PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO DE UNA ARQUITECTURA PARA FPGA PARA CORRECCIÓN DE ERRORES MEDIANTE CÓDIGOS BOSE-CHAUDHURI-HOCQUENGUEM (BCH) PARA APLICACIONES DE NANOSATÉLITES

Tesis para obtener el título profesional de Ingeniera Electrónica

AUTOR:

Mayte Rociel Giraldo Solis

ASESOR:

Mario Andrés Raffo Jara

Lima, Noviembre, 2022

Declaración jurada de autenticidad

Yo, MARIO ANDRÉS RAFFO JARA docente de la Facultad de Ciencias e Ingeniería de la Pontificia Universidad Católica del Perú, asesor(a) de la tesis/el trabajo de investigación titulado Diseño de una arquitectura para FPGA de corrección de errores mediante códigos Bose-Chaudhuri-Hocquenghem (BCH) para aplicaciones de nanosatélites del/de la autor(a)/ de los(as) autores(as) Mayte Rociel Giraldo Solís

dejo constancia de lo siguiente:

- El mencionado documento tiene un índice de puntuación de similitud de 56%. Así lo consigna el reporte de similitud emitido por el software *Tur*nitin el 08/11/2022.
- He revisado con detalle dicho reporte y confirmo que cada una de las coincidencias detectadas no constituyen plagio alguno. Añado que la citación que tiene 53% es la tesis de Bachiller de mi tesista, Mayte Rociel Giraldo Solis, donde yo figuro también como asesor, la misma tiene por título "Estudio de una arquitectura para corrección de errores mediante códigos Bose-Chaudhuri-Hocquenghem (BCH) para aplicaciones de nano-satélites" y difiere de la tesis de licenciatura, en que la primera es el estudio y la segunda es el diseño.
- Las citas a otros autores y sus respectivas referencias cumplen con las pautas académicas.

Lugar y fecha: 24/11/2022

<table>
<tr><td colspan="2">Apellidos y nombres del asesor / de la asesora:
Raffo Jara / Mario Andrés</td></tr>
<tr><td>DNI:40280202</td><td rowspan="2">Firma
  </td></tr>
<tr><td>ORCID: https://orcid.org/0000-0002-0290-4404</td></tr>
</table>

Resumen

La comunicación satelital implica la transmisión de datos a grandes distancias, además de la exposición a la radiación y fenómenos climáticos. Por ello, es necesaria la implementación de códigos que permitan no solo la detección sino también la corrección de estos errores. De acuerdo con el Comité Consultivo para Sistemas de Datos Espaciales (CCSDS por sus siglas en inglés) y la Cooperación Europea para Estandarización Espacial (ECSS por sus siglas en inglés) se recomienda el código BCH(63,56), el cual tiene la capacidad de corregir 1 bit y detectar 2 en los 63 bits de la palabra de entrada.

El diseño de un decodificador BCH(63,56) se basa en una máquina de estados algorítmica con datapath (ASM-D) en el cual los estados ejecutan los bloques de cálculo de síndrome, localización y corrección del error (búsqueda de Chien). Por otro lado, el decodificador tiene la capacidad de reconocer cuando la palabra de entrada posee más de 2 bits errados; y por tanto, no es posible su decodificación. El primer bloque es el encargado de obtener el síndrome y, a su vez, el peso de Hamming del mismo, lo cual es relevante para conocer si la palabra contiene o no errores y la posición de estos.

En el presente trabajo se realizó el diseño del decodificador BCH(63,56) por medio del software Matlab y el lenguaje de descripción de hardware Verilog HDL, obteniéndose la corrección de 1 bit errado y la detección de 2 bits errados. Esto se implementó en 9 estados de una ASM-D con la cual se obtuvo una frecuencia de operación máxima de 160.54MHz y 360 elementos lógicos, es decir, una utilización menor al 1 % de los elementos lógicos totales. Finalmente para la simulación se generó un Testbench en Verilog HDL, donde se colocaron distintas palabras de entradas para verificar el correcto funcionamiento del decodificador.

Palabra clave - Nanosatélites, Corrección de errores, BCH, decodificador, ASM-D, Arquitectura en HW, FPGA.

Dedicado a mi madre

por ser mi mayor ejemplo de superación,

a mis amigos que siempre me brindaron su apoyo

y a mi asesor que me alentó a seguir ante las dificultades.

