UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Diseño de un sistema embebido para el desarrollo de un watthorímetro con base en un microcontrolador PIC

TESIS

Que para obtener el título de

Ingeniero Eléctrico Electrónico

P R E S E N T A

José Luis Nieto Juárez

DIRECTOR DE TESIS

M.I. José Castillo Hernández

Ciudad Universitaria, Cd. Mx., 2022

Agradecimientos.

A mis padres, por todo.
A mi esposa, por todo su apoyo y ayuda.
Al Profesor José Castillo, por su sabiduría, guía, ayuda y tiempo.
A la UNAM, por todo lo ofrecido.
Al Padre, por dejarme estar aquí.

Agradecemos a la DGAPA por su apoyo para el desarrollo de esta tesis a través del proyecto PAPIME PE104819 «Videos para reforzar el desarrollo práctico de la enseñanza del diseño electrónico dirigido a los alumnos de Ingeniería Eléctrica Electrónica».

Contenido

Introducción.

A lo largo de la historia se han construido diferentes máquinas que han hecho más agradable y fácil nuestra existencia. Para el hombre del siglo XXI es habitual interactuar con dispositivos electrónicos que abstraen información, la procesan y devuelven un resultado. Básicamente todos los aspectos de la vida moderna tienen algún dispositivo electrónico que ayuda en alguna tarea, y que en su interior encierran una relación *hardware-software*. Existen diversas definiciones de un sistema embebido y estas han ido cambiando a lo largo del tiempo. Para este trabajo se propone definir a los sistemas embebidos como al conjunto de componentes físicos que, junto con una parte de software, desempeña una funcionalidad programada, responde a estímulos que recibe de su entorno (Murti, 2022) y frecuentemente son componentes de un sistema más grande (Barr, 1999).

Actualmente los watthorímetro se complementan trabajando junto con otros sistemas como relojes en tiempo real, medios de almacenamiento, transmisores de datos, etc., este conjunto produce los llamados sistemas inteligentes de medición de energía, los cuales no serían una realidad sin los sistemas embebidos.

En este trabajo se realiza un diseño de un watthorímetro apoyado en un sistema embebido.

Justificación.

En el laboratorio de electrónica del ICAT, se ha tenido un constante acercamiento en la medición de consumo energético. Como ejemplo, se puede citar el desarrollo de un prototipo probado en un vehículo eléctrico, que permitió medir su desempeño energético en la competencia *SHELL ECO-MARATHON AMERICAS* 2019, llevado a cabo en Sonoma, California (Castillo Hernández, et al., 2019). Derivado de esta experiencia se consideró extender este desarrollo para medir el consumo en fuentes de corriente alterna; más aún, considerando que una de las líneas de trabajo del laboratorio en donde se gestó este prototipo, es la electrónica de potencia, nuestro interés se centra en desarrollar un sistema de uso general que pueda ser integrado en su momento en otros proyectos, en donde sea posible modificar su estructura y programación en función de las necesidades que se requieran cumplir.

Objetivo general.

Diseñar e implementar un watthorímetro apoyado en el enfoque de sistemas embebidos.

Objetivos particulares.

1. Evaluar el desempeño de un microcontrolador *PIC* como procesador central de un sistema embebido aplicado a la medición de energía eléctrica.
2. Validar, implementar y evaluar dos algoritmos de integración numérica para la medición de energía eléctrica.

1. Antecedentes.

A continuación, se definen un conjunto de parámetros que serán usados a lo largo del presente trabajo. Es importante señalar que la información que se ofrece es puntual; sin embargo, un estudio más profundo se puede consultar en (W. Hart, 2001) (A. Svoboda & C. Dorf, 2015).

1.1. Potencia Instantánea.

La potencia instantánea *p(t)* es el valor de potencia en cada instante de tiempo *t*. Se define en la ecuación (1).

$$p(t) = v(t)i(t) \tag{1}$$

Su cálculo es a partir del producto de tensión *v(t)* y la corriente *i(t)*, que experimenta una red o carga, en un instante de tiempo *t*. Se expresa en *watts.* La definición es válida para cualquier dispositivo o circuito.

1.2. Energía.

La energía *W*, es la integral de la potencia instantánea medida en un periodo de tiempo *T*. Esto se muestra en la ecuación (2)

$$W = \int_0^T p(t)dt \tag{2}$$

1.3. Potencia Media o Promedio.

La potencia media o potencia promedio *P*, es la energía que se divide entre su mismo periodo sobre el cual se evalúa. La potencia promedio se puede expresar como se indica en la ecuación (3), donde *T* es el periodo.

$$P = \frac{1}{T}\int_0^T p(t)\,dt = \frac{W}{T} \tag{3}$$

En esta ecuación, cuando *P* es mayor que cero, la red o carga está generando potencia, en tanto que, si *P* es menor que cero, la red consume potencia. Por último, cuando *P* es cero, entonces la red o carga no consume ni genera potencia. Cabe mencionar que, la potencia media o promedio, también se le llega a conocer como potencia activa en circuitos de corriente alterna (CA).

1.4. Valor eficaz.

El valor eficaz o *RMS* del voltaje de una fuente que varía en el tiempo y está conectada a una carga, equivale al voltaje de una fuente constante capaz de suministrar la misma potencia promedio a dicha carga. Para el caso de la corriente se tiene una definición similar. Esto facilita los cálculos en el análisis de circuitos eléctricos, porque permite sustituir una fuente que depende del tiempo por otra que se mantiene constante. El valor eficaz del voltaje y la corriente se puede calcular a partir de sus valores instantáneos, como se indica en las ecuaciones (4) y (5).

$$V_{RMS} = \sqrt{\frac{1}{T}\int_0^T v^2(t)dt} \qquad (4)$$

$$I_{RMS} = \sqrt{\frac{1}{T}\int_0^T i^2(t)dt} \qquad (5)$$

1.5. ¿Qué es un Watthorímetro?

El watthorímetro se puede definir como un medidor de energía eléctrica que calcula la integral con respecto al tiempo de la potencia instantánea del circuito en el cual está conectado. Ésta es la energía eléctrica consumida por el circuito durante el intervalo en el que se efectúa la integración.

1.6. Del watthorímetro electromecánico a los medidores inteligentes.

Teniendo claro la definición del instrumento, se procede a dar antecedentes sobre su invención e historia. Todo empieza pocos años después del descubrimiento del campo magnético rotatorio por Galileo Ferraris en 1885. Con este descubrimiento, tiempo después aparece el primer medidor de corriente alterna, el cual fue hecho en Francia por Borel y Paccaud. A partir de esto, varios watthorímetro de corriente alterna empezaron a surgir. Uno de los primeros que llego a producirse en masa es el de Oliver B. Shallenberger en 1888 de la compañía *Westinghouse* (Lanphier, 1925).

Los medidores electromecánicos predominaron hasta antes de 1970. Deficiencias como la comunicación y medición de otras variables fue la motivación para la creación de un nuevo tipo de medidor. Así surge el medidor electrónico de estado sólido, con el que fue posible aumentar las prestaciones del medidor de energía eléctrica. Entre 1970 y 2000 se agregó al medidor electrónico el AMR (*Automatic Meter Reading*), que permitió transmitir de forma remota las lecturas para su uso en la compañía eléctrica, la limitante de estos medidores estriba en la comunicación de una sola vía, esto significa que sólo los medidores podían enviar información, pero no recibirla. Durante la primera década del 2000, llegan los medidores inteligentes, los cuales se han usado durante más de una década, incorporan el AMR, y numerosas funciones de medición, registro y comunicación. Estos medidores aumentan las posibilidades, su comunicación ha sido mejorada, ahora a dos vías, lo que significa que ahora pueden enviar datos y recibir instrucciones. Esta característica hace la diferencia entre un medidor electrónico y uno inteligente (Weranga, et al., 2014).

Es interesante resaltar la fecha de 1970 por su cercanía con la aparición del primer microprocesador Intel 4004 en 1971 (Barr, 1999), dando así inicio a los sistemas embebidos. El microprocesador fue desarrollado originalmente para reemplazar un bloque de lógica digital utilizado para crear las primeras calculadoras electrónicas a principios de la década de 1970 (Heath, 2003). La principal distinción entre un microprocesador y un microcontrolador consiste en que un microprocesador

requiere otros componentes para su funcionamiento, como memoria de programa, memoria de datos, etc., mientras que un microcontrolador tiene todos los circuitos de soporte incorporados en un mismo encapsulado (Ibrahim, 2008). Dentro del mercado de microcontroladores encontramos los *PIC* fabricados por la empresa *Microchip Technology*. La creación de éstos es producto de la necesidad de mejorar el rendimiento con periféricos de entrada y salida del procesador CP1600 del fabricante *General Instruments*, este dispositivo fue llamado *PIC* por «*Programmable Intelligent Computer*», actualmente se reconoce como «*Programmable Interface Controller*» (Sanchez & P. Canton, 2007).

Las ventajas de los medidores inteligentes, con respecto a los medidores electromecánicos son varias como se muestra en la tabla 1, desde almacenar datos, detección de interrupción de energía hasta la comunicación por dos vías. Aspectos como la corta expectativa de vida en el medidor inteligente, muestra detalles que en un futuro deberán mejorar para obtener su mayor eficiencia.

Tabla 1: Comparativa medidor electromecánico vs Inteligente.

Medidor Electromecánico	Medidor Inteligente
No almacena datos. [1]	Almacena datos, almacena el consumo cada media hora. [1]
Personal debe ir a recolectar la información y enviarla a la compañía eléctrica. [1]	Los datos se transmiten automáticamente a la compañía eléctrica. [1]
Sin acceso al medidor, la cuenta puede resultar en un consumo estimado. [1]	Los datos de consumo de energía y el TOU (Time Of Use) son proporcionados casi en tiempo real. [1]
Sin detección automática de interrupción, la compañía de distribución no puede actuar rápidamente para reestablecer el suministro. [1]	Detección automática de interrupción habilita a la compañía de distribución reestablecer el servicio eléctrico más rápido. [1]
La conexión y desconexión debe ser hecha manualmente. [1]	La conexión y desconexión es rápida y se maneja remotamente. [1]
Expectativa de vida larga (20,30 o 40 Años). [1]	Expectativa de vida corta (5-7 Años). [1]
Reacciona al cambio más despacio. [2]	Reacciona al cambio con rapidez. [2]
Diversas formas de robo de energía como bypass y reconexión ilegal de la línea. [2]	No hay formas conocidas de robo de energía. [2]

[1] (Bimenyimana & Osarumwense, 2018)

[2] (Weranga, et al., 2014)

1.7. El watthorímetro ideal.

El watthorímetro ideal, como se muestra en la figura 1, encierra el cálculo de la integral de la potencia instantánea durante un intervalo de tiempo. En este modelo, la tensión, la corriente y la energía consumida (*EC*), son funciones continuas. Llegamos a una propiedad de este modelo ideal, la cual es, la continuidad en el tiempo.

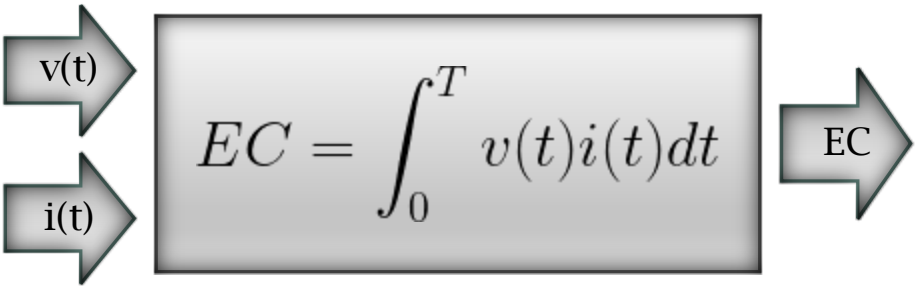


Figura 1: El watthorímetro ideal.

Esta propiedad, simboliza un enorme conjunto de datos, que pueden ser manejados de manera teórica y con dispositivos electrónicos analógicos. La historia es diferente si volteamos a ver soluciones basadas en la discretización. Son los recursos limitados, con los que generalmente cuenta un sistema digital, la razón por la que almacenar grandes volúmenes de muestras para la integración, es algo difícil y a veces imposible.

2. El algoritmo.

Para una implementación con electrónica digital, es natural pensar en la discretización como herramienta para obtener la información de las señales de tensión y corriente; pero al hacer esto, no es posible aplicar el modelo ideal directamente, ya que pierde el atributo de existir en cada instante de tiempo.

2.1. Método del rectángulo.

En el watthorímetro ideal, su función es calcular la integral del producto de las funciones continuas de tensión y corriente. Ahora, el problema es plasmar la ecuación (2), considerando valores tabulares. Una aproximación a la ecuación tendría que ver con la sumatoria, porque podemos trabajar con valores tabulados. Se muestra en la ecuación (6), el cálculo de la integral para un numero *N* de datos (Crenshaw, 2000).

$$\int_a^b f(x)dx \cong \frac{1}{N}\sum_{i=o}^{N} f_i \qquad (6)$$

Si se considera que las funciones continuas de tensión y corriente serán convertidas a un conjunto discreto de puntos, se concluye que, a este conjunto, se le puede aplicar la ecuación (6). Esta ecuación que representa una integral también se le conoce como método del rectángulo, este método consiste en dividir el área bajo la curva en N rectángulos de base 1/N, calculando la integral como la suma de las áreas de los rectángulos en los que se dividió.

Una opción para hacer viable la implementación en un sistema digital de la ecuación (7) es tener varios subintervalos de la energía consumida (*ECSUB*), los cuales se van acumulando (*ECA*), como se muestra en la ecuación (8), hasta completar el intervalo de medición deseado. De esta forma, se fragmenta una tarea en varias, evitando saturar la memoria y dando posibilidad a obtener todos los datos sin interrupciones.

$$ECSUB = \frac{1}{N}\sum_{n=0}^{N-1} v(n)i(n) \quad (7)$$

$$ECA_k = ECA_{k-1} + ECSUB \quad (8)$$

Donde *k* es un número entero mayor a cero, el cual representa el número de subintervalo, *N* es el número de muestras por subintervalo y ECA_0 es igual a cero.

Es posible combinar las ecuaciones (7) y (8) para generar el algoritmo que hará posible cuantificar la energía consumida durante un periodo de tiempo. En la figura 2, se muestra el diagrama de flujo que explica el algoritmo, donde *m* es el número de intervalos a sumar antes de mostrar el resultado final, y cuando *k* es igual a *m*, el valor final ECA_k se desplegará y se inicia de nuevo. La figura 3, muestra un ejemplo de 5 subintervalos, usando la ecuación (7) para procesar un bloque de *N* datos y obtener un acumulado usando la ecuación (8), donde el valor final es ECA_5.

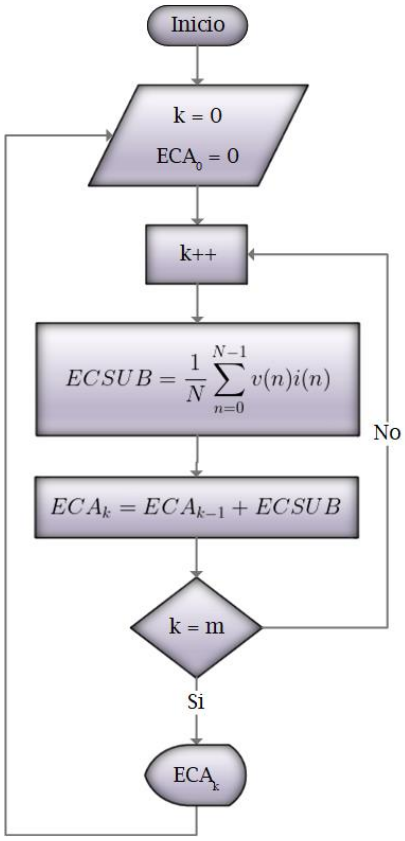


Figura 2: Algoritmo propuesto usando ecuación (7) y (8).

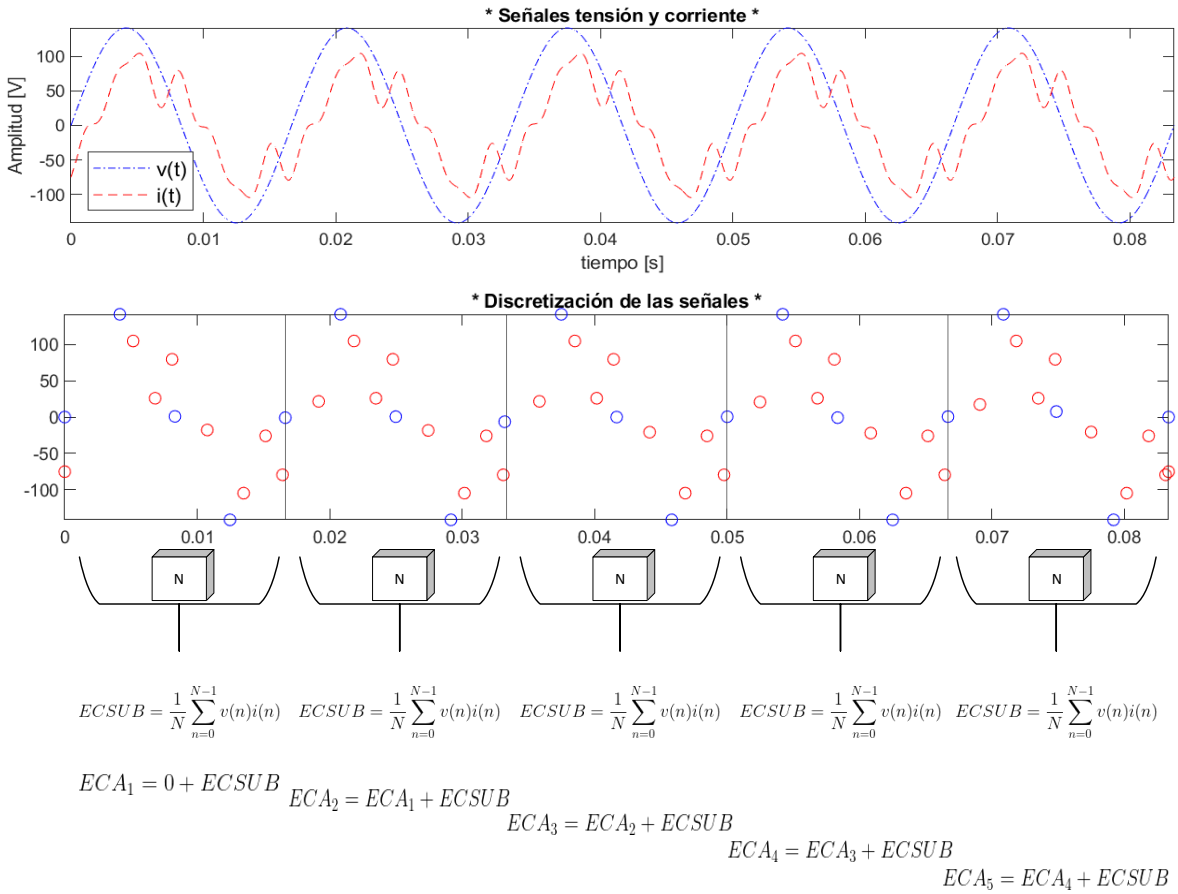


Figura 3: Ejemplificación del algoritmo con 5 subintervalos.

2.2. Regla de Simpson.

Además del método del rectángulo se propone el uso de la regla de Simpson para la integración numérica, ya que este método posee un buen balance entre exactitud y simplicidad (Crenshaw, 2000). En la ecuación (9), se define la regla de Simpson.

$$\int_a^b f(x)dx \cong \frac{1}{3 \cdot N} \sum_{i=0,2,4,6,\ldots}^{N-2} f_i + 4 \cdot f_{i+1} + f_{i+2} \qquad (9)$$

Donde *N* es el número total de datos para la integración. Este número debe ser impar, igual que el índice *i*.

La ecuación (9) puede ser reescrita para el cálculo de la potencia promedio, y así, sustituir la ecuación (7). La ecuación (10) muestra este cambio.

$$ECSUB = \frac{1}{3 \cdot N} \sum_{n=0,2,4,6,\ldots}^{N-2} v(n)i(n) + 4 \cdot v(n+1)i(n+1) + v(n+2)i(n+2) \quad (10)$$

Es de notar que implementar este método usando la ecuación (10), no supone proezas computacionales u operaciones complicadas, pero con este pequeño cambio existe una mejora en el error de cálculo, estos resultados se muestran en la siguiente sección.

2.3. Validación del algoritmo para cálculo de potencia eléctrica.

La validación del algoritmo es un proceso de verificación en el cual se asegura que los datos entregados como resultado son correctos. Esta validación consta en calcular la potencia promedio, tomando como referencia dos señales de prueba (una de voltaje y otra de corriente) y evaluar el error en los algoritmos de integración numérica.

2.3.1. Señales de prueba.

Se necesita definir dos señales de prueba, una de voltaje y otra de corriente, cuyos valores sean conocidos. Con estas señales se obtienen los valores ideales, que posteriormente serán comparados con los resultados que arroja el algoritmo de integración numérica. Para tal propósito, se utilizó como referencia el caso 4 descrito en «*A Controversial Issue: Power Components in Nonsinusoidal Single-Phase Systems* (Yumak & Usta, 2011). El caso 4, consta de dos señales, una de voltaje y otra de corriente no sinusoidales, las cuales están descritas por las ecuaciones (11) y (12):

$$v(t) = \sqrt{2}[100\sin(\omega_1 t) + 20\sin(\omega_3 t - 70°) + 25\sin(\omega_5 t + 140°) + 10\sin(\omega_7 t + 210°)] \quad (11)$$

$$i(t) = \sqrt{2}[60\sin(\omega_1 t - 30°) + 15\sin(\omega_3 t - 165°) + 12\sin(\omega_5 t + 285°) + 10\sin(\omega_7 t + 310°)] \quad (12)$$

La figura 4 muestra las formas de onda de estas ecuaciones.

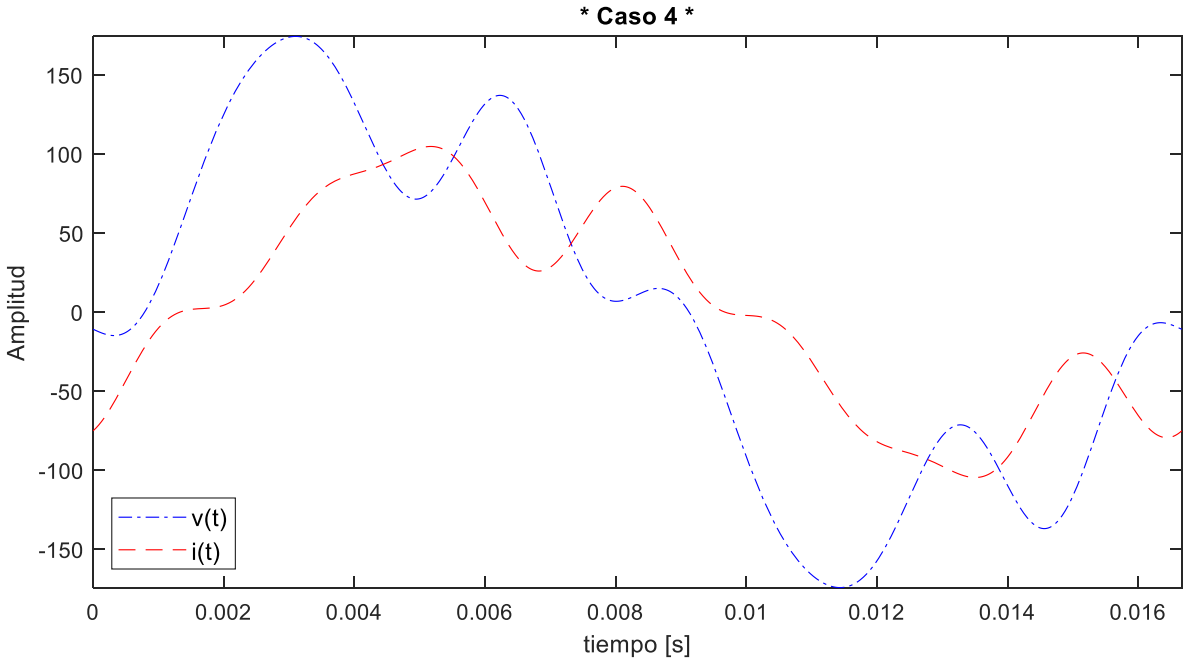


Figura 4: Formas de onda del caso 4.

2.3.2. Valor ideal y porcentaje de error.

Con las ecuaciones (11) y (12) que describen las señales de prueba, se obtiene la potencia promedio P, que es igual a 4906.9 watt. La cifra anterior será tomada como valor ideal para su posterior comparación con los resultados obtenidos por los métodos de integración descritos en las ecuaciones (7) y (10).

El porcentaje de error, definido en la ecuación (13), será la herramienta con la que se cuantificará el error producido por los dos métodos de integración que este trabajo evalúa.

$$\%\ Error = \left|\frac{valor\ obtenido - valor\ ideal}{valor\ ideal}\right| \cdot 100 \quad (13)$$

2.3.3. Programa de validación.

Todos los cálculos fueron hechos con ayuda del programa de cómputo *Matlab*, usando *Symbolic Math Toolbox*. El código fuente se incluye en el apéndice B de este trabajo.

2.3.4. Observaciones.

Es importante mencionar que esta comparativa tiene como objetivo la cuantificación del error, no se está contemplando los errores intrínsecos a un sistema físico como son: linealidad del sensor, ruido, error de redondeo, errores en la etapa de acoplamiento de la señal, etc. Dicho lo anterior, los resultados mostrados deben ser tomados como ideales bajo las condiciones descritas anteriormente.

