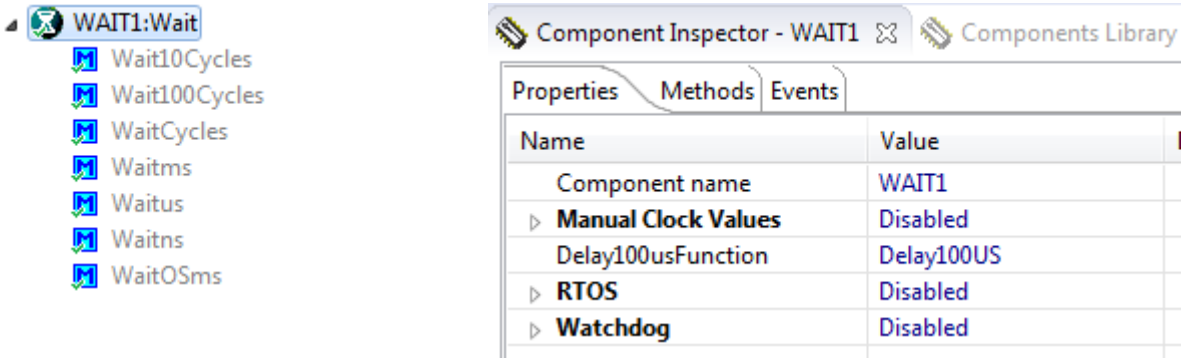


Figura 3. 25 Configuración del componente Wait.

Estos componentes tienen como métodos algoritmos que realizan retardos de tiempo al CPU, los cuales son necesarios al iniciar el protocolo de comunicación.

3.3.5 Configuración del puerto Serie

El puerto serie es el periferico utilizado para hacer la interfaz entre el microcontrolador y entre la pantalla. En la figura 3.26 se muestra las configuraciones del bean.

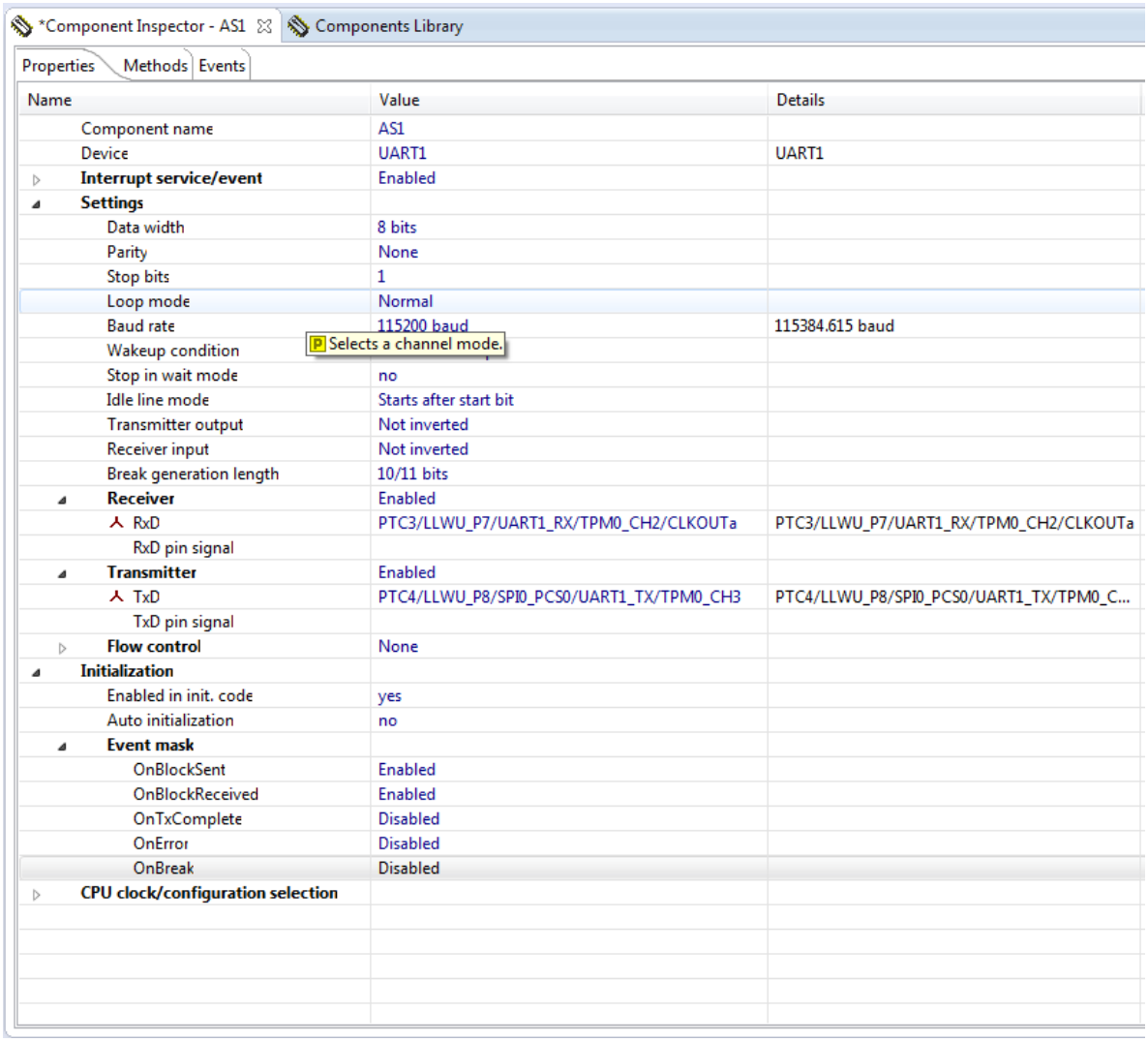


Figura 3. 26 Configuración del puerto serie.

La velocidad escogida está definida por la velocidad de transmisión que la pantalla soporta la cual es de 115200 Baudios.

3.3.6 Configuración del proyecto en Dwin

La creación de proyectos utilizando pantallas DWIN Technology consiste en la inserción de herramientas proporcionadas por el programa DGUS_SDK [30] a imágenes prediseñadas en cualquier software de edición de imágenes, ver Apéndice D.

Para crear un proyecto seguir el siguiente método.

1. **Abrir DGUS_SDK.**
2. **Configuración del proyecto**. Esta configuración consiste en indicarle al programa la resolución de la pantalla con la que se está trabajando, en este caso 800 x 600, además de indicarle la ubicación donde se guardara el proyecto.

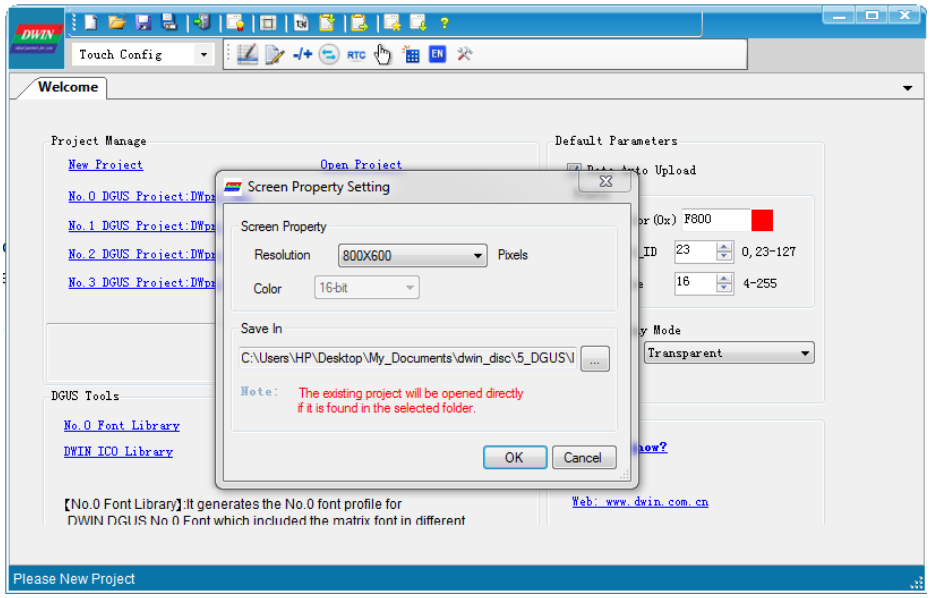


Figura 3. 27 Pantalla de inicio de DGUS_SDK.

Una vez configurado estos parámetros, la carpeta escogida para el proyecto, la carpeta ya contendrá información como: la carpeta que se utilizará para transferir el proyecto a la pantalla (DWIN_SET), el proyecto en sí, el cual tiene extensión *.hmi.

3. **Agregar imágenes al proyecto**. Presionar el botón Add el cual se encuentra en la parte superior izquierda del programa. El programa únicamente acepta imágenes *bmp, con 24 bits de resolución. El nombre de las imágenes debe ser por ejemplo “00_inicio”, y el número de imágenes es limitado por la resolución de las imágenes.Una vez agregadas todas las imágenes de las que constará el proyecto, ya es posible agregar las herramientas.
4. **Configuracion de las Herramientas**. Las herramientas agregadas dependerá de que es lo que el usuario desee mostrar.

Imagen 00.

Se tiene la primera imagen del proyecto.

Figura 3. 28 Imagen principal de pantalla.

En esta imagen se muestran cuatro botones, los cuales al ser presionados estos llevaran a otra imagen, por lo que para hacer esto se agrega la herramienta "Return Key Code".

Return Key code además de mostrar otra imagen, está es capaz de modificar una sección de memoria (VP, variable pointer) con el valor que el usuario desee (Key value). Por lo que en esta imagen se agregan cuatro objetos de este tipo con las siguientes configuraciones.

Tabla 3. 2 Configuración de las herramientas de la imagen 1

Botón	Imagen a Saltar	VP	Key Value
Mediciones	01_Mediciones	0x0000	0x0001
Graficas	02_graficas_slider	0x0000	0x0002
Historial	03_Historial	0x0000	0x0003
Configuracion	04_Configuracion	0x0000	0x0004

Y el proyecto se vería de la siguiente forma:

Figura 3. 29 Imagen 00 con las herramientas.

VP, Variable pointer, se refiere a una sección de memoria de 56Kb, la cual está destinada en este caso para almacenar el valor de key value en la VP asignadaa cada botón, es decir, cuando alguno de los botones es presionado, la pantalla guardará el Key Value en la VP que se configuro para cada botón (en este caso todos los botones tienen la misma VP.)

Imagen 01

A continuación se muestra cómo es que se configura la pantalla de **Mediciones** la cual tiene por objetivo mostrar los resultados de los algoritmos de medición que el microcontrolador ejecuta. Para mostrar información se hace uso de la herramienta "Data Var Display" la cual se encuentra en el menu de "Variable Config", y por lo tanto la pantalla de mediciones se veria de la siguiente manera:

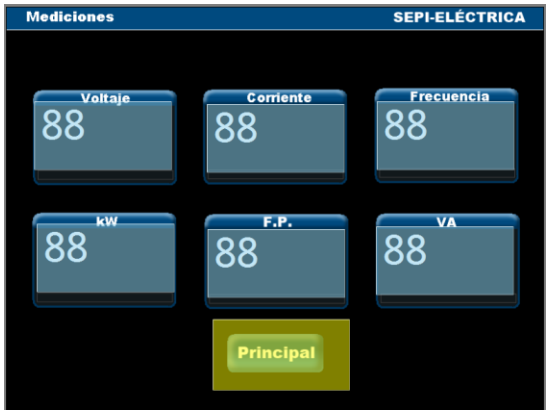


Figura 3. 30 Imagen 01.

Con las siguientes Configuraciones:

Tabla 3. 3 Configuración de las herramientas de la imagen 2

Objeto	VP	Number of Int bit	Number of Dec bit
Data Var Voltaje	0x0005	3	2
Data Var Corriente	0x0006	2	2
Data Var Frecuencia	0x0007	2	2
Data Var KW-h	0x0008	4	0
Data Var FP	0x0009	0	2
Data Var VA	0x000A	4	2
Return Key code Principal	0x0000	-	-

En este caso la el objeto Data Var Display ocupa la VP, como la fuente de donde leerá la información a mostrar, con 3 números enteros y dos decimales para el caso de Voltaje. Los Vp debe ser diferente, ya que de lo contrario todos los cuadros mostrarían exactamente el mismo número.

Imagen 02

Ahora se muestra la configuración de la imagen de gráficas. Esta pantalla se diseñó con el objetivo de poder observar las señales de tensión y corriente, por lo que es necesario agregar la herramienta de visualización de gráficos llamada Dinamic Curve Display. Esta herramienta funciona como si fuera un osciloscopio, el cual grafica una serie de puntos, los cuales los obtiene del ADC del microcontrolador.

La imagen seria:

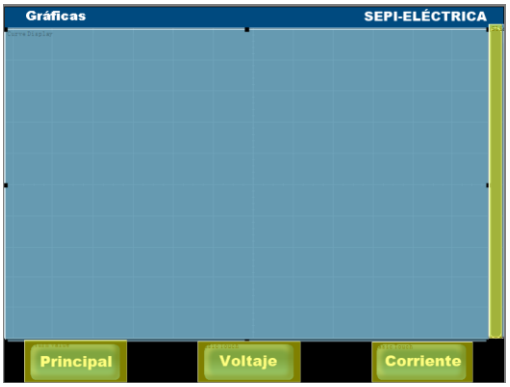


Figura 3. 31 Imagen 02

Con los siguientes objetos y configuraciones:

Tabla 3. 4 Configuración de Dinamyc Curve Display

Objeto	SP	Y_central	VD_central	MUL_Y	Channel	Axis Spacing
Dinamyc Curve Display "Tension"	5000	284	0	4	0	6
Dinamyc Curve Display "Corriente"	5500	284	0	0	1	6

Ahora se presenta la memoria SP, la cual representa la dirección inicial de los atributos del objeto al que se le asigno. Por ejemplo, de acuerdo a la siguiente tabla:

Address		Definition	Data Length	Description
0x00		0x5A20	2	
0x02		*SP	2	Stack pointer, default setting is 0xFFFF (set by Config. file).
0x04		0x000A	2	The whole process length (in terms of words).
0x06	0x00	0x0000	2	0x0000
0x08	0x01	Xs Ys Xe Ye	8	Scope of trend curve window, null if over range.
0x10	0x05	Y_Central	2	Center line coordinates of trend curve in Y-direction.
0x12	0x06	VD_Central	2	Trend curve value at center line, normally average of max & min value.
0x14	0x07	Color	2	Trend curve color.
0x16	0x08	MUL_Y	2	Magnification in Y-direction, by every 1/256, 0x0000 - 0x7FFF.
0x18	0x09:H	CHANEL	1	Chanel for trend curve, 0x00 – 0x07.
0x19	0x09:L	Dis_HOR	1	Transverse spacing between sample point, 0x01 – 0xFF.

Para cambiar el color de la curva de tensión, es necesario escribir en la dirección 0x5007 el color deseado.

Y_Central. Representa el centro de la curva

VD_Central. Representa el promedio del valor máximo y minimo de la curva

MULL_Y. Ganancia con respecto al eje Y.

Channel. Es posible graficar hasta ocho curvas, en este caso se configuro el canal 0 y 1 para graficar la señal de tensión y corriente respectivamente.

Horizontal Axis spacing. Representa el número de pixeles que existirá entre punto y punto a graficar.

Tabla 3. 5 Configuración de los botones de la imagen 02.

Objeto	VP	Imagen a saltar
Return Key code Principal	0x0000	00_Principal
Basic Touch Voltaje	-	02_Graficas_slider
Basic Touch Corriente	-	03_Graficass_slider.

Se presenta una herramienta más como es la de Basic Touch, y su función es cambiar de una imagen a otra sin modificar ninguna dirección de memoria de la pantalla.

Imagen 04.

Esta pantalla se diseñó para mostrar de forma gráfica el consumo de energía del usuario del último año. La imagen sería

Figura 3. 32 Imagen 03.

En esta imagen se hace uso de la herramienta llamada Basic Graph Display, la cual tiene la capacidad de poder dibujar dentro del área señalada cualquier tipo de figura, desde puntos pasando por rectas, círculos rectángulos, hasta rellenar figuras complejas.

A esta herramienta lo único que se configura es la VP. Y el botón de principal, el cual se configura como en las demás imágenes.

Una vez terminada la parte de diseño de las imágenes es necesario crear los archivos que en conjunto con las imágenes ya configuradas serán transmitidas a la pantalla via SD Card.

5. Para crear los archivos basta con guardar el proyecto de la siguiente barra
6. Crear los archivos: *.bin
7. Configurar los parámetros de la pantalla.

Presionar el botón y configurar la pantalla de acuerdo a la figura 3.33.

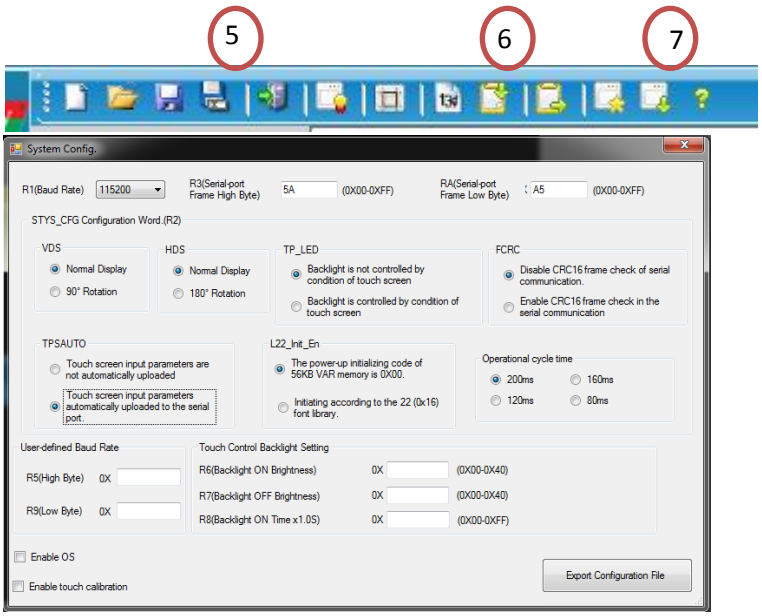


Figura 3. 33 Configuración de los parámetros de la pantalla

3.4 Herramientas de MQX Lite.

Mqx Lite [31] es un sistema operativo en tiempo real para microcontroladores de recursos limitados. En el Apéndice B se da una descripción de lo que es MQX lite y cómo funciona.

3.4.1 Diseño de una aplicación utilizando un sistema operativo como MQX lite.

En esta sección se propone el proceso para diseñar una aplicación utilizando un sistema operativo, así como las herramientas utilizadas en este trabajo.

1. Identificar las tareas e interrupciones que se implementaran.
2. Identificar lo que en sistemas operativos se le conoce como recursos. Es decir, el hardware o software que servirá como materia prima. Por ejemplo en éste trabajo se hace el muestreo de señales analógicas, por lo que el proceso de muestreo se traduce en un conjunto de números almacenados en una estructura de software conocida como arreglo. Este arreglo es el argumento de entrada para todos los algoritmos de medición por lo que sería el recurso del programa.
3. Conocer el sistema operativo, es decir, saber qué es lo que se puede y no se puede hacer con el sistema operativo, por ejemplo saber qué tipo de política maneja, si es posible manejar tareas con mismas prioridades etc.
4. Conocer las herramientas proporcionadas por el sistema operativo, es decir, saber qué es lo que hacen los eventos, timers, mutex etc.
5. Dado que una de las características de los sistemas operativos es de que el CPU, únicamente procesa una sola tarea, entonces se puede hacer una división del programa en tareas o actividades que debe cumplir y con base en esto jerarquizarlas.
6. La sincronización de tareas con otras tareas, o de tareas con interrupciones es un reto muy significativo en el diseño, por lo que se vuelve más importante conocer la política que maneja el sistema operativo. Sincronizar correctamente estas actividades permitirá que el CPU ejecute cálculos erróneos debido a la perdida de información.
7. Tener en cuenta la cantidad de memoria disponible, además de conocer que herramientas se tienen disponibles para la depuración del programa.

3.4.2 Lightweight Events

Los lightweight events son una herramienta de sincronización en MQX, la cual es opcional del SO y tiene que ser habilitada desde el bean MQX además de incluir la librería “lwevent.h” en el proyecto.

3.4.2.1 Funcionamiento.

Los lightweight events además de permitir la sincronización entre tareas, o entre tareas e interrupciones, éstos son capaces de controlar el flujo del programa. El componente de eventos consiste en grupos de 32 bits, los cuales son monitoreados por las tareas para poder ejecutar o no el código. En la figura 3.33 se muestra la arquitectura de los lwevents.

Los eventos tienen dos formas de funcionamiento como se puede observar en la figura: pueden esperar por cualquiera de los bits y la tarea se ejecutará o pueden esperar a que todos los bits sean puestos en uno y la tarea se ejecutará.

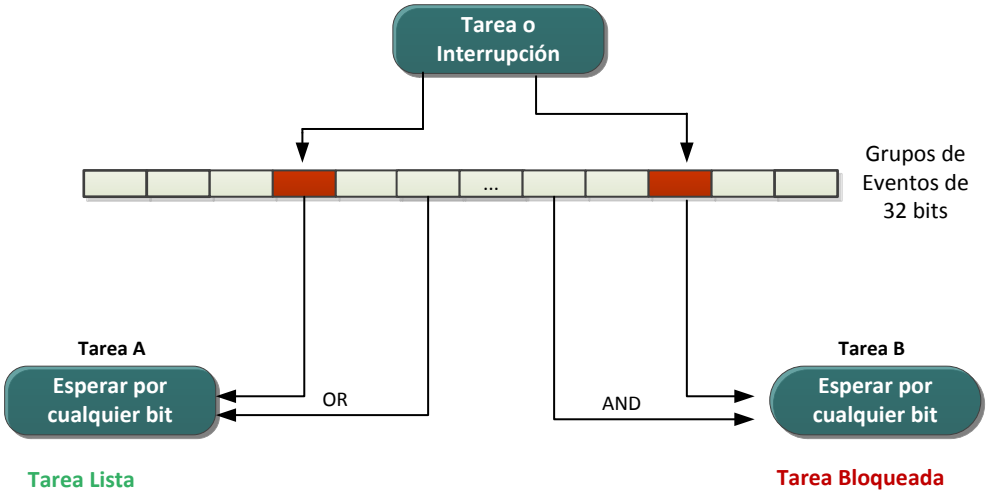


Figura 3. 34 Arquitectura de la herramienta de eventos [32].

3.4.2.2 Características.

- Los eventos constan de grupos de 32 bits.
- Las tareas pueden esperar por cualquiera de los bits o por una combinación de éstos (OR).
- Las tareas pueden esperar cuando todos los eventos (bits) han ocurrido (AND).
- Si el bit o el conjunto de bits que se están esperando, no están en 1, la tarea se bloquea.
- El evento bloquea la tarea si el tiempo de espera a que se pongan los bits expira.

Para crear un evento, en primer lugar se tiene que declarar el componente, y crear el componente eventos con las siguientes instrucciones.

```
LWEVENT_STRUCT my_event;

_lwevent_create(&my_event,LWEVENT_AUTO_CLEAR);
```

El segundo argumento de la función **_lwevent_create** es un macro que le indica a MQX que todos los bit se comportaran como bits de auto limpieza, esto quiere decir que cuando un bit es puesto en 1, este automáticamente se limpiara cuando la función siguiente sea ejecutada:

_lwevent_wait_ticks(&my_event,FIRST_BIT,FALSE,time)

El primer argumento de la función es la dirección de memoria de la variable LWEVENT_STRUCT, el segundo es el número de bits a esperar a que se pongan en uno. El tercer argumento corresponde a un valor booleano. El valor verdadero (TRUE) corresponde a que se deben esperar a que todos los valores del evento sean puestos en uno, el valor falso corresponde a que la función esperara a que cualquiera de los bits configurados por FIRST_BIT sean puestos en uno, y el último argumento es el numero en ticks que el evento esperara a que todos los eventos sean cumplidos, si este tiempo expira antes que los bits se hayan puesto en 1, entonces la tarea que contenga esa instrucción se bloqueara.

Hasta este punto se utilizan las funciones para verificar si el evento está en uno. La siguiente función sirve para poner en uno a los eventos.

_lwevent_set(&my_event,bits);

El primer argumento utiliza la dirección de memoria de la variable de eventos, posteriormente la siguiente variable representa el conjunto de bits que se pondrán en 1

En este trabajo se utilizan los light weight event con los siguientes propósitos

- Sincronizar las tareas con las interrupciones
- Controlar el flujo del programa
- Verificar que las conversiones no hayan rebasado el tiempo límite establecido.

3.4.3 Lightweight Timers.

Los lightweight timers (lwtimers) es un componente opcional de MQX Lite que proporciona la capacidad de interrumpir periódicamente una aplicación basada en este sistema operativo. Los lwtimers tienen distintos usos: desde controlar el tiempo de encendido y apagado de un led, activar alguna tarea en un tiempo específico, crear salidas PWM, hasta controlar la señal de disparo de algún periférico.

Esta herramienta debe ser habilitada desde la ventana Component Inspector del Bean MQX lite, y además agregar la librería **lwtimer.h**, la cual incluye los tipos de datos y declaración de funciones para el funcionamiento de la misma.

3.4.3.1 Funcionamiento.

Este componente crea una estructura de datos llamada cola, la cual se caracteriza por el periodo y el tiempo de retraso de su ejecución. A esta estructura se le agregan los timers que se deseen y estos se

ejecutan secuencialmente dependiendo del tiempo de retardo que se les configure. En la figura 3.35 se puede observar la arquitectura de esta herramienta, se puede ver que en una aplicación MQX lite es posible tener el número de lwtimers que se deseen (dependiendo de la memoria) y que cada estructura puede tener un periodo y número de timers independiente.

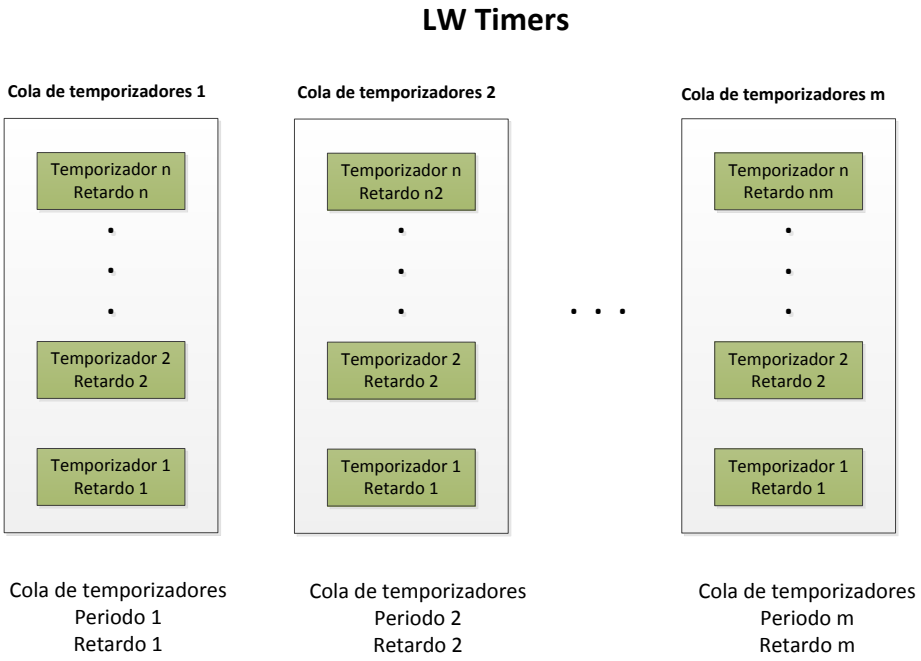


Figura 3. 35 Arquitectura de los LWTimers[33].

Los temporizadores tienen asociados alguna función que el CPU ejecutará cuando el tiempo programado expire, en la figura 3.36 se puede observar esta relación.

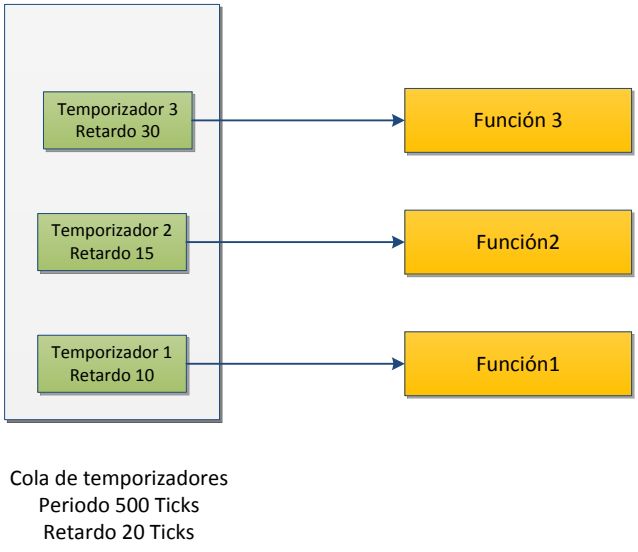


Figura 3. 36 Relación de los temporizadores con funciones.

En la figura 3.37 se bosqueja el funcionamiento de los temporizadores. Por ejemplo la función 1 entraría en el tiempo 530 mientras que la función 2 y 3 entrarían en funcionamiento en 535 y 550 respectivamente.

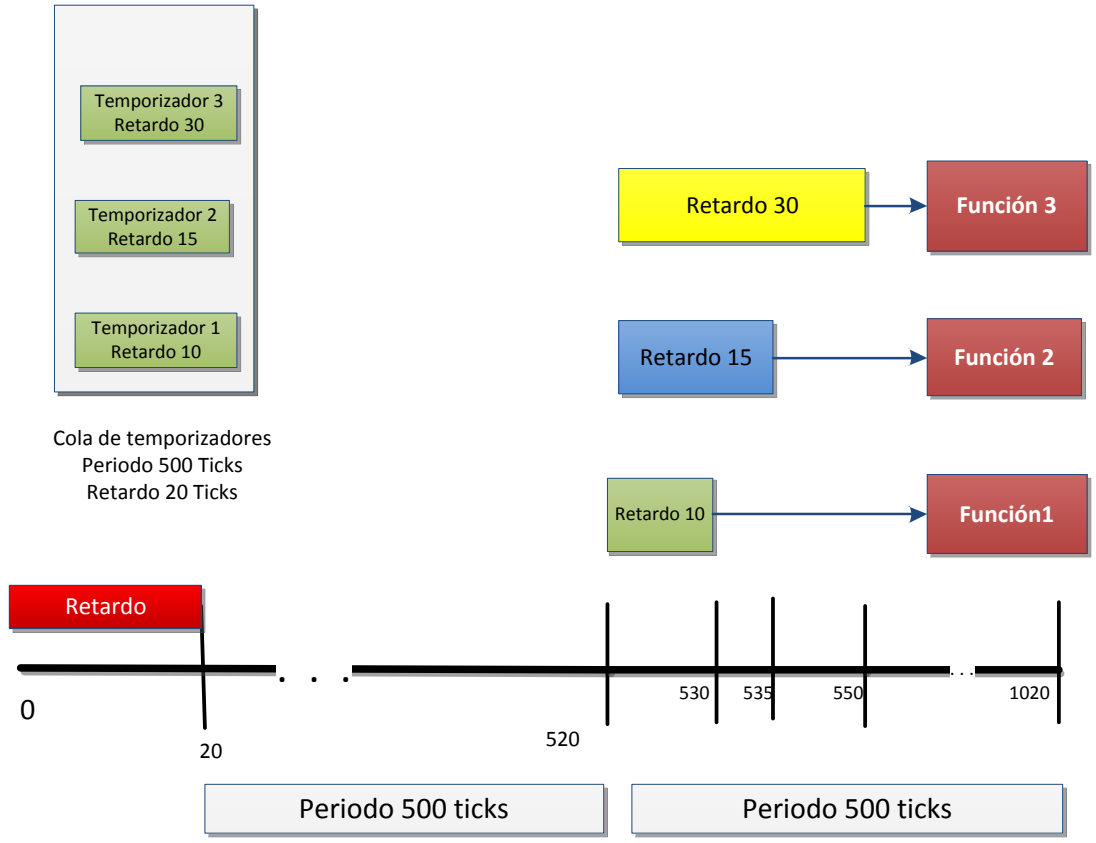


Figura 3. 37 Funcionamiento de los temporizadores.

El reloj en el que están basados los lightweight timers es el reloj proporcionado por MQX llamado System Timer, y la frecuencia establecida por este reloj es la frecuencia que se ejecutara dicha cola, y por lo tanto los timers que ésta tenga integrados.

3.4.3.2 Características de la Cola periódica.

Como se mencionó el periodo de la cola está en función de la frecuencia del reloj System Timer, pero además depende de un factor que multiplica dicha frecuencia, y determina de forma definitiva el periodo de la estructura de datos.

Para crear una cola se utiliza el siguiente tipo de dato.

