

**UNIVERSIDAD AUTÓNOMA DE BAJA CALIFORNIA**

**FACULTAD DE INGENIERÍA**

**MAESTRÍA Y DOCTORADO EN CIENCIAS E INGENIERÍA**



**“EVALUACIÓN DEL CONSUMO DE ENERGÍA A NIVEL DE  
CICLOS DE RELOJ EN UN MICROCONTROLADOR”**

**TESIS PARA OBTENER EL GRADO DE  
MAESTRO EN CIENCIAS**

**PRESENTADA POR  
MARIO ALBERTO CAMARILLO RAMOS**

**DIRECTORES DE TESIS:  
Dr. Miguel Enrique Bravo Zanoguera  
Dr. Roberto López Avitia**

**MEXICALI, B. C.**

**DICIEMBRE DE 2016**

# Resumen

Las actividades que realizan la mayoría de los aparatos electrónicos utilizados cotidianamente se desarrollan predominantemente con la ayuda de un microcontrolador por triviales que sean. Dichas actividades se ejecutan utilizando energía de baterías o de fuentes de energía fija. El consumo de energía aumenta en la medida que las actividades desarrolladas por estos aparatos se incrementa, tal es el caso del Internet de las cosas donde no solo es necesario realizar la tarea indicada, también se debe transferir la información inalámbricamente; el monitoreo de signos vitales en instrumentos médicos donde es imperativo tener la información confiable y transferirla a posibles actuadores. Estas aplicaciones entre otras han convertido el consumo de energía en un parámetro de suma importancia para el diseño de dichos instrumentos.

Los fabricantes de microcontroladores proporcionan herramientas de simulación y medición para sus dispositivos donde el consumo de energía que ellos señalan tener es optimizado tanto en software como hardware para obtener un valor de consumo menor. Si uno desea realizar una comparativa del consumo de energía a través de diferentes fabricantes no es posible coincidir en una métrica, aunque el *Embedded Microprocessor Benchmark Consortium* (EEMBC) intenta hacerlo con su *CoreMark* pero deben tener ciertas características los microcontroladores; una de ellas es tener un RTC interno y no todos los microcontroladores cuentan con este periférico.

Uno de estos fabricantes de microcontroladores es Microchip, que ha vendido billones de unidades tanto de 8 y 16 bits. Cuenta con una línea dedicada para aplicaciones de bajo consumo de energía: la XLP. Debido a la popularidad de esta compañía y al fácil acceso de microcontroladores se optó por utilizar un dispositivo de 16 bits que tuviera la característica de operar con un bajo consumo de energía.

El presente muestra el consumo de energía del microcontrolador PIC24FV32KA302 de la compañía Microchip con una resolución en ciclos de reloj. Se utiliza una batería de pruebas diseñada para ejecutar operaciones aritméticas a diferentes frecuencias de operación. Las operaciones se implementan en lenguaje de alto nivel “C” y el lenguaje ensamblador generado por el compilador PICC de Custom Computer Services es evaluado utilizando la medición de corriente en cada ciclo de reloj.

El estudio produce información sobre la energía consumida por el microcontrolador: es lineal solo hasta la frecuencia de 4 MHz. En la última frecuencia de operación que es de 8 MHz el consumo de energía decrece en un 10% respecto al incremento casi lineal del 9% de las frecuencias anteriores, las cuales son 500 KHz, 1 MHz, 2 MHz y 4 MHz. No se realizaron pruebas para frecuencias mayores de 8 MHz, aunque es posible generar frecuencias internas a 16 MHz y 32 MHz. El microcontrolador cuenta con un módulo interno de 8 MHz, para frecuencias mayores es necesario activar el PLL.

El mayor consumo de energía se presenta en la sección para despertar del algoritmo (*Wake Up*), teniendo como resultado un 84% en promedio del consumo total en todas las frecuencias y algoritmos diferentes.

Se obtiene información sobre la implementación en ciclos de reloj para las operaciones aritméticas, teniendo como resultado un incremento del 3.5% de 8 bits a 16 bits y de un 40% en promedio de 16 bits a 32 bits y de 32 bits a 64 bits. La arquitectura de 16 bits del microcontrolador propicia ese incremento de solo 3.5% ya que se consumen los mismos ciclos de reloj utilizando variables u operaciones de 8 bits y 16 bits.

El pulso de sincronía utilizado es el catalizador para realizar las mediciones tanto en ciclos de reloj como en energía.

La investigación genera una solución para la compañía Custom Computer Services. El compilador produce código necesario para configurar el oscilador interno si se le asigna una frecuencia que pueda generar internamente. Si se asigna una frecuencia la cual no pueda ser generada, el compilador asigna una configuración a los bits de control y protege al microcontrolador de cualquier escritura posterior. Se notificó a la compañía y corrigieron este error del compilador en versiones subsecuentes.

# Abstract

Most of the activities that electronics devices execute are done with a microcontroller, however trivial they might be. Such activities are done by using the energy stored in batteries or by using constant power supplies. Energy consumption rises as these activities become more complex, such is the case of The Internet of Things where is not only necessary that the device execute its task, but it also needs to send information wirelessly; monitoring vitals in medical instruments where is imperative to have reliable information and transfer it to possible actuator. These applications among others have made energy consumption an important parameter to consider in the design process.

Microcontroller manufacturers provide simulation and a measurement tools for their integrated circuits where the energy consumption that allegedly is consumed is optimized both in software and hardware for it to provide the least consumed energy. If one wishes to compare energy consumption between different microcontroller manufactures it is not possible to coincide in a standard metric, although the *Embedded Microprocessor Benchmark Consortium* (EEMBC) tries to do so with its *CoreMark* metric but the microcontroller needs to have certain characteristics for this metric to be measured; one of these is the that it needs an internal RTC and not all microcontrollers have this peripheral.

One of these microcontroller manufactures is Microchip, which has sold billions of 8 and 16 bit units. It has a line of low power consumption microcontrollers called XLP. Due to its popularity and ease of access to their products, this company was chosen for the study. A low power 16 bit microcontroller is to be evaluated.

This study evaluates the energy consumption of Microchip's PIC24FV32KA302 microcontroller within clock cycle resolution. A battery of tests is designed to execute arithmetic operations at different operation frequencies. These operations are implemented in high level language "C" and the generated assembler by Custom Computer Services PICC compiler is evaluated measuring the current consumption of every clock cycle.

The study shows that the energy consumption by the microcontroller is almost linear up to the 4 MHz frequency. In the last operation frequency which is 8 MHz the energy consumption was reduced by 10% with respect to the almost 9% linear increase from the other frequencies, which are 500 KHz, 1 MHz, 2 MHz and 4 MHz. No tests were

conducted beyond 8 MHz, although it can be implemented internally to 16 MHz and 32 MHz. The microcontroller has an internal oscillator module with an 8 MHz frequency, for greater frequencies one needs to activate its PLL.

Most of the energy consumption was in the *Wake Up* section of algorithm in the microcontroller, consuming in average 84% of the total energy of every algorithm and frequencies.

Information is also obtained regarding clock cycles implementation for the arithmetic operations, resulting in a 3.5% increase from 8 bits to 16 bits and a 40% on average increase from 16 bits to 32 bits and from 32 bits to 64 bits.

The sync pulse serves as the trigger to measure clock cycles as well as energy consumption.

The study generated an improvement to the PICC compiler. It generates the appropriate code to configure its internal oscillator if an adequate frequency is suggested. If a frequency cannot be implemented but it is assigned anyway, the compiler generates configuration bits that block (protect) from future writes to the program memory. This issue was notified to CCS and they fixed it a few compiler revisions later.

## **Agradecimientos**

A la Facultad de Ingeniería de la UABC por facilitar los recursos necesarios a través de CONACYT para la realización de esta investigación.

Al Dr. Miguel Enrique Bravo Zanoguera por haber sido el director de tesis. Por sus aportaciones, sobre todo en el pulso de sincronía para la adquisición del perfil de energía y en las atenciones cuando fue necesario tomar tiempo para la familia.

Al Dr. Roberto López Avitia por haber sido codirector de tesis. Por alentar siempre a realizar un posgrado y por sus valiosas aportaciones para el análisis de resultados.

Al Dr. Marco Antonio Reyna Carranza por haber sido codirector de tesis. Por todo su apoyo y atenciones brindadas.

Al Dr. Guillermo Galaviz por sus comentarios sobre el por qué es importante hacer investigación y la relación que tiene con la familia.

A lo profesores del posgrado MYDCI por su capacitación en las áreas del conocimiento que proporcionaron.

A mi esposa Ivonn por apoyarme siempre y de manera incondicional en todo. En las muy buenas y muy malas.

A mi hija Ivanna Mariela, por llegar a mi vida justo cuando estaba por terminar el posgrado y reestructurar todas mis prioridades.

A mis padres Mario y Graciela por mostrarme cómo se debe conducir una persona de bien.

A mis hermanas Mirna y Nadia por soportar todos mis comentarios sobre electrónica desde la preparatoria, ahora con el consumo de energía y microcontroladores.

# Índice

Índice de tablas.....	i
Índice de figuras .....	ii

## Capítulo 1

### Introducción

1.1 Antecedentes.....	1
1.2 Planteamiento del problema y justificación del estudio .....	4
1.3 Hipótesis.....	4
1.4 Objetivo general.....	5
1.4.1 Objetivos específicos .....	5

## Capítulo 2

### Marco teórico

2.1 Consumo de energía .....	6
2.1.1 Perfil de energía.....	7
2.1.2 Técnicas para bajo consumo de energía en Hardware.....	9
2.1.3 Técnicas de bajo consumo de energía en Software .....	10
2.2 Medición de energía.....	11
2.2.1 Bobina de Rogowski .....	12
2.2.2 Espejo de corriente .....	12
2.2.3 Carga en capacitor.....	13
2.2.4 Efecto Hall.....	13
2.2.5 Resistencia <i>shunt</i> .....	14
2.2.5.1 Medición de lado alto .....	15
2.2.5.2 Medición de lado bajo .....	16
2.2.6 Soluciones comerciales.....	17

## Capítulo 3

### Metodología

3.1 Algoritmo.....	18
3.1.1 Código del algoritmo y bits de configuración.....	19
3.1.2 Pulso de sincronía y ciclos de reloj .....	25
3.2 Equipo y material de medición .....	33
3.2.1 El microcontrolador.....	34
3.2.2 Osciloscopio.....	35
3.2.2.1 Configuración para medición de voltaje .....	35
3.2.2.2 Pulso de sincronía .....	37
3.2.3 Fuente de voltaje .....	37
3.3 Postprocesamiento .....	37
3.3.1 Perfil de energía completo.....	38
3.3.2 Perfil de energía por secciones .....	43

## Capítulo 4

### Resultados y discusión

4.1 Ciclos de reloj del algoritmo .....	46
4.2 Energía del algoritmo .....	49
4.2.1 Energía <i>Wake Up</i> y total .....	49
4.2.2 Energía del algoritmo para 8 bits .....	51
4.2.3 Energía del algoritmo para 16 bits .....	54
4.2.4 Energía del algoritmo para 32 bits .....	57
4.2.5 Energía del algoritmo para 64 bits .....	60
4.3 Conclusiones.....	63

<b>Bibliografía.....</b>	<b>65</b>
--------------------------	-----------

### Anexo 1

<b>Correo de Microchip por problema de hardware .....</b>	<b>71</b>
---	-----------

### Anexo 2

<b>Correo de CCS por problema de software .....</b>	<b>79</b>
---	-----------



## **Índice de tablas**

Tabla 1 Ciclos de reloj para cada sección del algoritmo .....	47
Tabla 2 Consumo de energía de 8 bits por secciones.....	51
Tabla 3 Consumo de energía de 16 bits por secciones .....	54
Tabla 4 Consumo de energía de 32 bits por secciones .....	57
Tabla 5 Consumo de energía de 64 bits por secciones .....	60

## Índice de figuras

Fig. 1 Perfil de energía típico.....	8
Fig. 2 Medición con espejo de corriente.....	12
Fig. 3 Medición de corriente por lado alto .....	15
Fig. 4 Medición de corriente por lado bajo.....	16
Fig. 5 Diagrama de flujo del algoritmo .....	19
Fig. 6 Código del algoritmo de a) 8 bit b) 16 bits .....	20
Fig. 7 Código del algoritmo de a) 32 bits b) 64 bits.....	21
Fig. 8 Implementación de funciones de a) 8 bit b) 16 bits.....	22
Fig. 9 Implementación de funciones de a) 32 bits b) 64 bits .....	22
Fig. 10 Bits de configuración.....	24
Fig. 11 Pulso de sincronía.....	25
Fig. 12 Medición del pulso de sincronía .....	26
Fig. 13 Ensamblador y ciclos de reloj.- C0 .....	27
Fig. 14 Ensamblador y ciclos de reloj.- Configuración.....	28
Fig. 15 Ensamblador y ciclos de reloj.- Pulso de sincronía.....	28
Fig. 16 Ensamblador y ciclos de reloj.- Inicialización de variables.....	29
Fig. 17 Ensamblador y ciclos de reloj a) Función suma b) Subrutina 64 bit .....	29
Fig. 18 Ensamblador y ciclos de reloj a) Función resta b) Subrutina 64 bit.....	30
Fig. 19 Ensamblador y ciclos de reloj a) Función multiplicación b) Subrutina 64 bit .....	31
Fig. 20 Ensamblador y ciclos de reloj a) Función división b) Subrutina 64 bit .....	32
Fig. 21 Ensamblador y ciclos de reloj.- WDT .....	32
Fig. 22 Diagrama a bloques de la medición .....	33
Fig. 23 Esquemático de la medición .....	34
Fig. 24 Configuración estándar del osciloscopio .....	36
Fig. 25 Diagrama de flujo para el postprocesamiento.....	38
Fig. 26 Creación de vector para ajuste de tiempo .....	39
Fig. 28 Creación de la nueva señal con la base de tiempo correcta.....	40
Fig. 29 Perfiles de energía de voltaje y corriente superpuestas .....	41
Fig. 30 Cálculo de la señal de potencia .....	42
Fig. 31 Cálculo de la energía total .....	43

Fig. 32 Medición de los pulsos de reloj del algoritmo utilizando 64 bits a 8 MHz .....	44
Fig. 33 Perfil de energía por secciones del algoritmo con variables de 64 bits a 8 MHz ....	45
Fig. 34 Ciclos de reloj del algoritmo en cada programa.....	48
Fig. 35 Ciclos de reloj del algoritmo .....	48
Fig. 36 Wake Up y Totales .....	50
Fig. 37 Consumo de energía Total para 8 bits .....	51
Fig. 38 Dispersión de energía por frecuencias con 8 bits .....	52
Fig. 39 Consumo de energía por secciones para 8 bits .....	53
Fig. 40 Consumo de energía Total para 16 bits .....	54
Fig. 41 Dispersión de energía por frecuencias con 16 bits .....	55
Fig. 42 Consumo de energía por secciones para 16 bits.....	56
Fig. 43 Consumo de energía Total para 32 bits .....	57
Fig. 44 Dispersión de energía por frecuencias con 32 bits .....	58
Fig. 45 Consumo de energía por secciones para 32 bits.....	59
Fig. 46 Consumo de energía Total para 64 bits .....	60
Fig. 47 Dispersión de energía por frecuencias con 64 bits .....	61
Fig. 48 Consumo de energía por secciones para 64 bits.....	62

# Capítulo 1

## Introducción

### 1.1 Antecedentes

La energía en un dispositivo electrónico es indispensable para su correcta operación y funcionamiento. El consumo de energía es una preocupación creciente en la implementación de tareas en dispositivos electrónicos, ya sea para el denominado Internet de las Cosas con aplicaciones industriales (*IIoT*) [1] o para sistemas de monitoreo portátiles de variables médicas (ECG) [2]. Es de particular interés lo referente a sistemas médicos, principalmente los de monitoreo.

Los sistemas de monitoreo para signos vitales dependen tanto del hardware como software para operar en armonía. La necesidad de que estos sistemas consuman una menor cantidad de energía es de vital importancia. También lo es cumplir con las regulaciones de organismos oficiales para el diseño de dispositivos médicos.

En la cuestión de hardware existen grandes avances para mejorar el consumo de energía utilizando técnicas de diseño e implementación. En cuestión de software no es tan claro ese rubro.

En la actualidad existen sistemas de monitoreo de variables físicas médicas que ayudan al operador, sea paciente, técnico paramédico o médico, a tomar decisiones críticas en función de los parámetros mostrados. Dichos instrumentos pueden ser móviles o

estacionarios. Tanto los móviles como estacionarios cuentan con baterías de respaldo para en caso de contingencia, continuar con su operación.

Aunque existen regulaciones y clasificaciones para instrumentos médicos [3] y software crítico [4] para ellos, no establecen un método para evaluar la capacidad de operación móvil o con baterías.

En este departamento se observa un área de oportunidad ya que en el Estado de Baja California existen 26 empresas [5] dedicadas a la industria de manufactura o diseño de insumos o instrumentos médicos entre ellos los de monitoreo, sin contar los más de 2000 hospitales que se encuentran en el Estado [6].

La generación de investigación y tecnología para el bajo consumo de energía es de suma importancia en la creación de sistemas capaces de subsistir periodos prolongados con una fuente de energía finita. Cobra mayor importancia cuando se pretende aplicar ese conocimiento en instrumentos o equipos encargados de monitorear la salud de las personas. Además de las cuestiones de ahorro de energía, es importante cumplir con las especificaciones internacionales para estos instrumentos médicos. La manera de estructurar el software y generar la documentación apropiada sugiere una revisión en cómo el dispositivo interactúa con la metodología de programación implementada por el fabricante sin descuidar los nuevos estándares para software médico que permitirán reducir los problemas inherentes de programación.

La FDA (Food and Drug Administration de Estados Unidos) ha realizado recomendaciones para retirar del mercado bombas de infusión debido a los más de 56,000 reportes de incidentes desde 2005 a 2009 donde los principales son problemas de software [7]. También sugiere retirar defibriladores automáticos externos en función de los más de 45,000 reportes de fallas de estos dispositivos entre 2005 y 2012 donde los errores de software también se encuentran entre los de mayor incidencia [8].

Se han realizado investigaciones donde es de interés los problemas relacionados con software médico dando como resultado hasta un 98% de ellos prevenibles o detectables aplicando técnicas básicas de medición de errores [4].

Lo anterior despertó interés para crear un estándar específico para software médico, resultando en la IEC-62304:2006. Esta norma fue sugerida en 2006 [9] y aceptada en 2010.

El Centro Nacional de Excelencia Tecnológica en Salud (CENETEC) establece en su documento para normas la IEC 62304 [10]. Dicha norma establece regulaciones en el desarrollo del software para dispositivos médicos. Estas medidas se acentúan con los problemas serios que pueden crear dispositivos médicos que no sean rigurosamente probados en software, tal es el caso de las muertes provocadas por el Therac-25 [11].

Mathworks ha presentado información respecto a este asunto, sugiriendo metodología para crear software y validarlo para equipo médico [12].

El estándar define a un dispositivo médico como cualquier instrumento, aparato, equipo, software, material u otro artículo, si se utiliza solo o en combinación con otros, incluyendo el software propuesto por el desarrollador para ser utilizado como diagnóstico y/o terapéutico. La inclusión de la palabra software es ahora de suma importancia, debido a que está condicionando una parte del proceso de desarrollo antes no contemplado [13]

Propone tres categorías de software médico basado en el posible daño que presenta al usuario o paciente:

- Clase A.- Ningún daño o lesión a la salud es posible.
- Clase B.- Daños o lesiones no catalogados como serios son posibles.
- Clase C.- Muerte o lesiones catalogadas como serias son posibles.

El estándar define “SOUP” (Software de procedencia desconocida) a cualquier software que no tenga una documentación formal o que fue desarrollado por otros grupos y no cuente con evidencia del control utilizado para su desarrollo. Este software puede influir en la categoría para el grado médico, ya que no existe manera de analizar el posible daño o riesgo que presenta [3].

La adopción del estándar presentará ventajas a los usuarios, ya que tendrán la certeza de utilizar equipos médicos que cuenten con la seguridad necesaria no solamente en hardware, también en software. No solamente ayudará a los pacientes, también lo hará con el mejoramiento del proceso para el software de los desarrolladores [14].

El estándar sugiere prácticas apropiadas para la documentación del software, sin embargo, no presenta cómo impacta en el consumo de energía del dispositivo. Para ello, es necesario medir el consumo de energía una vez aplicadas las sugerencias del estándar.

## **1.2 Planteamiento del problema y justificación del estudio**

Los microcontroladores y microprocesadores se han convertido en un aspecto importante de la vida cotidiana. Se utilizan en la mayoría de las actividades que realizamos y probablemente ni idea tenemos que uno de estos realiza un trabajo para nosotros. En la medida que se demanda mayor actividad de ellos, también lo hacen ellos de las fuentes que proporcionan la energía para que operen. Estas actividades aumentan en complejidad y por ende incrementan los recursos internos necesarios para realizar dichas actividades. Estos recursos internos adicionales demandan energía que ahora debe considerarse para operar. Si es posible establecer cuánta energía demandan, será factible realizar adecuaciones para que consuman menores cantidades. Algunos fabricantes de microcontroladores se han dado a la tarea de evaluar sus productos y lo realizan con herramienta especializada, tal es el caso de [15] y [16]. El hardware y software utilizado es optimizado para aprovechar las características específicas del elemento en prueba. Es aquí donde la evaluación para el consumo de energía de los dispositivos se complica, ya que no es posible utilizar una herramienta para establecer el consumo de energía a través de las diferentes compañías.

Si es posible evaluar el consumo de energía independientemente del fabricante se tendrá una herramienta que permita determinar dónde se consume mayor energía en el código (incluyendo el hardware). En el presente se pretende generar una metodología que permita evaluar el consumo de energía independiente del compilador (parte de software) y del dispositivo (parte de hardware).

## **1.3 Hipótesis**

Medir el consumo de energía en microcontroladores a nivel de ciclos de reloj permite analizar su comportamiento para optimizar el código y consumir menos energía.

## **1.4 Objetivo general**

Generar una metodología para la medición de energía en microcontroladores.

### **1.4.1 Objetivos específicos**

- Establecer una metodología para la adquisición del parámetro de energía en microcontroladores.
- Realizar pruebas de adquisición de energía.
- Documentar los resultados y presentar un análisis de los mismos.



## Capítulo 2

### Marco teórico

En este capítulo se presentan los antecedentes la metodología utilizada para la medición de la energía consumida por el microcontrolador. Se sugieren dos secciones: en primera instancia se presenta información sobre el impacto que tiene el software y el hardware en el consumo de energía y técnicas para reducir dicho consumo. La segunda es referente a la medición de corriente y energía en dispositivos que se estén probando (Dispositivo Bajo Prueba).

#### 2.1 Consumo de energía

La utilización de elementos inteligentes que controlen cuándo realizar tareas o no (microcontroladores principalmente) han permitido que dispositivos alimentados con fuentes de energía finita (baterías) extiendan su tiempo de vida. Es posible utilizar técnicas en estos dispositivos para el manejo de los diferentes módulos internos, así como los externos para hacer más eficiente el consumo de energía.

Así como existen métodos de monitoreo de consumo de energía de baterías para casa-habitación [15] que permiten el uso eficiente de la energía almacenada, es posible utilizar esas técnicas en microcontroladores. El microcontrolador puede gestionar el manejo de los módulos internos, así como los elementos externos para hacer más eficiente el consumo de energía. Por ejemplo en [16], un dispositivo que consume  $5\text{ mA}$  al operar de manera normal, polarizado con una batería alcalina estándar de  $2\text{ A}$ , podría estar en funcionamiento durante tres meses aproximadamente. El mismo dispositivo con técnicas

apropiadas para la utilización de módulos de baja potencia, optimizando los recursos cuando está encendido y no, haciendo eficiente el algoritmo de procesamiento podría bajar el consumo de corriente a  $500\mu A$ , extendiendo de esta manera la vida de la batería al doble, es decir, seis meses.

Las técnicas a las que se hace referencia principalmente se dividen en dos: hardware y software [17]. Las relacionadas con hardware sugieren utilizar diferentes tecnologías para el semiconductor y las uniones entre los diversos dispositivos a implementar. Esta configuración no es modificable por el usuario final. Varios fabricantes de microcontroladores cuentan con esas características de ahorro de energía integradas [18] [19] (Energymicro, Renesas, Texas Instruments, Microchip, entre otros). La optimización debido a la arquitectura del microcontrolador se logrará estructurando el conjunto de comandos del dispositivo para una ejecución armónica entre una instrucción y otra. Algunas de estas compañías proveen herramientas que calculan el consumo de energía de cada instrucción en un bloque de código. Con este resultado es posible estimar el perfil de energía [20].

Las categorías de manejo de la energía por medio del software son divididas en análisis de técnicas de manejo a nivel de código fuente, a nivel de algoritmo y arquitectura del software, y en esta categoría es donde el usuario final tiene mayor control del consumo de energía [17].

### **2.1.1 Perfil de energía**

La definición de bajo consumo de energía varía de una aplicación a otra. Para este experimento se considera el reducir al máximo la energía consumida por el microcontrolador. En algunos sistemas, existen fuentes de alimentación que no tendrán problema en suministrar la energía necesaria para la aplicación, sin embargo se puede optimizar el consumo para reducir costo o maximizar eficiencia [21]. Cuando la fuente de energía es limitada (baterías) o el sistema energizado se desconecta y debe operar con la

fuente secundaria (normalmente baterías), la operación del sistema dependerá de la capacidad de la fuente y en esta situación los sistemas requieren un análisis del perfil de energía. En los dispositivos electrónicos que utilizan la tecnología CMOS, como los microcontroladores y microprocesadores, la energía total que consume puede dividirse en dos rubros: la energía dinámica y estática. La energía dinámica es la que consume el dispositivo cuando está ejecutando alguna rutina del proceso. La energía estática es la que consume el dispositivo simplemente al estar en reposo y energizado [22]. En la figura 1 se observa una representación típica del perfil de energía. Se cuenta con un área donde el microcontrolador realiza actividades relacionadas con el consumo de energía nominal (inicio, despertar de un reposo, código a desarrollar de la aplicación). Se muestra esta actividad con el color amarillo en la figura 1. En color azul se tiene la energía estática, la cual se produce cuando se configura el microcontrolador en modo de reposo o *sleep*.