2.3.5. Resultados.

Para presentar de una forma más clara los resultados de esta validación, se generaron dos gráficas, mostradas en la figura 5, que presentan el error al usar regla de Simpson y método del rectángulo como procedimiento de integración numérica. Es de resaltar que, mientras el método del rectángulo con 31 muestras procesadas el error es 2.60 %, usando regla de Simpson el error es de 0.31 %, verificando la efectividad de este último método de integración, así como una mejor convergencia al valor real con un menor número de intervalos.

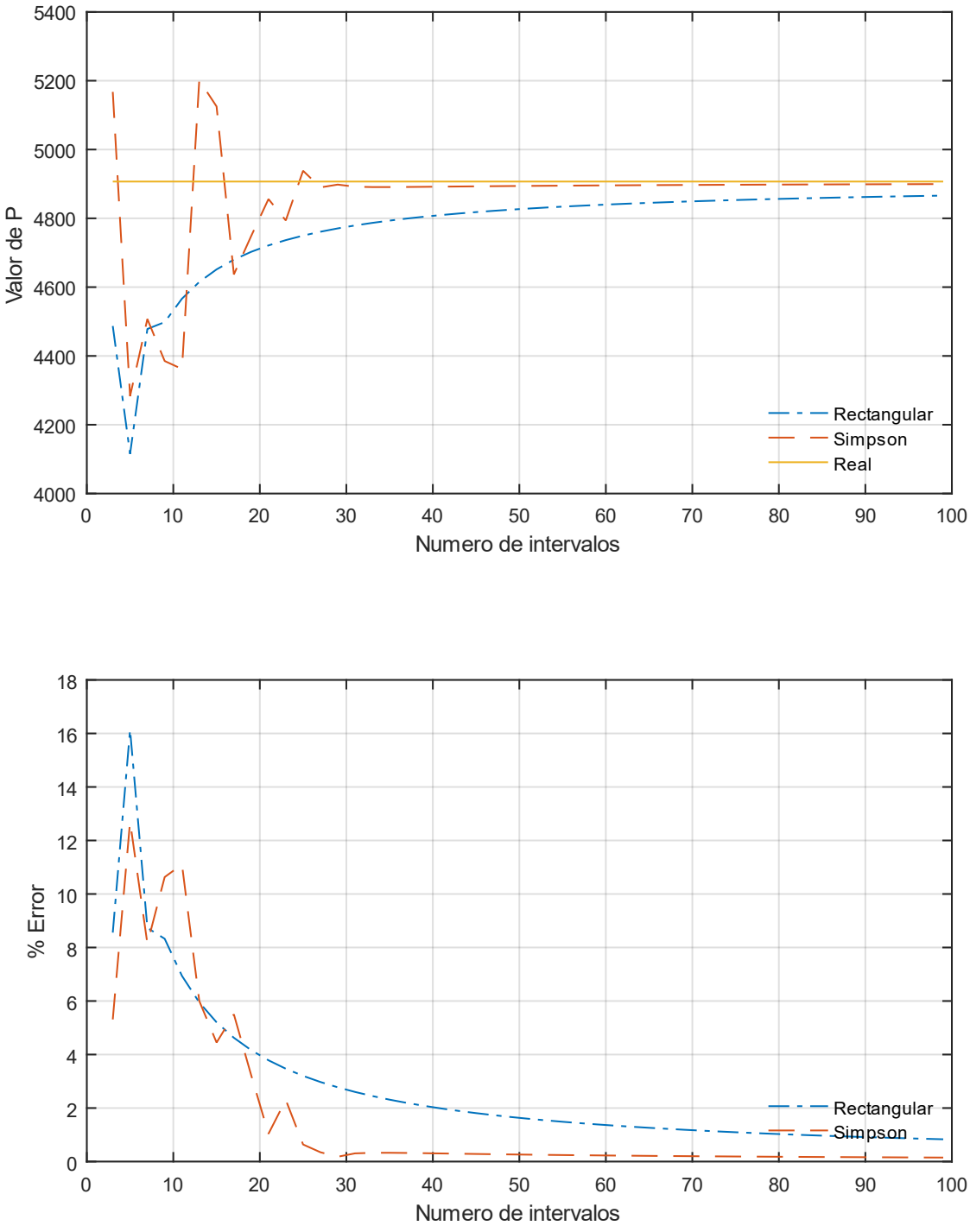


Figura 5: Graficas de error del caso 4.

3. Requisitos para el sistema embebido.

Antes de proceder con el diseño del sistema, es importante y necesario definir los requisitos a cumplir, los cuales fungirán como guía en nuestro diseño.

Los requerimientos para el watthorímetro consideran los siguientes puntos:

1. El sistema ejecutará ambos métodos de integración numérica propuestos en este trabajo.
2. Se plantea el diseño de un watthorímetro monofásico.
3. Diseño basado en un microcontrolador PIC.
4. Usar el módulo convertidor analógico digital (ADC) interno del microcontrolador para adquirir y procesar las señales, es recomendable tener una resolución de 12-bit.
5. Considerar un conector para la programación *in situ* del microcontrolador, apoyado en una terminal *In-Circuit Serial Programming* (*ICSP*).
6. El despliegue de datos será a través de un *display LCD* 16x2.
7. El *firmware* del microcontrolador se escribirá en lenguaje C, procurando agregar los comentarios pertinentes.
8. Usar una fuente de alimentación externa.
9. Registrar la señal de tensión y corriente considerando las características de la red doméstica, que corresponde a una señal senoidal en el caso de la tensión con un valor de tensión *RMS* de 127 V y una frecuencia de 60 Hz.
10. Considerar un muestreo continuo a partir de una base de tiempo.
11. Realizar la medición del voltaje utilizando la técnica de división de tensión, a través de una red de resistencias. Se debe agregar aislamiento eléctrico para protección del operador y del instrumento.
12. La medición de corriente se realiza con base en un sensor de efecto *Hall*. Se debe agregar aislamiento eléctrico para protección del operador y del instrumento.

4. Hardware.

El diseño iniciará por el *hardware*, ya que este impondrá en gran medida las limitaciones físicas, que posteriormente se considerarán en el diseño del *software*.

Antes de iniciar, es importante revisar el *hardware* de los medidores inteligentes para conocer de que módulos se componen. Los medidores inteligentes actuales constan de varios módulos y sistemas como son: Unidad de sensado de corriente y tensión, fuente de alimentación, unidad de medición de energía, Microcontrolador, Reloj en tiempo real y sistema de comunicación (Weranga, et al., 2014).

El sistema descrito en este trabajo es una propuesta para el desarrollo del núcleo de un sistema de medición inteligente. Por tanto, sólo se están considerando los siguientes módulos (Figura 6):

- Procesamiento de señal (microcontrolador uC).
- Medición y acondicionamiento de señal.
- Despliegue de información (Pantalla de cristal líquido *LCD*).
- Fuente de alimentación externa.

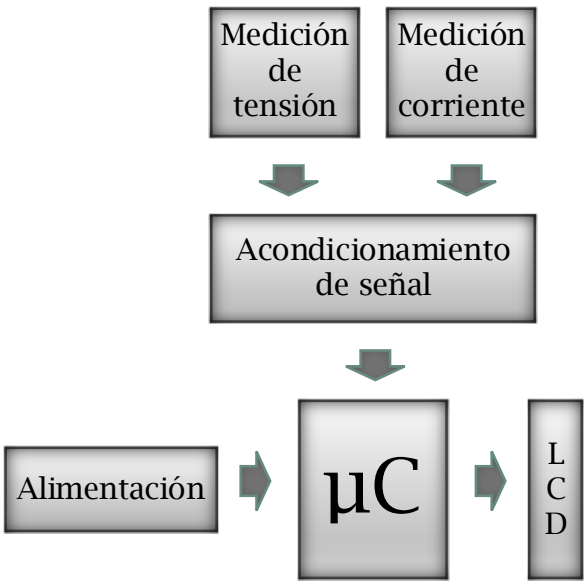


Figura 6: Partes del watthorímetro base para este trabajo.

4.1. Medición de la señal de tensión.

Para realizar la medición de la señal, se ocupó la técnica de divisor de tensión. Esta práctica tiene como finalidad atenuar un voltaje de entrada. El divisor se implementa con resistencias. El circuito del divisor se muestra en la figura 7. La ecuación (14), indica la relación que existe entre el voltaje de entrada y el voltaje de salida.

$$\frac{V_o}{V_i} = \frac{R_2}{R_1 + R_2} \qquad (14)$$

Donde, V_i es el voltaje de entrada, V_o es el voltaje de salida y las resistencias que conforman la red son R_1 y R_2. En la ecuación (14), se observa que el voltaje de salida es una fracción del voltaje de entrada.

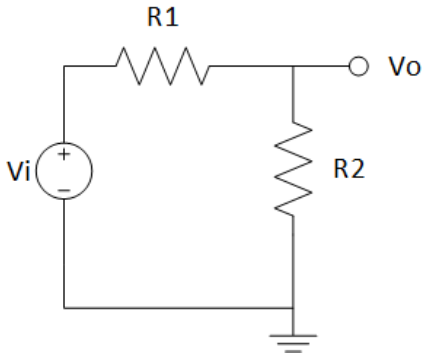


Figura 7: Divisor de tensión.

El voltaje V_o que se obtiene del divisor de tensión, debe recibir un acondicionamiento adicional, que garantice una señal apropiada, para que en un proceso posterior sea convertida en un valor digital.

4.1.1. Acondicionamiento de la señal de tensión.

Para procesar la señal de voltaje, es importante conocer las características de la señal y las limitaciones de la unidad que la procesa.

La señal de entrada al divisor tiene un voltaje pico teórico de 179.6 V, que equivale a un voltaje *RMS* de 127 V. Como unidad de procesamiento se usó un microcontrolador *PIC*, que incluye entre sus periféricos un convertidor analógico digital, el cual acepta señales entre 0 a 5 V. Con lo anterior, se observa que el divisor atenúa la señal de entrada para dejarla en el rango mencionado; esta señal, se le debe agregar una componente de DC para tener una polaridad positiva en todo momento, este valor será eliminado con ayuda del microcontrolador en un proceso posterior.

Considerando lo anterior y desarrollando los detalles, en la figura 8, se observa un bloque con ganancia *K,* conformado por la red divisora de tensión, que se encarga de atenuar la señal. El sumador, implementado con un amplificador no inversor sumador, agrega una componente de corriente directa a la señal atenuada.

En un paso posterior, es necesario aislar eléctricamente la señal de entrada del resto de la electrónica. Para tal propósito se usó un amplificador de aislamiento para establecer una barrera entre la señal de entrada y el resto del instrumento. Este amplificador, representado con el bloque de ganancia *K2*, está ubicado entre el divisor de tensión y el sumador.

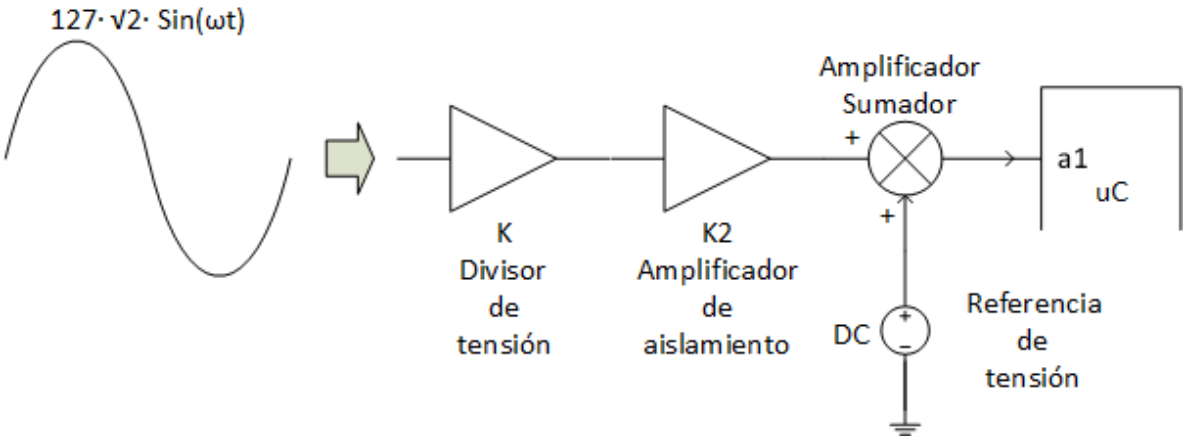


Figura 8: Acoplamiento de la señal.

Se utilizó el amplificador de aislamiento *HCPL-7840* del fabricante *Avago Technologies*. Originalmente, este circuito fue diseñado para la medición de

corriente en *drives* electrónicos para el control de motores, pero también se puede usar para el aislamiento general de señales analógicas, que requiera alta precisión y estabilidad bajo severas condiciones de ruido. Este amplificador viene en un encapsulado *DIP8* (Avago Technologies, 2012).

Se mencionan a continuación los parámetros más importantes del amplificador de aislamiento, que se tomaron en cuenta para realizar el acoplamiento:

1. Condición máxima recomendada para una tensión de entrada de ±200 mV.
2. Ganancia típica de 8.00 con tolerancia de ±5 % (valor de K2).
3. Condición recomendada para la tensión de alimentación: 4.5 a 5.5 V.
4. Requiere dos fuentes de alimentación.

El amplificador de aislamiento se polariza con dos fuentes. Para este propósito se usó un convertidor *DC/DC* aislado de 5 a 5 *volts* (U3).

El amplificador sumador no inversor, procede de uno de los cuatro disponibles en el circuito integrado *MCP6004* del fabricante *Microchip Technology Inc.* Este integrado acepta una sola fuente de 5 V para su sustento, evitando el uso de fuente simétrica y viene en un encapsulado *DIP14*. Se eligió este conjunto de amplificadores de propósito general por tener entrada y salida Fuente-a-Fuente (o *Rail-to-Rail*). Esta característica permite que la señal de entrada y salida puedan tomar valores muy cercanos a los de alimentación. En este amplificador, el intervalo de entrada en modo común es de aproximadamente 0.3 a 4.7 volts y la excursión de salida máxima va de 0.025 a 4.975 volts.

Se usó una referencia de voltaje para generar la componente de corriente directa de la señal de entrada. Esta referencia es generada por el *MCP1525* de *Microchip* que produce un valor de 2.5 V. Es importante resaltar que este circuito viene en un encapsulado TO-92, tiene una tolerancia máxima inicial de 1 % y se alimenta con 5 volts.

En la Figura 9, se presenta la referencia de tensión (U7) que a su salida usa un amplificador operacional en una configuración de seguidor de tensión (U5A), para evitar la caída de tensión y poder usar esta señal en distintos puntos del sistema.

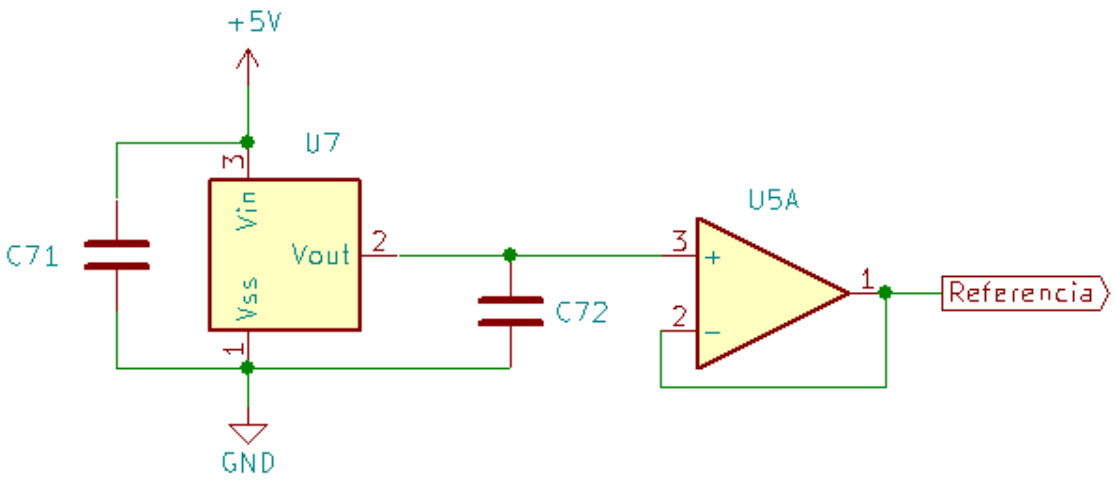


Figura 9: Referencia de tensión MCP1525 y amplificador buffer.

La Figura 10 muestra el arreglo final del acondicionamiento del voltaje, que considera el divisor de tensión formado por las resistencias R81 y R82, el amplificador de aislamiento U8 y el amplificador sumador no inversor U5B. El amplificador U5B suma la referencia de voltaje con la salida del amplificador de aislamiento.

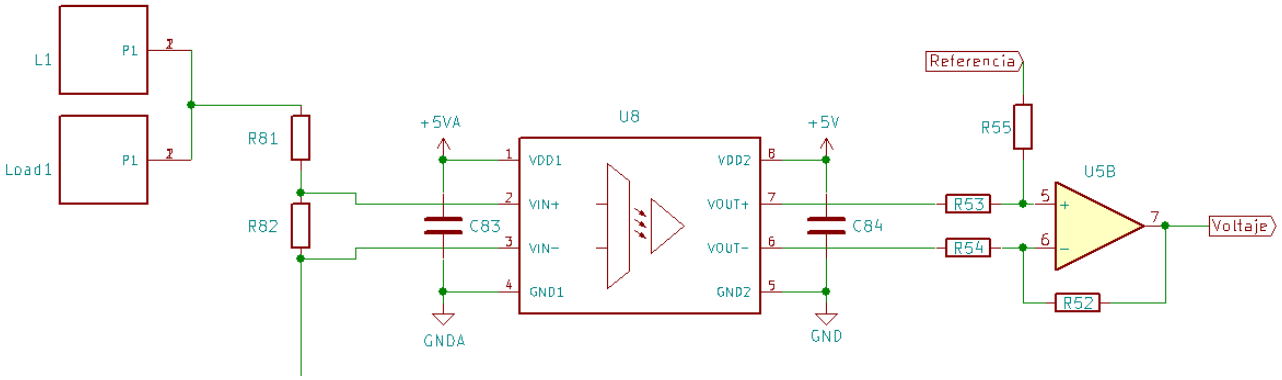


Figura 10: Acoplamiento de señal de tensión al puerto del microcontrolador.

La ganancia de cada uno de estos componentes interviene en la ganancia final de la señal de entrada. A continuación, se indica como se calculó cada uno de estos factores y se muestra su relación en el valor final.

4.1.2. Ganancia del divisor de tensión.

Considerando que el voltaje pico de entrada es de 179.6 V, y el voltaje de entrada al amplificador de aislamiento es de 100 mV, se necesita establecer una atenuación de 1/1796. Para esto, en la implementación se consideró una resistencia de 1 MΩ ±1 % y otra de 500 Ω ±1 % puestas en serie, generando de esta forma el divisor de voltaje. Esta combinación arroja una atenuación final de *1/2001*, que corresponde con el valor de la constante K en la señal de salida, ofreciendo un margen adicional de seguridad.

4.1.3. Ganancia del amplificador de aislamiento.

El amplificador de aislamiento U8 tiene una ganancia interna fija de 8.00 con una tolerancia de 5 %. (Avago Technologies, 2012)

4.1.4. Ecuación de acoplamiento de voltaje.

Juntando todos los parámetros anteriores, se puede definir la señal que se obtiene a partir del acondicionamiento con la ecuación (15):

$$V_{iadc} = \frac{8}{2001} V_i + 2.5 \qquad (15)$$

Donde V_i es la señal de voltaje a la entrada del watthorímetro y V_{iadc} es la señal de voltaje en la terminal del módulo *ADC* del microcontrolador.

4.2. Medición de la señal corriente.

Se conocen cuatro sensores de corriente que son usados ampliamente en medidores inteligentes (Weranga, et al., 2014), los cuales son sensores lineales de corriente basados en efecto Hall, transformadores de corriente, resistencia Shunt y

bobinas de Rogowski. A continuación, se describe brevemente el funcionamiento de estos cuatro y sus características, para comparar el funcionamiento y ventajas que cada uno posee.

El sensor lineal de corriente basado en efecto Hall, consiste en un empaquetado que contiene una pista conductiva, a la cual se le hace pasar una corriente, produciendo un campo magnético que es medido por un circuito Hall y convertido a un nivel de tensión proporcional, este sensor viene con circuitería integrada que permite la entrega de una señal ya acondicionada y amplificada.

Los transformadores de corriente producen una corriente secundaria que es proporcional a la corriente primaria, el primario es conectado en serie con la carga y son de alto costo.

La resistencia *Shunt*, es una resistencia de un valor muy pequeño, típicamente entre 100μΩ y 500mΩ. Se coloca en serie con la carga, y la señal de tensión que se obtiene en sus terminales es proporcional a la corriente que pasa a través de ella. El valor de la resistencia es muy estable y no cambia con la corriente, temperatura o envejecimiento, son de un costo moderado, pero son voluminosas.

La bobina de Rogowski, es una bobina con núcleo no magnético, da una salida de tensión que es proporcional a la razón de cambio de la corriente. Para obtener la señal original de corriente es necesario integrar la tensión de salida de la bobina con respecto al tiempo, estas bobinas tienen ventajas en comparación con los transformadores de corriente, como son la alta precisión en la medición, amplio rango de medición y frecuencia, pequeños en tamaño y peso. Si bien, la literatura menciona que son de bajo costo de producción, su costo comercial dista de esto. (Weranga, et al., 2014).

De los cuatro antes mencionados, en el mercado nacional el más accesible y fácil de encontrar al momento de realizar este trabajo fue el sensor de efecto Hall.

El componente seleccionado para este trabajo fue un sensor de la familia ACS756 de *Allegro Microsystems*, esta familia de sensores de corriente proporciona una solución para la medición de corriente tanto en AC o DC. La resistencia interna es

típicamente de 100μΩ, que permite operar al sensor por encima de cinco veces la condición de sobre corriente. Las terminales de la pista conductiva están eléctricamente aisladas de las terminales de señal, así que este sensor no requiere aislamiento eléctrico adicional. Este dispositivo se calibra en su fabricación, viene en encapsulado *CB5*. (Allegro MicroSystems. L.L.C., 2019)

Se utilizó el *ACS756SCB-050B-PFF-T* que permite una temperatura de operación de –20 a 85 °C, y una corriente máxima de muestreo de ±50 A. De los parámetros más importantes para trabajar con este sensor son las siguientes (Allegro MicroSystems. L.L.C., 2019):

- Tensión de alimentación típica de 5.0 V.
- Presenta una salida de 2.5 V bajo una operación de 0A.
- Corriente máxima de 50 A.
- Sensibilidad típica de 40 mV/A.
- Máximo error a la salida de ±5 % (de 25 a 85 °C).
- Aislamiento eléctrico incluido en el sensor.

La corriente máxima que llega a medir el sensor es de 50 A, considerando su sensibilidad de 40 mV/A. Con lo anterior, se deduce que el máximo valor de salida del sensor es 4.5 volts para +50 A, y el mínimo de 0.5 volts para -50 A. Se debe prestar atención en la señal que entrega el sensor, que ya va montada sobre un componente de corriente directa de 2.5 V.

Se muestra el sensor en la Figura 11, R1 y C2 constituyen un filtro paso bajas y C1 es un capacitor de bypass (Castillo Hernández, et al., 2019). La frecuencia de corte del filtro es de 3.3 kHz, fue suficiente para reducir el ruido que puede contaminar la línea. Por último, la señal de salida del sensor ya filtrada está conectada a uno de los puertos del convertidor analógico-digital del microcontrolador.

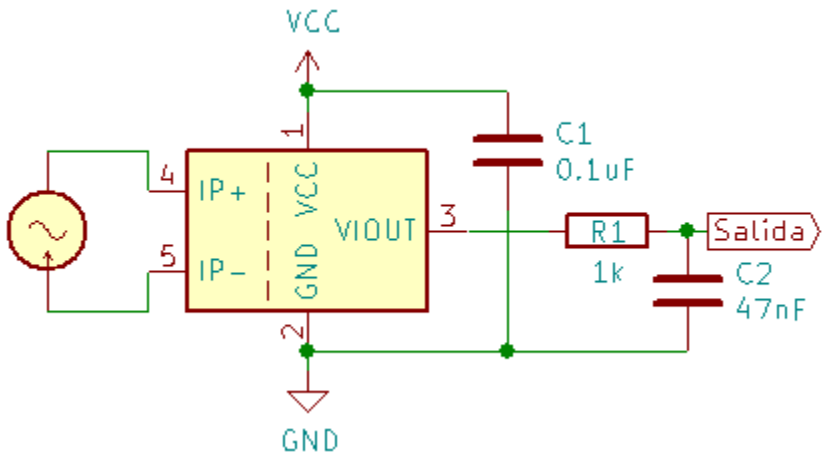


Figura 11: Sensor de Corriente.

Considerando lo anterior, se puede definir la señal que se obtiene a la salida del sensor de corriente con la ecuación (16):

$$V_{iadc} = \frac{1}{25} I_i + 2.5 \qquad (16)$$

4.3. Microcontrolador.

El microcontrolador es la piedra angular del sistema embebido, en él se fusiona el *hardware* y *software*.

