CAPÍTULO 1

INTRODUCCIÓN

1.1 Relevadores digitales

Los relevadores de protección con microprocesadores, también llamados relevadores digitales o relevadores numéricos, están siendo ampliamente aceptados en el ámbito de la protección de sistemas eléctricos de potencia. Esta tendencia está motivada por el hecho de que los relevadores digitales son dispositivos con múltiples capacidades que realizan funciones de protección, medición, control y supervisión.

Los relevadores digitales han revolucionado el enfoque tradicional de las protecciones eléctricas. Al estar implantada su lógica de protección en software, son pocos los límites en la innovación de nuevas técnicas de protección.

Desde el origen de los relevadores digitales muchos algoritmos han sido formulados y probados; algunos de estos algoritmos permiten tener relevadores digitales adaptables que se ajustan automáticamente a las condiciones prevalecientes de un sistema de potencia, o relevadores digitales que hacen uso de redes neuronales y de inteligencia artificial para identificar fallas con mayor eficacia. Como resultado, se tienen mejores soluciones para los problemas que se plantean los ingenieros de protecciones eléctricas.

También, la capacidad de comunicación de los relevadores digitales hace posible que interactúen con otros relevadores, o con estaciones de monitoreo y control en forma local o remota. Por lo tanto, se tiene una supervisión continua y una mayor cantidad de información de un sistema de potencia al usar relevadores digitales.

Estas y otras características de los relevadores de protección digitales los convierten en una pieza fundamental dentro de la estructura de automatización y control de los sistemas de potencia.

1.2 Antecedentes

Los antecesores de los relevadores digitales son los relevadores electromecánicos y los relevadores de estado sólido. El principio de funcionamiento de cada uno de estos relevadores es diferente debido a las diferentes tecnologías que usan. Históricamente, la evolución de los relevadores de protección se ha dado como se muestra en la figura 1.1 [17].

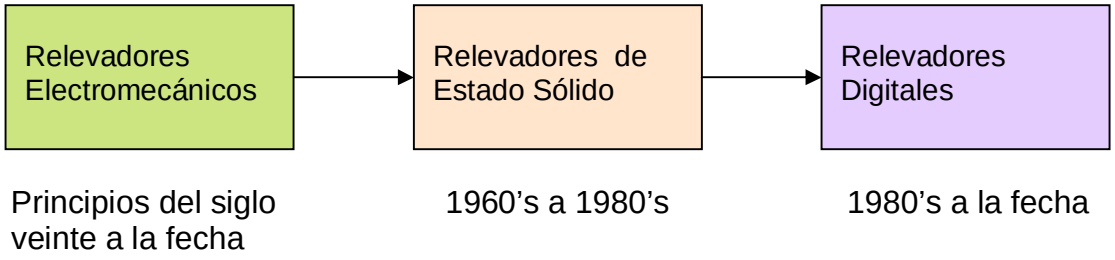


Figura 1.1: Evolución de los relevadores de protección**.**

1.2.1 Relevadores electromecánicos

Los primeros dispositivos automáticos que se emplearon para aislar fallas eléctricas en los sistemas de potencia fueron los fusibles. Los fusibles se siguen usando, pero, tienen la desventaja de tener que ser reemplazados después de que ha ocurrido una falla.

Para resolver el inconveniente de los fusibles, se diseñó el interruptor de potencia automático, el cual tenía una bobina de disparo, que operaba por sobrecarga o baja tensión. Después, durante el desarrollo de las protecciones eléctricas, la tendencia fue incorporar relevadores de protección independientes del interruptor. Estos relevadores eran electromecánicos y actuaban operando sus contactos sobre la bobina del interruptor de potencia cuando se presentaba una falla [12].

El principio de funcionamiento de los relevadores electromecánicos está basado en los fenómenos de la atracción e inducción electromagnética. Algunos de estos relevadores electromecánicos consisten de un disco de inducción, un núcleo electromagnético, un muelle en forma de espiral, una bobina secundaria o de sombra y de una unidad de sello indicadora. En la figura 1.2 se observa un relevador electromecánico.

Los relevadores electromecánicos de sobrecorriente con disco de inducción funcionan bajo el mismo principio de operación del motor de inducción. Un relevador de sobrecorriente con disco de inducción opera girando su disco de inducción cuando circula por la bobina del relevador una corriente que supera un valor máximo de corriente o punto de arranque [12], [17]. Al girar el disco de inducción, este cierra un contacto que inicia el disparo de un interruptor de potencia o acciona una alarma.

Figura 1.2: Relevador electromecánico.

El tiempo en que tarda el disco de inducción en cerrar el contacto de disparo, es inversamente proporcional a la corriente que circula por la bobina del relevador; este tiempo está controlado, también, por el ángulo de desplazamiento del disco que se ajusta a través de una palanca de tiempo. El punto de arranque se obtiene mediante derivaciones (taps) de la bobina, para variar el mismo número de amper-vueltas necesarias para mover el disco.

1.2.2 Relevadores de estado sólido

Los relevadores de estado sólido surgieron como alternativa de los relevadores electromecánicos. Los relevadores de estado sólido están diseñados con circuitos electrónicos analógicos, estos circuitos emulan las características de los relevadores electromecánicos [17].

Los primeros de estos circuitos, que usaban como elemento principal al transistor, presentaron problemas para soportar transitorios de tensión y sufrían cambios en sus características de operación debidos a la temperatura y a la baja capacidad para soportar sobretensiones o sobrecorrientes en régimen permanente; con el paso del tiempo estos problemas quedaron superados debido a la incorporación de circuitos que compensaban estas anomalías [4].

Finalmente, con el arribo de la tecnología de integración de circuitos los relevadores de estado sólido hicieron uso de los amplificadores operacionales.

Los relevadores de sobrecorriente instantáneos de estado sólido utilizan circuitos detectores de nivel como el de la figura 1.3

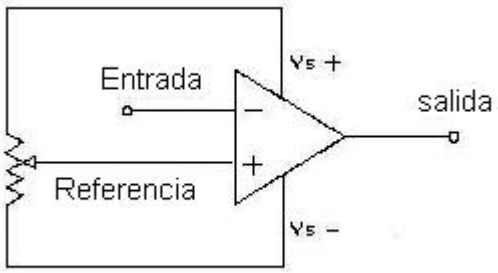


Figura 1.3: Detector de nivel para un relevador de sobrecorriente instantáneo de estado sólido.

En el circuito de la figura 1.3, si el valor de entrada no excede al de referencia, se tendrá en la salida el valor máximo negativo (Vs-) que alimenta al amplificador operacional. Cuando el valor de entrada excede al valor de referencia, en la salida se tendrá el valor máximo positivo (Vs+) que alimenta al amplificador operacional. Un valor máximo positivo (Vs+) en la salida del amplificador operacional es equivalente a la detección de una sobrecorriente. En un detector de nivel se usan señales de tensión, entonces, para la detección de sobrecorrientes, es necesario utilizar un circuito conversor de corriente a tensión.

Las ventajas que presentan los relevadores de estado sólido frente a los relevadores electromecánicos son: tiempos de respuesta más rápidos, mayor tiempo de vida, menor mantenimiento, mayor sensibilidad, reducción de tamaño y una operación silenciosa [12], [24].

Los relevadores de protección de estado sólido han quedado fuera de uso progresivamente desde el surgimiento de los relevadores digitales. Los relevadores de estado sólido no tienen la misma flexibilidad de los relevadores digitales, y su exactitud y rango dinámico están limitados por sus componentes electrónicos [17].

1.2.3 Relevadores digitales

Los relevadores digitales fueron planteados por primera vez por el Dr. Rockefeller en 1969 como una aplicación de una computadora digital para realizar funciones de protección de todos los equipos de una subestación [11], [13]. La evolución de los relevadores digitales ha devenido del rápido avance de la tecnología de los microprocesadores y al surgimiento de novedosos algoritmos de protección.

Algunos de los centros prominentes en investigación en relevadores digitales han sido la Universidad de Missouri, el Colegio Imperial de Ciencia y Tecnología en Londres, la Universidad de Calgary, la Universidad de Saskatchewan y la Universidad de Manitoba [13].

El objetivo de los relevadores digitales es igualar o exceder el desempeño de los relevadores de estado sólido y de los relevadores electromecánicos [21]. Un relevador digital convierte señales analógicas de tensión y corriente en cantidades binarias por medio de un convertidor analógico digital, luego, estas cantidades son procesadas numéricamente por los algoritmos o programas de cómputo del relevador [2]. Los algoritmos se encargan de la detección de fallas y del control de las señales de disparo. En la figura 1.4 se muestra un relevador digital comercial.

Figura 1.4: Relevador digital comercial.

Una de las ventajas destacables de los relevadores digitales es su capacidad para auto diagnosticarse [13], [16]. La función de autodiagnóstico se encarga de monitorear continuamente el estado del relevador (hardware y software), y cuando hay una falla interna del relevador digital, este queda fuera de servicio automáticamente, bloqueando sus funciones de protección y enviando una señal de alerta.

Además, los relevadores digitales pueden almacenar y enviar datos de algunos ciclos de prefalla y falla para su posterior análisis. Estas características de los relevadores digitales traen como consecuencia la reducción de las rutinas de mantenimiento y una gran confiabilidad en su operación.

También, los relevadores digitales simplifican el cableado necesario para un sistema de protección [16]. Cuando se requiere de modificar los esquemas de protección, un cambio en la programación de la lógica de protección del relevador, ahorra muchas conexiones que serían necesarias con los relevadores electromecánicos y los relevadores estáticos.

Otra ventaja de los relevadores digitales frente a sus antecesores, es su capacidad multifuncional, esto es, la capacidad de alojar en el mismo relevador diferentes funciones de protección como: protección contra sobrecorrientes, protección por bajo voltaje, protección de distancia, y otras, dependiendo de las características del hardware y software del relevador [3].

Las desventajas que pueden presentar los relevadores digitales se deben a los rápidos ciclos tecnológicos, que obligan a utilizar equipos que sean compatibles en protocolos de comunicación, en hardware, y en software.

1.3 Conceptos básicos

1.3.1 Microcomputadora

Un relevador digital prácticamente es una microcomputadora dedicada a realizar funciones de protección. Por lo tanto, es de utilidad conocer la estructura básica de una microcomputadora. Una microcomputadora esta formada por cinco elementos a saber:

a) Un microprocesador (μP): es el elemento principal de una microcomputadora. El microprocesador se encarga de ejecutar operaciones y de establecer prioridades en función de un programa interno almacenado en una memoria.

b) Una memoria de lectura y escritura o memoria de acceso al azar (RAM): donde suelen almacenarse los datos que deben ser procesados.

c) Una memoria de lectura solamente (ROM): donde se almacena el programa que le indica al microprocesador qué es lo que debe hacer

desde el momento en que se enciende la microcomputadora. El microprocesador no puede cambiar los datos almacenados en la memoria ROM, pues si así lo hiciera, el μP no sabría que hacer.

d) Un reloj (CLK): el reloj sincroniza todas las operaciones a realizar.

e) Dispositivos de entrada y salida (I/O): se encargan de llevar información hacia el μP y enviar información fuera del μP. El μP recibe y transfiere datos por medio de un bus de datos y decide con que dispositivos trabajar por medio de un bus de direcciones.

1.3.2 Microcontrolador

Un microcontrolador es una microcomputadora contenida en un solo chip, aunque no de uso general. En el mismo chip están la unidad central de procesamiento, la memoria RAM, la memoria ROM, y periféricos como temporizadores, contadores, convertidores analógicos digitales, convertidores digital analógicos, puertos paralelos de entrada y salida, entre otros. Los microcontroladores se usan principalmente en aplicaciones de control automático.

1.3.3 DSP

Un DSP es un microprocesador diseñado específicamente para el procesamiento digital de señales en tiempo real. El conjunto de instrucciones y la arquitectura de un DSP están altamente optimizadas para operaciones propias del tratamiento digital de señales.

La mayoría de las instrucciones de los DSP's se ejecutan en un ciclo de reloj, en comparación con otros microprocesadores o microcontroladores convencionales que requieren de un mayor número de ciclos de reloj para procesar las mismas instrucciones de un DSP.

1.3.4 Teorema de muestreo de Nyquist-Shannon

El teorema de muestreo de Nyquist-Shannon o criterio de Nyquist afirma que cuando se muestrea una señal, la frecuencia de muestreo debe ser mayor que dos veces el ancho de banda de la señal de entrada, para poder reconstruir la señal original a partir de las muestras [7], [9]. Si B es el ancho de banda de la señal y F_m es la frecuencia de muestreo, el teorema puede expresarse del siguiente modo:

$$F_m > 2B$$

Si el criterio no es satisfecho, existirán frecuencias cuyo muestreo coincide con otras (el llamado “aliasing”). Cuando esto sucede, la señal original no puede ser reconstruida de forma unívoca a partir de la señal digital. En la figura 1.5 se observa el efecto de muestrear una señal senoidal a una frecuencia menor a la del ancho de banda de la señal de interés, como resultado se obtiene una señal senoidal de baja frecuencia.

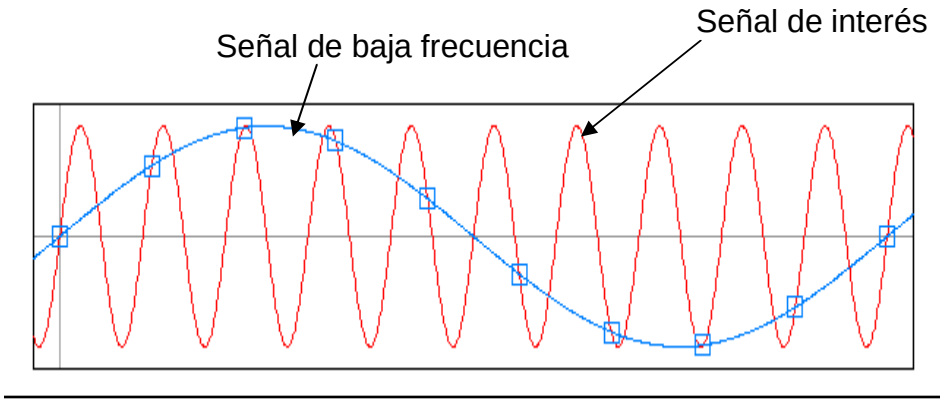


Figura 1.5: Efecto “aliasing”.

1.4 Objetivos de la tesis

El objetivo de esta tesis es diseñar e implementar un algoritmo para un relevador digital de sobrecorriente basado en el microcontrolador HCS12E64 de Motorola. Se toma como base el algoritmo de la Transformada Discreta de Fourier para la estimación de la corriente de falla.

Como objetivo particular se pretende emplear el lenguaje de programación C para la codificación del algoritmo. Además, se busca establecer la comunicación de datos entre el relevador y una computadora personal, y crear una subrutina para el ingreso de datos hacia el relevador desde un teclado matricial.

1.5 Motivación

La motivación central de este trabajo de tesis es la múltiple combinación de disciplinas académicas sobre la que esta basado el diseño de los relevadores digitales y sus algoritmos.

1.6 Justificación

En la carrera de Ingeniería en Control y Automatización se imparten cursos de programación de microcontroladores, control digital, circuitos eléctricos y análisis de Fourier, entre otros que se relacionan entre si. Estos cursos pueden ser aplicados para diseñar sistemas de control y para entender la operación de

equipos que hacen uso de tecnología digital . En el planteamiento del algoritmo para un relevador digital es necesario tener dominio de los cursos citados.

1.7 Estructura de la tesis

El presente trabajo de tesis se desarrolla en siete capítulos. El capítulo 1 es una introducción a los relevadores digitales y sus antecedentes, también, se mencionan algunos conceptos relacionados con los relevadores digitales.

En el capítulo 2 se describen los conceptos básicos de la filosofía de las protecciones eléctricas haciendo énfasis en los relevadores de sobrecorriente de tiempo inverso y la metodología para lograr su coordinación. En este capítulo se incluyen las fórmulas de las curvas características de las normas IEC-6023 y ANSI.

En el capítulo 3 se describen los principales elementos que conforman un relevador digital, y se explica como se procesan los datos en estos relevadores.

El capítulo 4 trata del análisis del algoritmo de la Transformada Discreta de Fourier. Este análisis parte de las series de Fourier y la Transformada de Fourier. Al final de capítulo se analiza la respuesta en frecuencia de los filtros digitales Seno y Coseno, filtros inherentes de la Transformada Discreta de Fourier.

En el capítulo 5 se discute el uso de un microcontrolador como alternativa en el diseño de un relevador digital, y se advierten las ventajas y desventajas del lenguaje Ensamblador frente al lenguaje C como lenguaje de programación de los algoritmos de los relevadores digitales. Después se analiza el circuito de acondicionamiento de señales usado en el prototipo y , posteriormente, se detalla la implementación de la Transformada discreta de Fourier en el microcontrolador HCS12E64.

En el capítulo 6 se exponen las pruebas de desempeño del algoritmo y los resultados obtenidos durante la elaboración de este trabajo de tesis.

En el capítulo 7 se dan las conclusiones del presente trabajo en base a los resultados obtenidos.

En el apéndice A se mencionan las herramientas de desarrollo usadas en la programación del microcontrolador HCS12E64 y la configuración de las periféricos utilizados en el programa.

El apéndice B contiene el programa final, codificado en lenguaje C, para el microcontrolador HCS12E64.

CAPÍTULO 2

FILOSOFÍA DE LAS PROTECCIONES ELÉCTRICAS

Este capítulo describe los conceptos básicos de la filosofía de las protecciones eléctricas. Principalmente, se hace énfasis en los relevadores de protección de sobrecorriente de tiempo inverso y sus curvas características. Al final del capítulo se presenta un ejemplo de la coordinación de dos relevadores de sobrecorriente de tiempo inverso.

2.1 Preámbulo

El punto de vista de la sociedad actual acerca de la energía eléctrica ha cambiado desde considerar la energía eléctrica como un servicio vital, hasta la interpretación más comercial de comodidad . Pero, la energía eléctrica tiene que generarse, transmitirse y distribuirse al mismo tiempo que se consume. Por lo tanto, para asegurar la continuidad del servicio de energía eléctrica, deben ser protegidos los equipos y sistemas dedicados a generar, transmitir y distribuir, contra posibles fallas causadas por el propio deterioro de los equipos, por condiciones ambientales severas o por errores humanos.

Ahora bien, la finalidad de las protecciones eléctricas, en cualquier caso, es asegurar la continuidad del servicio de energía eléctrica cuando se presenta un evento de falla, dejando inactivos un mínimo posible de equipos. En un sistema de potencia, las protecciones eléctricas son todos los dispositivos para detectar, localizar e iniciar la remoción de una falla.

En general, para cualquier sistema de protección bien diseñado y eficiente, de acuerdo con [23] , se consideran cuatro criterios :

1. Confiabilidad. Como se define en ANSI C37.100. Confiabilidad es la medida del grado de certeza de que un relevador se comporte correctamente.

2. Velocidad. Se requiere que un relevador opere en un tiempo mínimo para eliminar la falla (usualmente tres ciclos).

3. Selectividad. Término general que describe el comportamiento interrelacionado entre relevadores, interruptores y otros sistemas de protección. La selectividad completa se obtiene cuando una mínima cantidad de equipo es removida para aislar una falla u otra anormalidad.

4. Economía. Esto es, máxima protección con un mínimo costo.

2.2 Coordinación de protecciones eléctricas:

El objetivo de un estudio de coordinación de protecciones es determinar las características y ajustes de las protecciones contra sobrecorrientes para asegurar que un mínimo de cargas sean interrumpidas cuando los dispositivos de protección aíslan una falla o sobrecarga en cualquier parte del sistema. Un estudio de coordinación debe ser conducido en las primeras etapas de planeación de un sistema de potencia, o cuando un sistema de potencia sea modificado y nuevas cargas sean instaladas .

Para la coordinación de protecciones se hace uso de cálculos de corrientes de corto circuito. El calculo de las corrientes de corto circuito, es esencial para la selección de la capacidad adecuada de los dispositivos de protección. El máximo valor de la corriente de corto circuito esta directamente relacionado al tamaño y capacidad de la fuente de potencia, y es independiente de la corriente de carga del circuito protegido por el dispositivo de protección. Entre mayor es la capacidad de corto circuito de la fuente de potencia, mayor es la corriente de corto circuito [20].

Cuando se calculan las corrientes de corto circuito, es muy importante que se consideren todas las fuentes de corto circuito y que las características de las fuentes sean conocidas.

2.3 Relevadores de protección

La función principal de los relevadores de protección en un sistema de potencia, es remover rápidamente cualquier elemento que sufra un corto circuito, o cuando comience a operar de manera anormal, para evitar que cause daños o interfiera con la operación efectiva del resto del sistema .

Cuando ocurre una falla, los relevadores de protección son auxiliados por interruptores para desconectar la parte afectada del sistema de potencia. Los interruptores deben de tener una capacidad suficiente para manejar momentáneamente la máxima corriente de corto circuito que pueda fluir a través de sus contactos. Una función secundaria de los relevadores de protección es proveer indicación de la localización y tipo de falla [5].

Hay dos grupos de relevadores de protección: los relevadores de protección primarios y los relevadores de protección de respaldo.

2.3.1 Relevadores de protección primarios

Estos relevadores protegen a los equipos interconectados a un sistema de potencia, en base a zonas de protección. Los relevadores de protección primarios operan de manera individual o coordinados con otros relevadores dependiendo de la magnitud de la falla.

2.3.2 Relevadores de protección de respaldo

Son empleados para protección contra cortos circuitos. Y operan sólo cuando los relevadores de protección primarios fallan .

2.4 Relevadores de sobrecorriente

Los relevadores de sobrecorriente se usan para la protección de falla entre fases o de una fase a tierra en sistemas radiales de distribución y pueden ser instantáneos o temporizados. En la figura 2.1 se ilustra el diagrama unifilar de la conexión de un relevador de sobrecorriente.

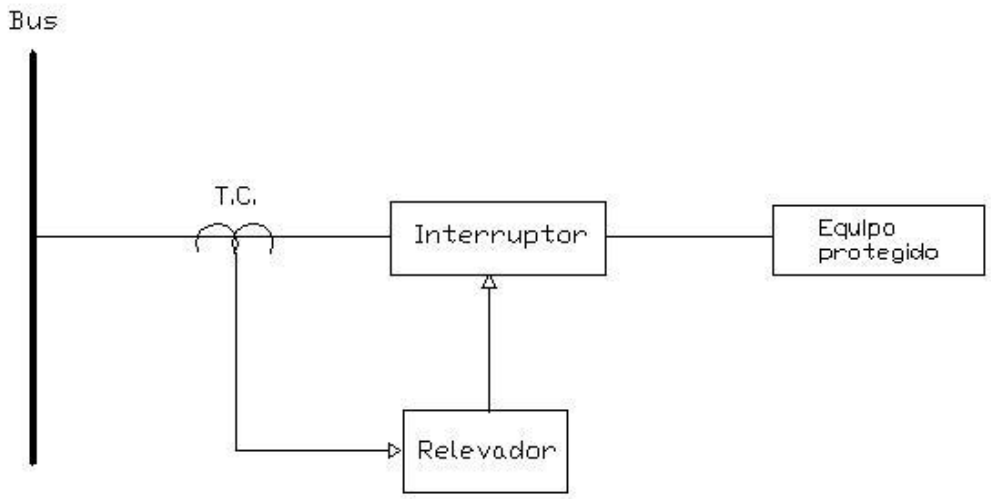


Figura 2.1: Diagrama unifilar de la conexión de un relevador de sobrecorriente.

2.4.1 Relevadores de sobrecorriente de acción instantánea

Los relevadores de sobrecorriente instantáneos operan sin ningún retrazo intencional de tiempo al superar un limite de corriente preestablecido.

2.4.2 Relevadores de sobrecorriente de tiempo inverso

El tiempo de disparo de estos relevadores esta en relación inversa a la magnitud de la corriente: a mayor corriente menor tiempo de operación. Las curvas características de tiempo inverso definen el tiempo de disparo de estos relevadores. Estos relevadores son utilizados donde se requiera de la coordinación entre protecciones de sobrecorriente.

2.5 Curvas de tiempo inverso

En la tabla 2.1 se muestran las fórmulas para diferentes tipos de curvas características de acuerdo con la norma IEC 6025–3 y la norma ANSI.

Tabla 2.1: Fórmulas de curvas características de de la norma ANSI y la norma IEC 6025-3.

Tipo de curva característica	IEC 60255 – 3	ANSI
Tiempo inverso	$t = \frac{0.14}{\left(\frac{I}{I_p}\right)^{0.02} - 1} \cdot T_p$	$t = \left(\frac{8.9341}{\left(\frac{I}{I_p}\right)^{2.0938} - 1} + 0.17966 \right) \cdot D$
Tiempo muy inverso	$t = \frac{13.5}{\left(\frac{I}{I_p}\right) - 1} \cdot T_p$	$t = \left(\frac{3.922}{\left(\frac{I}{I_p}\right)^{2} - 1} + 0.0982 \right) \cdot D$
Tiempo extremadamente inverso	$t = \frac{80}{\left(\frac{I}{I_p}\right)^{2} - 1} \cdot T_p$	$t = \left(\frac{5.64}{\left(\frac{I}{I_p}\right)^{2} - 1} + 0.02434 \right) \cdot D$
Tiempo inverso largo	$t = \frac{120}{\left(\frac{I}{I_p}\right) - 1} \cdot T_p$	$t = \left(\frac{5.6143}{\left(\frac{I}{I_p}\right)^{2} - 1} + 2.18592 \right) \cdot D$
t= Tiempo de disparo Tp= Valor del multiplicador de tiempo I= Corriente de falla I_p= Valor de la corriente de arranque D= Multiplicador de tiempo		

Las aplicaciones de algunas de estas curvas se enlistan a continuación :

a) Tiempo definido. Esta curva se aplica donde no hay necesidad de coordinar dispositivos de protección y se prefiere para la protección contra corrientes asimétricas de corta duración [8], [24].

b) Tiempo inverso. Se emplea donde hay grandes variaciones en la corriente de falla por cambios de generación o reconexiones de línea, permite una adecuada coordinación en sistemas muy grandes. La característica de tiempo inverso es suficiente para muchas aplicaciones [4], [24].

c) Tiempo muy inverso. Esta curva es generalmente empleada en casos donde la impedancia de la fuente es mucho más pequeña que la impedancia de la línea. Debido a la naturaleza de la pendiente de la curva, esta permite el uso del mismo multiplicador para varios relevadores en serie. Esto reduce errores de tiempo y sobreactuación. Esta curva es más adecuada para protección de fallas a tierra [10], [24].

d) Tiempo extremadamente inverso. Se utiliza para coordinar con fusibles y restauradores. Esta curva se recomienda para liberar rápidamente fallas que impliquen corrientes de tal magnitud que puedan causar daños graves a equipos [10], [24].

En la figura 2.2 se muestran las graficas de las curvas características mencionadas.

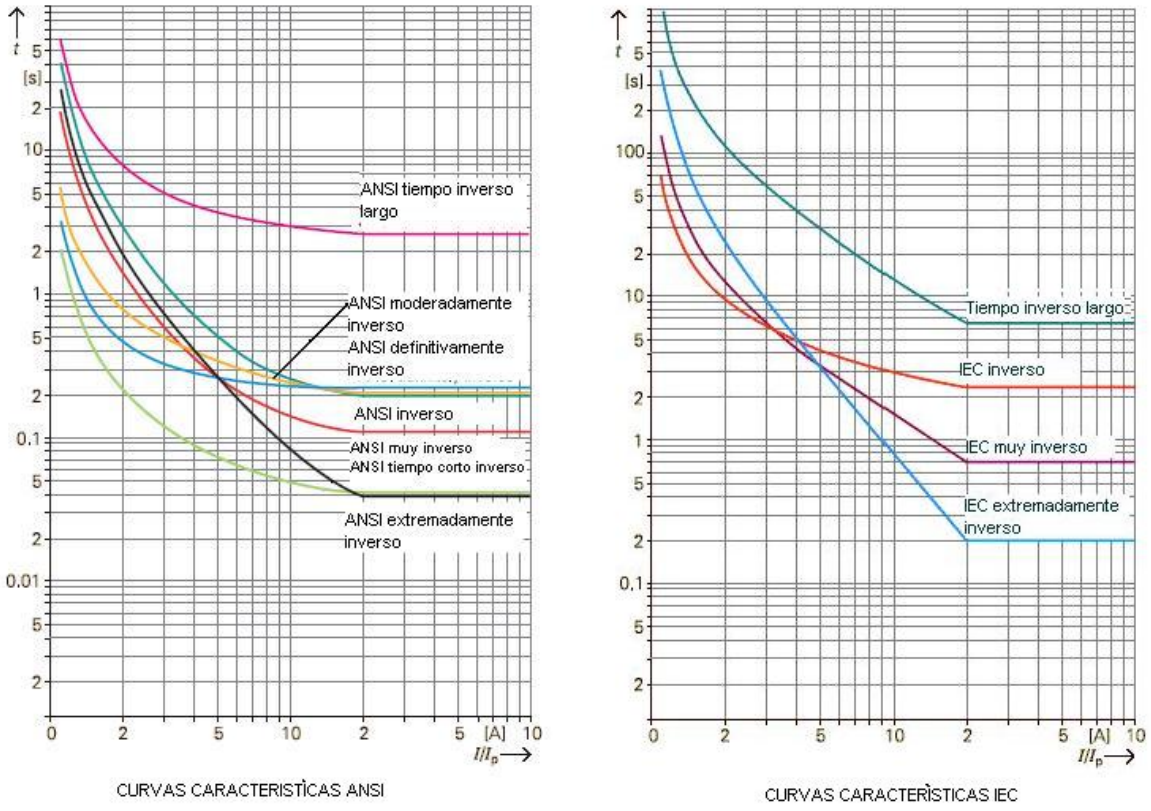


Figura 2.2: Gráficas de las curvas características tiempo-corriente de la norma IEC 6025−3 y ANSI.