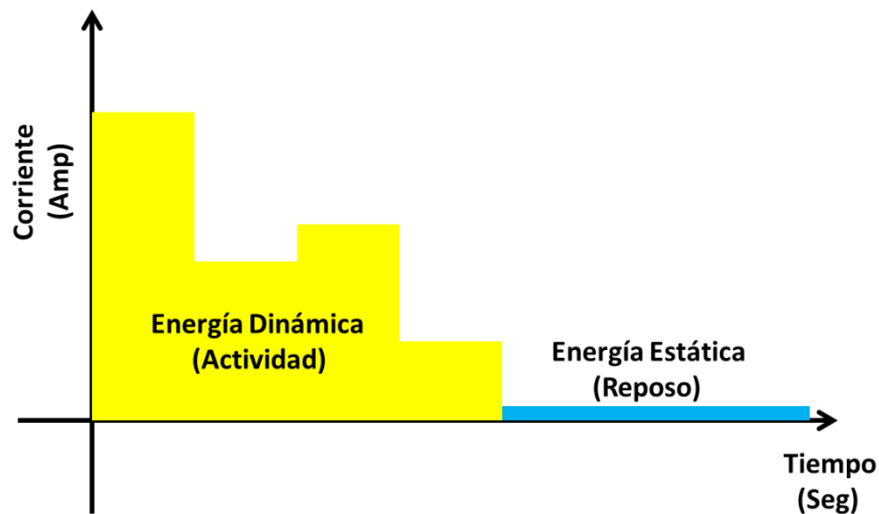


Fig. 1 Perfil de energía típico

### 2.1.2 Técnicas para bajo consumo de energía en Hardware

La Energía Dinámica incluye la pérdida de energía cuando los circuitos CMOS conmutan y las corrientes que polarizan los circuitos analógicos (osciladores, ADC, DAC, entre otros). Dicha Energía Dinámica se calcula para cada uno de los circuitos de la siguiente manera como se muestra en la ecuación 1:

$$E_{Dinámica} = V^2 \cdot f \cdot C \quad \text{Ecuación (1)}$$

donde  $V$  es el voltaje de operación ( $V_{DD}$ ),  $f$  es la frecuencia de conmutación y  $C$  es la capacitancia de carga de cada uno de los elementos CMOS [22] [23]. La frecuencia de conmutación es particular para cada diseño del fabricante, por lo que no se tiene control sobre este parámetro. La capacitancia de carga  $C$  es complicada medirla y por ende controlarla. El único parámetro accesible para el usuario es el voltaje de polarización  $V_{DD}$ . El reducir el voltaje de polarización es una forma de disminuir la energía consumida por el microcontrolador.

Otras técnicas específicas para el hardware se sugieren para las terminales no utilizadas del microcontrolador. Dichas terminales pueden ser configuradas como entradas o salidas. Si optan por entradas, deben estar conectadas con resistencias de jalón hacia  $V_{DD}$  o  $V_{SS}$ . Si la alternativa es configurarlas como salidas es posible dejarlas “flotando” pero deben generar un estado lógico para que el microcontrolador procure siempre tener un valor de voltaje estable en dicha terminal y no active o desactive la región lineal del circuito CMOS interno [21] [22] [24].

Por último se cuenta con las configuraciones de bajo consumo de energía específicos de los fabricantes. Aunque estas opciones pueden activarse en la etapa de preprocesamiento y/o en la ejecución del algoritmo y podrían considerarse como parte del software, se consideran parte del hardware debido a que son específicamente diseñadas en los microcontroladores para esas funciones [20] [21] [22] [25]. Dichas configuraciones permiten la desactivación o activación de sistemas internos y podrían optimizar el consumo dependiendo la aplicación.

### 2.1.3 Técnicas de bajo consumo de energía en Software

Tiwari [26] presenta en sus investigaciones el impacto del código generado en relación a la energía consumida por el sistema. Experimentó diferentes instrucciones y técnicas de programación para determinar cuánta energía consumía el programa. Obtuvo resultados favorables al reducir el consumo de energía en un 40% y plantear nuevas técnicas de compilación [27].

Además del algoritmo, varios estudios han demostrado que la manera de programar impacta directamente en el consumo de energía del dispositivo, otros estudios detectaron que el nivel de optimización utilizado para generar el código máquina tiene repercusiones en el uso eficiente de la energía [28] [29] [30].

Los niveles de optimización del compilador pueden reducir de manera importante el tamaño de código en lenguaje de alto nivel a transformar en código máquina y por ende, implementarlo en dispositivos de menores recursos. Sin embargo, este resultado produce código diseñado para una ejecución lo más rápida posible pero aumentando el consumo de energía [30].

El no utilizar optimizaciones para la generación de código máquina en los diferentes programas de compilación podría ser una manera para no incrementar el consumo de energía, pero el resultado puede ser hasta un 250% más instrucciones [30]. Se considera que esto no necesariamente garantiza un consumo de energía menor, por lo que se proponen herramientas de medición del consumo independiente de compiladores y estimaciones de fabricantes.

En la medida que la frecuencia aumenta, también lo hace el consumo de energía, pero contrario a lo que se podría pensar si se utiliza una frecuencia baja, esta provoca un consumo mayor de energía en algunos casos [20] [30]. El tiempo que se invierte en algoritmos complejos sería mayor, por lo que la energía acumulada tendería a elevarse. La tendencia es utilizar algoritmos eficientes durante periodos cortos al máximo de la capacidad que proporciona el dispositivo y regresar al estado de mínimo consumo de energía [20] [30].

Investigaciones han demostrado que además de las optimizaciones a la compilación de software, se debe cuidar el número de variables, la extensión de las mismas y las ocasiones

en las que se accede a ellas. Además, la estructura del dispositivo y el acceso a la memoria aportan significativamente al consumo de energía [27] [28] [29] [30].

## 2.2 Medición de energía

La medición de energía no es una actividad que solo requiera la medición de dicho parámetro. Para obtener la energía, es necesario medir la potencia que consume el dispositivo bajo prueba durante el periodo de interés. Al obtener dicha información, es necesario integrarla. En la mayoría de las ocasiones para obtener la energía, se mide la corriente promedio del dispositivo en conjunto con el voltaje para obtener la potencia y de esta manera tener acceso a la energía promedio (integrando la potencia). La corriente promedio de consumo del dispositivo se define como se muestra en la ecuación 2:

$$I_{promedio} = \frac{1}{T} \int_0^{T1} I_{(estática)} dt + \frac{1}{T} \int_{T1}^T I_{(dinámica)} dt \quad \text{Ecuación (2)}$$

Es la suma de productos del tiempo en el que se encuentra ejecutando alguna tarea específica y en modo de bajo consumo [25]. La energía consumida por el dispositivo es la integral de la potencia instantánea en un periodo de la señal.

Si la corriente se mide de manera instantánea y el voltaje también, la potencia será instantánea y la energía también. En este trabajo se realiza una medición de la corriente instantánea, por lo que la energía será también instantánea.

Al ser instantánea la medición permite generar un perfil de energía en armonía con los ciclos de reloj de ejecución del microcontrolador, proporcionando información sobre cuáles son las áreas de oportunidad para una mejor estructuración en software para el consumo de energía.

Para la medición de la corriente existen diferentes técnicas. Se presentan algunas de las más utilizadas.

### 2.2.1 Bobina de Rogowski

La bobina de Rogowski es utilizada para medir la corriente alterna que consume un dispositivo bajo prueba. Es una construcción de una bobina en forma circular que tiene sus extremos conectados a un integrador. Tiene la ventaja de no ser invasiva al lazo de corriente que debe establecerse con el dispositivo bajo prueba (como se mostrará con las otras técnicas). La medición de corriente y posible implementación de medición de energía se ven limitadas al ser solamente de corriente alterna y procurar corrientes significativamente mayores en comparación con los dispositivos de bajo consumo de energía (un orden de magnitud mayor) [31].

### 2.2.2 Espejo de corriente

El espejo de corriente es una técnica que permite igualar la corriente que consume una rama en un circuito con otra. Una de las ventajas de esta técnica es la impedancia de salida elevada que permite mantener la corriente independientemente de la carga [32]. En la figura 2 se observa una aplicación muy utilizada de espejo de corriente en [33] [34] [35] y [36]. Utilizan un espejo de corriente Wilson completo [32] para duplicar la corriente del dispositivo bajo prueba a medir. Para medir la corriente utilizan otras técnicas (de carga en capacitor y *shunt*) que se discutirán en los siguientes apartados.

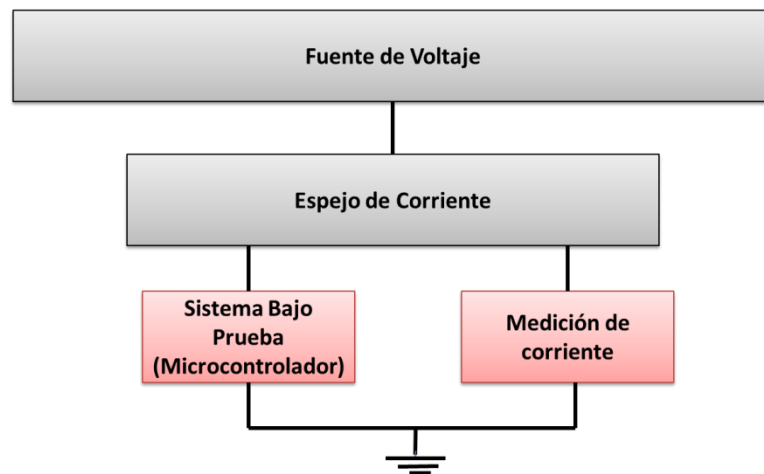


Fig. 2 Medición con espejo de corriente

El agregar cuatro transistores para realizar el espejo completo de Wilson complica la medición al realizar un instrumento de mayor complejidad. No solo eso, la medición de corriente utilizando *shunt* depende de la precisión de la carga utilizada para medir (como se explicará en la sección correspondiente).

### 2.2.3 Carga en capacitor

Otra técnica de medición de energía implica medir la carga almacenada en un capacitor y por ende, la corriente. Cuando un dispositivo opera en modo de bajo consumo de energía (microcontrolador), se comporta como una fuente constante de corriente. Esta técnica emplea la ecuación 3 de corriente en un capacitor para medir la descarga del capacitor [22] utilizando un voltaje definido durante un periodo establecido. Esto permite realizar una conversión para la corriente consumida por el microcontrolador.

$$I = C \frac{dv}{dt} \quad \text{Ecuación (3)}$$

Donde  $I$  es la corriente que consume el dispositivo en amperes,  $V$  es el voltaje generado en volts, tiempo  $t$  en segundos y  $C$  es la carga almacenada en el capacitor en Faradios. Esta técnica también es utilizada en [34] y [36], donde ahora se procura medir el tiempo que tarda la descarga para calcular directamente la carga del capacitor. Al utilizar espejos de corriente, la variable de interés ahora es la carga.

Esta técnica es muy precisa, ya que permite definir la carga que está utilizando el microcontrolador. Sin embargo, es necesario garantizar una correcta medición de la corriente, el voltaje y el tiempo, además de tener un capacitor con una corriente de fuga pequeña.

### 2.2.4 Efecto Hall

Es posible también medir la corriente utilizando el fenómeno conocido como efecto Hall. Dicho comportamiento se presenta cuando a través de un conductor se presenta una corriente, esta genera un campo magnético perpendicular al sentido de la misma y produce



una caída de tensión proporcional en magnitud y sentido a la corriente. La ecuación 4 presenta las expresiones necesarias para la derivación de la corriente a través del efecto Hall [37]:

$$V_{Hall} = \frac{IB}{qnd} \quad \text{Ecuación (4)}$$

Donde  $V_{HALL}$  es el voltaje en volts del efecto Hall,  $I$  es la corriente que fluye a través de un conductor,  $B$  es el campo magnético,  $q$  es la carga del electrón,  $n$  es la densidad de cargas en movimiento, y  $d$  es el diámetro del conductor por donde circula la corriente.

Esta técnica de medición puede ser utilizada para medir corrientes pequeñas como se sugiere en [38] donde es posible medir microamperes, sin embargo, el bajo consumo de energía se encuentra un orden de magnitud menor (en ocasiones dos) que eso. Existen fabricantes de dispositivos que integran esta tecnología como *Allegro* [39] y *Melexis* [40], aunque para la medición de energía en cuestión, no son adecuados debido a su pobre resolución (medición en décimas de Ampere).

### 2.2.5 Resistencia *shunt*

La técnica más utilizada por ser la más directa y sencilla es la conocida como *Shunt*. Consta de colocar una resistencia en el paso de la corriente a medir y realizar una medición de voltaje en ella. Es básicamente un convertidor de voltaje a corriente. Esta medición es proporcional respecto a la ecuación de corriente de Ohm, como se muestra en la ecuación 5:

$$I = \frac{V}{R} \quad \text{Ecuación (5)}$$

Donde  $I$  es la corriente en Amperes presente en la resistencia  $R$  en Ohms debido a la diferencia de potencial  $V$  en Volts. Cabe señalar que debe medirse voltaje diferencial y dicho voltaje puede ser medido por amplificadores operacionales.

Las principales ventajas en utilizar esta técnica es su bajo costo (al utilizar una resistencia), alta precisión y la posibilidad de medir corrientes pequeñas (micro, nano e inclusive pico Amperes). Las principales desventajas de esta técnica es la de agregar resistencia al lazo de polarización, debido a que es una medición directa y la disipación de energía debido a la potencia generada en la resistencia con la que probablemente se modifique el comportamiento del dispositivo si la caída de tensión es significativa en el *shunt* [41].

Aunque la medición de energía a través de resistencias es directa, existen dos técnicas a utilizar con *shunt*, la medición de lado alto y la medición de lado bajo.

### 2.2.5.1 Medición de lado alto

La medición de lado alto para un *shunt* obedece a la característica de la resistencia de estar entre la fuente de alimentación y el dispositivo de medición como se muestra en la figura 3. Una de las ventajas de esta medición es que el dispositivo bajo prueba está conectado directamente al punto en común (tierra) sin embargo, al realizar una medición donde directamente se mide la polarización del dispositivo, es necesario que el dispositivo utilizado para medir dicho voltaje tenga la capacidad de medir los probables valores elevados de polarización. Una muestra de esta implementación se observa en [42] donde se realizan modelados para la medición del consumo de energía por instrucciones.

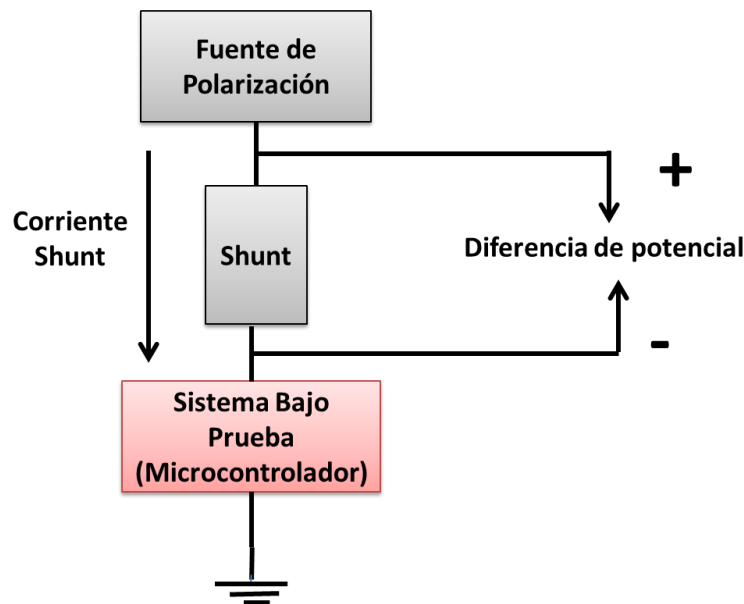


Fig. 3 Medición de corriente por lado alto

### 2.2.5.2 Medición de lado bajo

La medición de lado bajo sugiere una resistencia entre el sistema bajo prueba y tierra o punto en común como se muestra en la figura 4. Una de las ventajas de esta medición es que la referencia tanto del instrumento de medición como del *shunt* están conectados directamente al punto en común (tierra) sin embargo, el voltaje diferencial medido en el dispositivo bajo prueba se encuentra referenciado respecto al punto más positivo del *shunt*, y no al punto común (está flotado). Una representación de esta medición se observa en [43], donde se mide la energía consumida por aplicaciones biomédicas.

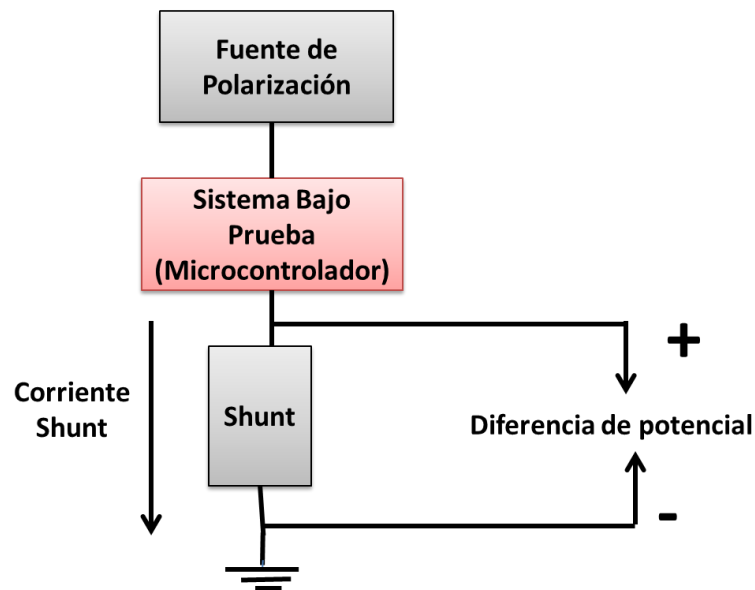


Fig. 4 Medición de corriente por lado bajo

### 2.2.6 Soluciones comerciales

Estas no son propiamente técnicas de medición de corriente, son soluciones comerciales para dicha medición. Tektronix cuenta con una punta de prueba (TCP312) para osciloscopio capaz de realizar mediciones del orden de miliamperes [44]. Utiliza el principio de la bobina de Rogowski y efecto Hall. Su elevado costo y el solo poder acceder a miliamperes lo hace impráctico para la medición de bajo consumo de energía (uno o dos órdenes de magnitud menor son necesarios).

EEMBC cuenta con un sistema de inyección de carga para el monitoreo del consumo de energía denominado EnergyBench [45]. Plantean una modulación de energía que se entrega a un sistema bajo prueba. Existen requerimientos para su utilización, por ejemplo debe tener un módulo *RTC* (un reloj y calendario con su propio oscilador) el dispositivo bajo prueba; además solo puede aplicarse a dispositivos de 3.3 volts.

Existe otro producto que utiliza la medición con resistencia *shunt* y es el *μCurrent Probe* de CMicrotek [46].

## Capítulo 3

### Metodología

En este capítulo se plantea la metodología utilizada para la medición de la energía consumida por el microcontrolador. Se establecen tres secciones: la primera señala lo referente al algoritmo. La segunda establece el equipo a utilizar y su configuración. La tercera etapa se refiere al postprocesamiento.

#### 3.1 Algoritmo

Se plantea un derivado de las pruebas sugeridas por Texas Instruments [47]. Se realizan operaciones aritméticas de adición, sustracción, multiplicación y división. Dichas operaciones se ejecutan en cuatro programas que realizan la misma operación aritmética utilizando diferente extensión de variable. El lenguaje utilizado para la implementación de las pruebas es C. En la figura 5 se muestra un diagrama de flujo de la prueba modificada. El algoritmo en primera instancia introduce el código C0 o el código que genera *main()* a través del compilador. Al terminar la inicialización con C0, se plantea un código de configuración para generar todas las terminales disponibles como salidas digitales y presentar un estado lógico definido (en este caso '0'). Posterior se envía un pulso de sincronía que permite disparar la adquisición de la señal utilizando el osciloscopio. Al terminar el pulso de sincronía, se inicializan las variables a utilizar por las funciones

aritméticas. Dichas funciones aritméticas se ejecutan después de inicializar las variables y se realiza primero la suma, resta, multiplicación y división. Al terminar las operaciones aritméticas se configura el *WDT* para despertar el microcontrolador. Cuando termina la configuración del *WDT* el microcontrolador se activa para bajo consumo de energía (se duerme). El *WDT* lo despierta y reinicia el proceso.

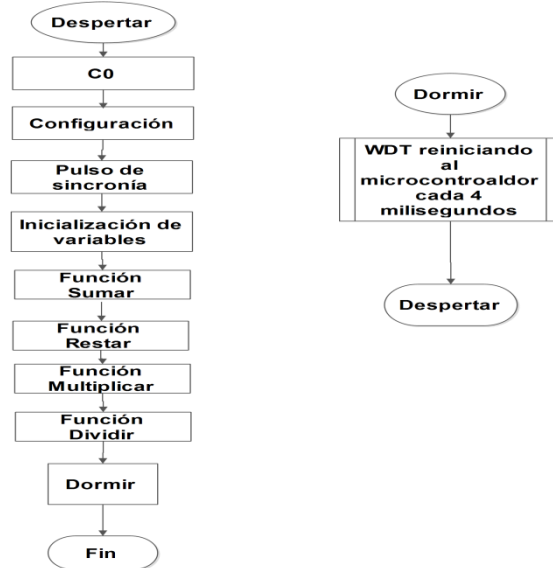


Fig. 5 Diagrama de flujo del algoritmo

### 3.1.1 Código del algoritmo y bits de configuración

En las figuras 6 y 7 se muestra el código utilizado para la prueba. Los programas utilizan variables enteras sin signo. El programa de la figura 6a ejecuta las operaciones con una extensión de variable de 8 bits. El programa de la figura 6b realiza las mismas operaciones aritméticas con variables de 16 bits. El programa de la figura 7a realiza las operaciones con variables de 32 bits y por último, el programa de la figura 7b hace lo propio con variables de 64 bits.

A las variables enteras sin signo se les asigna el valor máximo permitido para no tener sobreflujo y forzar al compilador a realizar el máximo de operaciones en lenguaje ensamblador.

Se utilizan funciones específicas del compilador para configurar salidas digitales en las terminales del microcontrolador. Esto podría ser un inconveniente si el algoritmo se plantea para otro compilador o compilador, sin embargo, la configuración de los registros de dirección de datos para los diferentes microcontroladores de otra compañía o en la misma es relativamente sencillo de implementar. El pulso de sincronía también es generado con funciones propias del compilador. El algoritmo ejecuta operaciones aritméticas después de la configuración inicial. Se sugiere definir estados lógicos para no realizar modificaciones a las terminales sin utilizar y de esta manera no consumir energía innecesaria [24].

El algoritmo planteado no cuenta con un lazo infinito. Para medir la energía estática [21] [22] se utiliza el reinicio a través del WDT (*Watch Dog Timer*) en software. El hardware para el WDT se activa después de la última instrucción que introduce el compilador en lenguaje ensamblador (*sleep*). Después de esta instrucción, el microcontrolador se configura en modo de bajo consumo de energía y espera los 4 milisegundos al cual el WDT se configura.

<pre> #include "P1_config_8.h" #include "P1_un_int_8_fun.h"  //C0 void main(void) {     //Configuración     output_a(0x0000);     output_b(0x0000);      //Pulso de sincronía     output_high(SYNC);     delay_cycles(5);     output_low(SYNC);      //Iniciación de variables     a = 255;     b = 255;      //Operaciones aritméticas     suma = fsuma(a, b);     resta = fresta(a, b);     multiplicacion = fmultiplicacion(a, b);     division = fdivision(a, b);      //WDT (Perro Guardián)     // Consumo bajo de energía     setup_wdt(WDT_4MS); } </pre>	<pre> #include "P2_config_16.h" #include "P2_un_int_16_fun.h"  //C0 void main(void) {     //Configuración     output_a(0x0000);     output_b(0x0000);      //Pulso de sincronía     output_high(SYNC);     delay_cycles(5);     output_low(SYNC);      //Iniciación de variables     a = 65535;     b = 65535;      //Operaciones aritméticas     suma = fsuma(a,b);     resta = fresta(a,b);     multiplicacion = fmultiplicacion(a,b);     division = fdivision(a,b);      //WDT (Perro Guardián)     // Consumo bajo de energía     setup_wdt(WDT_4MS); } </pre>
---	---

Fig. 6 Código del algoritmo de a) 8 bit b) 16 bits

<pre> #include "P3_config_32.h" #include "P3_un_int_32_fun.h"  //C0 void main(void) {     //Configuración     output_a(0x0000);     output_b(0x0000);      //Pulso de sincronía     output_high(SYNC);     delay_cycles(5);     output_low(SYNC);      //Inicialización de variables     a = 4294967295;     b = 4294967295;      //Operaciones aritméticas     suma = fsuma(a,b);     resta = fresta(a,b);     multiplicacion = fmultiplicacion(a,b);     division = fddivision(a,b);      //WDT (Perro Guardián)     // Consumo bajo de energía     setup_wdt(WDT_4MS); } </pre>	<pre> #include "P4_config_64.h" #include "P4_un_int_64_fun.h"  //C0 void main(void) {     //Configuración     output_a(0x0000);     output_b(0x0000);      //Pulso de sincronía     output_high(SYNC);     delay_cycles(5);     output_low(SYNC);      //Inicialización de variables     a = 18446744073709551615;     b = 18446744073709551615;      //Operaciones aritméticas     suma = fsuma(a,b);     resta = fresta(a,b);     multiplicacion = fmultiplicacion(a,b);     division = fddivision(a,b);      //WDT (Perro Guardián)     //Consumo bajo de energía     setup_wdt(WDT_4MS); } </pre>
--	---

Fig. 7 Código del algoritmo de a) 32 bits b) 64 bits

En las figuras 8 y 9 se presenta el código para las funciones utilizadas por el algoritmo. La figura 8a presenta el código para el programa utilizando variables de 8 bits. La figura 8b presenta el código para el programa utilizando variables de 16 bits. La figura 9a presenta el código para el programa utilizando variables de 32 bits. La figura 9b presenta el código para el programa utilizando variables de 64 bits.