Para empezar a definir las características del microcontrolador a utilizar en este trabajo, se puede empezar con los puertos *ADC* que son esenciales para la adquisición de señales, que para este trabajo se necesita como mínimo dos, uno para la señal tensión y el otro para la señal corriente. Además, se requiere una resolución de 12-Bits. También, es necesario incluir puertos digitales para manejar una pantalla de cristal líquido o *LCD*. Por último, es solicitado el uso de temporizadores para el control de tareas, y para este propósito el microcontrolador seleccionado debe tener al menos un temporizador con una resolución de 16-Bits para un mayor dominio.

En esta primera fase de diseño es deseable seleccionar un microcontrolador con la mayor cantidad de recursos disponibles, considerando que en esta etapa inicial los

algoritmos y procesos de organización de datos aun no toman su forma final. Esta fase requiere recursos de sobra para realizar pruebas y ensayar diferentes soluciones. Al final de este trabajo se conocerá si es posible ajustar el proyecto a un microcontrolador con medios austeros o si es necesario cambiar a uno con mayores prestaciones.

Para ilustrar lo anterior se ha realizado una estimación del tiempo que tomará el cálculo con los dos métodos de integración abordados en este trabajo. Se realizaron dos pequeños programas de prueba que se muestran en el apéndice E. Con dichos programas se procura saber el número de ciclos de instrucción aproximados, que utilizará para finalizar el cálculo considerando 45 intervalos en la integración. Se destaca que en esta estimación solo se considera el cálculo de la integral de una manera mínima, no se está incluyendo operaciones adicionales como multiplicación con números de punto flotante, conversión de tipos, entre otras, que necesitará el algoritmo para generar el resultado final. Se muestran los resultados en la tabla 2.

Tabla 2: Tiempos de ejecución.

Velocidad máxima en CPU (MHz)	Método del Rectángulo		Regla de Simpson	
	Tiempo aproximado en completar el calculo (s)	Interrupciones durante el cálculo.	Tiempo aproximado en completar el cálculo. (s)	Interrupciones durante el cálculo.
64	208.3E-6	0	353.7E-6	0
48	277.8E-6	0	471.6E-6	1
40	333.3E-6	0	565.9E-6	1
32	416.6E-6	1	707.4E-6	1
20	666.6E-6	1	1.1E-3	3

Se puede seleccionar cualquier microcontrolador con las velocidades de CPU mostradas en la tabla anterior. Con cada operación que se agrega se estará consumiendo más ciclos de instrucción que al final repercutirá en memoria necesaria para guardar datos que aún no se han procesado. En esta estimación, el reloj máximo de 64 MHz estará proporcionando el tiempo de ejecución más bajo, donde el cálculo se hará sin interrupciones y no sea necesario memoria adicional. Conforme el diseño vaya materializándose, estos lapsos de ejecución indudablemente aumentarán, se recurrirá a los recursos adicionales con los que

cuenta el microcontrolador. Aunque este aproximado indique que no es necesario memoria intermedia, la implementación de tareas como la de imprimir los resultados requerirá de alguna estructura de memoria que pueda gestionar información entrante que aún no pasa por el algoritmo de cálculo. Elegir un microcontrolador con grandes prestaciones hace de éste un núcleo versátil, necesario para esta primera etapa de diseño.

Para poder tener un listado de microcontroladores *PIC* con las características antes mencionadas, se utilizó la herramienta web *Parametric Search Tool* de *Microchip*. A partir de ésta se desarrolló la Tabla 3*,* que concentra los dispositivos disponibles con las características antes mencionadas y con la velocidad máxima de *CPU* disponible en este momento. Se descartan las familias *PIC24* y *dsPIC,* ya que la versión del compilador *CCS* utilizada en este trabajo no los soporta.

Tabla 3: Selección del microcontrolador.

Product	Status	Program Memory Size (KB)	SRAM (Bytes)	Pin count	Max CPU Speed (MHz)	ADC Input	Max ADC Resolution (Bits)	Max 8-Bit Digital Timers	Max 16-Bit Digital Timers	Packages
PIC18F25Q43	In Production	32	2048	28	64	24	12	3	4	QFN, SOIC, SPDIP, SSOP, VQFN
PIC18F26Q43	In Production	64	4096	28	64	24	12	3	4	QFN, SOIC, SPDIP, SSOP, VQFN
PIC18F27Q43	In Production	128	8192	28	64	24	12	3	4	QFN, SOIC, SPDIP, SSOP, VQFN
PIC18F24K42	In Production	16	1024	28	64	24	12	3	4	QFN, SOIC, SPDIP, SSOP, UQFN, VQFN
PIC18F25K42	In Production	32	2048	28	64	24	12	3	4	QFN, SOIC, SPDIP, SSOP, UQFN
PIC18F26K42	In Production	64	4096	28	64	24	12	3	4	QFN, SOIC, SPDIP, SSOP, UQFN
PIC18F27K42	In Production	128	8192	28	64	24	12	3	4	QFN, SOIC, SPDIP, SSOP, UQFN

Es así como se encuentran dos candidatos: *PIC18F27Q43* y *PIC18F27K42*; debido a la disponibilidad en el mercado nacional se optó por el microcontrolador *PIC18F27K42.*

Una vez hecha la elección del microcontrolador, se debe designar las terminales a emplear y su función. La programación y depuración del sistema es decisiva, y por esa razón, se inicia esta asignación considerando en primer lugar las terminales del

conector *In-Circuit Serial Programming ICSP*, que los microcontroladores *PIC* usan para su programación y depuración. Con respecto a este punto, se puede mencionar que, entre sus bondades, permite programar al microcontrolador sin la necesidad de desconectarlo de la tarjeta del circuito. El conector usa principalmente tres terminales, MCLR/RE3, PGC/ICSPCLK/RB6 y PGD/ICSPDAT/RB7, además de esto, es posible usar opcionalmente RB3 para la depuración en tiempo real con el programador ICD-U64 del fabricante *CCS Inc*. En la Figura 12 se muestra la configuración del conector *ICSP* y el botón de *reset*. Para evitar interferencia entre el programador y el circuito, se debe usar un acoplamiento entre las terminales de programación RB6, RB7 y RE3, el cual se realiza a través de un juego de resistencias que se conectan en estas terminales.

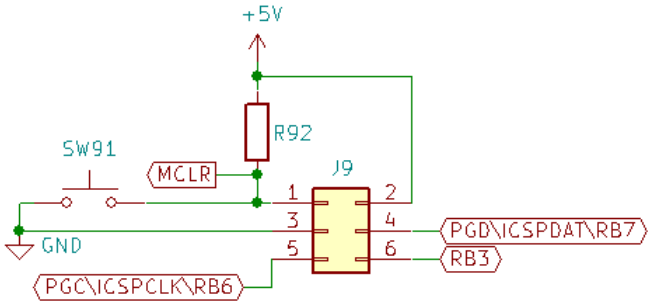


Figura 12: Conector ICSP y botón de reset.

Siguiendo con la designación, el circuito contempla una referencia de voltaje U7, de la cual, su valor de 2.5 V, será procesado por la terminal RA0, que corresponde al canal ANA0. La terminal responsable de medir la señal de voltaje que proviene de la línea es RA1, cuyo canal es el ANA1. Durante las pruebas, usando la configuración donde VSS está conectado a VREF- (NREF = 0), el microcontrolador no pudo medir señales entre 0 V a 0.5 V aproximadamente, este detalle se corrigió conectando RA2 a tierra, lo que significa usar la configuración donde VREF- está conectado a un VREF- externo (NREF = 1); RA2 corresponde al canal ANA2. La terminal con la cual se va a adquirir el voltaje que genera el sensor de corriente es RA3, que corresponde al canal ANA3 del microcontrolador. Las terminales RA0,

RA1 y RA3, están acomodadas de tal manera que, ninguna de estas interfiera con la otra al conectarse con el circuito integrado que contiene los amplificadores. El cristal oscilador se conecta usando las terminales RA6 y RA7 del microcontrolador. Para manejar y configurar el *display* de cristal líquido, se emplearán las terminales digitales RB1 a RB7; es de notar que, entre estas terminales, estarán conectadas las resistencias de acoplamiento R19, R110 y R111 para soportar adecuadamente el *ICSP*. RA4, RA5, RC0 a RC7 y RB0 son puertos sin uso, y para conservarlos en ese estado sin tener efectos no deseados se usan resistencias (de 10 kΩ) puestas a tierra (Microchip Technology Inc., 2009), además, deberán ser configurados como entrada (input); estas terminales podrán ser utilizados en un futuro al remover la resistencia que los conecta a tierra. En la Figura 13, se presenta el microcontrolador U1 PIC18F27K42.

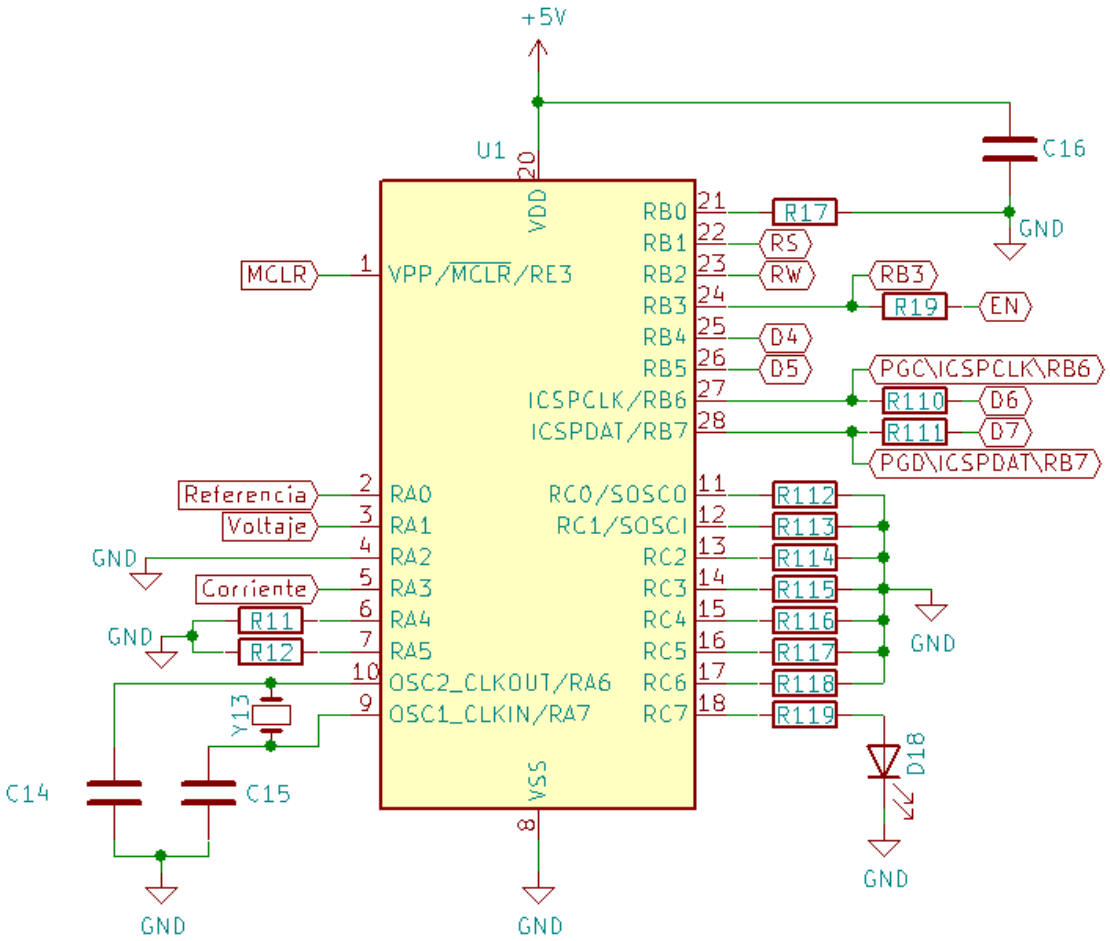


Figura 13: El microcontrolador y sus puertos

Para la programación y depuración, fue utilizado el programador ICD-U64, que es ideal por su fuerte integración con el *C-Aware IDE Compiler*, *IDE* utilizado para el desarrollo del *firmware*. Entre sus ventajas está su conexión y alimentación por *USB*, alimentación al microcontrolador con 3.3 V o 5.0 V sólo con un simple *jumper*, obtención de datos de diagnóstico en tiempo real (CCS Inc., 2020). Este programador viene con un conector *RJ12*, como se muestra en Figura 14.

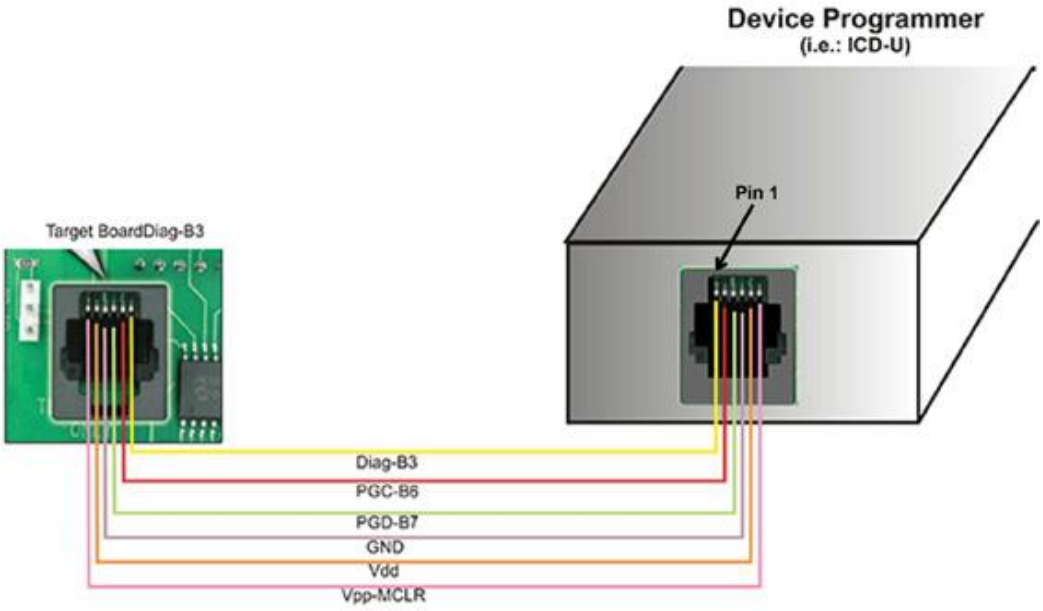


Figura 14: Configuración de terminales del ICD-U64.
Imagen con permiso de CCS Inc (CCS Inc., 2020)

4.4. LCD.

En los requisitos del sistema se había fijado usar un *LCD* de 16x2, pero la poca claridad con la que se leen las mediciones hizo que se meditara el uso de un *LCD* más grande. Al final se decidió un *LCD* de 20x4 *2004A*, sin marca de fabricante reconocible. Este *display* tiene una distribución de terminales compatible con otros fabricantes. La secuencia de inicialización es idéntica a la utilizada por el controlador de *display* de cristal líquido HD44780U, y las instrucciones aceptadas son en su mayoría compatibles. El voltaje de alimentación es de 5 V, muy apropiado al poder

seguir usando una fuente común. En la figura 15 se muestra su conexión y configuración de terminales y la figura 16 se muestra el *LCD* desplegando la información.

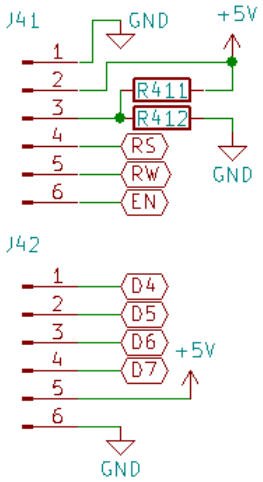


Figura 15: Terminales del LCD.

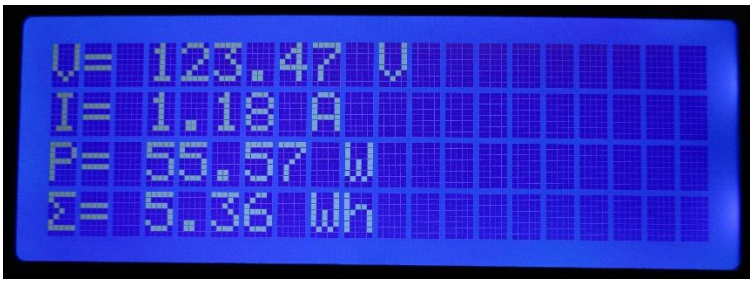


Figura 16: LCD 20x4 desplegando información.

4.5. Alimentación.

El sistema desarrollado se define como no autoalimentado, lo que representa la necesidad de una fuente externa principal para su funcionamiento. Existen en el mercado fuentes de alimentación conmutada, las cuales son una excelente opción por su pequeño tamaño y sus prestaciones. Esta clase de dispositivos se utilizan para energizar instrumental industrial, por nombrar un ejemplo. La fuente de alimentación conmutada disponible para este trabajo es la 78.12.1.230.1200, fabricada por *Finder*. Contiene protección contra cortocircuito y cuenta con un retardo de arranque menor a 1 s. La conexión de alimentación de entrada y de

voltaje de salida, se hace mediante borneras, y esta puede ser conectada junto con otra, formando un enlace dual, o también una conexión en serie. A continuación, se presentan los datos clave. Se aprecia la fuente en la figura 17.

- Tensión de entrada: 110 a 220 VAC, 50/60 Hz
- Tensión de salida: 12 V
- Corriente de salida: 1.25 A (-20 a +40 °C según el fabricante)
- Rango de temperatura de operación: -20 a 60 °C

Figura 17: Fuente de alimentación Finder.

La fuente provee la alimentación principal, pero esta a su vez debe dividirse en 2 valores, el primero es de 5 V, que requiere todos los circuitos del sistema. Para esta fuente de 5 volts principal, se contempló el regulador de voltaje *L7805ABP* del fabricante *ST Microelectronics*. Los puntos clave de este regulador, se citan a continuación:

- Voltaje de salida: 4.8 a 5.2 V, con valor típico de 5 V.
- Corriente de salida: 5 mA a 1 A.
- Voltaje de entrada: 7 V a 35 V.
- Rango de temperatura de operación: -40 °C a 125 °C
- Encapsulado TO- 220.

El fabricante recomienda aplicar un capacitor de derivación a la entrada del regulador, si entre este y la fuente de alimentación hay una gran distancia, o si existe

una gran capacitancia en la carga. El capacitor debe ser seleccionado con buenas características en alta frecuencia, para un funcionamiento estable en diferentes tipos de carga. Se recomienda un valor de 0.33 µF o mayor, además se pide materiales como tantalio, mylar u otros que tenga impedancia interna baja a altas frecuencias (ST Microelectronics, 2018). Un capacitor a la salida del regulador no es necesario para la estabilidad en la tensión de salida, pero mejora la respuesta transitoria; valores menores a 0.1uF podrían causar inestabilidad (Semiconductor Components Industries, LLC., 2014).

El segundo valor también es 5 V, pero debe tener aislamiento eléctrico del circuito principal; estos 5 volts aislados servirán para alimentar al amplificador de aislamiento. Para esta segunda fuente se eligió al convertidor *ROL-0505S* del fabricante *Recom Power*, entrega una tensión de salida aislada de 5 volts; además por su pequeño tamaño, es ideal para esta aplicación. Se nombran los detalles importantes sobre este convertidor.

- Voltaje de entrada: 5 V ± 10 %.
- Voltaje de salida: 5 V ± 5 %.
- Aislamiento: probado según el fabricante con 1 kV de corriente directa durante 1 s.
- Rango de temperatura de operación: -40 a 85 °C.
- Encapsulado SIP4.

5. Software.

5.1. El Planificador.

La coordinación y secuencia de ejecución de las tareas recae en un planificador (*scheduler*), el cual tiene la responsabilidad de manejar los recursos computacionales y de datos que el sistema necesita para su funcionamiento (Nahas & Nahhas, 2012). En particular, se requiere una planificación híbrida (*hybrid scheduling*), que se caracteriza por tener una sola tarea configurada como preventiva (*pre-emptive*), mientras las demás esperan a ser ejecutadas secuencialmente, no interrumpiendo a las otras (Nahas & Nahhas, 2012).

5.1.1. Implementando el planificador.

Para implementar el planificador, se usó un súper bucle (Nahas & Nahhas, 2012), que en su interior están las llamadas a las tareas que ejecuta secuencialmente. La tarea 0, es la única tarea que interrumpe a las demás. Se ilustra el súper bucle y la función tarea0.

main.c

```
int
main (void)
{
…
  for (;;)
{
        tarea1();
        tarea2();
        //...
        //tareaN();
  }
  return 0;
}
```

tarea0.c

```
#INT_CCP1 HIGH
void
tarea0(void)
{
…
};
```

5.2. Las tareas.

Para este trabajo se propone la clasificación de funcionamiento en tareas, las cuales deben cumplir con las operaciones para llegar al resultado final y despliegue de datos. Se considera que el sistema debe cumplir con cuatro tareas, las cuales se describen a continuación.

- Tarea 0 - Obtener las lecturas de voltaje y corriente, y transferirlas a un buffer temporal para su posterior lectura. Esta tarea no deberá ser interrumpida por ninguna otra.
- Tarea 1 – Transferir datos del buffer temporal a contenedores de datos para el cálculo de la potencia.
- Tarea 2 - Calcular el consumo de la energía eléctrica.
- Tarea 3 - Desplegar datos por *display LCD*.

La figura 18 muestra simplificadamente la relación y función de estas tareas.

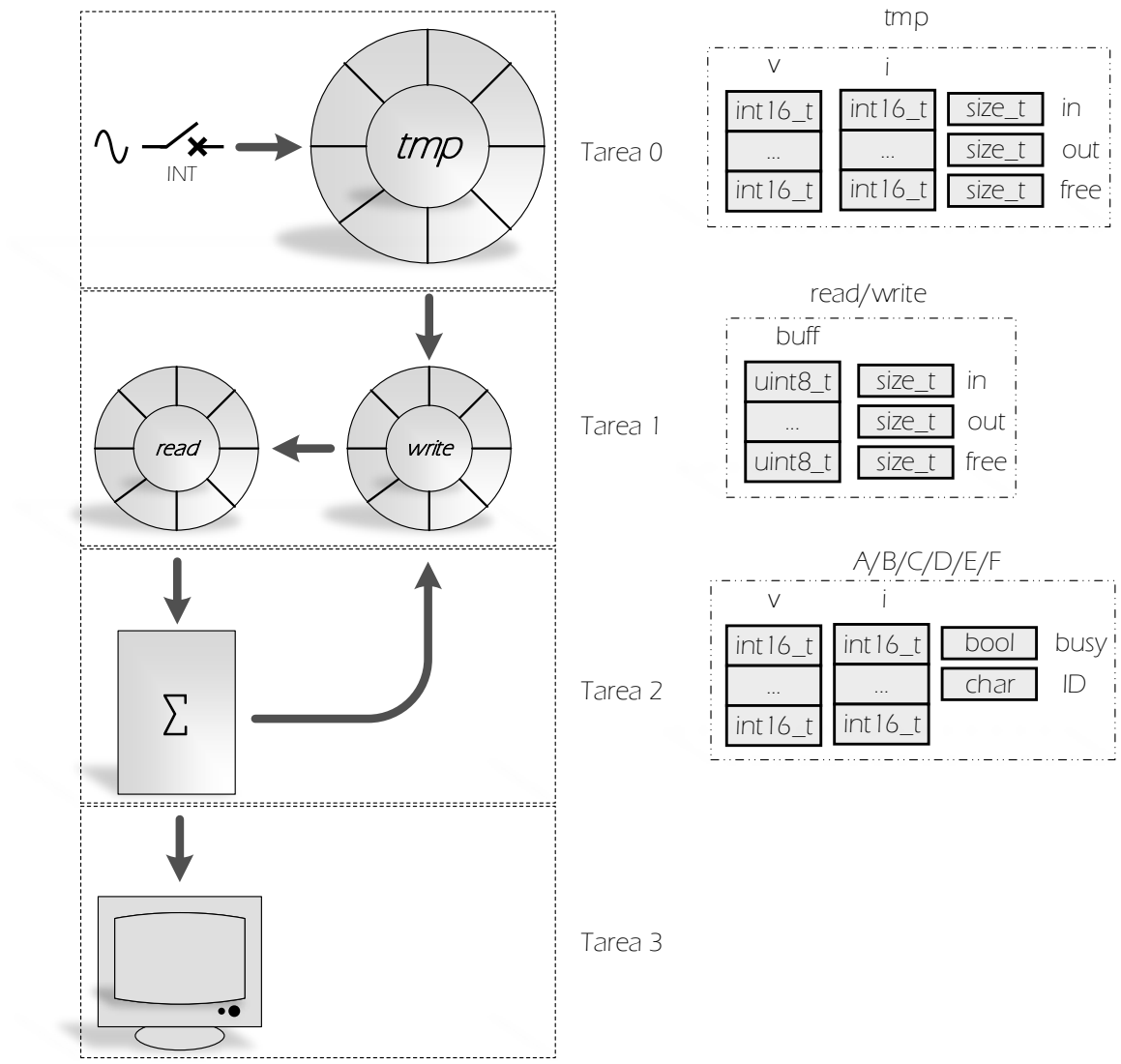


Figura 18: Relación entre las tareas.

