Artículo **Revista de Innovación Sistemática**
Junio 2017 Vol.1 No.2 19-30

Diseño de un registrador de datos usando memorias nand flash basado en un microcontrolador de ultrabajo consumo

DE LEÓN-GORDILLO, Dagoberto*†, MEDINA-RODRÍGUEZ, Cristian Micheel, RODRÍGUEZ-OLIVARES, Noé Amir' y SOTO-CAJIGA, Jorge Alberto

Centro de Ingeniería y Desarrollo Industrial (CIDESI), Av. Playa Pie de la Cuesta No. 702, Col. Desarrollo San Pablo, Querétaro, Qro. México
'Universidad Tecnológica de Querétaro (UTEQ), Av. Pie de la Cuesta 2501, Col. Unidad Nacional, Querétaro, Qro. México

Recibido 6 de Abril, 2017; Aceptado 9 de Junio, 2017

Resumen

Las memorias NAND Flash son uno de los sistemas de almacenamiento preferido en ingeniería, esto debido a su alta velocidad de escritura, bajo consumo de energía y alta densidad de almacenamiento. En este artículo presentamos la mejora de un registrador de datos o datalogger, el cual inicialmente estaba compuesto por una memoria NAND Flash y dos microcontroladores de bajo consumo en una arquitectura multimaestra. La mejora del dataloggeer se basa en explotar el paralelismo que permiten los microcontroladores con módulos de acceso directo a la memoria (DMA). El datalogger se comunica mediante el protocolo UART a 460800 baudios y almacena en una memoria NAND Flash mediante el protocolo ONFI 2.0. Adicionalmente, se ha agregado una función de codificación tipo Hamming (255,247), útil para corregir 64 bits erróneos cada 2040 bytes. Se ha caracterizado el tiempo que tarda esta función en corregir desde uno hasta 64 errores lo que permitiría predecir la degradación de la memoria. Se comparó experientalmente el datalogger mejorado con el anterior, y tiene una velocidad de almacenamiento de 47 KB/s con capacidad de almacenamiento desde 256 MB hasta 8 GB usando solo un microcontrolador de ultrabajo consumo.

Datalogger, ECC Hamming, NAND Flash, DMA

Abstract

NAND Flash chips are one of the preferred storage devices in engineering, due to their high writting speed, low power consumption and high storage density. In this paper, we present the improvement of a datalogger, which initially was composed of one NAND Flash memory and two ultra low power microcontrollers in a multi-master architecture. The improvement of the dataloggeer is based on exploiting the parallelism that allows the microcontrollers with modules of direct memory access (DMA). The datalogger communicates via the UART protocol to 460800 bauds and stores in a NAND Flash memory using the ONFI 2.0 protocol. Additionally, a Hamming coding (255,247) function has been added, which is useful for correcting 64 erroneous bits per 2040 bytes. We characterized the correcting-time function from 1 up to 64 errors. This function allows the prediction of memory chip degradation. The improved datalogger was compared to the previous one, and it has a storage speed of 47 KB/s with storage capacity from 256 MB to 8 GB using only one ultra low power microcontroller.

Datalogger, ECC Hamming, NAND Flash, DMA

Citación: DE LEÓN-GORDILLO, Dagoberto, MEDINA-RODRÍGUEZ, Cristian Micheel, RODRÍGUEZ-OLIVARES, Noé Amir' y SOTO-CAJIGA, Jorge Alberto. Diseño de un registrador de datos usando memorias nand flash basado en un microcontrolador de ultrabajo consumo. Revista de Innovación Sistemática 2017. 1-2:19-30

*Correspondencia al Autor (Correo Electrónico: leon@posgrado.cidesi.edu.mx)
† Investigador contribuyendo como primer autor.

Introducción

Un tipo de memoria no volátil y que es muy utilizada en varias aplicaciones es la memoria NAND Flash, las memorias NAND Flash se pueden encontrar en cámaras digitales, aplicaciones móviles, sistemas de almacenamiento no volátil, computadoras, etc. (Micheloni, Crippa, & Marelli, 2010) (Toshiba America Electronic Components, INC) (Bez, Camerlenghi, Modell, & Visconti, 2003) (Pavan, Bez, Olivo, & Zanoni, 1997) (C. Park, 2003).

Un dispositivo muy utilizado en ingeniería para almacenar datos provenientes de sensores es el registrador de datos o datalogger. Los componentes básicos de un datalogger son: una unidad de almacenamiento masivo y una unidad de control, comúnmente un microcontrolador.

Se hizo un estudio de varios dataloggers y se encontraron algunos con aplicaciones específicas y otros de propósito general. Algunos dataloggers con aplicaciones específicas consideran en su arquitectura sensores apropiados para el monitoreo de variables de medición, un ejemplo de ello es el dataologger de Nisha y Umesh (2015) el cual contiene sensores de temperatura, humedad, presión, etc. Otra ventaja presentada en dataloggers es que permiten adecuarse a tiempos de muestreo como es el caso del datalogger de Mahzan et al., (2013) con múltiples tarjetas SD o el caso del datalogger de Rajesh et al. (2003) utilizado en aplicaciones de detección fisiológica. Otro datalogger estudiado para aplicaciones de GPS almacena los datos en formato KML el cual permite leerlo mediante la aplicación Google Earth (Ibrahim, 2010).