LWTIMER_PERIOD_STRUCT adc_timer_queue;
Y la siguiente instrucción crea la cola periódica:

_lwtimer_create_periodic_queue(&adc_timer_queue,sampling_period,wait_ticks);

El primer parámetro es la dirección de memoria de la estructura de la cola periódica, el segundo parámetro es el factor por el que se multiplica el periodo del System Timer. El tercer parámetro representa el tiempo (en ticks) a esperar antes iniciar a procesar la cola de timers.

3.4.3.3 Características de los temporizadores.

Los timers tienen la misma frecuencia de operación que la cola. Para declarar dos timers se utilizan las siguientes instrucciones.

LWTIMER_STRUCT adc_ch0;
LWTIMER_STRUCT adc_ch1;
Y para agregarlas a la cola se utiliza la siguiente función:

```
_lwtimer_add_timer_to_queue(&adc_timer_queue,&adc_ch0,adc_delay_ch0,(LWTIMER_ISR_FPTR)measure,(pointer)0);
_lwtimer_add_timer_to_queue(&adc_timer_queue,&adc_ch1,adc_delay_ch1,(LWTIMER_ISR_FPTR)measure,(pointer)1);
```

El primer parámetro de esta función es la dirección de memoria de la cola a la que se va a agregar el timer, el segundo parámetro es la dirección de memoria del timer, el tercer parámetro representa un retardo de tiempo (tiempo programado en ticks), que esperara el timer a ejecutar la función indicada en el cuarto parámetro, el cuarto parámetro es la dirección de memoria de la función a llamar, y el último parámetro, es el argumento que se le pasara a la función.

Ya que las mediciones eléctricas consisten en el muestreo periódico de las señales, en este trabajo se utilizan los lightweight timers con los siguientes propósitos:

- Determinar la frecuencia de muestreo del ADC para obtener 64 muestras por ciclo.
- Controlar los disparos de cada uno de los canales del ADC.
- Compensar el ángulo de defasamiento provocado por elementos inductivos y capacitivos contenidos en el circuito de adecuación de señales.

3.5 Descripción del Software del Proyecto.

El programa consiste en un conjunto de tareas e interrupciones interrelacionadas mediante las herramientas de MQX lite. A continuación se describe la función que tiene cada una de estas tareas e interrupciones:

Measurement_task**.**	Esta tarea tiene la función de inicializar los periféricos, tales como el ADC, el UART, también de crear los timers y eventos del sistema y su función más importante es la de ejecutar los

	algoritmos de medición.
Frecuency_Task**.**	Esta tarea se encarga de calcular la frecuencia de la señal de tensión.
Serial_Task	Configura al puerto serie para recibir información de la pantalla.
Scale_Task	Tiene la función de ejecutar las funciones de escalamiento.
Graph_Screen_Task	Muestra de forma gráfica, las señales de tensión y corriente en la pantalla.
Write_Screen_Task	Tiene la función de escribir en la pantalla las mediciones hechas por los algoritmos de medición.
History_Task	Muestra el historial de consumo hecho por el usuario durante un año.
Date_task	En esta tarea se implementa un contador, el cual incrementa su valor cuando todas las demás tareas están bloqueadas.
Frecuency_task	Esta tarea tiene la función de calcular la frecuencia de la señal de tensión.
Create_file	Esta tarea tiene la función de crear un nuevo archivo de datos, cada 24 horas.
Write_file	Esta tarea tiene la función de escribir los kWh consumidos en un lapso de una hora.
Billing_task	Esta tarea tiene la función de mostrar el consumo de energía del último periodo de 60 días.
AD1_OnMeasurementComplete	Guarda el valor digitalizado en el buffer respectivo de tensión o corriente.
AS1_OnBlockReceived	Recibe datos vía Puerto serie y active tareas bloqueadas, dependiendo del valor recibido

En la figura 3.388 se observa un diagrama general del programa.

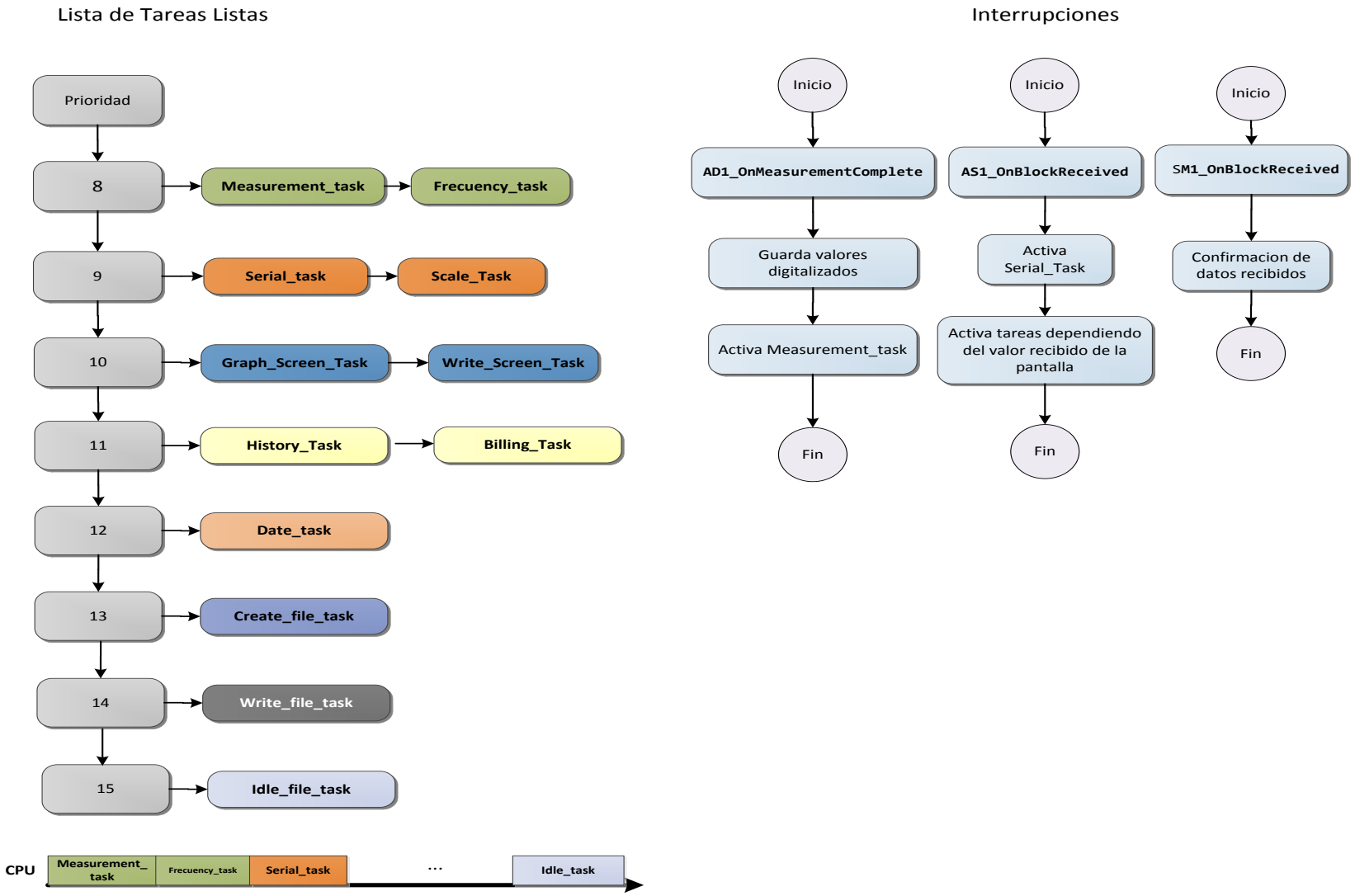


Figura 3. 38 Diagrama del programa principal.

3.5.1 Descripción de las tareas del programa.

En esta sección se muestra el diagrama de flujo de cada una de las tareas así como de las interrupciones.

3.5.1.1 Measurement_task.

Esta terea además de cumplir con la ejecución de los algoritmos de medición inicializa los periféricos utilizados en el proyecto, tal como, el ADC, el puerto serie, crea los eventos y timers.

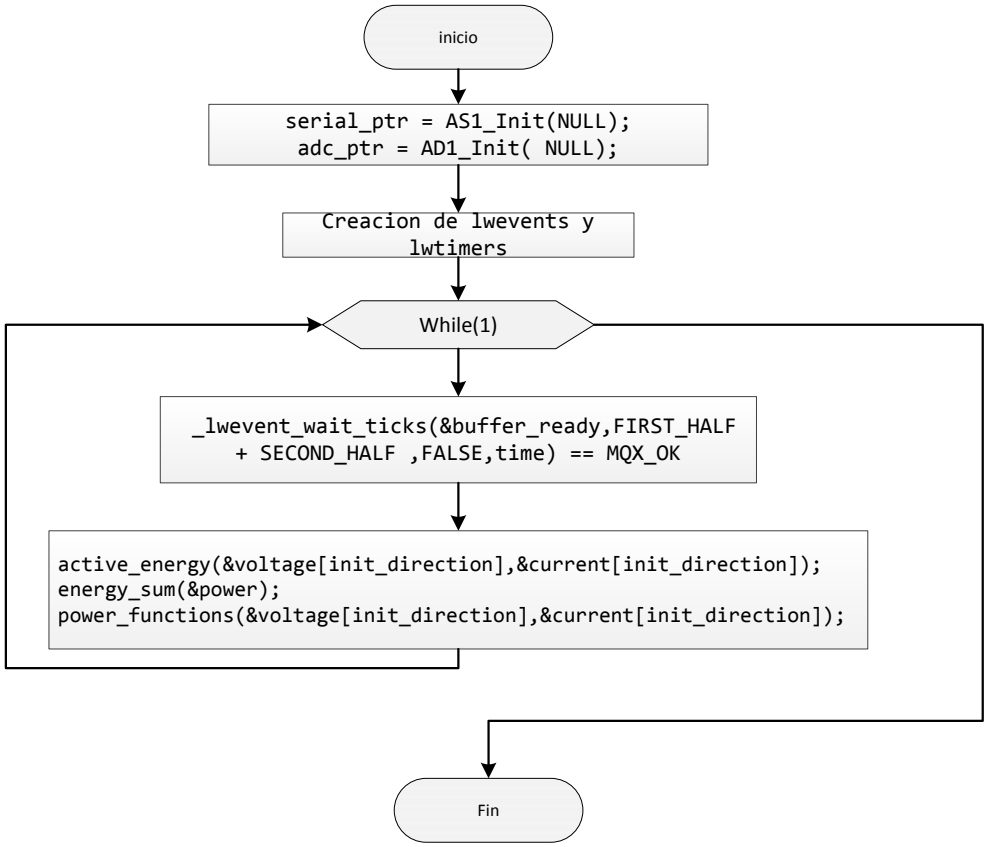


Figura 3. 39 Diagrama de flujo: Tarea de Mediciones.

Como se puede ver en el diagrama de flujo de la figura 3.39 las tareas se encuentran en un ciclo infinito por lo que se podría suponer que el proceso nunca terminaría, sin embargo debido a las características del sistema operativo y a la herramienta de eventos, esta tarea se bloqueará si no se cumple la condición de que los arreglos estén llenos y podrá ejecutarse las líneas de código siguientes.

A continuación se muestran los diagramas de flujo de las rutinas de mediciones.

Active_energy: La función de energía activa recibe como parámetros, las direcciones iniciales de los arreglos de tensión y corriente, y ejecuta el algoritmo de la ecuación 2.8. Posteriormente este valor lo guarda en Power.act_energy que es la variable utilizada para calcular la potencia activa por ciclo, mientras Power.Total_Act_energy, es la energía activa de todos los ciclos.

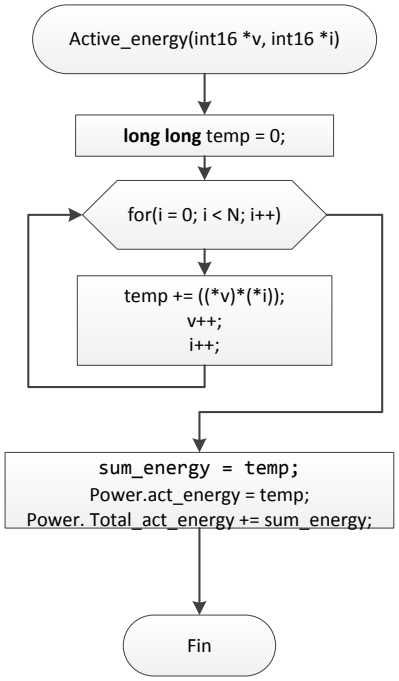


Figura 3. 40 Diagrama de flujo: Energía Activa.

Energy_sum. Esta función contabiliza el número de Kw-h consumidos o también el número de Kw-h aportados a la red.

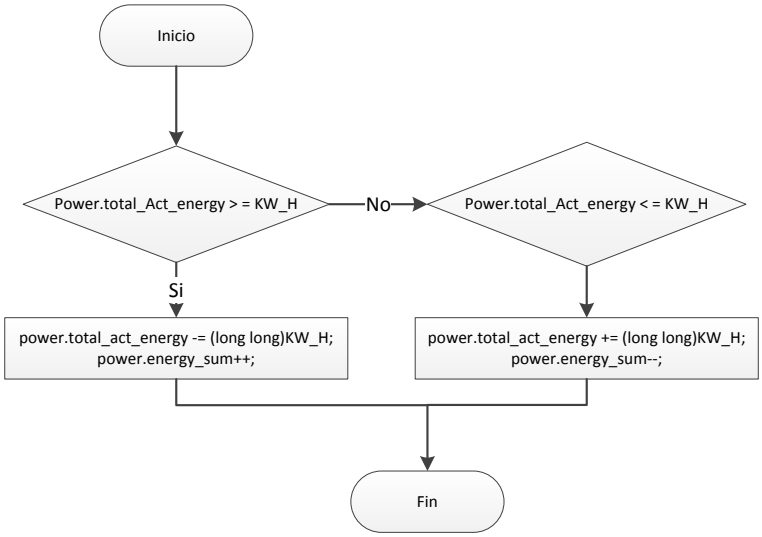


Figura 3. 41 Diagrama de flujo: Sumatoria de Energía.

Power_functions. Esta función recibe como parámetros las direcciones de los arreglos de tensión y corriente, y calcula:

- Tensión RMS, Corriente RMS.
- Potencia aparente.
- Factor de potencia.
- Potencia Activa.
- Potencia Reactiva.

Valores Eficaces. Para calcular tanto el valor de tensión o corriente eficaces se utilizó la misma función, la cual recibe como parámetro la dirección de memoria del primer elemento del arreglo de la señal. Esta función devuelve el valor RMS de la señal.

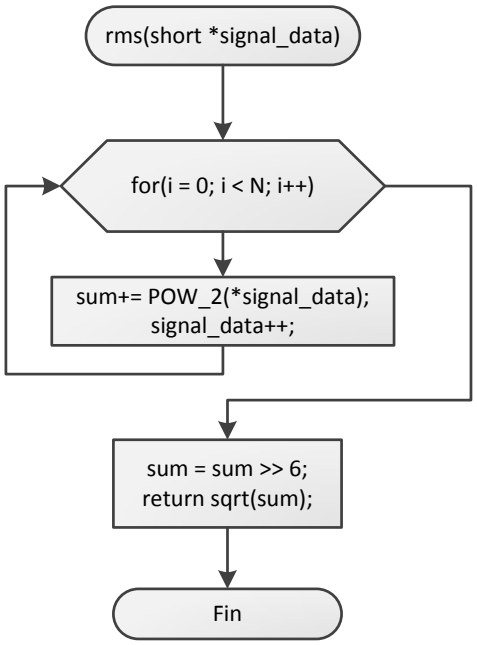


Figura 3. 42 Diagrama de flujo de Valores Eficaces.

La potencia aparente de acuerdo a la ecuación 2.12, es el resultado de la multiplicación de los valores eficaces de tensión y corriente. La potencia activa se puede calcular con la ec. 2.10, y el factor de potencia como la división entre P y S. A continuación se muestra el diagrama de flujo de esta función.

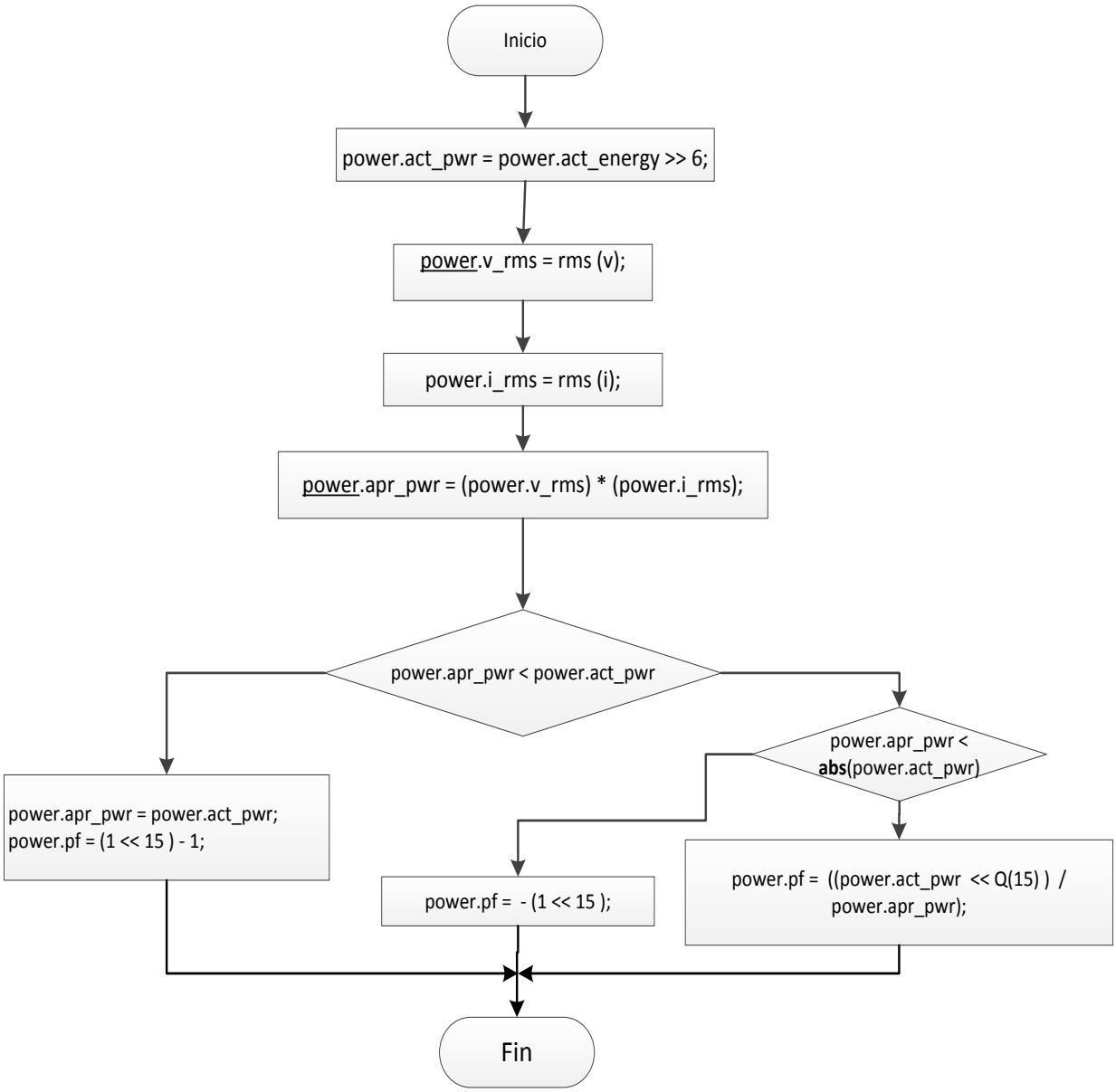


Figura 3. 43 Diagrama de flujo: Power_function.

3.5.1.2 Serial_task.

La función de esta tarea es la configurar al puerto serial para que reciba información via puerto serie. Esta tarea se encuentra bloqueada la mayor parte del tiempo lo que reduce carga de trabajo al CPU. En la figura 3.44 se muestra el diagrama de flujo de dicha tarea.

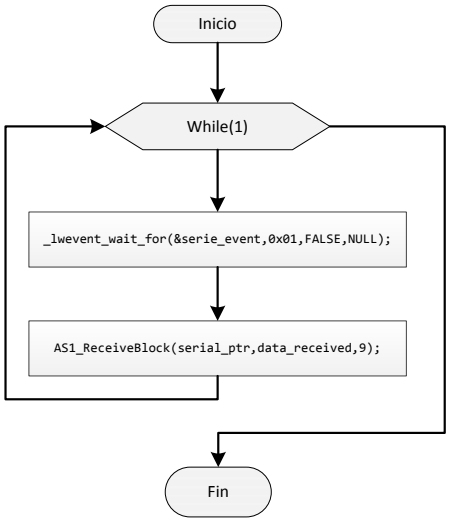


Figura 3. 44 Diagrama de flujo: Serial_Task.

3.5.1.3. Scale_task.

Esta tarea escala los resultados arrojados por los algoritmos de medición. Esta función se ejecuta cada segundo, figura 3.45.

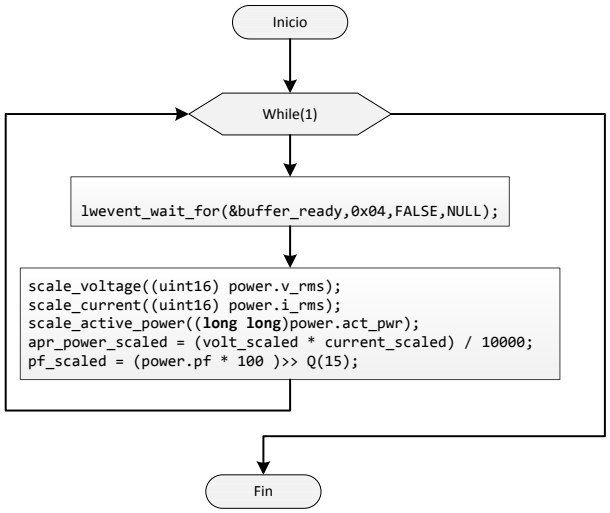


Figura 3. 45 Diagrama de Flujo Scale Task.

3.5.1.4 Graph_screen_task.

Esta función se encarga de graficar los valores digitalizados por el ADC. Esta función se encuentra la mayor parte del tiempo bloqueada y únicamente se ejecuta cuando se presiona el botón de Gráficas.

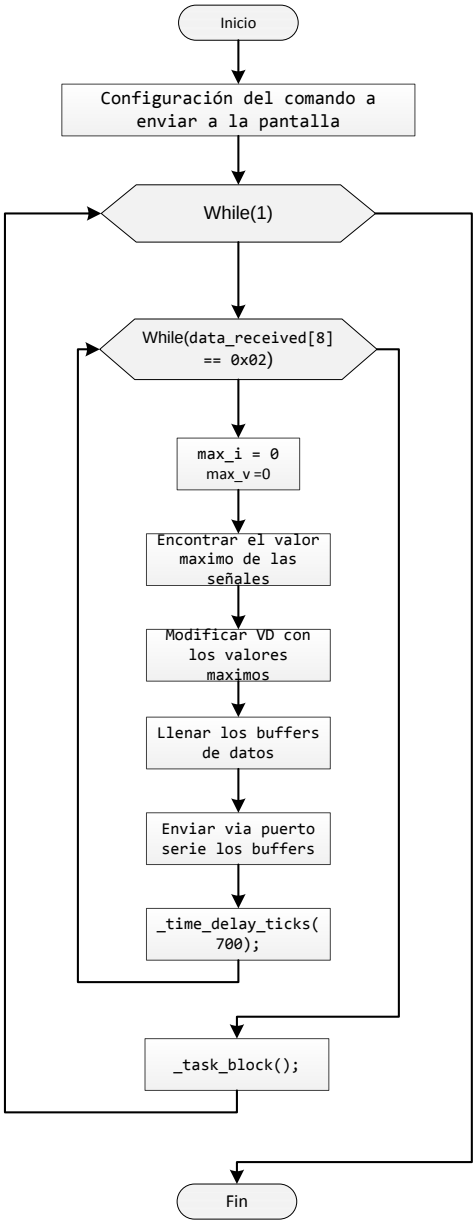


Figura 3. 46 Diagrama de flujo: Graph_screen_task.

3.5.1.4 Write_screen_task.

Esta tarea escribe los valores escalados de las tensiones eficaces y únicamente se ejecuta cuando se presiona el botón de mediciones.

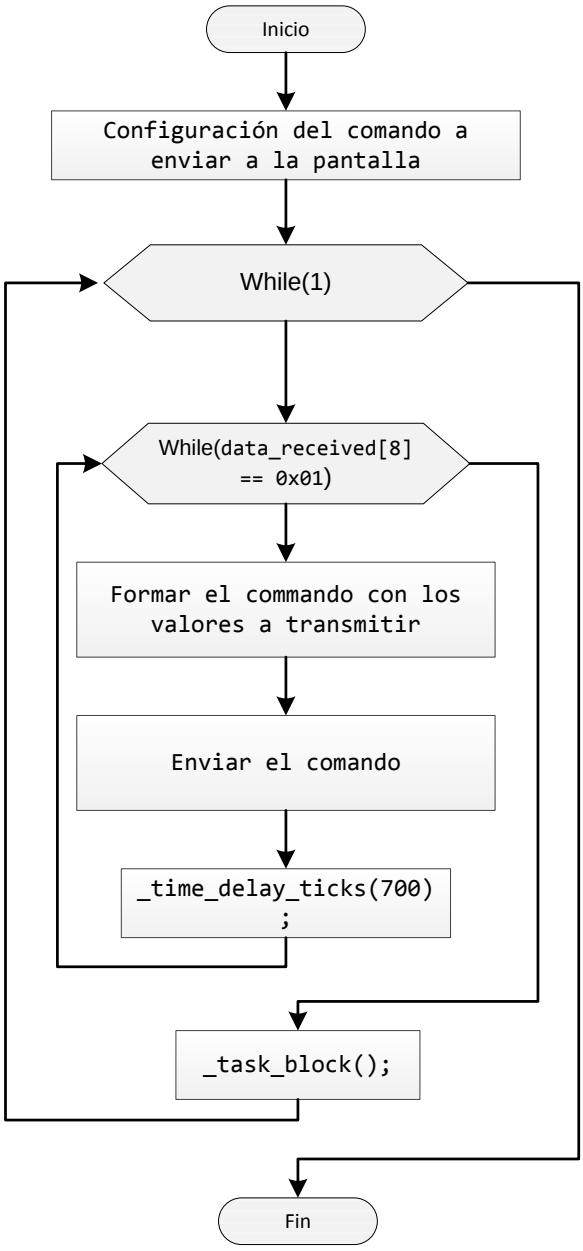


Figura 3. 47Diagrama de flujo: Write_screen_task.

3.5.1.5 Create_file_task.

Esta tarea hace uso del sistema de archivos FAT32 el cual permite la creación de archivos de datos en el cual se escribe la bitácora de consumo por día.

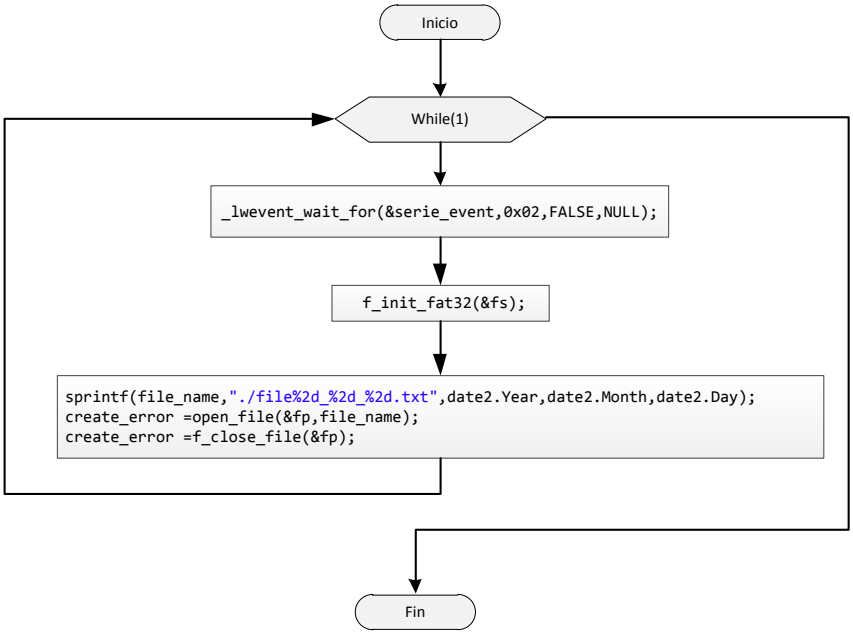


Figura 3. 48 Diagrama de flujo Create_file_task.

3.5.1.6 Write_file_task.

Esta tarea, tiene la función de crear la bitácora de consumo. Es decir esta tarea tiene el objetivo de escribir la cantidad de energía en (kWh).

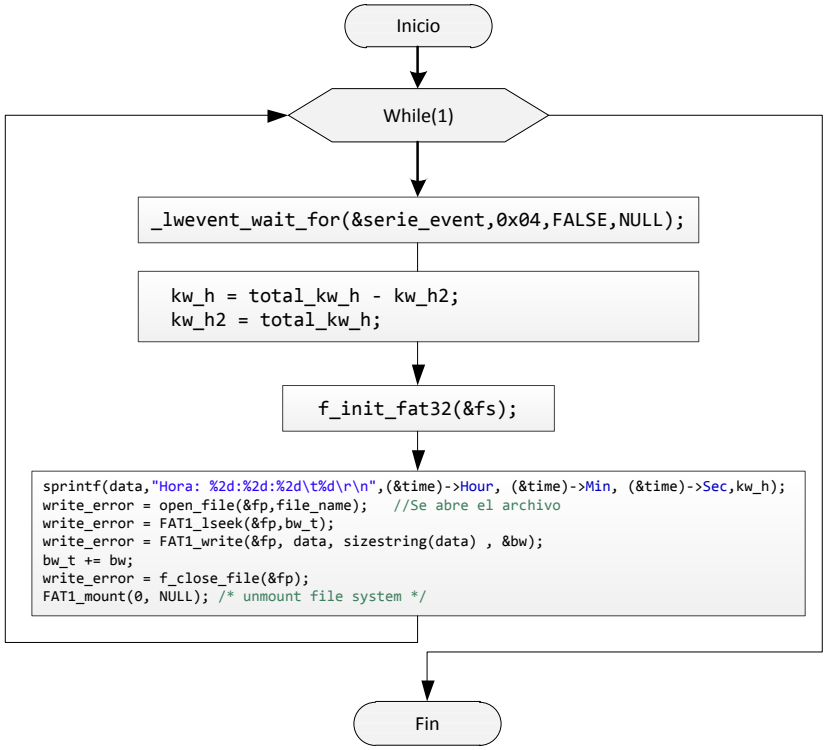


Figura 3. 49 Diagrama de fllujo: Write_file_task.

3.5.1.7 Billing_task.

Esta tarea tiene la función de calcular el costo de la energía consumida de los últimos 60 días. Esta tarea hace uso de una función que es la encarga de realizar los cálculos del costo de acuerdo al recibo mostrado en el capítulo 2.