5.2.1. Secuencia de ejecución de las tareas.

La secuencia de ejecución de las cuatro tareas debe seguir el siguiente orden, para hacer del tratamiento de datos un proceso coherente:

1. La tarea 0 debe ejecutarse cada vez que una interrupción sea activada, y esta acción no deberá ser detenida por ningún otro proceso.
2. La tarea 1 es la primera en ser invocada por el planificador, ya que esta función es la encargada de mover los datos listos para su cálculo.
3. La tarea 2 es llamada al terminar la tarea 1, para comenzar el cálculo con los datos transferidos por ésta.
4. La tarea 3 es invocada inmediatamente después de la tarea 2, esta imprime los resultados listos.
5. Se inicia el ciclo de nuevo.

En la figura 19, se observa la secuencia que sigue la ejecución de estas tareas, considerando la interrupción de la tarea 0.

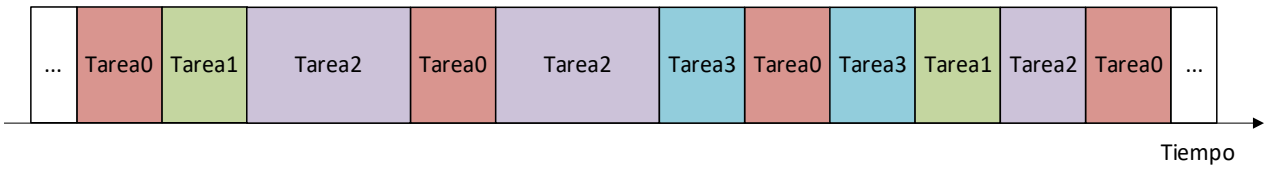


Figura 19: Diagrama de tiempo.

5.2.2. Tarea 0.

La tarea 0 es la encargada de obtener periódicamente los datos que provienen del convertidor *ADC* y almacenarlos. La obtención de los datos es una tarea que debe ejecutarse en tiempo y no puede ser interrumpida, ya que, de no ser así, las mediciones estarán comprometidas por falta de datos.

Para ejecutar la tarea 0 periódicamente, se usó el módulo *CCP1* (*Capture\Compare\PWM*) del microcontrolador, en modo comparación. El modo

comparación permite contrastar repetidamente el valor *Timer 1* con el de *CCP1*, y cuando estos dos valores coinciden, se produce una interrupción. Es posible modificar el tiempo en que la interrupción se vuelve a activar al cambiar el valor de *CCP1*.

Los datos obtenidos del convertidor analógico digital son almacenados en un buffer circular de tipo *c_buffer_tmp_t* llamado *tmp*, el cual es una estructura intermedia, donde se registran todos los datos obtenidos. La razón de usar esta estructura es evitar código con condicionales que varíen la periodicidad en la ejecución de la tarea 0.

5.2.3. Tarea 1.

La tarea 1 es la encargada de transferir los datos del buffer circular *tmp* a una de las seis estructuras contenedoras, definidas como *A, B, C, D, E* y *F* de tipo *d_buffer_t*. Este tipo de contenedor encierra el número de muestras que, representa un ciclo completo de las señales de voltaje y corriente. Cuando alguno de los seis contenedores ha sido escrito en su totalidad, significa que ya está listo para ser procesado, y para este fin, el contenedor lleno es enviado a otro *buffer* circular de tipo c_buffer_t llamado *read*, cuya función es almacenar todos los contenedores listos para ser leídos.

5.2.4. Tarea 2.

La tarea 2 es la encargada de procesar los datos de los contenedores *read*. Inicia consultando que contenedores temporales con datos están listos para ser leídos. Se extrae el contenedor listo que está lleno de datos y se inicia el cálculo. Una vez finalizado, el contenedor vacío es enviado a un *buffer* circular llamado *write* de tipo *c_buffer_t*. Este *buffer* contiene los contenedores libres listos para ser escritos con nuevos datos. Por último, los resultados son almacenados en una estructura llamada *final_result* de tipo *f_result_t*, que consultará la tarea 3.

5.2.4.1. Cálculo de la energía.

Para realizar el cálculo de la energía, se toma como base las ecuaciones (7), (8) y (10). Ya que este cálculo se basa en acumular resultados, y para evitar la posibilidad de desborde provocado por sumas consecutivas de grandes números, se contempló dividirlo en sumas más pequeñas. El resultado final es desplegado en unidades watt hora.

5.2.4.2. Programación del algoritmo.

Se implementaron dos algoritmos de integración numérica, el método rectangular (descrito por las ecuaciones (7) y (8)) y regla de Simpson (descrito en las ecuaciones (8) y (11)), ambos están disponibles en el *firmware*, y pueden ser activados definiendo *P_RECT* en el caso del método del rectángulo, y *P_SIMPSON* en el caso de la regla de Simpson. Se muestra el desarrollo del código para el cálculo de la energía eléctrica, este sirve de base para entender el código con el cual se computan los valores *RMS* y la potencia instantánea.

5.2.4.3. Cálculo de componente DC.

El sistema, calcula en tiempo de ejecución la componente *DC* con la ecuación (6) y la resta a cada muestra de la señal voltaje/corriente que registra. Los datos de las señales están almacenados en arreglos unidimensionales llamados *v* e *i*, los cuales se puede acceder directamente. Los valores finales de la componente se almacenan en las variables *i_dc* y *v_dc.*

```
float v_dc = 0.0, i_dc = 0.0;

for ( size_t j = 0; j < N; j++)
{
        i_dc += i[j];
        v_dc += v[j];
}

i_dc /= N;
v_dc /= N;
```

5.2.4.4. Método del rectángulo.

Se muestra la implementación de las ecuaciones (7) y (8) considerando la sustracción de la componente *DC*. En ésta, los datos de las señales están almacenados en arreglos unidimensionales llamados *v* e *i*, los cuales se puede acceder directamente.

```
static size_t count_s = 0;
static float ECA = 0.0;
float ECSUB = 0.0;

for ( size_t j = 0; j < N; j++)
{
        ECSUB = (v[j] – v_dc) * (i[j] – i_dc);
}
ECA += ECSUB / N;
cout_s++;
```

5.2.4.5. Regla de Simpson.

Se presenta la implementación de las ecuaciones (8) y (10) considerando la sustracción de la componente *DC*. En esta, los datos de las señales están almacenados en arreglos unidimensionales llamados *v* e *i*, a los cuales se puede acceder directamente.

```
static size_t count_s = 0;
static float ECA = 0.0;
float ECSUB = 0.0;

for ( size_t j = 0; j < N; j++)
{
        i_dc += i[j];
        v_dc += v[j];
}

I_dc /= N;
V_dc /= N;

for ( size_t j = 0; j < N - 2; j+2)
{
        ECSUB = (v[j] – v_dc) * (i[j] – i_dc) \
        + 4 * (v[j+1] – v_dc) * (i[j+1] – i_dc) \
        + (v[j+2] – v_dc) * (i[j+2] – i_dc);
}

ECA += ECSUB / (3 * N);
count_s++;
```

5.2.4.6. Obteniendo watt hora.

Para cualquiera de los dos métodos de integración que se use, se requiere procesar sesenta ciclos de señal. El resultado es acumulado en la variable llamada *ECA*. En esta variable se concentran las sesenta integrales que significa el procesamiento de 2700 muestras. En este punto, para ajustar el número de datos *N* en la ecuación (6), es preciso dividir *ECA* sobre 60, ajustando de esta forma el valor de *N* a 2700.

En este momento el valor de *ECA* muestra «watt segundos», ya que los sesenta ciclos procesados corresponden a 1 s. Para convertir este valor a watt hora, se debe hacer una conversión de unidades.

Considerando lo anterior, se muestra la implementación.

```
static float ECA_s = 0.0;
if (60 == count_s)
   {
       ECA_s += ECA / 216000.0F;
      ECA = 0.0;
      count_s = 0;
   }
```

5.2.5. Tarea 3.

Su función es desplegar los datos en el *LCD*, contenidos en la variable *final_result*. Se utiliza la función *printf* para enviar la información al *LCD*. En esta tarea, es posible agregar código adicional para tener otras opciones, como enviar los datos a otro dispositivo o almacenarlos en una memoria.

5.3. Programa de prueba.

Se realizó un programa de prueba de nombre *test.c*, cuyo fin es verificar el funcionamiento de las tareas que el *firmware* ejecuta usando una breve simulación de la interrupción principal y del planificador, esta práctica nos brinda la posibilidad de revisar los resultados obtenidos. Por otro lado, este programa de prueba ayuda a una segunda revisión de sintaxis por parte de otro compilador, y así revelar

posibles errores ocultos en código que el compilador *CCS* pueda omitir. El código fuente de este programa se incluye en el apéndice D de este trabajo.

6. Ajuste del valor CCP1.

Es necesario realizar un ajuste en el valor de *CCP1*, declarado en el archivo *adj.h*. El fin de este procedimiento es hacer coincidir el número de interrupciones por ciclo con el número de muestras por ciclo de señal, para de esta forma tener la base de tiempo en un valor correcto. Para lograrlo, es necesario cambiar el valor *CCP1*, compilar el programa, cargarlo al microcontrolador y medir la frecuencia de la señal proveniente de la terminal PIN_DBG (B0), este valor dividido entre 60 debe ser igual o muy cercano al valor de muestras por ciclo que se procesan.

En este caso, el número de muestras por ciclo a procesar es de 45, lo que significa que el valor de *CCP1* debe ser ajustado para que la frecuencia de la señal en la terminal B0 sea muy cercana a 2.7 kHz. En la figura 20 se muestra la forma de la señal en la terminal B0, así como la medición obtenida de un osciloscopio.

Figura 20: Medición de frecuencia con osciloscopio.

7. Pruebas.

Para evaluar el funcionamiento del instrumento se propone comparar el cálculo de energía consumida contra algún otro medidor de energía comercial. Para este fin, el instrumento que servirá como patrón de referencia es un medidor *Kill A Watt* del fabricante *P3 International* mostrado en la figura 21. Este instrumento cuenta con lectura en kilowatt hora.

Figura 21: Kill A Watt.

Las características del instrumento son las siguientes (P3 International Corporation., 2018):

- Voltaje Máximo: 125 VAC.
- Corriente Máxima: 15 A.
- Muestra Volts, Amperes, Watts, Hz.
- 0.2 % de exactitud

Como carga, se utilizó un motor monofásico marca *Robbins & Myers*, con las siguientes características:

- Voltaje: 115 V
- Corriente: 0.8 A
- frecuencia: 60 Hz
- Revoluciones:106 RPM

Se muestra el motor en la figura 22.

Figura 22: Motor para prueba.

La configuración usada se muestra en la figura 23. El medidor se conectó directamente a la línea, mientras que el medidor patrón *Kill A Watt* fue puesto en las terminales de carga del prototipo. A su vez, el motor fue puesto en la terminal de carga del instrumento patrón. Adicionalmente se utilizó un medidor de frecuencia para verificar que la señal que refleja la interrupción coincida con 2700 Hz. En la figura 24 se observa la configuración funcionando.

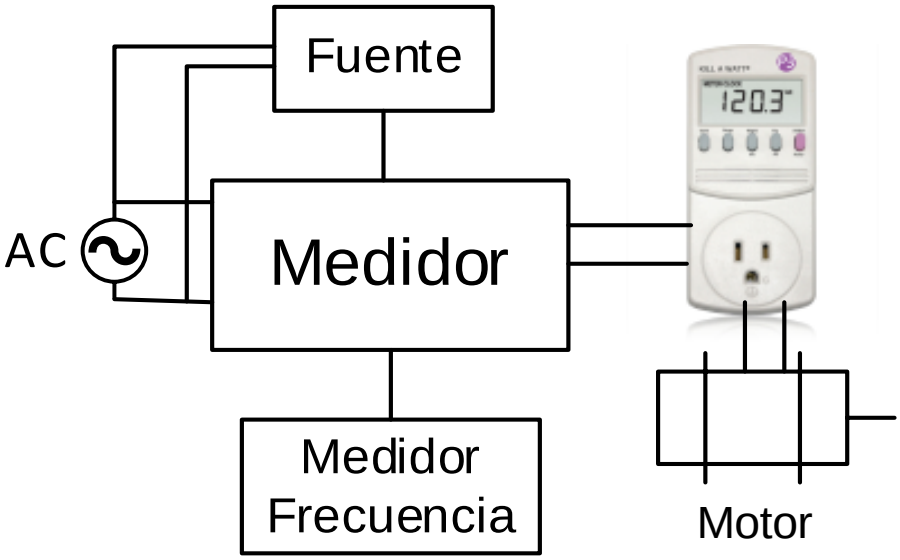


Figura 23: Configuración para comparación de datos.

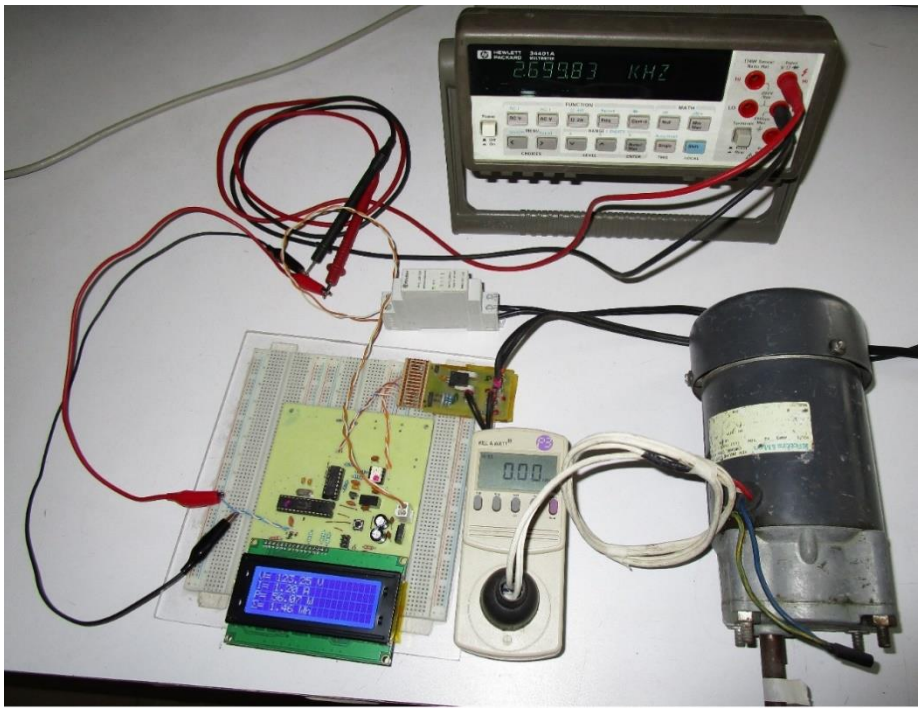


Figura 24: Conjunto de instrumentos en prueba.

Se plantea como prueba de evaluación, registrar diez lecturas de energía (10, 20, 30, 40, 50, 60, 70, 80, 90 y 100 Wh) usando los dos algoritmos de integración numérica y evaluar el resultado usando como referencia el factor de corrección aplicable a cada lectura. El factor de corrección (FC) se define en la ecuación (17).

$$FC = \frac{Lectura\ Patrón}{Lectura\ Medidor} \qquad (17)$$

Y con el fin de examinar el desempeño del microcontrolador, se propone el uso del índice de trabajo (IT) que se define en la ecuación (18):

$$IT = \frac{Número\ de\ datos\ nuevos\ entrantes}{Número\ de\ datos\ liberados} \qquad (18)$$

El índice IT se obtendrá al medir el número de ejecuciones de tarea 0 durante un numero conocido de llamadas a la función tarea 2, como se muestra en la figura 25.

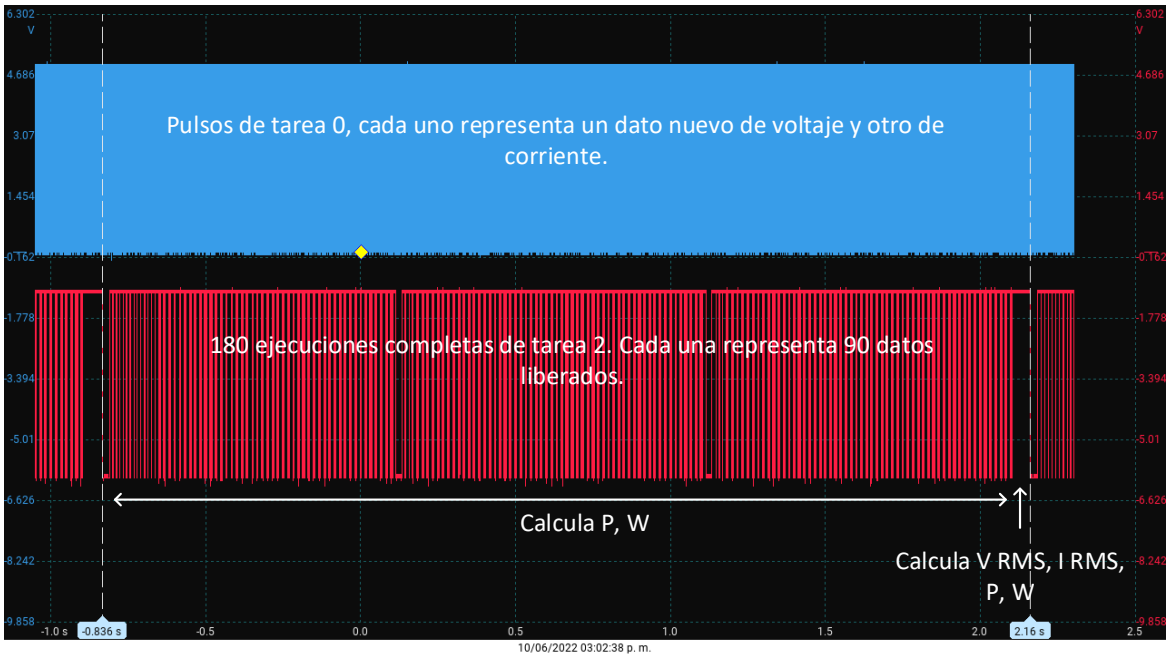


Figura 25: Pulsos producidos por tarea 0 y tarea 2.

Un valor de IT igual a uno, representa un equilibrio entre dato entrante y dato liberado. Es deseable un valor menor a uno, esto garantiza que el sistema se mantenga con los *buffers* de datos en un mínimo de uso. Un IT mayor a uno implica un error por pérdida de datos, muestra la incapacidad del sistema en liberar memoria a tiempo. Para este trabajo se propone un IT menor a 0.5, se considera que el sistema embebido tendrá recursos disponibles para otras tareas, sin poner en riesgo la integridad de los datos para la medición.

Para la estimación de la relación entre datos entrantes y liberados, se modifica el programa para que en cada llamada de la función tarea 2 un pulso indique la ejecución y su duración. Se propone medir los pulsos producidos por tarea 0 en 180 llamadas de la función tarea 2, éstas representan 16200 datos liberados. Se considera que en este número de invocaciones se incluyen en su totalidad las tareas que componen al sistema embebido. Contando los flancos ascendentes o descendentes que produce la tarea 0 en este periodo de tiempo, se obtiene el número de datos entrantes. Se realizan diez repeticiones de esta estimación.

7.1. Resultados.

Las lecturas obtenidas en las pruebas se muestran en la tabla 4 y 5, la gráfica de los factores de corrección se aprecia en la figura 26 y la variación del índice de trabajo se muestra en la figura 27. Se obtiene como resultado lo siguiente:

- De las diez lecturas propuestas para hacer la comparación sólo fue posible obtener cinco usando regla de Simpson y nueve usando método del rectángulo, ya que ocurre el error donde el buffer *tmp* se desborda y no hay espacio para almacenar datos mientras otros se siguen procesando.
- La estabilidad del factor de corrección usando el método del rectángulo fue mejor que con regla de Simpson.
- Es posible aplicar un factor de corrección de 1.087 a las lecturas del medidor teniendo un error de hasta 0.11 % al usar el método del rectángulo.
- Es posible aplicar un factor de corrección de 1.109 a las lecturas del medidor teniendo un error hasta de 0.21 % al usar regla de Simpson.
- La etapa de acoplamiento de señal funciona como se tenía previsto.
- Tanto la fuente de alimentación *Finder*, el regulador que provee a toda la electrónica de 5 *volts*, así como el convertidor *DC/DC* mostraron una buena estabilidad en la tensión de salida.
- Durante las pruebas no se presentaron problemas de funcionamiento anómalo. No fue necesario instalar las resistencias que se tenían previstas para evitar problemas por ruido excesivo en el microcontrolador.
- El índice de trabajo no es constante y presenta fluctuaciones. Estas variaciones en el tiempo que tarda la ejecución de la tarea 2, se relaciona a que los cálculos que efectúa el microprocesador no siempre se hacen de la misma manera todas las veces, esto se puede notar en la primera estimación de tiempo que se hizo en la sección «microcontrolador» de este trabajo, donde en el conteo de ciclos de instrucción se notan saltos condicionales. Por otro lado, la señal que produce la función tarea 0 no cambió de frecuencia, descartando alguna perturbación en el reloj de CPU.

- Los valores de IT oscilan el valor unitario, mostrando un frágil equilibrio que puede romperse con alguna situación donde se requiera consumir más tiempo.

Tabla 4: Medición con patrón y prototipo, y cálculo de FC.

	Método del Rectángulo				Regla de Simpson			
Patrón (Wh)	Prototipo (Wh)	FC Rectangular	Lectura corregida	% Error	Prototipo (Wh)	FC Simpson	Lectura corregida	% Error
10	9.21	1.0858	10.01	0.11	9.03	1.107	10.01	0.16
20	18.39	1.0875	19.99	0.05	18	1.111	19.96	0.21
30	27.6	1.0870	30.00	0.00	27.02	1.110	29.97	0.13
40	36.79	1.0873	39.99	0.02	36.09	1.108	40.02	0.07
50	45.99	1.0872	49.99	0.02	45.05	1.110	49.96	0.09
60	55.18	1.0874	59.98	0.03	FC	1.109		
70	64.38	1.0873	69.98	0.03				
80	73.57	1.0874	79.97	0.04				
90	82.78	1.0872	89.98	0.02				
	FC	1.087						

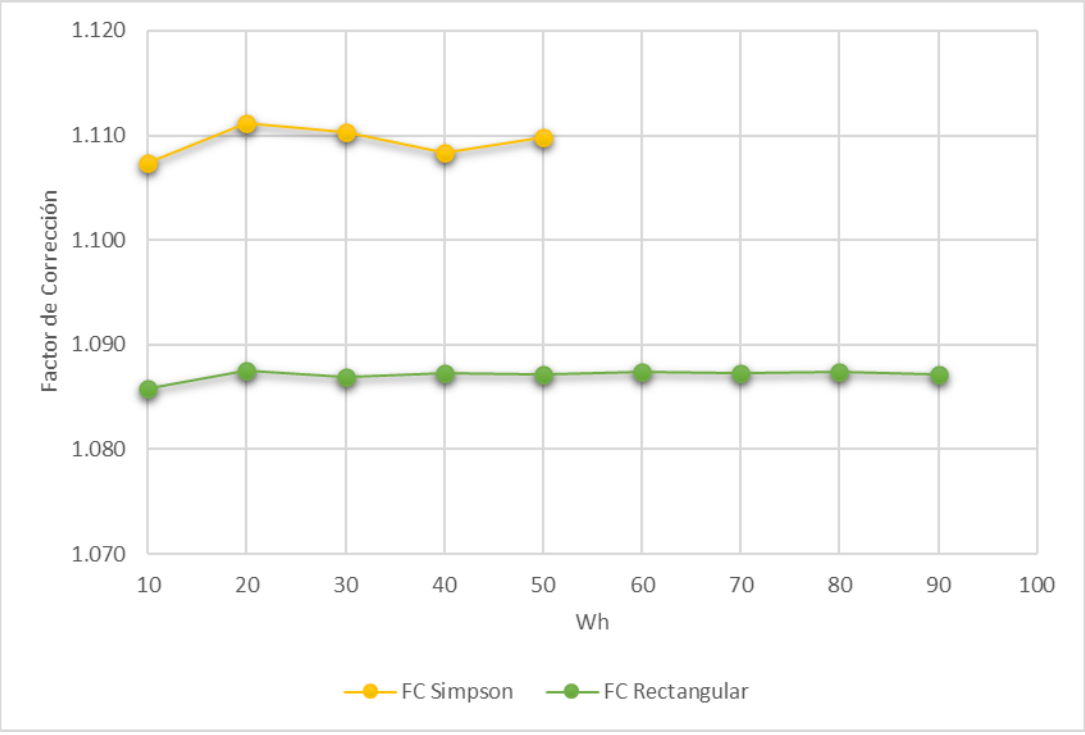


Figura 26: Grafica donde se muestra la variación en el factor de corrección.

Tabla 5: Medición de cuenta de pulsos y cálculo del IT.