De los dataloggers encontrados de uso general, algunos han sido utilizados en aplicaciones de biomecánica, tal es el caso del trabajo realizado por Kobsar et al. (2014) el cual utiliza un datalogger logomatic v2 de sparkfun con tarjetas SD.

Algunos ingenieros prefieren utilizar memorias de menor capacidad y a la vez más económicas que dan soporte para aplicaciones de bajo almacenamiento o en las que no se necesita de tiempos de muestreo muy elevados, tal es el caso del datalogger realizado por Rivera (2010) en el que consiste de un PIC 18F2550, una memoria EEPROM 24LC512 y una interfaz para comunicación, la memoria es de 64 KB y el tiempo de muestreo mínimo que lo controla Labview® es de 1 segundo.

El datalogger realizado por Febrian et al. (2016) permite el intercambio de información con máquinas remotas y también puede ser controlado por otros dispositivos, el cual es una ventaja si se considera el uso compartido de datos.

El datalogger realizado por Medina Rodríguez, et al. (2016) es de uso general, este datalogger utiliza una memoria NAND Flash marca Hynix® y dos microcontroladores MSP430 de Texas Instruments® en una estructura de tipo multimaestra. Para garantizar el flujo continuo de datos la arquitectura utiliza el paralelismo de operaciones: mientras el microcontrolador A adquiere y almacena en RAM, el microcontrolador B transmite lo almacenado en RAM a una memoria NAND Flash, el microcontrolador que termina primero su tarea entra a modo de bajo consumo de energía (por las siglas en inglés de Low Power Mode).

ISSN 2523-6784

Con la finaldad de reducir el número de dispositivos utilizados, nuestro datalogger aprovecha el módulo de operaciones de acceso directo a la memoria que tienen algunos microcontroladores, conocido como DMA y con ello se utiliza un solo microcontrolador. Mientras se escribe o lee datos a la memoria NAND Flash en conjunto con sus operaciones de codificación y decodificación, también se transmite vía UART la información que se escribe o lee de la memoria NAND Flash con la utilización del módulo DMA.

Las operaciones de lectura, escritura y borrado degradan las memorias NAND Flash (Micheloni, Crippa, & Marelli, 2010). Un código ECC tipo Hamming (255,247) es implementado debido al requerimiento mínimo de 24 bits ECC por cada 1080 bytes de datos
El código Hamming implementado agrega 8 bits de paridad por cada 247 bits de mensaje por lo que el código es de 255 bits. El código Hamming implementado puede corregir 1 bit erróneo en un código de 255 bits. En el caso de una página, el código implementado puede corregir 64 bits erróneos por cada 2040 bytes almacenados en página.

Las contribuciones del presente trabajo se resumen en los siguientes puntos:

1) La sustitución del uso de dos microcontroladores actuando en paralelo por el uso de un solo microcontrolador con operaciones de acceso directo a la memoria DMA utilizando el protocolo UART (Universal Asynchronous Receiver Transmitter por sus siglas en inglés).

2) El microcontrolador utiliza el modo de bajo consumo en un 70 % del tiempo de operación para operación de escritura de memoria.

3) El datalogger utiliza un código ECC tipo Hamming (255,247) que permite corregir 64 bits erróneos en una página de 2040 bytes siempre que exista un solo error por cada 255 bis de código.

En la sección 2, se presenta la arquitectura del datalogger desarrollado, en la sección 3 los resultados obtenidos, en la sección 4 la discusión del trabajo y finalmente en la sección 5 las conclusiones.

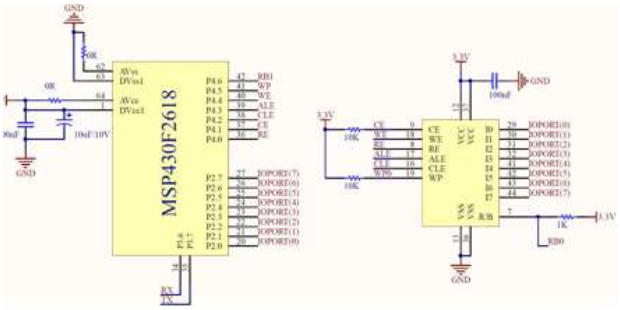

Figura 1 Diagrama esquemático del datalogger

Arquitectura del Datalogger

La **Figura 1** es el diagrama esquemático del datalogger el cual muestra las conexiones del microcontrolador MSP430F2618 a la memoria NAND Flash JY27UF084G2B que tiene 2040 bytes en el área de datos y 64 bytes en el área de reserva.

A continuación se presenta la arquitectura del *datalogger* de nuestro trabajo, primero se da a conocer el control mediante operaciones de acceso directo a la memoria, luego la operación de escritura de memoria NAND Flash, luego la operación de lectura y finalmente el código de corrección de errores tipo Hamming (255,247).

ISSN 2523-6784

Control DMA

El controlador de acceso directo a la memoria transfiere datos desde una dirección a otra, sin intervención del CPU. El uso del control DMA puede incrementar el rendimiento de los módulos periféricos. También puede reducir el consumo de energía permitiendo que el CPU permanezca en modo de bajo consumo hasta que exista una interrupción externa por parte de algún periférico.