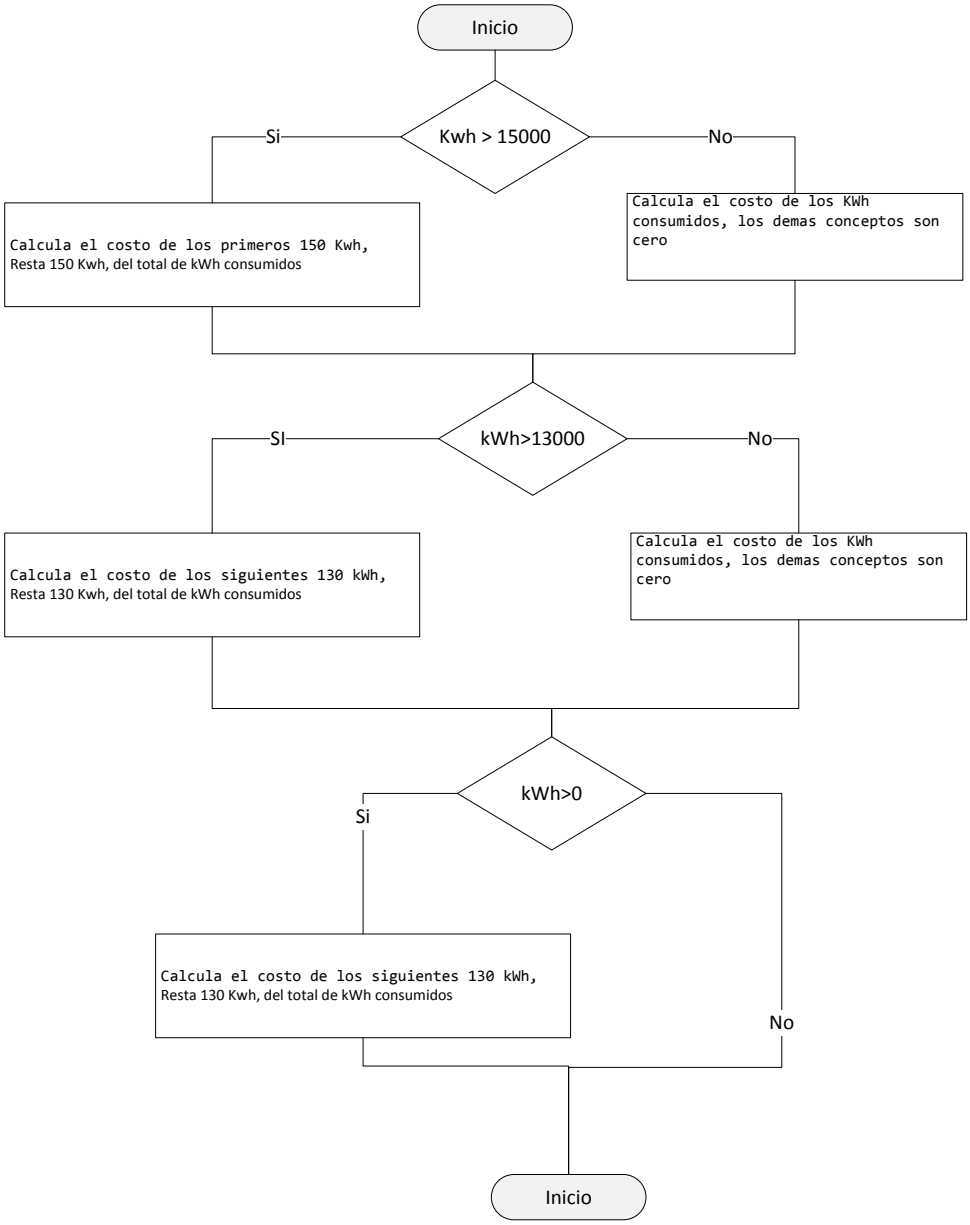


Figura 3. 50 Diagrama de flujo de Billing_Task..

3.5.2 Interrupciones.

Las interrupciones, es un mecanismo para informar al CPU que un evento ha ocurrido. En este trabajo se tienen distintas interrupciones como:

3.5.2.1 AD1_OnMeasurementComplete.

Esta interrupción es provocada por el Convertidor Analógico Digital, y es quizás la más importante ya que en esta se guarda la información en los arreglos y se informa a las tareas que los arreglos están listos para ser tratados por los algoritmos de medición. En la figura 3.51 se observa el diagrama de flujo.

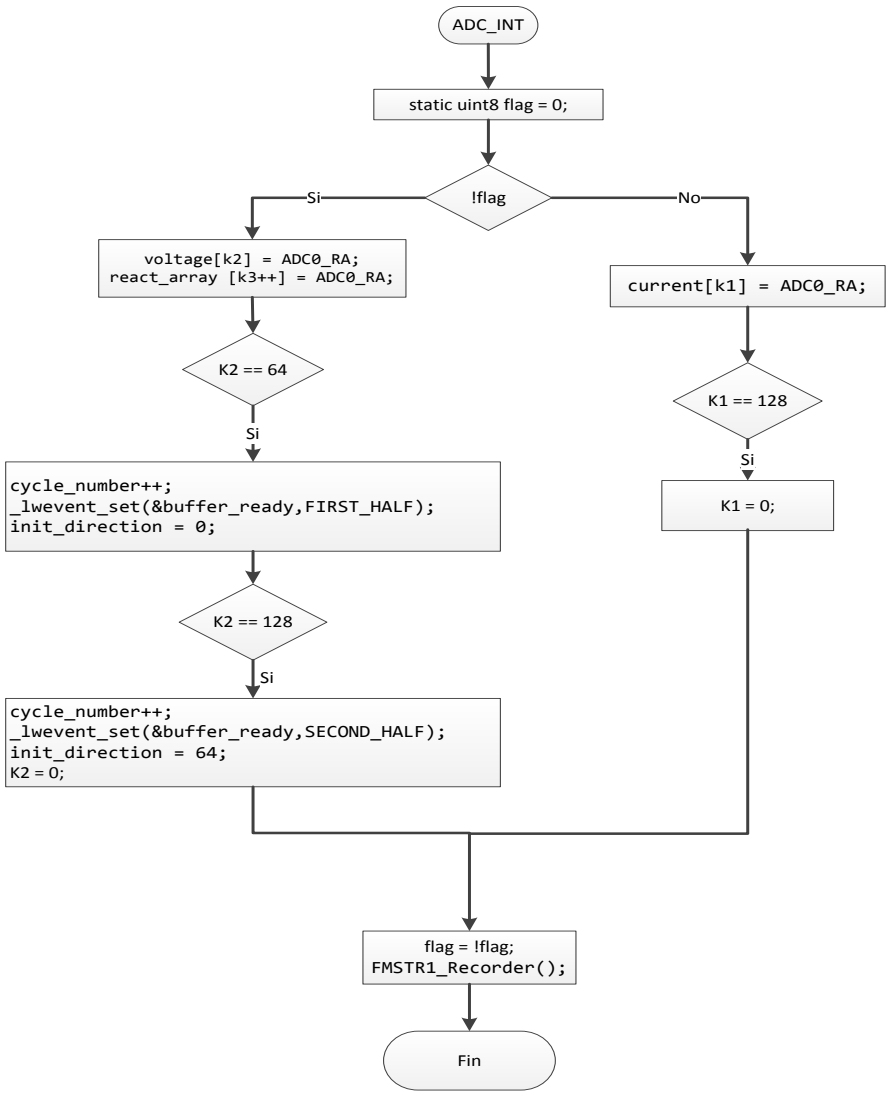


Figura 3.51 Diagrama de flujo: Interrupción del ADC.

3.5.2.2 AS1_OnBlockReceived.

Esta interrupción ocurre cuando el puerto serial recibe información de la pantalla. El objetivo de esta interrupción es desbloquear la tarea Serial_task y además configurarlo nuevamente para que reciba más información.

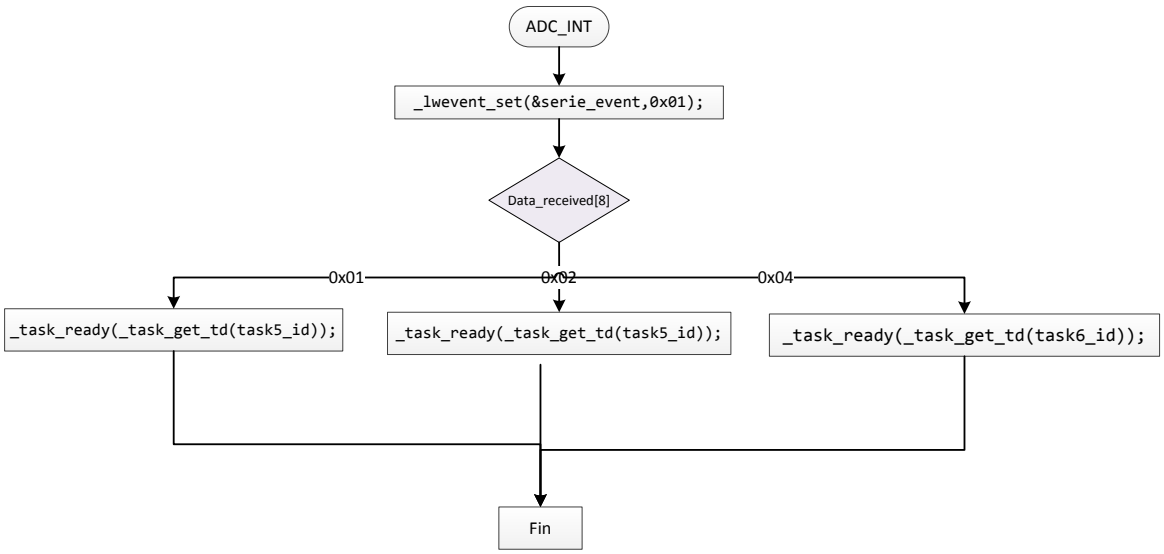


Figura 3. 52 Diagrama de Flujo de la interrupción del puerto serie.

Capítulo 4. Pruebas y resultados

4.1 Introducción.

En este capítulo se muestran las pruebas hechas al prototipo tanto al hardware como al software y los resultados arrojados por lo algoritmos de medición implementados.

En primer lugar se detallan las pruebas al hardware , con el objetivo de asegurar el correcto funcionamiento y por tanto asegurar el resultado de las mediciones.

Después se muestran los resultados arrojados por los algoritmos de medición y la forma en que se escalaron las variables de tensión y corrientes eficaces además de la potencia activa y reactiva.

Por último se muestran un análisis con respecto al porcentaje de error obtenido de las mediciones.

4.2 Pruebas al Hardware.

En primer lugar se hicieron pruebas de funcionamiento del hardware, por medio de un osciloscopio se verifica que las señales de tensión y corriente no excedan los valores limites que admite el ADC del microcontrolador.

Los valores de tensión y corriente utilizados en esta prueba no son de importancia, lo significativo es observar las formas de onda de las señales a analizar. En la figura 4.1 se muestran las señales entregadas por los circuitos de acondicionamiento.

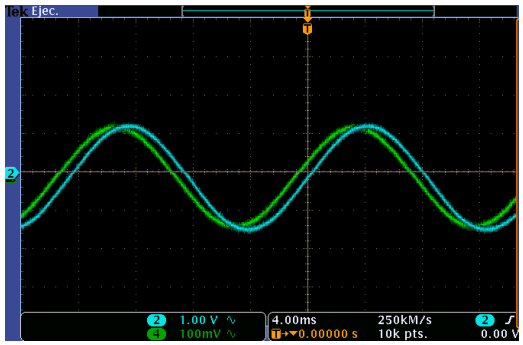


Figura 4. 1 Defasamiento entre la tensión y la corriente.

Como se puede observar en la figura 4.1, existe un defasamiento entre las señales (corriente en adelanto), esto es debido a los elementos inductivos y/o capacitivos presentes en el circuito, tales como el sensor de corriente y los capacitores de filtro.

Este defasamiento representa una dificultad ya que si se aplican los algoritmos de medición estos arrojarían que existe una carga capacitiva ya que la corriente se encuentra adelantada con respecto a la señal de tensión, lo cual no es posible ya que la fuente patrón se configuró de forma de que estas señales estén en fase.

4.2.1 Cálculo del ángulo de defase.

Como punto de partida se utilizó la fuente patrón como instrumento de medición del ángulo de defase. Debido a que esta fuente es capaz de generar retrasos o adelantos de las señales, ya sea de tensión o corriente, el procedimiento del cálculo del ángulo de defase es el siguiente:

1. Configurar la fuente con una tensión y corriente de alimentación prueba, por ejemplo: 127 V, 4 A en fase.
2. Conectar el osciloscopio en las terminales del prototipo.
3. Configurar el osciloscopio de forma que tanto la corriente y la tensión tengan la misma amplitud.
4. Generar por medio de la fuente el retraso o adelanto de tal forma que se pueda apreciar que las señales estén en fase.

Siguiendo este procedimiento se pudo notar que el ángulo de defase es de 24.4 ° aproximadamente, con la corriente en adelanto. En la figura 4.2 se puede observar que configurando un retraso en la corriente de 24.4° es posible que las señales estén en fase.

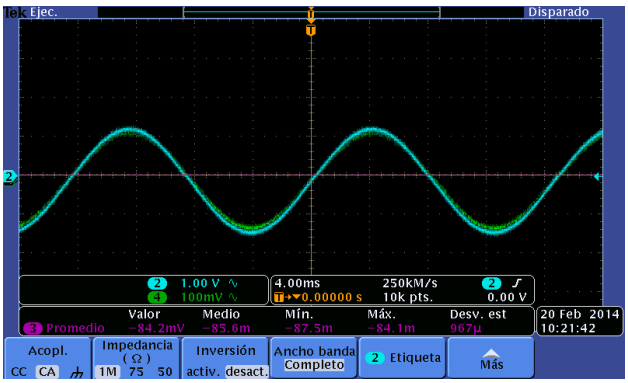


Figura 4. 2 Corrección del ángulo de defasamiento por medio de la fuente patrón.

Cabe mencionar que el defasamiento es constante para cualquier magnitud de corriente. A continuación se muestran los resultados de una prueba realizada a 127 V y variando la corriente

Prueba	Corriente	Ángulo
1	500 mA	24.4º
2	1 A	24.4º
3	5 A	24.4º
5	15 A	24.4º
6	30 A	24.4º
7	40 A	24.4º
8	45 A	24.4º

4.2.2 Corrección del ángulo de desfasamiento.

Como ya se mencionó este ángulo causaría errores en las mediciones si no se corrigiera. La corrección de este desfasamiento se hace por software y a continuación se muestra el procedimiento.

En primer lugar se hace notar que el convertidor analógico digital trabaja a una frecuencia de muestreo de 3840 Hz, por lo tanto se obtienen un total de 64 muestras por ciclo eléctrico de cada una de las señales. Un ciclo eléctrico consta de 360 ° eléctricos y por lo tanto 1 muestra en grados equivale a:

$$1\ Muestra = \frac{360\ °}{64\ muestras} = 5.625\ ° \qquad 4.\ 15$$

Se puede obtener el retraso de la corriente modificando el índice del arreglo donde será almacenada la información. Por lo que para obtener un retraso de 24.4 ° es necesario

$$Numero\ de\ muestras = \frac{24.4\ °}{5.625} = 4.333\ muestras \qquad 4.\ 16$$

Basta que el índice del arreglo inicie en el elemento 4 del arreglo para generar un retraso de 22.5 °. Por lo que aún resta por compensar 1.9 °.

Este ajuste de 1.9 º eléctricos se hace por medio del uso de la herramienta de timers presentada en la sección 3.4. En esa sección se menciona que los timers se agregan a una estructura de datos llamada cola y que ésta se ejecuta periódicamente, y que cuando el periodo de los timers expira se llama a una función. Además en esta estructura es posible asignar un retardo de tiempo con respecto al tiempo en que inicia a ejecutarse la cola de timers.

En la figura 4.3 se muestran las gráficas de los buffers de tensión y corriente después del ajuste del defasamiento del ángulo, es decir, son los datos que obtiene el ADC. Se puede ver que el ajuste fue correcto.

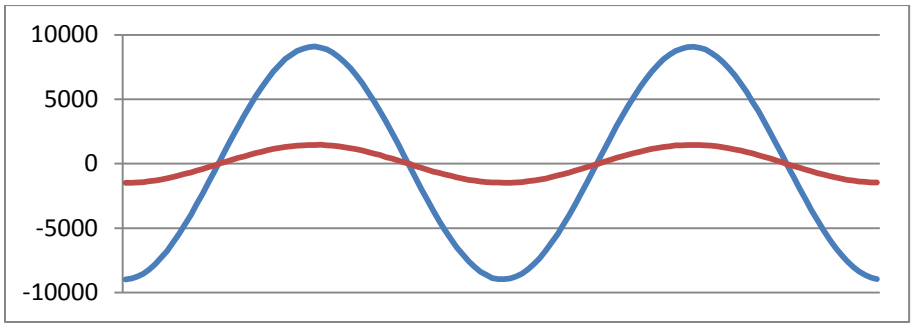
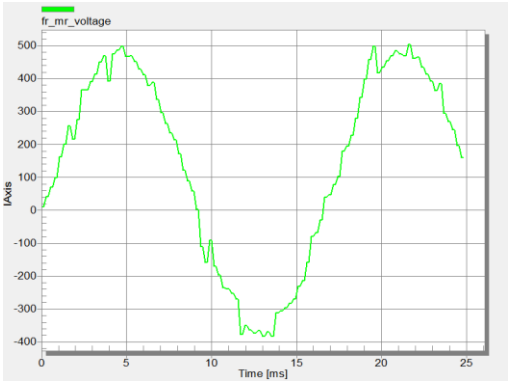


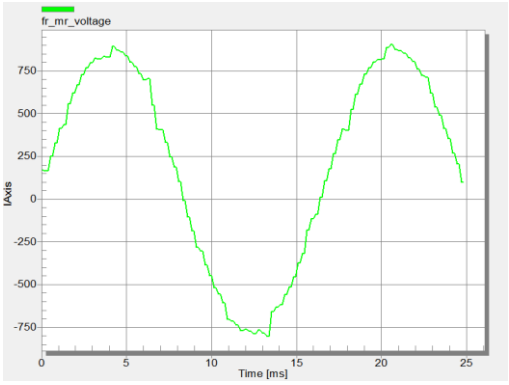
Figura 4. 3 Corrección del ángulo de defasamiento por software.

En este trabajo se utilizó una herramienta llamada Freemaster la cual es un programa capaz de mostrar de forma gráfica el valor de alguna de las variables globales en el programa. En el apéndice F se muestra a más detalle lo que es Freemaster y se muestra como se configuró [34].

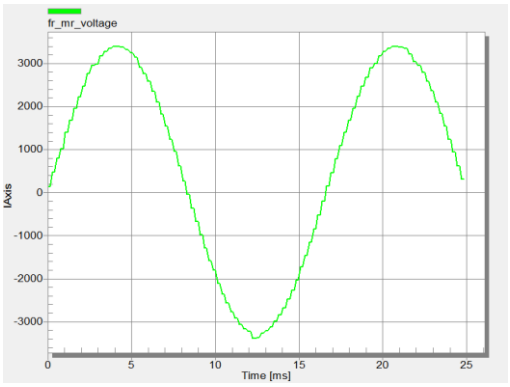
Señales de Tensión



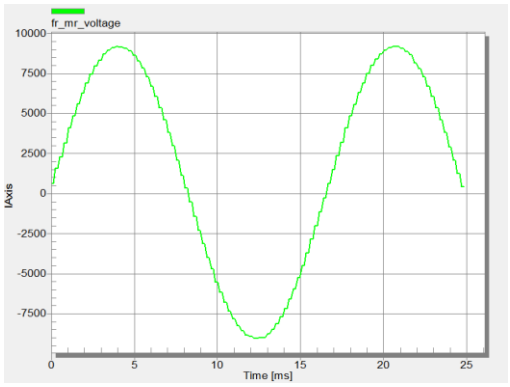
a) 5 V



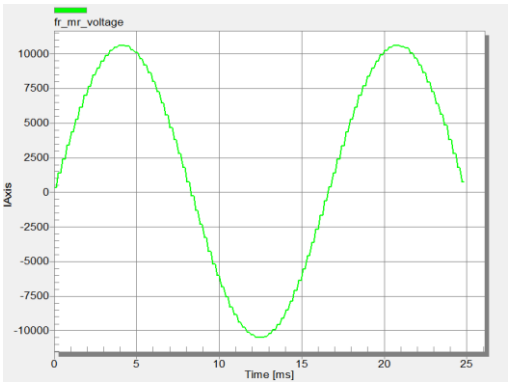
b) 10 V



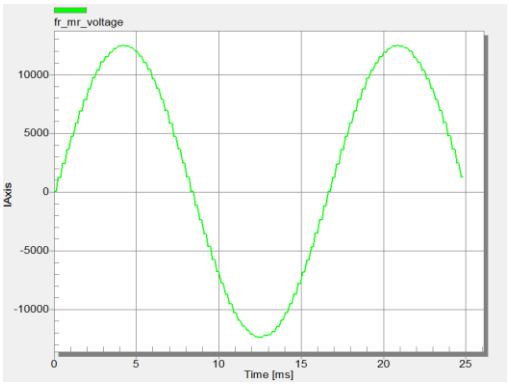
c) 40V



d) 110V



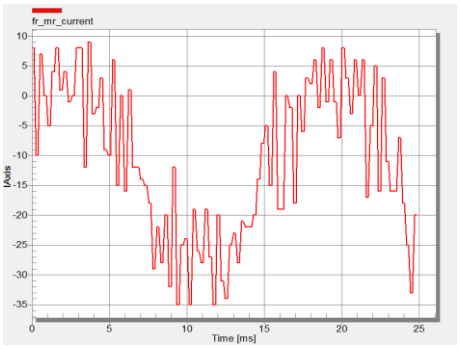
e) 127



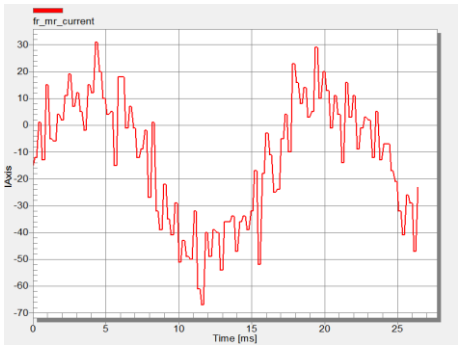
f) 150 V

Figura 4. 4 Señales de tensión capturadas con Freemaster

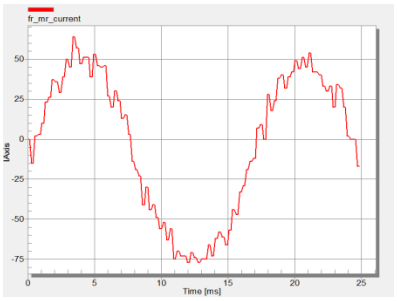
Señales de Corriente.



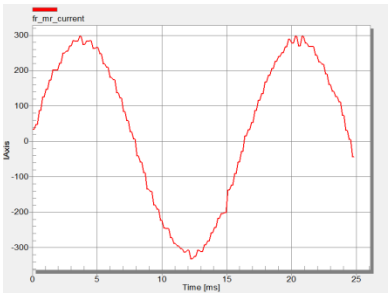
a) 20mA.



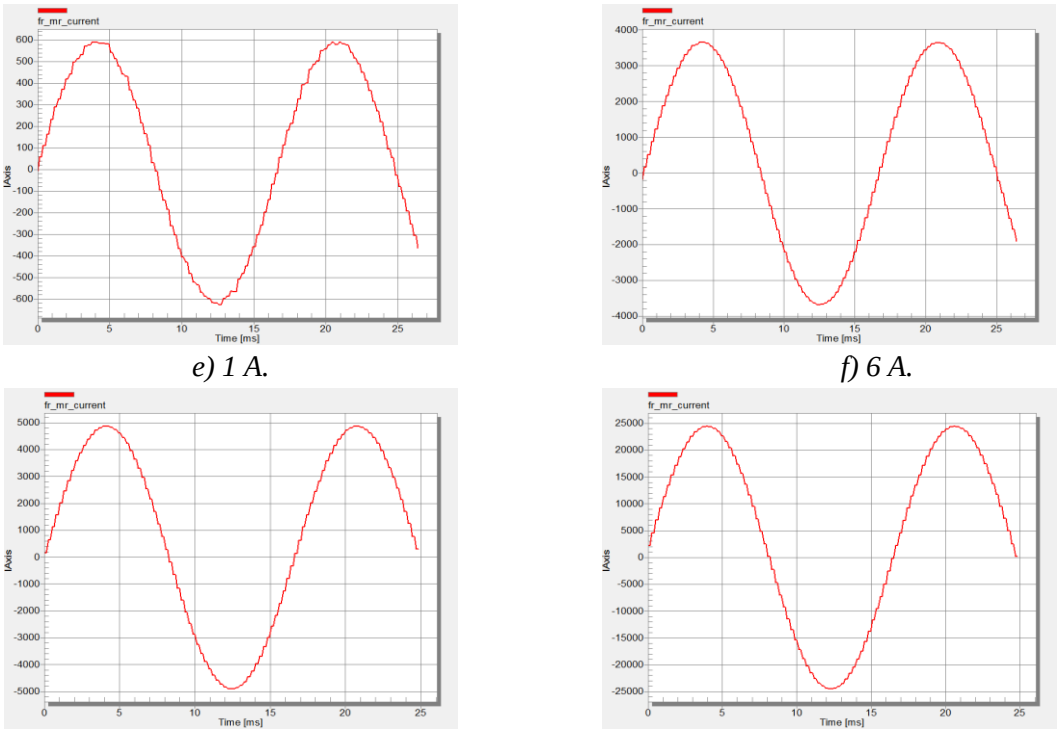
b) 40 mA.



c) 100 mA.



d) 500 mA.



e) 1 A. *f) 6 A.*

g) 10ª. *h)40 A.*

Figura 4. 5 Señales de corriente capturadas por Freemaster.

Como se pueden observar en la figura 4.5a. cuando se presentan magnitudes de corriente del orden de 20 mA, la forma de onda capturada por el ADC del microcontrolador se ve muy distorsionada debido a que la relación de transformación del instrumento de medición es de 1000:1 por lo que cuando circulan 20 mA por el primario, por el secundario circularían 20 µA.

4.3 Escalamiento de señales.

El escalamiento es el proceso de convertir los resultados de los algoritmos de medición en valores reales, y consiste en la recopilación y aplicación del método de regresión lineal de los valores arrojados por los algoritmos, para obtener los valores reales de tensión y corriente.

4.3.1 Escalamiento de Tensión.

Consiste en comparar y anotar en una tabla los valores arrojados por el algoritmo de medición y los valores reales aplicados al prototipo. En este trabajo se realizaron distintas escalas de valores de tensión con el objetivo de tener mejor precisión de medición.

Tabla 4. 1 Escalamiento de tensión

Tensión Real volts	Valor rms	Lectura deseada	Tensión Real volts	Valor rms	Lectura deseada
0	55	0	**40**	2345	4000
0.5	63	50	**50**	2920	5000
1	83	100	**60**	3500	6000
1.5	105	150	**70**	4080	7000
2	133	200	**80**	4660	8000
2.5	160	250	**90**	5242	9000
3	188	300	**100**	5818	10000
3.5	217	350	**100**	5783	10000
4	246	400	**105**	6068	10500
4.5	275	450	**110**	6352	11000
5	303	500	**115**	6642	11500
5.5	332	550	**120**	6925	12000
6	359	600	**125**	7210	12500
6.5	388	650	**130**	7495	13000
7	412	700	**135**	7777	13500
7.5	443	750	**140**	8062	14000
8	470	800	**145**	8348	14500
8.5	500	850	**150**	8630	15000
9	529	900	**155**	8912	15500
9.5	555	950	**160**	9191	16000
10	587	1000	**165**	9471	16500
20	1174	2000	**170**	9752	17000
30	1762	3000			

Las escalas utilizadas son: 0a 10V de 10V a 100V y de 100V a 170V

Escala de 0 a 10 V

Como se mencionó se aplica el método de regresión lineal con el objetivo de obtener la mejor recta que se ajuste a los puntos. En la figura 4.6 se observa la recta y su ecuación.

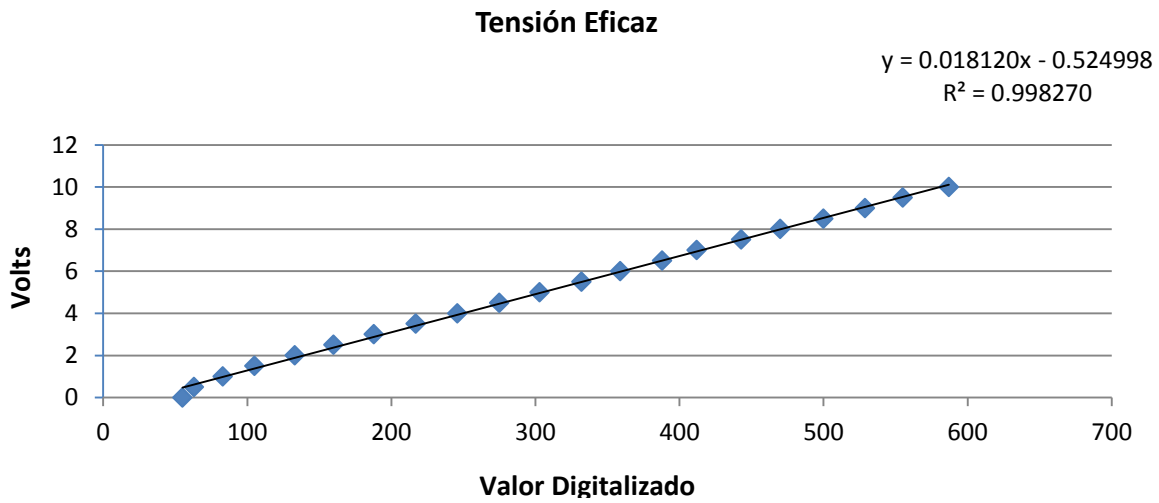


Figura 4. 6 Escala de 0 a 10 V

Debido a que el microcontrolador es un dispositivo de punto fijo, no es posible programar directamente los valores obtenidos de la ecuación, por lo tanto se utilizó un factor Qn para multiplicar el valor de la ecuación y de esta manera poder realizar los cálculos con valores enteros en lugar de valores decimales, este factor al final de las operaciones debe ser eliminado de lo contrario tendríamos resultados equivocados.

Aplicando Q15 a la primera escala 0-10 V

$$y = 593\,x - 17203 \qquad 4.\,17$$

De esta manera ya se tienen valores enteros, los cuales ya pueden ser utilizados por el microcontrolador

Escala de 10 a 100V

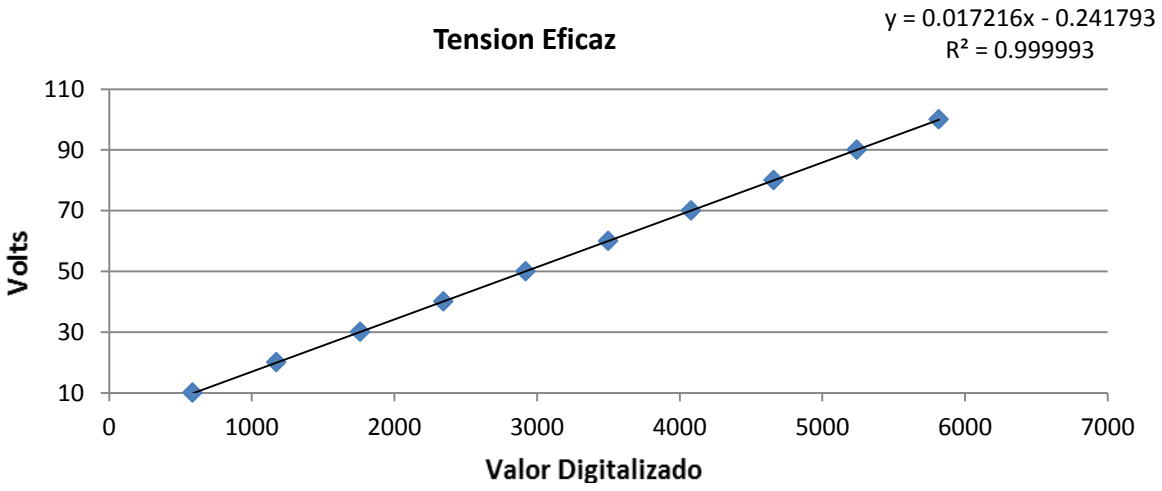


Figura 4. 7 Escala de 10 a 100V

La ecuación a programar en el microcontrolador es:

$$y = 563\,x - 7923 \qquad 4.\,18$$

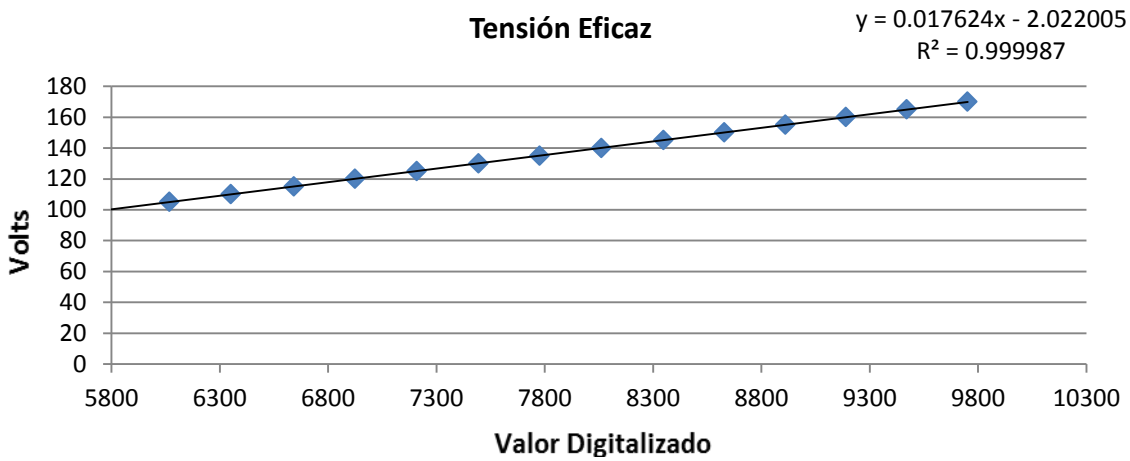


Figura 4. 8 Escala de 100 a 150 V

$$y = 577\,x - 66257 \qquad 4.\,19$$

Dónde:

x representa el valor rms calculado con el algoritmo de medición

m representa la pendiente de la ecuación y es el resultado de la multiplicación de la ecuación original por el formato Q15

y representa el valor escalado de la tensión, es decir, el valor que ya es posible mostrar en un display.