	Cuenta de pulsos			
Lectura	Rectángulo	IT	Simpson	IT
1	8115	1.0019	8111	1.0014
2	8122	1.0027	8102	1.0002
3	8092	0.9990	8088	0.9985
4	8094	0.9993	8113	1.0016
5	8104	1.0005	8108	1.0010
6	8087	0.9984	8066	0.9958
7	8097	0.9996	8133	1.0041
8	8081	0.9977	8055	0.9944
9	8087	0.9984	8097	0.9996
10	8092	0.9990	8095	0.9994

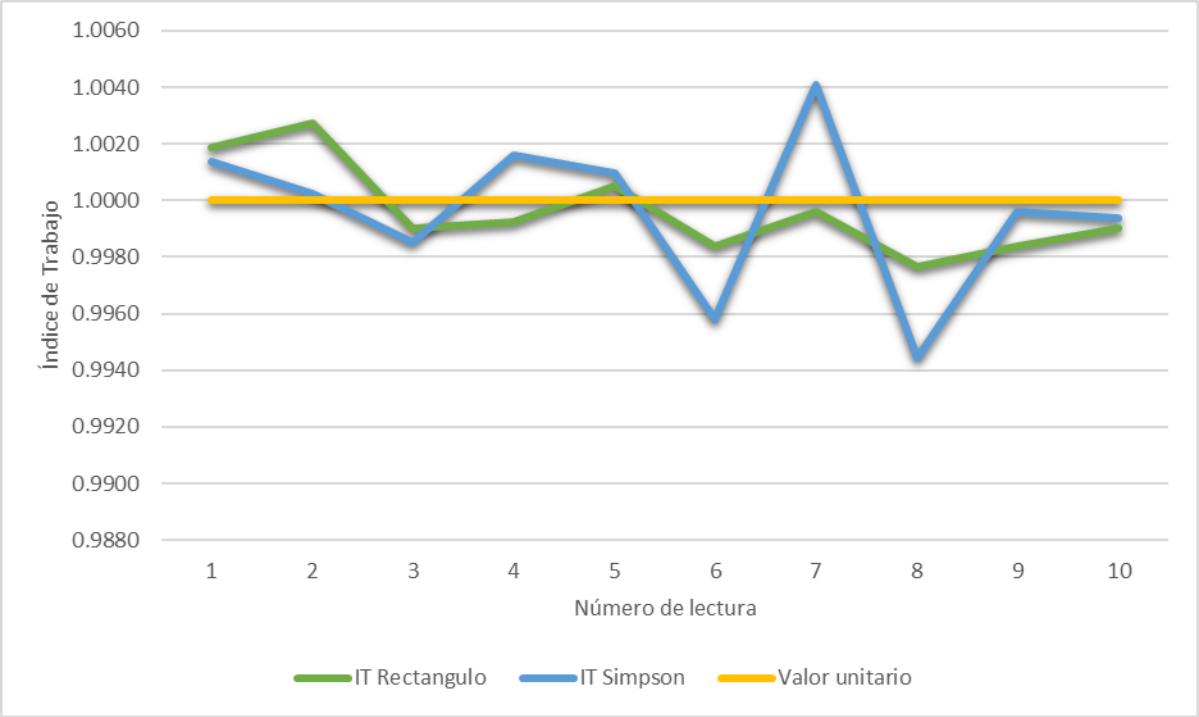


Figura 27: Grafica donde se muestra la variación del índice de trabajo.

8. Conclusiones.

Se concluye que es posible diseñar un watthorímetro como un sistema embebido basado en un microcontrolador *PIC*, teniendo buenos resultados en la medición, pero la combinación *hardware-software* mostrada en este trabajo tiene algunos puntos a desarrollar.

Es posible mantener al modelo de microcontrolador *PIC18F27K42* como núcleo del sistema embebido, pero se deberá optimizar la función tarea 2 con el objetivo de lograr un valor de IT menor a uno en todo momento. Un valor ideal para este proyecto será 0.5 si se desea agregar más tareas al sistema. En el estado actual del proyecto la operación es posible, pero la variación que tiene el índice de trabajo por arriba de uno hace que un error de desbordamiento sea inminente. Otras opciones para obtener un valor de IT ideal es cambiar de microcontrolador por alguno que incluya una unidad de punto flotante (*FPU*), la cual pueda realizar el cálculo de una forma más ágil, un candidato podrá ser la familia *PIC32MZ*, pero cambiar a este microcontrolador significará invertir en software de desarrollo y algunos cambios en tensión de alimentación, así como de nivel de voltaje en las señales. Otro camino a seguir es hacer del cálculo de la potencia algo externo mediante electrónica analógica, la cual se encargue de las operaciones de multiplicación e integración de las señales y posteriormente el microcontrolador adquiera, acumule e imprima el resultado.

Los algoritmos implementados en este trabajo tienen buenos resultados, la forma en que se programan es funcional pero susceptible de optimización. El tiempo de ejecución de ambos algoritmos es similar dado que el valor de IT es muy semejante. Lo anterior indica que cualquiera de los dos puede usarse sin tener grandes repercusiones en la operación del instrumento, pero es muy deseable bajar los tiempos de ejecución para mantener un trabajo constante y sin errores de desbordamiento en memoria.

La electrónica para el tratamiento de las señales, así como alimentación funcionaron correctamente, pero debido a que el sistema es propenso a presentar errores, no se pudo confirmar el comportamiento bajo múltiples horas de funcionamiento.

Por último, el tener claras las necesidades a cubrir con el sistema embebido, hizo del proceso de diseño algo más fluido ya que, de esta forma, se entiende al sistema como una unidad y no como algo fragmentado o separado. Un buen diseño deberá comprender como interactúan las partes entre sí antes de empezar alguna implementación.

Bibliografía

[1] A. Svoboda, J. & C. Dorf, R., 2015. *Circuitos Electricos.* Novena ed. s.l.:Alfaomega.

[2] Allegro MicroSystems. L.L.C., 2019. *ACS756xCB - Fully Integrated, Hall-Effect-Based Linear Current Sensor IC with 3 kVRMS Voltage Isolation and a Low-Resistance Current Conductor.* s.l.:s.n.

[3] Avago Technologies, 2012. *HCLP-7840 Isolation Amplifier Data Sheet.* s.l.:s.n.

[4] Barr, M., 1999. *Programming Embedded Systems in C and C++.* [En línea]
Available at: https://barrgroup.com/embedded-systems/books/programming-embedded-systems/introduction
[Último acceso: 19 febrero 2020].

[5] Bimenyimana, S. & Osarumwense, G., 2018. Traditional Vs Smart Electricity Metering Systems: A Brief Overview. *Journal of Marketing and Consumer Research,* Volumen 46.

[6] Castillo Hernández, J., de Gortari Briseño, J., Caballero Ruiz, A. & Ruiz Huerta, L., 2019. DESARROLLO DEL PROTOTIPO DE UN WATTHORÍMETRO DIGITAL. *Pistas Educativas,* noviembre, 41(134), pp. 119-134.

[7] CCS Inc., 2020. *Frequently Asked Questions.* [En línea]
Available at: http://www.ccsinfo.com/faq.php?page=connect_icd

[8] CCS Inc., 2020. *ICD-U64 In-Circuit Programmer/Debugger.* [En línea]
Available at: http://www.ccsinfo.com/product_info.php?products_id=icd_u64

[9] Crenshaw, J. W., 2000. *Math Toolkit for Real-Time Programming.* s.l.:CMP Books, CMP Media, Inc..

[10] Heath, S., 2003. *Embedded Systems Design.* Segunda ed. s.l.:Newnes.

[11] Ibrahim, D., 2008. *Advanced PIC Microcontroller Projects in C: From USB to RTOS with the PIC.* Primera ed. s.l.:Elsevier.

[12] Lanphier, R., 1925. *Electrical Meter History and Progress.* Springfield, Illinois: Sangamo Electric Company.

[13] Microchip Technology Inc., 2009. *Compiled Tips 'N Tricks Guide.* s.l.:s.n.

[14] Murti, K., 2022. *Design Principles for Embedded Systems.* s.l.:Springer.

[15] Nahas, M. & Nahhas, A. M., 2012. Ways for Implementing Highly-Predictable Embedded Systems Using Time-Triggered Co-Operative (TTC) Architectures. *EMBEDDED SYSTEMS – THEORY AND DESIGN METHODOLOGY,* Febrero.pp. 3-30.

[16] P3 International Corporation., 2018. *P3 International Products.* [En línea]
Available at: http://www.p3international.com/products/p4400.html
[Último acceso: 8 mayo 2022].

[17]Sanchez, J. & P. Canton, M., 2007. *Microcontroller Programming, The Microchip PIC..* Primera ed. s.l.:CRC Press.

[18]Semiconductor Components Industries, LLC., 2014. *MC7800, MC7800A, MC7800AE, NCV7800 1.0 A Positive Voltage Regulators.* s.l.:s.n.

[19]ST Microelectronics, 2018. *L78 Datasheet Positive voltage regulator ICs.* s.l.:s.n.

[20]W. Hart, D., 2001. *Electrónica de Potencia..* s.l.:PRENTICE HALL .

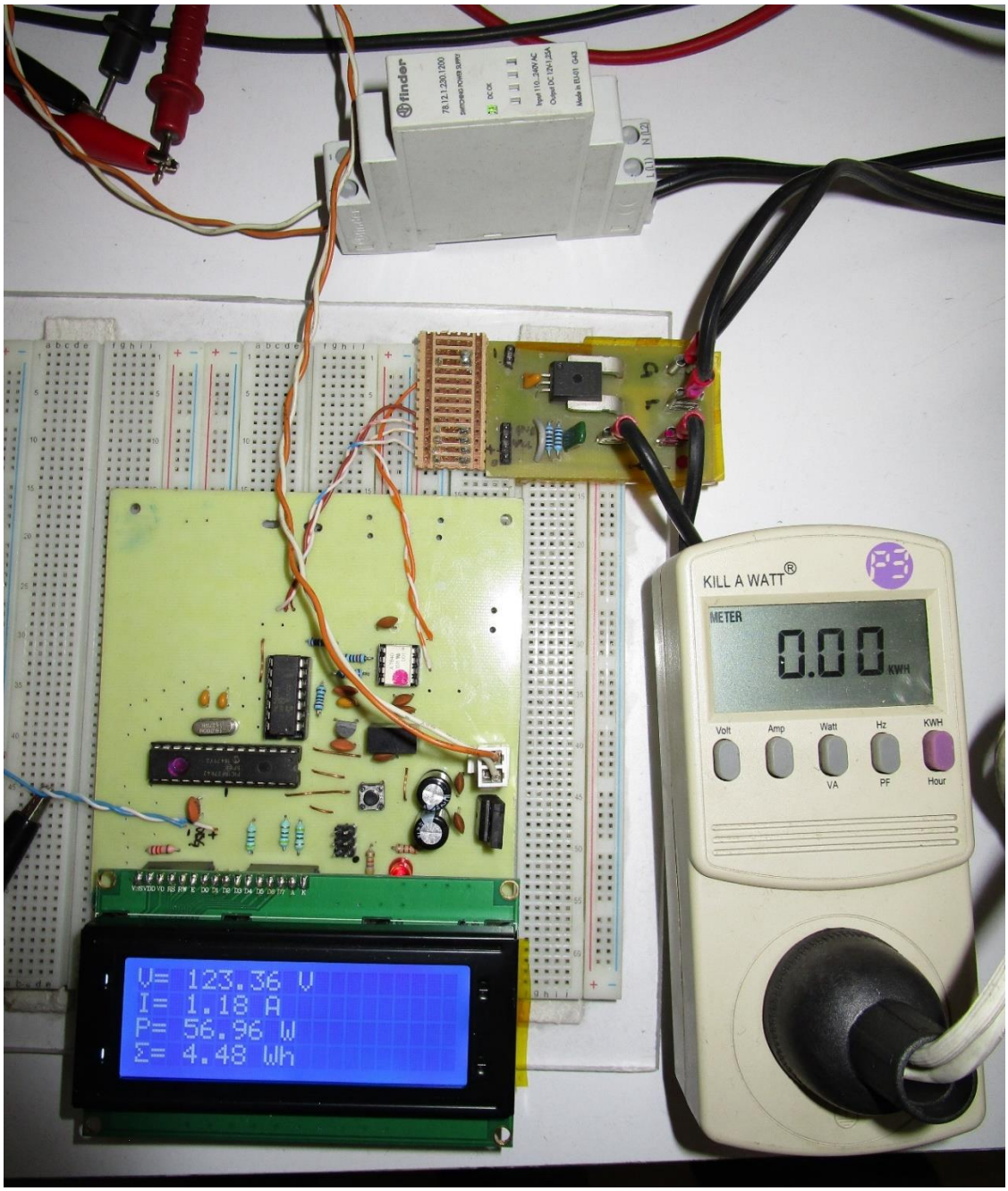
[21]Weranga, K. S. K., Kumarawadu, S. & Chandima, D. P., 2014. *Smart Metering Design and Applications.* s.l.:Springer.

[22]Yumak, K. & Usta, O., 2011. *A Controversial Issue: Power Components in Nonsinusoidal Single-Phase Systems.* s.l.:s.n.

Apéndice A: fotografía del prototipo, PCB, lista de materiales y esquemático.

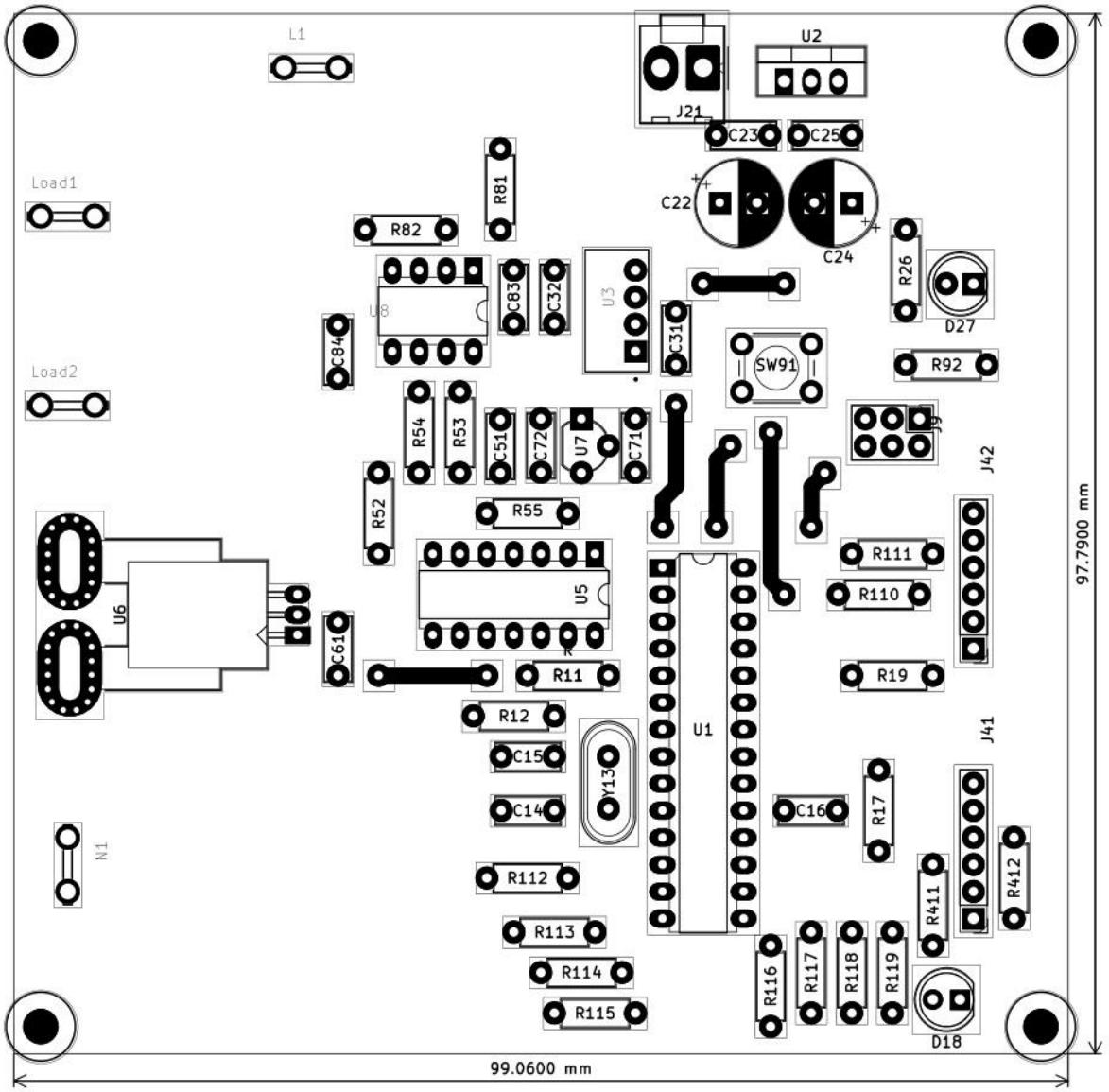
Fotografía del prototipo.

Se observa el sistema en su totalidad en la siguiente fotografía.

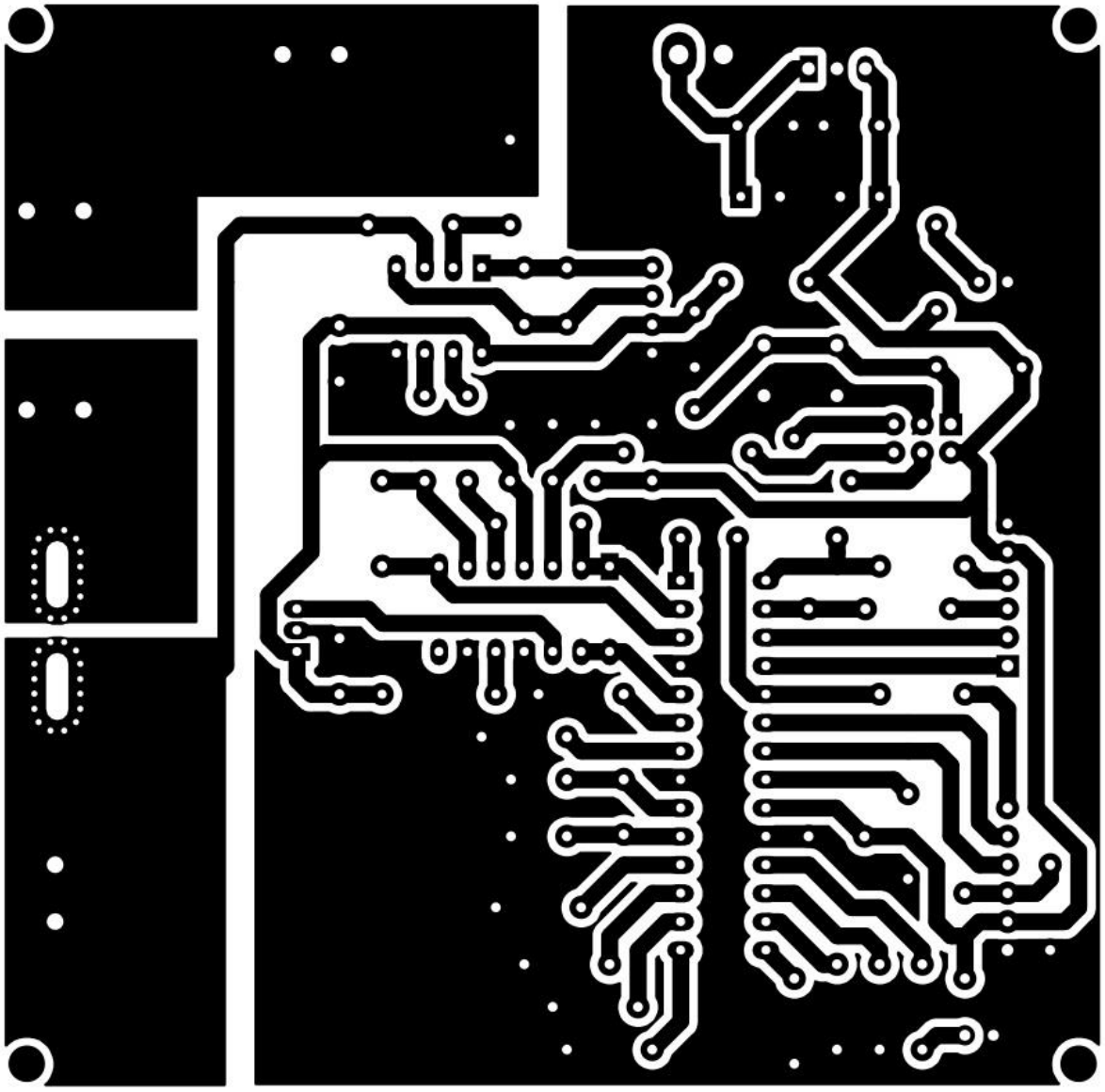


El esquemático y el circuito impreso fueron realizados con KiCad 6.0.2.

Cara superior PCB.



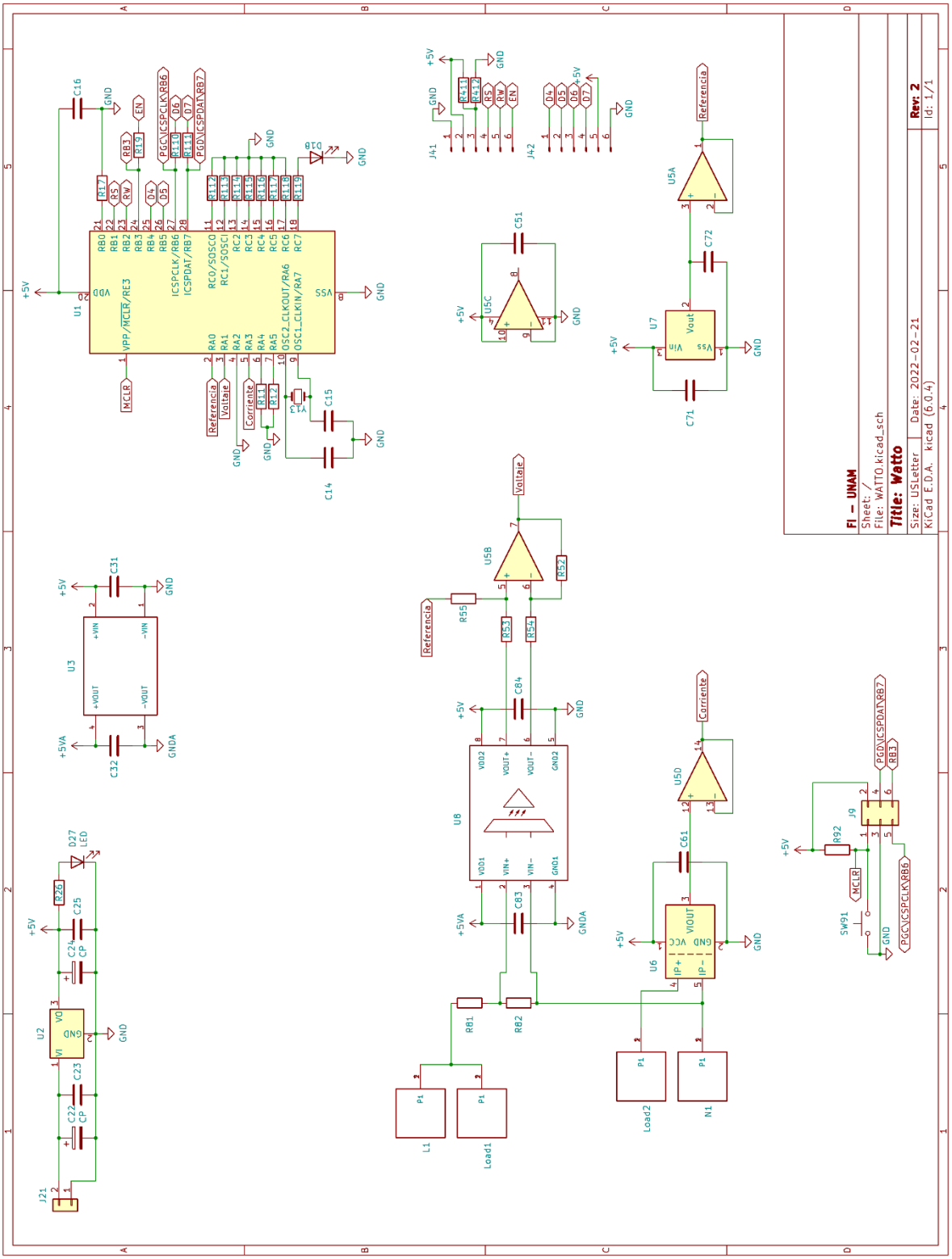
Cara inferior PCB.



Lista de Materiales.

Circuito No.	*Descripción*
C14	Cerámico 68 pF
C15	Cerámico 68 pF
C16	Cerámico 0.1 uF
C22	Electrolítico 220uF
C23	Cerámico 0.1uF
C24	Electrolítico 100uF
C25	Cerámico 0.1 uF
C31	Cerámico 0.1 uF
C32	Cerámico 0.1 uF
C51	Cerámico 0.1 uF
C61	Cerámico 0.1 uF
C71	Cerámico 0.1 uF
C72	Cerámico 1 uF
C83	Cerámico 0.1 uF
C84	Cerámico 0.1 uF
D18	LED
D27	LED
J9	Header Macho ICSP
J21	Molex Macho
J41	Header Hembra LCD
J42	Header Hembra LCD
R11	1 kΩ ±5 %
R12	1 kΩ ±5 %
R17	1 kΩ ±5 %
R19	470 Ω ±5 %
R26	330 Ω ±5 %
R52	2.2 kΩ ±1 %
R53	2.2 kΩ ±1 %
R54	2.2 kΩ ±1 %
R55	2.2 kΩ ±1 %
R81	1 MΩ ±1 %
R82	500 Ω ±1 %
R92	47k Ω ±5 %
R110	470 Ω ±5 %
R111	470 Ω ±5 %
R112	1 kΩ ±5 %
R113	1 kΩ ±5 %
R114	1 kΩ ±5 %
R115	1 kΩ ±5 %
R116	1 kΩ ±5 %
R117	1 kΩ ±5 %
R118	1 kΩ ±5 %
R119	1 kΩ ±5 %
R411	12k Ω ±5 %
R412	1k Ω ±5 %
SW91	RESET SW
U1	PIC18F27K42SP
U2	L7805ABP
U3	ROL-0505S
U5	MCP6004I/P
U6	HCPL-7840
U7	MCP1525I
U8	A7840

Esquemático.