En la Figura 2 se presenta el diagrama a bloques del control DMA que se usó para el presente artículo. El control DMA requiere seleccionar el periférico a usar por medio de un multiplexor, en el presente trabajo, se usa el periférico UART para la transmisión y recepción de información. Luego del multiplexor de perifércos, se configuran los registros del control DMA, donde se destacan tres registros.

El registro fuente, el registro destino, y la cantidad de datos que se deben de mover, todo está condicionado por un bit que habilita el control DMA. En la recepción de datos mediante UART controlada por DMA, la dirección origen es el búfer que contiene el dato recibido por serial, este dato es transmitido por el módulo DMA hacia una memoria RAM interna. La transmisión de datos mediante UART tiene como dirección de origen una memoria RAM interna, que por medio del control DMA, mueve uno a uno los datos contenidos en la memoria RAM hacia el búfer de transmisión.

La cantidad de datos que se transmiten vía UART se configura en el registro de "cantidad de datos" que son 1020 correspondientes cuatro códigos Hamming generado (247 x 4 bytes de mensaje + 8 x 4 bytes de paridad).

A la derecha de la configuración DMA encontramos la dirección de la memoria interna el cual, los datos en ella son enviados de una dirección a otra de acuerdo a la configuración previamente realizada.

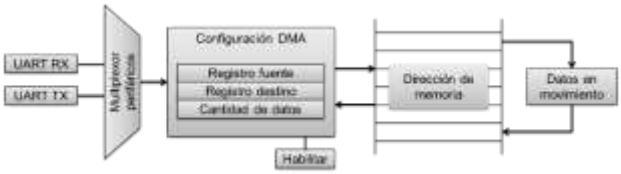

Figura 2 Diagrama de bloques de control DMA

Escritura de página

En la Figura 3 se muestra el diagrama a bloques de la lógica interna del datalogger para la escritura de una página. La recepción de los datos por serial se realiza de manera continua, cada dato recibido se mueve a un vector B mediante el módulo DMA.

El vector B tiene una capacidad de 988 bytes, si la recepción de datos llega a 988 bytes, entonces B se encuentra llena y la información almacenada en B es movida al vector A, el cual tiene una longitud de 1020 bytes. En el tiempo en que se hace el movimiento de la información de B a A, se continúa recibiendo datos por UART, los cuales se continúan moviendo a B, iniciando nuevamente en la posición 1 hasta llegar a los 988. El tiempo que lleva mover los datos de B a A es menor que el tiempo en que se recibe un dato por serial, por lo que no existen perdidas de información.

Del vector A se generan los códigos de corrección de errores por medio de la función "conversión", dando como resultado 32 bytes nuevos, los cuales son concatenados con A, teniendo ahora 1020 bytes de los cuales 988 bytes de información de interés para el usuario y 32 bytes de código de corrección de errores.

ISSN 2523-6784

El datalogger solo hace uso de 2040 bytes por página, estos bytes lo toma del área de datos. La función conversión crea cuatro códigos Hamming (255,247).

Los datos en A son almacenados en la primera o segunda sección de la página mediante un selector el cual está controlado por el número de paquete recibido (posición 0 a 1020), solo se pueden almacenar dos paquetes por página.

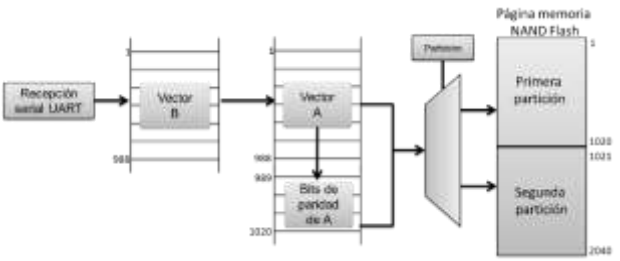

Figura 3 Diagrama de bloques de *datalogger* para escritura de página

Lectura de página

En la **Figura 4** se muestra el diagrama a bloques de la lógica interna del datalogger para la lectura de una página. La asignación de datos a una página de memoria NAND Flash se hace en dos subsecciones de 1020 bytes cada una, dando como resultado 2040 bytes por página. La sección a leer de la página es elegida por un selector y transferido a dos vectores, A' de 988 bytes y ParA de 32 bytes.

El vector A' contiene los datos de interés del usuario y ParA contiene los bits de paridad, ambos vectores fueron almacenados en la página realizada en el proceso de escritura. Con A' se generan nuevos bits de paridad y son almacenados en el vector ParA'. Haciendo uso de ParA y ParA' es posible corregir errores en A' que se hayan generado en el momento de la escritura.

Los datos de A' son transferidos a uno de dos vectores llamados X e Y. X e Y cuentan con 988 bytes y reciben los datos de interés del usuario, la selección para mover los datos de A' a X o A' a Y se realiza por medio de un selector.