Las variables se definen como *volatile* para forzar al compilador a no realizar optimizaciones sobre ellas. Se utiliza un nuevo tipo de variable conocida como *uintx\_t* donde *x* es la extensión de la variable en bits. Esto se realiza para tener uniformidad cuando el algoritmo sea transferido a otra arquitectura de microcontrolador.



```

#ifndef P1_un_int_8_fun_H
#define P1_un_int_8_fun_H

typedef unsigned short uint8_t;

volatile uint8_t suma, resta,
multiplicacion, division,
a, b;

uint8_t fsuma(uint8_t x, uint8_t y)
{
    return (x+y);
}

uint8_t fresta(uint8_t x, uint8_t y)
{
    return (x-y);
}

uint8_t fmultiplicacion(uint8_t x, uint8_t y)
{
    return (x*y);
}

uint8_t fddivision(uint8_t x, uint8_t y)
{
    return (x/y);
}

#endif

```

```

#ifndef P2_un_int_16_fun_H
#define P2_un_int_16_fun_H

typedef unsigned int uint16_t;

volatile uint16_t suma, resta,
multiplicacion, division,
a, b;

uint16_t fsuma(uint16_t x, uint16_t y)
{
    return (x+y);
}

uint16_t fresta(uint16_t x, uint16_t y)
{
    return (x-y);
}

uint16_t fmultiplicacion(uint16_t x, uint16_t y)
{
    return (x*y);
}

uint16_t fddivision(uint16_t x, uint16_t y)
{
    return (x/y);
}

#endif

```

Fig. 8 Impementación de funciones de a) 8 bit b) 16 bits

```

#ifndef P3_un_int_32_fun_H
#define P3_un_int_32_fun_H

typedef unsigned long uint32_t;

volatile uint32_t suma, resta,
multiplicacion, division,
a, b;

uint32_t fsuma(uint32_t x, uint32_t y)
{
    return (x+y);
}

uint32_t fresta(uint32_t x, uint32_t y)
{
    return (x-y);
}

uint32_t fmultiplicacion(uint32_t x, uint32_t y)
{
    return (x*y);
}

uint32_t fddivision(uint32_t x, uint32_t y)
{
    return (x/y);
}

#endif

```

```

#ifndef P4_un_int_64_fun_H
#define P4_un_int_64_fun_H

typedef unsigned long long uint64_t;

volatile uint64_t suma, resta,
multiplicacion, division,
a, b;

uint64_t fsuma(uint64_t x, uint64_t y)
{
    return (x+y);
}

uint64_t fresta(uint64_t x, uint64_t y)
{
    return (x-y);
}

uint64_t fmultiplicacion(uint64_t x, uint64_t y)
{
    return (x*y);
}

uint64_t fddivision(uint64_t x, uint64_t y)
{
    return (x/y);
}

#endif

```

Fig. 9 F Impementación de funciones de a) 32 bits b) 64 bits

Cada programa se ejecuta a cinco diferentes frecuencias. Dichas frecuencias son 500 KHz, 1 MHz, 2 MHz, 4 MHz y 8 MHz. La especificación de las frecuencias se realiza en la configuración inicial para cada programa con la función prototipo *#use delay (internal = frecuencia oscilador)*. Esta función es propia del compilador, aunque también es posible configurar directamente el oscilador interno escribiendo la palabra necesaria a los registros

indicados. Cada ocasión en que se programa el microcontrolador es cuando se cambia de frecuencia, es decir, se repite el programa a la misma frecuencia cada vez que se reinicia debido al *WDT*.

Los bits de configuración es otra etapa no directamente del algoritmo pero si de la programación. Con ellos se definen acciones posibles a tomar del microcontrolador al ser programado. En los bits de configuración se define cuáles serán las terminales para programación, el oscilador interno, y uno de los que compete al algoritmo, la modificación del *WDT* para configurarlo en software (también puede ser configurado en hardware). En la figura 10 se presentan los bits de configuración utilizados para la medición.

```

#ifdef P1_config_8_H
#define P1_config_8_H

#include <24FV32KA302.h>

#define SYNC    PIN_B5

//Programación a través de PGD1 y PGC1
#FUSES ICSP1

//Sin protección
#FUSES NOWRT

//Sin protección
#FUSES NOPROTECT

//Oscilador Interno 8MHz con PLL -->32MHz (Fcyc= 16MHz)
//#FUSES FRC_PLL

//Oscilador Interno 8MHz con Divisor
#FUSES FRC_DIV

//Oscilador Interno 8MHz sin PLL -->8MHz (Fcyc= 4MHz)
//#FUSES FRC

//Oscilador Interno 31KHz de bajo consumo de energía
//#FUSES LPRC

//Esperar la estabilización de la fuente
#FUSES PUT
//MCLR activado (para prevenir futuros bloqueos)
#FUSES MCLR

//Sin oscilador Externo Primario (solo Interno)
#FUSES NOPR

//Habilitar WDT en software
#FUSES WDT_SW

//Con Perro Guardian
//#FUSES WDT

//Fosc para perro guardián (poscalador 1:32) 1mS
//#FUSES WDT32

//Poscalador para perro guardián (1:4) 8mS
//#FUSES WPOSTS4

//Sin protección
#FUSES NOWRTB

//Sin protección
#FUSES NOBSS

//El oscilador secundario para usarse como terminales digitales o reloj externo
#FUSES SOSC_DIGITAL

//Precaución de cambio de oscilador Interno-Externo desactivado
#FUSES NOIESO

//Terminal del Oscilador2 es digital
#FUSES OSCIO
//Cambio de oscilador desactivado; el monitoreo de ese error está desactivado
#FUSES NOCKSFSM

//No reinicio por bajo voltaje
#FUSES NOBROWNOUT

//No reinicio por bajo voltaje en Deep Sleep
#FUSES NODSBOR

//Perro Guardian en Deep Sleep desactivado
#FUSES NODSWDT

//Configuración de Fcyc a 500KHz (Fosc = 250KHz)
#use delay(internal=500K)

#endif

```

Fig. 10 Bits de configuración

### 3.1.2 Pulso de sincronía y ciclos de reloj

El microcontrolador evaluado configura de forma predeterminada sus terminales como entradas después de cada reinicio sin importar si fue debido a una baja de voltaje de polarización, cuando recién es polarizado o en este caso, cuando el *WDT* lo reinicia. Se decide forzar la configuración para salidas y de esta manera tener una base para cuando sea necesario implementar el algoritmo en otro microcontrolador que no presente dicho comportamiento.

Un factor crítico para determinar el consumo de energía a nivel de instrucción en lenguaje ensamblador es la introducción del pulso de sincronía como se observa en la figura 11 con la línea verde. Dicho pulso tiene dos funciones primordiales:

1. Establecer el disparo del osciloscopio para la captura de la señal. De esta manera se podrá promediar la señal y,
2. permitir la medición en ciclos de reloj del lenguaje ensamblador generado por el compilador.

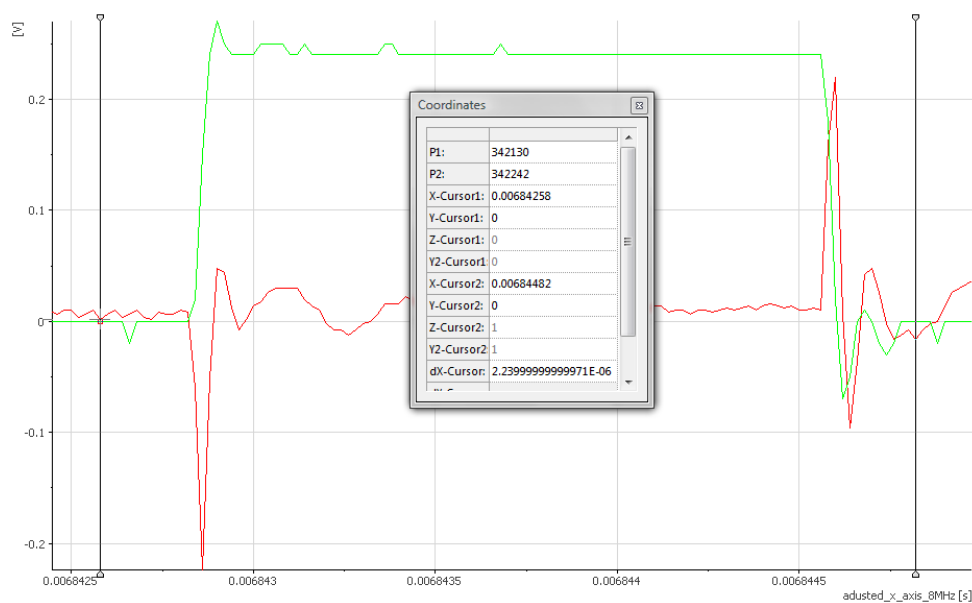


Fig. 11 Pulso de sincronía

El pulso de sincronía permite tener un indicador en la señal de energía adquirida para diferenciar entre el consumo de energía estático y dinámico. Se utilizan los ciclos de reloj

generados por el pulso como base para rastrear dónde inicia el código C0 y poder separar los dos consumos de energía.

El pulso de sincronía establece la base para la medición. Son nueve ciclos de reloj para independientemente de la frecuencia del oscilador. La figura 12 muestra un programa ejecutando el código con tiempo de instrucción para el microcontrolador de 500 nanosegundos. La línea azul establece el pulso, la roja el consumo de voltaje, la verde el consumo de corriente y las dos líneas verticales a los costados del impulso son el inicio y fin de la medición.

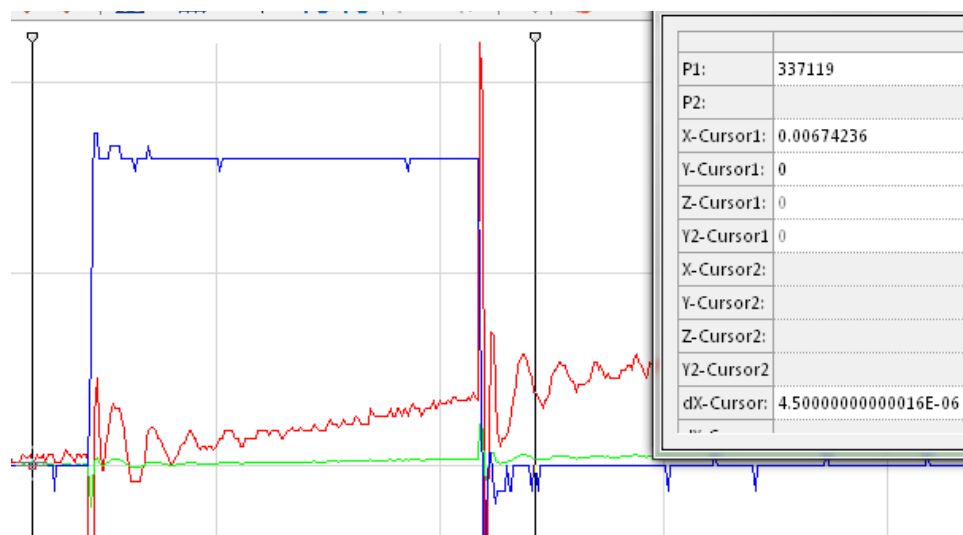


Fig. 12 Medición del pulso de sincronía

La mayoría de los ambientes de programación proporcionan herramientas para observar el código ensamblador del lenguaje de alto nivel, es decir, el código o listado desensamblado. Además de dicha herramienta también es posible simular el código y de esta manera determinar cuántos ciclos de reloj se generan para cada instrucción de alto nivel.

MPLAB X proporciona estas herramientas y se les conoce como *dissassembly listing* y *stopwatch*. Utilizándolas fue posible establecer el número de ciclos de reloj que utiliza el algoritmo en cada sección. Se utilizaron puntos de quiebre en el código ensamblador generado por el compilador en cada sección del algoritmo.

Se muestra el código generado por el compilador para las operaciones de 64 bits utilizando una frecuencia de oscilador de 500 KHz. Estas requieren el mayor procesamiento debido a

que el microcontrolador es de 16 bits y cuentan con llamados a subrutinas dentro de las funciones. La metodología para adquirir los ciclos de reloj para el algoritmo a 8, 16 y 32 bits es la misma. Es posible utilizar emulación pero no es recomendable ya que la información entre el programador o sistema de emulación y el microcontrolador producirá un consumo de energía no deseado. Existen herramientas específicas de algunas compañías en las que utilizando dichos sistemas son capaces de medir la energía de sus dispositivos mientras es emulado el código [19] [18]. Sin embargo es específico para algunos microcontroladores de dichas compañías y no es posible utilizar dicho equipo en otros dispositivos diferentes a la compañía.

La figura 13 muestra el código *C0* generado por el compilador. Es importante señalar que este código, independiente del compilador, siempre es incluido. El compilador en este caso está configurando el oscilador interno para una frecuencia de 500 KHz y forzando terminales digitales en el microcontrolador. Cabe señalar que este código varía dependiendo del dispositivo, aunque sea el mismo compilador, al tener diferente arquitectura el microcontrolador, será diferente.

```

0x0: GOTO 0x40380
! * File: P4_uint64.c
! * Author: Mario Camarero
! *
! * Created on 26 de dic
! */
#include "P4_config_64.
#include "P4_un_int_64_
void main(void)
0x380: MOV #0xF80, W15
0x382: MOV #0xFFF, W0
0x384: MOV WREG, SPLIM
0x386: NOP
0x388: BSET 0x81, #7
0x38A: MOV #0x300, W4
0x38C: MOV W4, CLKDIV
0x38E: CLR ANSA
0x390: CLR ANSB
Stopwatch cleared. Stopwatch cycle count = 0 (0 ns)
Target halted. Stopwatch cycle count = 11 (44 µs)

```

Fig. 13 Ensamblador y ciclos de reloj.- C0

La figura 14 representa la configuración inicial para que las terminales del microcontrolador sean salidas digitales y generen un cero lógico.

```
!      output_a(0x0000);
0x392: CLR TRISA
0x394: CLR LATA
!      output_b(0x0000);
0x396: CLR TRISB
0x398: CLR LATB
|Target halted. Stopwatch cycle count = 4 (16 µs)
```

Fig. 14 Ensamblador y ciclos de reloj.- Configuración

La figura 15 presenta el lenguaje ensamblador que genera el compilador para establecer el pulso de sincronía. Se envía un uno lógico y se esperan cinco ciclos de reloj; al terminar se envía un cero lógico. Este código no está optimizado, ya que anteriormente se realizó una configuración para salidas y no es necesario que se configure de nuevo la terminal utilizada (en este caso el bit 5 del puerto B) como salida. La intención es que el algoritmo se aplique en cualquier compilador y realice lo necesario para generar dicho pulso.

```
!      output_high(SINCRONIA);
0x39A: BCLR TRISB, #5
0x39C: BSET LATB, #5
!      delay_cycles(5);
0x39E: REPEAT #0x3
0x3A0: NOP
!      output_low(SINCRONIA);
0x3A2: BCLR TRISB, #5
0x3A4: BCLR LATB, #5
|Target halted. Stopwatch cycle count = 9 (36 µs)
```

Fig. 15 Ensamblador y ciclos de reloj.- Pulso de sincronía

En la figura 16 se muestra la inicialización de las variables a utilizar con el valor máximo permitido para un dato entero sin signo. El compilador utilizará lo necesario para ajustar el tipo de dato a la arquitectura utilizada.

```
!      a = 18446744073709551615;
0x3A6: SETM a
0x3A8: SETM 0x822
0x3AA: SETM 0x824
0x3AC: SETM 0x826
!      b = 18446744073709551615;
0x3AE: SETM b
0x3B0: SETM 0x82A
0x3B2: SETM 0x82C
0x3B4: SETM 0x82E

Target halted. Stopwatch cycle count = 8 (32 µs)
```

Fig. 16 Ensamblador y ciclos de reloj.- Inicialización de variables

Al tener las variables con los datos asignados, la primer operación aritmética a realizar es la de sumar. En la figura 17a se observa que existe un llamado a la dirección `0x20200` después de realizar la asignación de las variables globales `a` y `b` en las locales `x` e `y`. La rutina de suma para operaciones de 64 bits muestra en la figura 17b cómo el compilador implementa dicha operación en el microcontrolador de 16 bits.

<pre>!      suma = fsuma(a,b); 0x3B6: PUSH a 0x3B8: POP x 0x3BA: PUSH 0x822 0x3BC: POP 0x832 0x3BE: PUSH 0x824 0x3C0: POP 0x834 0x3C2: PUSH 0x826 0x3C4: POP 0x836 0x3C6: PUSH b 0x3C8: POP y 0x3CA: PUSH 0x82A 0x3CC: POP 0x83A 0x3CE: PUSH 0x82C 0x3D0: POP 0x83C 0x3D2: PUSH 0x82E 0x3D4: POP 0x83E 0x3D6: CALL 0x20200 0x3DA: MOV WREG, suma 0x3DC: MOV W1, 0x802 0x3DE: MOV W2, 0x804 0x3E0: MOV W3, 0x806</pre>	<pre>!uint64_t fsuma(uint64_t x, uint64_t y) !{ !    return(x+y); 0x200: MOV x, WREG 0x202: ADD y, WREG 0x204: MOV 0x83A, W4 0x206: MOV 0x832, W3 0x208: ADDC W3, W4, W1 0x20A: MOV 0x83C, W4 0x20C: MOV 0x834, W3 0x20E: ADDC W3, W4, W2 0x210: MOV 0x83E, W4 0x212: MOV 0x836, W3 0x214: ADDC W3, W4, W3 0x216: RETURN</pre>
---	--

Target halted. Stopwatch cycle count = 44 (176 µs)

Fig. 17 Ensamblador y ciclos de reloj a) Función suma b) Subrutina 64 bit



La segunda operación aritmética es la de sustracción o resta. Como en el caso de la suma, se observa en la figura 18a después de mover los datos de las variables a y b a la x e y, se realiza un llamado a la dirección 0x20218. La rutina de resta para operaciones de 64 bits se muestra en la figura 18b y cómo el compilador implementa dicha operación en el microcontrolador de 16 bits.

<pre> !      resta = fresta(a,b); 0x3E2: PUSH a 0x3E4: POP x 0x3E6: PUSH 0x822 0x3E8: POP 0x832 0x3EA: PUSH 0x824 0x3EC: POP 0x834 0x3EE: PUSH 0x826 0x3F0: POP 0x836 0x3F2: PUSH b 0x3F4: POP y 0x3F6: PUSH 0x82A 0x3F8: POP 0x83A 0x3FA: PUSH 0x82C 0x3FC: POP 0x83C 0x3FE: PUSH 0x82E 0x400: POP 0x83E 0x402: CALL 0x20218 0x406: MOV WREG, resta 0x408: MOV W1, 0x80A 0x40A: MOV W2, 0x80C 0x40C: MOV W3, 0x80E </pre>	<pre> !uint64_t fresta(uint64_t x, uint64_t y) !{ !      return(x-y); 0x218: MOV x, W4 0x21A: MOV y, W3 0x21C: SUB W4, W3, W0 0x21E: MOV 0x832, W4 0x220: MOV 0x83A, W3 0x222: SUBB W4, W3, W1 0x224: MOV 0x834, W4 0x226: MOV 0x83C, W3 0x228: SUBB W4, W3, W2 0x22A: MOV 0x836, W4 0x22C: MOV 0x83E, W3 0x22E: SUBB W4, W3, W3 0x230: RETURN </pre>
--	---

Target halted. Stopwatch cycle count = 45 (180 µs)

Fig. 18 Ensamblador y ciclos de reloj a) Función resta b) Subrutina 64 bit

La operación de multiplicación tiene dos llamados a subrutinas. Aunque existe un módulo interno en el microcontrolador de multiplicación, este es para operaciones enteras de 16 por 16 bits. En la figura 19 se muestra la rutina de multiplicación de cómo el compilador implementa dicha operación en el microcontrolador de 16 bits se muestra en la siguiente figura para operaciones de 64 bits.

<pre> !      multiplicacion = fmultiplicacion(a,b); 0x40E: PUSH a 0x410: POP x 0x412: PUSH 0x822 0x414: POP 0x832 0x416: PUSH 0x824 0x418: POP 0x834 0x41A: PUSH 0x826 0x41C: POP 0x836 0x41E: PUSH b 0x420: POP y 0x422: PUSH 0x82A 0x424: POP 0x83A 0x426: PUSH 0x82C 0x428: POP 0x83C 0x42A: PUSH 0x82E 0x42C: POP 0x83E 0x42E: CALL 0x20288 0x432: MOV WREG, multiplicacion 0x434: MOV W1, 0x812 0x436: MOV W2, 0x814 0x438: MOV W3, 0x816 </pre>	<pre> !uint64_t fmultiplicacion(uint64_t x, uint64_t y) 0x288: MOV W5, [W15++] 0x28A: MOV W6, [W15++] 0x28C: MOV W7, [W15++] !{ !      return (x*y); 0x28E: MOV x, WREG 0x290: MOV 0x832, W1 0x292: MOV 0x834, W2 0x294: MOV 0x836, W3 0x296: MOV y, W4 0x298: MOV 0x83A, W5 0x29A: MOV 0x83C, W6 0x29C: MOV 0x83E, W7 0x29E: CALL 0x20232 0x2A2: MOV [--W15], W7 0x2A4: MOV [--W15], W6 0x2A6: MOV [--W15], W5 0x2A8: RETURN </pre>
---	--

Target halted. Stopwatch cycle count = 102 (408  $\mu$ s)

Fig. 19 Ensamblador y ciclos de reloj a) Función multiplicación b) Subrutina 64 bit

La operación de división tiene dos llamados a subrutinas. Aunque existe un módulo interno en el microcontrolador de división, este es para operaciones enteras de 32 bits entre 16 bits. En la figura 20 se muestra la rutina de multiplicación de cómo el compilador implementa dicha operación en el microcontrolador de 16 bits se muestra en la siguiente figura para operaciones de 64 bits.

<pre> !      division = fdivision(a,b); 0x43A: PUSH a 0x43C: POP x 0x43E: PUSH 0x822 0x440: POP 0x832 0x442: PUSH 0x824 0x444: POP 0x834 0x446: PUSH 0x826 0x448: POP 0x836 0x44A: PUSH b 0x44C: POP y 0x44E: PUSH 0x82A 0x450: POP 0x83A 0x452: PUSH 0x82C 0x454: POP 0x83C 0x456: PUSH 0x82E 0x458: POP 0x83E 0x45A: CALL 0x2035C 0x45E: MOV WREG, division 0x460: MOV W1, 0x81A 0x462: MOV W2, 0x81C 0x464: MOV W3, 0x81E </pre>	<pre> !uint64_t fdivision(uint64_t x, uint64_t y) 0x35C: MOV W5, [W15++] 0x35E: MOV W6, [W15++] 0x360: MOV W7, [W15++] !{ !      return (x/y); 0x362: BCLR 0x43, #0 0x364: MOV x, WREG 0x366: MOV 0x832, W1 0x368: MOV 0x834, W2 0x36A: MOV 0x836, W3 0x36C: MOV y, W4 0x36E: MOV 0x83A, W5 0x370: MOV 0x83C, W6 0x372: MOV 0x83E, W7 0x374: CALL 0x202AA 0x378: MOV [--W15], W7 0x37A: MOV [--W15], W6 0x37C: MOV [--W15], W5 0x37E: RETURN </pre>
---	---

Target halted. Stopwatch cycle count = 129 (516 µs)

Fig. 20 Ensamblador y ciclos de reloj a) Función división b) Subrutina 64 bit

La última parte del código implementado para el algoritmo es la configuración y activación del WDT para que el microcontrolador reinicie cada 4 milisegundos. Se observa en la figura 21 cómo el compilador activa el bajo consumo de energía mediante la instrucción de ahorro de energía.

```

!      setup_wdt(WDT_4MS);
0x288: BSET RCON, #5
0x28A: PWRSAV #0
0x28A: PWRSAV #0

```

Target halted. Stopwatch cycle count = 4 (16 µs)

Fig. 21 Ensamblador y ciclos de reloj.- WDT

### 3.2 Equipo y material de medición

Se optó por realizar la medición en lado bajo [48] [49] como se muestra en la figura 22. Se observa que el dispositivo bajo prueba, en este caso el microcontrolador se encuentra “flotado” respecto al punto más negativo del sistema (tierra).

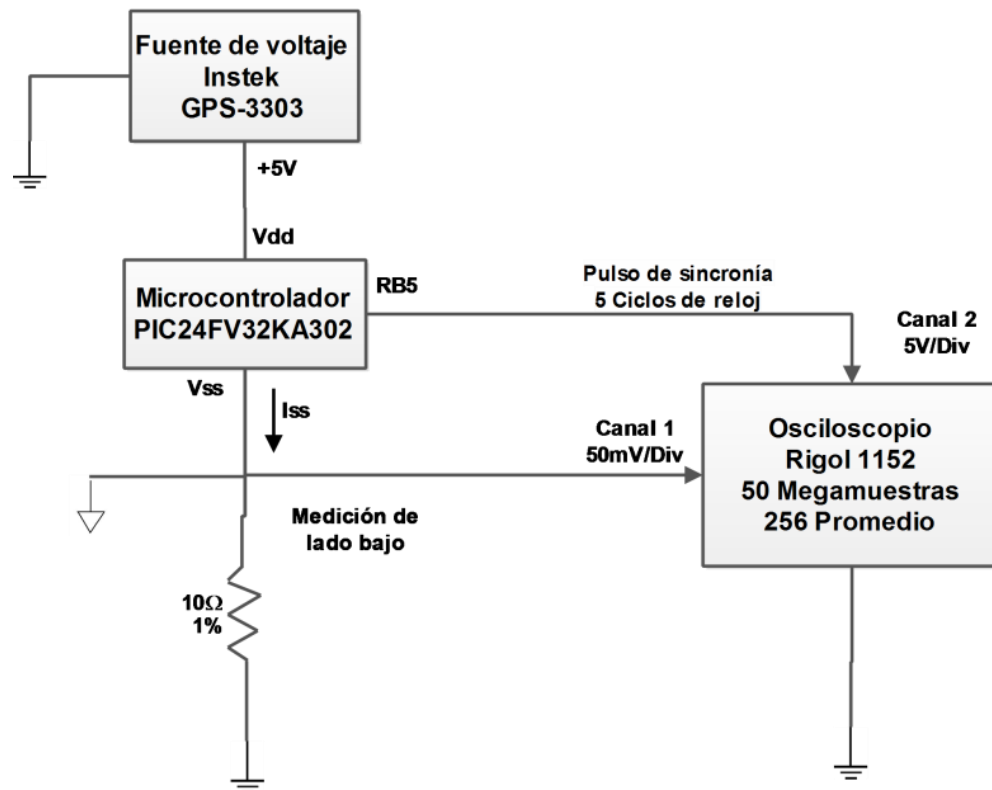


Fig. 22 Diagrama a bloques de la medición

### 3.2.1 El microcontrolador

Se utilizó para implementar el algoritmo un microcontrolador de la marca Microchip, específicamente el PIC24FV32KA302 [50]. Este dispositivo fue diseñado para bajo consumo de energía.