4.3.2 Escalamiento de corriente.

Al igual que la señal de tensión, la corriente también se le aplica el proceso de escalamiento con las siguientes escalas:

- 0 – 0.1 A
- 0.1-1 A,
- 1-10 A
- 10 A a 45 A.

Tabla 4. 2 Escalamiento de Corriente

I real	Corriente Calculada	Lectura Escalada	I real	Corriente Calculada	Lectura Escalada	I real	Corriente Calculada	Lectura Escalada
0.01	19	1	1.5	642	150	21	9099	2100
0.02	22	2	2	860	200	22	9535	2200
0.03	24	3	2.5	1078	250	23	9971	2300
0.04	27	4	3	1293	300	24	10405	2400
0.05	30	5	3.5	1511	350	25	10835	2500
0.06	33	6	4	1727	400	26	11270	2600
0.07	37	7	4.5	1945	450	27	11705	2700
0.08	40	8	5	2160	500	28	12138	2800
0.09	45	9	5.5	2379	550	29	12570	2900
0.1	49	10	6	2595	600	30	13005	3000
0.15	68	15	6.5	2811	650	31	13435	3100
0.2	89	20	7	3028	700	32	13866	3200
0.25	109	25	7.5	3246	750	33	14300	3300
0.3	131	30	8	3462	800	34	14735	3400
0.35	152	35	8.5	3680	850	35	15164	3500
0.4	173	40	9	3898	900	36	15600	3600
0.45	195	45	9.5	4115	950	37	16032	3700
0.5	214	50	10	4331	1000	38	16460	3800
0.55	237	55	11	4764	1100	39	16900	3900
0.6	258	60	12	5194	1200	40	17333	4000
0.65	279	65	13	5629	1300	41	17765	4100
0.7	300	70	14	6063	1400	42	18198	4200
0.75	321	75	15	6496	1500	43	18623	4300
0.8	344	80	16	6929	1600	44	19050	4400
0.85	365	85	17	7363	1700	45	19483	4500
0.9	385	90	18	7797	1800			
0.95	407	95	19	8232	1900			
1	428	100	20	8665	2000			

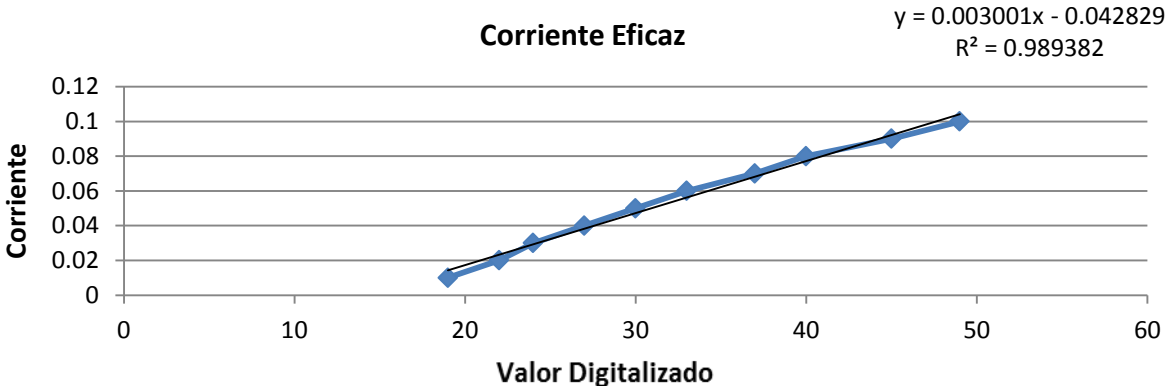


Figura 4. 9 Escala de 0-0.1A

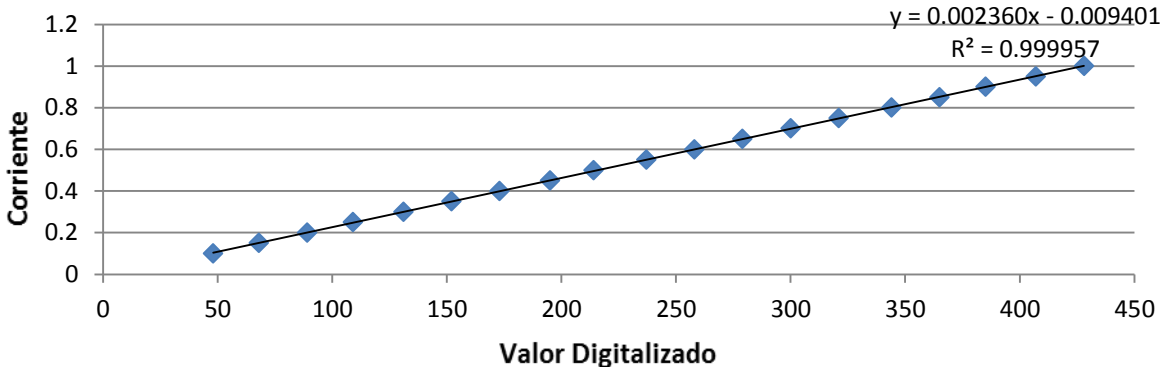


Figura 4. 10 Escala 0.1-1A

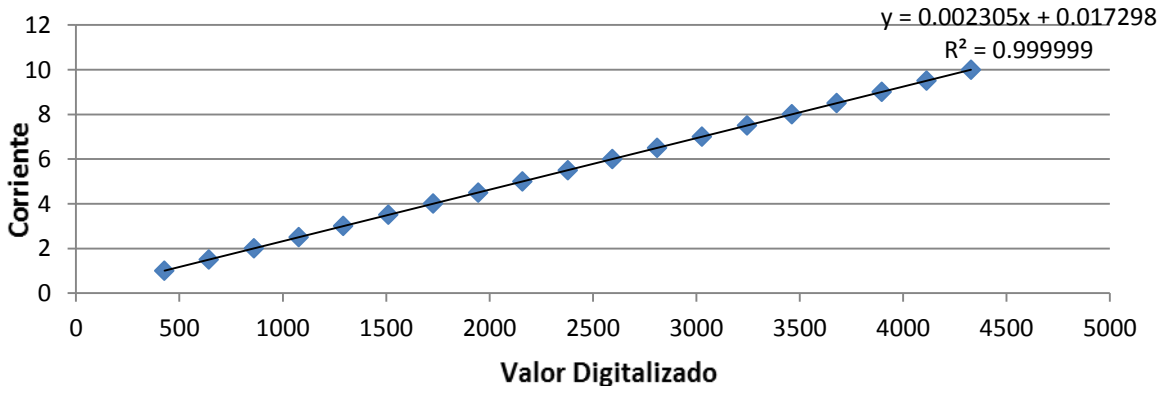


Figura 4. 11 Escala 1-10A

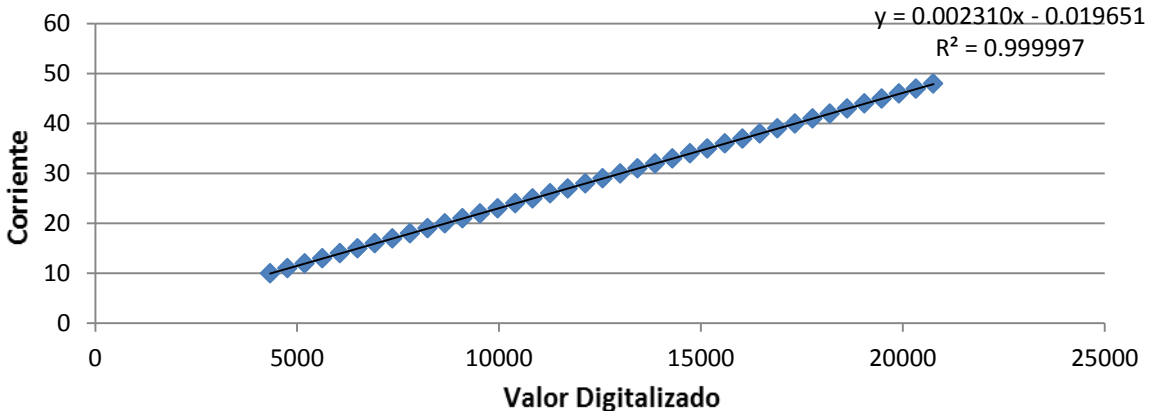


Figura 4. 12 Escala 10-45A

Resumen de las ecuaciones de escalamiento de corrientes.

$$y = 98\,x - 1403 \qquad 4.\ 20$$

$$y = 77\,x - 308 \qquad 4.\ 21$$

$$y = 75\,x - 566 \qquad 4.\ 22$$

$$y = 76\,x - 643 \qquad 4.\ 23$$

Estas ecuaciones representan el escalamiento en cada una de las escalas. Sin embargo estos valores pueden ser un poco diferentes al momento de programarlas en el micro.

4.3.3 Escalamiento de Potencia Activa.

La potencia activa se le aplica el mismo procedimiento que la tensión y la corriente, caso diferente a la potencia aparente ya que como se puede observar en la ecuación 2.12 la potencia aparente es el resultado de la multiplicación de la tensión y corriente eficaz y como estas ya fueron escaladas entonces, su escalamiento es automático.

La potencia activa depende de dos variables, es por eso que para realizar este proceso se utilizó una tensión constante de 127 VCA y una corriente variable. Se utilizaron escalas similares a las de la corriente.

En la Tablas 4.3 se pueden observar los resultados, utilizando las escalas mencionadas.

Tabla 4. 3 Escalamiento potencia activa

I rms	Lectura	Lectura deseada	I rms	Lectura	Lectura Deseada	I rms	Lectura	Lectura Deseada
0.05	169993	6.35	6	18973486	762	26	83486576	3302
0.06	199204	7.62	6.5	20544854	825.5	27	86724584	3429
0.07	223153	8.89	7	22143445	889	28	89919648	3556
0.08	270771	10.16	7.5	23723982	952.5	29	93108509	3683
0.09	286085	11.43	8	25288949	1016	30	96357579	3810
0.1	331380	12.7	8.5	26889148	1079.5	31	99546181	3937
0.2	649287	25.4	9	24482073	1143	32	102785329	4064
0.3	963098	38.1	9.5	30069988	1206.5	33	105970061	4191
0.4	1265314	50.8	10	32155488	1270	34	109276130	4318
0.5	1587269	63.5	11	35268541	1397	35	112304697	4445
0.6	1890195	76.2	12	38478974	1524	36	115609868	4572
0.7	2227443	88.9	13	41666040	1651	37	118759622	4699
0.8	2524884	101.6	14	44895880	1778	38	121972545	4826
0.9	2848138	114.3	15	48091890	1905	39	125254562	4953
1	3128270	127	16	51303006	2032	40	128346283	5080
1.5	4699308	190.5	17	54533248	2159	41	131604539	5207
2	6291826	254	18	57736613	2286	42	134811408	5334
2.5	7872960	317.5	19	60958095	2413	43	138070277	5461
3	9447106	381	20	64134422	2540	44	141239758	5588
3.5	11042182	444.5	21	67379655	2667	45	144420273	5715
4	12630547	508	22	70629366	2794			
4.5	14216724	571.5	23	73842848	2921			
5	15792327	635	24	77062970	3048			
5.5	17381189	698.5	25	80251552	3175			

En las siguientes figuras se observan las gráficas obtenidas de la tabla 4.3, además se puede observar la ecuación de la recta que mejor se ajusta a los puntos.

En las figuras también se puede observar el índice R el cual indica que tan bien se ajustó la recta a los puntos.

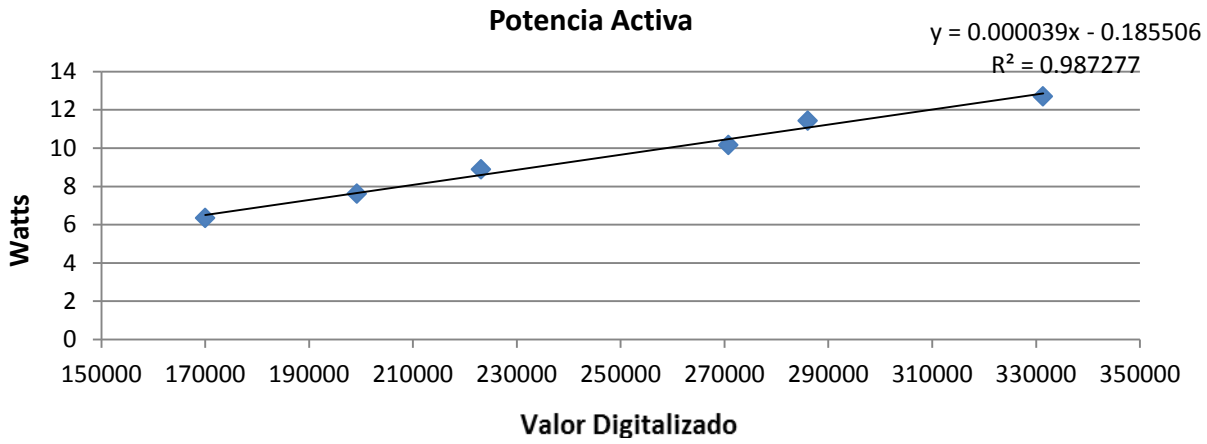


Figura 4. 13 Escala 0-0.1A

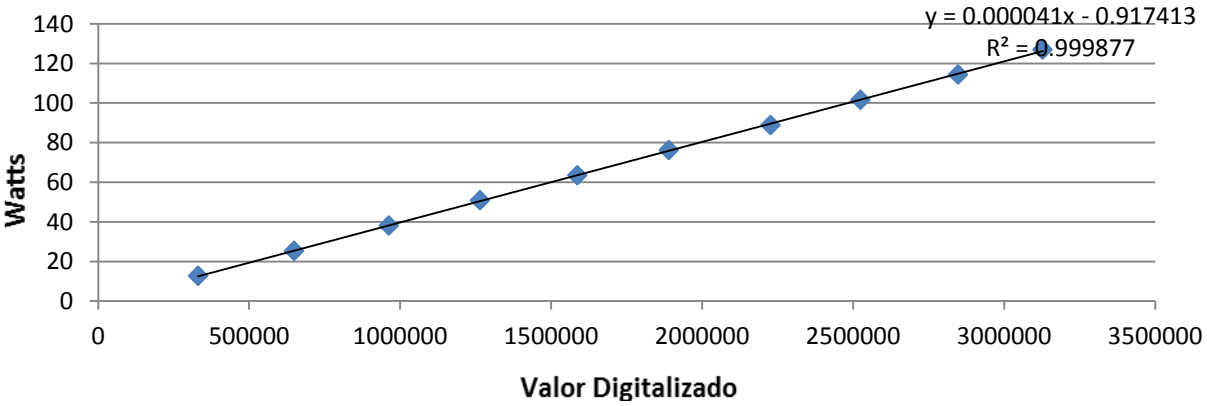


Figura 4. 14 Escala 0.1-1A

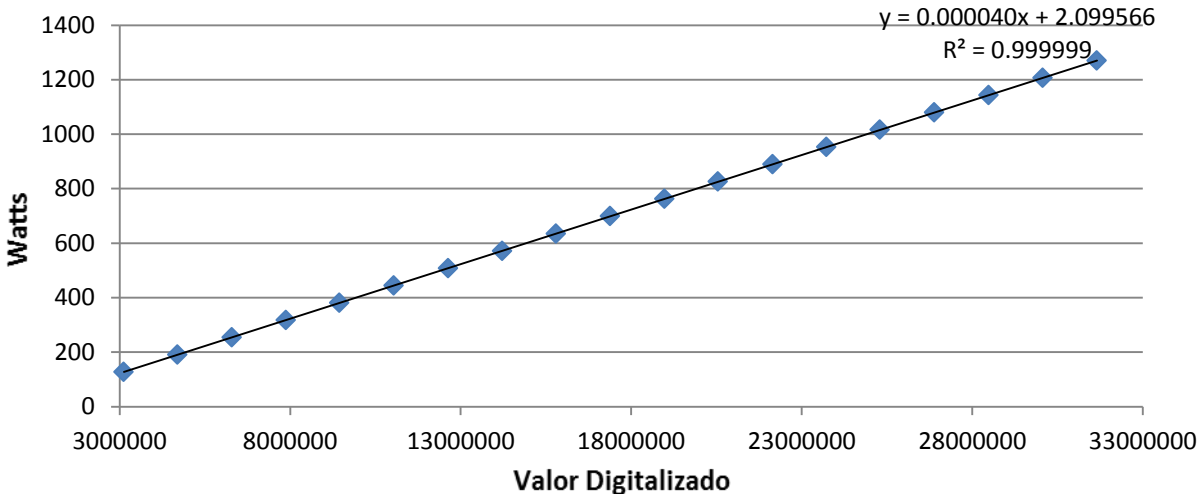


Figura 4. 15Escala 1-10A

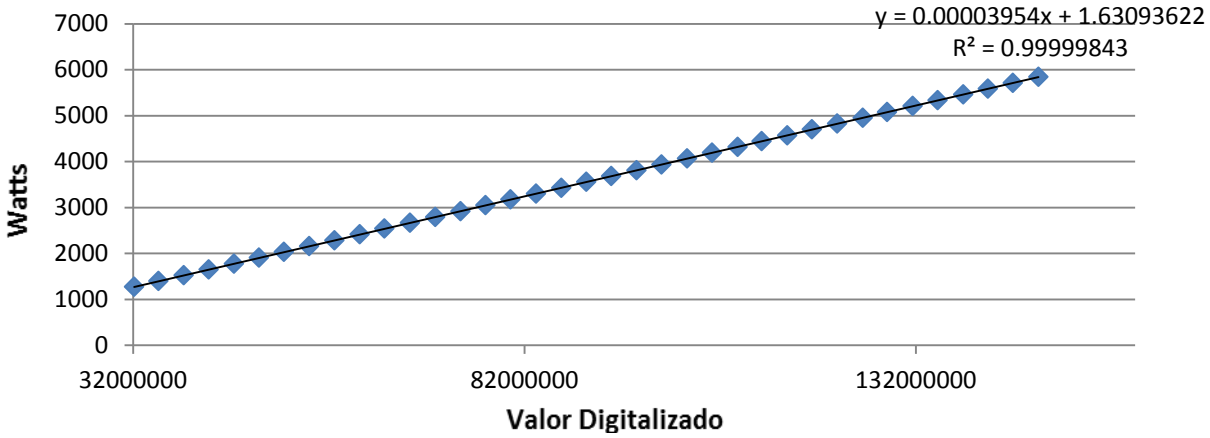


Figura 4. 16 Escala 10-45A

Resumen de las ecuaciones

$$y = 20x - 97258 \qquad 4.\ 24$$

$$y = 343x - 7695818 \qquad 4.\ 25$$

$$y = 338x - 17612436 \qquad 4.\ 26$$

$$y = 41x + 1710160 \qquad 4.\ 27$$

Cabe resaltar que estos resultados son ideales por lo que podrían ser ligeramente diferentes a los programados en el microcontrolador.

4.4 Resultados de Exactitud.

A continuación se presenta un análisis de la exactitud de los resultados arrojados por los algoritmos de medición implementados, estos resultados se presentan en la tabla 4.4. Posteriormente se presenta de forma gráfica los resultados que arrojaron los algoritmos de medición, las principales variables de medición son:

- Potencia real
- Factor de potencia
- Potencia aparente

También se presentan los resultados de la tensión y corriente eficaz.

Tabla 4. 4 Resultados de mediciones de potencias

Corriente	Potencia real	Potencia Reactiva	Potencia Aparente	Factor de potencia	Potencia Real medida	Potencia Reactiva	Potencia Aparente	Factor de potencia	%error potencial real	%error potencia aparente	%error fp	% Potencia Reactiva
0.1	10.9985	6.35	12.7	0.86602	11.4	6	12.71	0.87	3.650284	0.07874	0.45894	5.511811
0.2	21.9970	12.7	25.4	0.86602	22.8	11.7	25.42	0.896	3.650284	0.07874	3.46116	7.874015
0.3	32.9955	19.05	38.1	0.86602	34.14	18.1	38.15	0.892	3.468441	0.131233	2.99928	4.986876
0.4	43.9940	25.4	50.8	0.86602	45.53	24.3	50.87	0.887	3.491172	0.13779	2.42193	4.3307086
0.5	54.9926	31.75	63.5	0.86602	56.53	30.89	63.6	0.882	2.795624	0.157480	1.84458	2.7086614
0.6	65.9911	38.1	76.2	0.8660	67.96	37.4	76.36	0.881	2.983528	0.209973	1.72911	1.837270
0.7	76.989	44.45	88.9	0.86602	79.38	44.01	89.05	0.879	3.104756	0.168728	1.49817	0.989876
0.8	87.9881	50.8	101.6	0.86602	90.64	49.12	101.85	0.878	3.013835	0.246062	1.38270	3.307086
0.9	98.9867	57.15	114.3	0.86602	101.34	56.36	114.66	0.876	2.377386	0.31496	1.15176	1.382327
1	109.985	63.5	127	0.86602	112.85	62.45	127.39	0.876	2.604689	0.307086	1.15176	1.653543
2	219.970	127	254	0.86602	212.15	125.5	256.19	0.869	3.555228	0.862204	0.34347	1.181102
3	329.9556	190.5	381	0.86602	323.29	188.1	382.17	0.867	2.020174	0.307086	0.11253	1.25984
4	439.9409	254	508	0.86602	435.39	252	509.14	0.866	1.03443	0.224409	0.00293	0.787401
5	549.9261	317.5	635	0.86602	547.5	314.9	636.11	0.866	0.441174	0.174803	0.00293	0.818897
6	659.9113	381	762	0.86602	659.01	379.1	765.99	0.866	0.136587	0.523622	0.002933	0.498687
7	769.896	444.5	889	0.86602	770.24	441.23	893.09	0.865	0.044605	0.460067	0.11840	0.735658
8	879.8818	508	1016	0.86602	882.55	501.32	1019.16	0.865	0.303244	0.311023	0.118403	1.31496
9	989.8670	571.5	1143	0.86602	994.89	562.23	1148.8	0.863	0.5074	0.507435	0.34934	1.6220472
10	1099.852	635	1270	0.86602	1107.14	625.36	1275.18	0.863	0.662610	0.407874	0.34934	1.518110
15	1649.7783	952.5	1905	0.86602	1651.28	940.23	1911.06	0.862	0.091018	0.318110	0.46481	1.28818
20	2199.7045	1270	2540	0.86602	2194.59	1259.23	2547.95	0.862	0.232509	0.312992	0.46481	0.848031
25	2749.630657	1587.5	3175	0.86602	2736.02	1570.26	3187.92	0.862	0.494999	0.406929	0.46481	1.08598
30	3299.556788	1905	3810	0.86602	3272.23	1890.66	3828.01	0.862	0.82819	0.472703	0.46481	0.75275
35	3849.48292	2222.5	4445	0.86602	3814.98	2211.33	4462.26	0.861	0.896300	0.388301	0.58028	0.502587
40	4399.4090	2540	5080	0.86602	4354.65	2529.3	5100	0.861	1.017387	0.393700	0.580283	0.421259
45	4949.3351	2857.5	5715	0.86602	4885.19	2840.33	5737.23	0.861	1.296036	0.388976	0.580283	0.600874

4.4.1 Porcentaje de error de tensión y corriente eficaz.

En la siguiente sección se presenta un análisis de la exactitud de las mediciones arrojadas por los algoritmos, donde el eje de las horizontal representa la magnitud de la variable eléctrica a analizar y el eje vertical el error porcentual presentado.

Como se puede observar en la figura 4.17 el porcentaje de error se mantuvo por debajo del 1% de error, presentándose el mayor error en una tensión de 10 V que como se pudo observar en la figura 4.4a esta se muestra deformada con respecto a las demás y esto es debido a la baja amplitud de la señal.

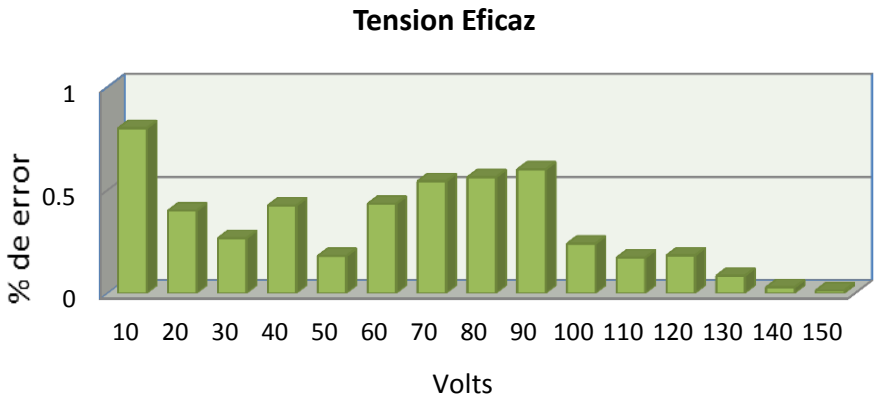


Figura 4. 17. Error Porcentual de la Tensión Eficaz

Con respecto a la señal de corriente ésta tuvo dos errores por arriba del 1% cuando la corriente era menor a 1 A que es el rango donde la corriente es más deformada de acuerdo a las figuras 4.5. Cuando la corriente supero 1 A RMS esta presentó errores por debajo del .5 % de error.

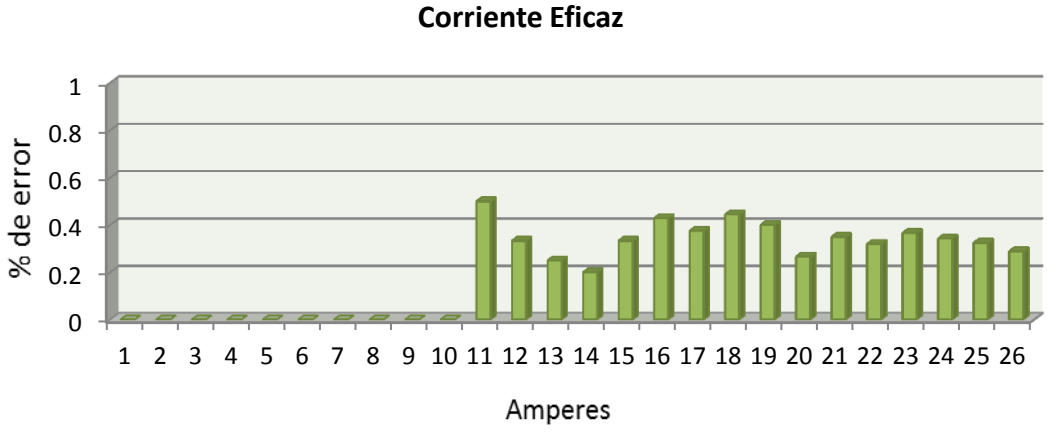


Figura 4. 18 Error Porcentual de la Corriente Eficaz

4.4.2 Porcentaje de error de la potencia Activa

La prueba de potencia activa se hizo utilizando una tensión de 127 VCA y corrientes variables con un ángulo de retraso de 30° con respecto a la señal de tensión El porcentaje de error de la potencia activa presento en general un error porcentual menor al 4 %.

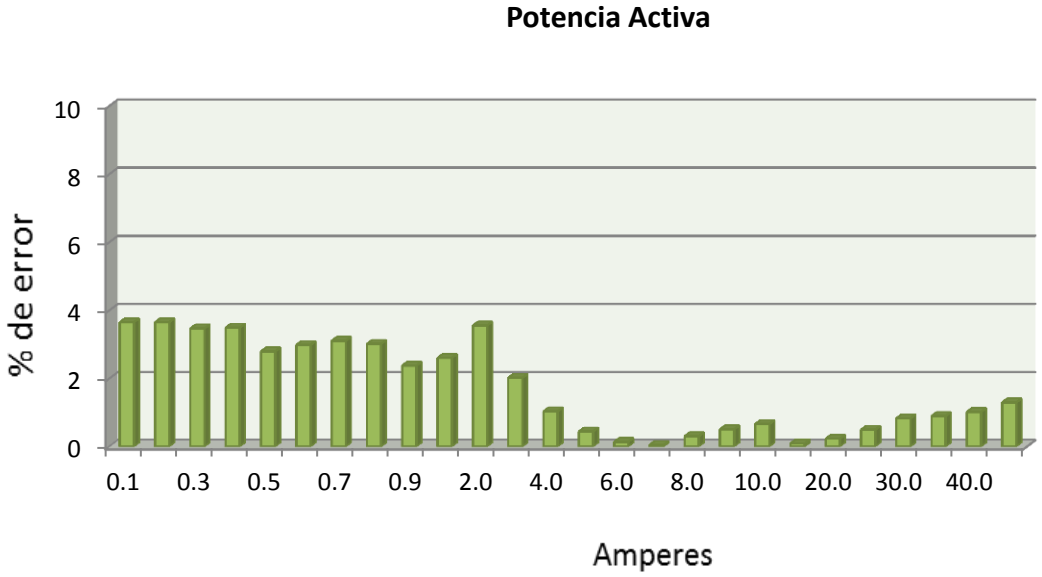


Figura 4. 19. Error porcentual dela Potencia Activa

4.4.3 Porcentaje de error de la potencia Aparente.

En el caso de la potencia aparente se presentó un porcentaje de error menor al 1 %

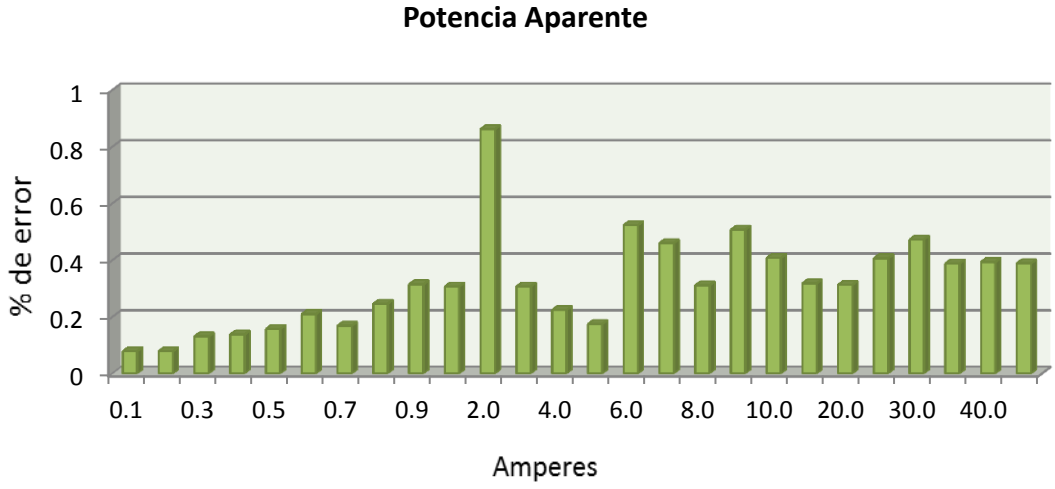


Figura 4. 20 Error porcentual de la potencia aparente

4.4.4 Porcentaje de error del factor de potencia

La medición de factor de potencia presentaron de forma general un error menor al 3.5% acentuándose dicho error en las mediciones menores a 1A.

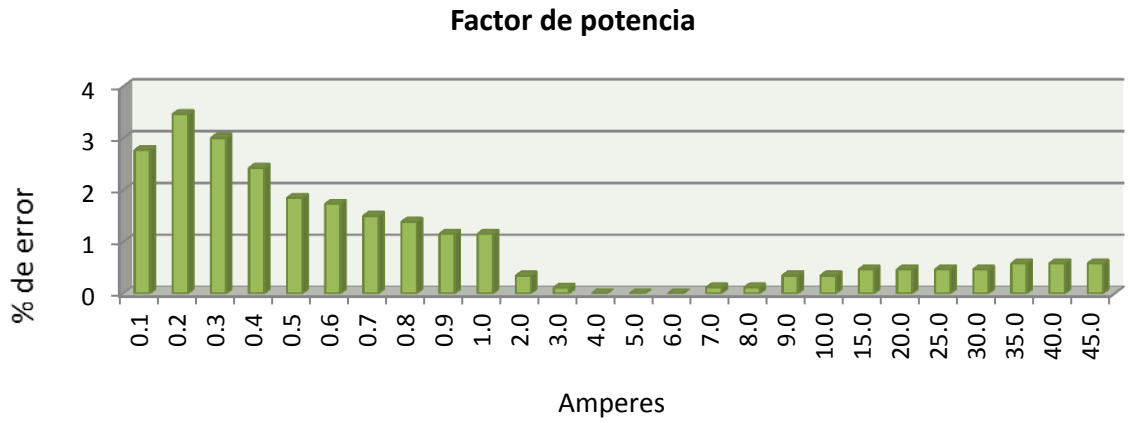


Figura 4. 21 Error porcentual del factor de potencia

4.4.5 Porcentaje de error Potencia Reactiva.

En la figura 4.22 se muestra el porcentaje de error que presento esta variable.