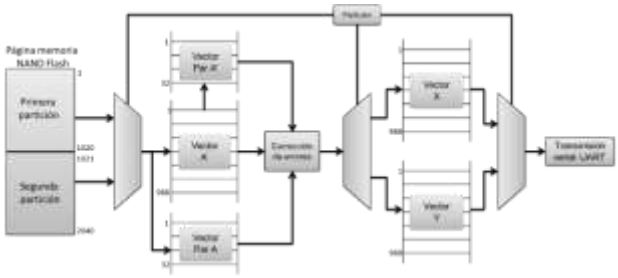

Figura 4 Diagrama de bloques del *datalogger* para lectura de página

Finalmente el vector seleccionado, X o Y, es transferido por protocolo UART mediante el módulo DMA, la transmisión de estos vectores por UART es continua y se realiza por paquetes de 988 bytes. El tiempo de transferencia por UART permite al datalogger realizar un nuevo ciclo de lectura, en el cual se lee la siguiente partición, en caso de ser necesario corrige el vector A' y se mueven los datos de A' al vector que no está siendo usado por el módulo DMA. Los tres selectores usados en los multiplexores es el controlador por la sección leída, donde a la primera sección le corresponde el vector X y a la segunda sección le corresponde el vector Y.

Módulo Hamming (255,247)

En la Figura 5 se presenta el sistema de codificación y decodificación de mensajes para una memoria NAND Flash, en el presente trabajo, tanto el ingreso como la extracción es la transmisión UART.

ISSN 2523-6784

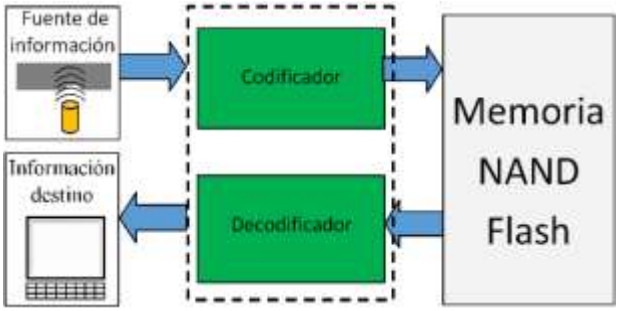

Figura 5 Sistema de codificación y decodificación de mensajes para una memoria NAND Flash

En la construcción de códigos Hamming se deben reacomodar las palabras de código a generar reservando las posiciones que sean potencia de dos para los bits de paridad (Figura 6) (Jiménez, 2016). En la **Figura 6**, las m son para los bits de mensaje y las u son para los bits de paridad. En la Figura 7, los bits de paridad se calculan utilizando operaciones XOR entre los bits que se encuentran en su fila sombreados en azul y se almacenan en los que se encuentran sombreados en verde.

	Posición						
mensaje	1	2	3	4	5	6	7
	u1	u2	m1	u3	m2	m3	m4
1110	-	-	1	-	1	1	0

Figura 6 Ejemplo de estructura de una palabra código de Hamming

Se realizaron tres arquitecturas de codificación Hamming, las cuales se mencionan en los apartados 2.4.1, 2.4.2 y 2.4.3.

	Posición						
	1	2	3	4	5	6	7
	u1	u2	m1	u3	m2	m3	m4
u1			1		0	1	1
u1	1		1		0	1	1
u2		1	1		0	1	1
u3				0	0	1	1

Figura 7 Ejemplo de construcción de un código Hamming (7,4)

Hamming de forma horizontal

Para poder almacenar los datos y los bits de paridad en una memoria NAND Flash, primero se optó por tomar los bits de mensaje de forma horizontal. Las filas de la **Figura 8** son los bytes que serán almacenados, las columnas son los bits de cada byte.

1	…	7	8
9	…	15	16
⋮	⋮	⋮	⋮
241	…	247	248

Figura 8 Primera arquitectura de codificación Hamming la cual toma los mensajes horizontalmente

La arquitectura que codificó utilizando mensajes de forma horizontal fue la presentada en la Figura 9.

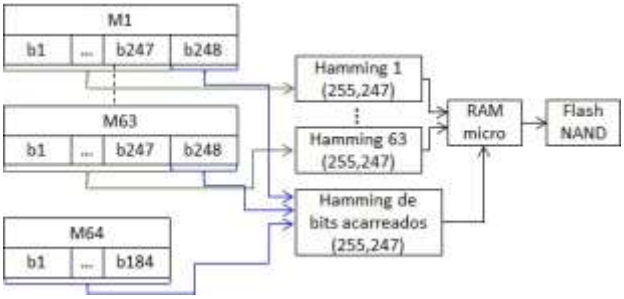

Figura 9 Arquitectura para codifcación Hamming (255,247) de forma horizontal

Los 247 bits se reacomodaron siguiendo la metodología de la **Figura 6**, por lo que se obtuvo la
Figura 10, en donde se nota que las posiciones potencias de 2 se reservan para los bits de paridad.

Hamming de forma vertical

El segundo método para codificar fue tomando los bits de mensaje de forma vertical (
Figura 11). Cada fila representa un byte y cada columna representa un mensaje a codificar.