2.6 Coordinación de relevadores de sobrecorriente

La coordinación de relevadores de protección es necesaria para obtener disparos selectivos ante un evento de falla. La primera regla para la protección por relevadores es que el relevador debe operar para una falla que esté solamente en su zona de protección.

La segunda regla se refiere a que el relevador no debe de operar para una falla fuera de su zona de protección, excepto para ofrecer protección de respaldo cuando un relevador o un interruptor próximo ha fallado. Un relevador de respaldo tiene que tener un tiempo suficiente de retraso para que permita la operación del relevador primario y su interruptor [15].

2.6.1 Ajustes de un relevador de sobrecorriente de tiempo inverso

Para ajustar un relevador de sobrecorriente de tiempo inverso, es necesario establecer los valores de dos parámetros, que son: la corriente de arranque del relevador y el retraso temporal.

El valor de corriente de arranque es el valor mínimo de corriente para que opere el relevador de sobrecorriente. Por ejemplo, un relevador de sobrecorriente con corriente de arranque de 4A operará solamente si la corriente a través del relevador es de 4A o superior.

El retraso de tiempo o palanca de tiempo es una variable independiente. El efecto de agregar ajustes independientes de tiempo es convertir la curva característica del relevador en una familia de curvas como se muestra en la figura 2.1. Las curvas se grafican en términos de múltiplos de la corriente de arranque y no en términos de la corriente de arranque. Esto permite usar las mismas curvas independientemente de los valores reales de la variable de entrada al relevador correspondiente a un tap determinado.

En el ajuste de relevadores de sobrecorriente se deben de considerar en los cálculos de corto circuito las posibles conexiones adicionales de cargas y fuentes de potencia, para evitar operaciones en falso de los relevadores o la pérdida de coordinación entre estos dispositivos de sobrecorriente [18].

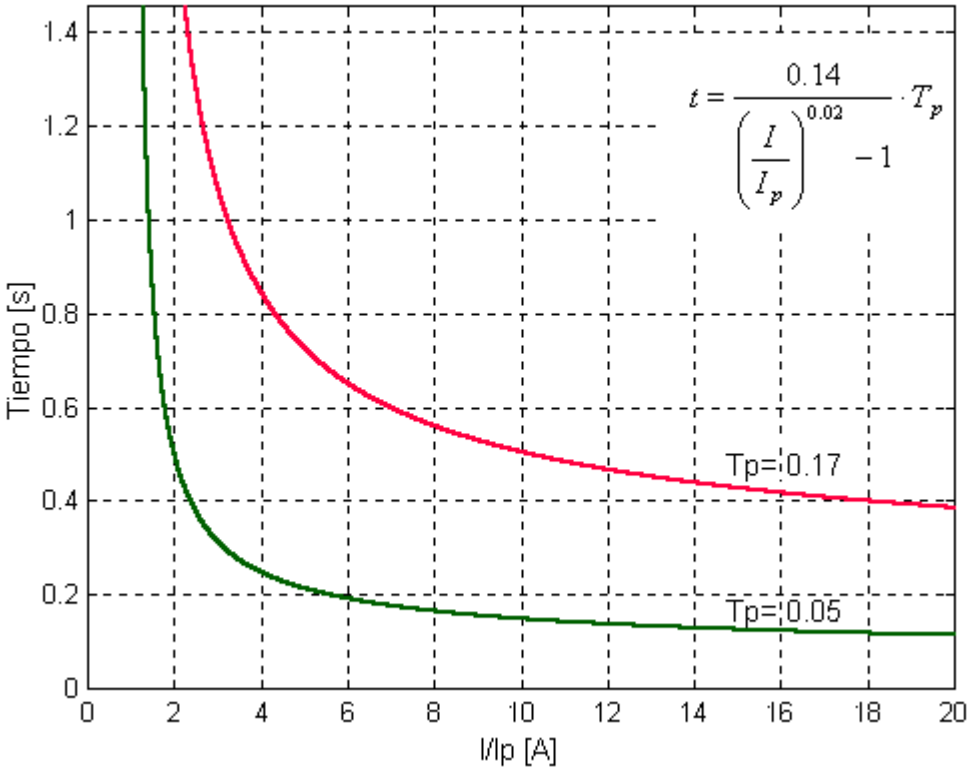


Figura 2.3: Dos curvas características de tiempo inverso con diferentes multiplicadores de tiempo.

2.6.2 Ejemplo de la coordinación de dos relevadores de sobrecorriente

Los siguientes datos tienen que estar disponibles para el ajuste de un relevador de sobrecorriente de tiempo inverso: multiplicador de tiempo, tap o corriente de arranque, tipo de curva característica, relación de transformación de los TC's para cada relevador, y los valores de corrientes de falla (mínimos y máximos).

En la figura 2.3 se muestra un sistema de distribución radial con dos buses A y B. Con los datos de cada bus se calcularon los ajustes para los relevadores R1 y R2 que protegen fallas entre fases, siguiendo el criterio siguiente:

1. R1 deberá operar para un valor excedido una tercera parte de la corriente mínima de falla medida por el relevador R2 (Este valor nunca estará abajo del doble del valor máximo de la corriente de carga).

2. R1 deberá operar para la máxima corriente de falla medida por R2 pero no antes de 0.5 segundos (tiempo de coordinación).

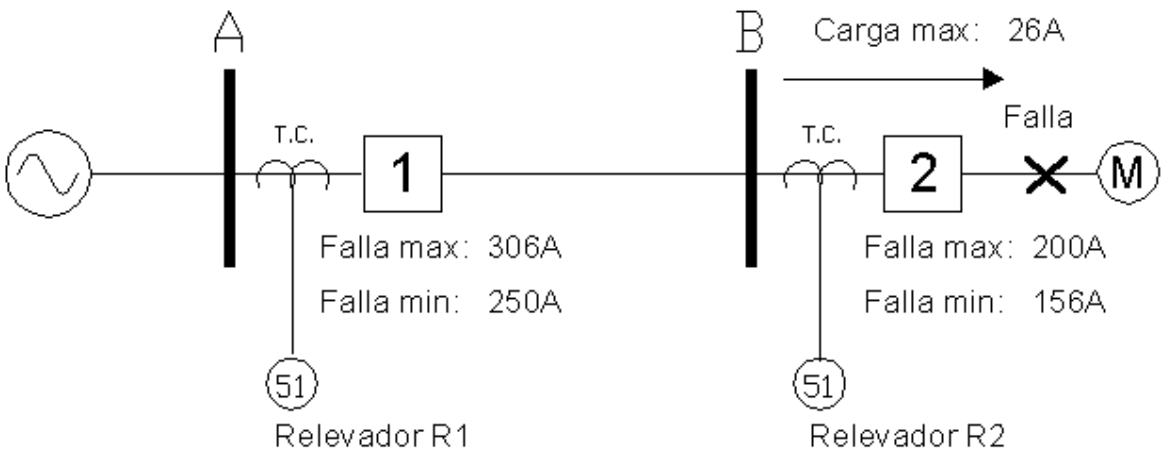


Figura 2.4 Protección de un sistema radial con dos relevadores de sobrecorriente.

Tabla 2.2: Datos del sistema de distribución radial de la Figura 2.3

Datos	Relevador	
	R1	R2
Corriente máxima de falla	306	200
Corriente mínima de falla	250	156
Relación de transformación del TC.	50:5	50:5
Tap	5	5
Multiplicador de tiempo	0.17	0.05

Con los datos de la tabla 2.2, se tienen los siguientes pasos para ajustar los relevadores R1 y R2.

Paso 1: Ajustes para el relevador R2

El relevador R2 tiene que operar para todas las corrientes arriba de 156 A. Por confiabilidad, es seleccionada una tercera parte de la corriente mínima de falla. Esto resulta en una corriente de falla en el primario del TC de 153/3= 52A. En base a esto, se selecciona un relación de 50/5=10 para el TC.

La corriente en el secundario del TC es de 52/10=5.2A. Con el fin de nivelar estos resultados, el tap del relevador se selecciona para 5A. Para asegurar un rápido disparo ante fallas hacia abajo del relevador R2, es seleccionado un

multiplicador de tiempo de 0.05. El tiempo de operación de R2 tomado de la gráfica tiempo/corriente de la figura 2.1 para el tap de 5 A, es de 0.2 segundos**.**

Paso 2: Ajustes para el relevador R1

El relevador R1 deberá de actuar como respaldo del relevador R2, y, consecuentemente, este tiene que operar para la corriente mínima de falla medida por el relevador R2, que es de 156 A. Por lo tanto, la selección de la relación del TC y del tap son los mismos para el relevador R1 y para el relevador R2.

La selección del multiplicador de tiempo, está basada en el hecho de que el relevador R1 debe de actuar 0.5 segundos después de que el relevador R2 haya operado. Eso significa que el relevador R1 tiene un retrazo de 0.5 segundos en respuesta para la corriente máxima de falla medida por el relevador R2. El tiempo de operación del relevador R1 será: 0.2 s + 0.5 s = 0.7 s.

Finalmente para el mismo tap de 5A , pero, para un tiempo de operación 0.7s del relevador R1, el valor del multiplicador o palanca de tiempo es de 0.17. Con este multiplicador de tiempo, en caso de que el relevador R2 falle, el relevador R1 deberá operar después de un retrazo de tiempo de 0.5 segundos.

CAPÍTULO 3

DESCRIPCIÓN DE UN RELEVADOR DIGITAL

En este capítulo se examinan, en forma de bloques, los diferentes elementos que conforman un relevador digital como el convertidor analógico digital, la etapa de acondicionamiento de señales, los filtros pasa bajas, y la fuente de alimentación. Al final del capítulo se describe brevemente el procesamiento de datos en un relevador digital.

3.1 Diagrama de bloques de un relevador digital

En la figura 3.1 se muestra el diagrama de bloques de un relevador digital de protección.

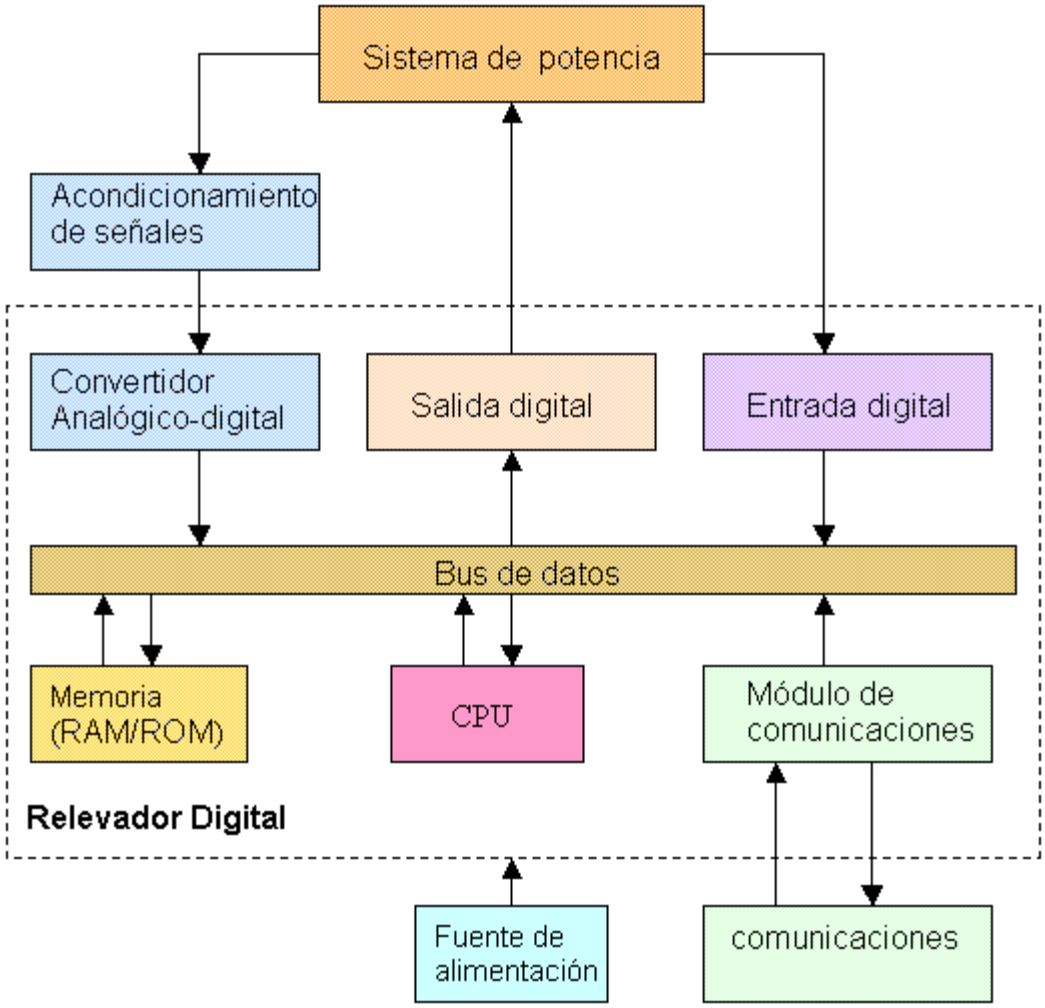


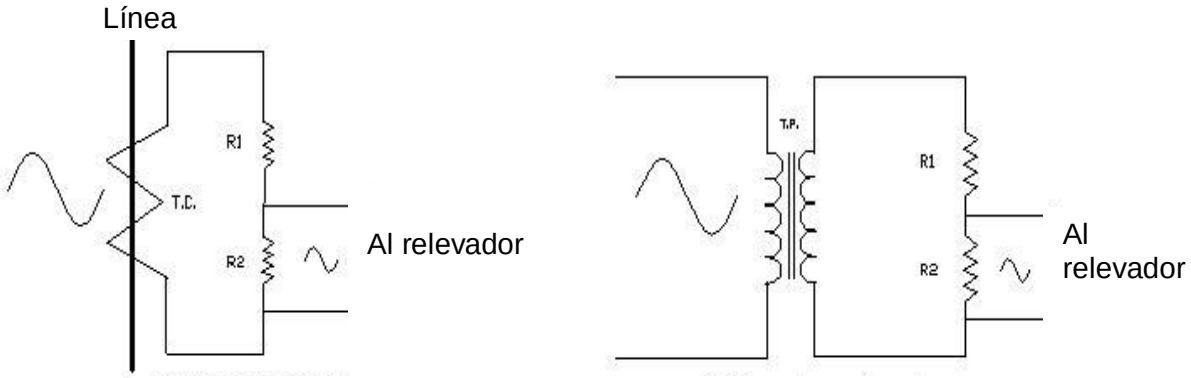
Figura 3.1: Diagrama de bloques de un relevador digital.

3.2 Acondicionamiento de señales

En esta etapa se acondicionan las señales de corriente y tensión que se toman de los transformadores de corriente y potencial del sistema de potencia. El acondicionamiento de las señales se refiere a escalar o atenuar las señales de corriente y tensión, a niveles de voltaje que puedan ser convertidos en cantidades binarias para ser procesadas por un microprocesador, un microcontrolador o un DSP.

Se deben prever en el diseño de un circuito de acondicionamiento de señales, que los valores máximos posibles de voltaje de salida de este circuito no excedan el rango aceptado por el convertidor analógico digital que se este utilizando.

En la figura 3.2 se muestra la forma de atenuar las señales de los transformadores de tensión y corriente por medio de divisores de tensión.



a) Transformador de corriente

b) Transformador de voltaje

Figura 3.2: Acondicionamiento de señales en transformadores de corriente y tensión.

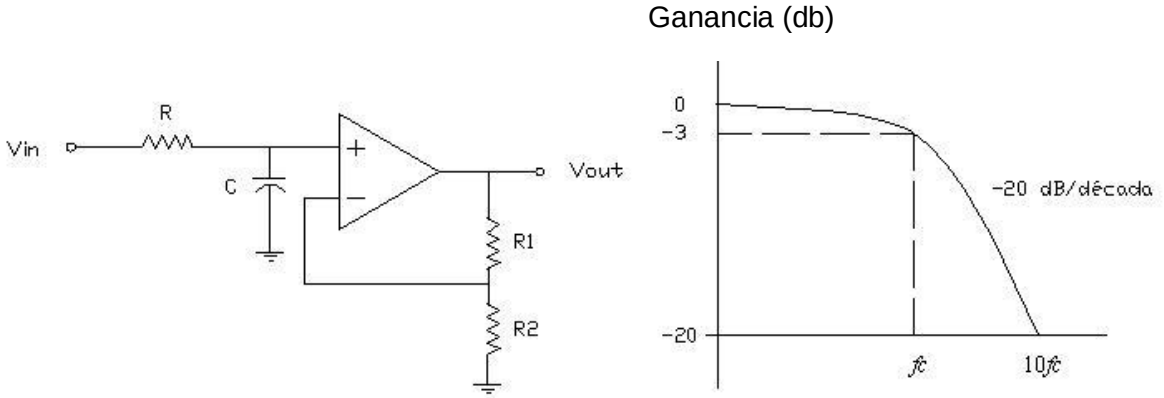
El aislamiento galvánico de las señales de tensión y corriente se obtiene a través de los transformadores de potencial y corriente. También, se emplean circuitos con amplificadores de aislamiento o circuitos con opto acopladores para evitar que las señales del sistema de potencia causen daños al relevador.

Los circuitos de acondicionamiento de señales deben de apegarse a las normas correspondientes para cumplir requerimientos como: inmunidad a transitorios, inmunidad a picos de tensión e inmunidad a señales de radiofrecuencia.

3.3 Filtros pasa bajas

En los circuitos de acondicionamiento de señales de los relevadores digitales generalmente se usan filtros pasa bajas para impedir que componentes de alta frecuencia y ruido pasen al convertidor analógico digital. El nivel de filtrado depende del tipo relevador que se trate. Algunos relevadores necesitan de filtros con frecuencias de corte especificas de acuerdo al algoritmo que se este empleando.

Por ejemplo, los relevadores diferenciales para protección de transformadores, detectan flujos de corrientes magnetizantes al cuantificar la cantidad de segundas armónicas presentes. Entonces, el filtro pasa bajas debe de preservar las señales de 60 Hz y 120 Hz. Para este caso, es suficiente una frecuencia de corte de 180 Hz [2].



a) Filtro pasa bajas activo de un solo polo

b) Curva de respuesta

Figura 3.3: Filtro pasa bajas.

Para el filtro pasa bajas de la figura 3.3 la frecuencia de corte f_c ocurre cuando X_C=R, donde: X_C es la reactancia capacitiva y R es la resistencia de la red.

$$f_c = \frac{1}{2\pi RC} \tag{3.1}$$

En el circuito del filtro pasa bajas el amplificador operacional, proporciona ganancia, por lo que la señal no se atenúa cuando pasa a través del filtro. La ganancia de voltaje en lazo cerrado A_{lc} para este filtro, está dada por:

$$A_{lc} = \frac{R_1}{R_2 + 1} \tag{3.2}$$

3.4 Convertidor Analógico Digital

En un relevador digital el convertidor analógico digital se encarga de digitalizar o convertir en cantidades binarias los voltajes de salida del circuito acondicionador de señales. Al digitalizar una señal analógica, se deben de tomar muestras con una frecuencia de muestreo mínima del doble de la frecuencia fundamental de la señal, de acuerdo con el teorema de Nyquist [9]. La figura 3.4 ilustra la conversión de una señal analógica a digital.

Para una señal de 60 Hz, la frecuencia de muestreo de Nyquist sería mayor que 120 Hz. Pero, normalmente en los relevadores digitales se usan frecuencias de muestreo de 480 Hz o superiores. Una frecuencia de muestreo típica es la de 720 Hz; con esta frecuencia de muestreo se tienen 12 muestras por cada ciclo de 60 Hz de una señal senoidal.

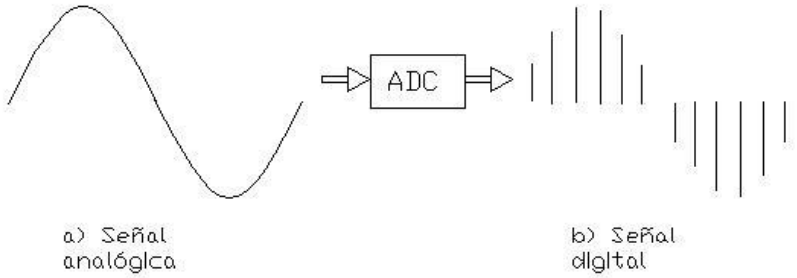


Figura 3.4: Conversión de una señal analógica a una señal digital.

La principal característica de un convertidor analógico digital es su tamaño de palabra expresada en bits. El tamaño de palabra del convertidor afecta su resolución al representar una señal analógica en formato digital.

El valor de cuantificación (valor de voltaje mínimo de conversión) de un convertidor analógico digital puede encontrarse por:

$$\Delta V = \frac{V_{RH} - V_{RL}}{2^b} \tag{3.3}$$

Donde V_{RH} es el valor máximo de voltaje que puede manejar el convertidor, V_{RL} es el voltaje mínimo que el convertidor puede manejar, y b es el número de bits del convertidor. Para un convertidor de 10 bits y con un rango de 0 a 5 volts, el valor de cuantificación es de:

$$\Delta V = \frac{5-0}{2^{10}} = 4.88\, mV \tag{3.4}$$

3.5 Fuente de alimentación

Los relevadores digitales usualmente son alimentados por una banco de baterías para asegurar la operación de los relevadores, durante un corte en el suministro de la energía de corriente alterna.

3.6 Procesamiento de datos

Las muestras del convertidor analógico digital y otros datos temporales son almacenados en la memoria de acceso aleatorio (RAM) del relevador digital. El algoritmo del relevador digital y datos fijos son almacenados en la memoria (ROM).

El algoritmo es procesado en la unidad de procesamiento central (CPU). De acuerdo con la lógica del algoritmo, la CPU activará una salida digital para disparar un interruptor en el caso de que el algoritmo detecte una falla (por ejemplo, un incremento de corriente o voltaje).

Las entradas digitales informan al relevador digital del estado actual del interruptor de potencia. La CPU a través del módulo de comunicaciones envía y recibe información de otros relevadores o de estaciones de monitoreo y control. Valores de corrientes, tensiones, tiempos de disparo, alarmas y otros, es la información que puede enviar un relevador digital. Los datos recibidos por un relevador digital están relacionados con puntos de ajuste y pruebas de funcionamiento.

Hay tres tipos de cálculos que se llevan a cabo en un relevador digital: en línea, fuera de línea y lógica de disparo [18].

Los cálculos fuera de línea se realizan para determinar los parámetros fijos que se usan durante el procesamiento en línea, evitándose de esta manera, cálculos innecesarios durante la operación del relevador. Un ejemplo de estos parámetros son: los coeficientes de los filtros digitales o las tablas de las curvas características.

Los cálculos en línea se realizan mientras el relevador esta en operación. Por medio de estos cómputos se determinan las magnitudes o fasores de corriente o voltaje actuales; estas magnitudes son obtenidas por medio de algoritmos de procesamiento digital de señales.

La lógica de disparo describe las comparaciones que se hacen durante la operación en línea del relevador, estas son necesarias para generar o no generar una señal de disparo. Las comparaciones se realizan con los resultados de los cálculos en línea (magnitud de una corriente o tensión) contra los valores de los ajustes del relevador.

CAPÍTULO 4

ANÁLISIS DEL ALGORITMO DE LA TRANSFORMADA DISCRETA DE FOURIER

En este capítulo se explican las principales diferencias en la forma de operación entre los relevadores electromecánicos, los de estado sólido y los digitales. Después, se hace el análisis matemático del algoritmo de la Transformada Discreta de Fourier, partiendo de las series de Fourier y de la Transformada de Fourier. Posteriormente, se presenta el análisis de la respuesta en frecuencia de los filtros Seno y Coseno.

4.1 Operación de los relevadores electromecánicos, de estado sólido y digitales

Los relevadores electromecánicos producen un par que es proporcional al cuadrado del flujo magnético producido por la corriente que circula por sus bobinas. Los mecanismos y funcionamiento de los relevadores electromecánicos son difíciles de analizar matemáticamente; usualmente, sus características son obtenidas por medio de datos experimentales. Estos relevadores son de respuesta rms (raíz cuadrática media), mientras que los relevadores de estado sólido o estáticos utilizan circuitos lineales y detectores de nivel para responder al valor pico de la señal entrante.

Los relevadores digitales operan de una manera diferente, estos usan filtros digitales para estimar el fasor de frecuencia fundamental de la señal entrante; además de atenuar o eliminar los armónicos de la señal. Un relevador digital tiene características analíticas que pueden ser descritas por medio de ecuaciones, lo que conlleva a poder calcular o simular la respuesta del relevador para señales específicas de entrada [10].

El algoritmo empleado para determinar la magnitud de la corriente de falla del relevador digital de sobrecorriente propuesto, corresponde a la transformada discreta de Fourier de ciclo completo. El Dr. Ramamoorty en un trabajo de 1971 fue el primero en proponer el uso de la Transformada Discreta de Fourier para calcular fasores de frecuencia fundamental desde muestras cuantizadas [13]. El algoritmo de la transformada discreta de Fourier es ampliamente usado en relevadores digitales de protección comerciales.

Como se verá más adelante, la combinación de la Transformada Discreta de Fourier y su inversa proporcionan las componentes real e imaginaria del fasor de frecuencia fundamental de una señal dada [1]. Una vez obtenidas la parte real e imaginaria de la señal de interés, es posible calcular su magnitud y fase, a través de operaciones básicas de números complejos. En la figura 4.1 se muestra la representación fasorial de una señal senoidal.

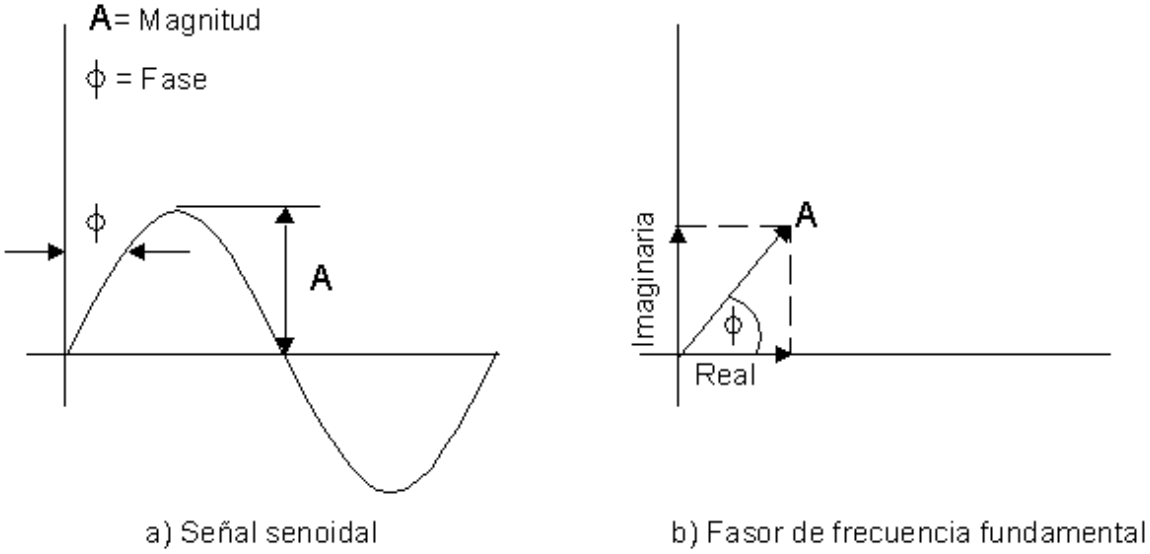


Figura 4.1: Representación fasorial de una señal senoidal.