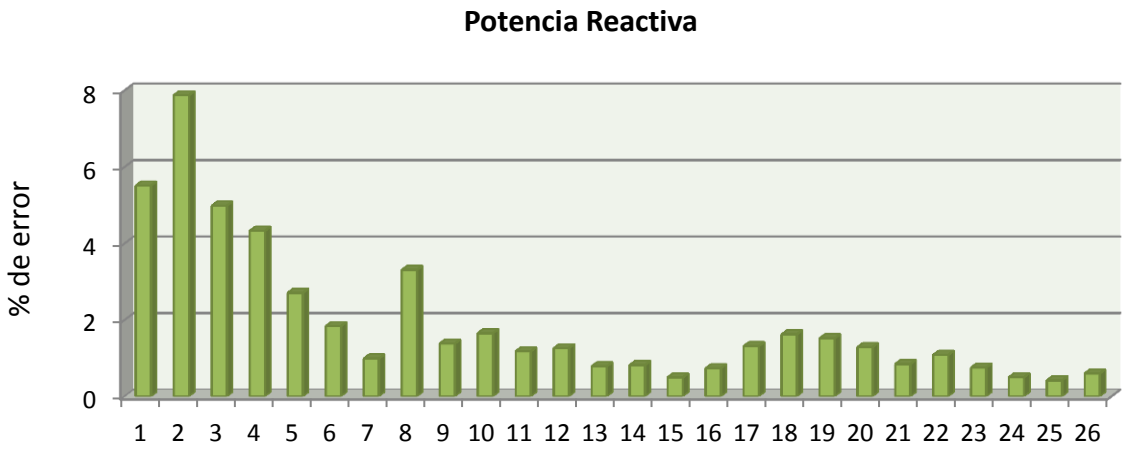


Figura 4. 22 Error porcentual de la potencia Reactiva

4.4.6 Medición de energía.

Estas pruebas de medición de energía consistieron en comprobar el funcionamiento del algoritmo de medición de energía activa y de suma de energía. A continuación se muestran los resultados.

Parámetros de prueba: Tensión: 100 V, Corriente: 10 A

Tiempo (minutos)	kWh exactos	kWh medidos	% de error
15	0.25	0.25	0
30	0.50	0.5	0
45	0.75	0.75	0
60	1	1.00	0.4

Parámetros de prueba: Tensión: 127 V, Corriente: 1 A

Tiempo (min)	kWh exactos	kWh medidos	% de error
15	0.03175	0.03	5.51181102
30	0.0635	0.06	5.51181102
45	0.09525	0.09	5.51181102
60	0.127	0.12	5.51181102

Parámetros de prueba: Tensión: 127 V, Corriente: 5 A

Tiempo (min)	kWh exactos	kWh medidos	% de error
15	0.15875	0.15	5.51181102
30	0.3175	0.31	2.36220472
45	0.47625	0.47	1.31233596
60	0.635	0.63	0.78740157

4.4.7 Medición de Frecuencia.

Como se mencionó en el capítulo 2 la frecuencia se calculó con el algoritmo de cruce por cero y arrojo resultados cercanos a 60Hz. Esta frecuencia es la frecuencia de la señal de tensión.

Capítulo 5. Conclusiones y Trabajos a Futuro

5.1 Introducción.

En esta sección se dan las conclusiones de los resultados obtenidos del trabajo, además se dan algunas recomendaciones y se mencionan los trabajos a futuro surgidos de este trabajo.

5.2 Conclusiones.

En primer lugar se dan las conclusiones acerca del microcontrolador y del hardware utilizado en el desarrollo del proyecto.

- Debido al uso de entradas diferenciales como medio de adquisición de datos fue posible eliminar la operación de resta de la tensión de desplazamiento, lo que se traduce en una disminución en el número de operaciones que el microcontrolador tendría que ejecutar.

- Debido a la estructura de programación de MQX Lite fue posible dividir el proyecto en tareas, lo cual facilitó la planeación del proyecto.

- MQX lite permite la jerarquización de actividades, por lo tanto, al analizar el programa es más sencillo visualizar cuales son las acciones que el programa debe ejecutar en primer lugar.

- Con el uso del sistema operativo se optimiza el CPU ya que las tareas se diseñan para que una vez cumplan su objetivo se bloqueen y por lo tanto permitan que el CPU atienda otras actividades.

- Los eventos en MQX permiten sincronizar las tareas e interrupciones además de que es posible verificar si alguna tarea se ha cumplido en el tiempo límite que se le ha permitido.

- Mediante los timers de MQX lite fue posible la sincronización y disparo de los canales configurados en el ADC, además de compensar el desfasamiento provocado por el transformador de corriente.

- Utilizar un sistema operativo, mejora la legibilidad del programa ya que al crear una tarea, si a esta se le asigna un nombre relacionado con la actividad que va a realizar el programa se va autodocumentando.

- La depuración de un programa en tiempo real es una tarea más sencilla debido a que si alguna parte del código falla o es necesario hacerle actualizaciones basta con dirigirse y modificar a dicha sección de código sin afectar el funcionamiento de las demás tareas.

5.3 Aportaciones.

Con la realización de este trabajo se ha aportado lo siguiente.

- Se diseñó y construyó un prototipo medidor de energía eléctrica como un kilowattorimetro en un sistema residencial domiciliario monofásico.
- El prototipo implementado es un medidor el cual es capaz de medir tanto la energía consumida o generada por el usuario.
- Los resultados de los algoritmos de medición son presentados en una interfaz hombre máquina la cual muestra los valores eficaces de tensión, corriente, frecuencia, factor de potencia y el consumo total de energía en kWh.
- Se muestra la capacidad que tiene el sistema operativo MQX lite en el campo de las mediciones eléctricas además de mostrar la definición y utilización de las herramientas: lwevents y lwtimers como herramientas de sincronización
- Se implementa un circuito de medición de tensión diferencial, con la entrada positiva conectada a la línea viva y la entrada negativa al circuito del hilo neutro.

5.4 Recomendaciones y Trabajos a Futuro.

- Utilizar un microcontrolador orientado a mediciones, tales como la nueva generación de microcontroladores Kinetis MK70 los cuales cuentan con el convertidor analógico/digital de 24 bits sigma delta y amplificador de ganancia programable.
- Utilizar el sistema operativo MQX en lugar de MQX Lite, con el objetivo de mostrar ventajas y/o desventajas del uso de estos RTOS.
- Utilizar transformadores de corriente de alta precisión.
- Utilizar el sistema operativo en otras áreas de ingeniería tal como, control de motores, electrónica de potencia, automatización.
- Migrar el programa realizado en este trabajo a otro microcontrolador el cual soporte MQX en la versión completa.
- Implementar un medidor trifásico con circuitos de acondicionamiento de tensión configurados de forma diferencial.

REFERENCIAS

[1] Wan, Natasha, *"Exceeding 60-Year Life Expectancy from an Electronic Energy Meter", http://www.analog.com/static/imported-files/tech_articles/5779410760_year_life.pdf*

[2] "WATT-HOUR METERMAINTENANCE AND TESTING", http://www.usbr.gov/power/data/fist/fist3_10/vol3-10.pdf

[3] Norma ANSI C12.20, https://www.nema.org/Standards/ComplimentaryDocuments/ANSI-C12-20-Contents-and-Scope.pdf

[4] http://www.analog.com/static/imported-files/tech_articles/16540508432000energymeterua_42000.pdf

[5] Designer Reference Manual MQX-Enabled MCF51EM256 Single-Phase Electricity Meter, DRM121

[6] Han, V., & Venkat , K. Implementation of a Three-Phase Electronic Watt-Hour Meter using the MSP430F471xx. EUA.

[7] Design of a Smart Meter Techno-Economic Model for Electric Utilities in Ontario, 2010. Ontario, Canadá.

[8] Di Lella Daniel, http://www.edudevices.com.ar/download/articulos/buceando/BC_MCU_46_ED.pdf

[9] Design Reference Manual. MQX-Enabled MK30X256 Single-Phase Electricity Meter, EUA.

[10] Ortega, J. O. Tesis : *"Diseño de un medidor electrico digital de prepago".* D.F., México. IPN-CIC

[11] Valdiosera Marroquin, A. Tesis *"Diseño de medidor inteligente e implementacion de sistema de comunicación bidireccional"*. D.F., México, IPN-ESIME-SEPI.

[12] Veronica, B., & Hernández Gómez , B. V. *Tesis " Diseño e implementacion de un medidor fasorial sincrono normalizado con el estandar IEEE C37.118."*, D.F., México, IPN-ESIME-SEPI.

[13] http://mathworld.wolfram.com/SamplingTheorem.html

[14] Oppenheim, Alan, Alan Willsky, and Hamid Nawab. *Signals y Systems*. 2nd ed. Englewood Cliffs, New Jersey: Pearson PrenticeHall, 1996. Impreso.

[15] Referencia a un documento donde se explique que es el fenómeno de ALIASING

[16] Alexander, C. K., & N. O. Sadiku, M. (2006). *Fundamentos de Circuitos Electricos.* McGraw-Hill.

[17] http://www.cfe.gob.mx/paginas/home.aspx

[18] http://app.cfe.gob.mx/Aplicaciones/CCFE/Tarifas/Tarifas/tarifas_casa.asp

[19] http://app.cfe.gob.mx/Aplicaciones/CCFE/Tarifas/Tarifas/Tarifas_casa.asp?Tarifa=DACTAR1&anio=2014

[20] Reference Manual Freescale "FRDM-KL25Z User's Manual"

[21] KL25 Sub-Family Reference Manual

[22] OpenSDA, User´s Guide. EUA. http://www.pemicro.com/opensda/

[23] http://www.tme.eu/en/Document/534f5a97fe05592b537eb7b285aa5dea/ac1050.pdf

[24] http://www.dwin.com.cn/english/

[25] http://www.dwin.com.cn/english/products/bt-14358490280881.html

[26] Appication Note. "Writing your First Application MQX lite"

[27] http://www.steinerberg.com/EmbeddedComponents/FAT_FileSystem/

[28] http://www.ntfs.com/fat-systems.htm

[29] http://www.steinerberg.com/EmbeddedComponents/

[30] Reference Manual "DGUS_SDK User Guide"

[31] Getting Started with Freescale MQX™ RTOS, http://cache.freescale.com/files/soft_dev_tools/doc/quick_ref_guide/FSL_MQX_getting_started.pdf?fasp=1&WT_TYPE=Users%20Guides&WT_VENDOR=FREESCALE&WT_FILE_FORMAT=pdf&WT_ASSET=Documentation&Parent_nodeId=1228773250613737641153&Parent_pageType=product

[32] http://www.freescale.com/webapp/sps/site/training_information.jsp?code=WBT_MQX_RTOS_S7

[33] http://www.freescale.com/webapp/sps/site/training_information.jsp?code=WBT_MQX_RTOS_S8

[34] Application Note," FreeMASTER Usage" http://cache.freescale.com/files/microcontrollers/doc/app_note/AN4752.pdf

[35] Kicad Site web http://www.kicad-pcb.org/display/KICAD/KiCad+EDA+Software+Suite

APENDICE A. CodeWarrior 10.5

CodeWarrior 10.5 (CW) (Figura A1), es un software para la programación de microcontroladores basado en el entorno de desarrollo Eclipse. La plataforma Eclipse se ha vuelto popular en los últimos años como un software de código libre y ha sido adoptada por muchas compañías de semiconductores incluida Freescale.

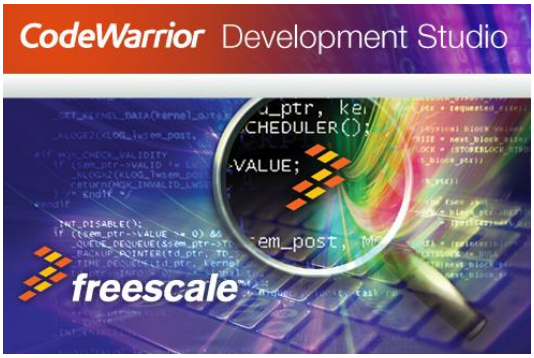


Figura A 1. Pantalla de inicio de CodeWarrior 10.5 IDE.

CodeWarrior trabaja con espacios de trabajo (Woskspaces), lo que lo hace bastante potente al momento de trabajar con distintos proyectos. Un workspace define una carpeta en la cual se guardan los proyectos desarrollados por el usuario, en esta carpeta se crean subcarpetas que es la ubicación donde se guardaran los archivos de cada uno de los proyectos.

CW utiliza perspectivas para la navegación, edición y ejecución del programa. Una perspectiva es un conjunto de ventanas que definen ciertas acciones de control, CW maneja las siguientes perspectivas:

- C/C++.
- Debug.
- Hardware.
- Resource.
- Team Synchronizing.

Al iniciar CW IDE la perspectiva que se muestra es la de C/C++, y al momento de descargar el programa a la tarjeta de desarrollo se muestra la perspectiva Debug.

Perspectiva C/C++.

En esta perspectiva, figura A2, el usuario puede navegar y editar los archivos del proyecto. En esta vista se encuentran las siguientes ventanas:

1. **Component Inspector**. En esta ventana se visualizan y editan las características de cada uno de los componentes de software agregados al proyecto.
2. **Codewarrior Projects**. En esta ventana es posible navegar en los archivos de los proyectos contenidos en el workspace definido.

3. **Components**. Muestra de forma gráfica los componentes de software agregados al proyecto.
4. **Ventana de edición del programa.** En esta área es posible la edición del código del programa.
5. **Component library.** Muestra los beans disponibles para el CPU seleccionado.

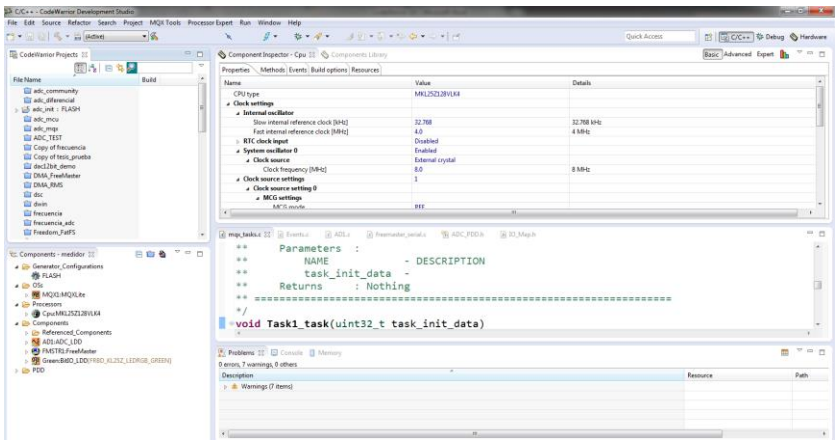


Figura A 2. Perspectiva C/C++.

Perspectiva: Debug.

Esta perspectiva permite al programador depurar y correr el programa. Aquí se puede controlar el flujo del programa colocando puntos de ruptura, pausando el programa, ejecutándolo paso a paso, y examinando el contenido de las variables de interés.

La perspectiva Debug muestra la siguiente información:

- El proceso del programa que se está ejecutando.
- Cada hilo del programa es representado como un nodo.
- Es posible visualizar la cantidad de memoria utilizada por las tareas creadas por MQX lite.
- Esta perspectiva también maneja la vista del código. Mientras el programa se ejecuta paso a paso esta vista señala la línea de código a ejecutar.

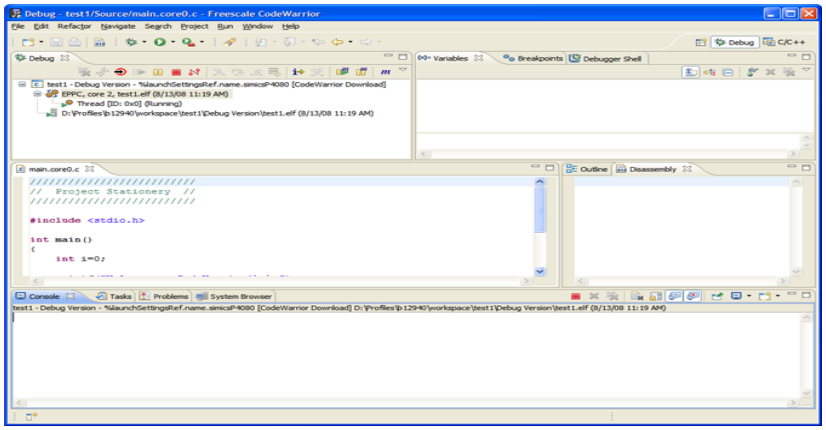


Figura A 3. Perspectiva Debug.

Perspectiva: Hardware

En esta vista se muestra de forma gráfica el empaquetado del CPU, así como la distribución de los beans en los pines del chip. Esta perspectiva es útil ya que permite visualizar la ubicación de los periféricos en el momento del desarrollo del proyecto.

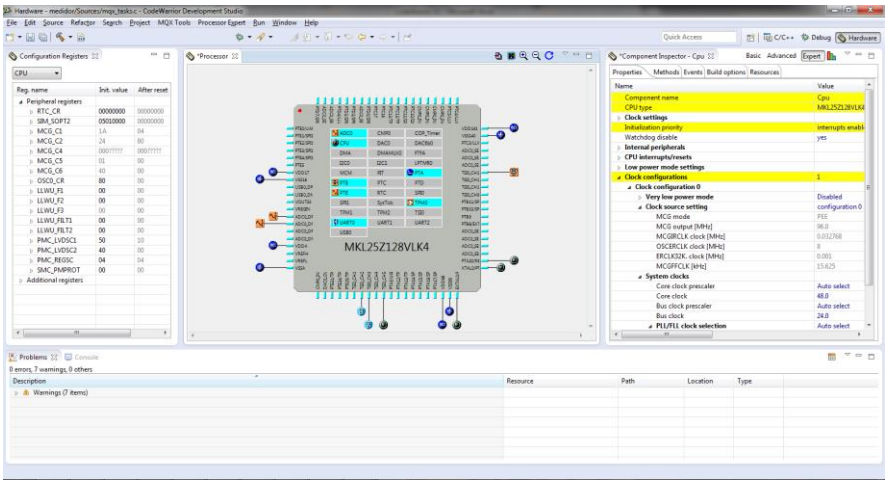


Figura A 4. Perspectiva Hardware.

CodeWarrior 10.5 permite programar múltiples familias de microcontroladores como:

- DSC: 56800/E (DSC)
- Coldfire: V1, V2, V3, V4, V4e
- Coldfire +
- Kinetis: Serie E,K,L y M
- Qorivva
- PX
- RS08
- S08
- S12Z

Requerimientos de CodeWarrior 10.5:

- 2.6GHz Pentium® o superior.
- 4GB RAM.
- 20GB de disco duro.
- 400MB on Windows system disk.
- DVD drive para instalación.
- Puerto USB para la comunicación con el microcontrolador.
- Puerto Ethernet: para comunicación (opcional.

APÉNDICE B. MQX LITE

Sistemas Operativos en Tiempo real

Un sistema operativo en tiempo real (RTO, Real Time Operating System), es un sistema informático que maneja el tiempo de un microprocesador y/o microcontrolador. Sus principales características y funciones son:

- Permite el procesamiento multitarea.
- Sincroniza el acceso a diferentes recursos: memoria, hardware, periféricos, etc.
- Manejo de interrupciones.
- Permite la comunicación entre tareas.
- Programación de tareas por medio de prioridades.

El CPU puede ejecutar muchas tareas, manejar distintas interrupciones, pero solamente una tarea/interrupción está en ejecución.

MQX LITE: Sistema Operativo en Tiempo Real (RTOS)

Es un sistema operativo en tiempo real para microcontroladores de bajos recursos en memoria RAM y memoria FLASH. Es un subconjunto del programa Freescale MQX™ Software Solutions, y contiene las herramientas de éste en versiones light, como:

- Manejo de Tareas: Task Managment.
- Manejo de Errores.
- Política FIFO.
- Eventos: Lightweight Events.
- Semáforos: Lightweight Semaphores.
- Mutex: Lightweight Mutex.
- Mensajes: Lightweight Messages.
- Timers: Lightweight Timers.

En la figura B 1 se observan las herramientas tanto de MQX y de MQX lite.

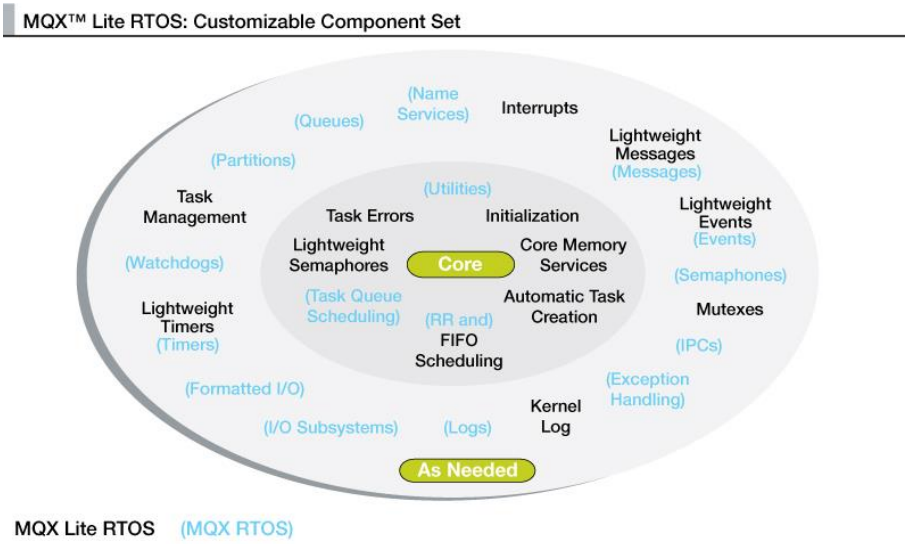


Figura B 1. MQX lite y MQX RTOS.

Características:

- Es fácilmente configurable dentro de CodeWarrior 10.5.
- Se puede agregar a una aplicación ya existente que no haya sido creada desde el principio con MQX Lite.
- El código creado en MQX lite funciona en **MQX™ Software Solutions.**
- Funcionamiento basado en prioridades (Preemptive Kernel).
- No es un software independiente, está integrado como un componente de Processor Expert.
- Compatible con otros compontes de Processor Expert.

Conceptos Básicos.

Un Sistema operativo en tiempo real como MQX Lite, gestiona los recursos del CPU (memoria, periféricos de entrada/salida, hardware) mediante actividades llamadas **tareas**, estas tareas son ejecutadas mediante un conjunto de reglas (llamada política) que define el orden en que serán ejecutadas las tareas. Enseguida se describe con más detalle los conceptos mencionados.

Procesamiento Multitarea.

EL procesamiento multitarea es una característica de los RTOS que permiten ejecutar una lista de tareas compartiendo el mismo CPU, lo que admite dar servicio a más de una tarea a la vez.

Kernel.

El kernel es un programa que forma parte del RTOS, el cual es el responsable del manejo de las tareas y la comunicación entre éstas. El principal servicio que ofrece el kernel es el de cambio de contexto, es decir, cuando el kernel decide ejecutar otra tarea, este simplemente guarda el contexto de la tarea activa, los registros actuales del CPU en el área de memoria de la tarea interrumpida, y coloca los registros de la nueva tarea a ejecutar en los registros del CPU.

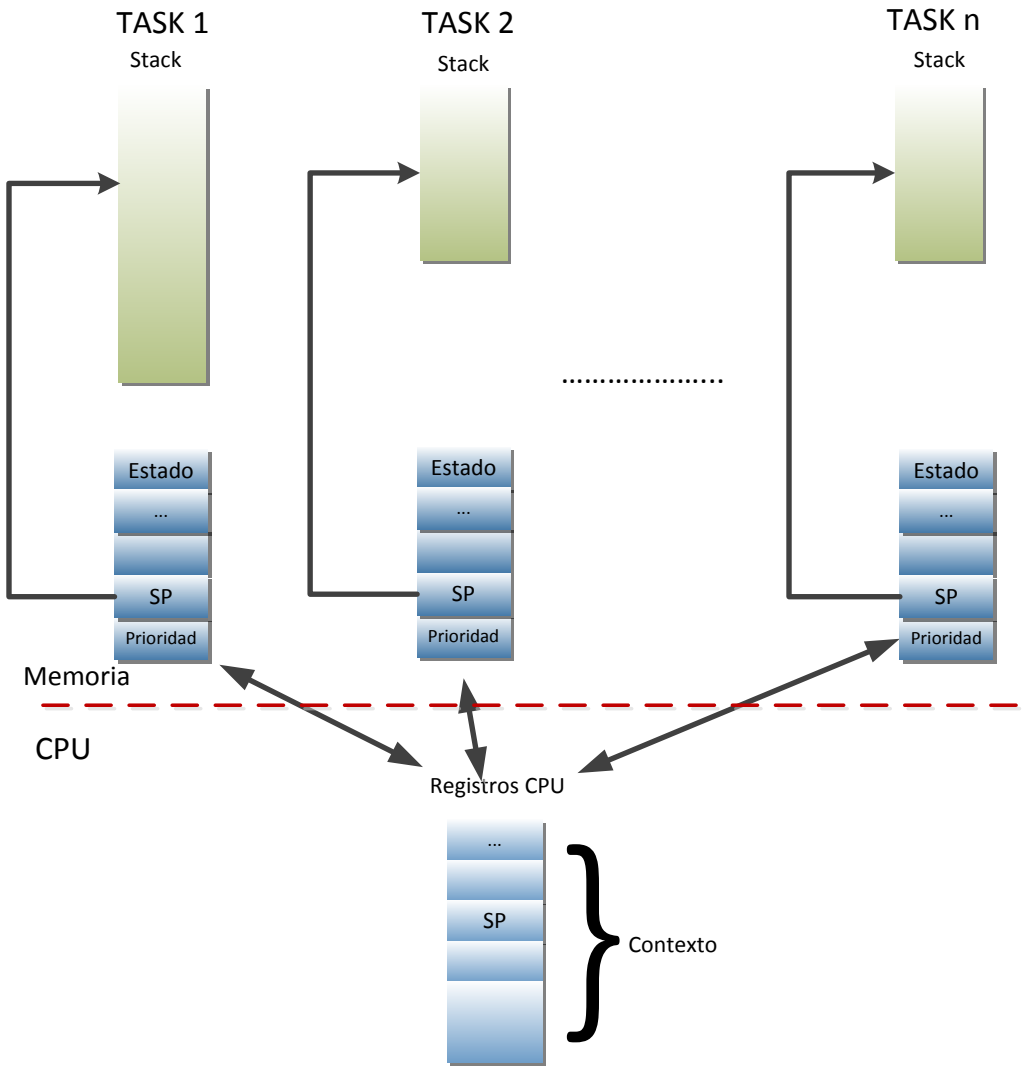


Figura B 1. Proceso Multitarea.

Scheduler.

El scheduler es la parte del kernel responsable de determinar cuál es la siguiente tarea a ejecutar. La mayoría de los RTOS están basados en prioridades, es decir a cada tarea se le asigna una prioridad basada en su importancia, por lo que el scheduler hace es verificar que el CPU ejecute la tarea más importante.

Secciones criticas de código.

Una sección critica de código, es un conjunto de líneas de código que debido a su importancia éste debe ser tratado individualmente. Una vez que esta sección inicie su ejecución esta no debe ser interrumpida y para asegurarlo un método es deshabilitar las interrupciones y cuando esta termine habilitarla nuevamente.

Recursos

Un recurso es cualquier entidad de software o hardware usada por una tarea de MQX. Un recurso puede ser por lo tanto un dispositivo de entrada o salida, tal como leds, un teclado, un display, una variable, una estructura de datos o un arreglo.

Los recursos pueden ser usados por varias tareas y a estos se les conoce como recursos compartidos. Las tareas deben ganar acceso exclusivo al recurso con el objetivo de evitar perdida de información.

Tareas

Una tarea es una porción de software que cumple con un objetivo determinado, y desde su perspectiva ésta tiene el control completo del CPU. Tienen la característica que se encuentran en un ciclo infinito, por lo que su ejecución nunca termina, a menos que se le indique explícitamente. En MQX lite es posible asignarles la prioridad, nombre, espacio que utilizará en bytes, y el tipo de tarea.

La prioridad, es un número que se le asigna a una tarea, el cual describe la importancia de ésta, por ejemplo una tarea con prioridad 8 se ejecutara antes que alguna otra con prioridad 9.

Cuando se genera un proyecto utilizando MQX Lite:

- Es necesario que haya al menos una tarea de AUTOSTART.
- La tarea creada por default tiene prioridad 8.
- Las prioridades de las tareas se pueden configurar a partir del número 8 hacia arriba, siendo ésta la más alta o más importante.
- Se pueden tener tareas con las mismas prioridades.
- Existe una tarea llamada idletask, y tiene la más baja prioridad de todas las creadas. Cuando todas las tareas están bloqueadas el CPU ejecuta idletask.
- Se le denomina Stack Size al espacio en bytes que utiliza una tarea.
- Tienen la estructura de una función de C.
- Es posible pasarle argumentos a las tareas.

Las tareas dependiendo del programa se pueden encontrar en distintos estados, los cuales se ilustran en la figura B.2.

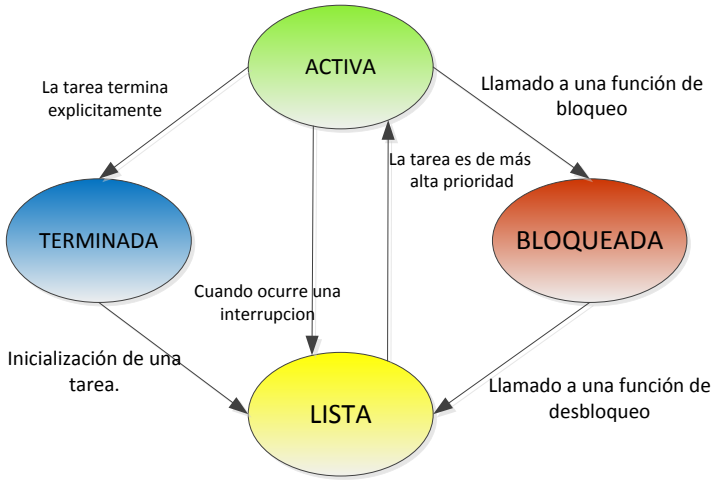


Figura B 2. Estados de las Tareas.

- **Tarea Activa.** Es la tarea que está ejecutando el CPU
- **Tarea Lista.** La tarea esta lista para ejecutarse pero tiene menor prioridad que la tarea activa.
- **Tarea Bloqueada**. La tarea se encuentra bloqueada, ya sea por un evento, un semáforo, o por un retardo de tiempo.
- **Tarea Terminada.** La tarea es terminada explícitamente.

Estructura de una tarea en MQX

En CodeWarrior la estructura de una tarea tiene el siguiente formato:

mqx_tasks.c

```
55 void Task1_task(uint32_t task_init_data)
56 {
57   int counter = 0;
58
59   while(1) {
60     counter++;
61
62     /* Write your code here ... */
63
64   }
65 }
```

Figura B 3. Estructura de una Tarea.

Política FIFO

Como ya se mencionó, la política FIFO se basa en prioridades de las tareas, por lo que:

MQX ejecutará la tarea que tenga la más alta prioridad y tenga más tiempo esperando.