Índice General

Índice de Figuras

Índice de Tablas

Introducción

Las comunicaciones satelitales cada día son más necesarias en diversos ámbitos como la transmisión para televisión, el uso del GPS y el estudio de la luna, el sol, asteroides, entre otros por parte de agencias espaciales como la NASA y universidades como es el caso de la PUCP. Sin embargo, los datos transmitidos pueden verse afectados por diversos factores espaciales, por lo que es necesaria la implementación de una técnica que pueda detectar los errores introducidos. El Comité Consultivo para Sistemas de datos Espaciales (CCSDS por sus siglas en inglés) y la Cooperación Europea para la Estandarización Espacial (ECSS por sus siglas en inglés), los que son organizaciones encargadas de presentar protocolos en el diseño de sistemas de comunicación satelital. Con los códigos BCH se tiene la posibilidad de una corrección de errores múltiple y sin importar las posiciones de estos; por ello, serán estos los que se implementarán, específicamente el código BCH(63,56).

En el presente trabajo se realizará el diseño de una arquitectura adecuada, en términos de eficiencia, para el diseño de un decodificador BCH. En el capítulo 1 se presenta el marco problemático, la justificación del trabajo de tesis y los objetivos del mismo. En el capítulo 2 se realiza el estudio del decodificador y los fundamentos necesarios para su entendimiento, así como una revisión de las arquitecturas que podrían implementarse como modelo de solución. En el capítulo 3 se diseña el decodificador BCH(63,56) por medio de una ASM-D considerando los bloques de cálculo de síndromes, localización y corrección del error. En el capítulo 4, se muestran los resultados de las simulaciones de la ASM-D a distintos estímulos de entrada, así como la verificación de la detección de 2 bits errados. Finalmente, se muestran las conclusiones, recomendaciones y trabajos futuros.

Capítulo 1

Marco problemático de errores en transmisiones de satélites

En la actualidad, la transferencia de datos está cumpliendo un papel importante, debido al incremento de la necesidad por transmitir digitalmente información (comunicación de datos digitales, comunicaciones móviles y satelitales).

En los 10 últimos años ha habido avances que indicarían el comienzo de una nueva era de exploración espacial [1], por lo que la comunicación satelital, se vuelve fundamental. De acuerdo con la NASA, los temas de investigación conciernen a la luna, al sol, a asteroides, entre otros, dentro de ellos se tienen diversos objetivos como caracterizar los efectos de la radiación sobre organismos vivos y apoyar la investigación del clima espacial. No obstante, a medida que pasan los años se busca llegar más lejos, explorar el espacio profundo, por ello, para el presente año se tiene el lanzamiento de la misión de exploración 1 (del inglés Exploration Mission 1, nemónico EM-1), el cual proporcionará una base para una posterior exploración humana del espacio profundo [1], [2].

Se debe destacar que, para el cumplimiento de los objetivos mencionados en el anterior párrafo, es sustancial la implementación de satélites que posibiliten la transferencia de información obtenida hacia la estación base. Es necesario indicar que debido a las dificultades de traslado de peso, la NASA, la industria y algunos centros educativos prefieren el uso de nanosatélites que, a diferencia de los satélites medianos (comúnmente implementados por grandes compañías por el elevado financiamiento) tienen un peso que se encuentra en el rango de 1kg a 10kg, y las medidas se reducen a 10cm por lado como se muestra en la figura 1.1.

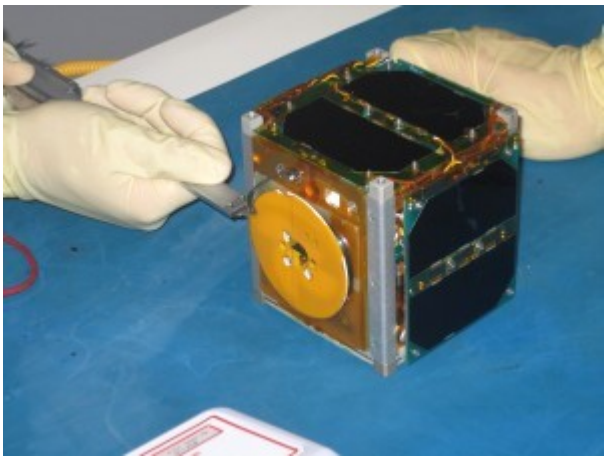

Figura 1.1: Cubo satélite PUCP-SAT-1 [3]

La implementación de los nanosatélites no solo abarca compañías, sino que su diseño actualmente es accesible a universidades. Por ejemplo, la Pontificia Universidad Católica del Perú (PUCP) en el año 2013 permitió ingresar al Perú en la era espacial [3], con el lanzamiento de PUCP-SAT-1, el primer satélite peruano. Para este proyecto fue necesaria la implementación de un software de control y seguimiento, además de la estación terrestre.