4.2 Representación de una señal de voltaje en series de Fourier

Las señales de los sistemas de potencia como voltajes y corrientes son esencialmente señales periódicas. Idealmente los voltajes y corrientes presentes en el sistema en estado estacionario son puramente senoidales con una frecuencia de 60 Hz. Pero, los transformadores de potencia, los inversores, convertidores y algunas cargas crean distorsión armónica en las señales de estado estacionario. Las señales medidas por los relevadores de protección también pueden ser señales de frecuencia no fundamental [2].

La naturaleza de las señales de frecuencia no fundamental tienen una importante influencia en el desempeño de los relevadores de protección [10]. Las series de Fourier proveen una técnica para examinar estas señales y determinar su contenido armónico.

Una señal de voltaje que es considerada periódica, puede ser representada en series de Fourier de la siguiente manera:

$$v(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos(n\omega_o t) + \sum_{n=1}^{\infty} b_n \sin(n\omega_o t) \qquad (4.1)$$

Los valores de los coeficientes a_0, a_n, y b_n se obtienen a partir de:

$$a_0 = \frac{1}{T}\int_{t_0}^{t_0+T} v(t)\,dt \tag{4.2}$$

$$a_n = \frac{2}{T}\int_{t_0}^{t_0+T} v(t)\cos\left(n\omega_0 t\right)dt \quad n = 1,2,\ldots\infty \tag{4.3}$$

$$b_n = \frac{2}{T}\int_{t_0}^{t_0+T} v(t)sen\left(n\omega_0 t\right)dt \quad n = 1,2,\ldots\infty \tag{4.4}$$

Donde T es el periodo de la componente fundamental de la señal, $\omega_0 = 2\pi f_0$, y $f_0 = 1/T$.

4.3 Transformada de Fourier

Las series de Fourier son un caso especial de la transformada de Fourier y se emplean en el análisis de señales periódicas. Sin embargo, en la práctica, suelen aparecer señales no periódicas durante los transitorios de una falla [2].

Para ampliar el concepto de las series de Fourier a señales no periódicas, se considera una señal no periódica como una señal continua de periodo infinito, de esta forma el espacio entre frecuencias es cero y la señal no periódica se convierte en una función continua. Lo anterior resulta en la definición de la Transformada de Fourier [7], [9]. La esencia de la Transformada de Fourier es descomponer o separar una señal en una suma infinita de senoides de diferentes frecuencias.

La transformada de Fourier se define como:

$$X(f) = \int_{-\infty}^{\infty} x(t)e^{(-j2\pi ft)}dt \tag{4.5}$$

y su inversa esta dada por:

$$x(t) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(f)e^{(j2\pi ft)}dt \tag{4.6}$$

4.4 Transformada Discreta de Fourier

En aplicaciones de la Transformada de Fourier con computadoras digitales, las señales analógicas son discretizadas en un rango finito, y se recurre a la Transformada Discreta de Fourier (TDF), porque en una computadora no es realizable procesar las sumas infinitas que están implícitas en la definición de la Transformada de Fourier [9].

La Transformada Discreta de Fourier se expresa de la siguiente manera:

$$X(k) = \sum_{n=0}^{N-1} x(n) e^{-j 2\pi kn/N} \quad k = 0,1,2,\ldots,N-1 \tag{4.7}$$

y su inversa como:

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(k) e^{j 2\pi kn/N} \quad n = 0,1,2,\ldots,N-1 \tag{4.8}$$

Donde:

$x(n)$ señal digitalizada de una señal x(t)

$X(k)$ son los coeficientes de la TDF.

$$N = \frac{T}{t_s}$$

N= número de muestras

T es el periodo fundamental de la señal

t_s es el periodo de muestreo

$$t_s = \frac{1}{f_s}$$

f_s es la frecuencia de muestreo

$$T = \frac{1}{f}$$

f es la frecuencia fundamental de la señal.

Expandiendo la sumatoria dada en (4.8), agrupando términos y teniendo en cuenta que las señales muestreadas son reales, se llega a la ecuación (4.9).

$$x(n) = \frac{1}{N}\left[X(0) + 2|X(1)|\cos\left(\frac{2\pi n}{N} + \theta_1\right) + 2|X(2)|\cos\left(\frac{2\pi n}{N} + \theta_2\right) + \ldots \right] \qquad (4.9)$$

en esta ecuación θ_1 y θ_2,..., son los ángulos de $X(1)$, $X(2)$, respectivamente.

El término que reconstruye la componente de frecuencia fundamental de la señal *x(n)* esta dado por:

$$x_0(n) = \frac{2|X(1)|}{N}\cos\left(\frac{2\pi n}{N} + \theta_1\right) \qquad n = 0,1,2,\ldots,N-1 \qquad (4.10)$$

$|X(1)|$ y θ_1 se obtienen de la ecuación (4.7). Las componentes ortogonales de $x_0(n)$ se determinan con las ecuaciones (4.11) y (4.12).

$$X_0\ real = \frac{2}{N}\sum_{n=0}^{N-1} x(n)\cos\left(\frac{2\pi n}{N}\right) \qquad (4.11) \qquad X_0\ imag = -\frac{2}{N}\sum_{n=0}^{N-1} x(n) sen\left(\frac{2\pi n}{N}\right) \qquad (4.12)$$

X_0 *real* Representa el valor pico de la componente real del fasor de frecuencia fundamental de la señal $x(n)$.

X_0 *imag* Representa el valor pico de la componente imaginaria del fasor de frecuencia fundamental de la señal $x(n)$.

Por medio de las componentes real e imaginaria de la señal x(n), se puede obtener la magnitud (valor pico) de la señal x(n), y su fase a través de:

$$|X_0| = \sqrt{(X_0\ real)^2 + (X_0\ imag)^2} \qquad (4.13) \qquad \varphi_0 = arctg\frac{X_0\ imag}{X_0\ real} \qquad (4.14)$$

4.5 Prueba de ortogonalidad de los filtros Seno y Coseno:

En el algoritmo de la TDF las componentes ortogonales del fasor de frecuencia fundamental (componentes real e imaginaria), correspondientes a la entrada $x(n)$, se obtienen mediante la correlación de las muestras de $x(n)$ con un filtro Seno y un filtro Coseno, de acuerdo con las ecuaciones (4.11) y (4.12).

Los filtros Seno y Coseno son funciones ortogonales. Se considera que la mayoría de las señales pueden ser expresadas como la suma de una variedad de conjuntos de señales mutuamente ortogonales.

Dos funciones f(t) y g(t), son ortogonales sobre un periodo de tiempo T si:

$$\int_{t=0}^{T} f(t)g(t)dt = 0 \tag{4.15}$$

Si $f(t) = \cos(\omega t)$ y $g(t) = \sin(\omega t)$, se tiene:

$$\int_{t=0}^{T} f(t)g(t)dt = \int_{t=0}^{2\pi/\omega} \sin(\omega t)\cos(\omega t)dt$$

$$= \frac{1}{2}\int_{t=0}^{2\pi/\omega} \sin(2\omega t)dt = \frac{1}{2}\cdot\frac{1}{2\omega}\left|\cos(2\omega t)\right|_{0}^{2\pi/\omega} = 0 \tag{4.16}$$

$\therefore$ Las funciones Seno y Coseno son ortogonales.

4.6 Respuesta en frecuencia de los filtros Seno y Coseno:

Los filtros Seno y Coseno son Filtros de Respuesta Finita al impulso (FIR). Algunas propiedades de los filtros FIR son:

- No hay recursión, es decir, la salida sólo depende de la entrada y no de valores pasados de la salida.
- Su respuesta es una suma ponderada de valores pasados y presentes de la entrada.

- La función de transferencia tiene un denominador constante y sólo tiene ceros.
- La respuesta es de duración finita porque si la entrada se mantiene en cero durante *N* periodos consecutivos, la salida será también cero.
- Los filtros FIR son de fase lineal
- Los filtros FIR son siempre estables

Un filtro FIR se puede escribir como:

$$y(n) = \sum_{k=0}^{M-1} b_k \; x(n-k) \tag{4.17}$$

Donde: x*(n)* representa la señal de entrada de tensión o corriente, *y(n)* corresponde a la salida y b_k es el conjunto de coeficientes del filtro.

El número de coeficientes b_k está determinado por el tamaño de la ventana de datos. Una ventana de datos para un filtro FIR, corresponde a un número determinado de muestras. Entonces, para *N* muestras, se tendrán *N* coeficientes b_k. La ventana de datos es deslizante, ya que cada vez que entra una nueva muestra se descarta la más antigua y se calcula un nuevo valor de la señal de salida.

La respuesta en frecuencia del filtro representado por (4.17) se obtiene primero aplicándole la transformada Z:

$$Y(z) = \sum_{k=0}^{M-1} b_k X(z) z^{-k} \tag{4.18}$$

Luego se reacomoda (4.18) para expresarla como función de transferencia:

$$\frac{Y(z)}{X(z)} = H(z) = \sum_{k=0}^{M-1} b_k z^{-k} \tag{4.19}$$

Donde $Z = e^{j2\pi f t_s}$

Los coeficientes b_k del filtro Seno se calculan con:

$$b_k = sen\left(\frac{2\pi\, k}{N}\right) \qquad k = 0,1,2,...N-1 \tag{4.20}$$

Los coeficientes b_k del filtro Coseno se calculan con:

$$b_k = \cos\left(\frac{2\pi\, k}{N}\right) \qquad k = 0,1,2,...N-1 \tag{4.21}$$

N esta dado por:

$$N = \frac{T}{t_s}$$

Donde: t_s es el periodo de muestreo y f_s es la frecuencia de muestreo.

En la tabla 4.1: Se listan los coeficientes para los filtros Seno y Coseno con N=8.

Tabla 4.1: Coeficientes para los filtros Seno y Coseno con N=8.

K	b_k Seno	b_k Coseno
0	0	1.0
1	0.7071	0.7071
2	1	0
3	0.7071	-0.7071
4	-0	-1
5	-0.7071	-0.7071
6	-1	-0
7	-0.7071	0.7071

La respuesta en frecuencia para el filtro Seno, considerando N=8, se obtiene al sustituir (4.20) en (4.19), y da como resultado la ecuación (4.22):

$$\begin{aligned} H(z) = \quad & 0.0 * z^{0} + \ 0.7071 * z^{-1} + \ 1 * z^{-2} + \ 0.7071 * z^{-3} - 0.0 * z^{-4} \\ & - 0.7071 * z^{-5} - 1 * z^{-6} - 0.7071 * z^{-7} \end{aligned} \tag{4.22}$$

La respuesta en frecuencia del filtro Seno se observa en la figura 4.2.

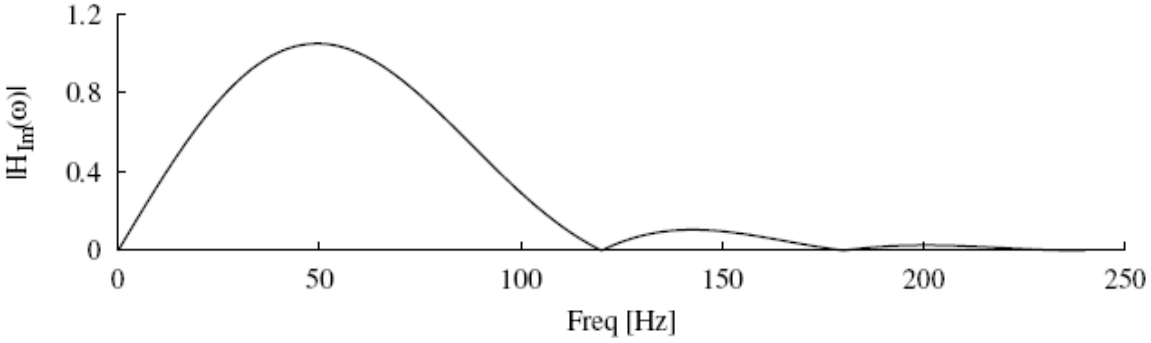


Figura 4.2: Respuesta en frecuencia del filtro Seno.

La respuesta en frecuencia para el filtro Coseno, considerando N= 8, se obtiene al sustituir (4.20) en (4.19), y da como resultado la ecuación (4.23):

$$H(z) = 1*z^{0} + 0.7071*z^{-1} + 0.0*z^{-2} - 0.7071*z^{-3} - 1*z^{-4} - 0.7071*z^{-5} - 0.0*z^{-6} + 0.7071*z^{-7} \quad (4.23)$$

La respuesta en frecuencia del filtro Coseno se observa en la figura 4.3

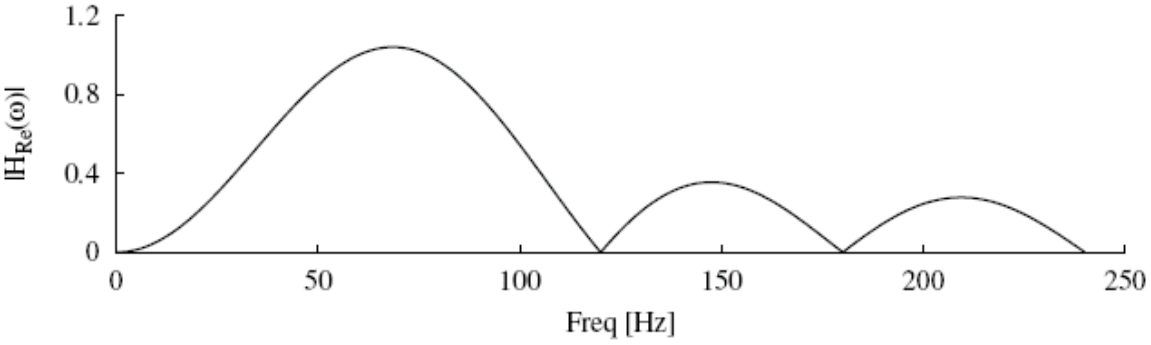


Figura 4.3: Respuesta en frecuencia del filtro Coseno.

La respuesta en frecuencia de los filtros Seno y Coseno presenta ceros en los múltiplos enteros de la frecuencia fundamental y para la componente no decreciente de corriente continua, de modo que los filtros Seno y Coseno rechazan todas estas componentes.

Una de las desventajas del algoritmo de Fourier es el error que se produce con las señales exponenciales decrecientes de corriente continua, que se presentan en condiciones de falla. La magnitud calculada puede desviarse del valor real en más del 15% para los casos peores [26].

CAPÍTULO 5

IMPLEMENTACIÓN DEL ALGORITMO DE LA TRANSFORMA DISCRETA DE FOURIER EN EL MICROCONTROLADOR HCS12E64 DE MOTOROLA

En este capítulo se discute el uso de un microcontrolador como alternativa en el diseño de un relevador digital, y se advierten las ventajas y desventajas del lenguaje Ensamblador frente al lenguaje C como lenguaje de programación de los algoritmos de los relevadores digitales. También, se explica una metodología para la implementación de la Transformada Discreta de Fourier. Asimismo, se describe el circuito de acondicionamiento de señales empleado en el prototipo y se analiza el programa final para el microcontrolador HCS12E64.

5.1 ¿Por qué usar un microcontrolador para un relevador digital?

Aunque es más recomendable emplear un DSP en el diseño de un relevador digital, por sus instrucciones particulares para el procesamiento digital de señales, y su capacidad de cómputo paralelo, los microcontroladores son una alternativa para fines educativos y de experimentación en el área de los relevadores digitales. Los DSPs suelen ser más costosos, y un mal manejo puede dañarlos, repercutiendo económicamente en un proyecto experimental.

En el avance de la tecnología de los microcontroladores, se han mejorado las velocidades de procesamiento y ampliado la cantidad de memoria y periféricos disponibles. Además, hoy en día se dispone de una mayor cantidad de información técnica acerca de los microcontroladores.

Además, las herramientas de desarrollo actuales hacen uso de plantillas de código que permiten concentrarse más en la estructuración del programa, evitándose la configuración de archivos e inicialización de periféricos, ahorrando con esto, tiempo y esfuerzo. De igual forma, los depuradores en tiempo real de las herramientas de desarrollo son muy útiles en la fase de pruebas de los algoritmos.

Por lo tanto, resulta práctico utilizar microcontroladores en relevadores digitales, donde no se requieran características a las soportadas por microprocesadores especializados como los DSPs.

5.2 Lenguaje de programación utilizado

En los inicios de los relevadores digitales el lenguaje de programación utilizado era el lenguaje Ensamblador. El lenguaje Ensamblador resultaba conveniente por varias razones: una de ellas era el tiempo de ejecución de las instrucciones; las instrucciones en ensamblador se ejecutan en un menor número de ciclos de reloj en comparación con las instrucciones de lenguajes de programación de medio y alto nivel como el lenguaje C [2].

El tiempo de ejecución de las instrucciones era importante porque la velocidad de los microprocesadores estaba limitada. En tal caso, como mejor opción, se usaba el lenguaje Ensamblador para resolver los inconvenientes de realizar cálculos y decisiones en tiempo real en estos microprocesadores.

Otra razón del uso de lenguaje Ensamblador, es que este esta más apegado al funcionamiento interno de los microprocesadores, y por lo tanto, se tiene un mayor control sobre sus recursos. Sin embargo, el lenguaje Ensamblador esta basado en mnemónicos y se requiere de un conocimiento especializado de estos para cada microprocesador en particular. Como consecuencia, el transporte de código es difícil o imposible con el lenguaje Ensamblador.

La programación del algoritmo del relevador digital propuesto se realizó en lenguaje C, por las múltiples ventajas que ofrece este lenguaje de programación en comparación con el lenguaje Ensamblador.

La principal ventaja del Lenguaje C, frente al lenguaje Ensamblador, es su naturaleza para la programación estructurada. La programación estructurada facilita la comprensión de la lógica de un programa. Y algunas de las librerías comunes del lenguaje C incorporan útiles funciones matemáticas que ahorran la programación de subrutinas complejas. Además, el código fuente de un programa en lenguaje C para un microprocesador en particular, puede usarse en otros diferentes haciendo sólo algunos cambios, esto debido a las características de transporte del lenguaje.

En programas complejos muchas veces se recurre a combinar el lenguaje C con el lenguaje Ensamblador con el fin de optimizar operaciones críticas [7]. Finalmente, la depuración de un programa en lenguaje C, es más sencilla y comprensible que la depuración de programas en lenguaje Ensamblador.

5.3 Descripción del microcontrolador MCHS12E64

En este proyecto se usó el microcontrolador MC9S12E64 de Motorola. El microcontrolador MC9S12E64, es un microcontrolador de bajo costo de uso general. Su unidad de procesamiento central (HCS12 Cpu) es de 16 bits, opera a 50 Mhz equivalente a 25 Mhz en velocidad de Bus. El tiempo mínimo de ejecución de una instrucción es de 40 nanosegundos. Al igual que otros microcontroladores, su arquitectura está optimizada para compiladores de lenguaje C.

Cuenta también con 64 Kilobytes de memoria Flash EEPROM, que puede ser borrada y escrita hasta 100, 000 veces, y 4 Kilobytes de memoria RAM; en este microcontrolador 92 de las 112 patas se pueden emplear como entradas y salidas digitales. Además, la incorporación de un circuito PLL permite ajustar el consumo de energía del microcontrolador, dependiendo de los requerimientos de operación.

Algunas de las características del microcontrolador MC9S12 E64 son:

- HCS12 CPU

Modo de depuración en segundo plano con un único cable (BDM)

Soporte para instrucciones de lógica difusa

Control de interrupciones

Múltiples puntos de ruptura por hardware

- 16 entradas de interrupción del tipo "Wake up" con filtrado digital
- Dos convertidores digitales analógicos (DAC) de 1 canal, resolución de 8 bits
- Convertidor analógico digital (ADC) de 16 canales, resolución de 8 y 10 bits, tiempo mínimo de conversión de 7micro segundos.
- Tres temporizadores de 4 canales
- Dos moduladores de ancho de pulso (PWM)
- Tres interfases de comunicación serial asíncrona (SCI)
- Interfase periférica serial sincronía (SPI)
- Bus I2C
- Reloj y generador de reset
- Interrupción en tiempo real
- Monitor de reloj
- COP
- Regulador interno de 2.5 volts

En la figura 5.1 se muestra el diagrama a bloques de la familia MC9S12E.

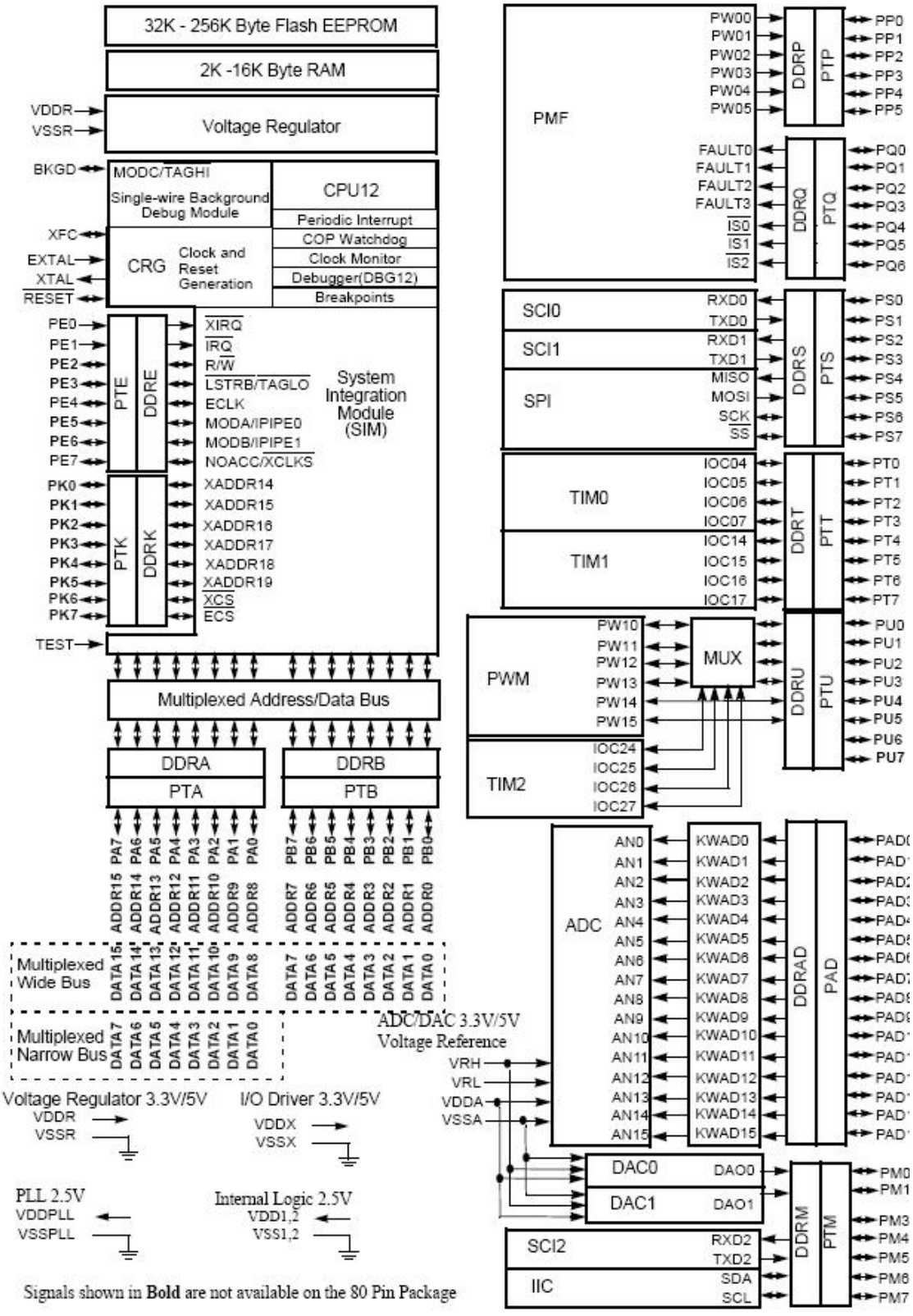


Figura 5.1: Diagrama a bloques de la familia MC9S12E

5.4 Metodología para la implementación del algoritmo de la transformada discreta de Fourier

El proceso para implementar el algoritmo de la TDF se divide en dos partes:

1. Selección de la frecuencia de muestreo y determinación de los coeficientes de los filtros Seno y Coseno.

2. Correlación de las muestras de la señal con los coeficientes de los filtros Seno y Coseno.

5.4.1 Selección de la frecuencia de muestreo y determinación de los coeficientes de los filtros Seno y Coseno

La frecuencia de muestreo debe de ser como mínimo el doble de la frecuencia de la señal de interés (Criterio de Nyquist). Se eligió para este caso una frecuencia de muestreo de 480 Hz. Con esta frecuencia de muestreo se tienen 8 muestras en un ciclo de una señal con frecuencia de 60 Hz. Las 8 muestras tomadas en un ciclo de la señal forman la ventana de datos que se utiliza en la estimación del fasor de la señal de corriente.

Los coeficientes de los filtros Seno y Coseno se obtienen con las ecuaciones (4.20) y (4.21). En la Tabla 4.1 están todos los coeficientes para los filtros Seno y Coseno, tomando en cuenta N=8 (N= 480/60).

5.4.2 Correlación de las muestras de la señal con los coeficientes de los filtros Seno y Coseno.

Como se mencionó en el capítulo 4, con la correlación de las muestras de la señal con los filtros Seno y Coseno se obtienen las componentes ortogonales de la señal. Esta correlación se desarrolla de la siguiente manera:

- La primera muestra de la señal se multiplica con el primer coeficiente de los filtros Seno y Coseno o señales de referencia.

- Los resultados serán la parte real e imaginaria de una fracción del fasor total de la señal, estos resultados se almacenan en dos variables diferentes tipo arreglo.

- Se repiten las multiplicaciones, ahora se multiplica la segunda muestra con el segundo coeficiente de los filtros Seno y Coseno. Los resultados de estas segundas multiplicaciones se suman a los resultados de las primeras multiplicaciones.
- Siguiendo la lógica anterior, los resultados de los productos previos son sumados a los resultados de las multiplicaciones siguientes hasta considerar todas las muestras o datos de la ventana.

- Los resultados de los productos desde el primero hasta el último, se dividen entre 2/N (4 para este caso) de acuerdo con las ecuaciones (4.11) y (4.12).

- Al final, se tendrán las componentes real e imaginaria del fasor total de la señal de interés.

- Para calcular la magnitud y fase del fasor se emplean las ecuaciones (4.13) y (4.14). Estos cálculos se realizan con cada resultado de la correlación entre muestras y coeficientes de los filtros.

Cuando se tienen ocho muestras almacenadas estas se recorren de lugar como se indica en la figura 5.2.

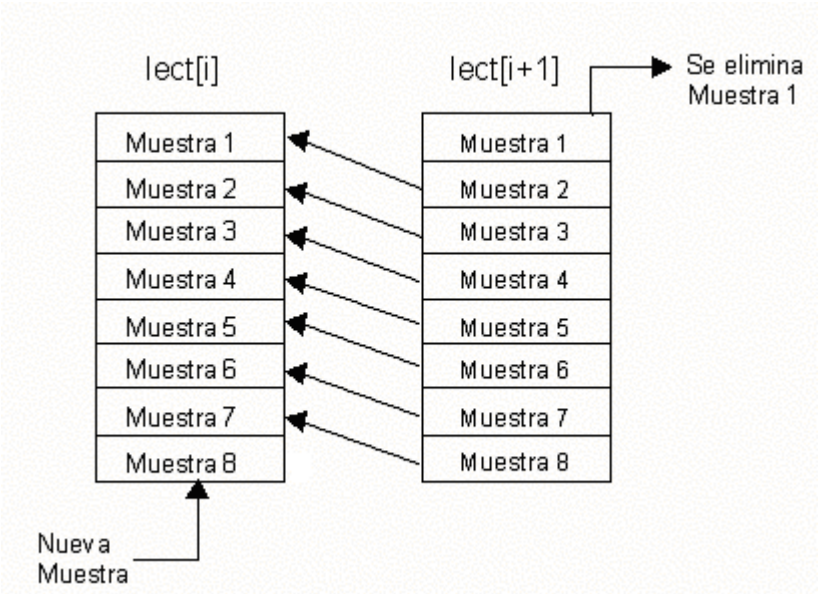


Figura 5.2: Recorrido de muestras

En la figura 5.3 se aprecia gráficamente como se recorren las muestras.

Al recorrer las muestras se elimina la primera muestra y se queda vacía la localidad de la octava muestra, después se toma otra muestra que pertenece al segundo ciclo de la señal, y se almacena en la octava localidad , de esta forma se completa la ventana de las muestras y se continúan los cálculos de la parte real e imaginaría de la señal.