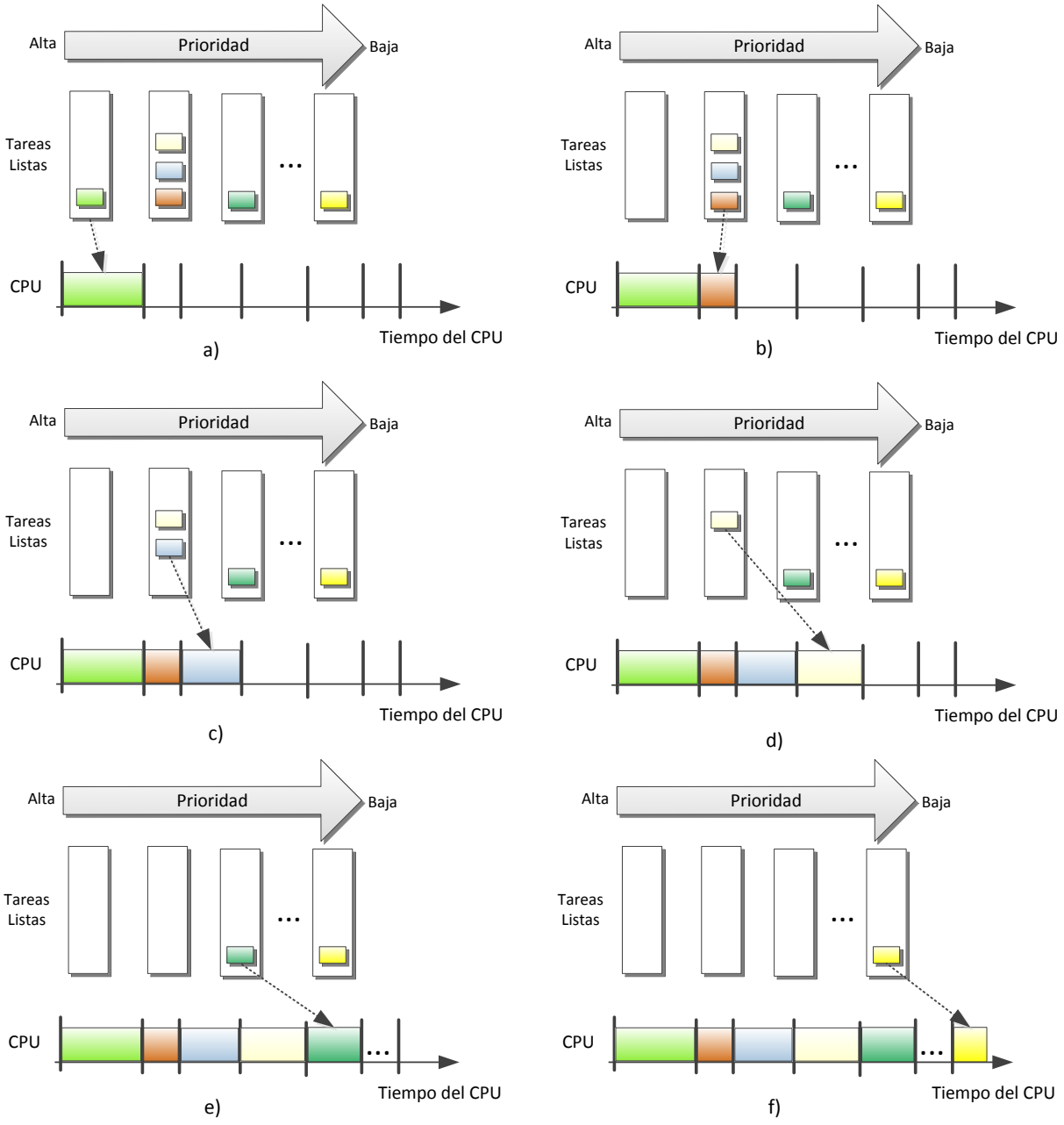


Figura B 4. Política FIFO.

Como se puede observar en B.4.a se ejecutará la tarea de más alta prioridad, en este caso el bloque de color verde, en B4.b se observa que hay 3 tareas que tienen la misma prioridad, por lo que se

ejecutará la que tenga más tiempo esperando, este proceso se repite hasta B4.d. En B4.e y B4.f se ejecutan las tareas de menos prioridad.

Funciones utilizadas de MQX lite

En la siguiente sección se describen las funciones utilizadas en este trabajo.

```
_lwevent_create(&serie_event,LWEVENT_AUTO_CLEAR);
```

Crea un objeto del tipo evento el cual consta de un grupo de 32 bits, los cuales funcionan como un grupo de banderas, por las cuales se puede preguntar el estado (1 o 0) de cada una de éstas y a partir de este valor poder sincronizar las tareas con otras tareas o con interrupciones.

Argumentos:

&serie_event. Apuntador a una variable del tipo LWEVENT_STRUCT
LWEVENT_AUTO_CLEAR. Es decir el comportamiento que tendrán las banderas, es decir si estas son de Autoclear o no.

Valor de retorno

```
_lwevent_set(&serie_event,0x01);
```

Pone en 1 los bits especificados en el segundo argumento.

Argumentos

Apuntador a una variable del tipo LWEVENT_STRUCT
El segundo argumento especifica el número de bit que se pondrá en 1

```
_lwtimer_create_periodic_queue(&my_periodic_queue,period_queue,wait_ticks);
```

Crea una estructura llamada cola de temporizadores, la cual tiene la característica de que es periódica. El periodo de esta función está basado en el reloj del sistema de MQX LITE llamado **SystemTimer1.**

Argumentos

&my_periodic_queue. Apuntador a una variable del tipo *LWTIMER_PERIOD_STRUC,* declarada en el archivo lwtimer.h

period_queue. Periodo que tundra la cola periódica, en ticks.
wait_ticks. Número de ticks a esperar antes de ejecutar la estructura de datos.

```
_lwtimer_add_timer_to_queue(&my_periodic_queue,&adc_ch0,adc_delay_ch0,(LWTIMER_ISR_FPTR)measure,(pointer)CH_0);
```

Esta función agrega un lwtimer a la cola periódica

Argumentos:

- *&my_periodic_queue*. Apuntador a una variable del tipo *LWTIMER_PERIOD_STRUCT,* declarada en el archivo lwtimer.h
- *&adc_ch0*. Puntero al timer que se agregara a la cola. Este argumento es del tipo LWTIMER_STRUCT
- *adc_delay_ch0*. Representa el tiempo en ticks despues de la creación de la cola periódica, que el lwtimer entrará en ejecución.
- *(LWTIMER_ISR_FPTR)measure*. Puntero a la función que el timer llamará cuando el timer expire.
- *(pointer)CH_0*. Argumento que se le pasará a la función.

```
_lwevent_wait_ticks(&buffer_ready,FIRST_HALF + SECOND_HALF ,FALSE,time)
```

Esta función pregunta si los bits del segundo argumento se encuentran en alto, si esto no es así la tarea se bloquea y por lo tanto no ejecuta las instrucciones que se encuentran después de esta línea. Si estos bits están en alto entonces el código que se encuentre después de esta línea se ejecutan, y si al momento de crear el evento éste se creó como de AUTOCLEAR, entonces los bits especificados se ponen en cero.

Argumentos:

&buffer_ready. Apuntador a una variable del tipo *LWEVENT_STRUCT*

FIRST_HALF + SECOND_HALF. Bits por los que la función pregunta si están en alto.

FALSE. Este argumento le especifica a la funcion que esperará a que cualquiera de los bits a que estén en alto.

Time. Número de ticks a esperar por los eventos. Es un argumento con el cual se puede asegurar el correcto funcionamiento del programa.

```
_time_delay_ticks(300);
```

Esta función suspende la tarea por el número de ticks especificado en el argumento. Una vez que se ejecuta esta instrucción el sistema operativo por medio de la política que maneja decide que tarea ejecutar, por lo que esta función no es una función de retardo de tiempo ya que no ata al microcontrolador en una línea de código.

Argumento.

Este argumento es un numero el cual especifica el número de ticks el cual la tarea va a estar suspendida.

_task_block();

Esta instrucción bloquea la tarea que la ejecute. Para ejecutar nuevamente esta tarea se tiene que llamar a la función _task_ready();

_lwevent_wait_for(&serie_event,0x01,FALSE,NULL);

Esta función es parecida a la función _lwevent_wait_ticks(), con la diferencia de que en está funcion no se especifica el número de ticks a esperar.

Argumentos:

&serie_event. Apuntador a una variable del tipo *LWEVENT_STRUCT*

0x01. Bits por los que la función pregunta si están en alto.

FALSE. Este argumento le específica a la función que esperará a que cualquiera de los bits a que estén en alto.

NULL.

_task_get_id();

Esta función obtiene el Task_id de la tarea activa.

Retorno

Regresa un número que identifica a la tarea dentro de la plantilla de tareas de MQX lite.

_task_ready(_task_get_td(task5_id));

Poner una tarea bloqueada en la lista de tareas listas es la función de esta instrucción.

Argumentos

(_task_get_td(task5_id). Puntero al task descriptor.

_task_get_td(task5_id. Obtiene un puntero al task descriptor.

Argumentos:

task5_id. Representa un número el cual identifica a una tarea dentro de la plantilla de tareas.

APÉNDICE C. Esquemáticos

En este apéndice se muestran los esquemáticos utilizados en este trabajo, realizados en el software libre Kicad.

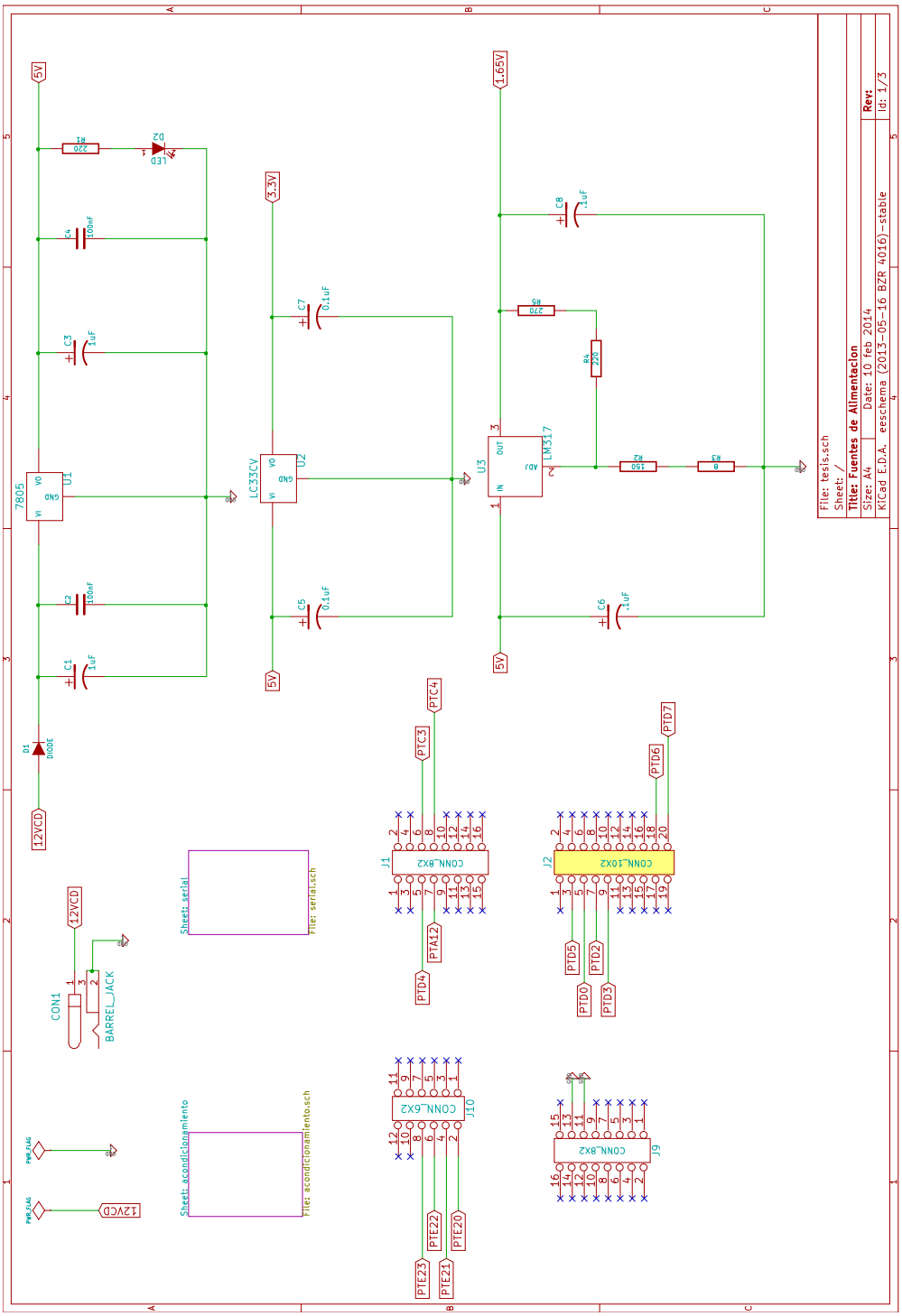


Figura C 1. Fuentes de Alimentación

Circuitos de Acondicionamiento

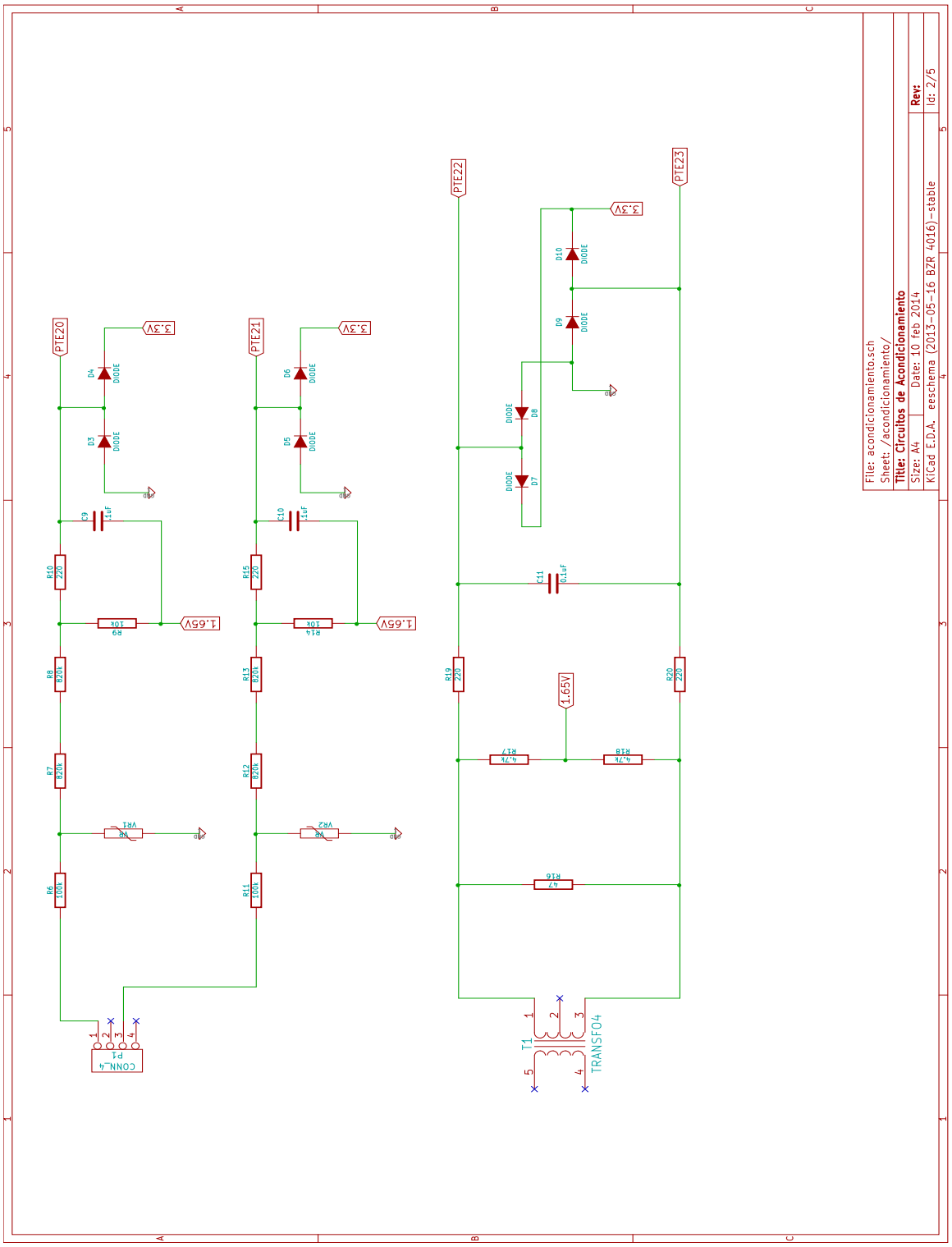


Figura C 2. Circuitos de Acondicionamiento de tensión y de corriente.

Comunicación Serial y SPI.

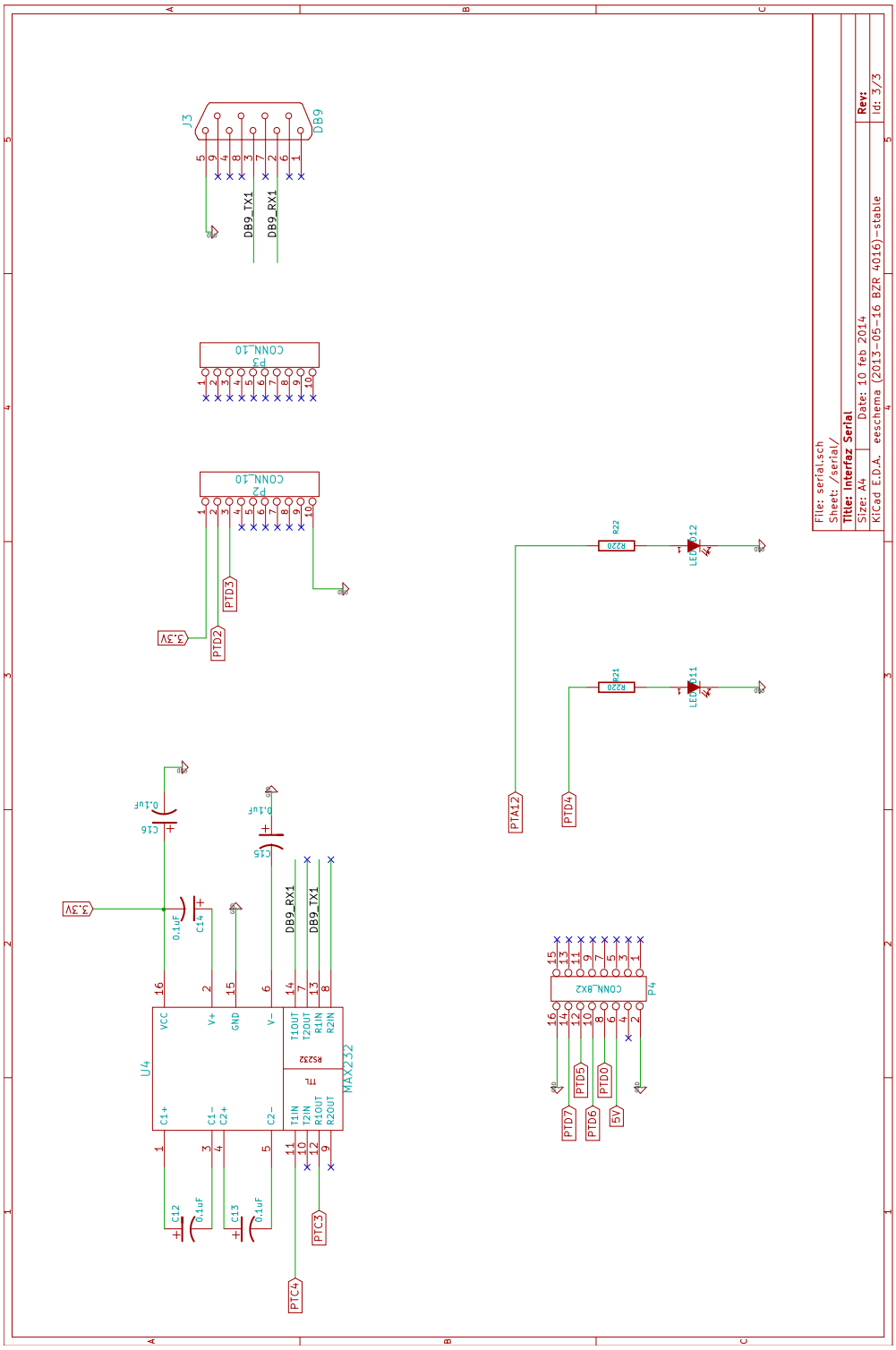


Figura C 3. Circuitos de Comunicación: Serial y SPI.

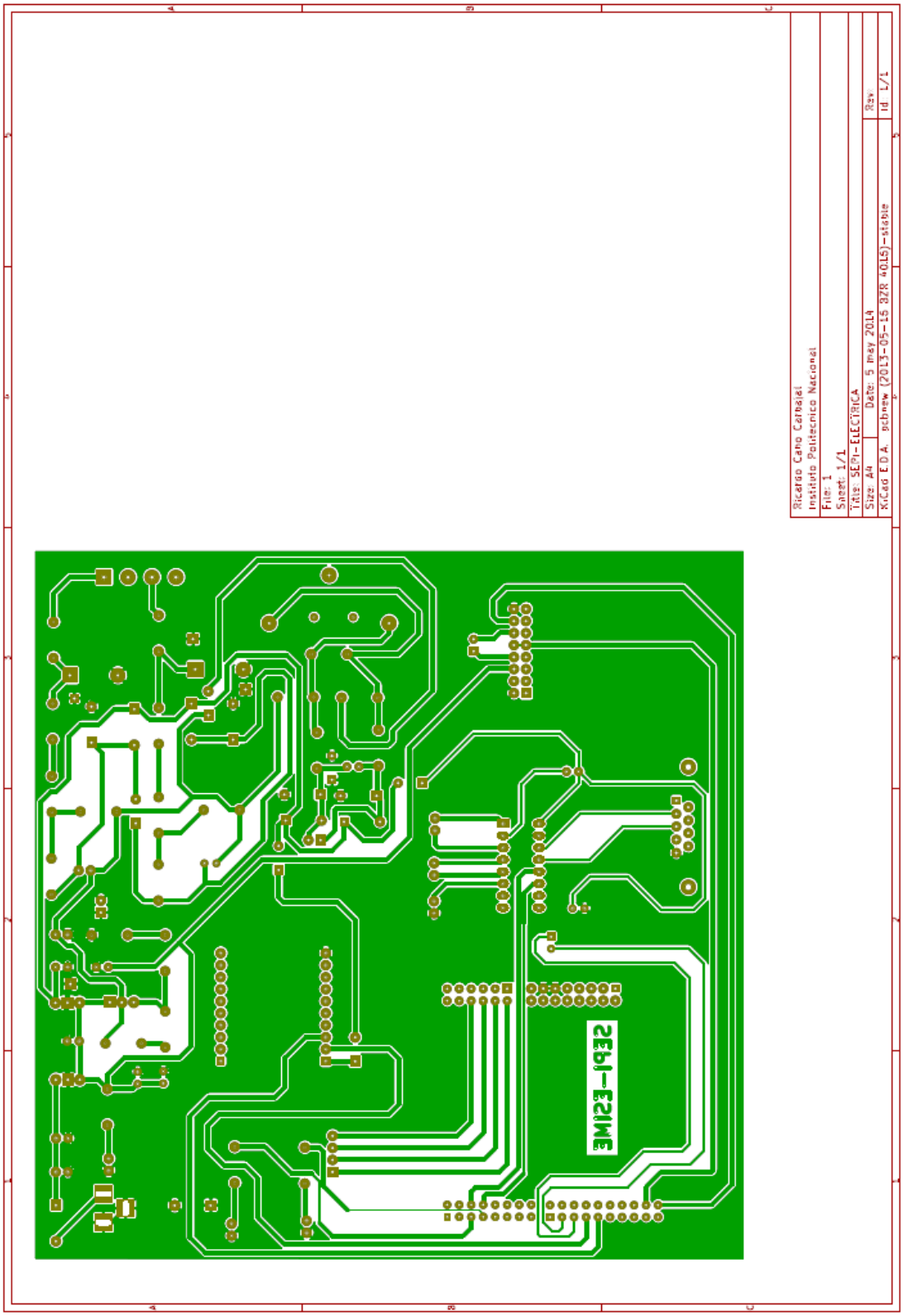


Figura C 4. Diagramas generados por Kicad. Capa Button

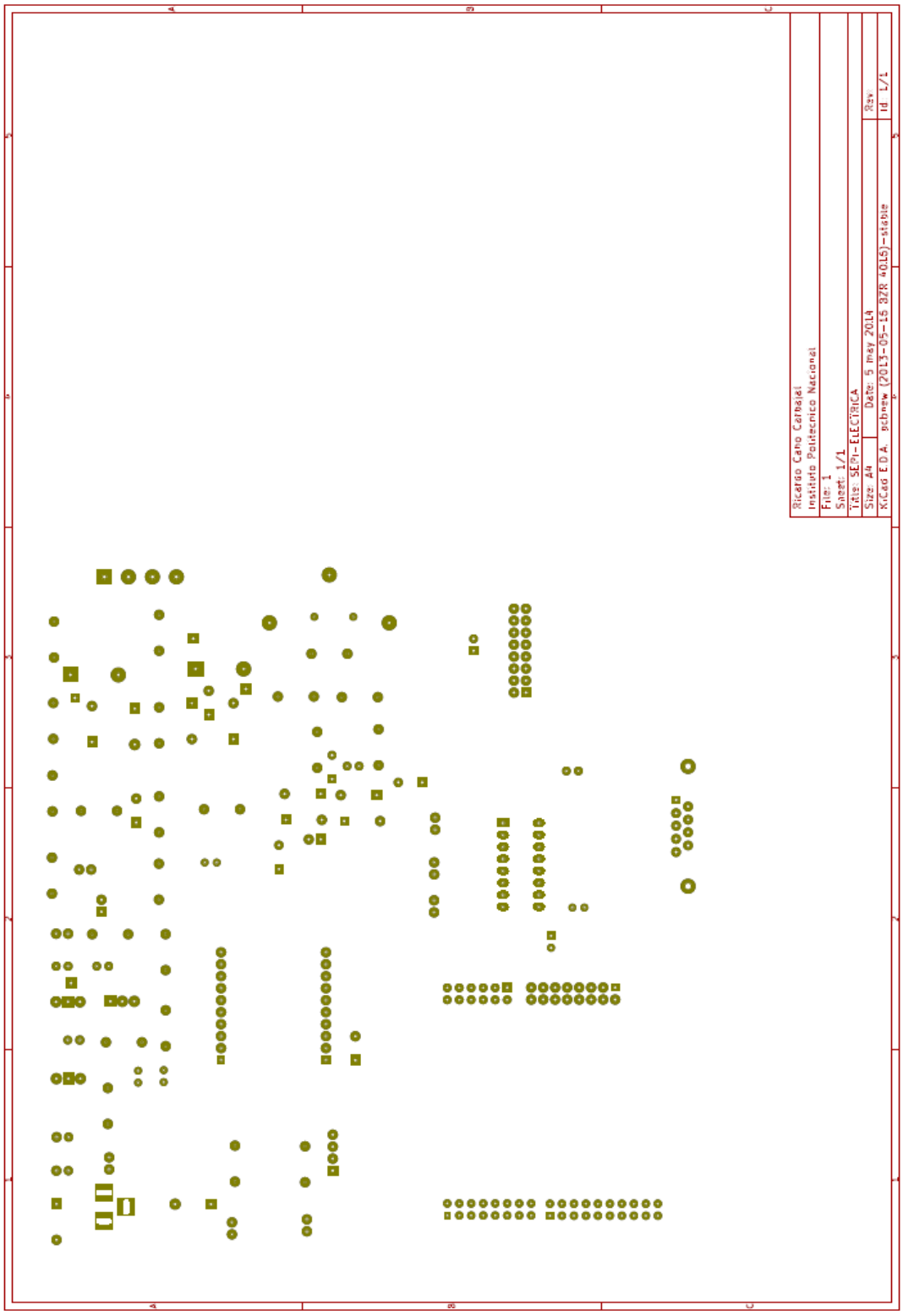


Figura C 5. Diagrama de esmerilado

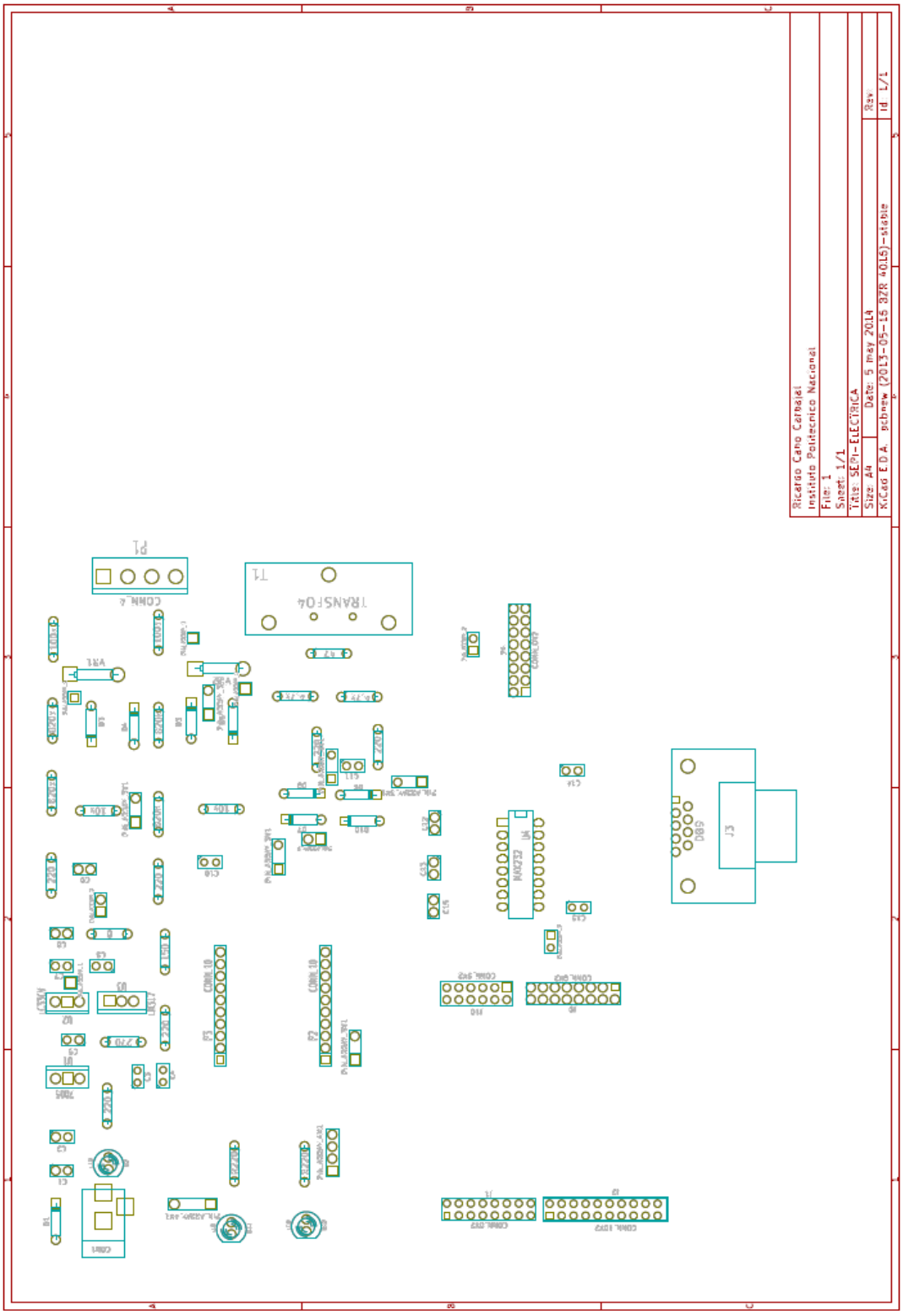


Figura C 6. Diagrama de distribución de Componentes

talema

AC1050 • 50 Amp Current Transformer

Low Cost 50/60Hz Current Transformers

Applications
- Sensing Overload Current
- Ground fault detection
- Metering
- Analog to Digital Circuits

- Competitive pricing due to high volume production
- Manufactured in an ISO-9001:2000, TS-16949:2002 and ISO-14001:2004 certified Talema facility
- Fully RoHS compliant

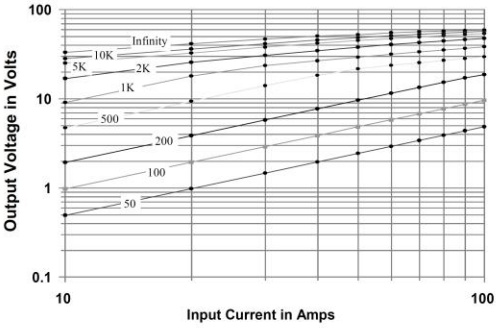
Electrical Specifications @ 20°C ambient

Electrical Specifications	
Primary Current	50A nom., 125A max.
Turns Ratio	1000:1 nominal
Volt per Amp Ratio at 50A for 100 ohm load	0.100 V/A
Volt per Amp Ratio at 5A for 100 ohm load	0.098 V/A
DC Resistance at 20°C	49.3 ohms
Dielectric Withstanding Voltage (Hi-pot)	4KVrms

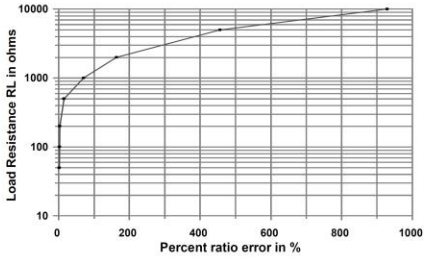
Mechanical Specifications	
Case	Polycarbonate
Encapsulant	Epoxy
Flammibility	Conforms to UL94-VO
Terminals	Pins Ø 1.0mm
Marking	TALEMA Date Code (W/Y) AC1050, Dot at start pin
Approximate Weight	47.3 grams
Tolerance	±0.2mm

Environmental Specifications	
Storage Temperature	-55° to +130°C
Insulation Resistance	100 megohms min.