ISSN 2523-6784

Este método es menos complejo ya que las operaciones xor consideran toda la fila o byte.

u1	u2	m1	u3	m2	m3	m4	u4
m5	m6	m7	m8	m9	m10	m11	u5
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
m20	m21	m22	m23	m24	m25	m26	u6
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
m51	m52	m53	m54	m55	m56	m57	u7
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
m114	m115	m116	m117	m118	m119	m120	u8
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
m241	m242	m243	m244	m245	m246	m247	m248

Figura 10 Estructura de un código Hamming de forma horizontal (255, 247)

b8_1	b7_1	…	b1_1
b8_2	b7_2	…	b1_2
⋮	⋮	⋮	⋮
b8_247	b7_247	…	b1_247

Figura 11 Construcción Hamming de forma vertical

Cada página tiene 2048 bytes de datos y 64 bytes para los códigos ECC, por lo que cada página tiene en total 2112 bytes, que corresponden a 8.28 códigos de 255 Bytes. Se optó por tomar 8 códigos que corresponden a 2040 bytes (**Figura 12**).

Bits de mensaje							Bits de paridad						
0	…	246	…	1729	…	1975	0	…	7	…	56	…	63

Figura 12 Método de almacenamiento en una página

Se procedió a calcular los códigos Hamming mediante operaciones xor mediante el algoritmo de la **Figura 13.**

El algoritmo de la **Figura 13** no está optimizado en cuanto al tiempo que se requiere para la generación de los bits de paridad, es por ello que se implementó un algoritmo que optimizara este tiempo.

```
1. pos_men=0, nx[8]=-1, x_ini={1,2,4,8,16,32,64,128}, xo[8x8]={0}
2. for pos_cod=1:255
3.    if(pos_cod==1,2,4,8,16,32,64,128)
4.       pos_men=pos_men+1;
5.    end if
7.    for i=1:8
8.       if (pos_cod>=x_ini(i))
9.          nx(i)=nx(i)+1;
10.      end if
11.      if ((nx(i)>0) & (mod(nx(i),x_ini(i)*2)<x_ini(i)))
12.         xo(i,:)=xor(x(pos_men,:),xo(i,:));
13.      end if
14.   end for
15. end for
```

Figura 13 Algoritmo para el cálculo de códigos Hamming

Hamming vertical optimizado

En la **Figura 14** se puede observar que m11 es parte de la operación xor tanto para u1, u2, u3 y u4; m10 es parte de u2, u3, y u4; m8 y m9 es parte de u3 y u4. Lo mismo se puede decir para: m4 es parte de u1, m3 de u2 y de u3. Esta observación se utilizó para que el cálculo de u1 ayudara a u2, u2 a u3 y u3 a u4. Con esto fue posible mejorar el tiempo de cálculo de los bits de paridad y con ello optimizar el tiempo de codificación.

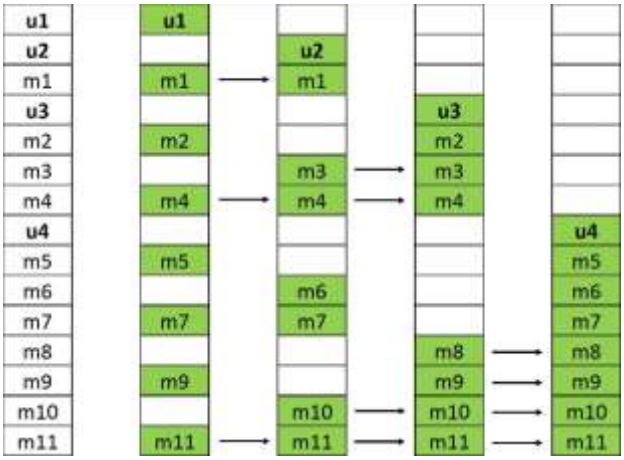

Figura 14 Método para encontrar bits de paridad Hamming de forma optimizada

Las operaciones xor fueron para un código Hamming (255,247).

La Figura 15 muestra el diagrama de bloques para la decodificación. Para la decodificación, primero hay que leer la memoria NAND Flash, este nos entrega el vector de datos A y el vector de bits de paridad ParA.

Luego, se calculan los bits de paridad del vector de datos A' dando como resultado el vector ParA'. Una vez calculados los bits de paridad ParA', se realiza la operación XOR entre los bits de paridad ParA' y ParA.

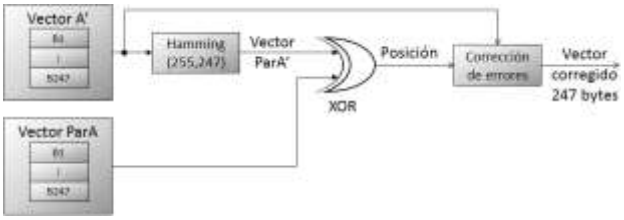

Figura 15 Diagrama de bloques para decodificación Hamming

El resultado de la operación XOR entrega la posición que hay que corregir, que es conocida como síndrome. Una vez teniendo la posición a corregir se corrigen los errores y se entrega como resultado el vector de datos corregido.

Evaluación del módulo Hamming

Para evalar la corrección de errores, se generaron 1, 2, 4, 8, 16, 32 y 64 errores de forma aleatoria y obligando a que en cada código (n, k) existiera a lo mucho un error. La **Figura 16** muestra un ejemplo de generación de 16 errores. Las cruces en rojo representan los códigos en los que existe un error y los números representan la dirección de los bits que mutaron en cada uno de los códigos.