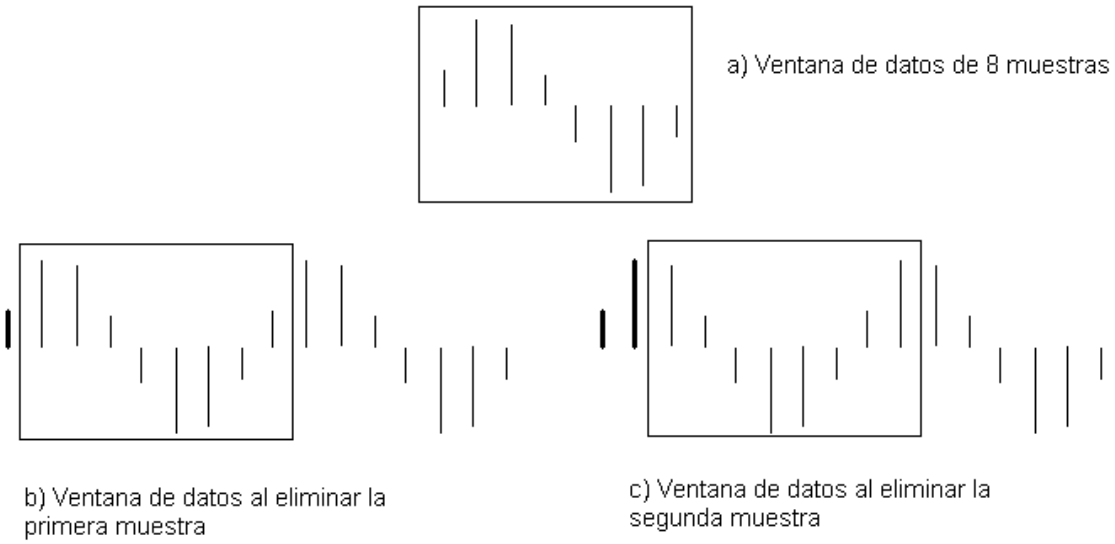


Figura 5.3: Ventanas de datos de ocho muestras.

En la figura 5.4 se muestra gráficamente como se realizan los cálculos para obtener las componentes ortogonales de una señal de corriente alterna de 60 Hz muestreada a 480 Hz.

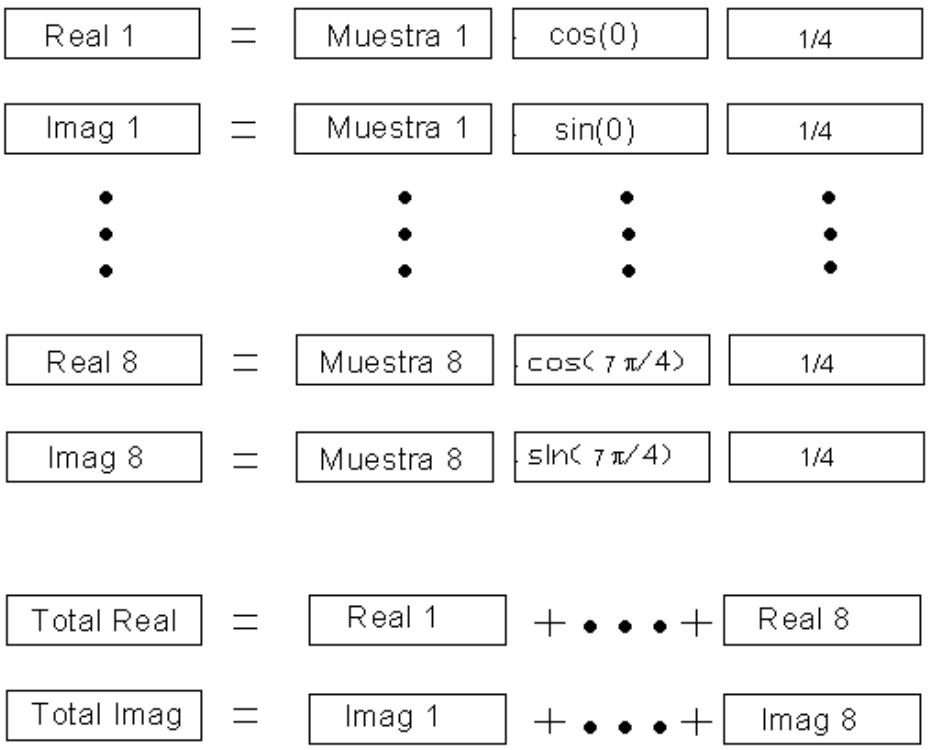


Figura 5.4: Esquema de los cálculos para determinar las componentes ortogonales de una señal de corriente alterna de 60 Hz muestreada a 480 Hz.

5.5 Acondicionamiento de señales

En el diseño del circuito acondicionar de señales se consideraron los valores mínimos y máximos de tensión permitidos por el convertidor analógico digital del microcontrolador HCS12E64. El convertidor analógico digital del HCS12E64 sólo acepta señales analógicas que varíen dentro del rango de 0 a 5 volts; lo que significa que no se pueden realizar la conversiones de señales senoidales directamente.

Para lograr la conversión analógica digital de una señal senoidal cuando un convertidor analógico digital sólo acepta tensiones positivas, se atenúa la señal senoidal y se suma una tensión de corriente continúa positiva a la señal senoidal para desplazar la señal senoidal hacia un rango de valores positivos. Posteriormente, en el programa del relevador se elimina el valor de la tensión de corriente continua. El valor resultante de la suma de la señal senoidal y la señal de corriente directa no debe de exceder el rango de tensión permitido por el convertidor analógico digital.

El circuito acondicionador de señales que se utilizó se muestra en la figura 5.5. En el circuito hay un amplificador en configuración de seguidor de tensión y un amplificador operacional en configuración de sumador. En este circuito se suman dos señales: una señal de 0.6 volts RMS de corriente alterna (VCA) y una señal 2.4 volts de corriente directa (VCD). En la Figura 5.6 se puede apreciar el oscilograma de la señal de salida del circuito de acondicionamiento de señales.

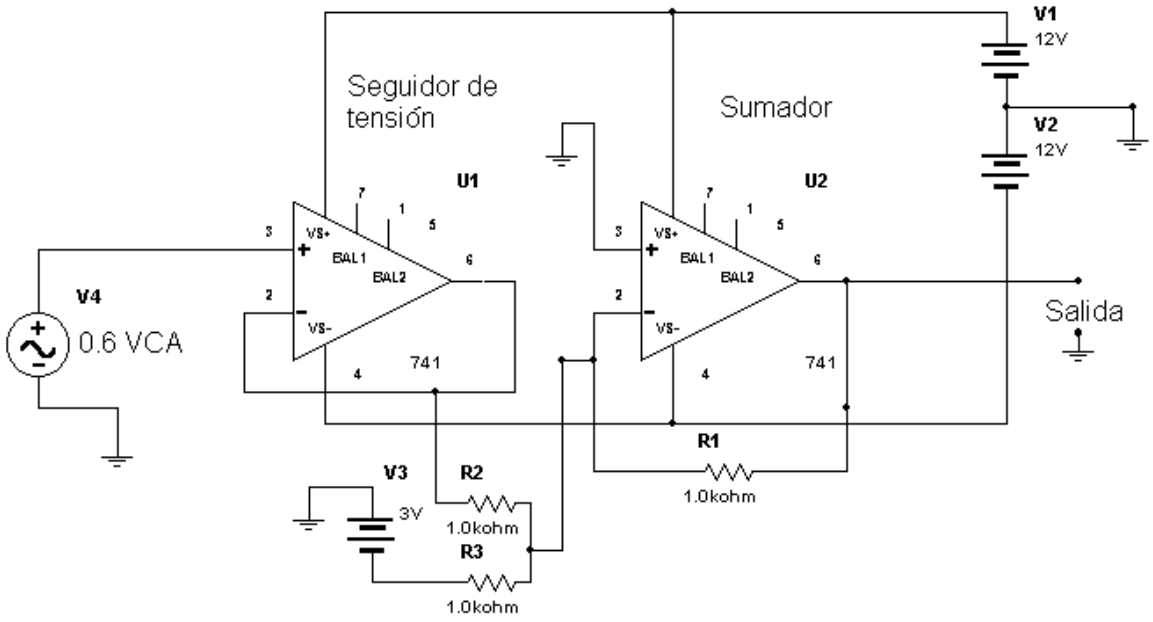


Figura 5.5: Circuito acondicionador de señales.

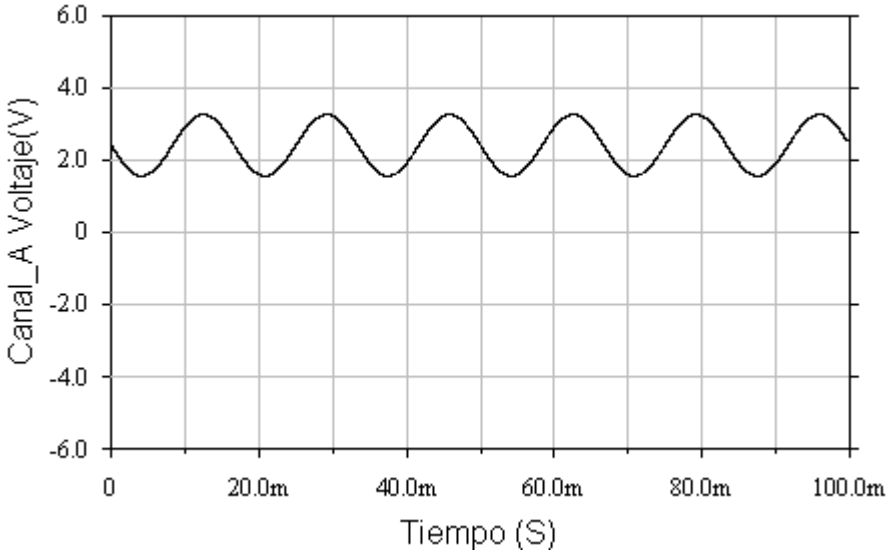


Figura 5.6: Oscilograma de la señal de salida del circuito acondicionador de señales (0.6 V.C.A + 2.4 V.C.D.).

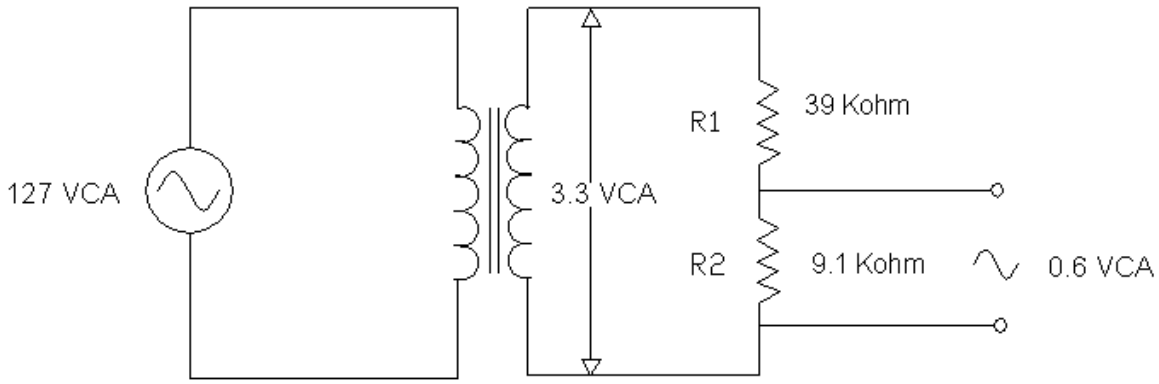


Figura 5.7: Circuito para generar la señal de prueba de 0.6 VCA.

Cabe mencionar que la señal de 0.6 VCA se tomo como señal de prueba, y no corresponde a la señal de corriente de algún circuito en particular. Esta señal de 0.6 VCA se obtuvo por medio de un transformador reductor de tensión y divisores de tensión como se muestra en la figura 5.7.

5.6 Descripción del programa para el microcontrolador HCS12E64

El programa se compone de:

A) Una rutina principal donde se calcula la parte real e imaginaria de la señal de corriente, por medio del la Transformada Discreta de Fourier.

B) Módulo de eventos para manejar las interrupciones necesarias.

C) Subrutina para la curva característica de tiempo inverso.

D) Subrutina de comunicación de datos (Puerto serial RS-232E).

E) Subrutina para el control del teclado matricial.

F) Subrutina para el control de la pantalla de cristal liquido (LCD).

En el figura 5.8 se muestra el diagrama de flujo del algoritmo. La lógica del algoritmo en forma resumida es la siguiente:

Primero se inicializan los periféricos del microcontrolador para poder trabajar con ellos. Posteriormente, el convertidor analógico digitaliza la señal de salida del circuito de acondicionamiento señales en razón de una frecuencia de muestreo de 480 Hz.

En cada intervalo de muestreo se envía la muestra tomada hacía una computadora personal (PC) de monitoreo, y se calculan las componentes real e imaginaria de la señal de corriente. En el programa las componentes real e imaginaria son elevadas al cuadrado y se almacenan en las variables A y B, la suma de estas dos variables se almacena en la variable C.

De la misma manera, en cada periodo de muestreo, la variable C se compara contra el cuadrado de la corriente de arranque (constante Id). Si C es mayor que Id se incrementa el contador llamado *Falla*. Al valer tres el contador *Falla,* se considera que hay una falla por sobrecorriente, e inmediatamente se interrumpe el muestreo de la señal y se genera una señal de disparo con un retraso definido por la ecuación de la curva característica de tiempo inverso de la norma IEC 6025-3.

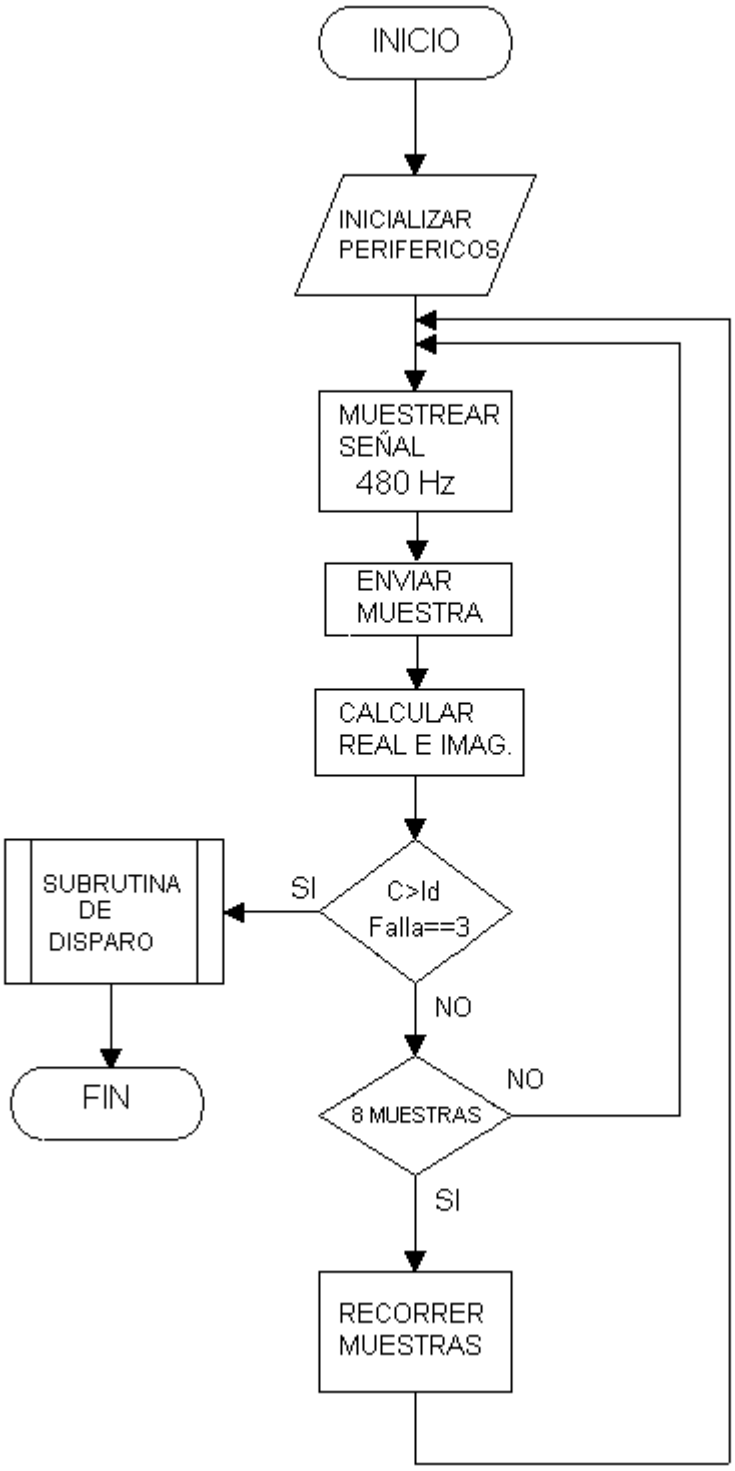


Figura 5.8: Diagrama de flujo del algoritmo

En el diagrama de flujo mostrado en la figura 5.8 se puede observar que se ha usado un procedimiento indirecto en la detección de sobrecorrientes, porque se usa el valor de la variable C y no la raíz cuadrada de este valor, para decidir si existe una sobrecorriente. La situación anterior se discute ampliamente en el capítulo 6.

Como se menciona en el apéndice A, se usó el Ambiente de Desarrollo Integrado (IDE) de Code Warrior para la edición, compilación y depuración del código de del algoritmo, y el Processor Expert para la familia de microcontroladores HCS12 como herramienta auxiliar en la generación del código correspondiente a la inicialización y configuración de periféricos. Todas las configuraciones de los periféricos utilizados se localizan en el apéndice A.

El Processor Expert además de generar el código de inicialización y configuración de los periféricos, crea un módulo especial llamado “Events.c”; en este módulo se encuentran las funciones que se solicitan cuando ocurre una interrupción por software o hardware. En la figura 5.9 se observa gráficamente como interactúa la función o rutina principal con el módulo “Events.c” y las subrutinas del programa.

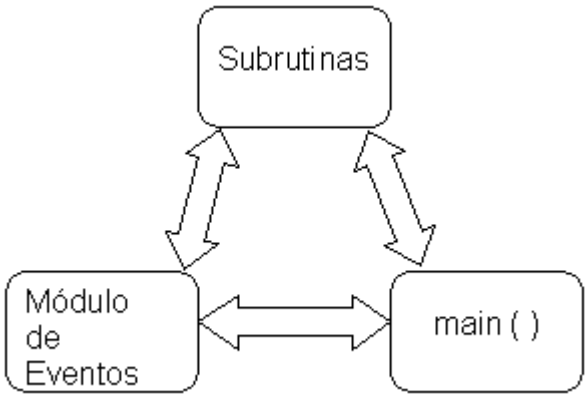


Figura 5.9: Interacción de la rutina principal con las subrutinas y el módulo de Eventos.

5.6.1 Generación de retardos en el programa

La base de operación del programa son tres interrupciones periódicas: una interrupción periódica principal y dos interrupciones periódicas secundarias. La interrupción periódica principal establece el periodo de muestreo de la señal y es invocada por la función principal.

Una interrupción periódica secundaria se encarga del control del tiempo de activación de la señal de disparo. Esta interrupción es habilitada en la subrutina de disparo cuando se ha detectado una falla por sobrecorriente.

La tercera interrupción periódica se utiliza en los retardos requeridos para inicializar la pantalla de LCD. En la figura 5.10 se ilustran los diagramas de flujo para generar retardos de acuerdo con el valor del variable *tiempo*. Estos diagramas de flujo son comunes para los retardos mencionados arriba. La variable contador es reducida una unidad en el módulo de eventos en intervalos iguales al valor en tiempo de la interrupción periódica correspondiente en cada caso (diagrama de flujo B). Cuando el valor de contador se hace cero, finaliza la subrutina de retardo.

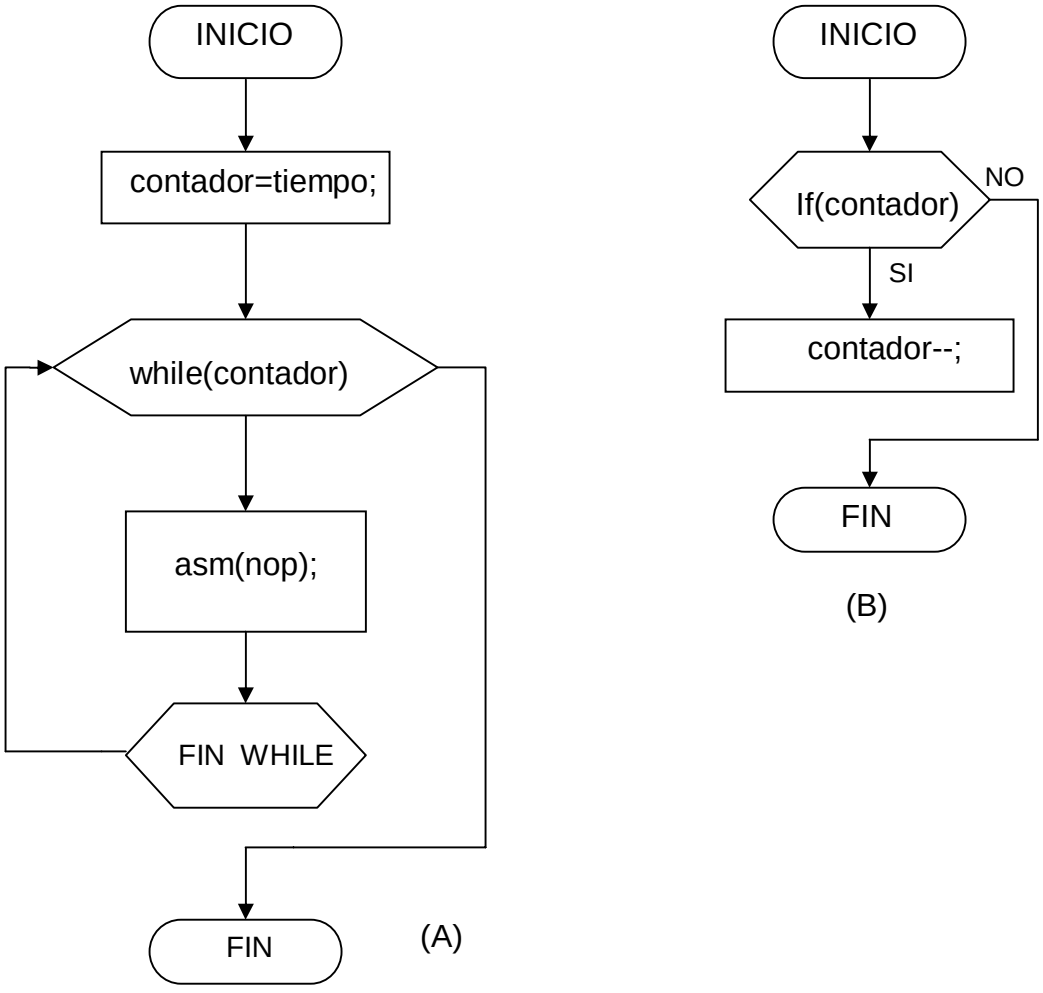


Figura 5.10: Diagramas de flujo de las subrutinas que generan los retardos del programa

Estas tres interrupciones periódicas junto con la interrupción inherente del convertidor analógico digital y las interrupciones propias de la comunicación de datos, son manejadas en el módulo de eventos.

5.6.2 Rutina principal

En la rutina principal se calculan las componentes real e imaginaria de la señal de corriente y se comprueba sí existe un incremento anormal en la corriente medida. En la figura 5.11 se muestra el diagrama de flujo del procedimiento que se utiliza en el cómputo de las componentes ortogonales de la señal de corriente.

En el diagrama de flujo de la figura 5.11 se tiene un contador llamado “K”, este contador se incrementa cada vez que se toma una muestra de la señal de corriente. Cuando K vale ocho, se tendrán ocho muestras almacenadas en la variable tipo arreglo “lect”.

Una vez que se han tomado ocho muestras, estas se recorren de lugar. Al recorrer las muestras se elimina la primera muestra que se tomó, entonces, se toma una nueva muestra para completar ocho muestras y se procede con el cálculo de las componentes real e imaginaria de la señal.

Al finalizar los cálculos de las componentes ortogonales habrá ocho muestras almacenadas, después, sólo es necesario guardar la siguiente muestra en “lect[7]” ; en tal caso, se fija K con el valor siete y al tomarse una nueva muestra, K se incrementará y el programa regresará a recorrer las muestras y calcular las componentes correspondientes. Este procedimiento se repite indefinidamente hasta que se cumple la condición para atender la subrutina de disparo.

5.6.3 Subrutina de disparo

Cuando en el programa se ha detectado una sobrecorriente, se deshabilita la interrupción periódica principal que fija el periodo de muestreo de la señal y se ejecutan los cálculos pertinentes para obtener el tiempo de disparo. El tiempo de disparo es inversamente proporcional a la corriente medida; a mayor corriente menor será el tiempo de disparo.

Después de obtener el tiempo de disparo, se habilita una interrupción periódica secundaria con la finalidad de generar el retardo correspondiente al tiempo de disparo. Al cumplirse el tiempo de disparo, se activa un salida digital del microcontrolador, esta salida puede emplearse para controlar la apertura de un interruptor de potencia. Los diagramas de flujo de la subrutina de disparo y del procedimiento para detectar cuando se presenta una sobrecorriente, se muestran respectivamente en las figuras 5.12 y 5.13.

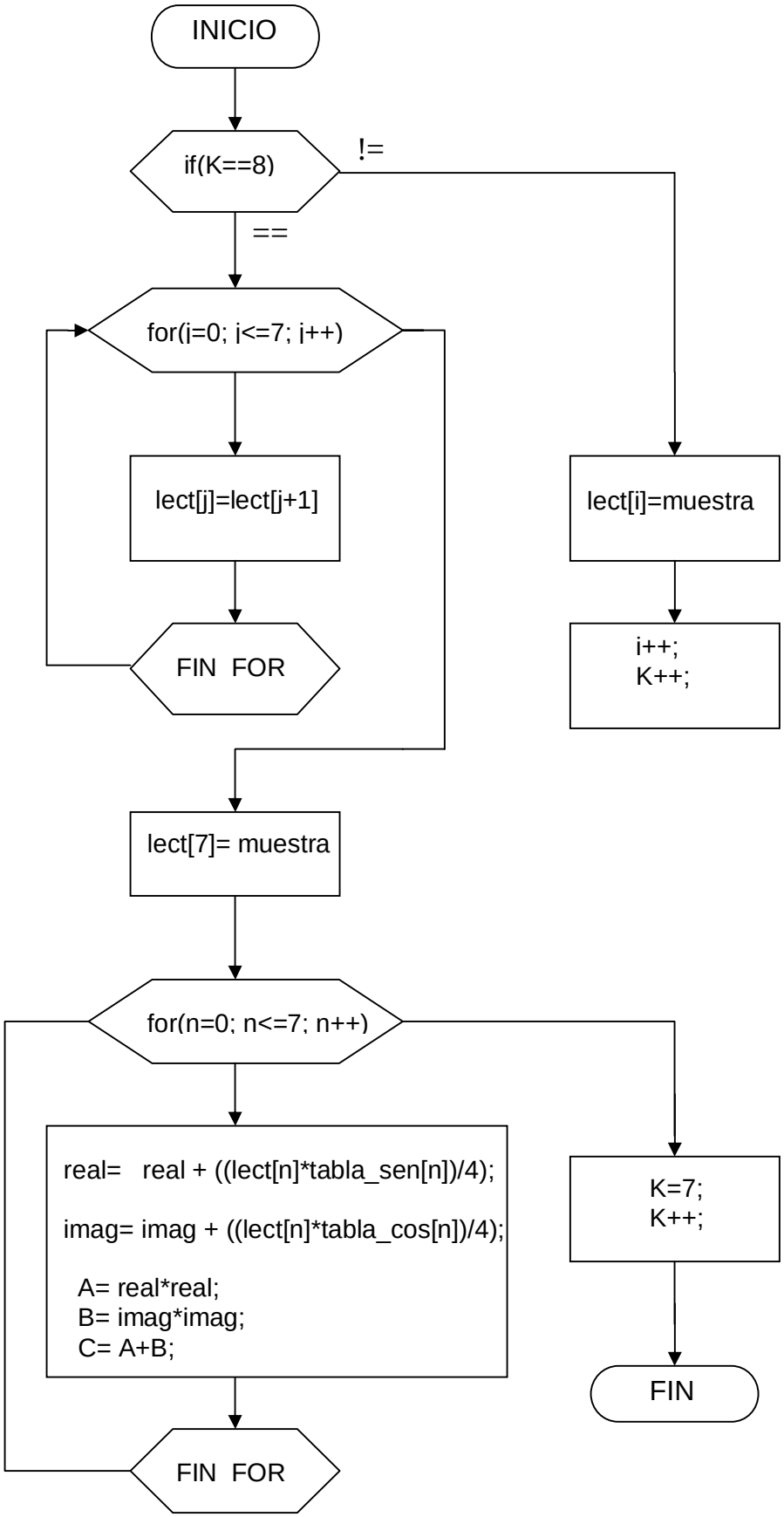


Figura 5.11: Diagrama de flujo de la rutina principal

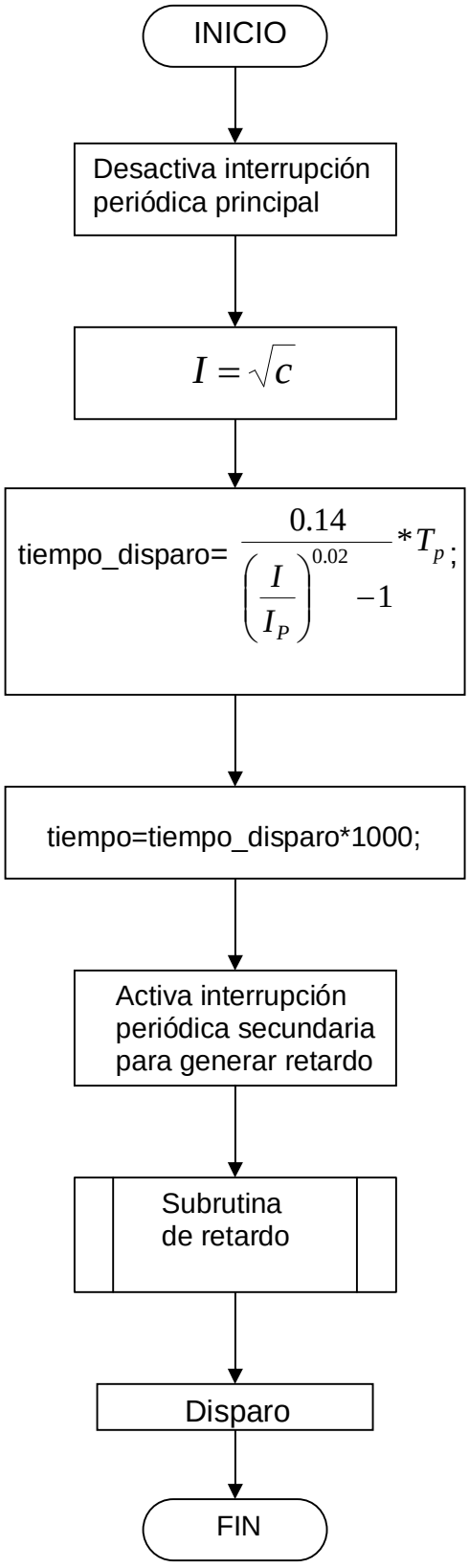


Figura 5.12: Diagrama de flujo de la subrutina de disparo

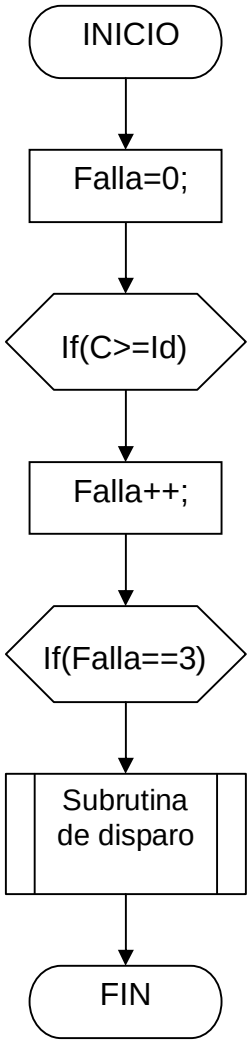


Figura 5.13: Diagrama de flujo del procedimiento usado en la detección de fallas por sobrecorriente

5.6.4 Comunicación de datos

En programa se contempló la comunicación de datos entre el relevador y una computadora personal (PC). La PC recibe vía puerto serial cada una de las muestras que se toman de la señal de corriente. En la PC se despliega esta información a través de un instrumento virtual creado en LabVIEW. Desde este instrumento virtual también se puede ajustar el valor de la corriente de arranque del relevador. La figura 5.14 muestra el instrumento virtual usado en este prototipo y en la figura 5.15 se observa su diagrama de bloques.