Apéndice B: caso4.m

El siguiente programa fue usado para validar dos distintos métodos de integración numérica. Ejecute el programa en Matlab.

```
% Programa para validación de algoritmo
% J. Nieto
disp('>>>>>>>>>>>>>>>>>>>INICIO CASO 4');
clearvars
format longEng
syms t f;
omega = 2.*pi.*f;
v1 = sqrt(2).*100.*sin(omega.*t);
v3 = sqrt(2).*20.*sin(3.*omega.*t - degtorad(70));
v5 = sqrt(2).*25.*sin(5.*omega.*t - degtorad(-140));
v7 = sqrt(2).*10.*sin(7.*omega.*t - degtorad(-210));
i1 = sqrt(2).*60.*sin(omega.*t - degtorad(30));
i3 = sqrt(2).*15.*sin(3.*omega.*t - degtorad(165));
i5 = sqrt(2).*12.*sin(5.*omega.*t - degtorad(-285));
i7 = sqrt(2).*10.*sin(7.*omega.*t - degtorad(-310));
v = v1 + v3 + v5 + v7;
i = i1 + i3 + i5 + i7;
p = v.*i;
P = f.*(int(p, t, 0, 1./f));
disp('-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-|-');
disp('P = ');
disp(vpa(P));
i = subs(i, f, 60);
v = subs(v, f, 60);
P_discreta_f = subs(p, f, 60);
t_simul = 0:1./(63.*60):1./60;
disp('v = ');
vpa(subs(v, t, t_simul))
disp('i = ');
vpa(subs(i, t, t_simul))
disp('P = ');
vpa(subs(P_discreta_f, t, t_simul))
Grafica_Rectangular = zeros(49,1);
Grafica_Simpson = zeros(49,1);
Grafica_P = zeros(49,1);
Grafica_Error_Rectangular = zeros(49,1);
Grafica_Error_Simpson = zeros(49,1);
Grafica_P(:) = vpa(P);
k = 1;
for MuestrasPorCiclo = 3:2:99
    X_Grafica_Convergencia(k) = MuestrasPorCiclo;
    ciclo = 0:1./(MuestrasPorCiclo.*60):1./60;
    P_discreta_t = subs(P_discreta_f, t, ciclo);
    P_discreta_rect = mean(P_discreta_t);
    Grafica_Rectangular(k) = P_discreta_rect;
    % calculo Simpson
    P_discreta_s = 0;

        for j = 1:2:MuestrasPorCiclo - 2
```

```
        P_discreta_s = P_discreta_s + P_discreta_t(j) + 4.*P_discreta_t(j+1) + P_discreta_t(j+2);
      end
   P_discreta_simpson = P_discreta_s./(MuestrasPorCiclo.*3);
   Grafica_Simpson(k) = P_discreta_simpson;
   P_discreta_rectangular_error = vpa(100.*abs(vpa(P_discreta_rect) - vpa(P))./vpa(P));
   Grafica_Error_Rectangular(k) = P_discreta_rectangular_error;
   P_discreta_simpson_error = vpa(100.*abs(vpa(P_discreta_simpson) - vpa(P))./vpa(P));
   Grafica_Error_Simpson(k) = P_discreta_simpson_error;
   k = k + 1;
   fprintf(['Numero de intervalos: %d | %%Error Rectangular: %f |' ...
      '%%Error Simpson: %f | P_rectangular: %f | P_Simpson: %f\n'], ...
      MuestrasPorCiclo, P_discreta_rectangular_error, P_discreta_simpson_error, ...
      vpa(P_discreta_rect), vpa(P_discreta_simpson));
   ciclo = 0;
   P_discreta_t = 0;
   P_discreta_rect = 0;
   P_discreta_s = 0;
   P_discreta_simpson = 0;
   P_discreta_rectangular_error = 0;
   P_discreta_simpson_error = 0;
end
   subplot(2,1,1);
   plot(X_Grafica_Convergencia, Grafica_Rectangular,'-.',...
      X_Grafica_Convergencia, Grafica_Simpson,'--', X_Grafica_Convergencia, Grafica_P,'-');
   xlabel('Numero de intervalos');
   ylabel('Valor de P');
   legend({'Rectangular', 'Simpson', 'Real'},'Location','southeast');
   legend('boxoff');
   grid on;
   subplot(2,1,2);
   plot(X_Grafica_Convergencia, Grafica_Error_Rectangular,'-.',...
      X_Grafica_Convergencia, Grafica_Error_Simpson,'--');
   xlabel('Numero de intervalos');
   ylabel('% Error');
   legend({'Rectangular', 'Simpson'},'Location','southeast');
   legend('boxoff');
   grid on;
clearvars
disp('>>>>>>>>>>>>>>>>>>>>FIN CASO 4');
%EOF
```

Apéndice C: Firmware

El firmware de este trabajo fue escrito en lenguaje C usando el compilar para PIC de CCS. Los siguientes son los archivos fuente que componen el programa.

adj.h

```
/*
*  adj.h -   Archivo cabecera que contiene constantes usadas en el programa y
             su calibración.
*            Modifique y compile según sea necesario.
*  Autor:   J. Nieto
*/
#ifndef ADJ_H_
```

```
#define ADJ_H_
/* Valor que se carga al registro CCP1 para generar la interrupción.
*/
#define  CCP1   5925
/*
* Tamaño del buffer de datos. El numero indica cuantos datos serán adquiridos
* en voltaje y corriente.
*/
#define   SIZE_DATA_ARRAY_BUFFER    45
#define   SIZE_DATA_ARRAY_BUFFER_F  45.0F
#endif /*ADJ_H_*/
/*END OF FILE: adj.h*/
```

c_buffer.c

```
/*
*  c_buffer.c -   Contiene la definicion del buffer circular, ademas de las
*                 funciones para manipularlo. Modifique segun sea necesario.
*  Autor:   J. Nieto
*/
typedef struct
{
   volatile size_t in;
   volatile size_t out;
   volatile int16_t v[SIZE_C_BUFFER_TMP];
   volatile int16_t i[SIZE_C_BUFFER_TMP];
   volatile int16_t ref[SIZE_C_BUFFER_TMP];
   volatile size_t free;
}
c_buffer_tmp_t;
/*
* tmp contiene los datos del adc.
*/
c_buffer_tmp_t tmp;

/*
* Definicion del buffer circular
*/
typedef struct
{
  volatile size_t in;
  volatile size_t out;
  volatile uint8_t buff[SIZE_C_BUFFER];
  volatile size_t free;
}
c_buffer_t;
/*
* buffers circulares que manejan el flujo de datos, son declarados como variables globales.
*/
c_buffer_t write, read;
/*
Esta función revisa si el buffer circular está vacío, devuelve true si está vacío, false si tiene información.
*/
inline bool
isempty_c (c_buffer_t bffr)
{
```

```
  if (SIZE_C_BUFFER == bffr.free)
  {
    return true;
  }
  else
  {
    return false;
  }
};
/*
Esta función revisa si el buffer circular tiene algún dato, devuelve true si tiene datos, false si está vacío.
*/
bool
hasdata_c (c_buffer_t bffr)
{
  if (SIZE_C_BUFFER > bffr.free)
  {
    return true;
  }
  else
  {
    return false;
  }
};
/*
Esta función revisa si el buffer circular está lleno, devuelve true si está lleno, false si no lo está.
*/
inline bool
isfull_c (c_buffer_t bffr)
{
  return  0 == bffr.free;
};
/*
Función que agrega un elemento al buffer circular, no devuelve valor.
*/
void
push_c (c_buffer_t *bffr, char ID)
{
  if (0 == bffr->free)
    {
      FATAL_ERROR ("c_buff lleno");
    }
  bffr->buff[bffr->in] = ID;
  bffr->in = ((bffr->in == (SIZE_C_BUFFER - 1)) ? 0 : bffr->in + 1);
  bffr->free--;
};
/*
Función que saca un elemento del buffer circular. Devuelve un puntero a una estructura d_buffer_t.
*/
d_buffer_t*
pop_c (c_buffer_t *bffr)
{
   char tmp_ID = 0;
   tmp_ID = bffr->buff[bffr->out];
   bffr->buff[bffr->out] = 0;
   bffr->out = ((bffr->out == (SIZE_C_BUFFER - 1)) ? 0 : bffr->out + 1);
```

```
   bffr->free++;
   switch (tmp_ID)
      {
         case 'A':
            return &A_;
            break;
         case 'B':
            return &B_;
            break;
         case 'C':
            return &C_;
            break;
         case 'D':
            return &D_;
            break;
         case 'E':
            return &E_;
            break;
         case 'F':
            return &F_;
            break;
         case 'G':
            return &G_;
            break;
         case 'H':
            return &H_;
            break;
         case 'I':
            return &I_;
            break;
         case 'J':
            return &J_;
            break;
         default:
            FATAL_ERROR("switch error");
            break;
      }
   /*
   * No debería estar aquí.
   */
   return 0;
};
/*
Esta función vacía el buffer circular, la cabeza y cola apuntan al mismo punto, y el arreglo es rellenado con
ceros.
*/
void
clr_c (c_buffer_t *bffr)
{
   size_t i;
   bffr->in = 0;
   bffr->out = 0;
   bffr->free = SIZE_C_BUFFER;
   for ( i = 0; i < SIZE_C_BUFFER; i++)
     {
        bffr->buff[i] = 0;
```

```
    }

};
void
init_c (void)
{
   size_t i;
  /*
  * vacía los buffers circulares
  */
  clr_c (&write);
  clr_c (&read);
  /* Limpia y vacía tmp*/
  tmp.in = 0;
  tmp.out = 0;
  tmp.free = SIZE_C_BUFFER_TMP;
  for (i = 0; i < SIZE_C_BUFFER_TMP; i++)
  {
     tmp.v[i] = 0;
     tmp.i[i] = 0;
     tmp.ref[i] = 0;
  }
  /*
  * carga los punteros de los buffers de datos al buffer circular "escritura"
  */
  push_c (&write, A_.ID);
  push_c (&write, B_.ID);
  push_c (&write, C_.ID);
  push_c (&write, D_.ID);
  push_c (&write, E_.ID);
  push_c (&write, F_.ID);
  push_c (&write, G_.ID);
  push_c (&write, H_.ID);
  push_c (&write, I_.ID);
  push_c (&write, J_.ID);
};
/*END OF FILE: c_buffer.c*/
```

const.h

```
/*
*  const.h -   Archivo cabecera que contiene constantes usadas en el programa.
*              Modifique según sea necesario.
*  Autor:   J. Nieto
*/
#ifndef CONST_H_
#define CONST_H_
#include "adj.h"
/*
* Tamaño del buffer circular global, es múltiplo de SIZE_DATA_ARRAY_BUFFER
*/
#define  SIZE_C_BUFFER_TMP 650
/*
* Tamaño buffer circular.
*/
#define SIZE_C_BUFFER 10
```

```
/*
* Canales ADC a usar
* Canal de Corriente CH_I
* Canal de Voltaje   CH_V
* Canal de Referencia CH_REF
*/
enum { CH_I = 3   \
     ,CH_V = 1   \
     ,CH_REF = 0 \
     };
/*
* Constantes para el cálculo del voltaje y corriente
* K_I = 125.0 / 4096.0;
* K_V = 10000.0 / 32768.0;
* K_VI = K_I * K_V
*/
//#define K_V      250.125F
//#define K_I      25.0F
//#define K_VI   6253.125F
//#define K_ADC   0.001221F
//#define K_ADC2   0.000001490841F
/*
*
* K_V_ADC = K_V * K_ADC
* K_I_ADC = K_I * K_ADC
* K_VI_ADC2 = K_V * K_I * K_ADC^2
*
*/
#ifdef FIRMWARE

#define  K_V_ADC   0.305254F
#define  K_I_ADC   0.031266F
#define  K_VI_ADC2 0.009322415128125F
#else
#define  K_V_ADC   0.305402625F
#define  K_I_ADC   0.030525F
#define  K_VI_ADC2 0.009322415128125F
#endif
// K_KWH = 60 * 3600 * 1000
// K_WH = 60 * 3600
// K_WS = 60
#define  K_KWH 216000000.0F
#define  K_WH  216000.0F
//#define  K_WS  60.0F
/*
* C_25V es la constante que representa el valor teórico de 2.5V usando un convertidor de 12bits y un rango
de adquisición de 0 a 5 V
*/
#define C_25V   2047.0F
#endif /*CONST_H_*/
/*END OF FILE: const.c*/
```

d_buffer.c

```
/*
*  d_buffer.c -   Contiene la definición del buffer de datos, además de las
```

```
*               funciones para manipular su acceso.
*               Modifique según sea necesario.
*  Autor:   J. Nieto
*/
/*
* Definición del buffer de datos
*/
typedef struct
{
  volatile int16_t v[SIZE_DATA_ARRAY_BUFFER];
  volatile int16_t i[SIZE_DATA_ARRAY_BUFFER];
  volatile int16_t ref[SIZE_DATA_ARRAY_BUFFER];
  volatile bool busy;
       char ID;
}
d_buffer_t;
/*
* Se definen 6 buffers para almacenar y procesar datos del ADC, son del tipo volatile ya que puede cambiar
su valor en cualquier parte del programa.
*/
d_buffer_t A_;
d_buffer_t B_;
d_buffer_t C_;
d_buffer_t D_;
d_buffer_t E_;
d_buffer_t F_;
d_buffer_t G_;
d_buffer_t H_;
d_buffer_t I_;
d_buffer_t J_;
inline
void
init_d (void)
{
  //desbloquea los buffers de datos
  A_.busy = false;
  A_.ID = 'A';
  B_.busy = false;
  B_.ID = 'B';
  C_.busy = false;
  C_.ID = 'C';
  D_.busy = false;
  D_.ID = 'D';
  E_.busy = false;
  E_.ID = 'E';
  F_.busy = false;
  F_.ID = 'F';
  G_.busy = false;
  G_.ID = 'G';
  H_.busy = false;
  H_.ID = 'H';
  I_.busy = false;
  I_.ID = 'I';
  J_.busy = false;
  J_.ID = 'J';
};
```

```
/*END OF FILE: d_buffer.c*/
```

f_result.c

```
/*
*  f_result.c -   Contiene la definición de estructura de datos contenedora
*                 de resultados finales y sus funciones para acceso y modificación.
*                 Modifique según sea necesario.
*  Autor:   J. Nieto
*/
typedef struct
{
  float rms_v;
  float rms_i;
  float energia;
  float p;
  bool renew;
}
f_result_t;
f_result_t final_result;
inline void
clr_f (void)
{
  final_result.energia = 0.0;
  final_result.rms_i = 0.0;
  final_result.rms_v = 0.0;
  final_result.p = 0.0;
  final_result.renew = false;
};
void
set_rms_v (float rms_v)
{
  final_result.rms_v = rms_v;
  final_result.renew = true;
};
void
set_rms_i (float rms_i)
{
  final_result.rms_i = rms_i;
  final_result.renew = true;
};
void
set_energia (float energia)
{
  final_result.energia = energia;
  final_result.renew = true;
};
void
set_p (float p)
{
  final_result.p = p;
  final_result.renew = true;
};
inline float
get_rms_v (void)
{
```

```
  return final_result.rms_v;
};
inline float
get_rms_i (void)
{
  return final_result.rms_i;
};
inline float
get_energia (void)
{
  return final_result.energia;
};
inline float
get_p (void)
{
  return final_result.p;
};
bool
get_renew (void)
{
  return final_result.renew;
};
void
put_renew (bool renew)
{
  final_result.renew = renew;
};
/*END OF FILE: f_result.c*/
```

fatal.c

```
/*
*  fatal.c -   Contiene función de depuración y definiciones para su
*              funcionamiento. Modifique según sea necesario.
*  Autor:   J. Nieto
*/
volatile int8_t tsk;
#ifdef   FIRMWARE
volatile char filename[16];
volatile char line;
volatile char msj[16];
#define  FATAL_ERROR(err)  (filename = __FILENAME__,  \
                    line = __LINE__,          \
                    msj = err,                \
                    goto_address (0x1FF00))
/*
* Esta función es llamada para mostrar un error fatal y donde ocurrió.
* 0x1FF00 es una localidad valida de memoria flash de programa,
* la función puede ocupar de espacio el rango de 0x1FF00 a 0x1FFFF.
* 0x1FFFF es la localidad límite de la memoria de programa
* para el modelo de microcontrolador que se está usando (18f27k42).
*/
#ORG  0x1FF00, 0x1FFFF
void
fatalerror (void)
{
```

```
  disable_interrupts (INT_CCP1);
  printf (lcd_putc, "\a\f%s:%d\n%s %i", filename, line, msj, tsk);
  //después de mostrar el error, el microcontrolador duerme
  sleep (SLEEP_FULL);
}
#else
#define  FATAL_ERROR(err)  printf("\a\f%s:%d\n%s", __FILE__, __LINE__, err)
#endif
/*END OF FILE: fatal.c*/
```

lcd.c

En la versión del compilador CCS 5.091 el driver incluido para manejar el módulo LCD (lcd.c) no maneja correctamente el LCD de veinte caracteres por cuatro líneas. Para que este driver funcione con este tipo de LCD, se deberá hacer la siguiente modificación.

Buscar las siguientes líneas dentro de la función *void lcd_gotoxy(unsigned int8, unsigned int8)*, ubicada en el archivo *lcd.c* incluido en el compilador:

```
  if(y!=1)
    address=LCD_LINE_TWO;
  else
    address=0;
```

Sustituir por:

```
  switch(y)
  {
   case 1:
      address = 0x00;
      break;
  #if defined(LCD_EXTENDED_NEWLINE)
   case 2:
      address = 0x40;
      break;
   case 3:
      address = 0x14;
      break;
   case 4:
      address = 0x54;
      break;
  #endif
   default:
      address = LCD_LINE_TWO;
      break;
  }
```

local.h

```
/*
*  local.h -   Archivo cabecera que contiene definiciones y otras librerías
*              usadas en el programa.
*              Modifique según sea necesario.
*  Autor:   J. Nieto
*/
#ifndef LOCAL_H_
#define LOCAL_H_
/*
* Compilar firmware
*/
#define FIRMWARE
/*
* Método de integración del rectángulo
*/
#define RMS_RECT
#define P_RECT
//#define RMS_RECT_OVERFLOW_CHECK

/*
* Método de integración por Simpson
*/
//#define RMS_SIMPSON
//#define P_SIMPSON
//#define RMS_SIMPSON_OVERFLOW_CHECK
/*
* Definiciones para el uso del LCD usando lcd.c
*/
// Necesario para 20 x 4
#define LCD_EXTENDED_NEWLINE
// Definición de terminales para LCD
#define  LCD_RS_PIN     PIN_B1
#define  LCD_RW_PIN     PIN_B2
#define  LCD_ENABLE_PIN PIN_B3
#define  LCD_DATA4      PIN_B4
#define  LCD_DATA5      PIN_B5
#define  LCD_DATA6      PIN_B6
#define  LCD_DATA7      PIN_B7
/*
* Carácter sigma
*/
#define SIGMA 0xF6
/*
* Terminal de depuración
*/
#define  PIN_DBG  PIN_B0
#byte ADPREH = getenv ("sfr:ADPREH")
#byte ADPREL = getenv ("sfr:ADPREL")
#byte ADACQH = getenv ("sfr:ADACQH")
#byte ADACQL = getenv ("sfr:ADACQL")
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
```

```
#include <math.h>
#include <float.h>
#include "lcd.c"
#include "const.h"
#include "fatal.c"
#include "d_buffer.c"
#include "c_buffer.c"
#include "f_result.c"
#include "tarea0.c"
#include "tarea1.c"
#include "tarea2.c"
#include "tarea3.c"
#endif /*LOCAL_H_*/
/*END OF FILE: local.h*/
```

main.c

```
/*
*  Watto - Firmware del proyecto de tesis Watto. Obtiene la energía eléctrica consumida.
*  Autor:   J. Nieto
*/
#include <18f27k42.h>
#device  ADC=12
#device  HIGH_INTS=TRUE
#device  ICD=TRUE
#fuses   MCLR
#fuses   HS
#fuses   RSTOSC_EXT_PLL
#fuses   NOWDT
#fuses   NOLVP
#fuses   NOPROTECT
#OCS   64 MHz
#include "local.h"
int
main (void)
{
  setup_oscillator (OSC_EXTOSC_ENABLED | OSC_EXTOSC_PLL | OSC_CLK_DIV_BY_1,  \
                    OSC_PLL_READY);
  set_analog_pins (PIN_A0, PIN_A1, PIN_A2, PIN_A3);
  setup_adc_ports (sAN0 | sAN1 | sAN3, VREF_VDD);
  setup_adc (ADC_CLOCK_DIV_32);
  ADACQH = 0x00;
  ADACQL = 0x00;
  ADPREH = 0x00;
  ADPREL = 0x00;
  /*
  * Inicializando las estructuras de almacenamiento de datos
  */
  init_d ();
  /*
  * Inicializando las estructuras de control
  */
  init_c ();
  /*
  * Limpiando la estructura de resultado final
  */
```

```
  clr_f ();
  /*
  * inicializando el LCD
  */
  lcd_init ();
  /*
  * Configura el Timmer 1 con Fosc/4
  */
  setup_timer_1 (T1_INTERNAL | T1_DIV_BY_1);
  /*
  * inicializa con 0
  */
  set_timer1 (0);
  /*
  * Configura el Comparador 1
  */
  setup_ccp1 (CCP_COMPARE_PULSE_RESET_TIMER | CCP_USE_TIMER1_AND_TIMER2);
  /*
  * inicializa valor del CCP1
  */
  CCP_1 = CCP1;
  /*
  * activa interrupciones
  */
  enable_interrupts (INT_CCP1);
  enable_interrupts (GLOBAL);
  for (;;)
    {
    tarea1 ();
    tarea2 ();
    tarea3 ();
    }
  return 0;
}
/*END OF FILE: main.c*/
```

tarea0.c

```
/*
*  tarea0.c - Contiene la función tarea0, encargada de la adquisición
*             de datos de las señales Voltaje y corriente. Es invocada cada
*             vez que la interrupción del comparador CCP1 es activada.
*             Modifique según sea necesario.
*  Autor:   J. Nieto
*/
/*
* NOTA IMPORTANTE:
* El program counter (PC) es guardado en el stack dedicado PC del
* microcontrolador. Los registros del CPU son guardados incluyendo STATUS,
* WREG, BSR, FSR0/1/2, PRODL/H y PCLATH/U, tambien por el microcontrolador.
* El compilador genera codigo para salvar TABLAT y TBLPTRL/H/U.
* Si se desea mantener compatibilidad con otros modelos de microcontroladores
* debera escribir la rutina para salvar y restaurar el contexto.
*/
#ifdef   FIRMWARE
#INT_CCP1 HIGH
```

```
#endif
void
tarea0 (void)
{
  int16_t v_tmp;
  int16_t i_tmp;
  int16_t ref_tmp;
  int8_t tsk_tmp;
  /*
  * Variable para depuración. Contiene la última tarea ejecutada
  */
  tsk_tmp = tsk;
  tsk = 0;
#ifdef   FIRMWARE
  output_toggle(PIN_DBG);
#endif
  /*
  * Revisa si el buffer circular tmp está lleno,
  */
  if (0 == tmp.free)
  {
    /*
    * Entonces el buffer está lleno y es un error fatal
    */
    FATAL_ERROR("tmp lleno");
  }
  v_tmp = 0;
  i_tmp = 0;
#ifdef   FIRMWARE
  for (size_t z = 0; z < 8; z++)
  {
    /*
    *  Se configura el canal del ADC
    */
    set_adc_channel (CH_V);
    /*
    *  Se obtiene la lectura y la guarda en la estructura adc_v
    */
    v_tmp += read_adc ();

    /*
    * Configura canal para adquisicion de señal corriente
    * Obtiene dato y lo guarda en el buffer de datos
    */
    set_adc_channel (CH_I);
    i_tmp += read_adc ();
  }
  v_tmp >>= 3;
  i_tmp >>= 3;
#else
  set_adc_channel (CH_V);
  v_tmp = read_adc ();
  set_adc_channel (CH_I);
  i_tmp = read_adc ();
#endif
  /*
```

```
  * Configura canal para adquisición de señal referencia
  * Obtiene dato y lo va acumulando.
  */
  set_adc_channel (CH_REF);
  ref_tmp = read_adc ();
  /*
  * Envía los valores de las mediciones al buffer temporal tmp
  */
  tmp.v[tmp.in] = v_tmp;
  tmp.i[tmp.in] = i_tmp;
  tmp.ref[tmp.in] = ref_tmp;
  /*
  * Reconfigura el índice que va marcando el inicio del buffer
  */
  tmp.in = ((tmp.in == (SIZE_C_BUFFER_TMP - 1)) ? 0 : tmp.in + 1);
  tmp.free--;
#ifdef   FIRMWARE
  output_toggle (PIN_DBG);
#endif
  tsk = tsk_tmp;
};
/*END OF FILE: tarea0.c*/
```