Ejemplo de 16 errores en una página

	B0	B1	B2	B3	B4	B5	B6	B7		B0	B1	B2	B3	B4	B5	B6	B7
C0	✓	✓	✓	×	✓	×	✓	✓	C0	0	0	0	94	0	176	0	0
C1	✓	✓	✓	✓	✓	✓	✓	✓	C1	0	0	0	0	0	0	0	0
C2	✓	✓	✓	✓	×	✓	✓	×	C2	0	0	0	0	206	0	0	233
C3	×	×	×	✓	✓	✓	×	✓	C3	153	78	139	0	0	0	89	0
C4	✓	✓	×	✓	✓	✓	✓	✓	C4	0	0	62	0	0	0	0	0
C5	✓	✓	✓	✓	✓	✓	×	✓	C5	0	0	0	0	0	0	0	10
C6	✓	✓	×	×	✓	×	×	✓	C6	0	0	0	116	0	21	10	0
C7	×	×	✓	✓	×	✓	✓	✓	C7	31	18	0	0	155	0	0	0

Figura 16 Ejemplo de generación aleatoria de 16 errores

Se generó una señal seno con 1976 puntos (o bytes). Se realizó el cálculo de los bits de paridad de la señal seno. A continuación, a la señal seno se le asignaron las mutaciones de bits de acuerdo a la **Figura 16** por lo que se generó una señal seno con puntos erróneos. La señal seno con puntos erróneos y los bits de paridad de la señal sin errores se envían al microcontrolador. El microcontrolador adquiere la señal seno con puntos erróneos y los bits de paridad de la señal sin errores. El microcontrolador realiza la decodificación y corrige errores como se muestra en la **Figura 17**.

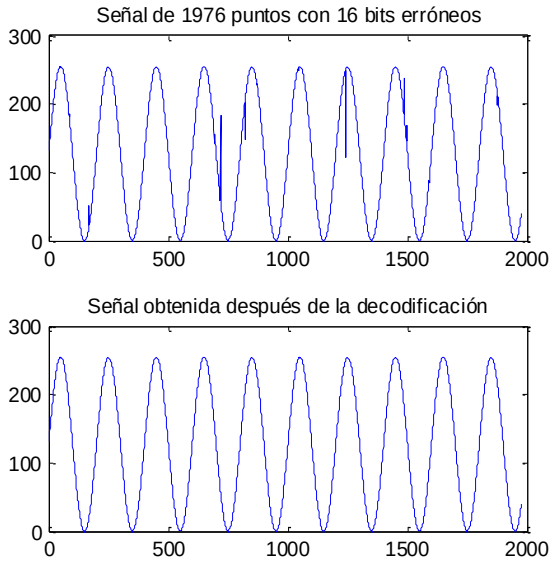

Figura 17 Comparación de la señal con errores enviada y la señal corregida

ISSN 2523-6784

Resultados y discusión

La implementación de la lectura y escritura de una memoria NAND Flash se realizó en un microcontrolador de Texas Instruments MSP430F2618 operado a 16MHz que es de bajo consumo y tiene operaciones de acceso directo a la memoria (DMA), el almacenamiento se realizó en una memoria marca Hynix® de 256 MB.

El método seleccionado el cual resultó en menor tiempo de codificación y menor complejidad en el programa en C fue el Hamming vertical optimizado.

Los resultados obtenidos para la lectura, escritura y codificación Hamming se mencionan en las secciones 3.1, 3.2 y 3.3.

Escritura de memoria

El resumen de tiempos para la operación de escritura se puede observar en la **¡Error! No se encuentra el origen de la referencia.** y en la Figura 18.

Operación	Tiempos (ms)
Traspaso del vector B a A	2.626
Codificación	6.438
Escritura de NAND Flash	3.374
LPM	30.374
Recepción de datos (RX)	42.85

Tabla 1 Resumen de tiempos para escritura de memoria NAND Flash

De la Figura 18, se puede observar que el tiempo de recepción de datos por UART transcurre en paralelo a las operaciones de codificación, escritura y LPM. En la **¡Error! No se encuentra el origen de la referencia.** se observa que el tiempo en modo de bajo consumo LPM es de 30.374 ms, lo que equivale a un 70% del tiempo total.

En la **Figura 18** se puede observar las operaciones necesarias para la escritura de una página: traspaso de B a A, codificación, escritura y LPM. La escritura de página se realiza en dos partes, cada una de 1020. También se observa que por cada página se realizan dos veces las operaciones de escritura debido a las dos subsecciones que tiene cada página.

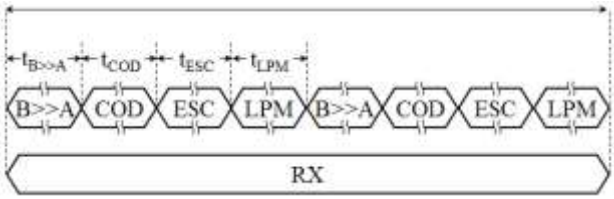

Figura 18 Diagrama de tiempo para escritura de una página en memoria NAND Flash

Lectura de memoria

La **¡Error! No se encuentra el origen de la referencia.** muestra el resumen de tiempos en la operación de lectura para cantidad de errores igual o menor a 14 bits por página.