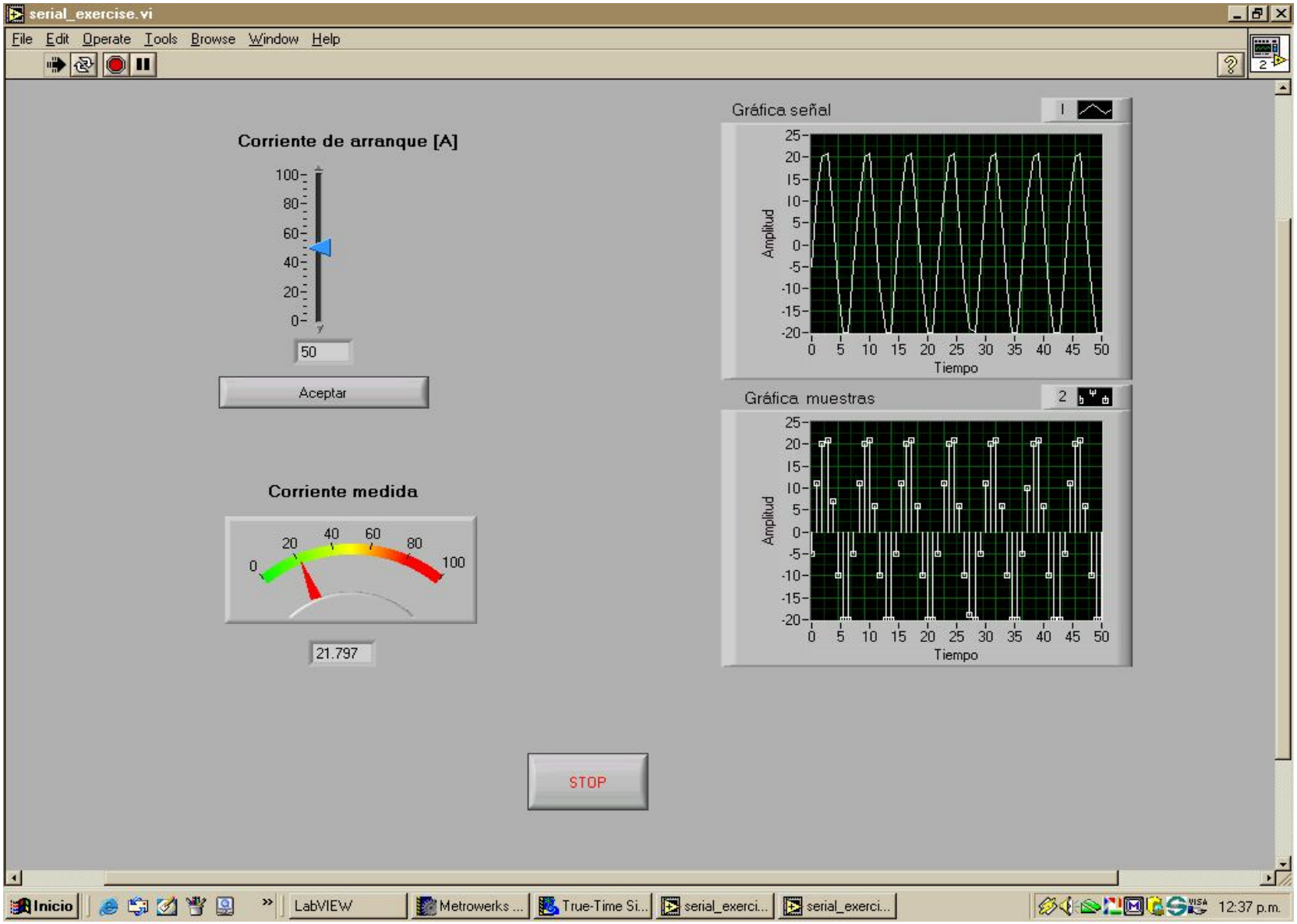


Figura 5.14: Vista del Instrumento virtual usado en el prototipo.

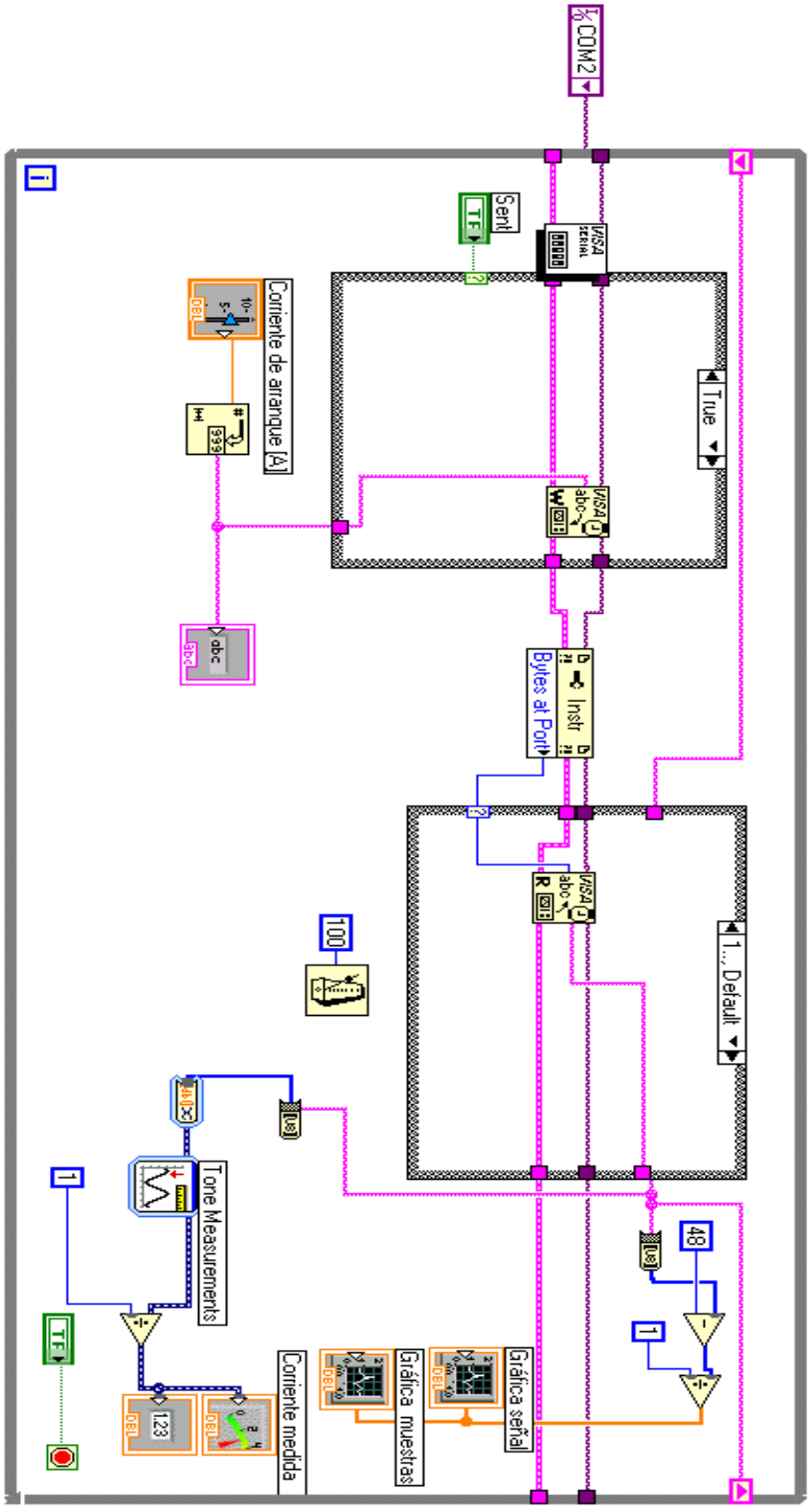


Figura 5.15: Diagrama de bloques del instrumento virtual.

5.6.5 Teclado matricial y pantalla de LCD

Por medio del teclado matricial, al igual que con la interfaz gráfica, se ajusta la corriente de arranque del relevador. El valor de la corriente ingresada desde el teclado se despliega en la pantalla de LCD.

En la subrutina de control del teclado no se usa el método de exploración continua en la localización de teclas presionadas, ya que este método requiere del uso continuo de la CPU del microcontrolador y no permite la ejecución de otras tareas. En este caso, se usan interrupciones por hardware o externas. Las interrupciones externas son apropiadas para evitar la exploración continua del teclado, ahorrando con esto, tiempo de procesamiento. En la figura 16 se muestra la tarjeta de desarrollo usada en el prototipo.

Figura 5.16: Vista de la tarjeta de desarrollo usada en el prototipo.

CAPÍTULO 6

PRUEBAS DEL ALGORITMO Y RESULTADOS

En este capítulo se mencionan las pruebas de desempeño del algoritmo realizadas y los resultados obtenidos. Principalmente, se compara el tiempo de procesamiento del algoritmo para dos frecuencias de muestreo: 480 Hz. y 720 Hz.

6.1 Tiempo de procesamiento del algoritmo

El algoritmo se tiene que procesar en tiempo real, esto significa que todas las operaciones involucradas en el algoritmo se tienen que ejecutar dentro del intervalo del periodo de muestreo de la señal; de lo contrario, si se rebasa este límite, se perderán muestras de la señal y como consecuencia, los cálculos serán erróneos ocasionando falsas operaciones del relevador. En la figura 6.1 se observa gráficamente el intervalo de ejecución del algoritmo para una frecuencia de muestreo de 480 Hz.

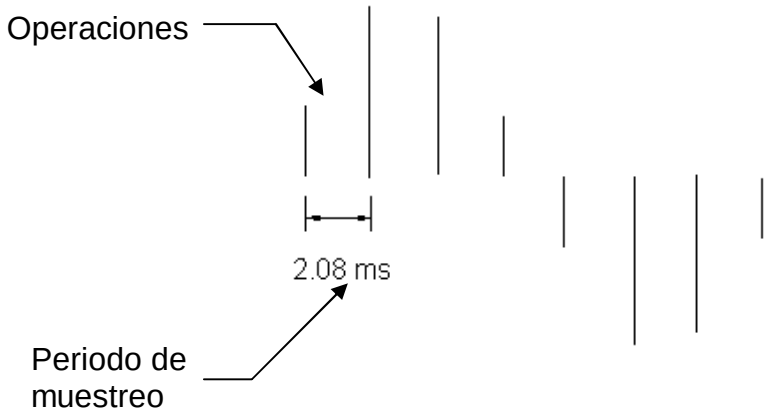


Figura 6.1: Intervalo de ejecución del algoritmo.

En un principio, en la codificación del algoritmo se usaron constantes y variables de tipo flotante para las operaciones relacionadas con la obtención de la magnitud de la señal de corriente, y se consideró una frecuencia de muestreo de 720 Hz. Con una frecuencia de muestreo de 720 Hz., el número de coeficientes necesarios de los filtros Seno y Coseno es de 12; por lo tanto, se requiere que en un ciclo de 60 Hz. de la señal de corriente, se repitan 12 veces las operaciones del algoritmo.

Durante la fase de pruebas de la comunicaciones vía puerto serial RS-232E, con las consideraciones anteriormente mencionadas, se descubrió que había pérdida de muestras de la señal al establecer la transmisión de datos del microcontrolador hacia la PC de monitoreo.

En las figura 6.2 se pueden apreciar las gráficas de los datos recibidos en la PC de monitoreo usando una frecuencia de muestreo de 720 Hz. y constantes y variables de tipo flotante.

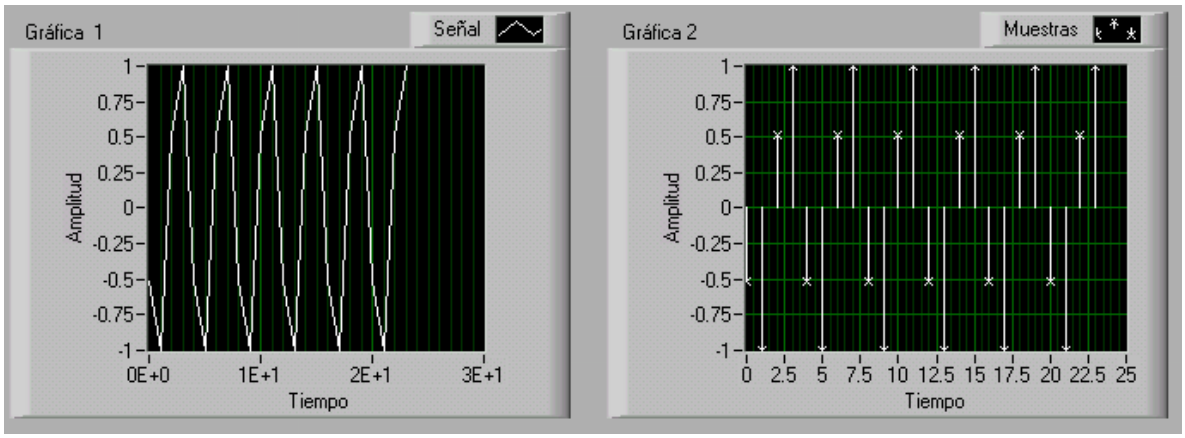


Figura 6.2: Gráficas de los datos recibidos cuando se utiliza una frecuencia de muestreo de 720Hz y constantes y variables tipo flotante.

En la figura 6.2 se observa que se han perdido cuatro muestras por semiciclo. En la figura 6.3 se tienen las 12 muestras por ciclo previstas; en este caso, se utilizan constantes y variables tipo entero.

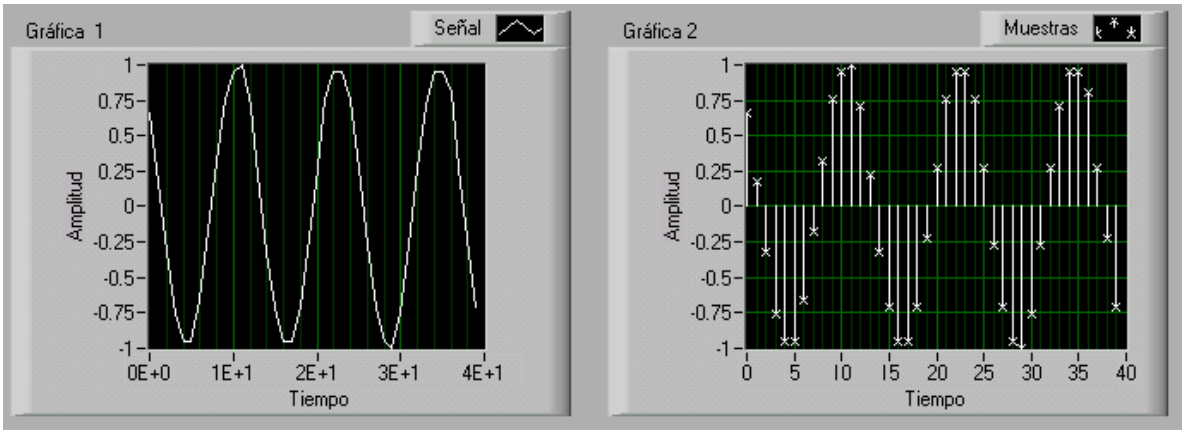


Figura 6.3: Gráficas de los datos recibidos cuando se utiliza una frecuencia de muestreo de 720Hz y constantes y variables tipo entero.

De la observación de la figura 6.2, se deduce que en este caso, el algoritmo no se procesa en tiempo real cuando se recurre a constantes y variables de tipo flotante, porque hay pérdida de datos en la comunicación entre el microcontrolador y la PC de monitoreo.

Aunque en la figura 6.3 se observa que la ejecución del algoritmo con constantes y variables tipo entero no obstruye la comunicación de datos, se tiene como desventaja que las variables y constantes tipo entero limitan la exactitud de las mediciones del relevador. Con formatos de tipo entero no se pueden representar números decimales. Por ejemplo, el número 0.8 será redondeado a el número 0 usando el formato tipo entero.

Como el objetivo es manejar constantes y variables de tipo flotante para tener una mayor exactitud en los cálculos de la magnitud de la señal de corriente, fue necesario precisar el tiempo de ejecución de las instrucciones del algoritmo, para identificar las operaciones críticas que interferían en la transmisión de datos.

6.2 Estimación del tiempo de procesamiento del algoritmo

El tiempo de procesamiento requerido por las instrucciones del algoritmo, se obtuvo por medio del depurador incluido en el Ambiente de Desarrollo Integrado (IDE) de CodeWarrior™ para el microcontrolador HCS12E64. La prueba consistió en asignar valores a la variable de tipo arreglo (lect) donde se almacenan las muestras, y ejecutar parte del algoritmo en un nuevo proyecto. En esta prueba se evitaron las interrupciones periódicas para poder fijar puntos de ruptura, y se eligió la depuración del código en la opción de simulación del chip del microcontrolador.

La forma de obtener los ciclos de reloj de las instrucciones del algoritmo se explica a continuación:

1. Colocar un punto de ruptura en el procedimiento o línea de código del que se requiere determinar el número de ciclos de reloj.

2. Poner en marcha el depurador con el botón "*Start*". Ver figura 6.3.

3. Tomar el número de ciclos de reloj transcurridos hasta ese punto.

4. Ejecutar un "*Step Over*" o salto de instrucción y tomar el número de ciclos de reloj. Ver figura 6.4.

5. La diferencia entre los ciclos de reloj tomados en el paso 4 y el paso 3, da el número de ciclos de reloj consumidos por la línea de código donde se fijo el punto de ruptura.

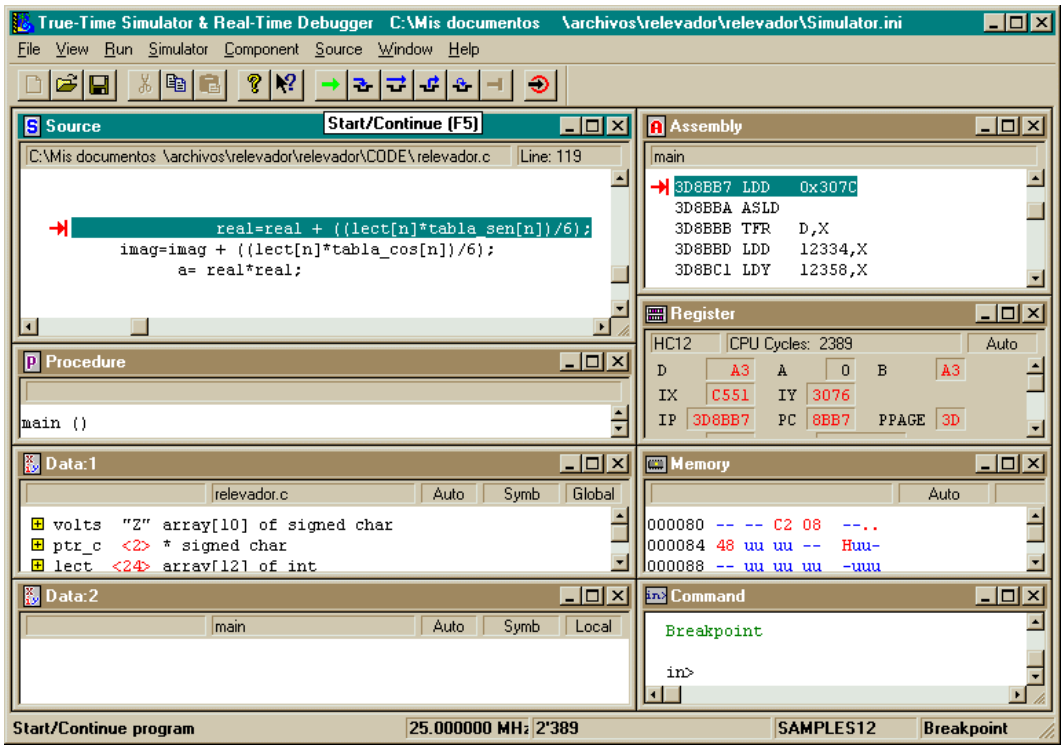


Figura 6.3: Depurador atendiendo un comando de punto de ruptura al poner en marcha el depurador.

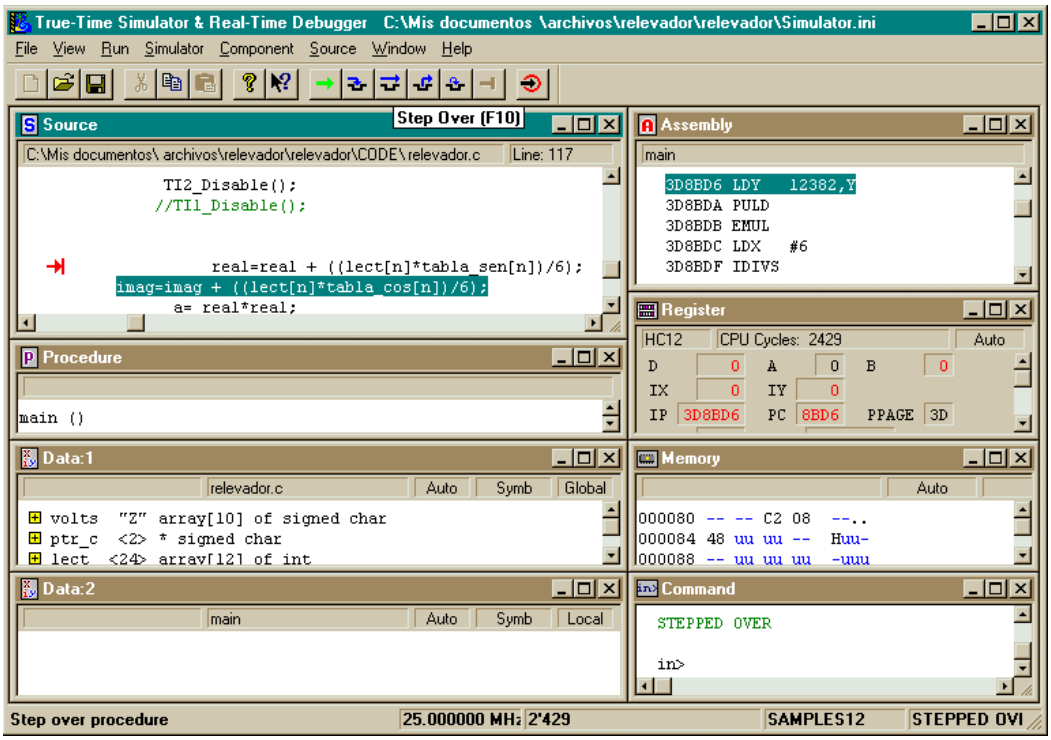


Figura 6.4: Ventana del depurador después de un "*Step Over*".

En la tabla 6.1 se muestran el número de ciclos de reloj consumidos por algunas de las líneas más importantes de código del algoritmo. De la tabla 6.1 se concluye que las funciones para calcular raíces cuadradas y potencias (sqrt();) y (pow();) de la biblioteca “math.h”, consumen un mayor número de ciclos de reloj que las operaciones para multiplicar y sumar.

El cálculo sucesivo de una raíz cuadrada en un ciclo de 12 iteraciones, toma 2.6 ms en completarse en el mejor de los casos. Por lo tanto, no es posible incorporar la función de raíz cuadrada en la parte principal del algoritmo cuando se utiliza una frecuencia de muestreo de 720 Hz., porque solamente con 12 iteraciones de la función raíz cuadrada se rebasa el límite de 1.38 ms.

Tabla 6.1: Número de ciclos de reloj consumidos por las líneas más importantes del código del algoritmo.

Operación	Tiempo
$real = \quad real + \left(\frac{lect[i] * tabla\_sen[i]}{} \right) \frac{N}{2};$	*1604 ciclos de reloj*
	64.16 μs
$imag = \quad imag + \left(\frac{lect[i] * tabla\_\cos[i]}{} \right) \frac{N}{2};$	*1584 ciclos de reloj*
	63.36 μs
$a = \quad pow(real,2);$	*1495 ciclos de reloj*
	59.98 μs
$b = \quad pow(imag,2);$	*1498 ciclos de reloj*
	59.92 μs
$I = \quad sqrt(a+b);$	*5603 ciclos de reloj*
	224.12 μs
$A = \quad real * real;$	*282 ciclos de reloj*
	11.28 μs
$B = \quad imag * imag;$	*276 ciclos de reloj*
	11.04 μs
$C = \quad A + B;$	*309 ciclos de reloj*
	12.36 μs

De igual manera, en el algoritmo se calculan dos potencias, en un ciclo de 12 iteraciones, el tiempo total requerido por estas funciones es de 1.43 ms. Al igual que en el caso anterior, con el cálculo de estas dos potencias en 12 iteraciones, se rebasa el límite de 1.38 ms.

6.3 Solución a las restricciones del tiempo de procesamiento del algoritmo

Para solucionar las limitaciones en el tiempo de procesamiento impuestas por la frecuencia de muestreo y las funciones de raíz cuadrada y potencia de un número, se probaron varias combinaciones de operaciones. En la tabla 6.2 se muestran los tiempos totales de procesamiento para 12 y 8 iteraciones de dos de las combinaciones examinadas.

Tabla 6.2: Comparativa entre el tiempo total tomado por dos distintos conjuntos de operaciones en 12 y 8 iteraciones.