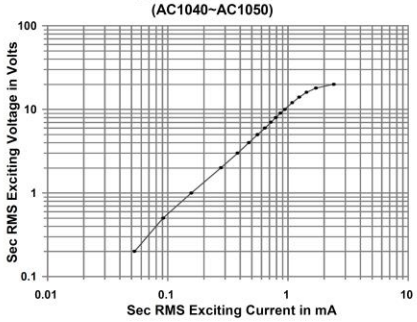
Output Volts vs Input Current
For various ohmic loads



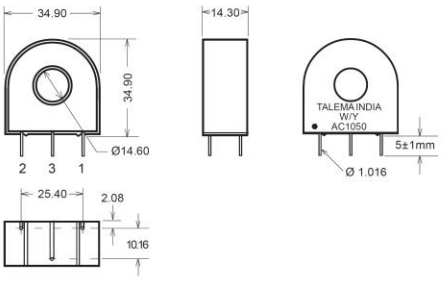
%RE vs RL at Rated primary current
(AC1050)



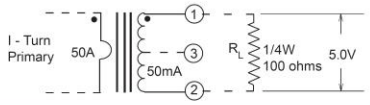
Typical Excitation Curve
(AC1040~AC1050)



Dimensions



Notes:
1) Unless requested, the terminating resistor and the one-turn primary are not supplied
2) Pin 3: Normally for mechanical support only but will be used on center tapped designs



Germany: Int.+49 89 - 841 00-0 • Ireland: Int.+35 374 - 954 8666 • Czech Rep: Int.+420 377 - 338 351 • India: Int.+91 427 - 244 1325
http://www.talema-nuvotem.com
(06-06)CT's\AC1050

Figura C 7 Hoja de datos del transformador de corriente

APENDICE D. SOFTWARE DGUS_SDK 4.9

Introducción

DGUS_SDK permite la creación de proyectos para las pantallas DWIN Technology. Un proyecto en este software consiste en un conjunto de imágenes predefinidas en algún software de diseño que en conjunto con las herramientas proporcionadas por DGUS_SDK es posible crear interfaces hombre-máquina.

DGUS_SDK cuenta con las siguientes herramientas:

Tabla D 1. Herramientas DGUS SDK.

Touch Config	Variable Config
➢ Pop up Window	➢ Variable Icon
➢ Variable Data input	➢ Animation Icon
➢ Incremental Adjustment	➢ Slider
➢ Slider Adjustment	➢ WordArt
➢ RTC Settings	➢ Image Animation
➢ Touch Control	➢ Icon Rotation
➢ Return Key Code	➢ Data Variable Display
➢ Ascci Input	➢ Text Dispaly
➢ Firmware Parameter Config Setting	➢ Analog Clock Display
	➢ RTC Display
	➢ Dymamic Curve Display
	➢ Table Display
	➢ Basic Graphic Display
	➢ Special Industrial Aplication
	➢ Bit Icon
	➢ Timer Variable

Las imágenes se usan como imagen de fondo y por medio de DGUS_SDK se definen los botones, ventanas emergentes, campos de entrada de datos etc., y de es esta forma se especifica el comportamiento que tendrá la pantalla.

El objetivo de este Apéndice es mostrar cómo se crean los proyectos en este software, no se incluye la forma en que funciona cada uno de los componentes de la tabla.

Pantalla Principal

Al ejecutar DGUS_SDK aparece la siguiente pantalla:

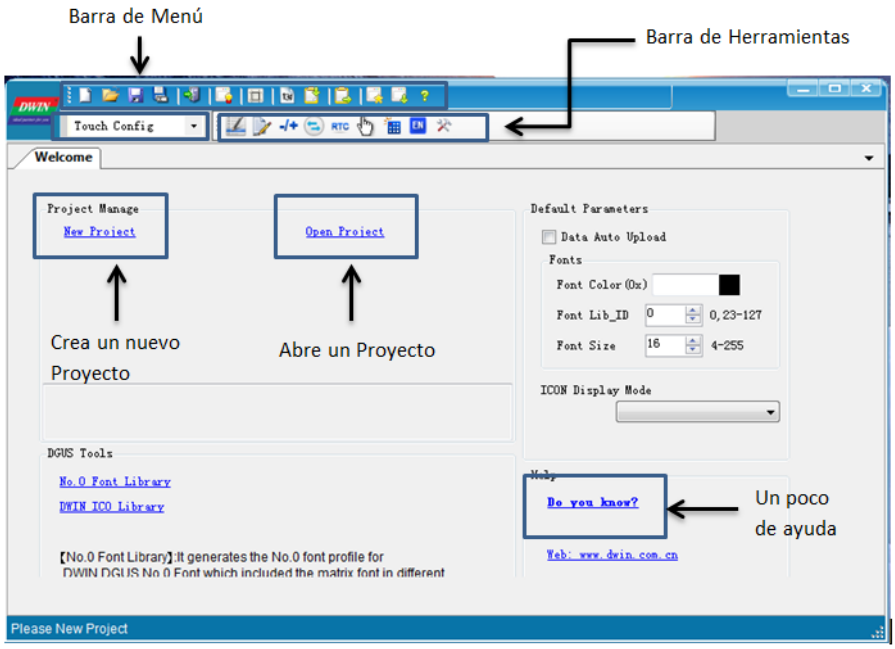


Figura D. 1. Pantalla Inicial de DGUS_SDK 4.9.

Modelo de Programación

Las pantallas DGUS adoptan el concepto de "Despliegue de Variables", es decir la información mostrada en la pantalla está basada en un conjunto de variables las cuales se encuentran en un archivo llamado ***14Variable_Config.bin***. Cuando un proyecto está terminado y listo para ser transferido a la pantalla se crean una serie de archivos que determinan el comportamiento de la misma.

Estos archivos son:

- **Config.txt**. Configura la velocidad de transmisión, el frame header, y determina si se habilita el sistema operativo.
- **13Touch_Control_Config.bin.** Si un botón es presionado este archivo contendrá información como:
  - Si salta a otra imagen o no.
  - Regresa un valor vía puerto serie.
  - Si modifica una dirección VP.
- **14Variable_Config.bin. Contiene la información referida a las direcciones de memoria (VP)**. Por ejemplo si en el proyecto está definido un campo **Data Variable,** la pantalla mostrara la información contenida en la dirección VP.
- **22_Config.bin. Archivo que contiene la inicialización de las variables del programa.**
- **23NameOfProject.bin.** Contiene el programa que el Sistema Operativo ejecutara periódicamente (80 / 120 / 160 /200 ms).

Variable Pointer: Se refiere a las direcciones de memoria definidas por cada elemento (botones, slider, tablas etc.) creado en el proyecto.

SP (Stack Pointer). Es la dirección inicial de memoria, donde se almacenan los datos referentes a los atributos de las variables, tales como color, tamaño, coordenadas etc.

Características de la pantalla

- Memoria para imagines: 224 MB
- Memoria para fuentes e iconos: 32 MB
- Memoria SRAM: 56 kB memoria VP
- Registros: 256 registros de 8 Bytes

Creación de un Proyecto.

Para crear un nuevo proyecto basta con dar clic en el icono  New Project, seleccionar la resolucion de la pantalla, y seleccionar la ubicación del proyecto. DGUS SDK, creará en la ubicación seleccionada los siguientes archivos y carpetas:

Carpetas:

- **DWIN_SET**. Esta carpeta es la que se transfiere a la pantalla (vía SD CARD), es decir contiene todas las imágenes, los archivos 13.bin, 14.bin ,22.bin, 22_Config.bin y 23.bin.
- **ICON.** Contiene los iconos usados por la pantalla
- **Image.** Contiene las imágenes predefinidas por el usuario**.**

Archivos:

- **DWprj.hmi**. Este archivo lo crea DGUS_SDK cuando se guarda el proyecto. Es decir es el archivo del proyecto

Una vez creado el proyecto la pantalla principal de DGUS_SDA aparecerá la figura D 2.

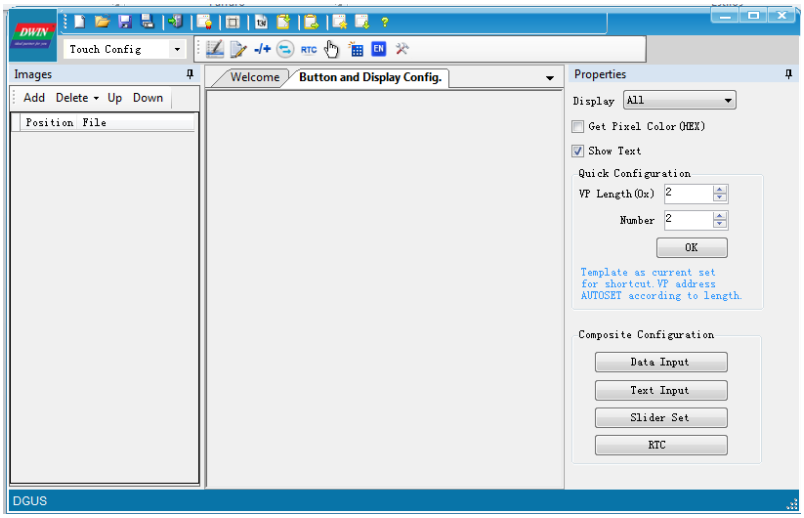


Figura D. 2. Pantalla de inicio del proyecto.

En esta pantalla ya es posible agregar las imágenes que serán usadas como fondo para el proyecto.

Agregar imágenes

Para agregar las imágenes se presiona el botón Add, el cual mostrará un cuadro de dialogo para seleccionar tanto la ubicación como las imágenes a agregar.

Las imágenes deben estar numeradas: 00_inicio, 01_menu, por ejemplo. El único formato admitido para las imágenes es bmp.

Une vez agregadas las imágenes, se pueden agregar los elementos (herramientas) que proporciona el software. Si el proyecto está terminado es necesario crear los archivos 13.bin, 14.bin, Config.txt, 22_config.bin, y esto se hace con el icono  ubicado en la barra de menús. También se debe configurar el sistema mediante el icono  , aquí se configura:

- Velocidad de Transmisión.
- Frame Header
- Habilitación del Sistema operativo.

APENDICE E. Microcontroladores ARM

El microcontrolador utilizado en este trabajo está diseñado bajo la arquitectura ARM Cortex- M0+, desarrollada por la empresa ARM Holdings.

ARM Holdings es una empresa líder en el desarrollo intelectual de semiconductores, quien ha vendido sus diseños a más de 350 fabricantes de microcontroladores como: Texas Instruments, Samsung, Freescale, Toshiba etc. ARM autoriza el uso y modificación de este diseño a las empresas fabricantes de microcontroladores y son éstas las que producen los procesadores finales basados en el diseño original y adecuándolos para la aplicación que requiera el fabricante.

La arquitectura ARM se diseñó para permitir implementaciones de tamaño reducido y de alto rendimiento, algunas de las características de este diseño son:

- Set de instrucciones reducido (RISC, Reduce Instruction Set Computer).
- Instrucciones de 32 bits.
- Todas las instrucciones se ejecutan en un ciclo de reloj.
- Pipeline.

Microcontroladores ARM Cortex-M0

El procesador ARM Cortex-M0+ es el procesador más eficiente de los microcontroladores ARM disponibles. A continuación se enlistan las características de estos procesadores:

- Bajo consumo de energía. El procesador Cortex M0+ consume 9.8µW/MHz, lo que lo hace atractivo para dispositivos portátiles.
- Simplicidad. Cuenta con un set de 56 instrucciones.
- Conectividad. Está diseñado para dar soporte a dispositivos de bajo consumo tales como Bluetooth Low Energy (BLE).

En la figura E1, se muestra la distribución de un microcontrolador ARM Cortex M0+.

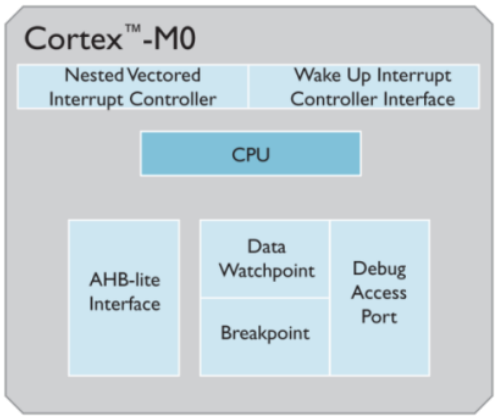


Figura E 1. Arquitectura del procesador ARM Cortex M0+.

APENDICE F. FreeMaster

Freemaster es una herramienta de visualización grafica de informacionen tiempo real, que permite la depuración, lectura y escritura en memoria del microcontrolador.

Freemaster fue una herramienta diseñada principalmente para el área de control de motores, pero debido a su flexibilidad y capacidad para visualizar fenómenos de alta velocidad puede ser utilizado para proyectos de distinta índole. En este trabajo FreeMaster fue utilizado para observar las señales de tensión y corriente que se adquirían de la red, además se utilizó como herramienta en el proceso de escalamiento de las variables eléctricas. En la figura F1 se observa la ventana de inicio de esta herramienta, la cual se encuentra en la versión 1.4 en el momento de realización de este trabajo.

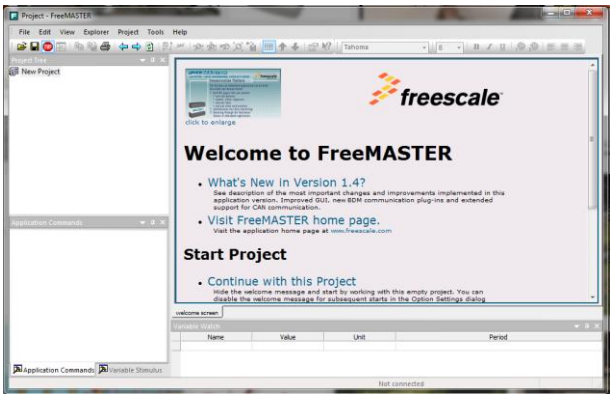


Figura F 1. Ventana de inicio de Freemaster

La comunicación del uC KL25Z128 con la PC es vía puerto serial y los datos de configuración son los siguientes: velocidad de transmisión de 115200, formato de 8 bits, 1 bit de parada y sin bit de paridad

Freemaster cuenta principalmente con dos herramientas llamadas: Scope y Recorder, las cuales son:

Scope. Permite visualizar la información de la forma en que la mostraría un osciloscopio tradicional. Es muy importante que la señal analizada no sea de una señal demasiado rápida ya que de lo contrario lo que veríamos seria el fenómeno de Aliasing debido a la baja velocidad de transmisión.

Recorder. Con esta herramienta se pueden analizar señales de alta velocidad, ya que ésta permite guardar la información a la misma velocidad que se está generando. Por lo que esta herramienta le proporciona al programa un buffer de memoria, donde la función recorder guarda la información. Una vez que el buffer está lleno, el uC transfiere la información a Freemaster y éste la muestra a la velocidad del fenómeno que se esté midiendo. En la figura se puede ver la información de tensión y corriente adquiridas de la tarjeta de adquisición de datos.

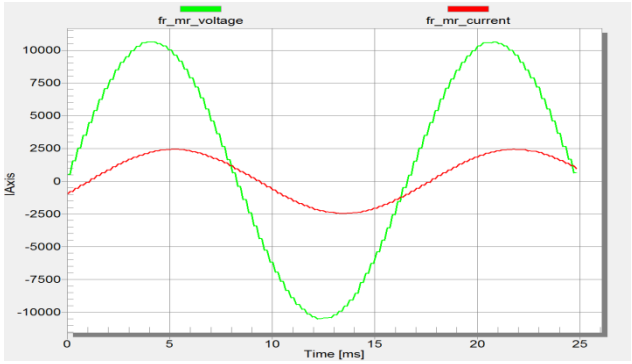


Figura F 2. Freemaster función recorder.

A continuación se muestran las configuraciones del bean freemater.

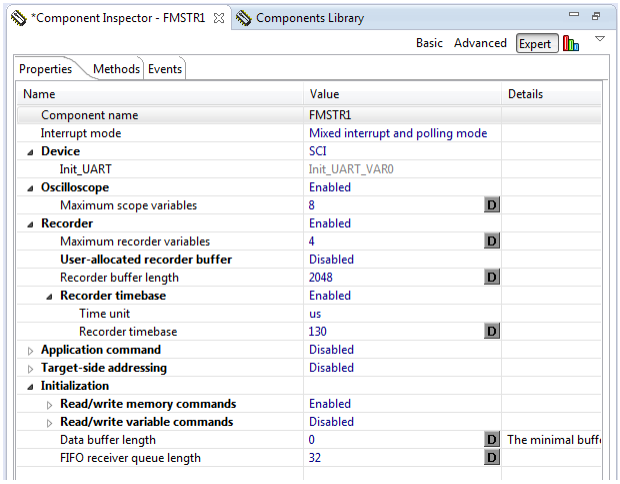


Figura F 3. Configuración del bean Freemaster.

De las configuraciones más importantes es:

- La utilización del puerto serie como protocolo de comunicación.
- La habilitación de las herramientas osciloscope y recorder.
- La configuración del tiempo base.

La configuración del tiempo base, es una constante de tiempo que indica que tan frecuente se llama a la función recorder. Esta configuración es dependiente de la aplicación, para este caso 130 µs fue la configuración correcta para visualizar 128 muestras de dos señals (tensión y corriente) de 60 Hz.

La ubicación de estas funciones normalmente se ubican en el ciclo infinito programa principal, sin embargo la ubicación en este proyecto fue dentro de la interrupción del ADC, arrojando los mejores resultados.

Creación de un proyecto en Freemaster.

A continuación se muestra el procedimiento para crear un proyecto en Freemaster.

1. Abrir Freemaster
2. Es posible que aparezca un mensaje de error indicando que el puerto de comunicación no se pudo abrir, este mensaje es normal cuando se inicia un proyecto nuevo en el programa.
3. Ahora hay que indicarle el puerto de comunicación, con el que Freemaster hará comunicación con el microcontrolador. Seguir la siguiente ruta: Project > Options >Direct RS232 y aquí elegir el número de puerto COM, así como la velocidad de transmisión.
4. Indicar a freemaster la ubicación del archivo *.elf del projecto. Este archivo se encuentra en la carpeta Flash del directorio donde se encuentra el projecto completo. Los archivos *.elf contienen una lista de variables utilizadas en el programa así como las direcciones de memoria donde están ubicadas. En la figura se observa este procedimiento.

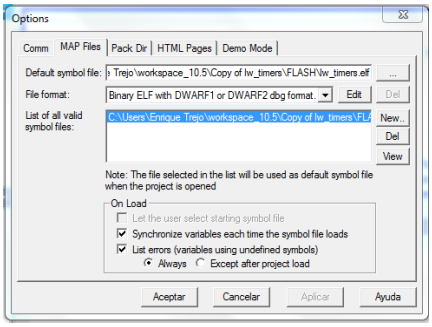


*Figura F 4. Selección del archivo *.elf.*

5. Ahora Freemaster ya tiene el archivo donde se encuentran las direcciones de memoria de las variables que se quieren analizar, sin embargo aún no se le ha dicho cuál es el nombre de estas variables. Seguir la siguiente ruta: Project > Variables y en la ventana emergente presionar el botón "Generate" y aparecerá otra ventana donde se mostrará una lista de variables que CodeWarrior genera cuando el proyecto se genera. Es en esta ventana se enlistan todas las variables globales que Codewarrior utiliza así como las variables declaradas por el usuario. Aquí se seleccionan las variables a analizar y se presiona el botón "Generate single variables". Dar clic en Close, y aparecerá por ejemplo la siguiente ventana.

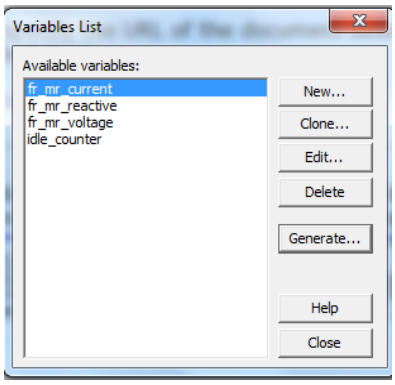


Figura F 5. Generación de variables.

6. Ahora hay que crear los bloques de Scope y Recorder. Dar clic derecho en el árbol del proyecto y agregar un subloque. Aparecerá una ventana y en la pestaña watch, dar clci en el botón "Add ->>"
7. En el subbloque creado agregar la herramienta Scope y/o Recorder.
8. En la ventana emergente dar clic en la pestaña Setup, y es aquí donde se escogen las variables a analizar.

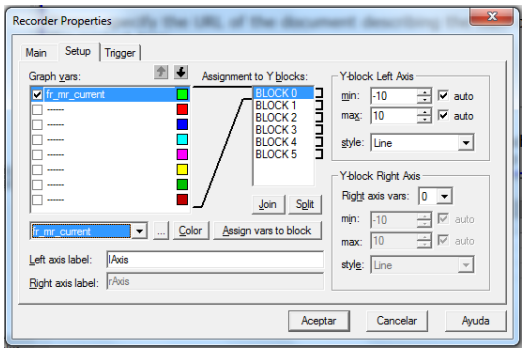


Figura F 6. Selección de las variables a graficar.

A continuación se muestran algunas de las imágenes que se capturaron con freemaster.