tarea1.c

```
/*
*  tarea1.c - Contiene la función tarea1, encargada de mover los datos
*           contenidos en el buffer tmp a las estructuras que
*           contendrán los datos para el posterior calculo.
*  Autor:   J. Nieto
*/
void
tarea1 (void)
{
   static d_buffer_t* writing = 0;
   static size_t count_data = 0;
   tsk = 1;
  /*
  * Si -tmp- tiene por lo menos 1
  */
  if ((SIZE_C_BUFFER_TMP - 2) >= tmp.free)
  {
      while ((SIZE_C_BUFFER_TMP - 1) > tmp.free)
      {
      /*
      * Si el contador count_data es igual a 0, significa que se inicia con un
      * nuevo buffer a escribir datos, por tal motivo, se inicia la configuración de los punteros.
      */
      if (0 == count_data)
      {
         /*
         * Si el buffer circular write está vacío, y tmp está lleno
         * significa que no hay lugar donde escribir los nuevos datos
         * que provienen del ADC, por tal motivo, constituye un error fatal.
         * Pero si write está vacío y tmp tiene cupo, no hay nada que hacer
         * sólo esperar a que se desahogue el buffer read.
```

```
   */
   if (isempty_c(write) == true)
   {
     if (0 == tmp.free)
     {
       FATAL_ERROR("wr\tmp vacio");
     }
     else
     {
     /*
     * Si el buffer write esta vacío y el buffer tmp aún tiene cupo
     * no constituye un error, ya que aún hay lugar donde almacenar datos.
     */
       return;
     }
   }
   /*
   * Se obtiene un puntero al nuevo buffer que se usara para escribir datos.
   */
   writing = pop_c(&write);
   /*
   * Si el buffer de datos está bloqueado, significa que otro proceso
   * lo está utilizando y esto constituye un error fatal.
   */
   if (writing->busy == true)
   {
      FATAL_ERROR("writing bloqu");
   }
   /*
   * Se bloquea el nuevo buffer de datos, ya que se estará ocupando.
   */
   writing->busy = true;
}
writing->v[count_data] = tmp.v[tmp.out];
writing->i[count_data] = tmp.i[tmp.out];
writing->ref[count_data] = tmp.ref[tmp.out];
tmp.v[tmp.out] = 0;
tmp.i[tmp.out] = 0;
tmp.ref[tmp.out] = 0;
tmp.out = ((tmp.out == (SIZE_C_BUFFER_TMP - 1)) ? 0 : tmp.out + 1);
tmp.free++;
/*
* Incrementa el índice de control.
*/
count_data++;
/*
* Si cout_data = SIZE_DATA_ARRAY_BUFFER, significa que el buffer de
* datos ha sido escrito en su totalidad.
*/
if (SIZE_DATA_ARRAY_BUFFER == count_data)
{
   /*
   * Revisa que el buffer lectura tenga espacio, si está lleno es un error fatal.
   */
   if (isfull_c(read))
   {
```

```
            FATAL_ERROR("read lleno");
          }
          /*
          * Libera el buffer escribir de su bloqueo, ya que se ha terminado de usarlo.
          */
          writing->busy = false;
          /*
          * Reinicia el contador a 0
          */
          count_data = 0;
          /*
          * Envía escribir al buffer lectura para ser procesado.
          */
          push_c(&read, writing->ID);
          }
       }
   }
};
/*END OF FILE: tarea1.c*/
```

tarea2.c

```
/*
*  tarea2.c - Contiene la función tarea2, encargada del cálculo de la energía eléctrica.
*             Modifique según sea necesario.
*  Autor:   J. Nieto
*/
void
tarea2 (void)
{
   d_buffer_t* reading;
   size_t j;
   int32_t v_acc;
   int32_t i_acc;
   int32_t v_dc;
   int32_t i_dc;
   float rms_v;
   float rms_i;
  float data_v_1;
  float data_i_1;
   float data_v_2, data_v_3;
   float data_i_2, data_i_3;
   float ECSUB;
   float Pot;
   static float ECA = 0.0;
   static float ECA_s = 0.0;
   static size_t count_s = 0;
   static bool rms = false;
   static size_t calc_rms= 0;
   tsk = 2;
   /*
   * Si esta vacío -read- no hay nada que hacer.
   */
  if (true == isempty_c (read))
  {
    return;
```

```
  }
  /*
  * Obtiene el puntero al buffer de datos listo para procesar
  */
  reading = pop_c (&read);
  //comprueba que no esté bloqueado
  //ya que, si lo está, constituye un error en el manejo de datos
  if (reading->busy == true)
    {
     //si lo está, es un error fatal
      FATAL_ERROR ("leer bloqu");
    }
  //bloquea el buffer para que no lo consulte ninguna otra tarea reading->busy = true;
  /*
  * Calcula la componente de DC de las señales tensión, corriente y referencia
  */
  v_acc = 0.0;
  i_acc = 0.0;
  for (j = 0; j < SIZE_DATA_ARRAY_BUFFER; j++)
  {
    v_acc += reading->v[j];
    i_acc += reading->i[j];
  }
  v_dc = v_acc / SIZE_DATA_ARRAY_BUFFER;
  i_dc = i_acc / SIZE_DATA_ARRAY_BUFFER;
  ECSUB = 0.0;
#ifdef P_RECT
  for (j = 0; j < SIZE_DATA_ARRAY_BUFFER; j++)
  {
    data_v_1 = (float) (reading->v[j] - v_dc);
    data_i_1 = (float) (reading->i[j] - i_dc);
    ECSUB += data_v_1 * data_i_1;
  }
  Pot = (K_VI_ADC2 * ECSUB) / SIZE_DATA_ARRAY_BUFFER_F;
#endif
#ifdef P_SIMPSON
  for (j = 0; j < (SIZE_DATA_ARRAY_BUFFER - 2); j+=2)
    {
      data_v_1 = (float) (reading->v[j] - v_dc);
      data_v_2 = (float) (reading->v[j+1] - v_dc);
      data_v_3 = (float) (reading->v[j+2] - v_dc);
      data_i_1 = (float) (reading->i[j] - i_dc);
      data_i_2 = (float) (reading->i[j+1] - i_dc);
      data_i_3 = (float) (reading->i[j+2] - i_dc);
      ECSUB += data_v_1 * data_i_1 + 4.0F * data_v_2 * data_i_2 + data_v_3 * data_i_3;
    }
  Pot = (K_VI_ADC2 * ECSUB) / (3.0F * SIZE_DATA_ARRAY_BUFFER_F);
#endif
#ifndef FIRMWARE
  set_p (Pot);
#endif
  ECA += Pot;
  count_s++;
  /*
  * Si la cuenta de acumulados ha llegado a 60 es hora de refrescar el dato de energía y potencia
  */
```

```
#ifdef   FIRMWARE
   if (60 == count_s)
   {
      /*
      * wattsegundo -> kilowatt-hora
      */
      //ECA_s += ECA / K_KWH;
       /*
       * wattsegundo -> watt-hora
       */
       ECA_s += ECA / K_WH;
      set_p (Pot);
      set_energia (ECA_s);
      ECA = 0.0;
      count_s = 0;

      if (2 == calc_rms)
      {
         rms = true;
         calc_rms = 0;
      }
      else
      {
         calc_rms++;
         rms = false;
      }
   }
#else
   rms = true;
#endif
  /*
  * El cálculo de los valores RMS se hace cada 2 segundos, ya que si el
  * calculo es continuo, el tiempo que tarda en hacerlo rebasa la velocidad
  * con la que los buffers -tmp- o -write- se vacían, obteniendo así un error
  * de desbordamiento. Por tal motivo, se hace cada que cal_rms es 2.
  */
  if (true == rms)
  {
    rms = false;
    /*
    * calcula valor RMS de la señal voltaje y corriente, se corrigen los datos
    * restando el valor DC que se obtuvo anteriormente, además de realizar la
    * multiplicación punto a punto de voltaje y corriente
    */
    rms_v = 0.0;
    rms_i = 0.0;
#ifdef RMS_RECT
    for (j = 0; j < SIZE_DATA_ARRAY_BUFFER; j++)
    {
      rms_v += pow((float) (reading->v[j] - v_dc), 2);
      rms_i += pow((float) (reading->i[j] - i_dc), 2);
  #ifdef RMS_RECT_OVERFLOW_CHECK
       /*
       * Revisión de ovarlo.
       * valores RMS siempre positivos.
       */
```

```
      if (FLT_MAX <= rms_v || FLT_MAX <= rms_i)
      {
        /*
        * Si hay una falla es un error fatal y por el momento no hay forma de tratarlo.
        */
        FATAL_ERROR("overflow rms_v/i");
      }
      if (0.0F > rms_v || 0.0F > rms_i)
      {
        /*
        * Si hay una falla es un error fatal y por el momento no hay forma de tratarlo.
        */
        FATAL_ERROR("corrupción rms_v/i");
      }
  #endif
    }
    //printf(lcd_putc,"\frms %f", rms_v);
    rms_v /= SIZE_DATA_ARRAY_BUFFER;
    rms_i /= SIZE_DATA_ARRAY_BUFFER;
    //printf(lcd_putc,"\n div %f", rms_v);
#endif
#ifdef RMS_SIMPSON
    for (j = 0; j < (SIZE_DATA_ARRAY_BUFFER - 2); j+=2)
    {
      data_v_1 = (float) (reading->v[j] - v_dc);
      data_v_2 = (float) (reading->v[j+1] - v_dc);
      data_v_3 = (float) (reading->v[j+2] - v_dc);
      data_i_1 = (float) (reading->i[j] - i_dc);
      data_i_2 = (float) (reading->i[j+1] - i_dc);
      data_i_3 = (float) (reading->i[j+2] - i_dc);
      /*
      * función pow en otros compiladores regresa un double.
      * En CCS los double no están definidos. Si se compila este programa
      * con un compilador diferente, donde pow regrese un double, tendrá
      * un aviso de warning sobre perdida de información. Para corregirlo
      * deberá definir las variables y los cast necesarios como double.
      */
      rms_v += pow (data_v_1, 2) + 4.0F * pow (data_v_2, 2) + pow (data_v_3, 2);
      rms_i += pow (data_i_1, 2) + 4.0F * pow (data_i_2, 2) + pow (data_i_3, 2);
  #ifdef RMS_SIMPSON_OVERFLOW_CHECK
      /*
      * Revisión de overflow. valores RMS siempre positivos.
      */
      if (FLT_MAX <= rms_v || FLT_MAX <= rms_i)
      {
        /*
        * Si hay una falla es un error fatal y por el momento no hay forma de tratarlo.
        */
        FATAL_ERROR("overflw rms_v/i");
      }
      if (0.0F > rms_v || 0.0F > rms_i)
      {
        /*
        * Si hay una falla es un error fatal y por el momento no hay forma
        * de tratarlo.
        */
```

```
          FATAL_ERROR("corrupción rms_v/i");
        }
  #endif
    }
    //printf(lcd_putc,"\frms %f", rms_v);
    rms_v /= ( 3.0F * SIZE_DATA_ARRAY_BUFFER_F);
    rms_i /= ( 3.0F * SIZE_DATA_ARRAY_BUFFER_F);
    //printf(lcd_putc,"\n div %f", rms_v);
#endif
    rms_v = sqrt(rms_v);
    rms_i = sqrt(rms_i);
    /*
    * Envía datos para impresión
    */
    set_rms_v (K_V_ADC * rms_v);
    set_rms_i (K_I_ADC * rms_i);
  }
  /*
  * Desbloquea el buffer de datos, al cual apunta -leer-
  */
  reading->busy = false;
  /*
  * revisa si el buffer escritura está lleno, si lo esta es un error fatal
  * ya que no hay donde poner el buffer liberado
  */
  if (isfull_c (write))
    {
      FATAL_ERROR(">c buff lleno!<");
    }
  push_c (&write, reading->ID);
};
/*END OF FILE: tarea2.c*/
```

tarea3.c

```
/*
*  tarea3.c - Contiene la función tarea3, encargada de imprimir los resultados de la medición en el LCD.
*             Modifique según sea necesario.
*  Autor:   J. Nieto
*/
inline void
tarea3 (void)
{
  tsk = 3;
  /*
  * Imprime datos si hay alguna renovación de los mismos
  */
  if (true == get_renew ())
  {
#ifdef FIRMWARE
      printf(lcd_putc, "\fV= %3.2f V\nI= %3.2f A\nP= %3.2f W\n%c= %3.2f Wh",   \
      get_rms_v(), get_rms_i(), get_p(), SIGMA, get_energia());
#else
      printf("V= %f\tV\nI= %f\tA\nP= %f\tW\n%c= %f\tWh",    \
      get_rms_v(), get_rms_i(), get_p(), SIGMA, get_energia());
#endif
```

```
    /*
    * Como los valores ya se han impreso, se pone la variable renew como
    * false.
    */
    put_renew (false);
  }
};
/*END OF FILE: tarea3.c*/
```

Apéndice D: Programa de prueba

El siguiente programa fue hecho con dos objetivos; el primero es cerciorarse que el firmware funcione emulando interrupciones, el segundo es compilar la mayoría del código fuente del firmware en busca de algún error que el compilador CCS pase por alto. Este código fue compilado bajo Visual Studio 2019, pero debería compilarse en cualquier otro compilador de C/C++.

test.c

```
/*
*           test.c - Programa prueba del firmware watto
*             Modifique segun sea necesario.
*  Autor:   J. Nieto
*/
#define _USE_MATH_DEFINES
/*
#define RMS_RECT
#define P_RECT
*/
#define RMS_SIMPSON
#define P_SIMPSON
#define SIGMA 0x53
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <stddef.h>
#include <math.h>
#include "float.h"
#include "..\const.h"
#include "..\fatal.c"
#include "gen_data.c"
#include "simul_adc.c"
#include "..\d_buffer.c"
#include "..\c_buffer.c"
#include "..\f_result.c"
#include "..\tarea0.c"
#include "..\tarea1.c"
#include "..\tarea2.c"
#include "..\tarea3.c"
#include "show_data.c"
int
main (void)
```

```
{
	size_t i;
	// genera vectores de datos para la simulación del ADC
	gendata();
	// inicializa las estructuras de datos a usar.
	init_d ();
	init_c ();
	clr_f ();
	printf ("1 - estructuras de datos inicializadas\n");
	show_c ();
	show_d ();
	show_tmp ();
	printf ("2 - Se inicia la simulación de %i interrupciones\n", SIZE_DATA_ARRAY_BUFFER + 26);
	for (i = 0; i < SIZE_DATA_ARRAY_BUFFER + 26 ; i++)
		{
			tarea0 ();
		}
	printf("3 - Se muestran las estructuras de datos después de llamada a interrupción\n");
	show_c ();
	show_d ();
	show_tmp ();
	printf("4 - Se llama a la función de transferencia de datos entre buffers\n");
	tarea1 ();
	show_d ();
	show_c ();
	show_tmp ();
	printf("5 - Se llama a la función de cálculo de la energía\n");
	tarea2 ();
	show_d ();
	show_c ();
	show_tmp ();
	printf("6 - Se imprimen datos finales\n");
	tarea3 ();
	return 0;
}
/*END OF FILE: test.c*/
```

simul_adc.c

```
/*
*
*  simul_adc.c - Contiene las funciones que simulan el ADC.
*                  necesarias para el programa prueba test.cpp
*           Modifique según sea necesario.
*  Autor:   J. Nieto
*/
uint8_t channel;
int32_t
read_adc (void)
{
	static size_t v_index = 0, i_index = 0;
	int32_t res;
	switch (channel)
		{
			case CH_V:
				if (SIZE_DATA_ARRAY_BUFFER == v_index)
```

```
                                        {
                                                v_index = 0;
                                        }
                                        res = v_caso4_data[v_index];
                                        v_index++;
                                        break;
                                case CH_I:
                                        if (SIZE_DATA_ARRAY_BUFFER == i_index)
                                        {
                                                i_index = 0;
                                        }
                                        res = i_caso4_data[i_index];
                                        i_index++;
                                        break;
                                case CH_REF:
                                        res = 0x07FF;
                                        break;
                                default:
                                        FATAL_ERROR ("Numero no valido en switch");
                                        break;
                        }
                return res;
};
void
set_adc_channel (uint8_t ch)
{
                channel = ch;
};
/*END OF FILE: simul_adc.c*/
```

show_data.c

```
/*
*  show_data.c - Contiene las funciones para mostrar datos y estatus de las
*                        estructuras de datos y de control, necesarias para el
*                        programa prueba test.cpp
*              Modifique segun sea necesario.
*  Autor:   J. Nieto
*/
void
show_c (void)
{
                size_t i;
                printf("\n[write]\nfree:%i\n{", write.free);
                for (i = 0; i < SIZE_C_BUFFER; i++)
                        {
                        if (i == write.in)
                                {
                                        printf ("(in)->");
                                }
                        if (i == write.out)
                                {
                                        printf ("(out)->");
                                }
                                printf ("[%c]\t", write.buff[i]);
                        }
```

```
		printf ("}\n");
		printf ("[read]\nfree:%i\n{", read.free);
		for (i = 0; i < SIZE_C_BUFFER; i++)
			{
				if (i == read.in)
					{
						printf("(in)->");
					}
				if (i == read.out)
					{
						printf("(out)->");
					}
				printf ("[%c]\t", read.buff[i]);
			}
		printf("}\n");
};
void
show_d(void)
{
		size_t i;
		printf ("\n[%c]\t\t[%c]\t\t[%c]\t\t[%c]\t\t[%c]\t\t[%c]\n{", A.ID, B.ID, C.ID, D.ID, E.ID, F.ID);
		for (i = 0; i < SIZE_DATA_ARRAY_BUFFER; i++)
		{
			printf	("V-[%X] I-[%X] REF-[%X] BUSY-[%X]\t\
					V-[%X] I-[%X] REF-[%X] BUSY-[%X]\t\
					V-[%X] I-[%X] REF-[%X] BUSY-[%X]\t\
					V-[%X] I-[%X] REF-[%X] BUSY-[%X]\t\
					V-[%X] I-[%X] REF-[%X] BUSY-[%X]\n",\
					A.v[i], A.i[i], A.ref[i], A.busy,\
					B.v[i], B.i[i], B.ref[i], B.busy,\
					C.v[i], C.i[i], C.ref[i], C.busy,\
					D.v[i], D.i[i], D.ref[i], D.busy,\
					E.v[i], E.i[i], E.ref[i], E.busy,\
					F.v[i], F.i[i], F.ref[i], F.busy\
					);
		}
		printf("}\n");
};
void
show_tmp(void)
{
		size_t i;
		printf("TMP {\n");
		for (i = 0; i < SIZE_C_BUFFER_TMP; i++)
		{
			printf("V-[%X]\tI-[%X]\tREF-[%X]\t\n", tmp.v[i], tmp.i[i], tmp.ref[i]);
		}
		printf("}\n");
};
/*END OF FILE: show_data.c*/
```

gen_data.c

```
/*
*  gen_data.c - Contiene las funciones para generar dos vectores de datos
```

```
*                       para la simulación del ADC.
*             Modifique según sea necesario.
*  Autor:   J. Nieto
*/
double
degtorad(double degree)
{
        return degree * (M_PI / 180.0);
};
double
v_caso4 (double t)
{
        double v1, v3, v5, v7;
        v1 = sqrt(2) * 100 * sin(2 * M_PI * 60 * t);
        v3 = sqrt(2) * 20 * sin(3 * 2 * M_PI * 60 * t - degtorad(70));
        v5 = sqrt(2) * 25 * sin(5 * 2 * M_PI * 60 * t - degtorad(-140));
        v7 = sqrt(2) * 10 * sin(7 * 2 * M_PI * 60 * t - degtorad(-210));
        return (v1 + v3 + v5 + v7);
};
double
i_caso4 (double t)
{
        double i1, i3, i5, i7;
        i1 = sqrt(2) * 60 * sin(2 * M_PI * 60 * t - degtorad(30));
        i3 = sqrt(2) * 15 * sin(3 * 2 * M_PI * 60 * t - degtorad(165));
        i5 = sqrt(2) * 12 * sin(5 * 2 * M_PI * 60 * t - degtorad(-285));
        i7 = sqrt(2) * 10 * sin(7 * 2 * M_PI * 60 * t - degtorad(-310));
        return (i1 + i3 + i5 + i7);
};
int32_t v_caso4_data[SIZE_DATA_ARRAY_BUFFER];
int32_t i_caso4_data[SIZE_DATA_ARRAY_BUFFER];
void
gendata (void)
{
        double t[SIZE_DATA_ARRAY_BUFFER];
        double k = 0.0;
        // Genera vector de tiempo t
        printf("Generación de vector tiempo t\n");
        for (size_t j = 0; j < SIZE_DATA_ARRAY_BUFFER; j++)
        {
                t[j] = k;
                k += 1 / (60.0 * SIZE_DATA_ARRAY_BUFFER_F);
                printf("[%i] t - [%lf]\n",j ,t[j]);
        }
        // evalua las funciones v i con los datos del vector tiempo t
        printf("Generación de vector de datos\n");
        for (size_t j = 0; j < SIZE_DATA_ARRAY_BUFFER; j++)
        {
                v_caso4_data[j] = (int32_t) ( v_caso4 ( t[j] ) / (K_V_ADC) + C_25V);
                i_caso4_data[j] = (int32_t) ( i_caso4 ( t[j] ) / (K_I_ADC) + C_25V);
                printf("V - [%i]\tI - [%i]\n", v_caso4_data[j], i_caso4_data[j]);
        }
};
/*END OF FILE: gen_data.c*/
```

Apéndice E: Programas para estimación de tiempo de ejecución.