Operaciones	Tiempo total para 12 iteraciones	Tiempo total para 8 iteraciones
(1) $real = real + \left(\frac{lect[i] * tabla\_sen[i]}{} \right) \frac{N}{2};$ $imag = imag + \left(\frac{lect[i] * tabla\_\cos[i]}{} \right) \frac{N}{2};$ *a=pow(real,2);* *b=pow(imag,2);* *I= sqrt(a+b);*	5.02 ms > 1.38 ms	2.38 ms > 2.08 ms
(2) $real = real + \left(\frac{lect[i] * tabla\_sen[i]}{} \right) \frac{N}{2};$ $imag = imag + \left(\frac{lect[i] * tabla\_\cos[i]}{} \right) \frac{N}{2};$ *A=real*real;* *B=imag*imag;* *C= A+B;*	1.9464 ms > 1.38 ms	**1.06 ms < 2.08 ms**

La mejor solución es la combinación de operaciones donde no se usan en la función principal las funciones de raíz cuadrada y potencia de la biblioteca “math.h” (combinación número dos). En reemplazo de la función de potencia, se multiplica el número a elevar por si mismo, reduciendo con esto, el número de ciclos de reloj necesarios para este cálculo (ver tabla 6.1).

El tiempo que toman en procesarse 12 iteraciones para la segunda combinación, es de 1.946 ms. Este tiempo supera el límite de 1.38 ms. y se concluye que no es posible utilizar la frecuencia de muestreo de 720 Hz., porque las operaciones del algoritmo en 12 iteraciones, en ambas combinaciones, requieren un tiempo de procesamiento mayor al periodo de muestreo de la señal.

Para extender el límite de tiempo de procesamiento del algoritmo, se recurrió a una frecuencia de muestreo de 480 Hz. Con esta frecuencia de muestreo el límite de ejecución del algoritmo es de 2.08 ms., y el número de coeficientes de los filtros Seno y Coseno se reduce a ocho; por lo tanto, el número de iteraciones de las operaciones del algoritmo también se reduce a ocho.

Con ocho iteraciones de la segunda combinación, el tiempo total de procesamiento es de 1.06 ms. y este tiempo esta dentro del límite del periodo de muestreo de 2.08 ms. En la figura 6.5 se muestran las gráficas de los datos recibidos en la PC de monitoreo usando una frecuencia de muestreo de 480 Hz. y constantes y variables de tipo flotante. En estas graficas se puede apreciar que no hay pérdida de datos porque el algoritmo se procesa en tiempo real.

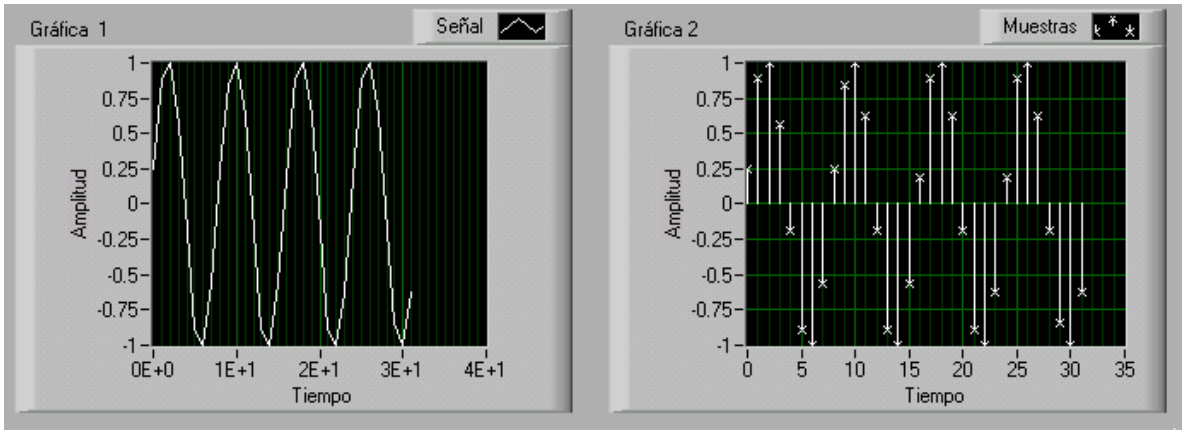


Figura 6.5 Gráficas de los datos recibidos cuando se utiliza una frecuencia de muestreo de 480 Hz y constantes y variables tipo entero.

Finalmente se optó por utilizar la frecuencia de muestreo de 480 Hz. y las operaciones de la segunda combinación. En este caso, la raíz cuadrada es calculada sólo si se cumple tres veces la condición que verifica que la variable C, donde se almacena el resultado de la suma de A y B, ha excedido el valor de la constante de referencia (Id). El valor de la constante Id es equivalente al valor pico de la corriente de arranque al cuadrado.

Abajo se muestra el segmento de código que se usó para eliminar el cálculo de la raíz cuadrada de la función principal. En este fragmento de código el contador llamado "*Falla*" se incrementa cuando C es mayor o igual que Id; cuando el

contador *Falla* vale tres, se interrumpen las operaciones de la función principal y se llama a la subrutina de disparo (curva_tinverso();).

```
    if(C>=Id)
{
    Falla++;
      if (falla==3)
       {
  curva_tinverso();
       }
 }
```

Listado 1: Fragmento de código usado para la validación de una falla por sobrecorriente.

El cómputo de la raíz cuadrada se hace en la subrutina de disparo y no en la función principal (main()), al cumplirse la condición anteriormente mencionada. Las instrucciones utilizadas para obtener el tiempo de disparo, emplean un número reducido de ciclos de reloj en comparación con los cálculos que se efectúan en la obtención de la magnitud de la señal de corriente.

6.2 Simulación de fallas de sobrecorriente

Una vez probado que el tiempo de procesamiento del algoritmo en el microcontrolador no rebasa el límite de 1.38 ms, se procedió a la simulación de fallas de sobrecorriente para comprobar la exactitud de las mediciones hechas por el prototipo.

Las señal de prueba se tomó del circuito acondicionador de señales examinado en el capítulo 5, este circuito proporciona una salida 3.26 Volts. El valor de esta tensión se adaptó para considerarla como una señal de corriente de 5A. El procedimiento para establecer la exactitud de las mediciones es el siguiente:

1. Utilizar el depurador del microcontrolador en función de monitor para poder analizar las mediciones efectuadas por el microcontrolador.

2. En el depurador localizar el código fuente de la subrutina que genera el tiempo de disparo del relevador (curva_tinverso.c).

3. Colocar un punto de ruptura en la línea que habilita la interrupción periódica de esta subrutina (TI2_Enable();). Ver figura 6.6.

4. Correr el depurador y verificar los resultados de las mediciones en la ventana “Data: 1” y anotar los valores del voltaje medido y el tiempo de disparo.

5. Después de haber tomado varias lecturas, se realizan los cálculos estadísticos convenientes para encontrar la exactitud de las mediciones del relevador.

Del ejemplo de la coordinación de dos relevadores de sobrecorriente del capítulo 2, se tomaron los datos de los taps y multiplicadores de los relevadores R1 y R2; estos datos se fijaron como ajustes de las curvas de tiempo inverso del relevador prototipo. En base a esto, se simuló la operación de los relevadores R1 Y R2.

Los resultados de las mediciones y los tiempos de disparo calculados por el prototipo se observan en la tabla 6.3.

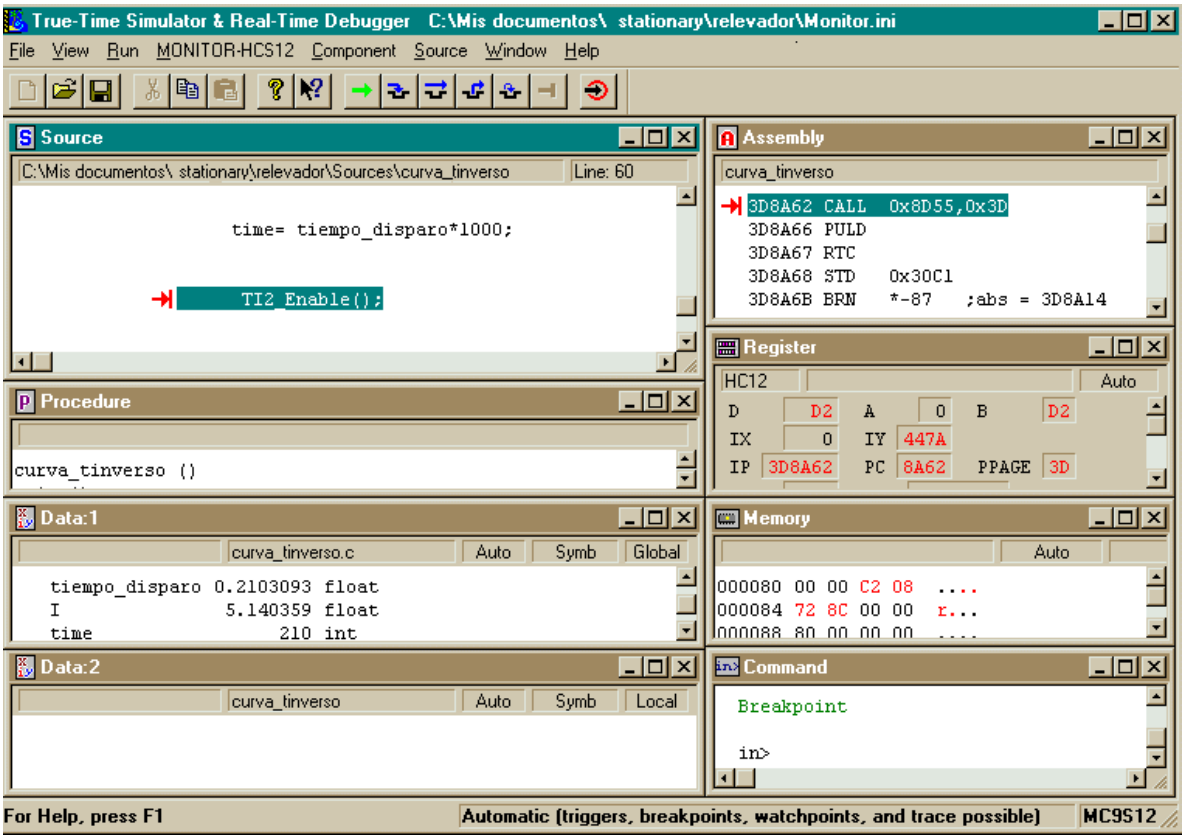


Figura 6.6: Punto de ruptura en la habilitación de la interrupción periódica de la subrutina “curva_tinverso”.

Tabla 6.3: Mediciones de corriente de falla y tiempos de disparo tomados del depurador.

Corriente de falla medida por el relevador R2	Tiempo de disparo [s] Palanca de tiempo igual a 0.05 en R2	Corriente de falla medida por el relevador R1 [A]	Tiempo de disparo [s] Palanca de tiempo igual a 0.17 en R1
5.14	0.21	4.98	0.72
4.94	0.21	5.14	0.71
5.14	0.21	5.07	0.72
5.17	0.20	4.98	0.72
5.2	0.20	5.05	0.72
4.97	0.21	4.94	0.73
5.14	0.21	5.05	0.72
4.99	0.21	5.07	0.72
5.05	0.21	5.08	0.71
5.16	0.20	5.07	0.72
5.09	0.21	4.99	0.72
5.07	0.21	4.98	0.72

La media aritmética de los valores de la primera columna de la tabla 6.3 es de 5.08; la media aritmética de la segunda columna es de 0.2075; la media aritmética de los datos de la tercera columna es de 5.033 y la media aritmética de los valores de la cuarta columna es de 0.7192.

En el ejemplo de la coordinación de dos relevadores de sobrecorriente descrito en el capítulo 4, el valor de tiempo de disparo para el Relevador R1 es de 0.2 segundos, y el valor de tiempo de disparo para el relevador R2 es de 0.7 segundos. Al comparar estos datos con las medias aritméticas de los tiempos de disparo de la tabla 6.3, se concluye que el relevador reproduce los valores de tiempo de disparo con un error mínimo.

CAPÍTULO 7

CONCLUSIONES

Desde algunas décadas se han estado haciendo investigaciones en el área de relevadores digitales, y se han producido muchos artículos científicos acerca de estos; pero, mucha de la información no esta al alcance de el común de los estudiantes de ingeniería en México. Esta tesis pretende ser una introducción al conocimiento general de los relevadores digitales.

En el desarrollo de esta tesis se logró implantar el algoritmo de la Transformada Discreta de Fourier en un microcontrolador de bajo costo, algoritmo que como se ha estudiado, requiere ser procesado en tiempo real. Sin embargo, se tuvieron limitaciones, debido principalmente, al tiempo de procesamiento de la función raíz cuadrada de la biblioteca math.h. La función raíz cuadrada sólo es calculada si se cumple la condición de validación de falla que se propuso como alternativa al no poder calcular la magnitud de la corriente en la rutina principal.

Finalmente, en base a los resultados obtenidos, se sugiere codificar la función de raíz cuadrada en lenguaje Ensamblador con la finalidad de optimizar el tiempo de ejecución de esta operación.

Autor: Jesús Cortés
Email: jescop_82@yahoo.com.mx

REFERENCIAS

[1] Antonio A. Quijano, Héctor O. Pascual, José A. Rapallini, “Implementación de un sistema de medida de impedancia para redes eléctricas en tiempo real”, CeTAD, Dpto. Electrotecnia, Facultad de Ingeniería, Universidad Nacional de La Plata, La Plata, Argentina, Marzo 2001.

[2] Arun G. Phadke, J.S. Thorp, “Computer Relaying for Power Systems”, Jhon Wiley and Sons, New York, 1988.

[3] Applications for SIPROTEC Protection Relays, Siemens Aktiengesellschaft, Power Transmision and Distribution Energy Automation Division, Nuernberg Germany, January 2005.

[4] B. Ravindranath, M. Chander, “Protección de sistemas de potencia e interruptores”, Editorial Limusa, 1989.

[5] C. Rusell Mason, “The art and science of protective relaying”, Jhon Wiley & Sons, Inc., (second printing) London 1958.

[6] CodeWarrior Development Tools C compilers Reference 3.2, Metrowerks Corporation, July 2004.

[7] Craig Marven, Gillian Ewers, “A simple Approach to Digital Signal Processing”, Jhon Wiley & Sons, inc., 1996.

[8] Curso de protecciones de la marca Siemens para el grupo de generadores más sus transformadores, Servicios Especializados de Ingeniería de Protecciones Eléctricas, Tula, Hidalgo, Julio 2006.

[9] Douglas K. Linder, “Introduction to Signals and Systems”, Mc Graw Hill International Editions, Electrical Engineering Series, 1999.

[10] Gabriel Benmouyal, Stanley E. Zocholl “How microprocessors respond to harmonics, saturation, and other wave distortions” Schweitzer Engineering Laboratories, Inc. Pullman, WA USA 1997.

[11] G. D. Rockefeller, “Fault Protection with a Digital Computer”, IEEE Trans. Power App. Syst., Vol. PAS-88, pp. 438-462, Apr. 1969.

[12] González Martínez Angel, “Estudio de la evolución tecnológica de relevadores de sobrecorriente usados en los esquemas de protección de los sistemas eléctricos de potencia”, Tesis de Licenciatura, ESIME IPN, Unidad Zacatenco, México, 1992.

[13] Harjeet Singh Gill, “ A Microprocessor Based System for Protecting Busbars”, A thesis for the Degree of Doctor of Philosophy in the Departament of Electrical

Engineering, University of Saskatchewan, Saskatoon, Saskatchewan, Canada, March 2000.
[14] Hfuda A. Mohamed, Sachdev S. Mohindar, Shidu S. Tarlochan, “Generating Relays Models for Protection Studies”, IEEE Computer Applications in Power,Volume 1, Number 4, October 1988.

[15] Javad Sadeh, “Optimal coordination of overcurrent relays in an interconected power system”, Electrical Department, Faculty of Engineering Ferdowsi, University of Mashhad, 15th PSCC, Session 19, Liege, Mashhad, Iran, August 2005.

[16] Karl Zimmerman, “Microprocessor Based Distribution Applications”, Schweitzer Engineering Laboratories, Inc., Belleville, IL USA, January 2001.

[17] Khaled Shehata, Ahmed Bahaa, Karim Morad, Ahmed Sharaf, “Design and Implementation of FPGA Based and Microcontroller Based Current Relay”, Modern Science and Arts University, Giza, Egypt, Oct. 2004.

[18] Mladen Kezunovic, “Fundamentals of Power System Protection”, Academic Press, 2005.

[19] MC912E – Family Device User Guide V01.04, Motorola, Inc., November 2003.

[20] Methods for Coordinating System Protection Equipment, Facilities Instructions, Standars and Techniques, Volume 3-9, Facilities Engineering Branch, Denver Office, Denver Colorado, 1991.

[21] M.M. Chen, W.D. Breingan, T.F. Gallen, “Field experience with a digital system for transmission line protection” IEEE trans. Power App. Syst. Vol PAS-98, pp. 1796-1804, sep/oct 1979.

[22] Processor Expert Help Manual for CodeWarrior Plug in for Motorola, version 2.48, October 6, 2006.

[23] Stanley H. Horowitz, *“Protective Relaying for Power Systems”*, A volumen in the IEEE PRESS selected reprinted series, prepared under the Sponsorship of the IEEE Power Engineering Society, IEEE PRESS 1980.

[24] T. S. Mahava Rao, “Power Systems Protection, Static Relays” Mc Graw Hill book company 1981.

[25] Thomas L. Floyd, “Dispositivos Electrónicos 2”, Editorial Limusa, 1994.

[26] Young Guo, Mladen Kezunovic, Deshu Chen, “Simplified Algorithms for Removal of the Effect of Exponentially Decaying DC-Offset on the Fourier Algorithm”, Department of Electrical Engineering, Texas A&M University, Texas, Mar. 2002.

APÉNDICE A

HERRAMIENTAS DE DESARROLLO USADAS EN LA PROGRAMACIÓN DEL MICROCONTROLADOR MCHCS12E64:

En este apéndice se mencionan las herramientas de desarrollo usadas en la programación del microcontrolador HCS12E64 y la configuración de las periféricos utilizados en el programa.

A.1 Code Warrior

Para editar, compilar y depurar el código del programa se eligió el ambiente de desarrollo Integrado (IDE) de Code Warrior™ versión 3.1. El IDE de Code Warrior es intuitivo y, además, permite depurar y simular programas en lenguaje Ensamblador, lenguaje C y C++. El IDE de Code Warrior esta compuesto por un organizador de proyectos (Project Manager), un editor de código fuente (Source Code Editor) y un examinador (Browser). En la Figura A.1 se observa la ventana del IDE de Code Warrior.

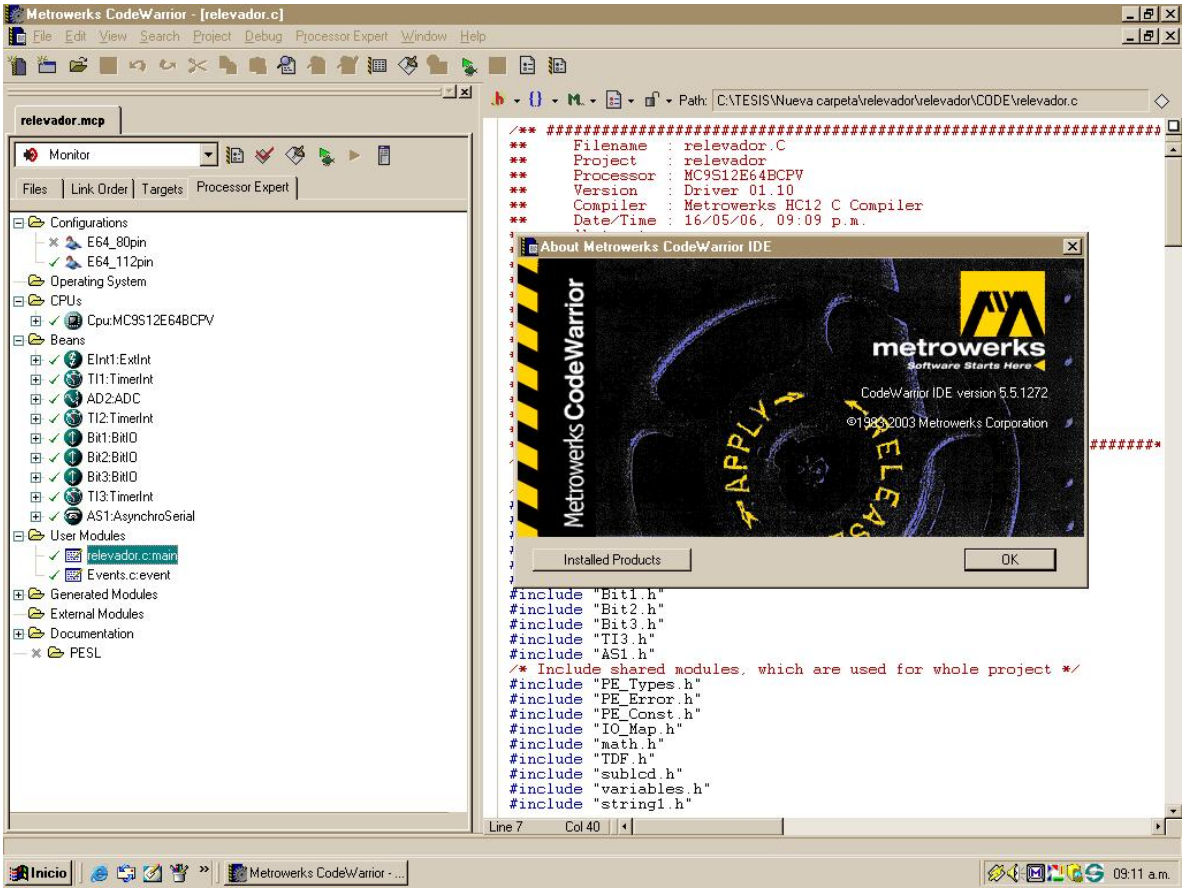


Figura A.1: Ventana del IDE de Code Warrior.

El IDE de Code Warrior usa proyectos (Projects) y construye objetivos (Targets) para organizar los archivos y opciones necesarias para la creación de un programa. Un proyecto es un archivo que contiene uno o más “Targets”. Un “Target” incluye: archivos de código fuente, librerías, opciones y otros proyectos. Los objetivos describen como se crea el software para un procesador particular o un sistema operativo.

A.2 Processor Expert

En el desarrollo de la programación del algoritmo se uso también el Processor Expert™ versión 2.71 de UNIS para la familia de microcontroladores HCS12. El Processor Expert es una herramienta diseñada para el desarrollo de aplicaciones rápidas a través de cápsulas configurables llamadas “Embedded Beans”. El Processor Expert está incluido en el IDE de Code Warrior 3.1. En la Figura A.2 se observa la ventana de presentación del Processr Expert.

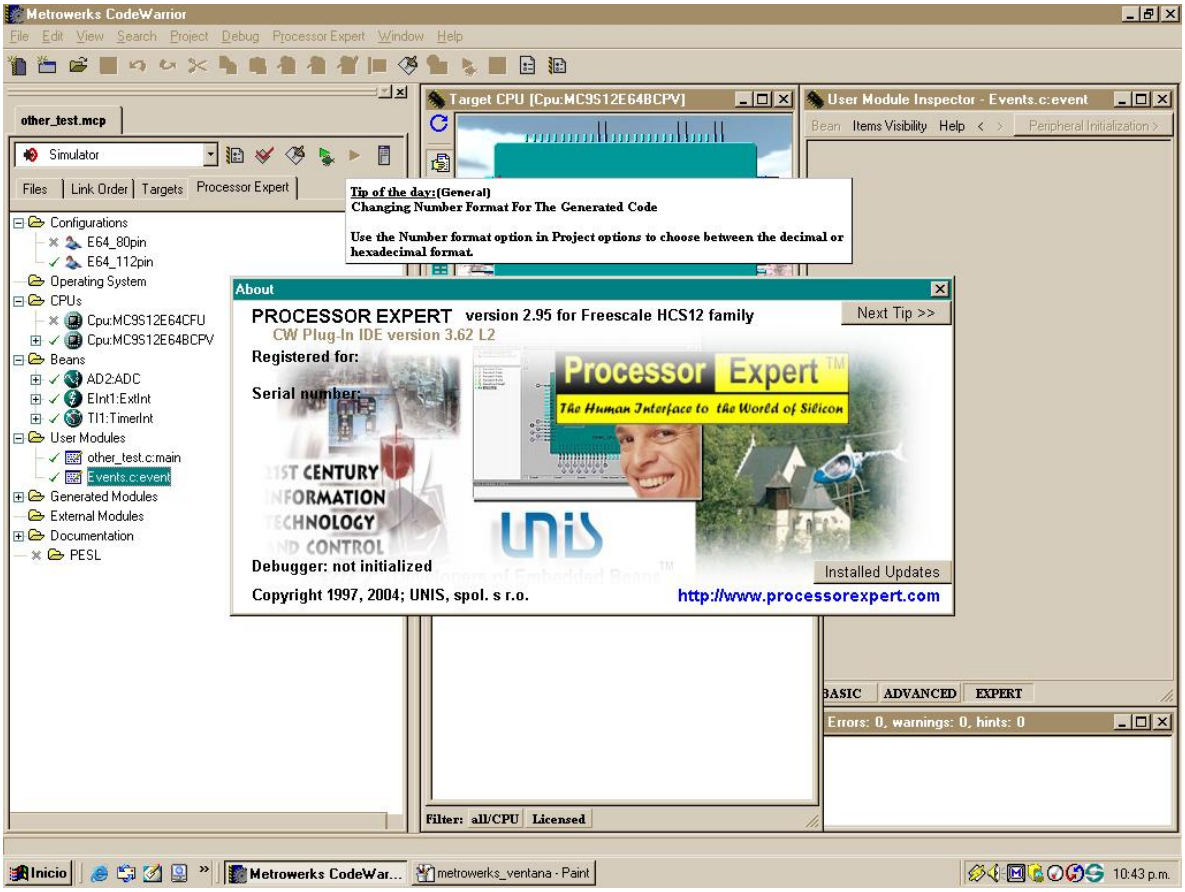


Figura A.2: Ventana de presentación del Processor Expert.

A.3 Cápsulas del Processor Expert

El Processor Expert reduce el esfuerzo de programación por medio de sus cápsulas configurables, la funcionalidad de la CPU del microcontrolador y sus periféricos están contenidos en estas cápsulas. Cada cápsula del Processor Expert provee una interfase para el usuario "Bean inspector"; en esta interfase el usuario puede seleccionar, habilitar, y deshabilitar las propiedades, métodos y eventos de la cápsula que se este utilizando.

Por ejemplo: En la interfase de la cápsula correspondiente al convertidor analógico digital (Figura A.3) podemos configurar a través de menús la resolución y el tiempo de conversión del convertidor.

La generación del código correspondiente a las cápsulas, con los comentarios necesarios, se realiza automáticamente por medio del Processor Expert, de acuerdo a la configuración de cada cápsula usada. Después, Code Warrior se encarga del manejo de los archivos de proyecto, la compilación y la depuración del código final.

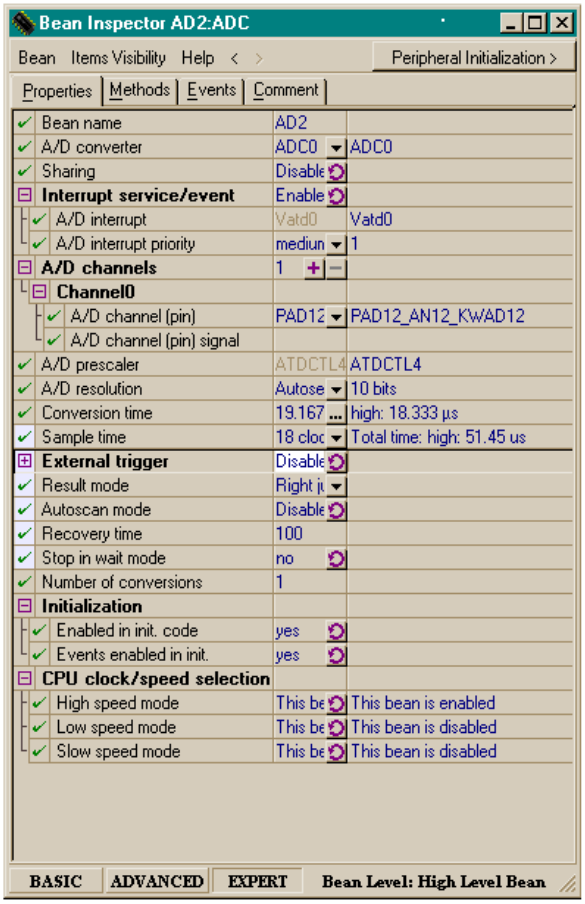


Figura A.3: Inspector de la cápsula correspondiente al convertidor analógico digital.