En la figura 23 se muestra el esquemático del circuito. Se colocan capacitores de 100 nanofaradios en las terminales de polarización para compensar los cambios de alta frecuencia.  $V_{CAP}$  es un capacitor de tantalum de 10 microfaradios como lo sugiere la hoja de datos del microcontrolador. Se optó por la terminal RB5 del microcontrolador para enviar la señal de sincronía al osciloscopio. Pudo ser cualquier terminal que se pueda configurar como salida digital. La resistencia *SHUNT* utilizada es de 10 Ohms con 1% de tolerancia [51].

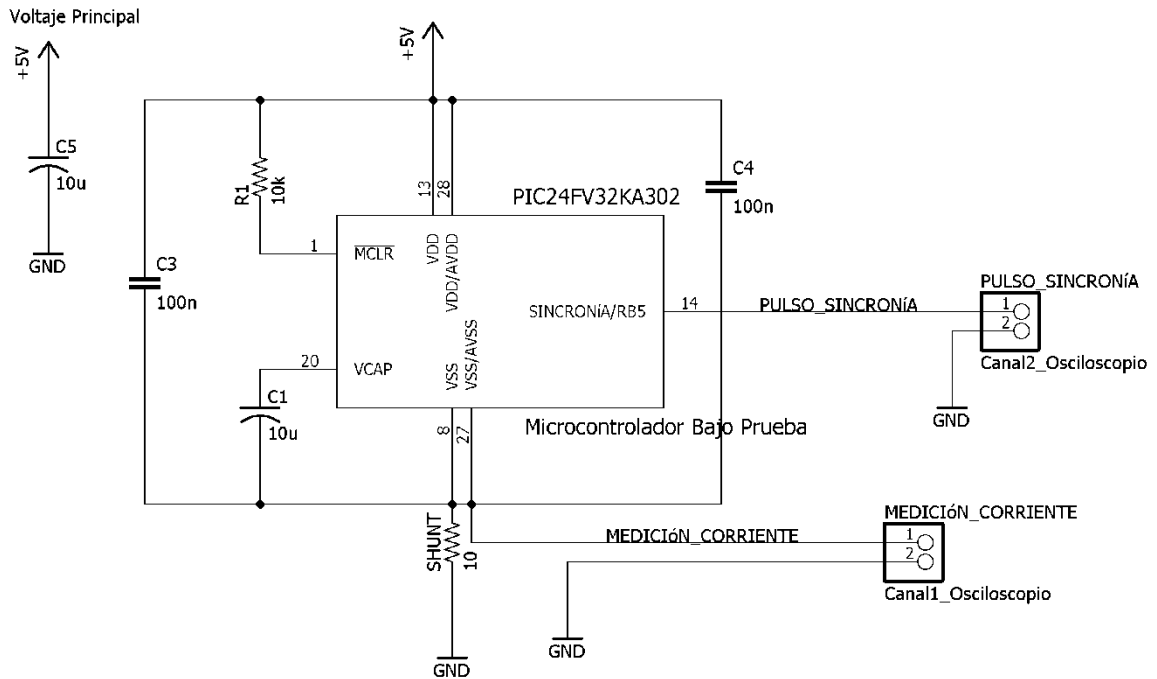


Fig. 23 Esquemático de la medición

### 3.2.2 Osciloscopio

La medición de la energía consumida a través de la corriente demandada por el microcontrolador se realiza monitoreando la caída de tensión generada por la resistencia de precisión en la medición de lado bajo. Dicha medición de voltaje se realiza con un osciloscopio accesible en la mayoría de los laboratorios de instrumentación.

El osciloscopio utilizado es un RIGOL 1102D de 100 MHz a 1 Giga muestras por segundo con una profundidad de memoria de 1 millón de datos [52]. La información se guarda en una memoria USB con formato csv (valores separados por coma).

#### 3.2.2.1 Configuración para medición de voltaje

El medir la caída de tensión con el osciloscopio genera implica conectar la punta de medición a través de la resistencia *shunt* en paralelo. Al ser una resistencia pequeña (10 Ohms@1%) es recomendable realizar el cálculo de la resistencia equivalente generada por el efecto de carga que inducirá la punta en el circuito.

La punta del osciloscopio tiene una impedancia de entrada de 1 Mega Ohm. Al conectarla en paralelo con la resistencia se obtiene un equivalente como se muestra en la ecuación 3:

$$R_{equivalente} = \frac{R_{shunt} \cdot R_{osciloscopio}}{R_{shunt} + R_{osciloscopio}} = \frac{10\Omega \cdot 1M\Omega}{10\Omega + 1M\Omega} = 9.9999\Omega \quad \text{Ecuación (6)}$$

El error de medición debido al efecto de carga se encuentra por debajo de un orden de magnitud (menos del 0.001%), el cual es aceptable para las mediciones a realizar.

Para efectos de comparación entre las diferentes mediciones de voltaje del microcontrolador a varias frecuencias, se optó por tener una configuración estándar. Dicha configuración obedece a la frecuencia menor utilizada ya que esta provocará que sea más lento cada ciclo de reloj y se requiera una mayor cantidad de memoria para capturarlo. Para capturar un periodo de la señal que genera el microcontrolador a la frecuencia menor con el algoritmo 250 KHz, el osciloscopio utiliza una frecuencia de muestreo de 20 nanosegundos

o 50 Mega muestras. Dicha frecuencia de muestreo es suficiente para capturar ciclos de reloj de 250 nanosegundos que será la frecuencia mayor a utilizar (4 MHz). En la Figura 20 se muestra la configuración estandarizada para todas las mediciones. La primera fila representa los canales del osciloscopio y la división por sección. Se configuró 50 milivolts para el canal 1 y 5 milivolts para el canal 2. Se utiliza un acoplamiento de corriente directa. En la segunda fila se configura la impedancia de entrada a un Mega Ohm para tener la resistencia equivalente mostrada en la ecuación 3.

En la tercera fila se refiere al tiempo. Para poder capturar toda la señal y aprovechar la memoria del osciloscopio al máximo, se ajustó a 500 microsegundos con un retardo de 1.5 milisegundos. Este ajuste captura en la pantalla la señal de interés.

En la cuarta fila se presenta el disparo para la captura de la señal en el osciloscopio. Se dispara en flanco positivo con un ajuste de 200 milivolts. Cabe señalar que se estableció la captura utilizando el canal 2 del osciloscopio.

La última fila muestra la configuración y que la adquisición de la señal. Se promedió en 256 ocasiones ya que el osciloscopio tenía esta función disponible. Esto permite tener una señal promediada (filtrada) sin necesidad de colocar filtros analógicos en la adquisición. La captura se realiza a 50 Mega muestras por segundo (20 nanosegundos para la frecuencia de muestreo). El pulso de sincronía permite tener esta configuración para todas las mediciones sin importar la frecuencia de operación del microcontrolador.

Analog ch	State	Scale	Position	Coupling	BW Limit	Invert
CH1	On	50.0mV/	0.00uV	DC	On	off
CH2	On	5.00V/	0.00uV	DC	on	off
Analog ch	Impedance	Probe				
CH1	1M ohm	1X				
CH2	1M ohm	1X				
Time Main	Time Ref Center	Main Scale 500.0us/	Delay -1.500000ms			
Trigger Edge	Source CH2	Slope Rising	Mode Normal	Coupling DC	Level 200mV	Holdoff 500ns
Acquisition Average	Sampling Realtime	Averages 256	Memory Depth Long Memory		Sample Rate 50.00MSa	

Fig. 24 Configuración estándar del osciloscopio

### **3.2.2.2 Pulso de sincronía**

Para comparar las señales generadas por el consumo de energía del microcontrolador se utiliza una señal para disparar la captura del osciloscopio en el canal 2. Al realizar esto la capacidad de memoria se divide a la mitad, es decir, se utilizan 0.5 Mega puntos para el pulso de sincronía y 0.5 Mega puntos para la señal de voltaje. Esta es una característica de la mayoría de los osciloscopios. El pulso de sincronía establece la captura en el mismo punto del algoritmo en cada ocasión que se reinicia. Esto permite utilizar la función del osciloscopio para promediar la señal y de esta manera hacer un promedio de la señal, es decir, ayudar a eliminar el ruido.

### **3.2.3 Fuente de voltaje**

Se utilizó una fuente de voltaje de uso común en la mayoría de los laboratorios de instrumentación o de mediciones eléctricas. Es una INSTEK GPS-3303 de triple salida [53]. Esta fuente proporciona una salida fija de 5 Volts con corriente de hasta 3 Amperes. Esa opción de voltaje fue la utilizada para polarizar el microcontrolador.

## **3.3 Postprocesamiento**

Después de capturar la información del osciloscopio, es necesario procesarla para darle un formato adecuado como se muestra en la Figura 21, y de esta manera realizar el análisis del perfil de energía.

La información se captura en un formato de valores separados por coma (csv) el cual proporciona tres valores: el valor del canal 1 (señal capturada a través de la resistencia *shunt*); el valor del canal 2 (señal de sincronía capturada); y la señal de tiempo.

Esta información no es capturada con unidades de voltaje para las señales de voltaje y tampoco de segundos para tiempo, es por esta razón que se plantea una etapa adicional para



el análisis de los datos. Lo presentado en esta sección se aplica a todos los perfiles de energía adquiridos con las diferentes frecuencias.

### 3.3.1 Perfil de energía completo

Para tener un perfil de energía con mayor información que la señal de voltaje adquirida por el osciloscopio es necesario darle formato a lo capturado. Dicho procedimiento se muestra en el diagrama a bloques de la siguiente figura. Lo que se captura es voltaje, no es energía. Para acceder a la energía se realiza una conversión de la señal adquirida en valores a una señal de voltaje respecto al tiempo. Posterior se convierte dicha señal de voltaje a una de corriente. Al terminar se multiplican las dos señales para obtener potencia. Al final se integra dicha señal para obtener la energía acumulada.

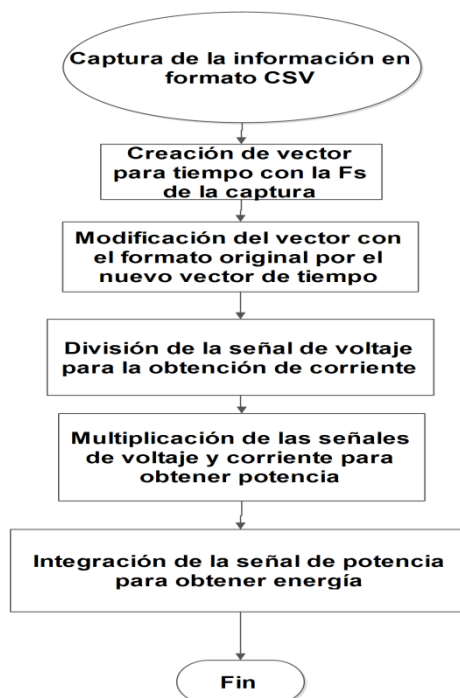


Fig. 25 Diagrama de flujo para el postprocesamiento

Este apartado puede realizarse con múltiples herramientas de software. Se consideraron Matlab, Octave, Excel. Sin embargo se optó por DIADEM de National Instruments. Este software permite realizar modificaciones sobre los datos de forma gráfica y resulta óptimo para lo que se desea analizar.

En primera instancia se muestra la señal de voltaje del algoritmo, en color rojo, capturada por el osciloscopio sin formato utilizando variables de 64 bits y ejecutándose a 8 MHz. Esta señal no cuenta con un índice adecuado en tiempo, solo presenta el número en el cual fue capturado el evento. Para realizar el ajuste de los valores incrementales del formato CSV a un eje de tiempo, se crea un vector con el mismo número de muestras que el original y se multiplica dicho vector por la frecuencia de muestreo a la cual fue adquirida. Este procedimiento se muestra en la figura 26 utilizando la opción de análisis→funciones para canales→generar canal numérico.

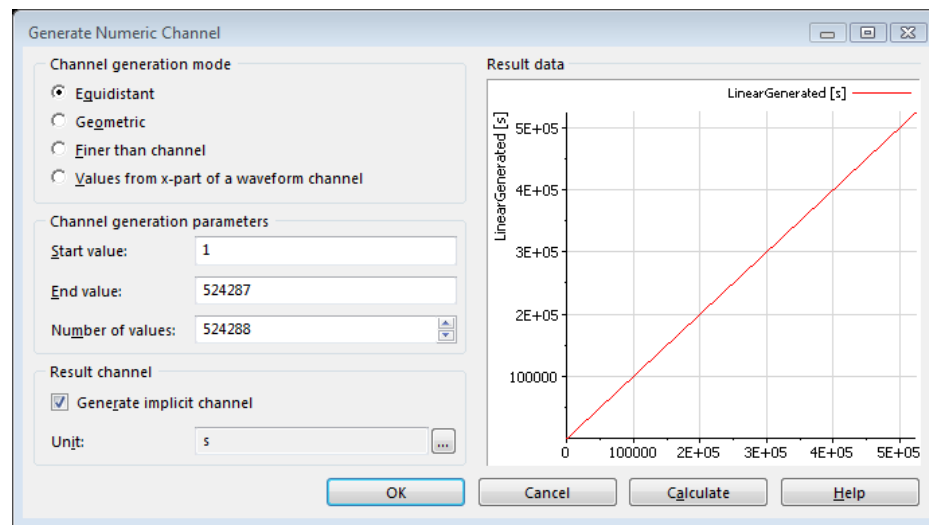


Fig. 26 Creación de vector para ajuste de tiempo

La figura 27 muestra la opción de la calculadora (función propia de DIADEM), donde se realiza el ajuste para el tiempo. Esto se obtiene mediante la multiplicación del vector creado con la frecuencia de muestreo como se muestra en la figura siguiente.

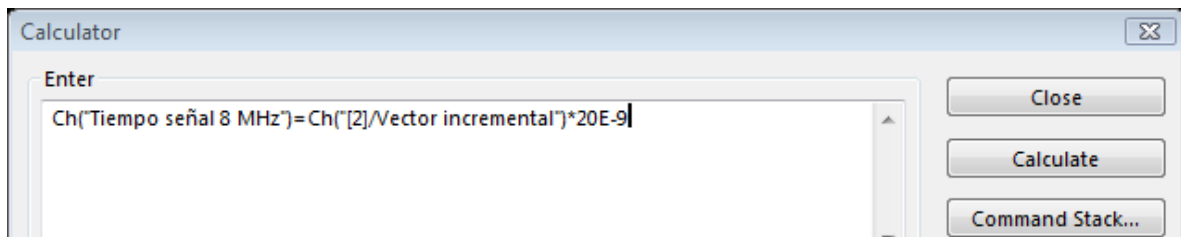


Fig. 27 Creación del vector Tiempo con la frecuencia de muestreo

Posteriormente se crea una nueva señal con la base de tiempo apropiada para la frecuencia de muestreo utilizando la señal de voltaje adecuada para la medición en cuestión. Este procedimiento se muestra en la figura 28 utilizando la opción de análisis→funciones para canales→conversión entre canales numéricos y canales de señales.

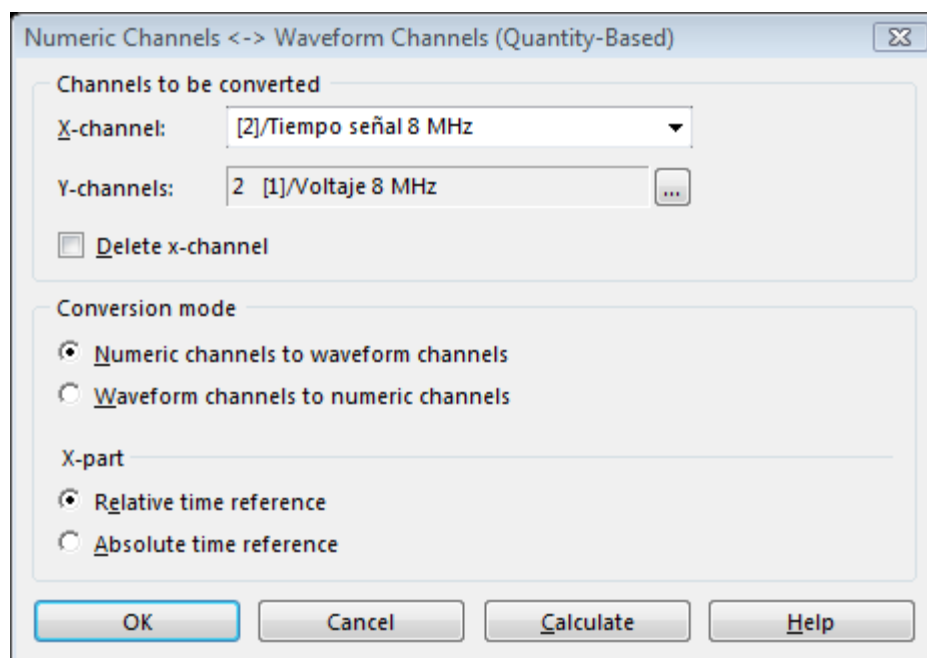


Fig. 28 Creación de la nueva señal con la base de tiempo correcta

Utilizando la misma función de la calculadora, se realiza el cálculo para determinar la corriente encontrada en el perfil de energía. Al utilizar una resistencia de precisión de 10 Ohms, la señal resultante de voltaje de la acción descrita anteriormente es dividida entre el valor de la resistencia (10 Ohms). Las dos señales se muestran en la figura 29. La señal roja es la representación del perfil de energía en voltaje; la señal verde es de corriente.

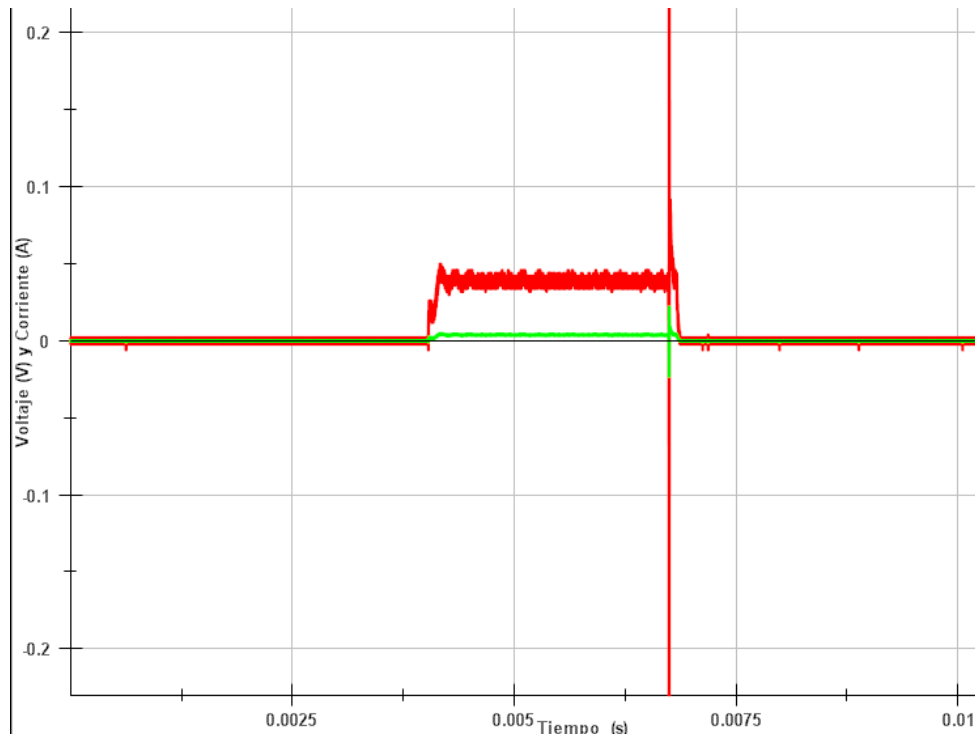


Fig. 29 Perfiles energía de voltaje y corriente superpuestas

Al tener las señales de voltaje y corriente, el siguiente paso es obtener la potencia generada por dichas señales. Esto se realiza multiplicando las dos señales. Es posible obtener la potencia solo utilizando la señal de voltaje y dividiendo entre el cuadrado de la resistencia, sin embargo, se optó por obtener la señal de corriente también. Utilizando la opción análisis → matemáticas básicas → multiplicación, se obtuvo la señal de potencia.

En la figura 30 se observan las dos señales de voltaje y corriente en azul, mientras que la señal resultante de la multiplicación se presenta en color rojo. Una de las ventajas de utilizar la herramienta DIADEM es la de realizar operaciones con unidades. Al multiplicar corriente y voltaje, automáticamente presenta el resultado en watts.

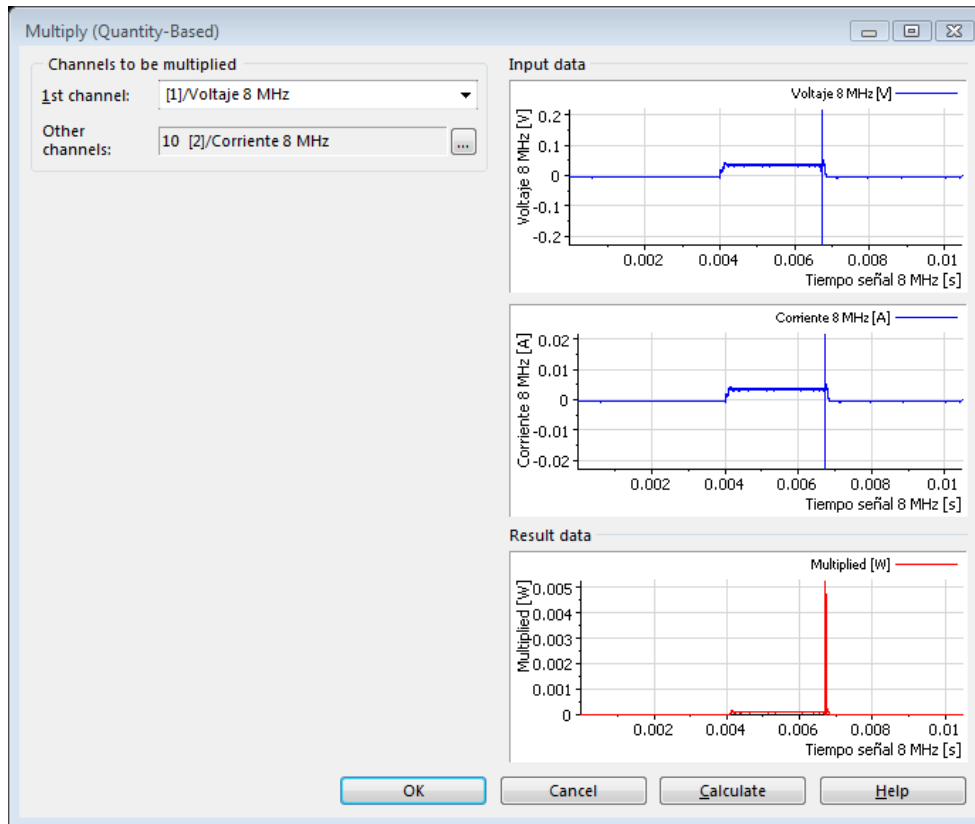


Fig. 30 Cálculo de la señal de potencia

Para obtener la energía consumida por el algoritmo incluyendo la estática y dinámica se integra la señal de potencia calculada. Esto se realiza con la opción análisis→matemáticas básicas→integrar. Cabe señalar que se utilizó la integración trapezoidal para realizar el cálculo. La herramienta puede realizar la integración utilizando la regla de Simpson, sin embargo se decidió utilizar la trapezoidal por ser la predeterminada. En la figura 31 se muestra cómo la herramienta utiliza la señal de potencia previamente calculada y realiza la integración. La señal azul muestra la potencia, mientras que la roja la energía. Se puede observar el incremento gradual de energía (esto se discutirá en el capítulo de resultados).

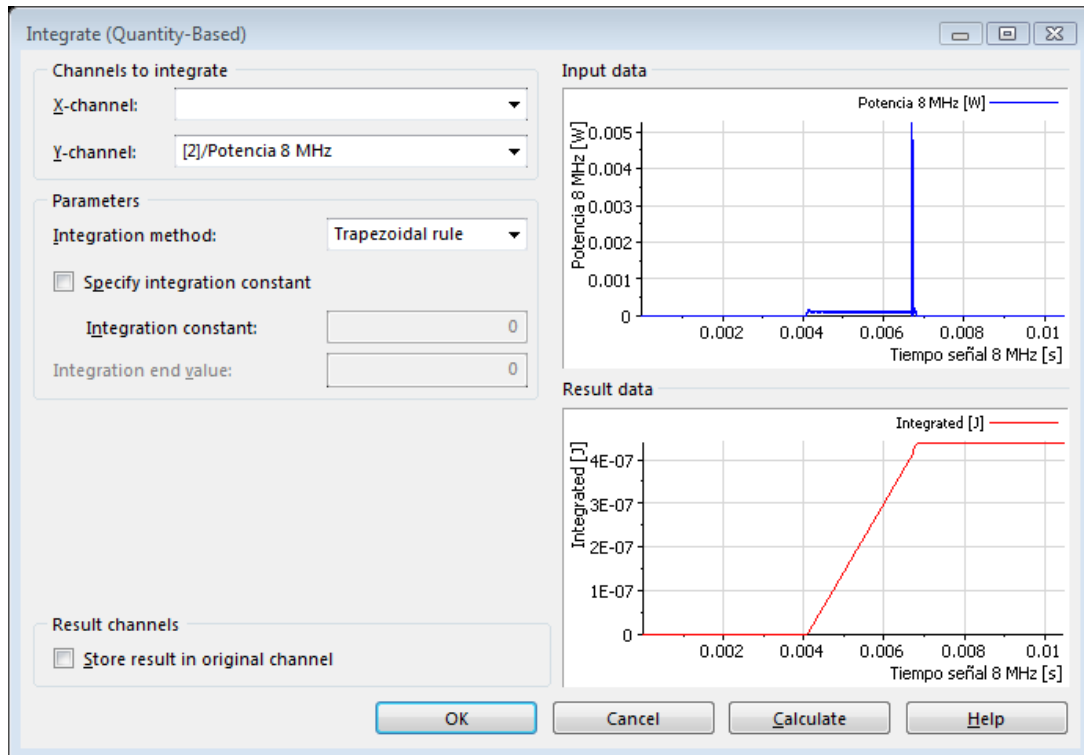


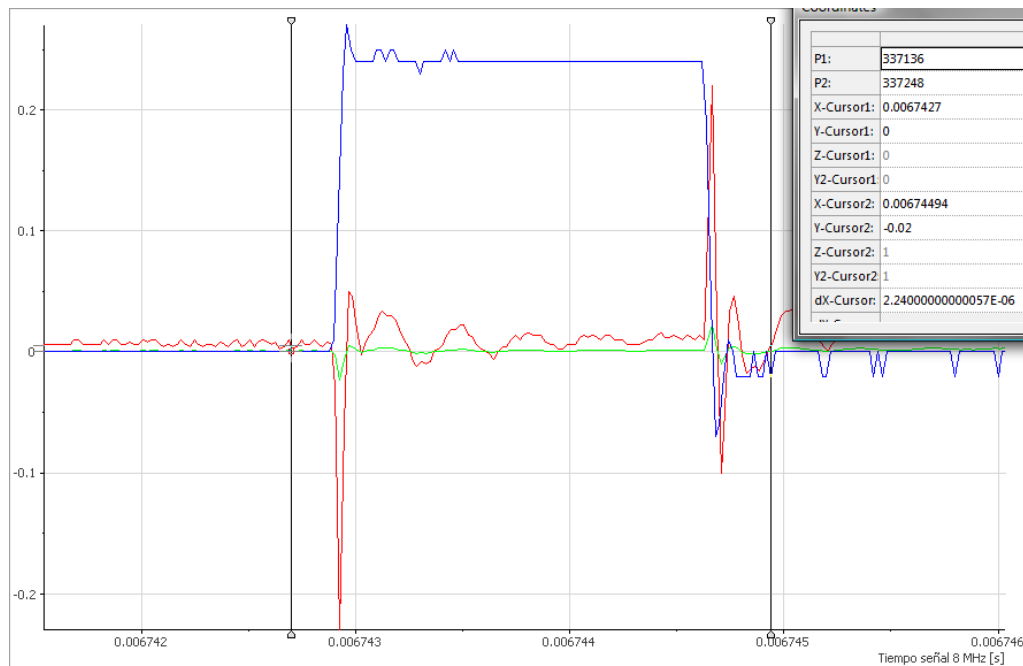
Fig. 31 Cálculo de la energía total

### 3.3.2 Perfil de energía por secciones

Lo que permite identificar dónde se encuentra el algoritmo en diferentes secciones durante la medición es el pulso de sincronía. Como se ha explicado anteriormente, este pulso indica cuándo debe ocurrir la captura de la señal sincronizando los diferentes perfiles creados por las frecuencias utilizadas.