En la **Figura 19** se puede observar que las operaciones de lectura de memoria y decodificación ocurren en paralelo con la transmisión TX, esto se debe a las operaciones de acceso directo a la memoria DMA. También es importante notar de la **¡Error! No se encuentra el origen de la referencia.** que el tiempo en modo de bajo consumo LPM casi representa la mitad del tiempo de transmisión.

Operación	Tiempos (ms)
Lectura de NAND Flash	13.1
Decodificación	6.5
Traspaso a vectores X y Y	2.6
LPM	20.3
Envío a usuario	42.85

Tabla 2 Resumen de tiempos de lectura de memoria NAND Flash para una página

ISSN 2523-6784

En la **Figura 19** se puede observar las operaciones necesarias para la lectura de una página: lectura de NAND Flash, decodificación, traspaso a vectores X e Y y el estado LPM. Al igual que en la escritura, como cada página tiene dos secciones, es necesario leer las dos secciones de la página de memoria NAND Flash.

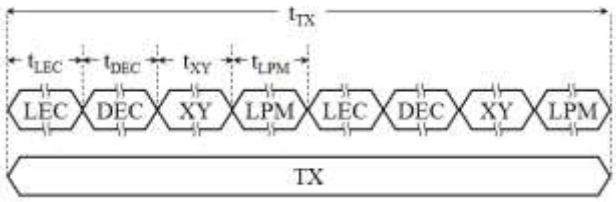

Figura 19 Diagrama de tiempos para lectura de una página en memoria NAND Flash

La **Tabla 3** muestra una comparativa entre trabajar con un microcontroador que utiliza DMA y ECC Hamming (255,247) contra el uso de dos microcontroladores desarrollado por Medina Rodríguez, et al., (2016). El tiempo de lectura y escritura no varía mucho, sin embargo, se observa que el tiempo de lectura/escritura es menor para el uso del DMA (un microcontrolador), esto debido a que la cantidad de datos que se almacena utilizando un microcontrolador es de 2040 bytes que es menor que 2048 bytes (utilizando dos microcontroladores). La diferencia en bytes almacenados es porque el código Hamming (255,247) utiliza 255 bytes de código por lo que en una página se almacenan múltiplos de 255 bytes. Para aprovechar la mayor cantidad de memoria, se utilizan 2040 bytes que es múltiplo de 255 bytes y no excede el tamaño de página (2112 bytes).

El DMA, no reduce el tiempo de lectura/escritura además de que permite reducir el consumo de energía manteniendo al microcontrolador en modo de bajo consumo LPM.

Característica	2 □c (Medina Rodríguez, et al., 2016)	Trabajo propuesto
Tiempo de lectura (una página)	44.95 ms	42.85 ms
Tiempo de escritura (una página)	44.95 ms	42.85 ms
ECC	×	Hamming (255,247)
Capacidad de memoria	Hasta 8 GB	Hasta 8 GB
Velocidad de almacenamiento	45.5 KB/s	47.6 KB/s
LPM	✓	✓

Tabla 3 Comparativa en características de la utilización de dos microcontroladores contra un microcontrolador que utiliza DMA

Corrección de errores

Para la codificación, el tiempo ocurre de acuerdo a la **¡Error! No se encuentra el origen de la referencia.** (6.438 ms). La operación de escritura no tiene ningún problema en tiempos ya que el microcontrolador termina en 12.438 ms en hacer el proceso de codificación y escritura de memoria. Los siguientes 30.374 ms el microcontrolador se encuentra en estado de bajo consumo, situación que no siempre ocurre en la lectura de página.

El tiempo de decodificación para una lectura de página depende de la cantidad de bits erróneos que tenga la página que se va a leer. En caso de tener más de 14 errores por página el tiempo de transmisión por UART se incrementa. En el **Gráfico 1** se muestra los tiempos de decodificación en función del número de bits erróneos.

ISSN 2523-6784

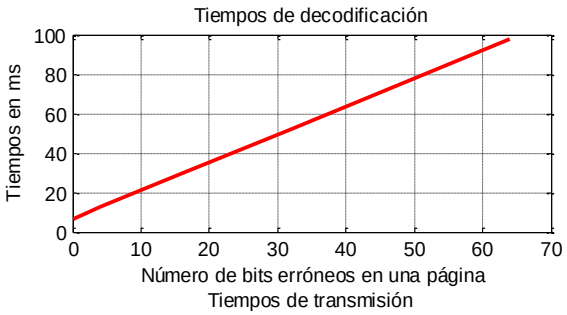

Gráfico 1 Tiempos de decodificación / transmisión para una página

La ecuación 1 es una aproximación del tiempo de transmisión de una página en función del número de bits erróneos por página.

$$t = \begin{cases} 43.5 & [ms]\ si\ be \leq 14 \\ 1.4 * be + 50.435\ [ms] & si\ be > 14 \end{cases}, be \in \mathrm{N} \quad (1)$$

Para una cantidad de bits erróneos menor o igual a 14 bits por página, es posible realizar la transmisión en un tiempo de 43.5 ms. En caso de tener de 15 a 64 bits erróneos, la decodificación corregirá los bits mutados sí y solo si un código de 255 bits tiene por mucho un error. En caso de tener más de un error, el algoritmo puede detectar e intentar corregir pero no garantiza la corrección. La caracterización del tiempo de decodificación podría apoyar a predecir la degradación de la memoria.