A.4 Configuración del proyecto de trabajo:

Antes de editar el código del programa se debe crear un proyecto en el IDE de Code Warrior. Para crear un proyecto nuevo, en la ventana principal de Code Warrior se selecciona el menú (File) y se elige (New Project). En la ventana (New) se elige (New Project Wizard) y se le pone un nombre al proyecto, después, se pulsa el botón “aceptar” (Ver Figura A.4) .

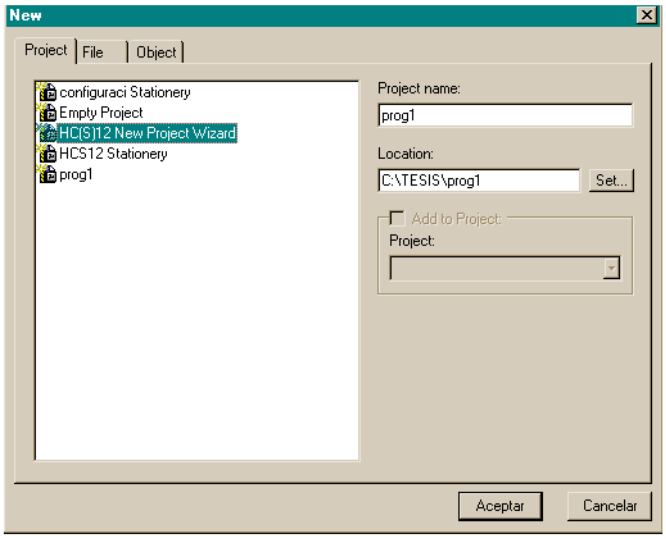


Figura A.4: Ventana “New Project”.

Después, en la ventana (New Project Wizard – Page 1) se selecciona el microcontrolador que se va a usar (Ver Figura A.5).

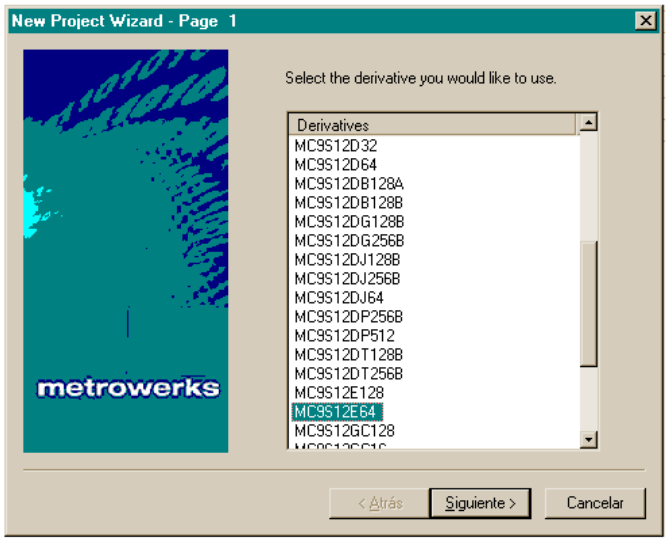


Figura A.4: Ventana New Project Wizard – Page1.
En la siguiente ventana (New Project Wizard – Page 2) se selecciona el lenguaje que debe soportar el compilador, para este caso, Lenguaje C. Se pulsa el botón "siguiente" (Ver Figura A.4).

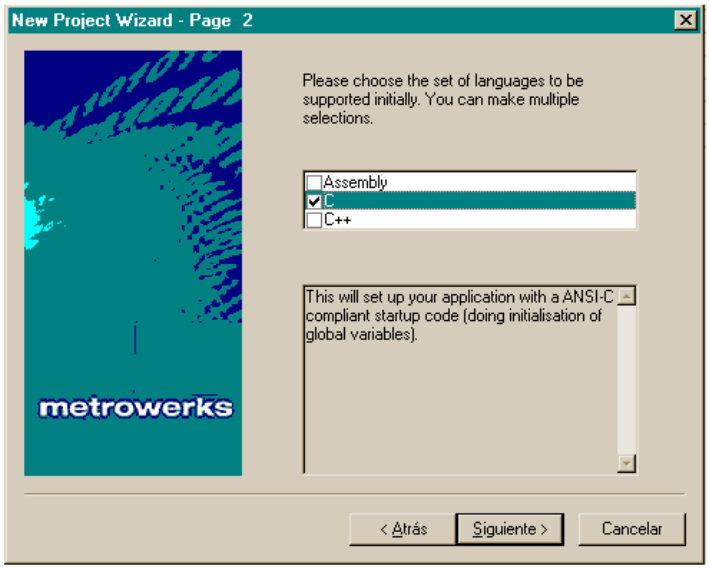


Figura A.4: Ventana New Project Wizard – Page 2.

En la ventana (New Project Wizard – Page 3) se pregunta si se desea usar el Processor Expert. Aquí se elige la opción "Si" (Ver Figura A.5).

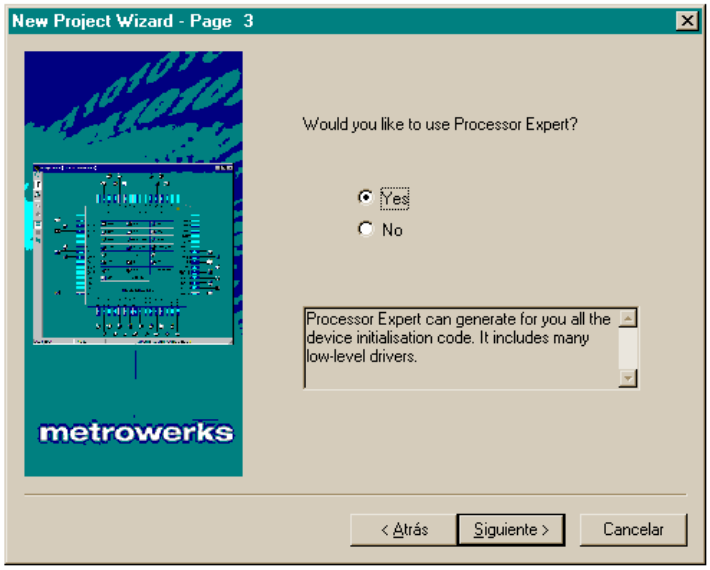


Figura A.5: Ventana New Project Wizard – Page 3.

Una vez seleccionada la opción de usar Processor Expert, en la ventana (New Project Wizard – Page 4), se pregunta si se desea usar Pc – lint. Se elige la opción “No” (Ver Figura A.6).

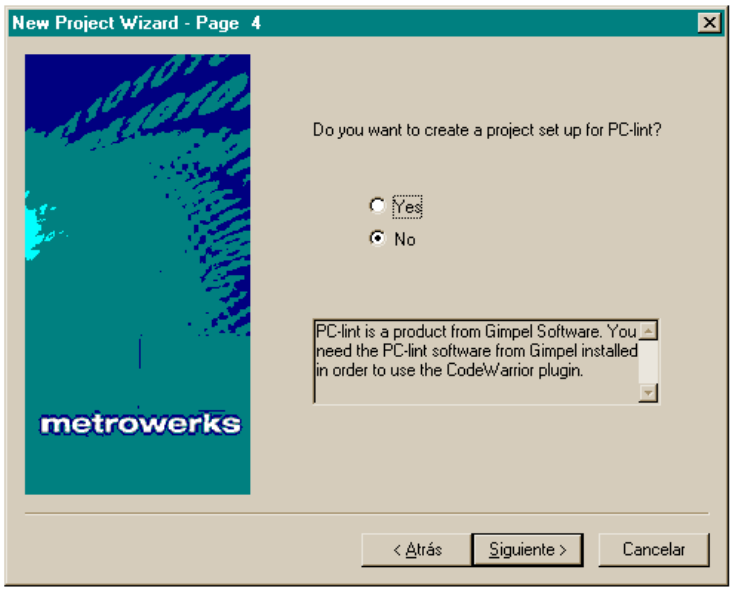


Figura A.6: Ventana New Project Wizard – Page 4.

En la ventana (New Project Wizard – Page 5) se da la opción de seleccionar el formato de punto flotante que debe soportar el compilador. Se elige la opción (float is IEEE32, double is IEEE32), (Ver Figura A.7).

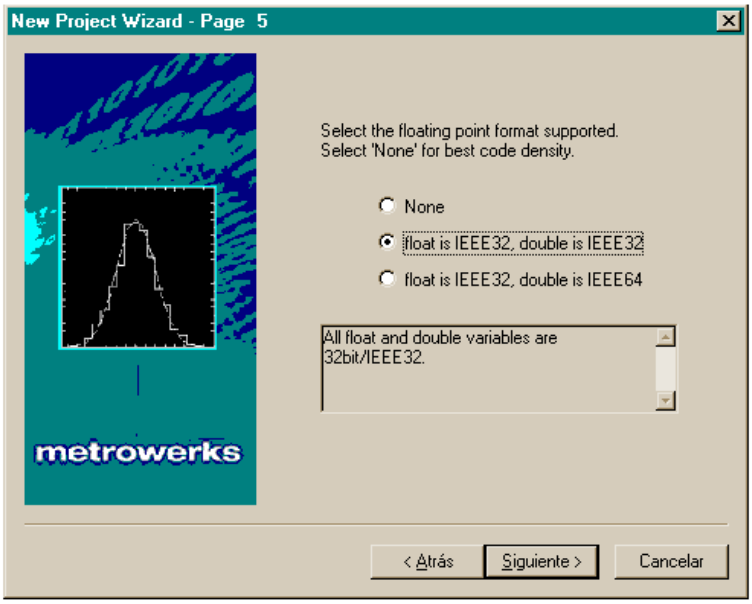


Figura A.7: Ventana New Project Wizard – Page 5.

La última ventana es la ventana (New Project Wizard – Page 6) aquí se eligen las opciones: "Metrowerks Full Chip Simulator" y "Motorola Serial Monitor Hardware Debugging" ; al elegir estas opciones se puede simular y monitorear el programa desde una Pc. Al presionar el botón de "Finalizar", el proyecto se configura automáticamente (Ver Figura A.8).

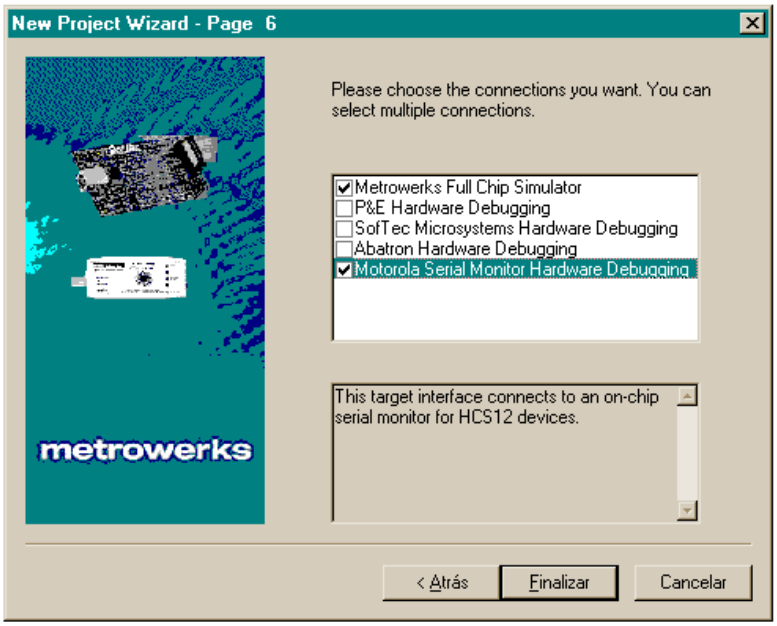


Figura A.8: Ventana New Project Wizard – Page 6.

A.5 Configuración de las cápsulas del Processor Expert utilizadas en el proyecto:

La configuración particular de cada cápsula es la siguiente:

En la cápsula CPU:MC9S12E64BCPV se elige la frecuencia del oscilador a 8 Mhz; se habilita el reloj del PLL y se establece una frecuencia interna de reloj de 25 Mhz (Ver Figura A.9).

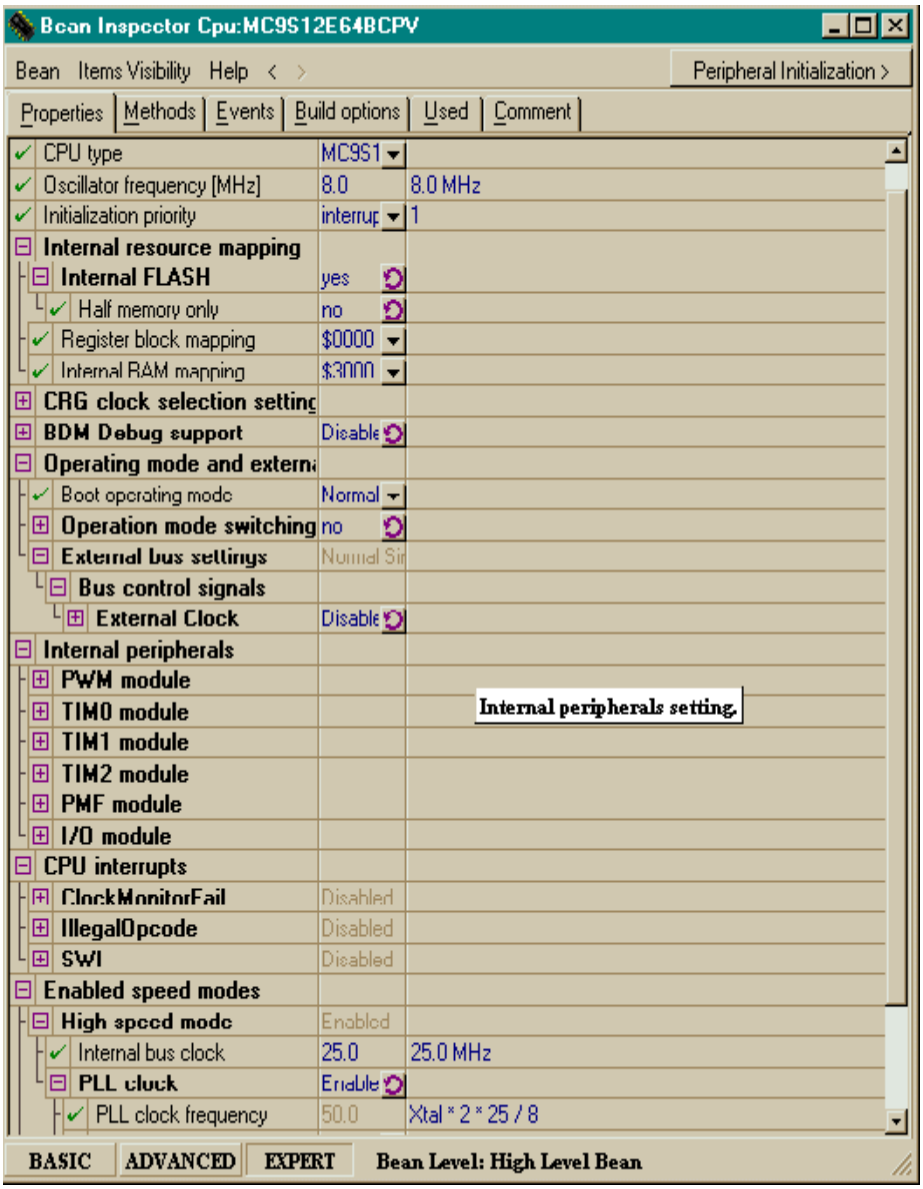


Figura A.9: Inspector de la cápsula correspondiente al CPU.

Para la cápsula TI1:TimerInt se selecciona el tiempo de interrupción en el inspector de cápsulas; el tiempo de interrupción es de 1.38 milisegundos, esta interrupción establece una frecuencia de muestreo de 480 Hz (Ver Figura A.10).

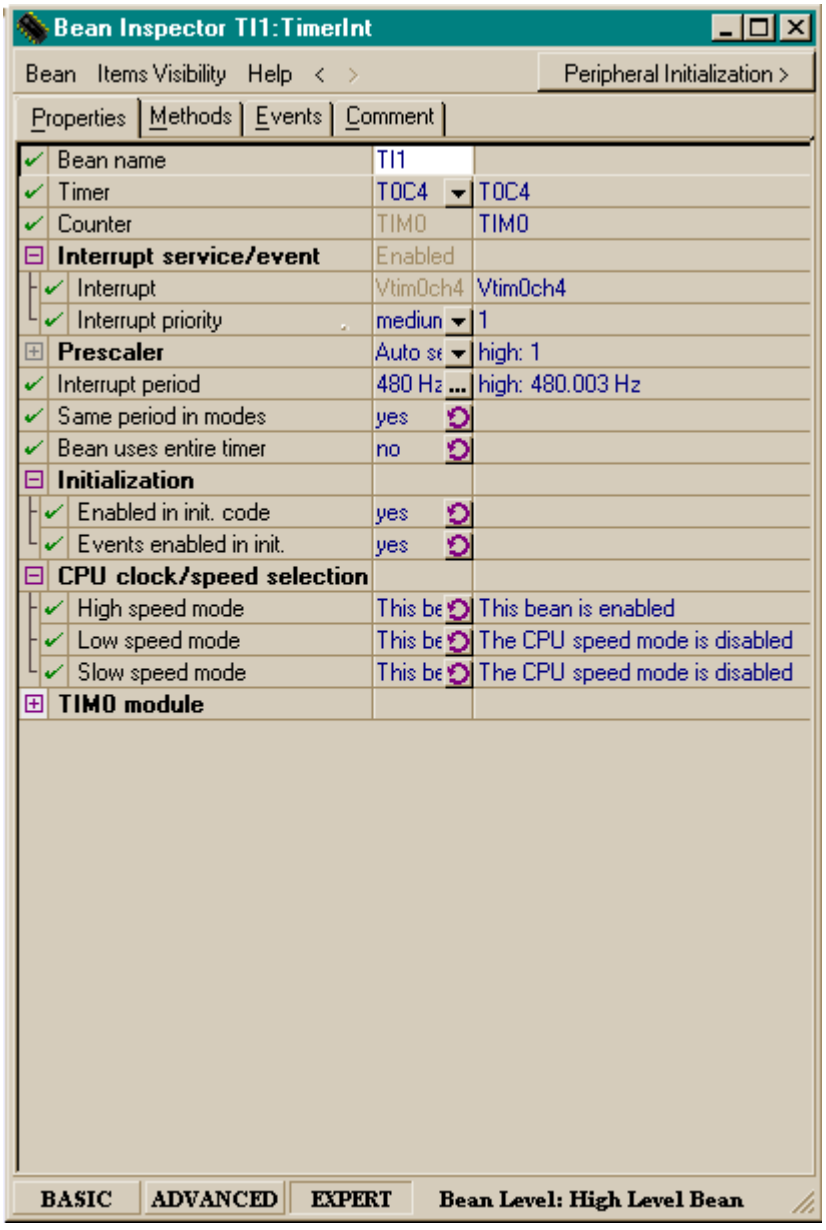


Figura A.10: Inspector de la cápsula correspondiente a la Interrupción TI1.

Para la cápsula AD2:ADC (convertidor analógico digital), se selecciona la pata por donde ingresa la señal proveniente del circuito acondicionador de señales; en este caso se trata de la pata PAD12_AN12_KWAD12. También, se elige la resolución de 10 bits, y el tiempo de conversión de 7.2 μS (Ver Figura A.11).

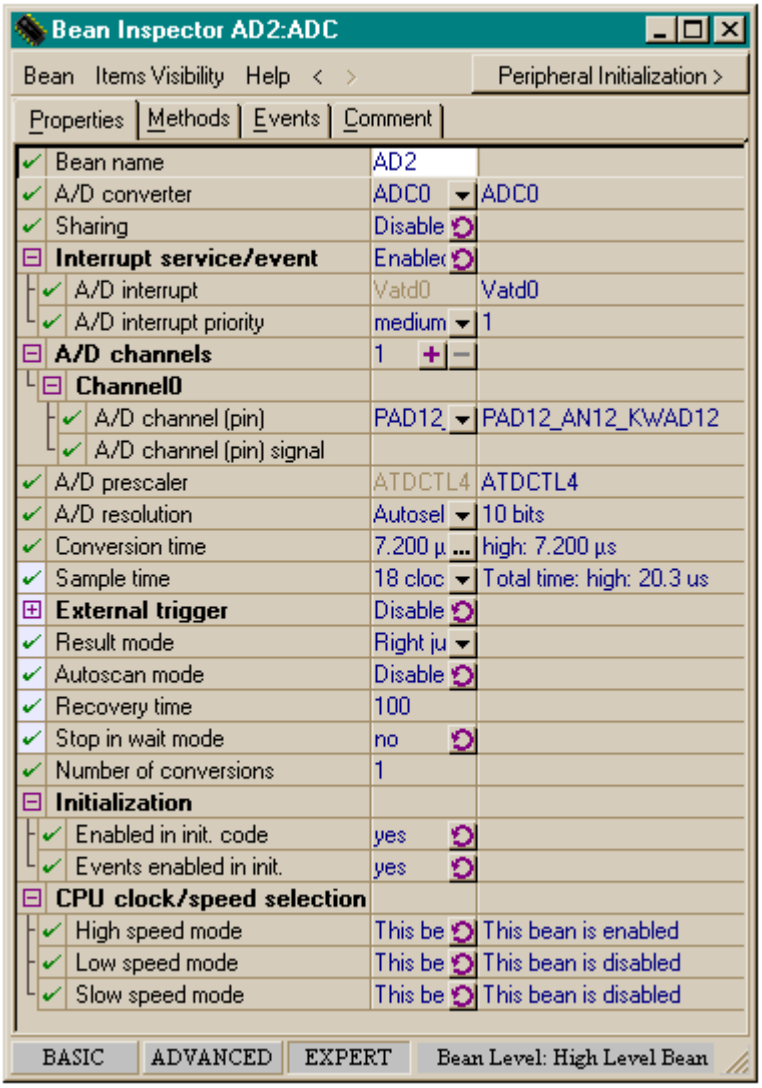


Figura A.11: Inspector de la cápsula correspondiente al convertidor analógico digital.

Para la cápsula TI2:TimerInt se selecciona un tiempo de interrupción de 1 milisegundo, esta interrupción establece es utilizada para la subrutina de la curva de tiempo inverso. La misma configuración se repite para T13(Ver Figura A.12).

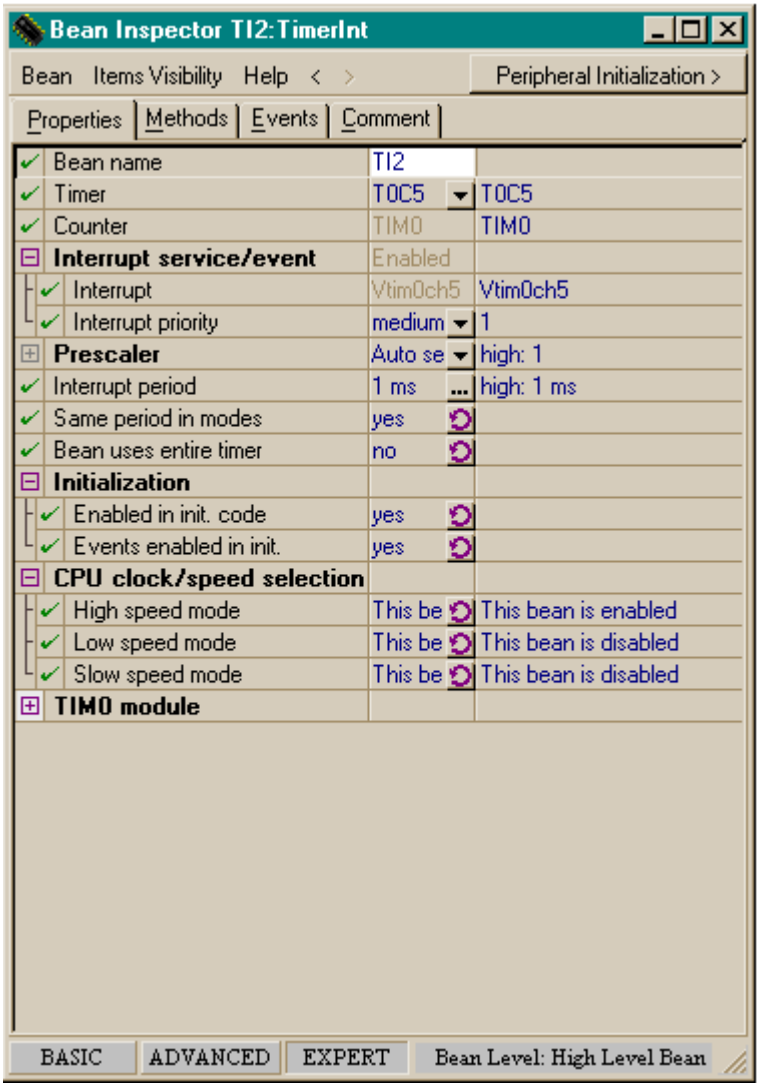


Figura A.12: Inspector de la cápsula correspondiente a la interrupción TI2.

En la cápsula AS1:AsynchroSerial se activan el servicio de interrupciones para poder transmitir y recibir datos. La velocidad de operación de este periférico es de 9600 baudios. También, se eligen las terminales del microcontrolador por donde ingresan y salen los datos (Ver Figura A.13).

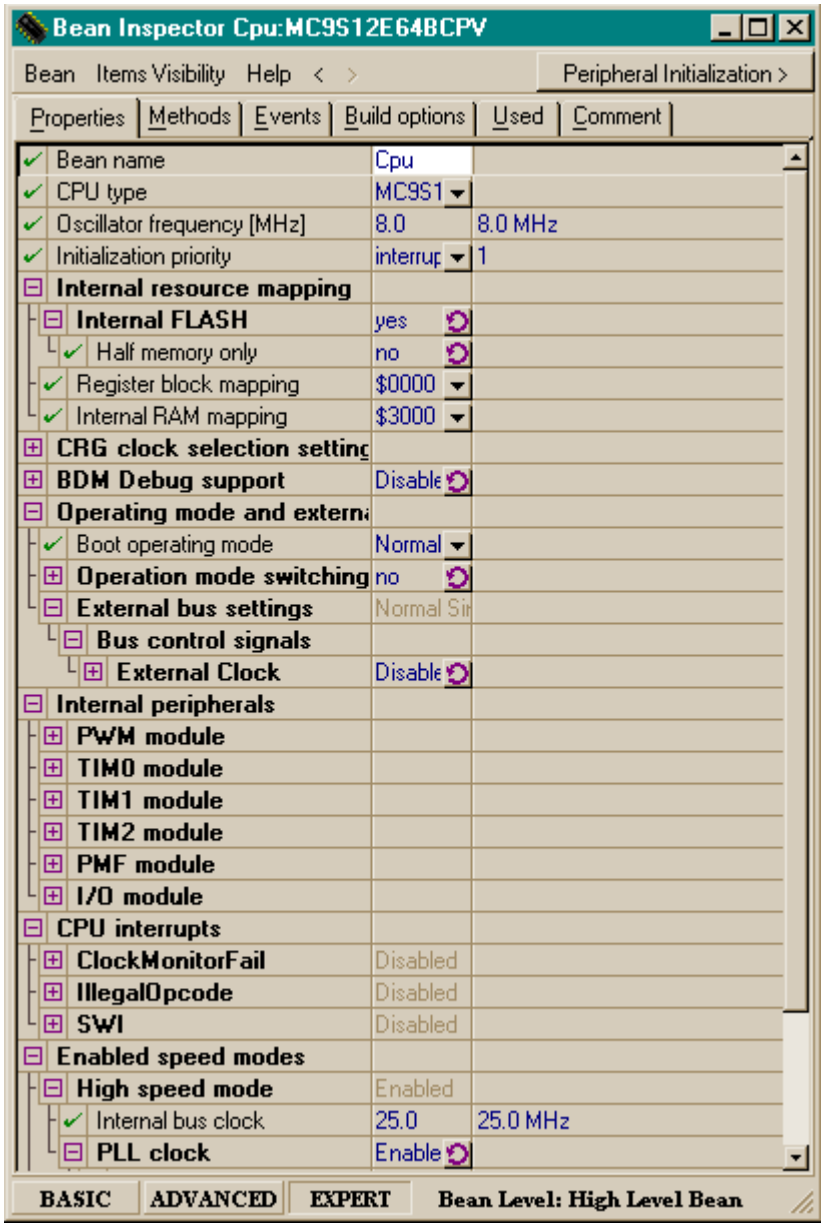


Figura A.13: Inspector de la cápsula correspondiente a la SCI.

Para la cápsula bit1 se tiene la siguiente configuración. La señal de salida de este bit indica la operación del relevador (Ver Figura A.14)..