Para realizar este apartado se reduce la señal de sincronía capturada utilizando una herramienta de DIADEM en análisis→matemáticas básicas→escalamiento. Se utiliza un factor de 1/20 para reducir la señal original y poder realizar una superposición de la señal del perfil de energía con la de sincronía. Al capturarse en el mismo punto, el disparo permite el conteo de los cinco ciclos de reloj del pulso, además de los cuatro necesarios para realizar la configuración de la terminal como salida, enviar un uno lógico, configurar la terminal de nuevo como salida y enviar un cero lógico. Se observa en la figura 32 las tres

señales superpuestas. La azul es el pulso de sincronía, la roja es el perfil de energía en voltaje y la verde es el perfil de energía en corriente; las dos líneas verticales determinan dos puntos para medición en todas las señales. Estas líneas verticales en conjunto con la herramienta de coordenadas permiten medir cuántos pulsos de reloj (en tiempo) se encuentran entre ellas.



**Fig. 32 Medición de los pulsos de reloj del algoritmo utilizando 64 bits a 8 MHz**

Al establecer dónde inicia y termina el pulso de reloj, aunado con el conteo de pulsos necesarios para cada fragmento de código generado por el compilador utilizando el algoritmo, se prosigue a segmentar cada sección del mismo contando el tiempo entre cada uno de ellos. En la gráfica 33 se observa cada sección del código para el algoritmo utilizando variables de 64 bits a una frecuencia de 8 MHz:

- C0 es la sección roja,
- la configuración para generar todas las terminales como salidas es la sección verde,
- el pulso de sincronía es la sección azul,
- la asignación de datos a las variables es la sección morada,
- la operación aritmética de suma es la sección turquesa,
- la operación aritmética de resta es la sección gris,

- la operación aritmética de multiplicación es la sección amarilla,
- la operación aritmética de división es la sección café y,
- el *WDT* es la sección verde oscuro.

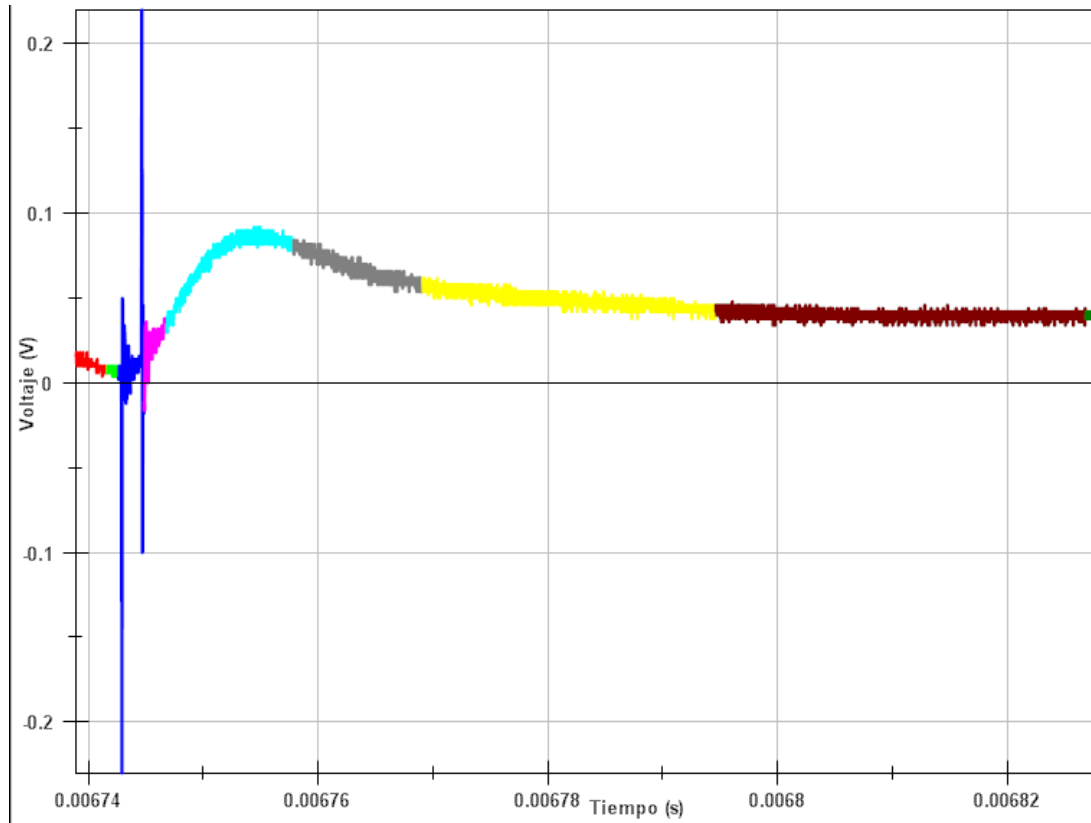


Fig. 33 Perfil de energía por secciones del algoritmo con variables de 64 bits a 8 MHz



### Resultados y discusión

En este capítulo se muestran los resultados de la investigación. La información se presenta en primer término con los datos obtenidos del algoritmo en cuestión de tiempo y posteriormente en relación a la energía consumida. Cada sección del algoritmo implementado por el compilador se revisa y compara respecto a las diferentes frecuencias utilizadas. Por último, se realiza una discusión de los resultados y posibles mejoras.

#### 4.1 Ciclos de reloj del algoritmo

Al aplicar el algoritmo con un lenguaje de alto nivel (C), el compilador genera el código ensamblador necesario para implementar dicho algoritmo. Además produce código de inicialización (C0) que no es posible evitar. Este código puede variar dependiendo el microcontrolador utilizado, ya que la arquitectura probablemente sea diferente y/o algún módulo del microcontrolador no esté implementado por el compilador.

La tabla 1 muestra la información generada para el algoritmo por el compilador utilizando variables de 8, 16, 32 y 64 bits. Dicho algoritmo se encuentra en secciones para analizar dónde se producen diferencias respecto a los ciclos de reloj necesarios. Se observa que independientemente del tipo de variable utilizada lo correspondiente a C0, configuración y pulso de sincronía utilizan el mismo número de ciclos de reloj para implementarse. Esto es debido a que el compilador no ha empezado a realizar las operaciones necesarias para el movimiento de datos.

Existe una diferencia del 3.5% en ciclos de reloj cuando se ejecutan operaciones de 8 bits respecto a los de 16 bits. El algoritmo de 8 bits se ejecuta más rápido que el de 16 bits, sin embargo no es significativo en tiempo. La explicación para este fenómeno es que el microcontrolador a utilizar tiene una arquitectura de 16 bits. El mover 8 bits o 16 bits es realizado en una instrucción que invierte un ciclo reloj. Es recomendable utilizar variables de 16 bits para realizar operaciones aritméticas, sin embargo se debe realizar el perfil de energía para asegurar que el movimiento de datos de 16 bits no afecte el consumo de corriente y por ende la energía a utilizar.

Al realizar operaciones aritméticas de 16 bits respecto a las de 32 bits existe una diferencia del 46% en ciclos de reloj. Al ser variables de 32 bits y la arquitectura de 16 bits, el número de ciclos de reloj necesarios para implementar el algoritmo es significativo. Es evidente a partir de la asignación de las variables que prácticamente se duplica el número de ciclos necesarios para realizar la sección del algoritmo.

Por último, al realizar operaciones aritméticas de 32 bits respecto a las de 64 bits existe una diferencia del 42% en ciclos de reloj necesarios. Este incremento es similar al de 16 bits a 32.

Tabla 1 Ciclos de reloj para cada sección del algoritmo

	8 bit	16 bit	32 bit	64 bit
C0	11	11	11	11
Configuración	4	4	4	4
Pulso sincronía	9	9	9	9
Inicialización variables	3	2	4	8
Operación Suma	12	15	26	44
Operación Resta	14	16	27	45
Operación Multiplicación	15	16	59	102
Operación División	35	34	62	129
WDT	5	5	5	5
<b>Total</b>	<b>108</b>	<b>112</b>	<b>207</b>	<b>357</b>

En la figura 43 es evidente que la relación de ciclos de reloj para el algoritmo con 8 bits y 16 bits es similar y, a partir de los 16 bits, la relación es casi lineal.

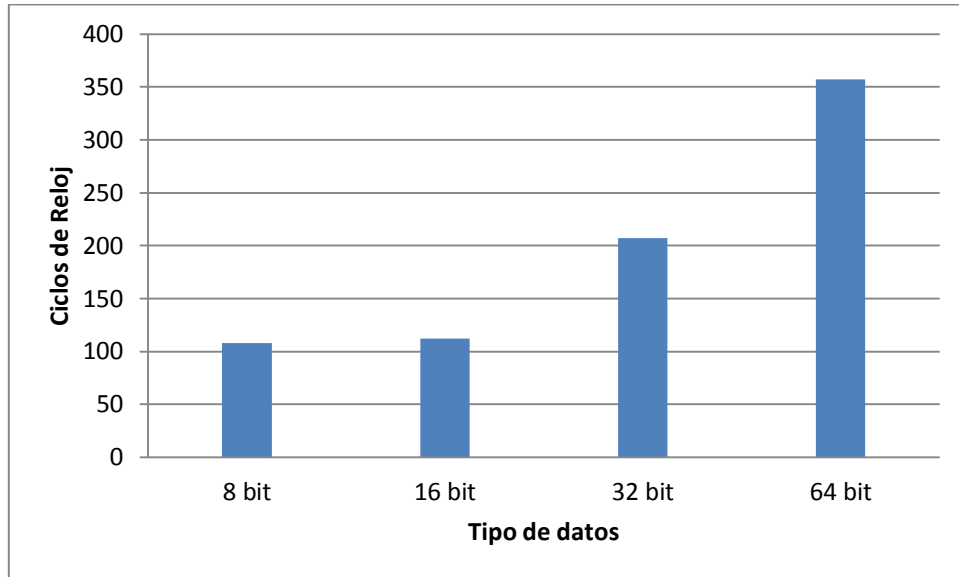


Fig. 34 Ciclos de reloj del algoritmo en cada programa

La figura 35 presenta el comportamiento del algoritmo desde su inicio hasta su término. La implementación de C0, configuración y pulso de sincronía es básicamente la misma en términos de ciclos de reloj. Se observa cómo para operaciones de 8 bits y 16 bits es básicamente el mismo número de ciclos de reloj. A partir de las operaciones aritméticas los ciclos de reloj necesarios se incrementan en cerca del doble.

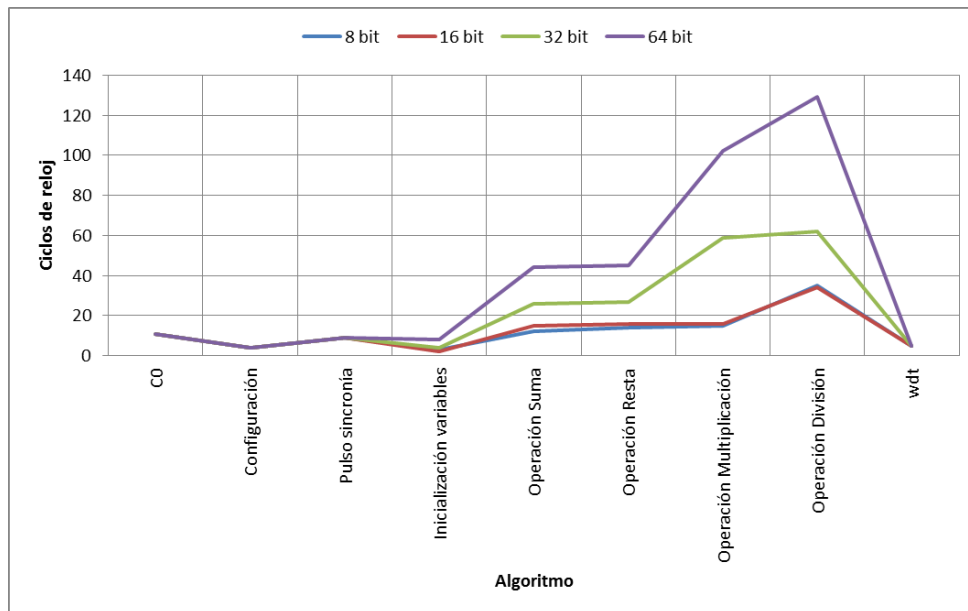


Fig. 35 Ciclos de reloj del algoritmo

## 4.2 Energía del algoritmo

La medición de la energía se explica en el capítulo anterior sobre metodología. Se presenta a continuación los resultados de energía estática y dinámica obtenidos por el algoritmo desde que inicia hasta que termina y por secciones. La información se presenta comparando los valores de energía totales respecto a las frecuencias utilizadas.

### 4.2.1 Energía *Wake Up* y total

El consumo de energía estática y dinámica del algoritmo a diferentes frecuencias presenta un fenómeno interesante en todas las pruebas. La energía consumida por la sección estática es significativamente mayor que su contraparte dinámica. Se observa en la figura 36 la relación entre la energía estática y la energía total (*Wake Up* y total). La sección *Wake Up* a diferentes frecuencias para el algoritmo utilizando variables enteras sin signo de 8 bits es en promedio el 84% del total. Para la misma sección pero ahora el algoritmo aplicado a variables enteras sin signo de 16 bits es en promedio el 84% del total. La figura permite revisar también que para las condiciones del algoritmo ejecutado con variables enteras sin signo de 32 bits el promedio entre la sección de *Wake Up* y el total es del 88%. Bajo el mismo principio, el promedio cuando el algoritmo se ejecuta con variables enteras sin signo de 64 bits es de 78%. La sección de *Wake Up* en promedio es el 83% del total del consumo de energía.

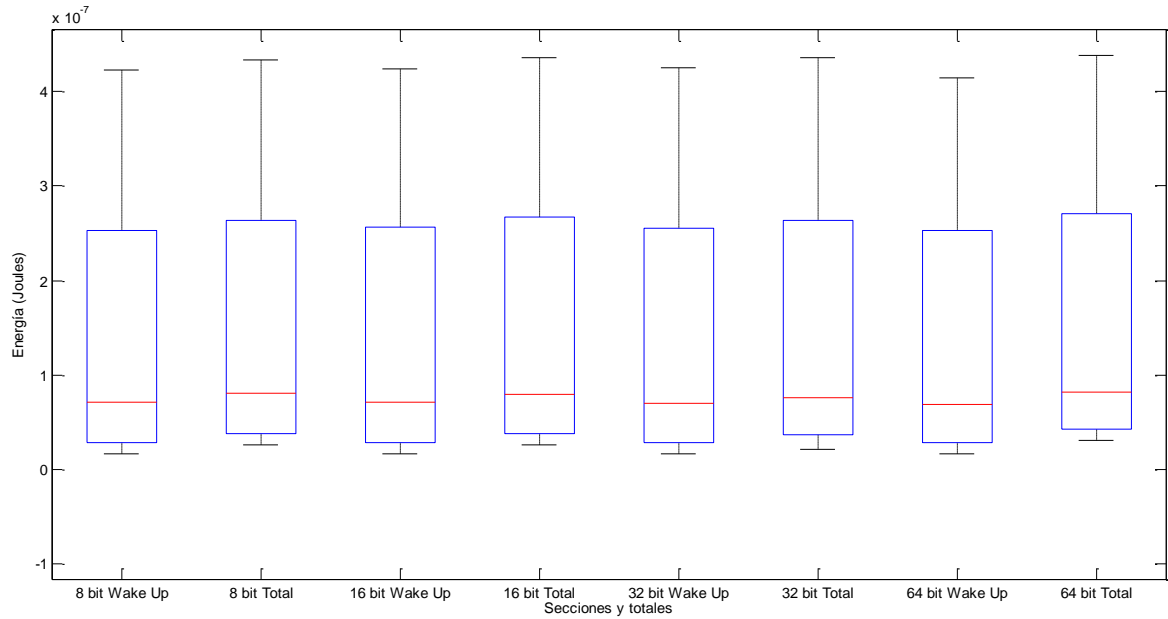


Fig. 36 Wake Up y Totales

Debido al aporte del consumo de energía de la sección de *Wake Up* respecto al total del algoritmo, se decide realizar el análisis del consumo de energía separando esta sección. Esto se observa en [54], donde el consumo de energía para el estado de *Power Up* puede ser muy alto en periodos cortos de tiempo (en el caso propio se le conoce como *Wake Up*).

## 4.2.2 Energía del algoritmo para 8 bits

La tabla 2 muestra el resultado de cada sección del algoritmo en Joules por cada frecuencia implementada de las secciones del algoritmo ejecutado con variables enteras sin signo de 8 bits.

**Tabla 2** Consumo de energía de 8 bits por secciones

8 bit	500KHz	1MHz	2MHz	4MHz	8MHz
8 bit Wake up	1.6138E-08	3.2199E-08	7.1321E-08	1.9663E-07	4.2299E-07
8 bit C0	1.0965E-10	1.0922E-10	6.4352E-11	4.0388E-11	3.7768E-11
8 bit Configuración	2.4476E-09	1.6595E-10	6.1920E-12	3.1880E-12	5.7840E-12
8 bit Pulso sincronía	6.7493E-09	7.6534E-09	3.0547E-09	4.9765E-10	3.3039E-10
8 bit Inicialización Variables	8.1680E-12	6.3744E-10	1.6104E-09	3.7296E-10	3.3796E-11
8 bit Operación Suma	1.5156E-11	4.1137E-10	2.9688E-09	2.9786E-09	4.8821E-10
8 bit Operación Resta	1.8232E-11	1.2916E-11	7.8444E-10	2.9286E-09	1.8231E-09
8 bit Operación Multiplicación	1.4923E-10	1.4923E-10	1.4923E-10	1.5691E-09	2.8130E-09
8 bit Operación División	3.2322E-10	6.1492E-11	5.5456E-11	1.5894E-09	4.9391E-09
8 bit WDT	1.8294E-10	3.4784E-11	1.1001E-10	1.1001E-10	4.8633E-10
8 bit Total	<b>2.6142E-08</b>	<b>4.1435E-08</b>	<b>8.0125E-08</b>	<b>2.0672E-07</b>	<b>4.3395E-07</b>

La información recabada muestra un aumento en el consumo de energía a medida que se incrementa la frecuencia de operación. La figura 37 muestra un consumo de energía no lineal donde el requerimiento de energía de 500 KHz a 1 MHz es 37% mayor; de 1 MHz a 2 MHz es 48% mayor; de 2 MHz a 4 MHz es 61% mayor y; de 4 MHz a 8 MHz es 52% mayor.

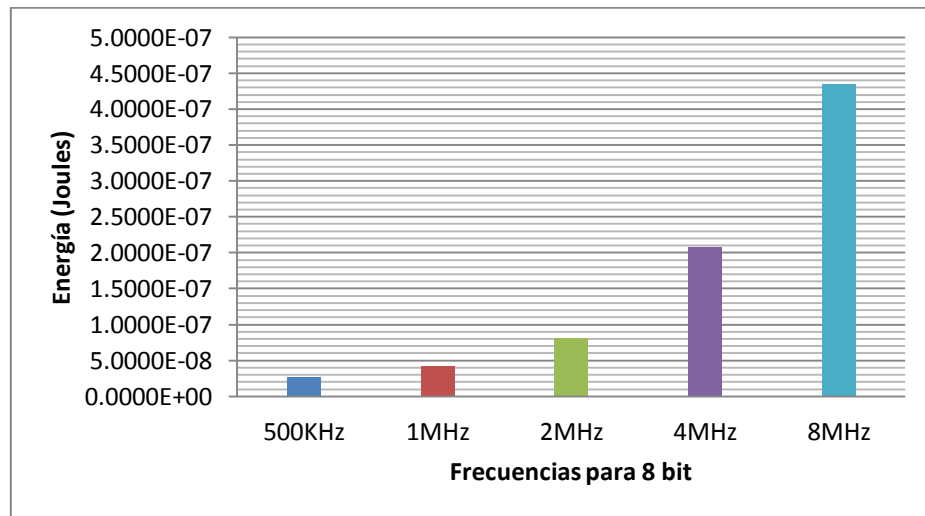


Fig. 37 Consumo de energía Total para 8 bits

La sección de *Wake Up* consume la mayor cantidad de energía en el algoritmo de 8 bits. Se observa en la figura 38 la dispersión en cada frecuencia. Los valores extremos muestran el aporte en consumo de energía de cada sección de *Wake Up* con diferentes frecuencias. Dicha sección representa un consumo de 61% del total en una frecuencia de 500 KHz; un consumo de 77% respecto al total con una frecuencia de 1 MHz. Al ejecutar el algoritmo en la frecuencia de 2 MHz, el consumo de la sección respecto al total es de 89%; con una frecuencia de 4 MHz el consumo de la sección respecto al total es de 95%. Por último, con una frecuencia de 8 MHz el consumo de energía de la sección respecto al total es de 97%.

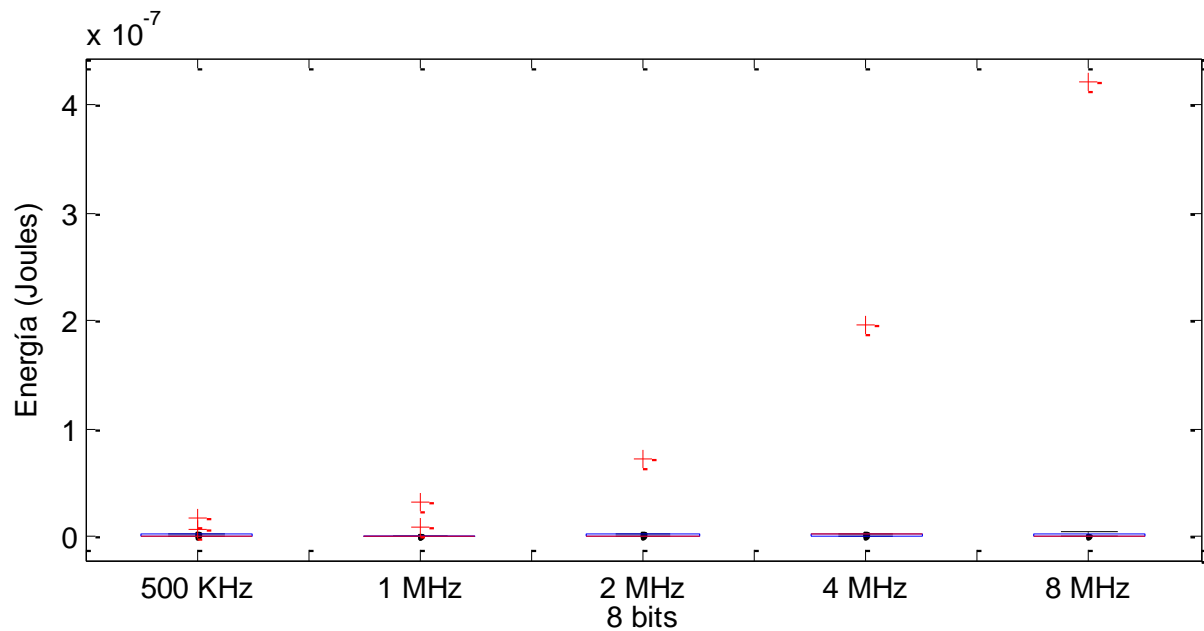


Fig. 38 Dispersión de energía por frecuencias con 8 bits

La figura 39 presenta el consumo de energía por sección en cada frecuencia cuando el algoritmo se ejecuta utilizando variables enteras sin signo de 8 bits. Se observa un consumo de energía mayor en la frecuencia de 500 KHz y 1 MHz en la sección del pulso de sincronía.