Estimación con regla de Simpson.

```
0000:  GOTO   0140
.....
..... int
..... main (void)
0140:  CLRF   FF8
0142:  BCF    FD0.7
0144:  MOVF   FC1,W
0146:  ANDLW  C0
0148:  IORLW  0F
014A:  MOVWF  FC1
014C:  MOVLW  07
014E:  MOVWF  FB4
..... {
.....
.....    int32_t int32_1;
.....
.....    int16_t int16_1, int16_2, int16_3, int16_4, int16_5, int16_6;
.....
.....    int32_1 += int16_1*int16_2 + 4 * int16_3*int16_4 + int16_5*int16_6;
0150:  MOVFF  0A,1A            -> 2 ciclo
0154:  MOVFF  09,19            -> 2 ciclo
0158:  MOVFF  0C,1C            -> 2 ciclo
015C:  MOVFF  0B,1B            -> 2 ciclo
0160:  RCALL  0004 -> 2 ciclo

0004:  MOVF   1A,W-> 1 ciclo
0006:  XORWF  1C,W             -> 1 ciclo
0008:  ANDLW  80   -> 1 ciclo
000A:  MOVWF  1E -> 1 ciclo
000C:  BTFSS  1A.7  -> 2 ciclo
000E:  GOTO   001C
0012:  COMF   19,F  -> 1 ciclo
0014:  COMF   1A,F  -> 1 ciclo
0016:  INCF   19,F     -> 1 ciclo
0018:  BTFSC  FD8.2            -> 1 ciclo
001A:  INCF   1A,F  -> 1 ciclo
001C:  BTFSS  1C.7  -> 2 ciclo
001E:  GOTO   002C
0022:  COMF   1B,F  -> 1 ciclo
0024:  COMF   1C,F  -> 1 ciclo
0026:  INCF   1B,F     -> 1 ciclo
0028:  BTFSC  FD8.2            -> 1 ciclo
002A:  INCF   1C,F  -> 1 ciclo
002C:  MOVF   19,W-> 1 ciclo
002E:  MULWF  1B  -> 1 ciclo
0030:  MOVFF  FF3,01           -> 2 ciclo
0034:  MOVFF  FF4,00           -> 2 ciclo
0038:  MULWF  1C  -> 1 ciclo
003A:  MOVF   FF3,W            -> 1 ciclo
003C:  ADDWF  00,F             -> 1 ciclo
003E:  MOVF   1A,W             -> 1 ciclo
0040:  MULWF  1B  -> 1 ciclo
0042:  MOVF   FF3,W            -> 1 ciclo
0044:  ADDWFC 00,W             -> 1 ciclo
0046:  MOVWF  02   -> 1 ciclo
0048:  BTFSS  1E.7  -> 2 ciclo
004A:  GOTO   0058
004E:  COMF   01,F  -> 1 ciclo
0050:  COMF   02,F  -> 1 ciclo
0052:  INCF   01,F     -> 1 ciclo
0054:  BTFSC  FD8.2            -> 1 ciclo
0056:  INCF   02,F     -> 1 ciclo
0058:  RETURN 0     -> 2 ciclo
```

```
0162:  MOVFF  02,16            -> 2 ciclo
0166:  MOVFF  01,15            -> 2 ciclo
016A:  CLRF   1A     -> 1 ciclo
016C:  MOVLW  04   -> 1 ciclo
016E:  MOVWF  19   -> 1 ciclo
0170:  MOVFF  0E,1C            -> 2 ciclo
0174:  MOVFF  0D,1B            -> 2 ciclo
0178:  RCALL  0004 -> 2 ciclo

0004:  MOVF   1A,W-> 1 ciclo
0006:  XORWF  1C,W             -> 1 ciclo
0008:  ANDLW  80   -> 1 ciclo
000A:  MOVWF  1E   -> 1 ciclo
000C:  BTFSS  1A.7  -> 2 ciclo
000E:  GOTO   001C
0012:  COMF   19,F  -> 1 ciclo
0014:  COMF   1A,F  -> 1 ciclo
0016:  INCF   19,F     -> 1 ciclo
0018:  BTFSC  FD8.2            -> 1 ciclo
001A:  INCF   1A,F     -> 1 ciclo
001C:  BTFSS  1C.7  -> 2 ciclo
001E:  GOTO   002C
0022:  COMF   1B,F  -> 1 ciclo
0024:  COMF   1C,F  -> 1 ciclo
0026:  INCF   1B,F     -> 1 ciclo
0028:  BTFSC  FD8.2            -> 1 ciclo
002A:  INCF   1C,F     -> 1 ciclo
002C:  MOVF   19,W-> 1 ciclo
002E:  MULWF  1B   -> 1 ciclo
0030:  MOVFF  FF3,01           -> 2 ciclo
0034:  MOVFF  FF4,00           -> 2 ciclo
0038:  MULWF  1C   -> 1 ciclo
003A:  MOVF   FF3,W            -> 1 ciclo
003C:  ADDWF  00,F             -> 1 ciclo
003E:  MOVF   1A,W             -> 1 ciclo
0040:  MULWF  1B   -> 1 ciclo
0042:  MOVF   FF3,W            -> 1 ciclo
0044:  ADDWFC 00,W             -> 1 ciclo
0046:  MOVWF  02   -> 1 ciclo
0048:  BTFSS  1E.7   -> 2 ciclo
004A:  GOTO   0058
004E:  COMF   01,F  -> 1 ciclo
0050:  COMF   02,F  -> 1 ciclo
0052:  INCF   01,F     -> 1 ciclo
0054:  BTFSC  FD8.2            -> 1 ciclo
0056:  INCF   02,F     -> 1 ciclo
0058:  RETURN 0     -> 2 ciclo

017A:  MOVFF  02,18            -> 2 ciclo
017E:  MOVFF  01,17            -> 2 ciclo
0182:  MOVFF  02,1A            -> 2 ciclo
0186:  MOVFF  01,19            -> 2 ciclo
018A:  MOVFF  10,1C            -> 2 ciclo
018E:  MOVFF  0F,1B            -> 2 ciclo
0192:  RCALL  0004 -> 2 ciclo

0004:  MOVF   1A,W-> 1 ciclo
0006:  XORWF  1C,W             -> 1 ciclo
0008:  ANDLW  80   -> 1 ciclo
000A:  MOVWF  1E   -> 1 ciclo
000C:  BTFSS  1A.7  -> 2 ciclo
000E:  GOTO   001C
0012:  COMF   19,F  -> 1 ciclo
0014:  COMF   1A,F  -> 1 ciclo
0016:  INCF   19,F     -> 1 ciclo
0018:  BTFSC  FD8.2            -> 1 ciclo
001A:  INCF   1A,F     -> 1 ciclo
001C:  BTFSS  1C.7  -> 2 ciclo
001E:  GOTO   002C
```

```
0022:  COMF   1B,F  -> 1 ciclo
0024:  COMF   1C,F  -> 1 ciclo
0026:  INCF   1B,F   -> 1 ciclo
0028:  BTFSC  FD8.2            -> 1 ciclo
002A:  INCF   1C,F   -> 1 ciclo
002C:  MOVF   19,W -> 1 ciclo
002E:  MULWF  1B  -> 1 ciclo
0030:  MOVFF  FF3,01           -> 2 ciclo
0034:  MOVFF  FF4,00           -> 2 ciclo
0038:  MULWF  1C  -> 1 ciclo
003A:  MOVF   FF3,W            -> 1 ciclo
003C:  ADDWF  00,F             -> 1 ciclo
003E:  MOVF   1A,W             -> 1 ciclo
0040:  MULWF  1B  -> 1 ciclo
0042:  MOVF   FF3,W            -> 1 ciclo
0044:  ADDWFC 00,W             -> 1 ciclo
0046:  MOVWF  02   -> 1 ciclo
0048:  BTFSS  1E.7  -> 2 ciclo
004A:  GOTO   0058
004E:  COMF   01,F  -> 1 ciclo
0050:  COMF   02,F  -> 1 ciclo
0052:  INCF   01,F   -> 1 ciclo
0054:  BTFSC  FD8.2            -> 1 ciclo
0056:  INCF   02,F   -> 1 ciclo
0058:  RETURN 0     -> 2 ciclo

0194:  MOVFF  02,03            -> 2 ciclo
0198:  MOVF   01,W -> 1 ciclo
019A:  ADDWF  15,F             -> 1 ciclo
019C:  MOVF   02,W -> 1 ciclo
019E:  ADDWFC 16,F             -> 1 ciclo
01A0:  MOVFF  12,1A            -> 2 ciclo
01A4:  MOVFF  11,19            -> 2 ciclo
01A8:  MOVFF  14,1C            -> 2 ciclo
01AC:  MOVFF  13,1B            -> 2 ciclo
01B0:  RCALL  0004 -> 2 ciclo

0004:  MOVF   1A,W -> 1 ciclo
0006:  XORWF  1C,W             -> 1 ciclo
0008:  ANDLW  80   -> 1 ciclo
000A:  MOVWF  1E  -> 1 ciclo
000C:  BTFSS  1A.7  -> 2 ciclo
000E:  GOTO   001C
0012:  COMF   19,F  -> 1 ciclo
0014:  COMF   1A,F  -> 1 ciclo
0016:  INCF   19,F   -> 1 ciclo
0018:  BTFSC  FD8.2            -> 1 ciclo
001A:  INCF   1A,F   -> 1 ciclo
001C:  BTFSS  1C.7  -> 2 ciclo
001E:  GOTO   002C
0022:  COMF   1B,F  -> 1 ciclo
0024:  COMF   1C,F  -> 1 ciclo
0026:  INCF   1B,F   -> 1 ciclo
0028:  BTFSC  FD8.2            -> 1 ciclo
002A:  INCF   1C,F   -> 1 ciclo
002C:  MOVF   19,W -> 1 ciclo
002E:  MULWF  1B  -> 1 ciclo
0030:  MOVFF  FF3,01           -> 2 ciclo
0034:  MOVFF  FF4,00           -> 2 ciclo
0038:  MULWF  1C  -> 1 ciclo
003A:  MOVF   FF3,W            -> 1 ciclo
003C:  ADDWF  00,F             -> 1 ciclo
003E:  MOVF   1A,W             -> 1 ciclo
0040:  MULWF  1B  -> 1 ciclo
0042:  MOVF   FF3,W            -> 1 ciclo
0044:  ADDWFC 00,W             -> 1 ciclo
0046:  MOVWF  02   -> 1 ciclo
0048:  BTFSS  1E.7  -> 2 ciclo
004A:  GOTO   0058
004E:  COMF   01,F  -> 1 ciclo
```

```
0050:  COMF   02,F  -> 1 ciclo
0052:  INCF   01,F    -> 1 ciclo
0054:  BTFSC  FD8.2           -> 1 ciclo
0056:  INCF   02,F    -> 1 ciclo
0058:  RETURN 0     -> 2 ciclo

01B2:  MOVFF  02,03           -> 2 ciclo
01B6:  MOVF   01,W-> 1 ciclo
01B8:  ADDWF  15,W            -> 1 ciclo
01BA:  MOVWF  01 -> 1 ciclo
01BC:  MOVF   16,W            -> 1 ciclo
01BE:  ADDWFC 03,F            -> 1 ciclo
01C0:  MOVFF  01,00           -> 2 ciclo
01C4:  MOVFF  03,01           -> 2 ciclo
01C8:  CLRF   02      -> 1 ciclo
01CA:  CLRF   03      -> 1 ciclo
01CC:  BTFSS  01.7  -> 2 ciclo
01CE:  BRA    01D4
01D0:  DECF   02,F  -> 1 ciclo
01D2:  DECF   03,F  -> 1 ciclo
01D4:  MOVF   00,W-> 1 ciclo
01D6:  ADDWF  05,F            -> 1 ciclo
01D8:  MOVF   01,W-> 1 ciclo
01DA:  ADDWFC 06,F            -> 1 ciclo
01DC:  MOVF   02,W            -> 1 ciclo
01DE:  ADDWFC 07,F            -> 1 ciclo
01E0:  MOVF   03,W-> 1 ciclo
01E2:  ADDWFC 08,F            -> 1 ciclo Hasta este punto se cuentan 240 ciclos de instrucción para ejecutar la línea.
.....................
.....................    int32_1 /= 135;
01E4:  BCF    FD8.1  -> 1 ciclo
01E6:  MOVFF  08,18           -> 2 ciclo
01EA:  MOVFF  07,17           -> 2 ciclo
01EE:  MOVFF  06,16           -> 2 ciclo
01F2:  MOVFF  05,15           -> 2 ciclo
01F6:  CLRF   1C      -> 1 ciclo
01F8:  CLRF   1B      -> 1 ciclo
01FA:  CLRF   1A      -> 1 ciclo
01FC:  MOVLW  87  -> 1 ciclo
01FE:  MOVWF  19  -> 1 ciclo
0200:  BRA    005A   -> 2 ciclo

005A:  BTFSC  FD8.1           -> 2 ciclo
005C:  BRA    0064
005E:  CLRF   FEA   -> 1 ciclo
0060:  MOVLW  1D  -> 1 ciclo
0062:  MOVWF  FE9-> 1 ciclo
0064:  MOVF   18,W -> 1 ciclo
0066:  XORWF  1C,W            -> 1 ciclo
0068:  ANDLW  80    -> 1 ciclo
006A:  MOVWF  22  -> 1 ciclo
006C:  BTFSS  18.7  -> 2 ciclo
006E:  BRA    0086
0070:  COMF   15,F  -> 1 ciclo
0072:  COMF   16,F  -> 1 ciclo
0074:  COMF   17,F  -> 1 ciclo
0076:  COMF   18,F  -> 1 ciclo
0078:  INCF   15,F    -> 1 ciclo
007A:  BTFSC  FD8.2           -> 1 ciclo
007C:  INCF   16,F    -> 1 ciclo
007E:  BTFSC  FD8.2           -> 1 ciclo
0080:  INCF   17,F    -> 1 ciclo
0082:  BTFSC  FD8.2           -> 1 ciclo
0084:  INCF   18,F    -> 1 ciclo
0086:  BTFSS  1C.7  -> 2 ciclo
0088:  BRA    00A0
008A:  COMF   19,F  -> 1 ciclo
008C:  COMF   1A,F  -> 1 ciclo
008E:  COMF   1B,F  -> 1 ciclo
0090:  COMF   1C,F  -> 1 ciclo
```

```
0092: INCF  19,F    -> 1 ciclo
0094: BTFSC FD8.2           -> 1 ciclo
0096: INCF  1A,F    -> 1 ciclo
0098: BTFSC FD8.2           -> 1 ciclo
009A: INCF  1B,F    -> 1 ciclo
009C: BTFSC FD8.2           -> 1 ciclo
009E: INCF  1C,F    -> 1 ciclo
00A0: CLRF  00      -> 1 ciclo
00A2: CLRF  01      -> 1 ciclo
00A4: CLRF  02      -> 1 ciclo
00A6: CLRF  03      -> 1 ciclo
00A8: CLRF  1D      -> 1 ciclo
00AA: CLRF  1E      -> 1 ciclo
00AC: CLRF  1F      -> 1 ciclo
00AE: CLRF  20      -> 1 ciclo
00B0: MOVF  1C,W            -> 1 ciclo
00B2: IORWF 1B,W            -> 1 ciclo
00B4: IORWF 1A,W            -> 1 ciclo
00B6: IORWF 19,W            -> 1 ciclo
00B8: BZ    0112    -> 1 ciclo
00BA: MOVLW 20 -> 1 ciclo
00BC: MOVWF 21 -> 1 ciclo
00BE: BCF   FD8.0 -> 1 ciclo
00C0: RLCF  15,F    -> 1 ciclo
00C2: RLCF  16,F    -> 1 ciclo
00C4: RLCF  17,F    -> 1 ciclo
00C6: RLCF  18,F    -> 1 ciclo
00C8: RLCF  1D,F    -> 1 ciclo
00CA: RLCF  1E,F    -> 1 ciclo
00CC: RLCF  1F,F    -> 1 ciclo
00CE: RLCF  20,F    -> 1 ciclo
00D0: MOVF  1C,W            -> 1 ciclo
00D2: SUBWF 20,W            -> 1 ciclo
00D4: BNZ   00E6    -> 1 ciclo
00D6: MOVF  1B,W            -> 1 ciclo
00D8: SUBWF 1F,W            -> 1 ciclo
00DA: BNZ   00E6    -> 1 ciclo
00DC: MOVF  1A,W            -> 1 ciclo
00DE: SUBWF 1E,W            -> 1 ciclo
00E0: BNZ   00E6    -> 1 ciclo
00E2: MOVF  19,W-> 1 ciclo
00E4: SUBWF 1D,W            -> 1 ciclo
00E6: BNC   0106    -> 1 ciclo
00E8: MOVF  19,W-> 1 ciclo
00EA: SUBWF 1D,F            -> 1 ciclo
00EC: MOVF  1A,W            -> 1 ciclo
00EE: BTFSS FD8.0           -> 1 ciclo
00F0: INCFSZ 1A,W           -> 1 ciclo
00F2: SUBWF 1E,F-> 1 ciclo
00F4: MOVF  1B,W-> 1 ciclo
00F6: BTFSS FD8.0-> 1 ciclo
00F8: INCFSZ 1B,W           -> 1 ciclo
00FA: SUBWF 1F,F            -> 1 ciclo
00FC: MOVF  1C,W            -> 1 ciclo
00FE: BTFSS FD8.0           -> 1 ciclo
0100: INCFSZ 1C,W-> 1 ciclo
0102: SUBWF 20,F -> 1 ciclo
0104: BSF   FD8.0   -> 1 ciclo
0106: RLCF  00,F    -> 1 ciclo
0108: RLCF  01,F    -> 1 ciclo
010A: RLCF  02,F    -> 1 ciclo
010C: RLCF  03,F    -> 1 ciclo
010E: DECFSZ 21,F -> 2 ciclo
0110: BRA   00BE
0112: BTFSS 22.7    -> 2 ciclo
0114: BRA   012C
0116: COMF  00,F    -> 1 ciclo
0118: COMF  01,F    -> 1 ciclo
011A: COMF  02,F    -> 1 ciclo
011C: COMF  03,F    -> 1 ciclo
```

```
011E:  INCF   00,F    -> 1 ciclo
0120:  BTFSC  FD8.2             -> 1 ciclo
0122:  INCF   01,F    -> 1 ciclo
0124:  BTFSC  FD8.2             -> 1 ciclo
0126:  INCF   02,F    -> 1 ciclo
0128:  BTFSC  FD8.2             -> 1 ciclo
012A:  INCF   03,F    -> 1 ciclo
012C:  MOVFF  1D,FEF-> 2 ciclo
0130:  MOVFF  1E,FEC-> 2 ciclo
0134:  MOVFF  1F,FEC-> 2 ciclo
0138:  MOVFF  20,FEC-> 2 ciclo
013C:  GOTO   0202 (RETURN)-> 2 ciclo

0202:  MOVFF  03,08             -> 2 ciclo
0206:  MOVFF  02,07             -> 2 ciclo
020A:  MOVFF  01,06             -> 2 ciclo
020E:  MOVFF  00,05             -> 2 ciclo Hasta este punto son 139 ciclos de instrucción para ejecutar la segunda línea
.....
.....    return 0;
0212:  MOVLW  00
0214:  MOVWF  01
.....
..... }
0216:  SLEEP
```

Estimación con Método del Rectángulo.

```
0000:  GOTO   0142
.....
.....
..... int
..... main (void)
0142:  CLRF   FF8
0144:  BCF    FD0.7
0146:  MOVF   FC1,W
0148:  ANDLW  C0
014A:  IORLW  0F
014C:  MOVWF  FC1
014E:  MOVLW  07
0150:  MOVWF  FB4
..... {
.....
.....    int16_t int16_1, int16_2;
.....    int32_t int32_1;
.....
.....
.....    int32_1 += int16_1 * int16_2;
0152:  MOVFF  06,0E             -> 2 ciclo
0156:  MOVFF  05,0D             -> 2 ciclo
015A:  MOVFF  08,10             -> 2 ciclo
015E:  MOVFF  07,0F             -> 2 ciclo
0162:  BRA    0004    -> 2 ciclo

0004:  MOVF   0E,W-> 1 ciclo
0006:  XORWF  10,W              -> 1 ciclo
0008:  ANDLW  80    -> 1 ciclo
000A:  MOVWF  12    -> 1 ciclo
000C:  BTFSS  0E.7    -> 2 ciclo
000E:  GOTO   001C
0012:  COMF   0D,F  -> 1 ciclo
0014:  COMF   0E,F  -> 1 ciclo
0016:  INCF   0D,F    -> 1 ciclo
0018:  BTFSC  FD8.2             -> 1 ciclo
001A:  INCF   0E,F    -> 1 ciclo
001C:  BTFSS  10.7    -> 2 ciclo
001E:  GOTO   002C
0022:  COMF   0F,F  -> 1 ciclo
0024:  COMF   10,F  -> 1 ciclo
0026:  INCF   0F,F    -> 1 ciclo
0028:  BTFSC  FD8.2             -> 1 ciclo
```

002A: INCF 10,F -> 1 ciclo
002C: MOVF 0D,W -> 1 ciclo
002E: MULWF 0F -> 1 ciclo
0030: MOVFF FF3,01 -> 2 ciclo
0034: MOVFF FF4,00 -> 2 ciclo
0038: MULWF 10 -> 1 ciclo
003A: MOVF FF3,W -> 1 ciclo
003C: ADDWF 00,F -> 1 ciclo
003E: MOVF 0E,W-> 1 ciclo
0040: MULWF 0F -> 1 ciclo
0042: MOVF FF3,W -> 1 ciclo
0044: ADDWFC 00,W -> 1 ciclo
0046: MOVWF 02 -> 1 ciclo
0048: BTFSS 12.7 -> 2 ciclo
004A: GOTO 0058
004E: COMF 01,F -> 1 ciclo
0050: COMF 02,F -> 1 ciclo
0052: INCF 01,F -> 1 ciclo
0054: BTFSC FD8.2 -> 1 ciclo
0056: INCF 02,F -> 1 ciclo
0058: GOTO 0164 (RETURN)-> 2 ciclo

0164: MOVFF 02,03 -> 2 ciclo
0168: MOVFF 01,00 -> 2 ciclo
016C: MOVFF 02,01 -> 2 ciclo
0170: CLRF 02 -> 1 ciclo
0172: CLRF 03 -> 1 ciclo
0174: BTFSS 01.7 -> 2 ciclo
0176: BRA 017C
0178: DECF 02,F -> 1 ciclo
017A: DECF 03,F -> 1 ciclo
017C: MOVF 00,W -> 1 ciclo
017E: ADDWF 09,F -> 1 ciclo
0180: MOVF 01,W -> 1 ciclo
0182: ADDWFC 0A,F -> 1 ciclo
0184: MOVF 02,W -> 1 ciclo
0186: ADDWFC 0B,F -> 1 ciclo
0188: MOVF 03,W -> 1 ciclo
018A: ADDWFC 0C,F -> 1 ciclo Hasta este punto son 71 ciclos de instrucción para ejecutar la línea
.....
..... int32_1 /= 45;
018C: BCF FD8.1 -> 1 ciclo
018E: MOVFF 0C,10 -> 2 ciclo
0192: MOVFF 0B,0F -> 2 ciclo
0196: MOVFF 0A,0E -> 2 ciclo
019A: MOVFF 09,0D -> 1 ciclo
019E: CLRF 14 -> 1 ciclo
01A0: CLRF 13 -> 1 ciclo
01A2: CLRF 12 -> 1 ciclo
01A4: MOVLW 2D -> 1 ciclo
01A6: MOVWF 11 -> 1 ciclo
01A8: BRA 005C -> 2 ciclo

005C: BTFSC FD8.1 -> 2 ciclo
005E: BRA 0066
0060: CLRF FEA -> 1 ciclo
0062: MOVLW 15 -> 1 ciclo
0064: MOVWF FE9-> 1 ciclo
0066: MOVF 10,W -> 1 ciclo
0068: XORWF 14,W -> 1 ciclo
006A: ANDLW 80 -> 1 ciclo
006C: MOVWF 1A -> 1 ciclo
006E: BTFSS 10.7 -> 2 ciclo
0070: BRA 0088
0072: COMF 0D,F -> 1 ciclo
0074: COMF 0E,F -> 1 ciclo
0076: COMF 0F,F -> 1 ciclo
0078: COMF 10,F -> 1 ciclo
007A: INCF 0D,F -> 1 ciclo
007C: BTFSC FD8.2 -> 1 ciclo

```
007E: INCF   0E,F   -> 1 ciclo
0080: BTFSC  FD8.2          -> 1 ciclo
0082: INCF   0F,F   -> 1 ciclo
0084: BTFSC  FD8.2          -> 1 ciclo
0086: INCF   10,F   -> 1 ciclo
0088: BTFSS  14.7   -> 2 ciclo
008A: BRA    00A2
008C: COMF   11,F  -> 1 ciclo
008E: COMF   12,F  -> 1 ciclo
0090: COMF   13,F  -> 1 ciclo
0092: COMF   14,F  -> 1 ciclo
0094: INCF   11,F   -> 1 ciclo
0096: BTFSC  FD8.2          -> 1 ciclo
0098: INCF   12,F   -> 1 ciclo
009A: BTFSC  FD8.2          -> 1 ciclo
009C: INCF   13,F   -> 1 ciclo
009E: BTFSC  FD8.2          -> 1 ciclo
00A0: INCF   14,F   -> 1 ciclo
00A2: CLRF   00     -> 1 ciclo
00A4: CLRF   01     -> 1 ciclo
00A6: CLRF   02     -> 1 ciclo
00A8: CLRF   03     -> 1 ciclo
00AA: CLRF   15     -> 1 ciclo
00AC: CLRF   16     -> 1 ciclo
00AE: CLRF   17     -> 1 ciclo
00B0: CLRF   18     -> 1 ciclo
00B2: MOVF   14,W-> 1 ciclo
00B4: IORWF  13,W           -> 1 ciclo
00B6: IORWF  12,W           -> 1 ciclo
00B8: IORWF  11,W           -> 1 ciclo
00BA: BZ     0114   -> 1 ciclo
00BC: MOVLW  20 -> 1 ciclo
00BE: MOVWF  19  -> 1 ciclo
00C0: BCF    FD8.0  -> 1 ciclo
00C2: RLCF   0D,F   -> 1 ciclo
00C4: RLCF   0E,F   -> 1 ciclo
00C6: RLCF   0F,F   -> 1 ciclo
00C8: RLCF   10,F   -> 1 ciclo
00CA: RLCF   15,F   -> 1 ciclo
00CC: RLCF   16,F   -> 1 ciclo
00CE: RLCF   17,F   -> 1 ciclo
00D0: RLCF   18,F   -> 1 ciclo
00D2: MOVF   14,W-> 1 ciclo
00D4: SUBWF  18,W           -> 1 ciclo
00D6: BNZ    00E8   -> 1 ciclo
00D8: MOVF   13,W-> 1 ciclo
00DA: SUBWF  17,W           -> 1 ciclo
00DC: BNZ    00E8   -> 1 ciclo
00DE: MOVF   12,W           -> 1 ciclo
00E0: SUBWF  16,W           -> 1 ciclo
00E2: BNZ    00E8   -> 1 ciclo
00E4: MOVF   11,W-> 1 ciclo
00E6: SUBWF  15,W           -> 1 ciclo
00E8: BNC    0108   -> 1 ciclo
00EA: MOVF   11,W           -> 1 ciclo
00EC: SUBWF  15,F-> 1 ciclo
00EE: MOVF   12,W-> 1 ciclo
00F0: BTFSS  FD8.0-> 1 ciclo
00F2: INCFSZ 12,W-> 1 ciclo
00F4: SUBWF  16,F -> 1 ciclo
00F6: MOVF   13,W-> 1 ciclo
00F8: BTFSS  FD8.0-> 1 ciclo
00FA: INCFSZ 13,W           -> 1 ciclo
00FC: SUBWF  17,F-> 1 ciclo
00FE: MOVF   14,W-> 1 ciclo
0100: BTFSS  FD8.0-> 1 ciclo
0102: INCFSZ 14,W-> 1 ciclo
0104: SUBWF  18,F -> 1 ciclo
0106: BSF    FD8.0  -> 1 ciclo
0108: RLCF   00,F   -> 1 ciclo
```

```
010A:  RLCF   01,F   -> 1 ciclo
010C:  RLCF   02,F   -> 1 ciclo
010E:  RLCF   03,F   -> 1 ciclo
0110:  DECFSZ 19,F -> 2 ciclo
0112:  BRA    00C0
0114:  BTFSS  1A.7   -> 2 ciclo
0116:  BRA    012E
0118:  COMF   00,F   -> 1 ciclo
011A:  COMF   01,F   -> 1 ciclo
011C:  COMF   02,F   -> 1 ciclo
011E:  COMF   03,F   -> 1 ciclo
0120:  INCF   00,F    -> 1 ciclo
0122:  BTFSC  FD8.2             -> 1 ciclo
0124:  INCF   01,F    -> 1 ciclo
0126:  BTFSC  FD8.2             -> 1 ciclo
0128:  INCF   02,F    -> 1 ciclo
012A:  BTFSC  FD8.2             -> 1 ciclo
012C:  INCF   03,F    -> 1 ciclo
012E:  MOVFF  15,FEF            -> 2 ciclo
0132:  MOVFF  16,FEC            -> 2 ciclo
0136:  MOVFF  17,FEC            -> 2 ciclo
013A:  MOVFF  18,FEC            -> 2 ciclo
013E:  GOTO   01AA (RETURN)-> 2 ciclo

01AA:  MOVFF  03,0C             -> 2 ciclo
01AE:  MOVFF  02,0B             -> 2 ciclo
01B2:  MOVFF  01,0A             -> 2 ciclo
01B6:  MOVFF  00,09             -> 2 ciclo Hasta este punto son 138 ciclos de instrucción para ejecutar la línea
.....
.....
.....    return 0;
01BA:  MOVLW  00
01BC:  MOVWF  01
.....
..... }
01BE:  SLEEP
```