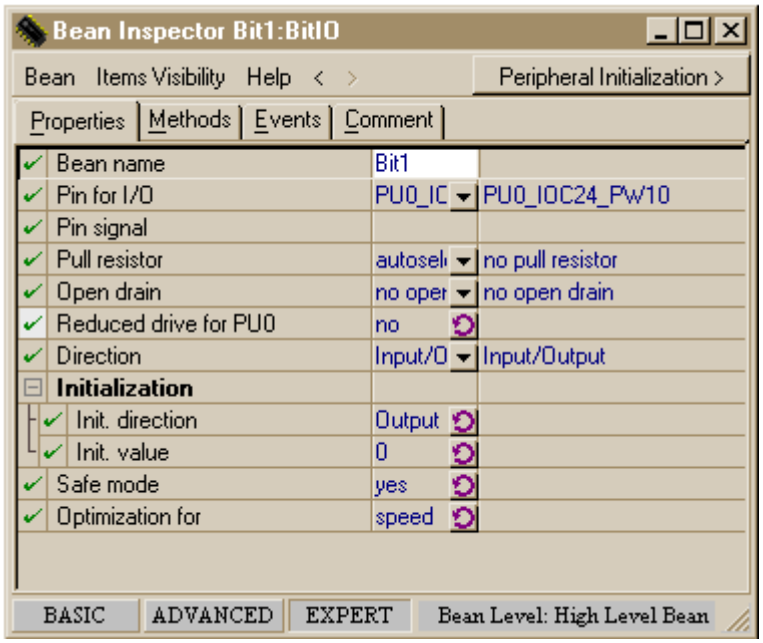


Figura A.14: Inspector de la cápsula correspondiente al Bit1.

La cápsula bit2 se configura de la manera siguiente. La señal de salida de este bit indica ha ocurrido un evento de falla (Ver Figura A.15).

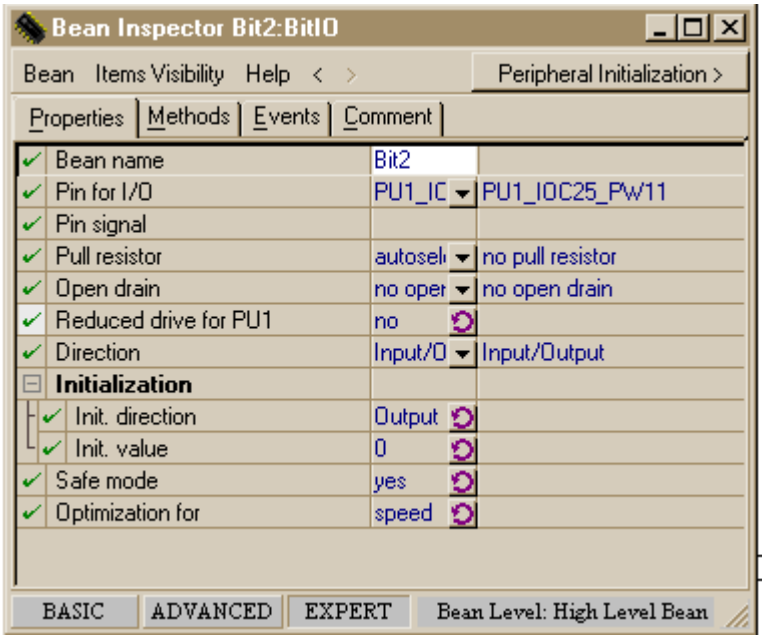


Figura A.15: Inspector de la cápsula correspondiente al Bit2.

La configuración para los bits que utiliza el teclado matricial es la siguiente (Ver Figura A.16).

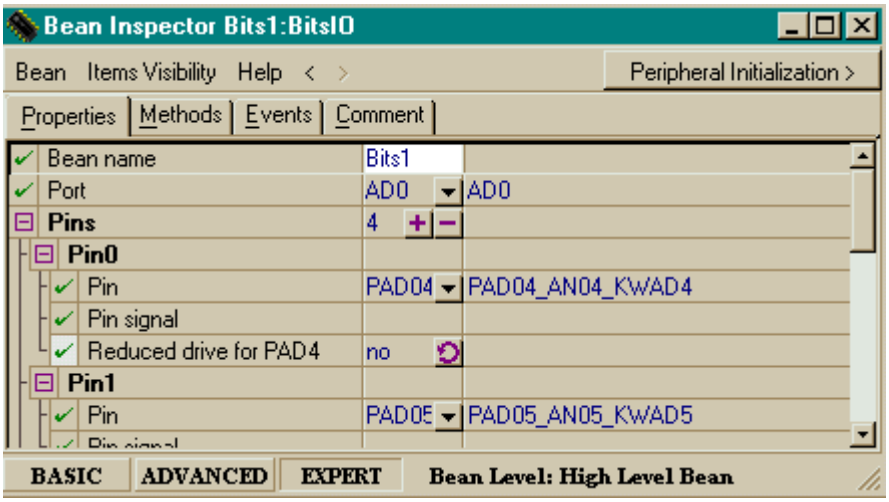


Figura A.16: Inspector de la cápsula correspondiente al Bits1.

La configuración para las interrupciones externas usadas por el teclado matricial tiene el formato siguiente. En las demás interrupciones externas sólo se elige la terminal de salida . (Ver Figura A.17).

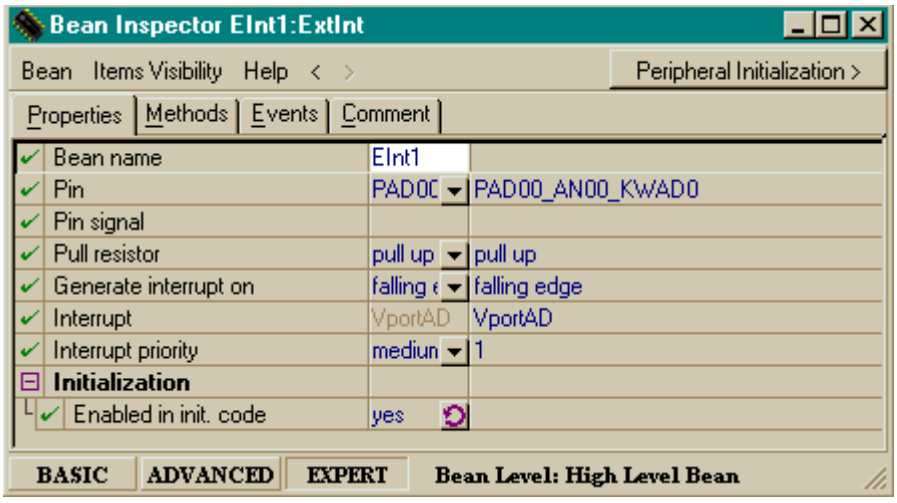


Figura A.17: Inspector de la cápsula correspondiente la interrupción externa EInt.

APÉNDICE B

CÓDIGO FUENTE DEL ALGORITMO

En este apéndice se muestra el código fuente del programa en lenguaje C. No se incluyen los archivos de configuración creados por el Processor Expert .

```
/**
##################################################################
#
**     Nombre de archivo : relevador.C
**     Proyecto : relevador
**     Autor:  Jesús Cortés Peña
**     Procesador : MC9S12E64BCPV
**     Versión   : Driver 01.10
**     Compilador  : Metrowerks HC12 C Compiler
**     Fecha/Hora : 16/05/06, 09:09 p.m.
**     Resumen  :  Este  programa sirve para la detección de fallas   por
**     sobrecorriente en sistemas monofásicos.  Se programó  una curva
* *    curva característica de tiempo inverso de la Norma IEC-6023.
**
**     Módulo principal.
**     Here is to be placed user's code.
**     Settings  :
**     Contents  :
**     No public methods
**
**     (c) Copyright UNIS, spol. s r.o. 1997-2004
**     UNIS, spol. s r.o.
**     Jundrovska 33
**     624 00 Brno
**     Czech Republic
**     http      : www.processorexpert.com
**     mail      : info@processorexpert.com
**
##################################################################
#*/
/* MODULE relevador */

/* Including used modules for compiling procedure */
#include "Cpu.h"
#include "Events.h"
#include "EInt1.h"
#include "TI1.h"
#include "AD2.h"
```

```
#include "TI2.h"
#include "Bit1.h"
#include "Bit2.h"
#include "Bit3.h"
#include "EInt1.h"
#include "EInt2.h"
#include "EInt3.h"
#include "EInt4.h"
#include "Bits1.h"
#include "TI3.h"
#include "AS1.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "math.h"
#include "TDF.h"
#include "sublcd.h"
#include "variables.h"
#include "string1.h"
#include "curva_tinverso.h"
#include “retardo.h”
#include “retardo_lcd.h”

/*DECLARACIÓN DE VARIABLES Y CONSTANTES GLOBALES*/

 int lect[8]={0,0,0,0,0,0,0,0};
 const float tabla_sen[8]= {0,0.7071,1,0.7071,0,-0.7071,-1,-0.7071 };
 const float tabla_cos[8]= {1,0.7071,0,-0.7071,-1,-0.7071,0,0.7071 };
 unsigned int k=0;
 unsigned int i=0;
 unsigned int j=0;
 unsigned int n=0;
 unsigned int counter=0;
 unsigned int contmseg=0;
 unsigned int valor=0;
 int resu=0;
 int real=0;
 int imag=0;
 int A=0;
 int B=0;
 int C=0;
 int Id=100;
 int Falla= 0;
 unsigned char val;
 unsigned int t_muestreo=1;
```

```
 unsigned char total;
 unsigned char ajuste[16]={0,0,0,32,32,32,32,32,32,32,32,32,32,32,32,32};
 unsigned char tec[16]={84,101,99,108,97,32,32,32,32,32,32,32,32,32,32,32};


 void main(void)

{

/*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
  PE_low_level_init();
/*** End of Processor Expert internal initialization.  ***/

/* Write your code here */
TI1_Disable();                  /*Desactiva interrupciones periódica principal*/
TI2_Disable()                   /*Desactiva interrupción  la subrutina de disaparo */
ini_LCD();                      /*Inicializa LCD */
PTADLo = 0x0;                   /*Pone en ceros los bits para el teclado matricial*/
retardo_lcd(10);                /* Retardo para activar el LCD*/

Despliega_L1("Pick up");        /*Despliega mensaje en la pantalla de LCD*/


TI3_Disable();                  /*Desactiva interrupción periódica del LCD*/
TI1_Enable();                   /*Activa interrupción periódica principal*/

                            while(1)        /*Ciclo infinito*/
       {

                        Despliega_L2(&ajuste);  /*Despliega ajuste del relevador*/
                        retardo(t_muestreo);    /*Retraso de 2.08 ms*/

                        AD2_Measure(TRUE);  /*Arranca  ADC*/
                        AS1_SendChar(total);  /*Envía muestra por sci*/
                        Bit1_SetVal();          /*Activa Led indicador*/

                         if(k==8)       /*Si hay ocho muestras,  estas se  recorren */
                                        /* y se calculan las componentes            */
                  {                     /* real e imaginaria                        */

                    real=0;   /*Fija a cero real*/
                    imag=0; /*Fija a cero imag*/


                              for(n=0;n<=7;n++)

                     {
```

```
            imag=imag + ((lect[n]*tabla_sen[n])/6);      /*cálculo de real*/
            real=real + ((lect[n]*tabla_cos[n])/6);  /*cálculo de imag*/

                        A= imagl*imag;         /*Cuadrado de imag*/
                        B= real*real;     /*Cuadrado de real*/

                        C= A+B;             /*suma de cuadrados de real e imag*/

                    if(C>=Id)               /*Compara C  contra Id*/
                                            /*Si se cumple la condición*/
                   {                        /*se incrementa Falla*/
                        Falla++;

                     If(Falla==3)          /*Sí Falla=3 se interrumpe la rutina */
                         {                 /*principal y  se genera la señal de */
                    curva_tinverso();     /* disparo                          */

                          }

                       }
                }

                k=7 ;   /*Se fija K=7 para regresar al los cálculos*/

            }


                 else           /*si K<8 se llena el arreglo de muestras  en  la */
                                /* en la función que maneja  la interrupción     */
              {                 /*del convertidor   */


                    i++;         /*se incrementa el contador para llenar el*/
                                 /*arreglo de muestras                       */

              }



               k++;             /*contador K, se incrementa en cada periodo*/
                                /*de muestreo                                */

    }

/*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
for(;;){}
```

```
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!!
***/
} /*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/

/* END relevador */
/*
**
#########################################################################
####
**
**     This file was created by UNIS Processor Expert 2.95 [03.62]
**     for the Freescale HCS12 series of microcontrollers.
**
**
#########################################################################
####
*/
```

MÓDULO DE EVENTOS

```
/**
#########################################################################
#
**     Nombre de archivo  : Events.C
**     Proyecto   : relevador
**     Procesador : MC9S12E64BCPV
**     Beantype  : Events
**     Versión   : Driver 01.03
**     Compilador  : Metrowerks HC12 C Compiler
**     Fecha/Hora : 16/05/06, 09:09 p.m.
**     Resumen  : Módulo de Eventos
**         This is user's event module.
**         Put your event handler code here.
**     Settings  :
**     Contents  :
**         AD2_OnEnd         - void AD2_OnEnd(void);
**         TI1_OnInterrupt    - void TI1_OnInterrupt(void);
**         EInt1_OnInterrupt - void EInt1_OnInterrupt(void);
**
**     (c) Copyright UNIS, spol. s r.o. 1997-2004
**     UNIS, spol. s r.o.
**     Jundrovska 33
**     624 00 Brno
**     Czech Republic
**     http      : www.processorexpert.com
**     mail      : info@processorexpert.com
**
#########################################################################
#*/
/* MODULE Events */


#include "Cpu.h"
#include "Events.h"
#include "variables.h"
#include "TDF.h"
#include “teclado.h”
/*
**
=========================================================================
====
```

```
**     Event       :  AD2_OnEnd (module Events)
**
**     From bean   :  AD2 [ADC]
**     Description :
**         This event is called after the measurement (which
**         consists of <1 or more conversions>) is/are finished.
**     Parameters  : None
**     Returns     : Nothing
**
================================================================
====
*/


void AD2_OnEnd(void)
{

  /* Write your code here ... */
 AD2_GetValue(&valor);         /*Toma una muestra*/

resu=valor*0.00500488758;   /*Se obtiene el valor de voltaje en formato flotante*/
resu= (resu-2.272219);          /*Se elimina el valor de la tensión de DC (offset)*/

 total= (valor/10);
 resu=valor/100;

                          if(k==8)
                         {
                                        /*Sí K=8*/
                  recorre_datos();   /*Recorre muestras*/
                  lect[7]=resu;         /*La siguiente lectura se guarda en lect[7]*/
                           }


                             else
                            {
                   lect[i]=resu;       /*Se llena el arreglo de muestras */

                            }

}

/*
**
================================================================
====
**     Event       :  TI1_OnInterrupt (module Events)
```

```
**
**     From bean   :  TI1 [TimerInt]
**     Description :
**         When a timer interrupt occurs this event is called (only
**         when the bean is enabled - "Enable" and the events are
**         enabled - "EnableEvent").
**     Parameters  : None
**     Returns     : Nothing
**
============================================================
====
*/

void TI1_OnInterrupt(void)
{

 if(counter)counter--;    /*Se decrementa el contador counter*/
                           /*esta interrupción periodica establece */
}                          /* el periodo de muestreo*/

/*
**
============================================================
====
**     Event       :  TI2_OnInterrupt (module Events)
**
**     From bean   :  TI2 [TimerInt]
**     Description :
**         When a timer interrupt occurs this event is called (only
**         when the bean is enabled - "Enable" and the events are
**         enabled - "EnableEvent").
**     Parameters  : None
**     Returns     : Nothing
**
============================================================
====
*/

void TI2_OnInterrupt(void)

{

  /* Write your code here ... */

       if(time)
     {
   time--;        /*Decrementa el contador time*/
```

```
    }              /*Esta interrupción periódica  fija el tiempo de disparo*/


         else

{
  TI2_Disable();    /*Desactiva interrupción */

Bit1_NegVal();    /*Activa Led de indicación de disparo */

     }



}

/*
**
================================================================
====
**     Evento       :  TI3_OnInterrupt (Módulo de eventos)
**
**     De la cápsula   :  TI3 [TimerInt]
**     Descripción :
**       Cuando una interrupción temporizada ocurre, este evento es llamado
**       (Solamente   cuando la cápsula esta habilitada "Enable"  y los eventos son
**       habilitados  - "EnableEvent").
**     Parámetros  : Ninguno
**     Retorna     : Nada
**
================================================================
====
*/
void TI3_OnInterrupt(void)
{
  /* Write your code here ... */


          if(contmseg) contmseg--;   /*Contador de la subrutina de LCD*/

}



/*
```

```
**
**===========================================================
====
**     Event      :  EInt1_OnInterrupt (module Events)
**
**     From bean   :  EInt1 [ExtInt]
**     Description :
**         This event is called when an active signal edge/level has
**         occurred.
**     Parameters  : None
**     Returns     : Nothing
**
**===========================================================
====
*/
void EInt1_OnInterrupt(void)
{
  /* place your EInt1 interrupt procedure body here */

 EInt1_Disable();          /*Desactiva interrupción externa 1*/
 barre_fila();             /*Barre fila de teclado*/
 decodi();                 /*Decodifica*/
 EInt1_Enable();           /*Activa interrupción externa 1*/


}


/*
**
**===========================================================
====
**     Event      :  EInt2_OnInterrupt (module Events)
**
**     From bean   :  EInt2 [ExtInt]
**     Description :
**         This event is called when an active signal edge/level has
**         occurred.
**     Parameters  : None
**     Returns     : Nothing
**
**===========================================================
====
*/
void EInt2_OnInterrupt(void)
{
  /* place your EInt2 interrupt procedure body here */
```

```
  EInt2_Disable();         /*Desactiva interrupción externa 2*/
  barre_fila();             /*Barre fila de teclado*/
  decodi();                 /*Decodifica*/
  EInt2_Enable();          /*Activa interrupción externa 2*/

}



/*
**
============================================================
====
**     Event       :  EInt3_OnInterrupt (module Events)
**
**     From bean   :  EInt3 [ExtInt]
**     Description :
**         This event is called when an active signal edge/level has
**         occurred.
**     Parameters  : None
**     Returns     : Nothing
**
============================================================
====
*/
void EInt3_OnInterrupt(void)
{
  /* place your EInt3 interrupt procedure body here */


  EInt3_Disable();     /*Desactiva interrupción externa 3*/
  barre_fila();         /*Barre fila de teclado*/
  decodi();             /*Decodifica*/
  EInt3_Enable();       /*Activa interrupción externa 3*/


}


/*
**
============================================================
====
**     Event       :  EInt4_OnInterrupt (module Events)
**
```

```
**     From bean   :  EInt4 [ExtInt]
**     Description :
**         This event is called when an active signal edge/level has
**         occurred.
**     Parameters  : None
**     Returns     : Nothing
**
================================================================
====
*/


void EInt4_OnInterrupt(void)
{
  /* place your EInt4 interrupt procedure body here */


  EInt4_Disable();       /*Desactiva interrupción externa 4*/

  barre_fila();          /*Barre fila de teclado*/
  decodi();              /*Decodifica*/
  EInt4_Enable();        /*Activa interrupción externa 4*/



}




/*
**
================================================================
====
**     Event       :  AS1_OnRxChar (module Events)
**
**     From bean   :  AS1 [AsynchroSerial]
**     Description :
**         This event is called after a correct character is
**         received.
**         DMA mode:
**         If DMA controller is available on the selected CPU and
**         the receiver is configured to use DMA controller then
**         this event is disabled. Only OnFullRxBuf method can be
**         used in DMA mode.
**     Parameters  : None
**     Returns     : Nothing
```

```
**
==========================================================================
====
*/

void AS1_OnRxChar(void)
{

  unsigned char recibe;

 static int r=0;
  /* Write your code here ... */


Bit3_NegVal();          /*Activa Led de indicación de recepción de datos*/


AS1_RecvChar(&recibe);

ajuste[r++]=recibe-48;    /*r se incrementa cuando llega un carácter */



if(r==2) {                        /*si r=2 se fija el ajuste recibido */


                  Ip= ajuste[0]*10+ajuste[1];

r=0;



}


}


/* END Events */

/*
**
###################################################################################
#
**
**     This file was created by UNIS Processor Expert 2.95 [03.62]
**     for the Freescale HCS12 series of microcontrollers.
```

```
**
**
#############################################################################
#
*/
```

SUBRUTINAS

****

Tdf.C

```
#include "Cpu.h"
#include "Events.h"
#include"TDF.h"
#include"variables.h"



void recorre_datos( void)        /* Esta función se utiliza para recorrer */
                                 /* las muestras                          */
{

     for(j=0;j<=11;j++)
    {

    lect[j]=lect[j+1];

    }

}

}
```

Fin de Tdf.C

****

****

curva_tinverso.C

```
#include "Cpu.h"
#include "Events.h"
#include "variables.h"
#include "curva_tinverso.h"
#include "math.h"


 float c1=0;
 float tiempo_disparo=0;
 int time=0;


void curva_tinverso( void)


{



                        TI1_Disable();   /*Deshabilita interrupción principal*/
                         I= sqrtf(C);   /*Se calcula la raíz cuadrada de C*/

  tiempo_disparo= (0.14/(((I/Id)^0.02) -1).17;  /*Curva de tiempo inverso*/



        time= tiempo_disparo*1000;  /*Se multiplica por 1000  para poder */
                                     /* decrementar el contador time   con   */
                                    /*la interrupción periódica                  */

                              TI2_Enable();   /*Habilita interrupción periódica para */
                                              /* generar el tiempo de disparo*/



}
```

Fin de curva_tinverso.C

****

****

retardo.c

```
#include "Events.h"
#include "variables.h"
#include "retardo.h"
void retardo(unsigned int time)

{

counter= time;
      while(counter)

           {
           asm(nop);
            }

}
```

Fin retardo.c

****

****

teclado.c

```
/* Autor: Roberto Galicia Galicia */

#include "Cpu.h"
#include "Events.h"
#include "variables.h"
#include "teclado.h"

  unsigned char fila;
  unsigned char col;

const unsigned char tabla_filas[4]=              /*Tabla de filas*/
{
  0xE0, //0
  0xD0, //1
  0xB0, //2
  0x70, //3
```

```
} ;


void barre_fila(void)                /* Función para barrer filas*/
  {

   unsigned char ind_tabla_filas=0;

for(i=0;i<=10000;i++);

   PTADLo = tabla_filas[ind_tabla_filas++];
      fila = (PTADLo & 0x0F);

          while(fila==0x0F)
           {
              PTADLo = tabla_filas[ind_tabla_filas++];
    if (ind_tabla_filas>3) ind_tabla_filas=0;
               fila=PTADLo & 0x0F;
    }
  }


void decodi(void)                /*Función para decodificar teclas presionadas*/
  {

   static int m=0;

  fila=PTADLo&0xf0;

  if(fila==0X70)
    tec[6]=0x00;
  if(fila==0XB0)
    tec[6]=0x04;
  if(fila==0XD0)
    tec[6]=0x08;
  if(fila==0XE0)
   tec[6]=0X0C;



   col = PTADLo&0x0f;
   if(col==0X0E)
    tec[6]=tec[6]+0;
  if(col==0X0D)
    tec[6]=tec[6]+1;
  if(col==0X0B)
```

```
   tec[6]=tec[6]+2;
  if(col==0X07)
   tec[6]=tec[6]+3;

//*******BINARIO-ASCII*******//
   tec[6] = tec[6] + 0x30;
   if(tec[6]>0x39)
     tec[6] = tec[6] + 7;
// ******************************//
   fila=PTADLo&0x0F;


                        ajuste[m++]=tec[6];     /*m se incrementa al ingresar un */
                                                /*número*/

                       if(m==2)     /*si m=2 se fija el ajuste ingresado*/
                          {
                           lp= (ajuste[0]-48)*10+(ajuste[1]-48);

                          m=0;
                         }
   while(fila != 0x0F)
   {
     fila = PTADLo & 0x0F;
   }
  for(i=0;i<=10000;i++);
  PTADLo = 0x0;


  }
```

Fin teclado.c

****

****

subLCD.c

```
/* LCD
   Rutinas para manejo del Display de Cristal Líquido
 hardaware utilizado para el display
 para display de 2 lineas  20

Autor: Luís Puebla Palma

1 GND
2 VCC
```

```
3 Contraste  POT 2
4 RS         PTB1 (11)
5 RW         PTB2 (12)
6 E          PTB3 (13)
7  d0        no conectado
8  d1        no conectado
9  d2        no conectado
10 d3        no conectado

11 d4        PTB4
12 d5        PTB5
13 d6        PTB6
14 d7        PTB7

                                                */


#include "subLCD.h"
#include "mc9s12e128.h"
#include "variables.h"
#include "retardo_lcd.h"
#define  BLINK                  0x01
#define  CURSOR                 0x02
#define  DISPLAY                0x04
#define  DISPLAY_ON_CTRL   0x08
#define  CGRAM_SET              0x40
#define  DDRAM_SET              0x80
#pragma CODE_SEG DEFAULT


void ini_LCD(void)
{

  DDRB = 0xFF;   // todo en salidas
  LCD_E_OFF();
  LCD_RS_OFF();
  LCD_RW_OFF();

  retardo_lcd(16);   // espera mas de 15 mseg
  retardo_lcd(5);     //espera mas de 4.1 mseg
  esc_lcd(0x30);  //1  interface data length is 8 bits
  retardo_lcd(5);      //espera mas de 4.1 mseg
  esc_lcd(0x30);  //2  interface data length is 8 bits
  retardo_lcd(110);    // espera mas de 100mseg
  esc_lcd(0x30);  //3  interface data length is 8 bits
  esc_lcd(0x20);  //4  interface data length is 4 bits
```

```
  // Ya esta en modo de 4 bits

  esc_lcd2(0x28);             // Funcion set, 2 lineas 5x7 dots.
  esc_lcd2(0x0C);             // display on; Incrementa.
  esc_lcd2(0x01);             // display clear.


}

void Despliega_l2(char *cad)  {             //puntero a arreglos
  unsigned int i=0;
  char ch;
  ch=cad[i];
  HazComandoDisplay(DISP_Return_Home);
  //LCD_RS_OFF();   //lo que sigue es comando

  esc_lcd2(0xC0);
  while(i<16 && ch!=0)
   {
   LCD_RS_ON();   //ya son datos
   esc_lcd2(ch);
    i++;
    ch=cad[i];

   }
 }



void Despliega_L1(char *cad){
  unsigned int i=0;
  char ch;
  ch=cad[i];
  HazComandoDisplay(DISP_Return_Home);
    // LCD_RS_OFF();   //lo que sigue es comando

  while(i<16 && ch!=0)
   {
   LCD_RS_ON();   //ya son datos
   esc_lcd2(ch);
   if(i==7)   // Ya escribió el dato 8 y se prepara para escribir en la segunda línea
      {
        LCD_RS_OFF();   //ya son datos
        esc_lcd2(0xC0);
      }

    i++;
```

```
    ch=cad[i];

  }
}
```

Fin de subLCD.c

****

****

sublcd.h

```
/* Inicio de encabezado
    Nombre archivo: subLCD.h
    Proyecto:       sis68E128
    Descripción   : Rutinas básicas para uso de Display de cristal líquido  LCD
    Procesador    : MC689S12E128
    Revisión      : Borrador
    Fecha         : 6/03/04
    Autor         : Luis Puebla Palma
    Compilador    : CodeWarrior
   Fin de encabezado
 */

/* Archivos *.h */

//#include <hidef.h>            /* Definiciones comunes y macros */
//#include <mc9s12e128.h>


#define LCD_RS 0x02
#define LCD_RW 0x04
#define LCD_E  0x08
#define LCD_E_ON()   asm(bset PORTB,LCD_E)
#define LCD_E_OFF()  asm(bclr PORTB,LCD_E)
#define LCD_RS_ON()  asm(bset PORTB,LCD_RS)
#define LCD_RS_OFF() asm(bclr PORTB,LCD_RS)
#define LCD_RW_ON()  asm(bset PORTB,LCD_RW)
#define LCD_RW_OFF() asm(bclr PORTB,LCD_RW)


/* Comandos para el display */
#define DISP_Return_Home  0x02
#define DISP_ClearDisplay 0x01
```

```
/* Funciones subLCD */
void Ini_LCD(void);
void HazComandoDisplay(unsigned char comando);
void Despliega_L1(char *cad);
void Despliega_l2(char *cad);
void Despliega_L2(char *cad);
```

Fin sublcd.h

VARIABLES EXTERNAS

```
extern unsigned int valor;
extern float resu;
extern unsigned int k;
extern unsigned int i;
extern unsigned int j;
extern unsigned int count;
extern float lect[8];
extern float C;
extern float real;
extern float imag;
extern unsigned int t_muestreo;
extern int time;
extern unsigned int contmseg;
extern unsigned char total;
extern unsigned  char ajuste[16];
extern unsigned char tec[16];
extern float Id;
extern unsigned char recibe;
```