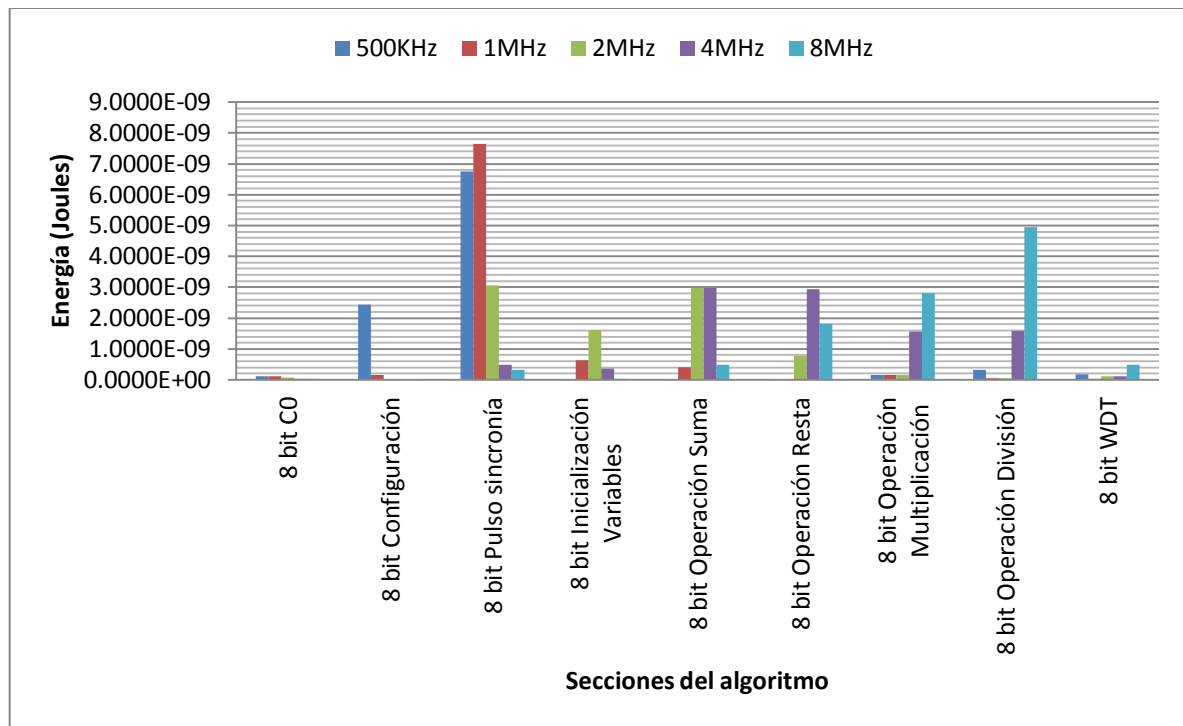


Fig. 39 Consumo de energía por secciones para 8 bits



### 4.2.3 Energía del algoritmo para 16 bits

La tabla 3 muestra el resultado de cada sección del algoritmo en Joules por cada frecuencia implementada de las secciones del algoritmo ejecutado con variables enteras sin signo de 16 bits.

Tabla 3 Consumo de energía de 16 bits por secciones

16 bit	500KHz	1MHz	2MHz	4MHz	8MHz
16 bit Wake up	1.6190E-08	3.2531E-08	7.0629E-08	2.0060E-07	4.2436E-07
16 bit C0	1.6171E-10	8.5624E-11	6.5404E-11	2.8976E-11	2.8192E-11
16 bit Configuración	2.5096E-09	1.6760E-10	7.0640E-12	2.1840E-12	4.1280E-12
16 bit Pulso sincronía	6.8509E-09	7.5297E-09	3.2179E-09	5.2511E-10	2.5592E-10
16 bit Inicialización Variables	2.6400E-12	4.8503E-10	1.1697E-09	2.5450E-10	1.7772E-11
16 bit Operación Suma	1.9352E-11	6.0224E-10	3.8017E-09	3.9434E-09	7.5623E-10
16 bit Operación Resta	2.1656E-11	1.2948E-11	6.6880E-10	2.9807E-09	2.5403E-09
16 bit Operación Multiplicación	3.2804E-11	1.3564E-11	9.4656E-11	1.4324E-09	3.0200E-09
16 bit Operación División	4.1686E-10	9.1324E-11	3.8888E-11	1.3405E-09	4.4543E-09
16 bit WDT	1.9477E-10	4.5880E-11	7.6560E-12	9.0592E-11	4.8363E-10
16 bit Total	<b>2.6400E-08</b>	<b>4.1565E-08</b>	<b>7.9701E-08</b>	<b>2.1120E-07</b>	<b>4.3592E-07</b>

La información recabada muestra un aumento en el consumo de energía a medida que se incrementa la frecuencia de operación. La figura 40 muestra un consumo de energía no lineal donde el requerimiento de energía de 500 KHz a 1 MHz es 36% mayor; de 1 MHz a 2 MHz es 47% mayor; de 2 MHz a 4 MHz es 62% mayor y; de 4 MHz a 8 MHz es 52% mayor.

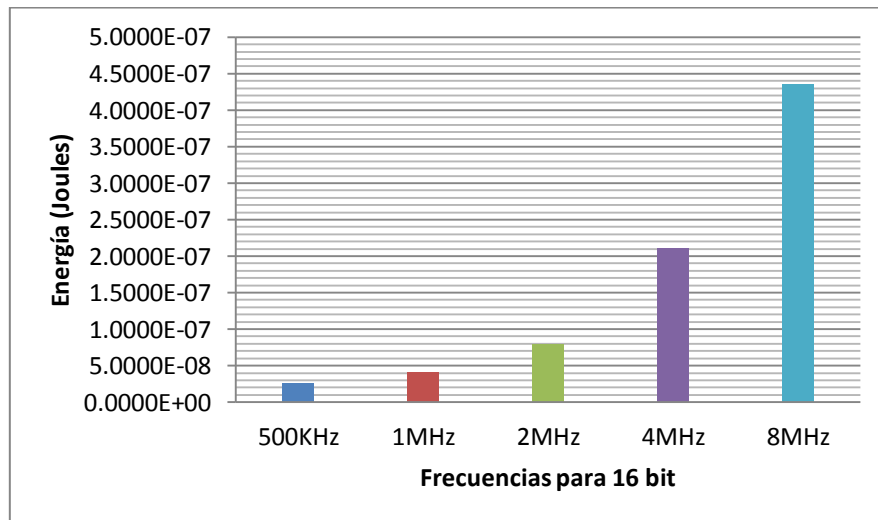


Fig. 40 Consumo de energía Total para 16 bits

La sección de *Wake Up* consume la mayor cantidad de energía en el algoritmo de 16 bits. Se observa en la figura 41 la dispersión en cada frecuencia. Los valores extremos muestran el aporte en consumo de energía de cada sección de *Wake Up* con diferentes frecuencias. Dicha sección representa un consumo de 61% del total en una frecuencia de 500 KHz; un consumo de 78% respecto al total con una frecuencia de 1 MHz. Al ejecutar el algoritmo en la frecuencia de 2 MHz, el consumo de la sección respecto al total es de 88%; con una frecuencia de 4 MHz el consumo de la sección respecto al total es de 94%. Por último, con una frecuencia de 8 MHz el consumo de energía de la sección respecto al total es de 97%.

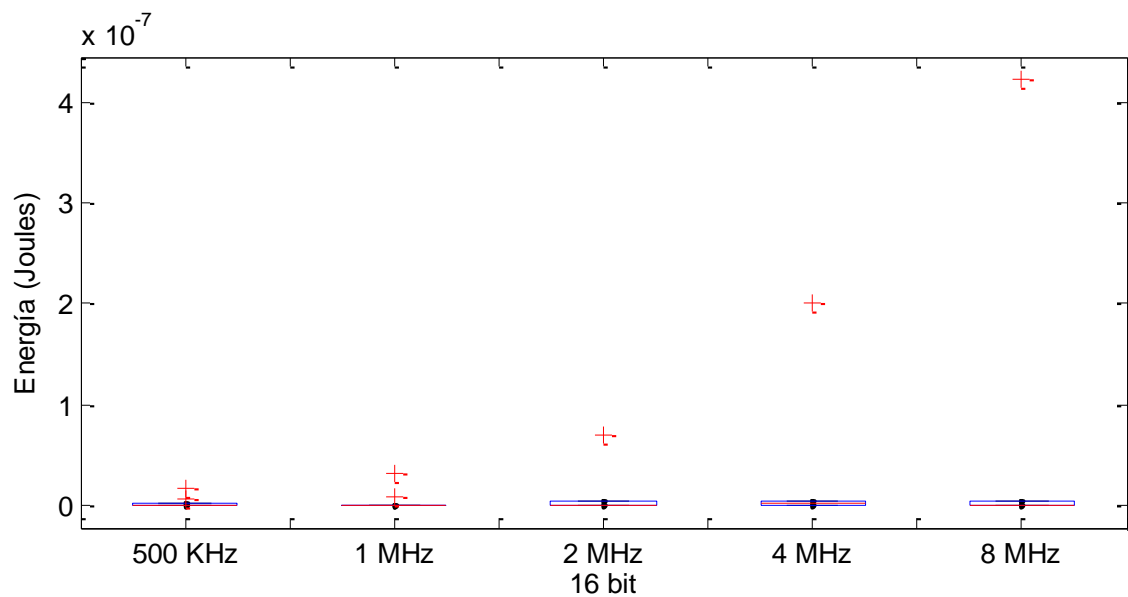


Fig. 41 Dispersión de energía por frecuencias con 16 bits

La figura 42 presenta el consumo de energía por sección en cada frecuencia cuando el algoritmo se ejecuta utilizando variables enteras sin signo de 16 bits. Se observa un consumo de energía mayor en la frecuencia de 500 KHz y 1 MHz en la sección del pulso de sincronía. En las secciones de resta, multiplicación y división se presenta un consumo de energía mayor en las frecuencias de 4 MHz y 8 MHz.

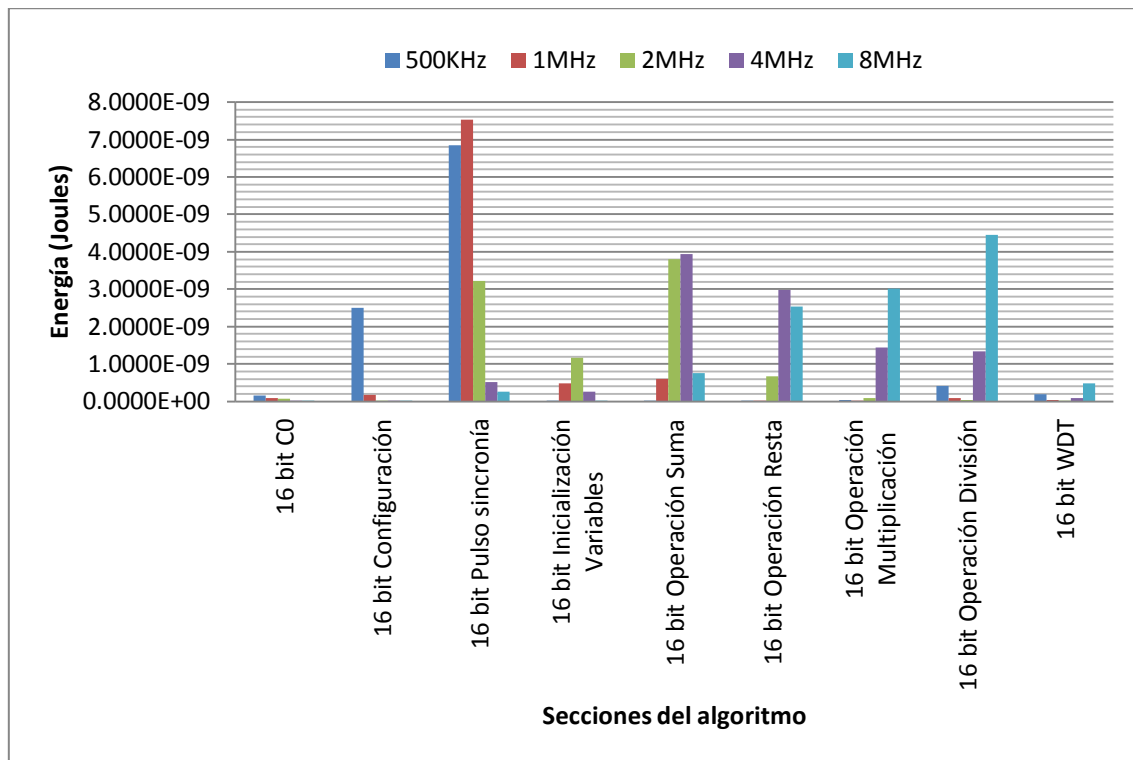


Fig. 42 Consumo de energía por secciones para 16 bits

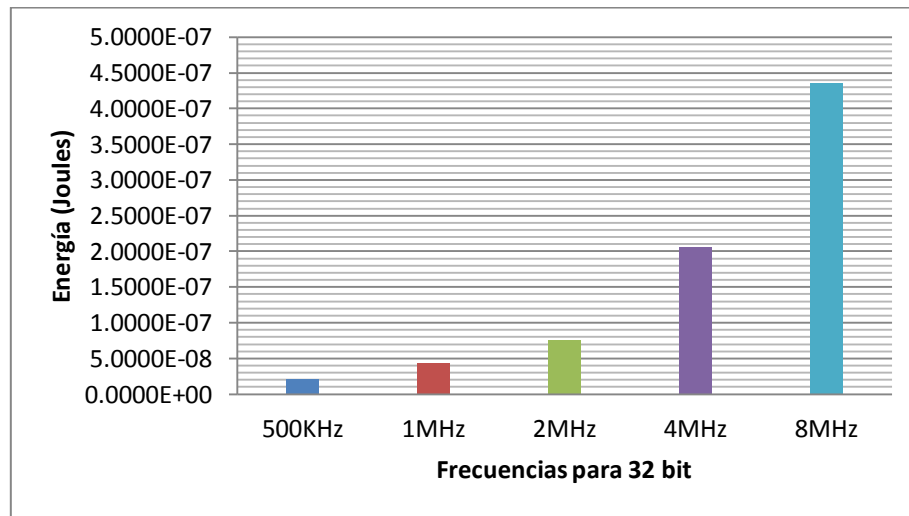
#### 4.2.4 Energía del algoritmo para 32 bits

La tabla 4 muestra el resultado de cada sección del algoritmo en Joules por cada frecuencia implementada de las secciones del algoritmo ejecutado con variables enteras sin signo de 32 bits.

**Tabla 4 Consumo de energía de 32 bits por secciones**

32 bit	500KHz	1MHz	2MHz	4MHz	8MHz
32 bit Wake up	1.6114E-08	3.3109E-08	7.0518E-08	1.9809E-07	4.2565E-07
32 bit C0	1.7402E-10	1.1669E-10	1.3358E-10	1.0952E-10	7.2656E-11
32 bit Configuración	1.6512E-11	1.2516E-10	4.1440E-12	6.6240E-12	9.3840E-12
32 bit Pulso sincronía	1.3494E-09	6.3343E-09	2.6216E-10	1.7005E-10	1.3372E-10
32 bit Inicialización Variables	1.1587E-10	6.9096E-10	1.9700E-10	3.9580E-11	1.3420E-11
32 bit Operación Suma	2.5669E-10	3.1614E-10	1.2099E-09	1.0433E-09	4.4130E-10
32 bit Operación Resta	5.4212E-10	2.0648E-11	7.2621E-10	1.2255E-09	1.3689E-09
32 bit Operación Multiplicación	1.1194E-09	6.4470E-10	1.2050E-09	2.4579E-09	3.5463E-09
32 bit Operación División	1.1252E-09	1.2216E-09	1.2601E-09	2.2759E-09	4.2478E-09
32 bit WDT	1.4416E-10	1.3180E-10	1.2266E-10	1.9051E-10	3.6755E-10
32 bit Total	<b>2.0957E-08</b>	<b>4.2711E-08</b>	<b>7.5639E-08</b>	<b>2.0561E-07</b>	<b>4.3585E-07</b>

La información recabada muestra un aumento en el consumo de energía a medida que se incrementa la frecuencia de operación. La figura 43 muestra un consumo de energía no lineal donde el requerimiento de energía de 500 KHz a 1 MHz es 50% mayor; de 1 MHz a 2 MHz es 43% mayor; de 2 MHz a 4 MHz es 66% mayor y; de 4 MHz a 8 MHz es 52% mayor.



**Fig. 43 Consumo de energía Total para 32 bits**

La sección de *Wake Up* consume la mayor cantidad de energía en el algoritmo de 32 bits. Se observa en la figura 44 la dispersión en cada frecuencia. Los valores extremos muestran el aporte en consumo de energía de cada sección de *Wake Up* con diferentes frecuencias. Dicha sección representa un consumo de 76% del total en una frecuencia de 500 KHz; un consumo de 77% respecto al total con una frecuencia de 1 MHz. Al ejecutar el algoritmo en la frecuencia de 2 MHz, el consumo de la sección respecto al total es de 93%; con una frecuencia de 4 MHz el consumo de la sección respecto al total es de 96%. Por último, con una frecuencia de 8 MHz el consumo de energía de la sección respecto al total es de 97%.

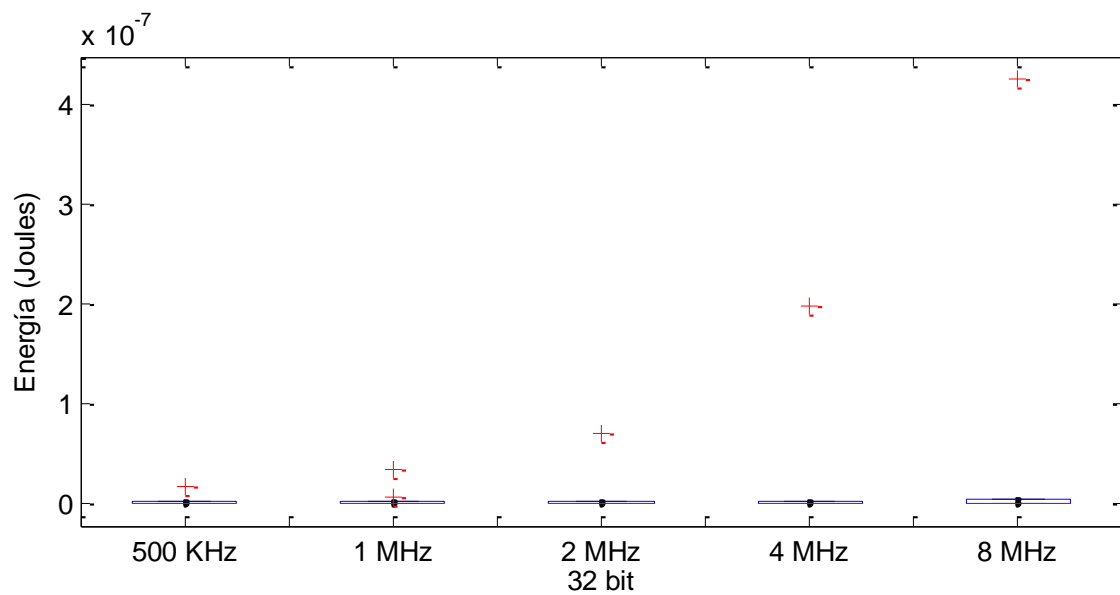


Fig. 44 Dispersión de energía por frecuencias con 32 bits

La figura 45 presenta el consumo de energía por sección en cada frecuencia cuando el algoritmo se ejecuta utilizando variables enteras sin signo de 32 bits. Se observa un consumo de energía mayor en la frecuencia de 1 MHz en la sección del pulso de sincronía. Las secciones de multiplicación y división muestran incrementos significativos respecto a las observadas en los algoritmos utilizando variables de 8 y 16 bits; entre estas secciones para variables de 32 bits presentan consumos similares.

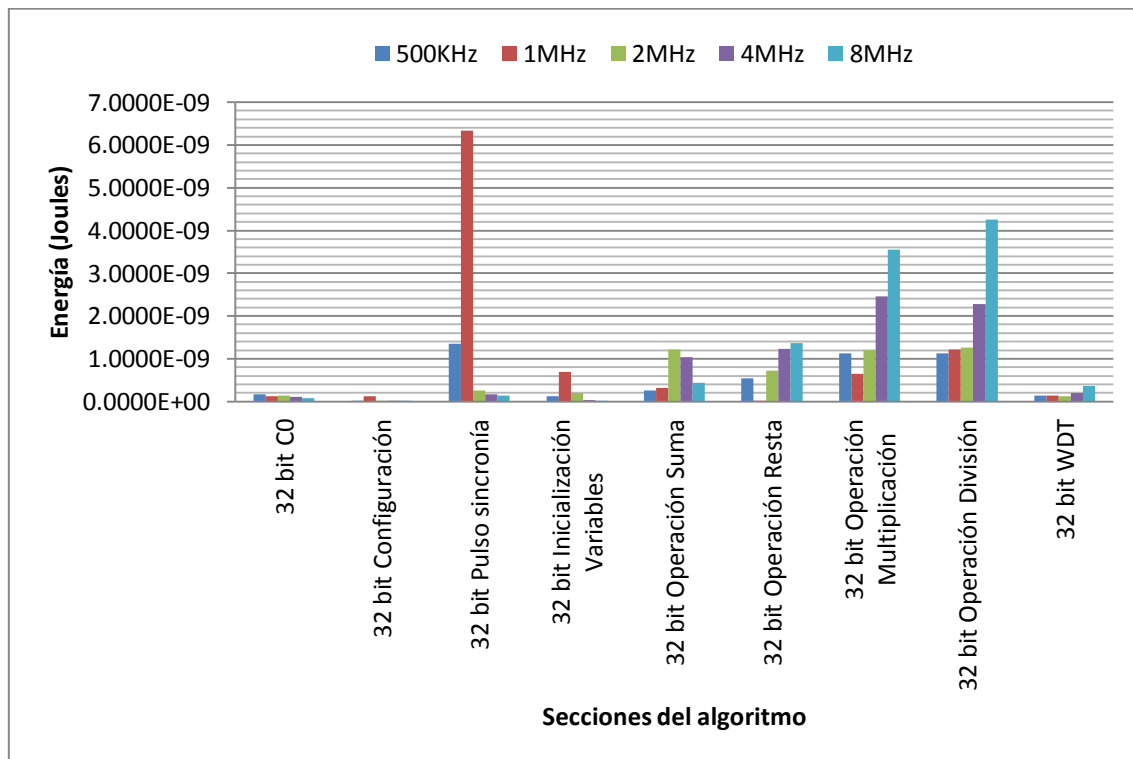


Fig. 45 Consumo de energía por secciones para 32 bits

## 4.2.5 Energía del algoritmo para 64 bits

La tabla 5 muestra el resultado de cada sección del algoritmo en Joules por cada frecuencia implementada de las secciones del algoritmo ejecutado con variables enteras sin signo de 64 bits.

Tabla 5 Consumo de energía de 64 bits por secciones

64 bit	500KHz	1MHz	2MHz	4MHz	8MHz
64 bit Wake up	1.6137E-08	3.3076E-08	6.8679E-08	1.9881E-07	4.1462E-07
64 bit C0	1.3088E-10	8.6616E-11	7.1928E-11	3.7008E-11	3.5012E-11
64 bit Configuración	2.0113E-09	1.4787E-10	5.4080E-12	2.9600E-12	5.3040E-12
64 bit Pulso sincronía	5.7897E-09	6.6823E-09	2.5402E-09	5.4835E-10	3.1699E-10
64 bit Inicialización Variables	1.3216E-11	9.5850E-10	3.0250E-09	1.4211E-09	1.4240E-10
64 bit Operación Suma	9.4992E-11	1.2812E-10	2.1586E-09	6.9237E-09	6.3528E-09
64 bit Operación Resta	7.5216E-10	3.2347E-10	6.4808E-11	1.4159E-09	5.2722E-09
64 bit Operación Multiplicación	1.8774E-09	2.0115E-09	2.1338E-09	1.1561E-09	6.2922E-09
64 bit Operación División	4.2742E-09	2.4703E-09	3.0988E-09	4.9056E-09	5.2710E-09
64 bit WDT	1.7523E-10	1.3021E-10	1.0924E-10	2.2162E-10	2.0566E-10
64 bit Total	<b>3.1256E-08</b>	<b>4.6014E-08</b>	<b>8.1887E-08</b>	<b>2.1544E-07</b>	<b>4.3852E-07</b>

La información recabada muestra un aumento en el consumo de energía a medida que se incrementa la frecuencia de operación. La figura 46 muestra un consumo de energía no lineal donde el requerimiento de energía de 500 KHz a 1 MHz es 32% mayor; de 1 MHz a 2 MHz es 43% mayor; de 2 MHz a 4 MHz es 61% mayor y; de 4 MHz a 8 MHz es 50% mayor.

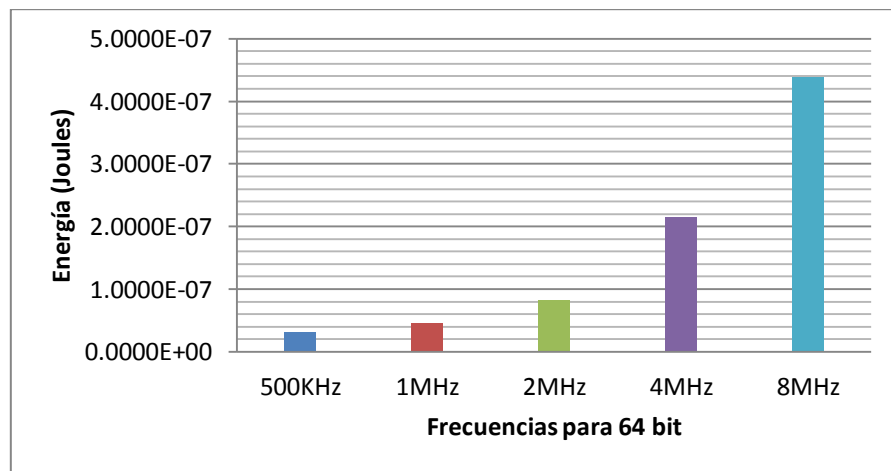


Fig. 46 Consumo de energía Total para 64 bits

La sección de *Wake Up* consume la mayor cantidad de energía en el algoritmo de 64 bits. Se observa en la figura 47 la dispersión en cada frecuencia. Los valores extremos muestran el aporte en consumo de energía de cada sección de *Wake Up* con diferentes frecuencias. Dicha sección representa un consumo de 51% del total en una frecuencia de 500 KHz; un consumo de 71% respecto al total con una frecuencia de 1 MHz. Al ejecutar el algoritmo en la frecuencia de 2 MHz, el consumo de la sección respecto al total es de 83%; con una frecuencia de 4 MHz el consumo de la sección respecto al total es de 92%. Por último, con una frecuencia de 8 MHz el consumo de energía de la sección respecto al total es de 94%.