En caso de requerir mayor capacidad de detección, los códigos Hamming pueden ser expandidos a códigos Hamming extendidos esto añadiendo un bit extra que se calcula mediante la ecuación 2.

$$\mu_x = c_0 \oplus c_1 \oplus + \cdots + c_{n-1} \quad (2)$$

Con los códigos Hamming extendidos se puede detectar hasta tres errores pero al igual que el Hamming básico, solo garantiza la corrección de un error (Jiang, 2010).

La detección de errores tomaría importancia en caso de que el almacenamiento de datos no fuese en tiempo real. En caso de ser tiempo real, y requerir mayor capacidad de corrección, se pensaría en un código con una capacidad mayor de corrección de errores. Algunos ejemplos de códigos con mayor capacidad de corrección son: LDPC, BCH, Reed Solomon.

Conclusiones

De las ventajas que lleva la arquitectura del datalogger encontramos: 1) la sustitución de un microcontrolador por operaciones de acceso directo a la memoria, esto tiene como resultado la utilización de un solo microcontrolador; 2) la implementación de códigos de corrección de errores, el cual puede corregir un error por cada 255 bits. Para una página se garantiza la corrección de 64 bits erróneos siempre y cuando un código de 255 bits tenga máximo un error. En caso de tener dos errores, puede detectar, intentar corregir, pero no se garantiza la corrección.

Para la decodificación, en caso de que una página tenga más de 14 bits erróneos, el tiempo de transmisión al usuario se incrementa.

Agradecimientos

Se agradece al CONACYT por el apoyo en la realización del presente trabajo. Fondo Secotrial de Investigación para el Desarrollo Aeroportuario y la Navegación Aérea (ASA-CONACYT), por el proyecto número 242864

Referencias

Bez, R., Camerlenghi, E., Modell, A., & Visconti, A. (2003). Introduction to Flash memory. *Proceeding of the IEEE*.

C. Park, J. S. (2003). Cost-efficient memory architecture design NAND Flash memory embedded systems. *International Conf. on Computer design*.

Febrian, H., Hilwadi, H., Desta, Y., & Muhammad, A. T. (2016). Design and Implementation of Data Logger Using Lossless Data Compression Method for Internet of Things. *IEEE 6th International Conference on System Engineering and Technology(ICSET)*.

Ibrahim, D. (2010). Design of a GPS data logger device with street-level map interface. *Advances in Engineering Software*, 859.

Jiang, Y. (2010). *A Practical Guide to Error-Control Coding Using MATLAB®.* Norwood, MA: Artech House.

Jiménez, L. A. (2016). *Diseño y desarrollo de arquitectura para la detección y correción de errores en un arreglo RAID-6 para un controlador de memorias NAND-Flash.* Querétaro: CIDESI.

Kobsar, D., Chad, O., Paranjape, R., Hadjistavropoulos, T., & Barden, J. (2014). Evaluation of age-related differences in the stride-to-stride fluctuations, regularity and symmetry of gait using a waist-mounted tri-axial accelerometer. *Gait & posture*, 553-557.

M. N.N., O. A. (2013). Design of Data Logger with Multiple SD Cards,» Clean Energy and Technology (CEAT). *2013 IEEE*, 175-180.

Medina Rodríguez, C. M., De León Gordillo, D., Rodríguez, N. A., G. Hernández, A., & Soto-Cajiga, J. A. (2016). Design of a data logger using a NAND Flash memory with a parallel architecture based on ultra-low power microcontroller. *Congreso Internacional sobre Innovación y Desarrollo Tecnológico*.

Micheloni, R., Crippa, L., & Marelli, A. (2010). *Inside NAND Flash Memories.* Agrate Brianza: Springer.

N.N., M., A.M., O., S.Z., M. N., & M.Z., M. R. (2013). Design of Data Logger with Multiple SD Cards,» Clean Energy and Technology (CEAT). *2013 IEEE*, 175-180.

Nisha, K., & Umesh, C. P. (2015). Multi Channel Data Acqusition and Data Logging System for Meteorology Application. *Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, 220-225.

Pavan, P., Bez, R., Olivo, P., & Zanoni, E. (1997). *Flash memory cells - an Overview.*

Rajesh, L., Gao, R., & Krishnamurty, S. (2003). Design and realization of a portable data logger for physiological sensing. *IEEE Transactions on Instrumentation and Measurement*, 1289-1295.

Rivera Fárez, J. L. (2010). Diseño e implementación de un módulo datalogger para registro de datos obtenidos de variables analógicas y/o digitales mediante el módulo USB del PIC18F2550 y el software labview para comunicación con un PC.

Toshiba America Electronic Components, INC. (s.f.). *NAND vs. NOR Flash Memory.*

ISSN 2523-6784