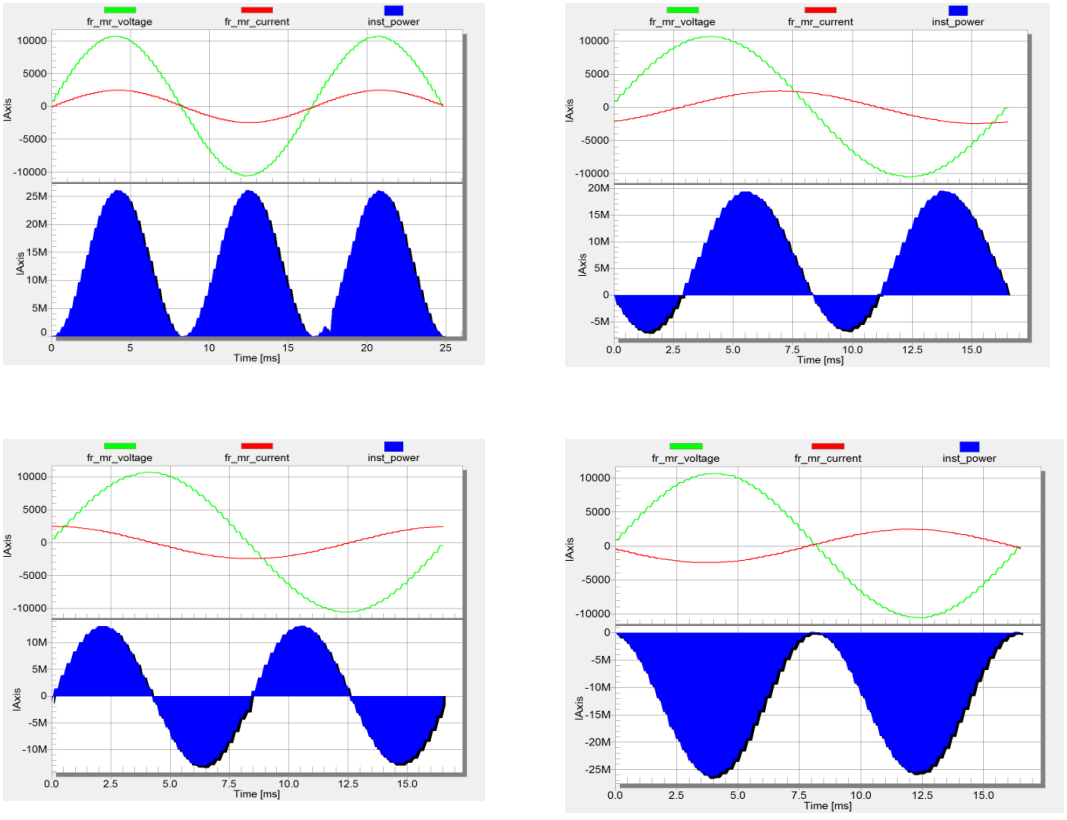


Figura F 7. Imágenes capturadas con FreeMaster

APENDICE G. Código del programa

```
/* ###################################################################
**     Filename    : mqx_tasks.c
**     Project     : ProcessorExpert
**     Processor   : MKL25Z128VLK4
**     Component   : Events
**     Version     : Driver 01.00
**     Compiler    : GNU C Compiler
**     Date/Time   : 2014-05-07, 13:49, # CodeGen: 0
**     Abstract    :
**         This is user's event module.
**         Put your event handler code here.
**     Settings    :
**     Contents    :
**         measurement_task - void measurement_task(uint32_t task_init_data);
**
** ###################################################################*/
/* MODULE mqx_tasks */

#include "Cpu.h"
#include "Events.h"
#include "mqx_tasks.h"
#include "lwtimer.h"
#include "lwevent.h"
#include "macros.h"

#ifdef __cplusplus
extern "C" {
#endif


extern int16 prev_sample;
extern int16 current_sample;
extern int32 frac_new_sample;
extern int32 frac_current_sample;
extern int32 sum_samples;
static FATFS fs;
static FIL fp;
FRESULT create_error;
FRESULT write_error;
FRESULT history_error;
uint32 kw_h = 0;
uint32 kw_h2 = 0;
TIMEREC time;
char data[19];


uint8 array[]= {0x5a,0xa5,0x7f,0x82,0x00,0xA0,0x00,0X04,0X00,0X0C,  //VP = 0004
                0X00,0X53,0X00,0X24,0X00,0X76,0X01,0X0E,0X11,0X11,
                0X00,0X90,0X00,0X24,0X0,0XB3,0X01,0X0E,0X22,0X22,
                0X00,0XCF,0X00,0X24,0X00,0XF2,0X01,0X0E,0X11,0X11,
                0X01,0X0B,0X00,0X24,0X01,0X2E,0X01,0X0E,0X22,0X22,
                0X01,0X47,0X00,0X24,0X01,0X6A,0X01,0X0E,0X11,0X11,
                0X01,0X83,0X00,0X24,0X01,0XA6,0X01,0X0E,0X22,0X22,
                0X01,0XC3,0X00,0X24,0X01,0XE6,0X01,0X0E,0X11,0X11,
                0X01,0XFF,0X00,0X24,0X02,0X22,0X01,0X0E,0X22,0X22,
                0X02,0X41,0X00,0X24,0X02,0X64,0X01,0X0E,0X11,0X11,
                0X02,0X7D,0X00,0X24,0X02,0XA0,0X01,0X0E,0X22,0X22,
                0X02,0XBD,0X00,0X24,0X02,0XE0,0X01,0X0E,0X11,0X11,
                0X02,0XF9,0X00,0X24,0X03,0X1C,0X01,0X0E,0X22,0X22};

/*
 * Estructura de datos de mediciones
 */
typedef struct
{
        uint16 v_rms;                                   //valor eficaz de la tension
        uint16 i_rms;                                   //corriente eficaz de la corriente
```

```
        signed long long total_act_energy;  //Energia activa Total consumida
        signed long long act_energy;        //Energia activa consumida por ciclo
        signed long energy_sum;             //kw-h consumidos
        signed long long act_pwr;           //Potencia activa
        unsigned long reac_pwr;             //Potencia reactiva
        signed long long total_reac_energy; //Potencia reactiva total consumida
        uint32 apr_pwr;                     //Potencia aparente
        int16 pf;                           //Factor de potencia
}POWER_STRUCT;


POWER_STRUCT  power;
/*TARIFAS 1A CFE2014*/
int16 voltage[2*N]={792,963,2817,
                    795,966,2826,
                    798,969,2835,
                    801,972,2844,
                    804,975,2853,
                    807,978,2862,
                    810,981,2871,
                    813,984,2880,
                    816,987,2889,
                    819,990,2898,
                    822,993,2907,
                    825,996,2917
};
;
int16 current[2*N];
//int16 react_array[2*N];

uint32 volt_scaled;
uint32 current_scaled;
uint32 power_scaled;
uint32 total_kw_h = 0;
uint32 frecuency;
uint32 apr_power_scaled;
uint32 pf_scaled;

/*
 * Variables para controlar la pantalla
 */
uint32 *ap_dir_tx[]={&volt_scaled,&current_scaled,&frecuency,
                     &total_kw_h,&pf_scaled,&apr_power_scaled};
int16 variable_pointer [] = {0x0005,0x0006,0x0007,0x0008,0x0009,0x000A};
int16 variable_pointer2 [] = {0x1000,0x1001,0x1002,0x1003,0x1004,0x1005,
          0x1006,0x1007,0x1008,0x1009,0x100A};

char tx_buff[20];
uint8 data_received[15];
char graph_buff_v[133];
char graph_buff_i[133];
char command_v[] = {0x5A,0xA5,0x05,0x82,0x50,0x06,0x00,0x64};  //Modifica el VD de las curvas
char command_i[] = {0x5A,0xA5,0x05,0x82,0x55,0x06,0x00,0x64};
DATEREC date;
DATEREC date2;
TIMEREC time;
TIMEREC time2;
uint8 days = 59;
char file_name[21];
uint16 tariff[3];
uint16 bill[11];


/*
 *                  Variables MQX Lite
 * Estas variables tienen el objetivo de crear componentes MQX lite como:
 *   > Eventos
 *   > Timers
 */
```

```
/*       Lightweight Timers
 * Los lightweight Timers crean una cola periodica de timers,
 * los cuales son agregados a esta estructura de datos y ejecutadas
 * secuencial y periodicamente.
 * El periodo de los timers esta definido por el reloj de MQX Lite.
 */

/* Declaracion de la cola periodica*/
LWTIMER_PERIOD_STRUCT my_periodic_queue;

/* Declaracion de los timers*/
LWTIMER_STRUCT adc_ch0;
LWTIMER_STRUCT adc_ch1;

LDD_TError error;

/*   Lightweight events
 * Los lwevents, son un componente opcional de MQX que crea grupos de 32 bits
 * por los que se puede preguntar por el valor de uno o varios de ellos, esto
 * con el objetivo de sincronizar tareas con otras tareas o con interrupciones.
 *
 */
/*Declaracion de eventos*/
LWEVENT_STRUCT serie_event;
LWEVENT_STRUCT buffer_ready;
_task_id task4_id,task5_id,task6_id,task13_id;

/*
 * Variables de inicializacion de perifericos
 */

LDD_TDeviceData *adc_ptr = NULL;
LDD_TDeviceData *serial_ptr = NULL;

extern uint8 init_direction;
extern uint8 final_direction;
extern uint8 cycle_number;
long long sum_energy = 0;


/*              Prototipos de funcion
 *
 */
void measure(unsigned char  group);
unsigned long rms(short *signal_data);
unsigned long  sqrt1(unsigned long long A);
void active_energy(int16 *v, int16 *i);
void energy_sum(POWER_STRUCT* aux);
void power_functions(short *v, short *i);
void scale_voltage(uint16 val_rms);
void scale_current(uint16 val_rms);
void reactive_energy(int16 *v, int16 *i);
void scale_active_power(long long val_power);
FRESULT f_init_fat32( FATFS *);
FRESULT open_file(FIL *,char*);
FRESULT f_close_file(FIL *);
uint8 sizestring(char *str);
uint8 add_offset(uint8 month);
void billing(dword kw_h);
word ptr;
byte error_flash = 0;
uint32 idle_counter = 0;
UINT bw_t = 0;

/* User includes (#include below this line is not maintained by Processor Expert) */

/*
** ===================================================================
**     Event       :  measurement_task (module mqx_tasks)
** ===================================================================
```

```
*/
void measurement_task(uint32_t task_init_data)
{
		//rtc = RTC1_Init(rtc,FALSE);
		TmDt1_GetTime(&time);
		(void)TmDt1_GetDate(&date);
		serial_ptr = AS1_Init(NULL);    //Inicializacion del puerto serial
		_lwevent_create(&serie_event,LWEVENT_AUTO_CLEAR);
		_lwevent_set(&serie_event,0x01);
		_lwevent_set(&serie_event,0x02);
		_mqx_uint period_queue = 10;    //Periodo de la cola 10 * 26.0416uS
		_mqx_uint wait_ticks = 0;       //Numero de ticks a esperar antes iniciar a procesar la cola de timers
		_mqx_uint adc_delay_ch0 = 0;    //Tiempo en ticks despues de la creacion de la cola periodica CH0
		_mqx_uint adc_delay_ch1 = 8;    //Tiempo en ticks despues de la creacion de la cola periodica CH1
		uint16 time = 640;//640
		adc_ptr = AD1_Init( NULL);      //Inicializacion dell ADC
		IFsh1_SetBlockFlash(voltage,0x19420,72);//guarda la tarifa en FLASH
		IFsh1_GetWordFlash(0x19414,&ptr);
		if(ptr == 0xFFFF)
		{
				ptr = 28300;
				IFsh1_SetLongFlash(0x19410,0);
				IFsh1_SetWordFlash(0x19414,0);
		}
		else
			IFsh1_GetWordFlash(0x19410,&ptr); //Lee los kwh acumulados
		total_kw_h = ptr;
		power.energy_sum = ptr;
		_lwevent_create(&buffer_ready,LWEVENT_AUTO_CLEAR);  //Creacion de un evento
		_lwevent_clear(&buffer_ready,0x1F);

		 _lwtimer_create_periodic_queue(&my_periodic_queue,period_queue,wait_ticks);  //Creacion de la cola periodica

_lwtimer_add_timer_to_queue(&my_periodic_queue,&adc_ch0,adc_delay_ch0,(LWTIMER_ISR_FPTR)measure,(pointer)CH_0);

_lwtimer_add_timer_to_queue(&my_periodic_queue,&adc_ch1,adc_delay_ch1,(LWTIMER_ISR_FPTR)measure,(pointer)CH_1);
		error_flash =IFsh1_GetLongFlash(0x19410,&ptr);
		while(1) {

				if( _lwevent_wait_ticks(&buffer_ready,FIRST_HALF + SECOND_HALF ,FALSE,time) == MQX_OK )
				{
		  active_energy(&voltage[init_direction],&current[init_direction]);
		  energy_sum(&power);
		  power_functions(&voltage[init_direction],&current[init_direction]);
		 //FMSTR1_Poll();
				}
				else
					RED_NegVal(RED_DeviceData);
		  /* Write your code here ... */

		 }

}

/*
** ===================================================================
**     Event       :  Serial_task (module mqx_tasks)
** ===================================================================
*/
void Serial_task(uint32_t task_init_data)
{
		while(1)
		{
		  _lwevent_wait_for(&serie_event,0x01,FALSE,NULL);
			AS1_ReceiveBlock(serial_ptr,data_received,9);
		 }
}
```

```
/*
** ===================================================================
**     Event       :  Scale_task (module mqx_tasks
** ===================================================================
*/
void Scale_task(uint32_t task_init_data)
{
		while(1)
		{
			 _lwevent_wait_for(&buffer_ready,0x04,FALSE,NULL);
			scale_voltage((uint16) power.v_rms);
			scale_current((uint16) power.i_rms);
			scale_active_power((long long)power.act_pwr);
			apr_power_scaled = (volt_scaled * current_scaled) / 10000;
			pf_scaled = (power.pf * 100)>> Q(15);
		}

}

/*
** ===================================================================
**     Event       :  Graph_screen_task (module mqx_tasks)
** ===================================================================
*/
void Graph_screen_task(uint32_t task_init_data)
{
		int16 max_v = 0;
		int16 max_i = 0;
		uint8 i = 0;
		uint8 j = 0;
		task4_id = _task_get_id();

		graph_buff_v [0]= 0x5A; //Header
		graph_buff_v [1]= 0xA5;  //Header
		graph_buff_v [2]= 130;  //Data number
		graph_buff_v [3]= 0x84;  //COMMAND
		graph_buff_v [4]= 0x01;  //Channel 0

   graph_buff_i [0]= 0x5A; //Header
		graph_buff_i [1]= 0xA5;  //Header
		graph_buff_i [2]= 130;  //Data number
		graph_buff_i [3]= 0x84;  //COMMAND
		graph_buff_i [4]= 0x02;  //Channel 1
		  while(1)
		  {
			while(data_received[8] == 0x02)
				{
				max_v = 0;
				max_i = 0;
				do{
				  if(abs(voltage[j])>= max_v)
				    max_v = abs(voltage[j] );
				  if(abs(current[j])>= max_i)
				  max_i = abs(current[j]);
				}while(++j <= 63);
				  command_v[6] = max_v >> 8;   //Modifica VD de tension
				  command_v[7] = max_v;
				  AS1_SendBlock(serial_ptr,command_v,8);
				  command_i[6] = max_i >>8;    //Modifica VD de corriente
				  command_i[7] = max_i;
				  max_v += 600;
				  max_i += 600;
				  for( i = 5, j= 0; i<= 132 ;i++,j++)
				  {
				   graph_buff_v[i] = (voltage[j] + max_v )>>8;
				   graph_buff_i[i] = (current[j] + max_i )>>8;
				   graph_buff_v[++i] = (voltage[j] + max_v );
				   graph_buff_i[i] = (current[j] + max_i );
				  }
				   i = 5;
```

```
                                        j = 0;
                                        AS1_SendBlock(serial_ptr,command_i,8);
                                        _time_delay_ticks(500);
                                        AS1_SendBlock(serial_ptr,graph_buff_v,133);
                                        _time_delay_ticks(700);
                                        AS1_SendBlock(serial_ptr,graph_buff_i,133);
                                        _time_delay_ticks(1000);  //700
                                    }
                                    _task_block();

            }

}
/*
** ===================================================================
**     Event      :  Write_screen_task (module mqx_tasks)
** ===================================================================
*/
void Write_screen_task(uint32_t task_init_data)
{
            task5_id = _task_get_id();
            static int i = 0;
             while(1) {
                        while(data_received[8] == 0x01)
                        {
                                    tx_buff[0] = 0x5a;   //Frame header
                                    tx_buff[1] = 0xa5;   //Frame header
                                    tx_buff[2] = 0x05;   //Data length
                                    tx_buff[3] = 0x82;   //Command: Write data into the specified variable SRAM
                                    tx_buff[4] = variable_pointer[i]>>8;   //Variable pointer
                                    tx_buff[5] = variable_pointer[i];       //Variable pointer
                                    tx_buff[6] = *ap_dir_tx[i]>>8;   //Data high byte
                                    tx_buff[7] = *ap_dir_tx[i];   //Data low byte
                                    i++;
                                          if(i>=6)
                                                i=0;
                                       AS1_SendBlock(serial_ptr,tx_buff,8);
                                    _time_delay_ticks(1500);  //Update every 1000 ticks
                        }
                        _task_block();
            }
}

/*
** ===================================================================
**     Event      :  History_task (module mqx_tasks)
** ===================================================================
*/
void History_task(uint32_t task_init_data)
{
            bool flag = 1;
            uint8 i = 0,j = 12;
            task6_id = _task_get_id();
             UINT br;      //bytes read
             UINT br2  = 0;
             uint8 size = 0;
             char month[11];
             int consumption;
             int32 x1 = 0, x2 = 0;
              while(1)
              {
                        f_init_fat32(&fs);
                        while(data_received[8] == 0x03 && flag == 1)
                        {
                        flag = 0;
                        history_error = open_file(&fp,"./Historial.txt");
                        for(i = 0, j = 12;i <= 12;i++,j+=10)
                                    {

                                    //At this point the file must already be created
```

```
                        history_error = FAT1_read(&fp,(void*)data,19,&br); //lee lo maximo que se pueda
                        size = sizestring(data);
                        sprintf(data,"                ");
                        FAT1_lseek(&fp,br2);
                    history_error = FAT1_read(&fp,(void*)data,size,&br);//lee hasta el salto de linea
                    br2+=br;
                        sscanf(data,"%4s %d",month,&consumption);
                        x1 = consumption * 100 / 600;
                        x2 = 100 - x1;
                        x2 = x2 * 270 /100;
                        x2 = x2 + (x1*32) /100;
                        array[j] = x2>>8;
                        array [j+1] = x2;
                        }
                        AS1_SendBlock(serial_ptr,array,130);
                        _time_delay_ticks(500);
                        f_close_file(&fp);
                        FAT1_mount(0,NULL);
                    }
                    br2 = 0;
                    flag = 1;
                    _task_block();
            }
}

/*
** ===================================================================
**     Event      :  Date_task (module mqx_tasks)
** ===================================================================
*/
void Date_task(uint32_t task_init_data)
{
  while(1) {
            (void)TmDt1_GetDate(&date2);
            TmDt1_GetTime(&time2);
            if(days == 60)  //En este punto han transcurrido 60 dias y es posible hacer el calculo del costo de la energia
                    {
                    days = 1;
                    IFsh1_SetLongFlash(0x19480,total_kw_h); //guarda los kwh de los ultimos 60 dias
                    total_kw_h = 0;
                    kw_h2 = 0;
                    _lwevent_set(&serie_event,0x08);  //60 dias han transcurrido
                    }
            _time_delay_ticks(38400);

  }
}

/*
** ===================================================================
**     Event      :  Frecuency_task (module mqx_tasks)
** ===================================================================
*/
void Frecuency_task(uint32_t task_init_data)
{
          uint8 i = 0;
  while(1) {
            if( _lwevent_wait_for(&buffer_ready,0x10 + 0x100 ,TRUE,NULL) == MQX_OK /*&& data_received[8] == 0x01*/)
            for(i = 0 ; i < 2*N; i++)
            {

              if(voltage[i] > 0 && voltage[i+1]<0)
              {
                    prev_sample = voltage[i];
                    current_sample = - voltage[i+1];
                    frac_new_sample = (current_sample * 1000) / (current_sample + prev_sample);
                    frac_current_sample = 1000 - frac_new_sample;
                    sum_samples += frac_current_sample;
                    frecuency = (uint32)((long)SAMPLING_FRECUENCY / (long)sum_samples);
                    sum_samples = frac_new_sample;
```

```
                }
                else
                {
                        sum_samples += 1000;
                }
              }
 }
}

/*
** ===================================================================
**     Event       :  Idle_task (module mqx_tasks)
** ===================================================================
*/
void Idle_task(uint32_t task_init_data)
{

 uint32 counter;
  while(1) {
                if(counter++ >= 1000000)
                {
                idle_counter++;
                counter = 0;
                }

    /* Write your code here ... */

  }
}

/*
** ===================================================================
**     Event       :  Create_file_task (module mqx_tasks)
** ===================================================================
*/
void Create_file_task(uint32_t task_init_data)
{

 //Esta tarea se ejecuta cada 24 horas
  while(1) {
                _lwevent_wait_for(&serie_event,0x02,FALSE,NULL);
                f_init_fat32(&fs);
               // sprintf(file_name,"./file%2d_%2d_%2d.txt",date2.Day,date2.Month,date2.Year);
                sprintf(file_name,"./file%2d_%2d_%2d.txt",date2.Year,date2.Month,date2.Day);
                create_error = open_file(&fp,file_name);
                create_error =          f_close_file(&fp);
                FAT1_mount(0, NULL); /* unmount file system */
                bw_t = 0;
    /* Write your code here ... */

  }
}

/*
** ===================================================================
**     Event       :  Write_file_task (module mqx_tasks)
**
**     Component   :  Task11 [MQXLite_task]
**     Description :
**         MQX task routine. The routine is generated into mqx_tasks.c
**         file.
**     Parameters  :
**         NAME            - DESCRIPTION
**         task_init_data  -
**     Returns     : Nothing
** ===================================================================
*/
void Write_file_task(uint32_t task_init_data)
{
 char data[30];
```

```
UINT bw = 0;

 while(1) {
              _lwevent_wait_for(&serie_event,0x04,FALSE,NULL);
              kw_h = total_kw_h - kw_h2;
              kw_h2 = total_kw_h;
              f_init_fat32(&fs);                    //Se monta el sistema de archivos Fat32
              sprintf(data,"Hora: %2d:%2d:%2d\t%d\r\n",(&time)->Hour, (&time)->Min, (&time)->Sec,kw_h);
              write_error = open_file(&fp,file_name);   //Se abre el archivo
              write_error = FAT1_lseek(&fp,bw_t);
              write_error = FAT1_write(&fp, data, sizestring(data) , &bw);
              bw_t += bw;
              write_error = f_close_file(&fp);
              FAT1_mount(0, NULL); /* unmount file system */
   /* Write your code here ... */

 }
}

/*
** ===================================================================
**     Event      :  Flash_task (module mqx_tasks)
**
**     Component   :  Task12 [MQXLite_task]
**     Description :
**         MQX task routine. The routine is generated into mqx_tasks.c
**         file.
**     Parameters  :
**         NAME            - DESCRIPTION
**         task_init_data  -
**     Returns     : Nothing
** ===================================================================
*/
void Flash_task(uint32_t task_init_data)
{
uint32 ptr = 0x19480;
  while(1)
  {
              _lwevent_wait_for(&serie_event,0x08,FALSE,NULL);
              IFsh1_GetBlockFlash(0x19420 + add_offset(date.Month),(byte*)tariff,6);
  }
}

/*
** ===================================================================
**     Event      :  Billing_task (module mqx_tasks)
**
**     Component   :  Task13 [MQXLite_task]
**     Description :
**         MQX task routine. The routine is generated into mqx_tasks.c
**         file.
**     Parameters  :
**         NAME            - DESCRIPTION
**         task_init_data  -
**     Returns     : Nothing
** ===================================================================
*/
void Billing_task(uint32_t task_init_data)
{
  task13_id = _task_get_id();
  uint8 counter = 0;
  long kw_h = 0;
  uint8 i = 0;
  while(1) {
              while(data_received[8]==0x04)
                        {
                         counter++;
                         IFsh1_GetLongFlash(0x19480,&kw_h); //kw_h contiene los kwh en el ultimo periodo de consumo de 60 dias
                         billing(kw_h);
                         tx_buff[0] = 0x5a;   //Frame header
```

```
                        tx_buff[1] = 0xa5;   //Frame header
                        tx_buff[2] = 0x05;   //Data length
                        tx_buff[3] = 0x82;   //Command: Write data into the specified variable SRAM
                        tx_buff[4] = variable_pointer2[i]>>8;   //Variable pointer
                        tx_buff[5] = variable_pointer2[i];       //Variable pointer
                        tx_buff[6] = bill[i]>>8;   //Data high byte
                        tx_buff[7] = bill[i];   //Data low byte
                        i++;
                       if(i>=11)
                         i=0;
                       AS1_SendBlock(serial_ptr,tx_buff,8);
                       _time_delay_ticks(1500);  //Update every 1000 ticks
                      }
              _task_block();


   /* Write your code here ... */

  }
}

/* END mqx_tasks */

#ifdef __cplusplus
}  /* extern "C" */
#endif

/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.3 [05.08]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/

/*
 * Funcion de disparo por software del ADC
 *
 *
 * Esta funcion es utilizada por los lwtimers y cumple con el objetivo de selecionar
 * el canal y disparar al ADC via software
 *
 * Parametro:
 *
 * uint8 group: Seleciona el grupo a medir
 *
 *
 * Valor de retorno: Ninguno
 *
 */
void measure(unsigned char  group)
{

              error = AD1_SelectSampleGroup(adc_ptr, group);
              error = AD1_StartSingleMeasurement(adc_ptr);
              Blue_NegVal(Blue_DeviceData);
}

/*
 *              Calculo de la energia consumida en un ciclo electrico
 *
 *
 * Parametros:
 *
 * int16 * . Direccion de memoria del vector de tension
 * int16 * . Direccion de memoria del vector de corriente
```

```
 * dlong   . Acumulador de energia.
 *
 */
void active_energy(int16 *v, int16 *i)
{
        uint8 j;
        long long temp = 0;
        for( j = 0 ;j < N; j++)
        {
                temp += ((*v) * (*i));                          //Calculo de la energia activa
                v++;
                i++;
        }
         sum_energy = temp;              //sum_energy solo es una variable auxiliar
        power.act_energy = sum_energy;      //Calculo de la energia activa en un ciclo
        power.total_act_energy += sum_energy;  //Sumatoria de la energia activa consumida

}

void reactive_energy(int16 *v, int16 *i)
{
        uint8 j;
        long long temp = 0;
        for( j = 0 ;j < N; j++)
        {
                temp += ((*v) * (*i));                          //Calculo de la energia reactiva
                v++;
                i++;
        }
        temp = temp >> 6;
        power.reac_pwr = temp;      //Calculo de la energia reactiva en un ciclo
        power.total_reac_energy += temp;  //Sumatoria de la energia reactiva consumida

}

/*
 * Sumatoria de la energia Consumida o generada
 * Parametro. Apuntador auxiliar del tipo POWER_STRUCT
 */

void energy_sum(POWER_STRUCT* aux)
{
        /*
         * Detecta si hay consumo o aportacion de energia
         */
        if(aux -> total_act_energy >= KW_H)
        {
                aux -> total_act_energy -= (long long)KW_H;
                aux-> energy_sum++;
                //total_kw_h += ++aux-> energy_sum;
                total_kw_h++;
           IFsh1_SetLongFlash(0x19410,total_kw_h);
        }
        else if (aux -> total_act_energy <= - KW_H)
        {
                aux -> total_act_energy += (long long)KW_H;
                //aux -> energy_sum--;
                total_kw_h = --aux-> energy_sum;
                IFsh1_SetLongFlash(0x19415,total_kw_h);
        }
}

/*
 * Calculo de potencias
 *
 */
void power_functions(short *v, short *i)
{
        power.act_pwr = power.act_energy >> 6;              //Calculo de la potencia activa por ciclo
        power.v_rms = rms (v);                             //Valor rms del voltaje
```

```
		power.i_rms = rms (i);                                  //valor rms de la corriente
		power.apr_pwr = (power.v_rms) * (power.i_rms);          //Potencia Aparente por ciclo


		if(power.apr_pwr < power.act_pwr)
		{
			power.apr_pwr = power.act_pwr;
			power.pf = (1 << 15 ) - 1;
			//power.total_reac_pwr = 0;
		}
		else
		{
			if(power.apr_pwr < abs(power.act_pwr))
				power.pf =  - (1 << 15 );
			else
				power.pf =  ((power.act_pwr  << Q(15) )  / power.apr_pwr);   // cos (theta) = P / S
			power.reac_pwr = sqrt1((unsigned long long) (POW_2((long long)power.apr_pwr) - POW_2(power.act_pwr ) ));

		}
}

/*
 * Valor Eficaz
 * Esta funcion cumple con el objetivo de calcular el valor RMS de una señal
 * ya sea de tension o de corriente.
 */

unsigned long rms(short *signal_data)
{
		uint8 i;
		unsigned long long sum = 0;
		for( i = 0; i < N; i++)
		{
			sum += POW_2(*signal_data);
			signal_data++;
		}

		sum = sum >> 6;
		return sqrt1(sum);

}

/*
 *		Funcion de raiz Cuadrada
 *
*/

unsigned long sqrt1(unsigned long long A)
{
   int j;
   uint32 Acc,Acc_temp;
   if (A == 1)
       return 1;
   else if(A == 0)
       return 0;
   else
   {
   Acc = A/2;
   for(j = 0;j<= MAX_ITE; j++)
   {
        Acc_temp =  (A/Acc + Acc) / 2;
        if((abs(Acc - Acc_temp) < 0x0001) || (Acc_temp == 0))
        break;
        Acc = Acc_temp;
   }
}
   return Acc_temp;
}

void scale_voltage(uint16 val_rms)
```

```
{
	if(val_rms <= 70 )
		volt_scaled = 0;
	else if (val_rms <= 587 )
		volt_scaled =  ((V_M1 * val_rms - V_B1 ) *  MUL_V) >> Q(15);
	else if(val_rms <= 5818)
		volt_scaled =  (V_M2 * val_rms - V_B2 ) *  MUL_V >> Q(15);
	else if(val_rms <= 11250)
		volt_scaled =  (V_M3 * val_rms - V_B3 ) *  MUL_V>> Q(15);
}

void scale_current(uint16 val_rms)
{
	if(val_rms <= 26)
		current_scaled = 0;
	else if (val_rms <= 49 )
		current_scaled =  ((I_M1 * val_rms - I_B1)  * MUL_I) >> Q(15);
   else if (val_rms <= 428 )
			current_scaled =  ((I_M2 * val_rms - I_B2 ) * MUL_I) >> Q(15);
   else if (val_rms <= 4329 )
			current_scaled =  ((I_M3 * val_rms + I_B3 ) * MUL_I) >> Q(15);
   else if (val_rms <= 20720 )
			current_scaled =  (((I_M4 * val_rms - I_B4 ) * MUL_I)- 5000) >> Q(15);
}

void scale_active_power(long long val_power)
{
	if(val_power <= 169993)
		val_power = 0;
	else if (val_power <= 331380)
			power_scaled = ((P_M1 * val_power - P_B1) * MUL_P) >> Q(19);
	else if (val_power <= 3128270)
				power_scaled = ((P_M2 * val_power - P_B2) * MUL_P) >> Q(23);
	else if (val_power <= 31652428)
				power_scaled = ((P_M3 * val_power - P_B3) * MUL_P) >> Q(23);
	else if (val_power <= 148180491)
				power_scaled = ((P_M4 * val_power + P_B4) * MUL_P) >> Q(20);
}



FRESULT f_init_fat32(FATFS *file_system)
{
	if (FAT1_isDiskPresent())
		return  FAT1_mount(0, file_system); /* mount file system */
}

FRESULT open_file(FIL * file,char *name_file)
{
	return FAT1_open(file,name_file, /*FA_OPEN_EXISTING*/FA_OPEN_ALWAYS | FA_WRITE | FA_READ);
}

FRESULT f_close_file(FIL * file)
{
	return FAT1_close(file); /* close file */
}


uint8 sizestring(char *str)
{
uint8 i = 0;
   while(str[i++]!='\n');
   return i;
}

uint8 add_offset(uint8 month)
{
	switch(month)
	{
	case 3:
```

```
                        return 6;
                        break;
                case 4:
                        return 12;
                        break;
                case 5:
                        return 18;
                        break;
                case 6:
                        return 24;
                        break;
                case 7:
                        return 30;
                        break;
                case 8:
                        return 36;
                        break;
                case 9:
                        return 42;
                        break;
                case 10:
                        return 48;
                        break;
                case 11:
                        return 54;
                        break;
                case 12:
                        return 60;
                        break;

                }
return 0;
                }

void billing(dword kw_h)
{
                if(kw_h > 15000)
                {
                        bill[0] = 150;
                        bill[1] = tariff[0];
                        bill[2] = bill[0] * bill[1] / 10;
                        kw_h -= 15000;
                }
                else
                {
                        bill[0] = kw_h /100;
                        bill[1] = tariff[0];
                        bill[2] = bill[0] * bill[1] / 10;
                        // lo de mas es cero
                        bill[3] = 0;
                        bill[4] = tariff[1];
                        bill[5] = 0;
                        bill[6] = 0;
                        bill[7] = tariff[2];
                        bill[8] = 0;
                        bill[9] = bill[0];
                        bill[10] = bill[2];
                        return;
                }
                if(kw_h > 13000)
                {
                        bill[3] = 130;
                        bill[4] = tariff[1];
                        bill[5] = bill[3] * bill[4] / 10;
                        kw_h -= 13000;
                }
                else
                {
                        bill[3] = kw_h /100;
                        bill[4] = tariff[1];
```

```
                        bill[5] = bill[3] * bill[4] / 10;

                        //lo de mas es cero
                        bill[6] = 0;
                        bill[7] = tariff[2];
                        bill[8] = 0;
                        bill[9] = bill[0] + bill[3] ;
                        bill[10] = bill[2] + bill[5];
                return ;
            }
            if(kw_h > 0)
            {
                        bill[6] = kw_h / 100;
                        bill[7] = tariff[2];
                        bill[8] = bill[6] * bill[7] / 10;
                        bill[9] = bill[0] + bill[3] + bill[6] ;
                        bill[10] = bill[2] + bill[5] + bill[8];


            }

}
```

EVENTS.C

```
/* ###################################################################
**     Filename    : Events.c
**     Project     : ProcessorExpert
**     Processor   : MKL25Z128VLK4
**     Component   : Events
**     Version     : Driver 01.00
**     Compiler    : GNU C Compiler
**     Date/Time   : 2014-02-11, 23:46, # CodeGen: 0
**     Abstract    :
**         This is user's event module.
**         Put your event handler code here.
**     Settings    :
**     Contents    :
**         Cpu_OnNMIINT - void Cpu_OnNMIINT(void);
**
** ###################################################################*/
/*!
** @file Events.c
** @version 01.00
** @brief
**         This is user's event module.
**         Put your event handler code here.
*/
/*!
**  @addtogroup Events_module Events module documentation
**  @{
*/
/* MODULE Events */

#include "Cpu.h"
#include "Events.h"
#include "mqx_tasks.h"
#include "macros.h"
//#include "TmDt1.c"

#ifdef __cplusplus
extern "C" {
#endif

extern uint8 days;
extern DATEREC date;
extern bool SD1_DataReceivedFlag;
extern LWEVENT_STRUCT serie_event;
```

```
extern LDD_TDeviceData *serial_ptr;
extern uint8 data_received[];
extern _task_id    task4_id;
extern _task_id    task5_id;
extern _task_id    task6_id;
extern _task_id    task13_id;


//extern uint16 frecuency;
extern int16 voltage[];
extern int16 current[];
//extern int16 react_array[];
//long inst_power;


int16 fr_mr_voltage;
int16 fr_mr_current;
//int16 fr_mr_reactive;
extern LWEVENT_STRUCT buffer_ready;
extern DATEREC date2;
extern TIMEREC time;
extern TIMEREC time2;
uint8 k1 = 5;
uint8 k2 = 0;
uint8 k3 = N/4;
uint8 k4 = 0;
uint8 cycle_number = 0;
uint8 init_direction;
uint8 final_direction;
int16 prev_sample = 0;
int16 current_sample = 0;
int32 frac_new_sample = 0;
int32 frac_current_sample = 0;
uint32 sum_samples;
/* User includes (#include below this line is not maintained by Processor Expert) */

/*
** ===================================================================
**     Event       :  Cpu_OnNMIINT (module Events)
**
**     Component   :  Cpu [MKL25Z128LK4]
*/
/*!
**     @brief
**         This event is called when the Non maskable interrupt had
**         occurred. This event is automatically enabled when the [NMI
**         interrupt] property is set to 'Enabled'.
*/
/* ===================================================================*/
void Cpu_OnNMIINT(void)
{
  /* Write your code here ... */
}


/*
** ===================================================================
**     Event       :  AD1_OnMeasurementComplete (module Events)
**
**     Component   :  AD1 [ADC_LDD]
*/
/*!
**     @brief
**         Called after measurement is done, [Interrupt service/event]
**         is enabled, OnMeasurementComplete event is enabled and ADC
**         device is enabled. See [SetEventMask()] method or [Event
**         mask] property group to enable this event and [Enable]
**         method or [Enabled in init. code] property to enable ADC
**         device. If DMA is enabled , this event is called after the
**         configured number of measurements and DMA transfer is done.
```

```
**     @param
**         UserDataPtr     - Pointer to the user or
**                         RTOS specific data. The pointer is passed
**                         as the parameter of measurement_task method.
*/
/* ===================================================================*/
void AD1_OnMeasurementComplete(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
              static uint8 flag = 0;

              if(!flag)
              {
                        voltage[k2] = ADC0_RA;
                        fr_mr_voltage = voltage[k2++];
                        if (k2 == 64 )
                         {
                                  cycle_number++;
                                  _lwevent_set(&buffer_ready,FIRST_HALF);
                                  init_direction = 0;
                         }
                        if(k2 == 128)
                         {
                                  cycle_number++;
                                 _lwevent_set(&buffer_ready,SECOND_HALF);
                                 init_direction = 64;
                                  k2 = 0;
                         }
                        if(k3 == 128)
                                  k3 = 0;
              }
              else
              {
                        current[k1++] = ADC0_RA;
                        fr_mr_current = current[k4++]; //k4
                         if (k1 >= 128)
                                  k1 = 0;
                         if(k4==128)
                                  k4 = 0;
              }
              flag = !flag;

              switch(cycle_number)
                        {
                        case 1:
                                  _lwevent_set(&buffer_ready,0x04);  //Activa funcion de escalamientos
                                  break;
                        case 2:
                                  _lwevent_set(&buffer_ready,0x10);           //Activa Calculo de Frecuencia
                                   break;
                        case 3:
                                  _lwevent_set(&buffer_ready,0x100);  //Activa Calculo de Frecuencia
                                  break;
                        default:
                                  break;
                        };
              if ( cycle_number >= 59)
                        cycle_number = 0;
               //FMSTR1_Recorder();
}



/*
** ===================================================================
**     Event       :  AS1_OnBlockReceived (module Events)
**
**     Component   :  AS1 [Serial_LDD]
*/
/*!
```

```
**    @brief
**        This event is called when the requested number of data is
**        moved to the input buffer.
**    @param
**        UserDataPtr     - Pointer to the user or
**                        RTOS specific data. This pointer is passed
**                        as the parameter of measurement_task method.
*/
/* ===================================================================*/
void AS1_OnBlockReceived(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
              _lwevent_set(&serie_event,0x01);
                          switch(data_received[8])
                          {
                          case 0x01:
                                      _task_ready(_task_get_td(task5_id));  //Activa la tarea de escribir en la pantalla
                                      break;
                          case 0x02:                          //Activa la tarea de graficar señales
                                      _task_ready(_task_get_td(task4_id));
                                      break;
                          case 0x03:                            //Activa la tarea de la SD Card
                                      _task_ready(_task_get_td(task6_id));
                                      break;
                          case 0x04:                    //Activa la tarea de facturacion
                                      _task_ready(_task_get_td(task13_id));
                                      break;
                          };
}

/*
** ===================================================================
**    Event      :  AS1_OnBlockSent (module Events)
**
**    Component  :  AS1 [Serial_LDD]
*/
/*!
**    @brief
**        This event is called after the last character from the
**        output buffer is moved to the transmitter.
**    @param
**        UserDataPtr     - Pointer to the user or
**                        RTOS specific data. This pointer is passed
**                        as the parameter of measurement_task method.
*/
/* ===================================================================*/
void AS1_OnBlockSent(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
              Green_NegVal(Green_DeviceData);
}

/*
** ===================================================================
**    Event      :  SM1_OnBlockSent (module Events)
**
**    Component  :  SM1 [SPIMaster_LDD]
*/
/*!
**    @brief
**        This event is called after the last character from the
**        output buffer is moved to the transmitter. This event is
**        available only if the SendBlock method is enabled.
**    @param
**        UserDataPtr     - Pointer to the user or
**                        RTOS specific data. The pointer is passed
**                        as the parameter of measurement_task method.
*/
/* ===================================================================*/
void SM1_OnBlockSent(LDD_TUserData *UserDataPtr)
```

```
{
  /* Write your code here ... */
}

/*
** ===================================================================
**     Event      :  SM1_OnBlockReceived (module Events)
**
**     Component  :  SM1 [SPIMaster_LDD]
*/
/*!
**     @brief
**         This event is called when the requested number of data is
**         moved to the input buffer. This method is available only if
**         the ReceiveBlock method is enabled.
**     @param
**         UserDataPtr     - Pointer to the user or
**                           RTOS specific data. The pointer is passed
**                           as the parameter of measurement_task method.
*/
/* ===================================================================*/
void SM1_OnBlockReceived(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
             SD1_DataReceivedFlag = TRUE;
}

/*
** ===================================================================
**     Event      :  TI1_OnInterrupt (module Events)
**
**     Component  :  TI1 [TimerInt_LDD]
*/
/*!
**     @brief
**         Called if periodic event occur. Component and OnInterrupt
**         event must be enabled. See [SetEventMask] and [GetEventMask]
**         methods. This event is available only if a [Interrupt
**         service/event] is enabled.
**     @param
**         UserDataPtr     - Pointer to the user or
**                           RTOS specific data. The pointer passed as
**                           the parameter of measurement_task method.
*/
/* ===================================================================*/
void TI1_OnInterrupt(LDD_TUserData *UserDataPtr)
{
  /* Write your code here ... */
             TmDt1_AddTick();
             if(date.Day != date2.Day)
                          {
                          date.Day   = date2.Day;
                          date.Month = date2.Month;
                          date.Year  = date2.Year;
                          days++;
                          _lwevent_set(&serie_event,0x02);  //un dia ha transcurrido
                          }
             /*if(time.Hour != time2.Hour)
                                  {
                             time.Hour = time2.Hour;
                                  time.Min = time2.Min;
                                  time.Sec = time2.Sec;
                                  _lwevent_set(&serie_event,0x02);  //una hora ha transcurrido
                                  }*/
             if(time.Min != time2.Min)
             {
                          time.Hour = time2.Hour;
                          time.Min = time2.Min;
                          time.Sec = time2.Sec;
                          _lwevent_set(&serie_event,0x04);  //una hora ha transcurrido
```

```
            }
}

/*
** ===================================================================
**     Event      :  IFsh1_OnWriteEnd (module Events)
**
**     Component  :  IFsh1 [IntFLASH]
*/
/*!
**     @brief
**         Event is called after a write operation to FLASH memory is
**         finished (except [SetPage]). This event is available only if
**         an [Interrupt service/event] is selected.
*/
/* ===================================================================*/
void IFsh1_OnWriteEnd(void)
{
  /* Write your code here ... */
}

/* END Events */

#ifdef __cplusplus
}  /* extern "C" */
#endif

/*!
** @}
*/
/*
** ###################################################################
**
**     This file was created by Processor Expert 10.3 [05.08]
**     for the Freescale Kinetis series of microcontrollers.
**
** ###################################################################
*/
```