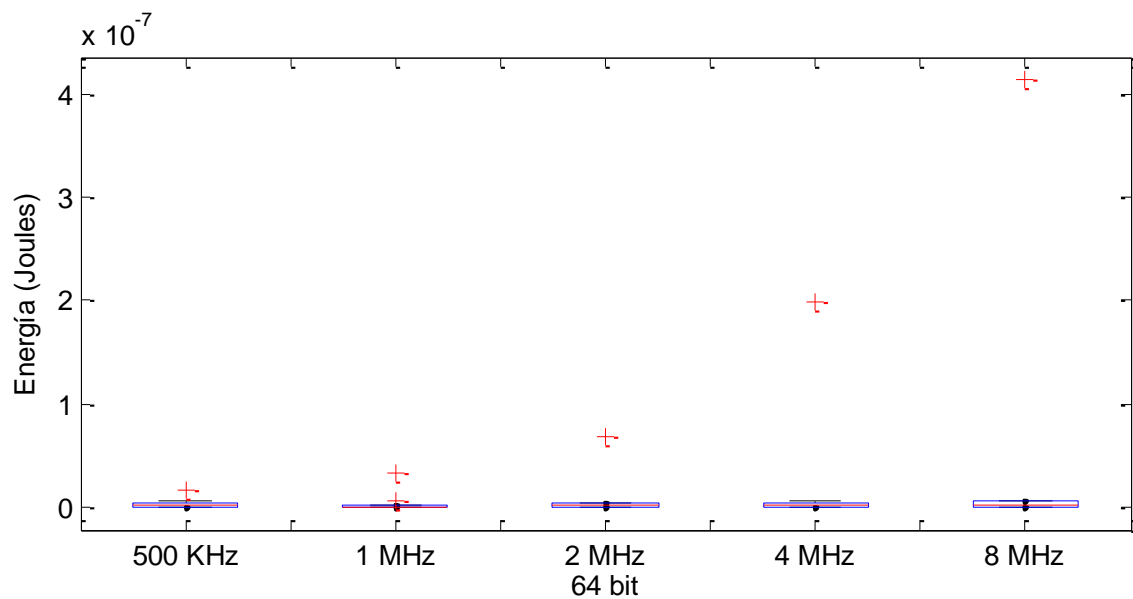


Fig. 47 Dispersión de energía por frecuencias con 64 bits



La figura 48 presenta el consumo de energía por sección en cada frecuencia cuando el algoritmo se ejecuta utilizando variables enteras sin signo de 64 bits. Se observa un consumo de energía mayor en la frecuencia de 500 KHz y 1 MHz en la sección del pulso de sincronía. Se observa que la frecuencia de 8 MHz en la mayoría de las secciones es la que provoca un mayor consumo de energía.

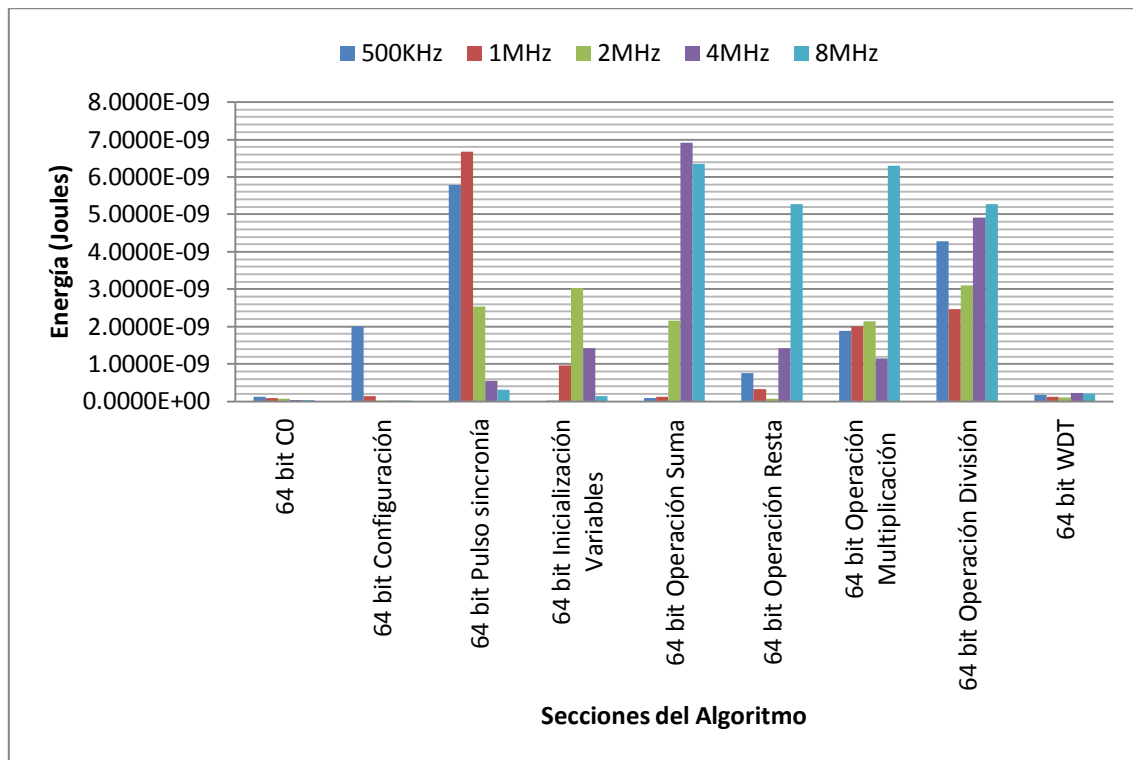


Fig. 48 Consumo de energía por secciones para 64 bits

### 4.3 Conclusiones

Se evaluó el consumo de energía del microcontrolador PIC24FV32KA302 utilizando la técnica de medición para el lado bajo de corriente. La batería de pruebas realizadas mostró un consumo de energía casi lineal en todas las frecuencias.

El consumo mayor de energía se presentó en la sección de reinicio del algoritmo para todas las frecuencias con un promedio del 84% de la energía total. Dicho consumo puede presentarse en casos extremos (8 MHz) de hasta un 97% del total. Por esta razón fue necesario realizar un análisis comparativo por separado de las secciones de *Wake Up* y el algoritmo de operaciones aritméticas.

En el apartado de las operaciones aritméticas el consumo de energía registrado fue similar entre frecuencias y algoritmos de diferentes extensiones de bits, sin embargo en la frecuencia mayor, aunque se consume la mayor cantidad de energía debido a la sección de *Wake Up*, presenta una reducción en el consumo.

En la medida que se aumentaba la frecuencia también lo hacía el consumo de energía, al encontrarse en promedio 38% para el cambio de 500 KHz a 1 MHz; para la frecuencia de 1 MHz a 2 MHz el consumo de energía en promedio es de 45% mayor; al utilizar la frecuencia de 2 MHz a 4 MHz el promedio en incremento del consumo es de 62%. El consumo mayor promedio para la frecuencia de 4 MHz a 8 MHz fue de 52%. Esto es una reducción en el consumo pero solo para el algoritmo.

Debido al pulso de sincronía fue posible detectar el punto en el que se encontraba cada instrucción en lenguaje ensamblador y de esta manera identificar en el perfil de energía dónde se encontraba el consumo de energía instantáneo. Esta medición fue posible realizarla en lado bajo ya que la referencia del osciloscopio era la misma que la de polarización y, al no tener una punta de prueba completamente diferencial, la única solución para sincronizarse y disparar la adquisición fue la antes señalada. Se utilizó un osciloscopio con una profundidad de 1 Mega muestras por lo que el pulso también debía ser capturado, razón por la cual se decidió utilizar un pulso de pocos ciclos de reloj en lugar de un pulso con un tiempo mayor (1 mSeg por ejemplo).

En relación a los ciclos de reloj necesarios para la implementación de la batería de pruebas, se observa que es cerca de 40% mayor en ciclos de reloj necesarios para implementar las funciones de 16 bits a 32 bits; algo similar ocurre de 32 bits a 64 bits. Una situación diferente es de 8 bits a 16 bits, solo es una diferencia del 3.5%. Como se sugiere en la investigación, esto se debe a que se trata de una arquitectura para un microcontrolador de 16 bits, por lo que mover 8 bits o 16 bits requiere el mismo número de ciclos de reloj.

Al realizar la batería de pruebas se presentó una problemática con los microcontroladores: no fue posible reprogramarlos. Después de presentar la configuración a las compañías del microcontrolador (Microchip) y del compilador (Custom Computer Services) tanto en hardware y software se presentaron los siguientes hallazgos:

1. Custom Computer Services se percató que su compilador al colocar los bits de configuración adecuados para una frecuencia interna que no pueda generarse por el módulo interno del oscilador no lo hacía. Su compilador simplemente colocaba información para proteger el microcontrolador. Después de estar en contacto con ellos y hacerles saber de esta situación, 12 versiones después del compilador utilizado (5.023→5.035) si el usuario solicita una frecuencia interna a través del módulo que no se pueda generar, el compilador detecta el error y no genera el código necesario para programación (se presenta esta conversación en el anexo 2 resaltando donde reconocen la situación).
2. Microchip reprodujo la configuración utilizada para programar los microcontroladores donde ya no era posible reprogramarlos y sugirió alternativas de programación (se presenta la conversación con el soporte técnico en el anexo 1 resaltando donde reconocen la situación).

Este descubrimiento alertó a las compañías del probable problema que se podría presentar cuando el usuario no utiliza de manera adecuada las directivas de preprocesamiento para la configuración del oscilador interno.

# Bibliografía

- [1] K. Dhondge, Shorey and R. Tew, "HOLA: Heuristic and opportunistic link selection algorithm for energy efficiency in Industrial Internet of Things (IIoT) systems," *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, pp. 1-6, 2016.
- [2] S. Sridhara, "Ultra-low power microcontrollers for portable, wearable, and implantable medical electronics," *16th Asia and South Pacific Design Automation Conference*, pp. 556-560, 2011.
- [3] Buntz, Brian, "http://www.emdt.co.uk," 1 Junio 2010. [Online]. Available: <http://www.emdt.co.uk/article/developing-medical-device-software-iso-62304>. [Accessed 13 Septiembre 2013].
- [4] K. Sandler, L. Ohrstrom, L. Moy and R. McVay, "Software Freedom Law Center," 21 Julio 2010. [Online]. Available: <http://www.softwarefreedom.org/resources/2010/transparent-medical-devices.html>. [Accessed 25 Septiembre 2013].
- [5] "http://www.industriamedica.org," [Online]. Available: <http://www.industriamedica.org/directorio.php>. [Accessed 14 Septiembre 2013].
- [6] "http://www.seccionamarilla.com.mx," [Online]. Available: <http://www.seccionamarilla.com.mx/resultados/hospitales/baja-california/1>. [Accessed 30 Septiembre 2013].
- [7] FDA, "Food and Drug Administration," 23 Abril 2010. [Online]. Available: <http://www.fda.gov/medicaldevices/productsandmedicalprocedures/generalhospitaldevicesandsupplies/infusionpumps/ucm206000.htm>. [Accessed 29 Octubre 2013].
- [8] E. Jefferson, "http://www.fda.gov," Food and Drug Administration, 22 03 2013. [Online]. Available: <http://www.fda.gov/NewsEvents/Newsroom/PressAnnouncements/ucm345062.htm>. [Accessed 12 Octubre 2013].
- [9] P. Jordan, "Standard IEC 62304 - Medical Device Software - Software Lifecycle

- Processes," *Software for Medical Devices, The Institution of Engineering and Technology Seminar on*, pp. 41-47, 2006.
- [10] J. Hoyo, "http://www.cenetec.salud.gob.mx," 28 Junio 2007. [Online]. Available: [http://www.cenetec.salud.gob.mx/descargas/normas/Normas\\_ISO.pdf](http://www.cenetec.salud.gob.mx/descargas/normas/Normas_ISO.pdf). [Accessed 5 Octubre 2013].
  - [11] N. Leveson and C. Turner, "An investigation of the Therac-25 accidents," *Computer*, vol. 26, no. 7, pp. 18-41, 1993.
  - [12] Mathworks, "Mathworks," 22 Octubre 2013. [Online]. Available: [http://www.mathworks.com/videos/developing-iec-62304-compliant-medical-device-software-using-model-based-design-86469.html?form\\_seq=conf1386&confirmation\\_page&wfsid=5070835](http://www.mathworks.com/videos/developing-iec-62304-compliant-medical-device-software-using-model-based-design-86469.html?form_seq=conf1386&confirmation_page&wfsid=5070835). [Accessed 22 Octubre 2013].
  - [13] M. McHugh, F. McCaffery and V. Casey, "Software process improvement to assist medical device software development organisations to comply with the amendments to the medical device directive," *Software, IET*, vol. 6, no. 5, pp. 431-437, 2012.
  - [14] P. Jordan, "Medical device software standards," *Medical Device Standards & Regulations, IEE Seminar on*, pp. 6/1 - 6/6, 2005.
  - [15] B. Belvedere, M. Bianchi, A. Borghetti, C. Nucci, M. Paolone and A. Peretto, "A Microcontroller-Based Power Management System for Standalone Microgrids With Hybrid Power Supply," *Sustainable Energy, IEEE Transactions on*, vol. 3, no. 3, pp. 422-431, 2012.
  - [16] Digi-Key, "http://www.designnews.com," 14 Enero 2013. [Online]. Available: [http://www.designnews.com/lecture-calendar.asp?p\\_l\\_ed=CEC\\_Semester\\_Three\\_2013#lecture\\_track\\_cgcid\\_168](http://www.designnews.com/lecture-calendar.asp?p_l_ed=CEC_Semester_Three_2013#lecture_track_cgcid_168). [Accessed 31 Agosto 2013].
  - [17] G. Luo, B. Guo, Y. Shen, H. Liao and L. Ren, "Analysis and Optimization of Embedded Software Energy Consumption on the Source Code and Algorithm Level," *Embedded and Multimedia Computing*, pp. 1-5, 2009.
  - [18] Microchip, "MPLAB REAL ICE Power Monitor," Microchip, 4 12 2013. [Online].

Available:

<http://www.microchip.com/Developmenttools/ProductDetails.aspx?PartNO=AC244008>. [Accessed 16 Febrero 2014].

- [19] Texas, "MSP EnergyTrace Technology," TI, 2014. [Online]. Available: <http://www.ti.com/tool/energytrace>. [Accessed 10 Febrero 2014].
- [20] Renesas Electronics, "<http://www.techonline.com>," 18 Septiembre 2013. [Online]. Available: <http://www.techonline.com/electrical-engineers/education-training/webinars/4420344/Implementing-Ultra-low-Power-Design-Techniques-for-High-performance-32-bit-MCU-based-Applications>. [Accessed 18 Septiembre 2013].
- [21] A. Holberg, "ATMEL," 02 2006. [Online]. Available: [www.atmel.com](http://www.atmel.com). [Accessed 13 Diciembre 2013].
- [22] B. Ivey, "Microchip," 2011. [Online]. Available: [www.microchip.com](http://www.microchip.com). [Accessed 7 Diciembre 2013].
- [23] Tsekoura, Rebel, Glösekötter and Berekovic, "An evaluation of energy efficient microcontrollers," *Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, pp. 1-5, 2014.
- [24] R. Richey, *Low Power Design Using PICmicro Microcontrollers AN606*, 1997.
- [25] "ST," Mayo 2013. [Online]. Available: [www.st.com](http://www.st.com). [Accessed 1 Diciembre 2013].
- [26] V. Tiwari, S. Malik and A. Wolfe, "Power Analysis Of Embedded Software: A First Step Towards Software Power Minimization," *Computer-Aided Design, IEEE/ACM International Conference on*, pp. 384-390, 1994.
- [27] V. Tiwari, S. Malik and A. Wolfe, "Compilation Techniques for Low Energy: An Overview," *Low Power Electronics, Digest of Technical Papers., IEEE Symposium*, pp. 38-39, 1994.
- [28] C. Robertson and C. Martinez, "Analyzing the software aspect of an embedded system's power consumption," *Electrical and Computer Engineering (CCECE), 24th Canadian Conference on*, pp. 853-856, 2011.
- [29] D. Ortiz and N. Santiago, "Impact of Source Code Optimizations on Power Consumption of Embedded Systems," *Circuits and Systems and TAISA Conference*,

- pp. 133-136, 2008.
- [30] M. Ibrahim, M. Rupp and S. E. -D. Habib, "Compiler-Based Optimizations Impact on Embedded Software Power Consumption," *Circuits and Systems and TAISA Conference*, pp. 1-4, 2009.
  - [31] J. Liao, X. Guo, C. Luo, M. Zhu and Z. Yang, "Studies of Rogowski coil current transducer for low amplitude current (100A) measurement," *Electrical and Computer Engineering, 2003. IEEE CCECE 2003*, vol. I, pp. 463-466, 2003.
  - [32] dmercer, "Analog Devices University," Analog Devices, 09 Enero 2014. [Online]. Available: <https://wiki.analog.com/university/courses/electronics/text/chapter-11>. [Accessed 25 Noviembre 2014].
  - [33] Borovyi, Kochan, Sachenko, Konstantakos and Yaskilka, "Analysis of Circuits for Measurement of Energy of Processing Units," *Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications*, pp. 42-46, 2007.
  - [34] Konstantakos, Chatzigeorgiou, Nikolaidis and Laopoulos, "Energy Consumption Estimation in Embedded Systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 57, no. 4, pp. 797-804 , 2008.
  - [35] Konstantakos and Laopoulos, "A Power Measuring Technique for Built-in Test Purposes," *IEEE Instrumentation and Measurement Technology Conference Proceedings*, pp. 90-95, 2006.
  - [36] Konstantakos, Kosmatopoulos, Nikolaidis and Laopoulos, "Measurement of Power Consumption in Digital Systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 55, no. 5, pp. 1662-1670, 2006.
  - [37] E. Ramsden, *Hall-Effect Sensors Theory and Applications*, Newnes, 2006.
  - [38] Cirstea, Cernaianu and Gontean, "An inductive system for measuring microampere currents," *Design and Technology in Electronic Packaging (SIITME)*, pp. 197-200, 2012.
  - [39] Allegromicro, "Allegro MicroSystems," 2004. [Online]. Available: <http://www.allegromicro.com/en/Products/Current-Sensor-ICs.aspx>. [Accessed 18 Septiembre 2014].

- [40] Melexis, "Melexis," 2012. [Online]. Available: <https://www.melexis.com/en/products/sense/current-sensors>. [Accessed 18 Septiembre 2014].
- [41] Zhen, "Current Sensing Circuit Concepts and Fundamentals AN1332," 2011. [Online]. Available: [ww1.microchip.com/downloads/en/AppNotes/01332B.pdf](http://ww1.microchip.com/downloads/en/AppNotes/01332B.pdf). [Accessed 12 Octubre 2014].
- [42] M. Bazzaz, S. M and A. Ejlaei, "An Accurate Instruction-Level Energy Estimation Model and Tool for Embedded Systems," *IEEE Transactions on Instrumentation and Measurement*, vol. 62, no. 7, pp. 1927-1934, 2013.
- [43] Kim, "Performance comparison of low current measurement systems for biomedical applications," *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pp. 3469-3472, 2010.
- [44] Tektronix, "TEquipment," Tektronix, 2009. [Online]. Available: [http://www.tequipment.net/TektronixTCP312.html?gclid=Cj0KEQjwztG8BRCJgseTvZLctr8BEiQAA\\_kBD7Zf1VqE-Ms0oFcwfs5eEoKzdRe\\_F\\_BZVaW0T8agDbwaAiKM8P8HAQ#tab-reviews](http://www.tequipment.net/TektronixTCP312.html?gclid=Cj0KEQjwztG8BRCJgseTvZLctr8BEiQAA_kBD7Zf1VqE-Ms0oFcwfs5eEoKzdRe_F_BZVaW0T8agDbwaAiKM8P8HAQ#tab-reviews). [Accessed 12 Octubre 2014].
- [45] EEMBC, "EEMBC," EEMBC, 13 Abril 2014. [Online]. Available: [http://www.eembc.org/benchmark/power\\_sl.php](http://www.eembc.org/benchmark/power_sl.php). [Accessed 13 Octubre 2014].
- [46] CMicrotek, "CMicrotek," CMicrotek, 2010. [Online]. Available: <http://www.cmicrotek.com/uCP.htm>. [Accessed 12 Enero 2015].
- [47] W. Goh, "Texas Instruments," 7 Julio 2009. [Online]. Available: [www.ti.com](http://www.ti.com). [Accessed 28 Noviembre 2013].
- [48] K. Dudacek and V. Vavrcka, "Experimental Evaluation of the MSP430 Microcontroller Power Requirements," *EUROCON, 2007. The International Conference on "Computer as a Tool"*, pp. 400, 404, 2007.
- [49] K. Mikhaylov and J. Tervonen, "Evaluation of Power Efficiency for Digital Serial Interfaces of Microcontrollers," *New Technologies, Mobility and Security (NTMS)*, pp. 1, 5, 2012.



- [50] Microchip, "PIC24FV32KA302," 19 Marzo 2013. [Online]. Available: <http://www.microchip.com/wwwproducts/Devices.aspx?product=PIC24F32KA302>. [Accessed Mayo 2013].
- [51] Vishay/Dale, "Mouser," [Online]. Available: <http://www.mouser.com/ds/2/427/cmfind-239942.pdf>. [Accessed 15 Enero 2014].
- [52] Rigolna, "DS1102D 100 MHz Mixed Signal Oscilloscope," [Online]. Available: <http://beyondmeasure.rigoltech.com/acton/attachment/1579/f-004b/0/-/-/-/file.pdf>. [Accessed 12 Septiembre 2014].
- [53] Instek, "GPS-x303 Series," [Online]. Available: [http://www.gwinstek.com/en-global/products/DC\\_Power\\_Supply/Multiple\\_Channel\\_DC\\_Power\\_Supplies/GPS-x303](http://www.gwinstek.com/en-global/products/DC_Power_Supply/Multiple_Channel_DC_Power_Supplies/GPS-x303). [Accessed 11 Octubre 2013].
- [54] Y. K. T. J. Song, "Energy consumption analysis of ZigBee-based energy harvesting wireless sensor networks," *IEEE International Conference on Communication Systems (ICCS)*, pp. 468-472, 2012.

**Anexo 1.-**  
**Correo de Microchip por problema de hardware**

## Microchip Support Community

- [Close Window](#)
- [Print This Page](#)

Case: 00037245

Case Number	00037245	IDE Name
Case Reason		IDE Version
Customer Subject	16-bit MCUs & Digital Signal Controllers - Configuration Bits	Compiler
Date/Time Opened	4/25/2014 5:33 PM	Compiler Version
Status	Closed - Resolved	Compiler Edition
Urgency		HPA Activation Key
Target Device	PIC24F32KA302	Stack / Library Name / App Note
Peripheral		Programmer / Debugger

## Description Information

Description Hello.

I use Custom Computer Services PCD compiler for these micros and MPLAB X 2.05 for IDE. I use PICKIT3 with firmware 1.30.

I've used these compiler for 6 years now and have been using it with several PIC24FV32KA302 since December last year without any problems.

I made a bad programming sequence and ended up with a wrong internal oscillator configuration (tried to use the 500 KHz LPFRC) which in turn ended up with some odd behavior that placed the microcontroller with all it's programming views protected.

After that I tried to program and erase the microcontrollers with no luck. The PICKIT3 can only read the device and that's how I assume it's protected, by looking at the configuration bits in MPLAB X after the read.

Here are the configuration bits that I can read:

// PIC24FV32KA302 Configuration Bit Settings

// 'C' source line config statements

#include <htc.h>

```
__CONFIG(BWRP_ON & BSS_H15K);
__CONFIG(GWRP_ON & GSS0_ON);
__CONFIG(FNOSC_FRC & SOSCSRC_DIG & LPRCSEL_LP & IESO_OFF);
__CONFIG(POSCMOD_EC & OSCIOFNC_OFF & // POSCFREQ = No Setting
& SOSCSEL_SOSCCLP & FCKSM_CSECME);
__CONFIG(WDTPS_PS1 & FWPSA_PR32 & FWDTEN_OFF & WINDIS_ON);
__CONFIG(BOREN_BOR0 & LVRCFG_ON & PWRTEN_OFF & I2C1SEL_SEC & BORV_LPBOR & MCLR_OFF);
__CONFIG(// ICS = No Setting
);
__CONFIG(DSWDTPS_DSWDTPS0 & DSWDTPS0_SOSC & DSBORON_OFF & DSWDTEN_OFF);
```

I had similar issues with other PIC18 microcontrollers and was able to erase them if something went wrong by simply modifying the configuration bits directly in the old MPLAB

## Additional Information

Experts	Categories
Design Stage	Application Details

**Customer Call-in Info****Phone Access Number** S3\_Migrated**Customer Can Call in From:**

**Note** All times shown are based on your Time Zone settings. Visit 'My Settings -> Location Settings' to update. (Page refresh may be required.)  
View our Support Phone Numbers Here

**Resolution****Proposed Resolution** Hi Mario,

This ticket is marked as resolved for now. The resolution to this ticket is:

ICSP connections were reviewed for the PIC:

For advanced ICSP connections (specially for Low-Pin count devices) you may want to see as reference:

Alternate ICSP Implementation Configuration of Multi-Tool Design Advisory:  
<http://ww1.microchip.com/downloads/en/DeviceDoc/51764C.pdf>

For basic configuration of your ICSP lines. Please see as reference:  
<http://ww1.microchip.com/downloads/en/DeviceDoc/52010A.pdf>

Please also take note of the pull-up resistor values on your /MCLR line. This value should be in the range of 4.7K - 10K or as indicated by your device datasheet

Bulk Erase programming has been made to the PICS that cannot be programmed to erase all form of Code protection.

Also bulk erase programming voltage was ensured.

MCLR\_ON is enabled on the configuration bits in order for the programmer to have full control of the reset state of the PIC.

As customer still continues to have problems on programming the PICS it is recommended as precaution to future design to place series resistors on the ICSP lines in the range at or below 100 ohms. This can prevent ESD damage of the ICSP. As reference please see:

2.5 ICSP Pins of your device datasheet:  
<http://ww1.microchip.com/downloads/en/DeviceDoc/39995d.pdf>

You can place these resistors on your /MCLR, PGC and PGD lines near your ICSP connector before these lines reaches your PIC.

This would prevent your ICSP pins from experiencing ESD events and also prevent excessive overshoot.

Best Regards,

Jonathan

**Web Email** mario-camarillo@micro-bios.com**Web Company** MicroBios**Web Name** Mario Camarillo**Web Phone****Case Comments****5/26/2014 1:30 AM**

**User** Jonathan Rodriguez  
**Hi Mario,**

**Comment** This ticket is marked as resolved for now. The resolution to this ticket is:

**5/21/2014 5:46 PM**

**User** Jonathan Rodriguez  
**Hi Mario,**

**Comment** If this continues then I recommend that you use place

ICSP connections were reviewed for the PIC:

For advanced ICSP connections (specially for Low-Pin count devices) you may want to see as reference:

Alternate ICSP Implementation Configuration of Multi-Tool Design Advisory:  
<http://ww1.microchip.com/downloads/en/DeviceDoc/51764C.pdf>

For basic configuration of your ICSP lines. Please see as reference:  
<http://ww1.microchip.com/downloads/en/DeviceDoc/52010A.pdf>

Please also take note of the pull-up resistor values on your /MCLR line. This value should be in the range of 4.7K - 10K or as indicated by your device datasheet

Bulk Erase programming has been made to the PICS that cannot be programmed to erase all form of Code protection.

Also bulk erase programming voltage was ensured.

MCLR\_ON is enabled on the configuration bits in order for the programmer to have full control of the reset state of the PIC.

As customer still continues to have problems on programming the PICS it is recommended as precaution to future design to place series resistors on the ICSP lines in the range at or below 100 ohms. This can prevent ESD damage of the ICSP. As reference please see:

2.5 ICSP Pins of your device datasheet:  
<http://ww1.microchip.com/downloads/en/DeviceDoc/39995d.pdf>

You can place these resistors on your /MCLR, PGC and PGD lines near your ICSP connector before these lines reaches your PIC.

This would prevent your ICSP pins from experiencing ESD events and also prevent excessive overshoot.

Best Regards,

Jonathan

series resistors on your ICSP lines in the range at or below 100 ohms. As reference please see:

2.5 ICSP Pins of your device datasheet:  
<http://ww1.microchip.com/downloads/en/DeviceDoc/39995d.pdf>

You can place these resistors on your /MCLR, PGC and PGD lines near your ICSP connector before these lines reaches your PIC.

This would prevent your ICSP pins from experiencing ESD events and also prevent excessive overshoot.

Hope this would prevent issues on your PICS.

Best Regards,

Jonathan

5/11/2014 9:53 PM

User Jonathan Rodriguez

Hi MARIO,

Your device PIC24FV32KA302 would have an internal voltage regulator output and a VCAP would be needed. You cannot select an ordinary capacitor. Please see:

2.4 Voltage Regulator Pin (VCAP) of device datasheet:  
<http://ww1.microchip.com/downloads/en/DeviceDoc/39995d.pdf>

5/16/2014 10:02 PM

User Mario Camarillo

Hello Jonathan.

I figured that much about the capacitor. I searched for the capacitors suggested in the data sheet and some are obsolete (you should check them out too).

Bought two capacitors from Mouser. One is a 10v with an ESR of 0.22 Ohms and the other is a 16v with an ESR of 3.2 Ohms:

<http://www.mouser.com/Search/ProductDetail.aspx?R=TAP106K016SCSvirtualkey58110000virtualkey581-TAP106K016SCS>

Comment <http://www.mouser.com/Search/ProductDetail.aspx?R=10SVP10Mvirtualkey66720000virtualkey667-10SVP10M>

Did a test with all the devices but it did not work, I couldn't erase the chips.

Did an erase sequence 20 times and 18 of them MPLAB IPE showed no that the device ID was 00 and not the device under test. The last two times MPLAB IPE told me that the pickit 3 was missing a memory object. That was new, never did MPLAB X or IPE showed me that message.

Comment

A bulk-erase of the PIC(ERASE and a check for Blank Check) would "always" erase the code protect configuration bits as long as you have the correct supply voltage for bulk-erase programming.

3.3.3 CODE-PROTECT ICSP ENTRY of device Programming Specs:  
<http://ww1.microchip.com/downloads/en/DeviceDoc/39919b.pdf>

Best Regards,

Jonathan

<p>Just to let you know, I'm using the INTEK power supply with the fixed 3amp@5v output to program the device (not the pickit3).</p> <p>This is very strange since I'm using more PIC24FV32KA302 and with them I can erase, program, all the other options, I just replace them like in the video.</p> <p>I might be doing something wrong but how can the other devices do well with the same conditions, and the faulty ones do not.</p> <p>Sorry for the very late reply (been busy).</p>		<p><b>5/4/2014 4:19 AM</b></p> <p>User Jonathan Rodriguez</p> <p>Attachment added: 9165IPE.docx - IPE bulk erase</p> <p>File Link: <a href="http://www.microchip.com/support/FileHandler.aspx?file=79884&amp;ticketNo=270265&amp;ticketID=246785">http://www.microchip.com/support/FileHandler.aspx?file=79884&amp;ticketNo=270265&amp;ticketID=246785</a></p>
<p><b>5/5/2014 10:01 PM</b></p> <p>User Mario Camarillo</p> <p>Hello.</p> <p>I have a bypass capacitor in VDD and VSS, just can't see it in the video.</p> <p>I don't have a capacitor placed since I will not use the onboard regulator for Vout. I don't have these special caps but have ordered some from mouser. Can I use a simple 10uF one?</p> <p>I've tried the IPE solution and it did work in the good chips but not in the bad ones, as if the bad ones can't communicate. I've read this document, section 4.7.3 CODE-PROTECT CONFIGURATION BITS: <a href="http://ww1.microchip.com/downloads/en/DeviceDoc/39919b.pdf">http://ww1.microchip.com/downloads/en/DeviceDoc/39919b.pdf</a> and that's what I need, erase them. Will post another video for you to see that if I swap the chips, the good ones will work and the bad ones won't.</p> <p>Will wait for that capacitor but that's strange, since I can do everything (program, erase, verify) with the good ones but can't with the bad ones.</p> <p>The information you requested: 1230U33 1230U33 1249YMS 1249YMS 1249YMS 1349W43 1349W43</p> <p>Thanks.</p> <p>I misread this document with respect to VDD: <a href="http://ww1.microchip.com/downloads/en/DeviceDoc/51764C.pdf">http://ww1.microchip.com/downloads/en/DeviceDoc/51764C.pdf</a></p> <p>Additionally, the MCLR /V PP signal is used by the development tool to provide the voltage used for programming some devices or to signal attention. In instances where the application has a large capacitor, it will cause the signal rise and fall time to degrade. This will hinder the ability of the tool and the device to communicate effectively.</p> <p>&lt;div data-canvas-width="378.3953486955643" data-font-name="He</p>		<p><b>5/4/2014 4:10 AM</b></p> <p>User Jonathan Rodriguez</p> <p>Hi Mario,</p> <p>Do you mean you are using a supply of 5.3V for VDD? And your /MCLR line is directly connected to pin 1 of your PICKIT3 and PIC?</p> <p>I would just like to clarify that VPP is a parameter for /MCLR line and is not a pin that should be applied external voltage as this comes to the PICKIT3 itself (just verifying).</p> <p>Have you verified that you haven't exchanged your PGD and PDC lines? Also I noticed you were not installing your VCAP capacitor (low ESR)</p> <p>Could I know where did you get the specs of 5.3V?</p> <p>Are you also doing some kind of sleep mode on your program?</p> <p>Can you just try this sample code that I have:</p> <p>Comment</p> <pre>#include &lt;xc.h&gt;  _FBS(BWRP_ON &amp; BSS_HI5K) _FGS(GWRP_ON &amp; GSS0_ON) _FOSSEL(FNOSC_FRC &amp; SOSC SRC_DIG &amp; LPRCSEL_LP &amp; IESO_OFF) _FOSC(POSCMOD_EC &amp; OSCIOFNC_OFF &amp; SOSCSEL_SOSCLP &amp; FCKSM_CSECME) _FWDT(WDTPS_PS1 &amp; FWPSA_PR32 &amp; FWDTEN_OFF &amp; WINDIS_ON) _FPOR(BOREN_BOR0 &amp; LVRCFG_ON &amp; PWRRTEN_OFF &amp; I2C1SEL_SEC &amp; BORV_LPBOR &amp; MCLRE_OFF) // __CONFIG(// ICS = No Setting //); _FDS(DSWDTPS_DSWDTPS0 &amp; DSWDTOSC_SOSC &amp; DSBOREN_OFF &amp; DSWDTEN_OFF)  int main(void)</pre>
<p><b>5/4/2014 4:15 AM</b></p> <p>User Jonathan Rodriguez</p> <p>Comment I'd also like to follow up to put the 0.1uF bypass capacitors across your VDD and VSS pins to make it complete.</p>		
<p><b>5/4/2014 1:17 AM</b></p> <p>User Mario Camarillo</p>		

	<p>Hi.</p> <p>Tried your suggestion and I might be doing something wrong.</p> <p>I've uploaded a video. If you can see what I'm doing maybe you can spot the error.</p> <p>Thanks for your advice.</p> <p><a href="https://www.youtube.com/watch?v=TCtp29YSbl&amp;feature=youtu.be">https://www.youtube.com/watch?v=TCtp29YSbl&amp;feature=youtu.be</a></p>	<pre>{ }</pre> <p>As reference could you also give the 7-digit alphanumeric code on the lower portion of the PICS?</p>
	<p><b>5/2/2014 4:50 PM</b></p> <p>User Jonathan Rodriguez</p> <p>Hi Mario,</p> <p>Thanks you for verifying this. I have replicated and verified your issue. Please use high voltage programming on your /MCLR line. This should work. On your MPLAB-X, right click on your project----&gt;Properties-----&gt;Option categories(drop-down)----&gt;Program Options-----&gt; Use High Voltage on MCLR(check box). Please check this check box.</p> <p>Best Regards,</p> <p>Jonathan</p>	<p>If this still fails, then we have a software for bulk erasing a device that can erase the config</p>
	<p><b>5/2/2014 3:18 AM</b></p> <p>User Jonathan Rodriguez</p> <p>Hi Mario,</p> <p>Also as follow-up, have you tried to program this good chips "twice" with the error making configuration bits? This is where it might happen.</p> <p>Jonathan</p>	<p><b>5/2/2014 5:58 PM</b></p> <p>User Jonathan Rodriguez</p> <p>Hi Mario,</p> <p>Also please ensure you have a VDD supply of at least 2.7V on your VDD since you have turned on code protection in your configuration bits. As reference, please see:</p> <p>3.3.3 CODE-PROTECT ICSP ENTRY of device Programming Specs:  <a href="http://ww1.microchip.com/downloads/en/DeviceDoc/39919b.pdf">http://ww1.microchip.com/downloads/en/DeviceDoc/39919b.pdf</a></p> <p>Awork around could be:  Once you have entered High Voltage Programming mode on your /MCLR line, you may now set your config bits to MCLRE_ON. After doing this, you may now be able to program your device in LVP or Low-Voltage Programming mode (normal mode) always.</p> <p>There are timing and voltage differences between Low voltage and High Voltage Programming as reference please see:</p> <p>3.3.1 LOW-VOLTAGE ICSP ENTRY and 3.3.2 HIGH-VOLTAGE ICSP ENTRY of device Programming Specs:  <a href="http://ww1.microchip.com/downloads/en/DeviceDoc/39919b.pdf">http://ww1.microchip.com/downloads/en/DeviceDoc/39919b.pdf</a></p> <p>Also for advanced ICSP connections (specially for Low-Pin count devices) you may want to see as reference:</p> <p>Alternate ICSP Implementation Configuration of Multi-Tool Design Advisory:  <a href="http://ww1.microchip.com/downloads/en/DeviceDoc/51764C.pdf">http://ww1.microchip.com/downloads/en/DeviceDoc/51764C.pdf</a></p>
	<p><b>4/29/2014 3:33 AM</b></p> <p>User Jonathan Rodriguez</p> <p>Hi Mario,</p> <p>A possible reason is that if your /MCLR is OFF then your programmer may not have full control of this line while your PGD and PGC pins might be used in your application as possible I/O pins. This may have an effect in programming specially because the /MCLR, PGD, and PGC lines are timing critical with each other. I suggest that during debugging mode of your application you first set the /MCLR line to ON and in your final build, you may set this OFF to use as an I/O port. You can always though program your PIC back to /MCLR ON so as you may not be stuck in programming your PICS.</p> <p>As references:</p> <p>FIGURE 3-4: ENTERING ICSP™; MODE USING LOW-VOLTAGE ENTRY  FIGURE 3-5: ENTERING ICSP™; MODE USING HIGH-VOLTAGE ENTRY</p> <p>of Device Programming Specs:  <a href="http://ww1.microchip.com/downloads/en/DeviceDoc/39919b.pdf">http://ww1.microchip.com/downloads/en/DeviceDoc/39919b.pdf</a></p> <p>Your /MCLR line is being used as Reset and as a Programming Pin reference:</p> <p>2.3 Master Clear (MCLR) Pin of device datasheet  <a href="http://ww1.microchip.com/downloads/en/DeviceDoc/39995d.pdf">http://ww1.microchip.com/downloads/en/DeviceDoc/39995d.pdf</a></p> <p>Would this solve your issue?</p>	<p>Comment</p>

Best Regards,  
Jonathan

Also for basic configuration of your ICSP lines. Please see as reference:

<http://ww1.microchip.com/downloads/en/DeviceDoc/52010A.pdf>

Please also take note of the pull-up resistor values on your /MCLR line. This value should be in the range of 4.7K - 10K or as indicated by your device datasheet

2.3 Master Clear (MCLR) Pin of your device datasheet  
<http://ww1.microchip.com/downloads/en/DeviceDoc/39995d.pdf>

Would this solve your Programming Problem?

Best Regards,  
Jonathan

5/2/2014 12:49 PM

User Mario Camarillo  
Hello Jonathan.

I just did what you suggest. The first time it programmed the chip, but the next time (second time) it did not recognize it. Now I have more faulty chips.

Comment It's strange since I can program them once but then I can't get the programmer to erase them. I have limited knowledge about the security features but it's my understanding that if you activate security (which it's happening for some reason) you can't reprogram but you can erase them, maybe I'm wrong.

The issue here is that I can't even erase them, and that's odd.

Hopefully this issue can be resolved, I have now 7 chips that are behave in the same manner.

5/2/2014 12:20 AM

User Mario Camarillo  
Hello Jonathan.

Comment Tried your recommendations and I might be doing something



wrong:

I got one new (blank)  
PIC24FV32KA302 from a friend  
and made the adjustments you  
told me.

Here's the new configuration bits  
that MPLAB X (2.05) built:

// PIC24FV32KA302 Configuration  
Bit Settings

// 'C' source line config  
statements

#include <htc.h>

```
__CONFIG(BWRP_OFF &
BSS_OFF);
__CONFIG(GWRP_OFF &
GSS0_OFF);
__CONFIG(FNOSC_FRCPLL &
SOSCSRC_DIG & LPRCSEL_HP &
IESO_OFF);
__CONFIG(POSCMOD_NONE &
OSCIOFNC_ON & POSCFREQ_HS
& SOSCSSEL_SOSCHP &
FCKSM_CSDCMD);
__CONFIG(WDTPS_PS2 &
FWPSA_PR32 & FWDTEN_ON &
WINDIS_OFF);
__CONFIG(BOREN_BOR0 &
LVRCFG_OFF & PWRTEN_OFF &
I2C1SEL_PRI & BORV_V20 &
MCLRE_ON);
__CONFIG(ICS_PGx1);
__CONFIG(DSWDTPS_DSWDTPSF
& DSWDTOSC_LPRC &
DSBOREN_OFF &
DSWDTEN_OFF);
```

And this is what I got when I  
programmed it:

Connecting to MPLAB PICkit 3...  
Firmware Suite  
Version.....01.30.09  
Firmware  
type.....dsPIC33F/24F/24H

Target detected  
Device ID Revision = 4  
<div s

4/29/2014 12:58 AM

User Jonathan Rodriguez


Hi Mario,


For a possible quick solution  
Comment please turn your MCLRE\_ON in  
your config bits.

Jonathan

## **Anexo 2**

### **Correo de CCS por problema de software**

**Custom Computer Services**

**CCS Software Maintenance & Upgrades Offers**  
Enter your customer number and receive special offers on maintenance, compiler add-ons and development kit upgrades.  
Visit [www.ccsinfo.com/fsmo](http://www.ccsinfo.com/fsmo) for more details

[FAQ](#) [Forum Help](#) [Official CCS Support](#) [Search](#)

[Profile](#) [You have no new messages](#) [Log out \[ mcr1981 \]](#)

CCS does not monitor this forum on a regular basis.  
Please do not post bug Reports on this forum. Send them to [support@ccsinfo.com](mailto:support@ccsinfo.com)

## A little advice on what not to do with internal oscillator

[new topic](#) [post reply](#) [CCS Forum Index -> General CCS C Discussion](#)

[View previous topic :: View next topic](#)

Author	Message
<b>mcr1981</b>  Joined: 27 Oct 2010 Posts: 28  <a href="#">profile</a> <a href="#">pm</a>	<b>A little advice on what not to do with internal oscillator</b> Posted: Thu Apr 24, 2014 6:39 pm  Hello,  Just telling you a little advice on what not to do with the internal oscillator of a PIC24FV32KA302:  do not use the preprocessor #use delay(internal = 31250), it will brick your micro.  At least it's what I think happen to three of my micros. Could program them with 32MHz internal and all the way down to 1MHz, but when I tried to use 31250, they are not recognized by the PICKIT3.  That's so strange since I can't even erase them. I've been looking around the microchip forum and the web for a possible solution and apparently when you use the Internal Oscillator with a frequency not known to the microcontroller and don't use the MCLR pin as a MCLR (normal digital instead) you get yourself a nice little brick.  Here's the sample code I use:  <div><b>Code:</b><pre>#include    &lt;24FV32KA302.h&gt;  #DEVICE ADC = 12 #FUSES ICSP1 #FUSES NOWRTB #FUSES NOBSS #FUSES NOWRT #FUSES NOPROTECT #FUSES NOWD7 #FUSES NOWSDOR #FUSES NOWLVB #FUSES NOWEUC #FUSES NOBROWNOUT #FUSES NOIESO #FUSES SOSCDIGITAL #FUSES NOOSCIO #FUSES NOMCLR #use delay(internal=31250)</pre></div>

**Code:**

```
void main(void)
{
    while(1)
    {
        output_low(PIN_B5);
        delay_ms(1000);
        output_high(PIN_B5);
        delay_ms(1000);
    }
}
```

I use PCD 5.023 and MPLAB X 2.05

Actually, PICKIT3 can read the device (although it tells me it does not recognize it) and the configuration bits are as follows:

**Code:**

```
// PIC24FV32KA302 Configuration Bit Settings
// 'C' source line config statements
#include <htc.h>

__CONFIG(BWRP_ON & BSS_HI5K);
__CONFIG(GWRP_ON & GSS0_ON);
__CONFIG(FNOSC_FRC & SOSCSRC_DIG & LPRCSEL_LP & IESO_OFF);
__CONFIG(POSCMOD_EC & OSCIOFNC_OFF & // POSCFREQ = No Setting
& SOSCSSEL_SOSCCLP & FCKSM_CSECME);
__CONFIG(WDTPS_PS1 & FWPSA_PR32 & FWDTEN_OFF & WINDIS_ON);
__CONFIG(BOREN_BOR0 & LVRCFG_ON & PWRTEN_OFF & I2C1SEL_SEC & BORV_LPBOR &
MCLRE_OFF);
__CONFIG(// ICS = No Setting
);
__CONFIG(DSWDTFS_DSWDTFS0 & DSWDTOSC_SOSC & DSBORN_OFF & DSWDTEN_OFF);
```

One can import the configuration bits from MPLAB X.

That's very strange since all the PIC is protected.

The only thing I did was to change the #use delay(internal=32MHz) to #use delay(internal=31250).

Hope this is useful for someone, don't fiddle with the wrong Oscillator Frequency and the MCLR configuration bit.

Will still try to program those micros, but don't know if I'll succeed. Will tell you if I can.

**asmboy**

Posted: Thu Apr 24, 2014 7:23 pm



Joined: 20 Nov 2007  
Posts: 2057  
Location: albania ny



bad fuse being written?

what does the end of the offending .LST file show ??

tried a readback of PIC program and fuses area ??  
i've done that with PICs,

**jeremiah**

Joined: 20 Jul 2010  
Posts: 818



using the Micro engineering labs USB pic programmers  
when situations like this arise, comparing fuses as written  
vs what i wanted . 😊

Posted: Thu Apr 24, 2014 9:18 pm



Your fuses would default to the FRC, which is much higher frequency than the LPRC. You should really set your fuses to reflect your LPRC. There are two to set:

#fuses NOPR //tells the pic to not use an external clock circuit  
#fuses LPRC //tells the pic that the clock select mux should select the LPRC

EDIT: also note, these chips (both the 304 and the 302) sometimes have trouble running at frequencies near or above 25MHz. The datasheet says they can go up to 25MHz, but I have had trouble in the past with glitching at those speeds. Simple hello world programs would fail sometimes until I dropped the speed down.

I've never tried using the LPRC as the system clock (I use it for timer1 wakeup from sleep a lot of times) nor do I have a picket3 to test with. I typically go with the FRC or a 3-11Mhz external crystal. Even with those, the PIC24F32KA30x will drop to nano amps in regular sleep, so I haven't needed to run off the LPRC. Also note that the LPRC has a 15% tolerance, so you have to be careful what you run on your PIC.

**mcr1981**

Joined: 27 Oct 2010  
Posts: 28



**Thanks for the replies but....**

Posted: Fri Apr 25, 2014 1:27 am



Thanks for the replies.....but:

I tried reading the PIC and it returns just NOP in every line (in HEX they are all 0x00).

When I read the configuration bits it always returns code protection to the max (can't read or write).

Can't erase them. Not even with a Xeltek SUPERPRO 5000E (same error, can't read the device).

Even tried with MPLAB 8.92 (trusty MPLAB). I remember several times I just went into the configuration bits area and forced MPLAB to program new configuration words.

To Jeremiah:

That's odd, the high frequency issue. I've been using these micros with some PWM, RS232, LCD and other stuff at 32MHz (8M FRC+PLL) and haven't had any problems.... well, having problems now at low frequencies.

Will order new PICs and see what happens. Will not give up on those three little bricks that I now possess. 😊

12:26 AM now.... need to sleep.

**temtronic**

Joined: 01 Jul 2010  
Posts: 4728  
Location: Greenville, Ontario



Posted: Fri Apr 25, 2014 9:00 am



sounds like you should contact Microchip and ask for help...at least they should replace the 'bricks' for free....

jay

**mcr1981**

Joined: 27 Oct 2010  
Posts: 28

[profile](#) [pm](#)

**Contacted Microchip now.**

Posted: Fri Apr 25, 2014 12:26 pm

[quote](#) [edit](#)

**temtronic wrote:**

sounds like you should contact Microchip and ask for help...at least they should replace the 'bricks' for free....

jay

That's news to me. Hopefully they will resolve this issue.

Will contact them and wait for a solution.

It's interesting that the Low Power RC configuration bits don't change. Looks like it's always working with the nominal internal frequency (8MHz).

EDIT:

Ticket has been created, now to wait.

270265 04/25/2014 05:33 PM

**Ttelmah**

Joined: 11 Mar 2010  
Posts: 10341

[profile](#) [pm](#)

Posted: Sat Apr 26, 2014 1:18 am

[quote](#)

I remember before, finding on these chips that the 'internal=' usage, does not switch to the LP oscillator. You have to explicitly select this in the fuses. So LPRC.

Just compiled with this selected, and imported to MPLAB, and the FNOSC fuses then change from the 8MHz with postscaler, to the LPRC oscillator.

On your ability to read the fuses, this sounds suspiciously like the old MPLAB switching to DEBUG mode problem. So several of the actual fuses are being overridden, including the ones that prevent the fuses from being read.

**mcr1981**

Joined: 27 Oct 2010  
Posts: 28

[profile](#) [pm](#)

**Update....**

Posted: Sun Dec 28, 2014 5:28 pm

[quote](#) [edit](#)

I have not tried this code with the faulty oscillator since the last time.

I'm surprised to see that the new version (5.035) does provide an error if one tries to assign an invalid internal oscillator with MPLAB X 2.26

CCS with the help of the ICD-U64 and tech support helped me a lot too.

Microchip reproduced my setup and they too bricked their chips.

**temtronic**

Joined: 01 Jul 2010  
Posts: 4728  
Location: Greenville, Ontario

[profile](#) [pm](#)

Posted: Mon Dec 29, 2014 2:24 pm

[quote](#)

hmm.. the fact that uC bricked their own devices says to me, they should send you new ones !!

It should NOT be possible to do. There should always be a way to 'reset' the PIC to a 'factory configuration'.

jay

**mcr1981**

**I hear you...**

Posted: Mon Dec 29, 2014 2:47 pm

[quote](#) [edit](#)

Joined: 27 Oct 2010  
Posts: 28

[profile](#) [pm](#)

Already told them that back in April 2014. Well, not that exactly like that.

Those chips were revision 4. Got new ones and now they are revision 6.

I'm just happy it works now and the CCS guys now have implemented that feature for the internal oscillator calculation.

If there's a way to destroy something, someone will find it.... I always say that, now I'm part of my own statistics 😊

EDIT:  
was not April 2013, it's 2014.

Last edited by mcr1981 on Tue Dec 30, 2014 12:45 am; edited 2 times in total

temtronic

Joined: 01 Jul 2010  
Posts: 4728  
Location: Greenville, Ontario

[profile](#) [pm](#)

Posted: Mon Dec 29, 2014 5:07 pm [quote](#)

hmm.. when I told them 'something' was wrong with my PicstartPlus (failed to update the new PIC internals), they sent me a new one, next day from Toronto, kinda suprised they didn't send you a 'tube' full of new PICs. Glad to hear you're 'up and running' though !!

jay

PCM programmer

Joined: 06 Sep 2003  
Posts: 19387

[profile](#) [pm](#)

Posted: Mon Dec 29, 2014 5:13 pm [quote](#)

It's possible you could get a limited number of replacements as samples.  
<http://www.microchip.com/samples/Default.aspx?DeviceFamily=PIC24F32KA302>

Display posts from previous: All Posts Oldest First Go

[new topic](#) [post reply](#)

CCS Forum Index -> General CCS C Discussion

All times are GMT - 6 Hours

Page 1 of 1

[Stop watching this topic](#)

Jump to: General CCS C Discussion Go

You can post new topics in this forum  
You can reply to topics in this forum  
You can edit your posts in this forum  
You can delete your posts in this forum  
You can vote in polls in this forum

Powered by phpBB © 2001, 2005 phpBB Group

5 of 5

09/10/2016 02:41 p.m.

84