1.1. Técnicas de corrección de errores

En la comunicación satelital, se debe cumplir con obtener la información correcta en la estación base en Tierra. Sin embargo, debido a efectos ionosféricos o troposféricos, atenuación en la propagación de ondas de radio debido a la precipitación (lluvia, nieve, granizo, etc.), absorción gaseosa entre oxígeno y vapor de agua, entre otros, no es posible garantizar una transmisión confiable de información [4]. Por lo tanto, es necesaria la implementación de técnicas que permitan asegurar la precisión de datos en el receptor. Para cumplir con lo anterior, se debe implementar una de las 2 formas de detección de errores: consulta automática de repetición (del inglés Automatic Repeat Query, nemónico ARQ) y corrección de errores hacia adelante (del inglés Forward error correction, nemónico FEC).

Para una comunicación satelital usando el método ARQ, se deben colocar bits adicionales a los de datos que servirán para detectar errores. Entonces, cuando se envié información se tendrán 2 casos: paquete correcto y paquete incorrecto. Para el primer caso, el receptor enviará un ACK (del inglés acknowledgement, reconocimiento) para confirmar que los datos son correctos, mientras que para el segundo caso el receptor enviará NACK (del inglés negative acknowledgment, reconocimiento negativo), y pedirá la retransmisión del mensaje original hasta

que este sea el correcto y pueda enviar ACK. Por otro lado, para una comunicación satelital usando el método FEC, se usará redundancia de bits en el mensaje lo que permite no solo la detección de errores sin retransmisión, sino también la capacidad de corregir errores [5].

De lo anteriormente expuesto, se obtiene que para comunicaciones que implican grandes distancias no es conveniente el uso del primer método, ARQ, debido al tiempo que toma la retransmisión del paquete de datos, sino el segundo que permitiría la corrección de errores en la estación base.

1.2. Justificación de la tesis

Para el método FEC, se tiene diferentes códigos: Bose-Chaudhuri-Hocquenghem (BCH), Reed-Solomon (RS), Hamming, entre otros. No obstante, como se hace referencia en [6], los códigos más utilizados en el área de comunicación satelital son los códigos BCH y RS[1], por lo que será necesario realizar una comparación entre estos para elegir el más adecuado para la aplicación.

1.2.1. Comparación entre códigos BCH y RS

En principio, los códigos BCH son códigos binarios, es decir, la detección y corrección de errores se realiza en información binaria o bits, mientras que los códigos RS (caso particular de los BCH) son no binarios, ya que se trabaja sobre símbolos, los cuales pueden contener más de 1 bit. De esto se obtiene que usando códigos BCH, una vez localizado el error lo único es cambiar el valor del bit; sin embargo, en el caso de los códigos RS no solo se debe obtener la ubicación sino también la magnitud del error para ser corregido. Adicionalmente, el BCH tiene la capacidad de corregir errores aleatorios, mientras que el RS es capaz de corregir ráfagas, con lo que se tiene una desventaja en caso se extienda sobre varios símbolos porque no podrá ser corregido, mientras que BCH sí podría hacerlo porque la posición de la ráfaga no depende del símbolo [7], [8]. La complejidad de los códigos BCH se concentra en su decodificación debido al uso de campos finitos o campos de Galois (los códigos RS también los usan), no obstante, mediante el uso de síndromes[1] en códigos BCH se tiene mayor facilidad respecto a la decodificación en RS [5].

Cómite consultivo para sistemas de datos

El cómite consultivo para sistemas de datos (del inglés Consultative Committee for Space Data Systems, nemónico CCSDS) fue fundado en 1982 por las 11 principales agencias espaciales

[1] En el capítulo 2 se presentarán los respectivos fundamentos teóricos.

como la NASA y UK Space Agency. Este es un comité encargado de presentar protocolos para el diseño de sistemas de comunicación satelital [9]. Para la técnica de control de errores, el comité recomienda la implementación de códigos BCH, debido a su corrección múltiple de errores sin importar la posición de ellos, además de su algoritmo de decodificación simple [10].

La figura 1.2 muestra el diagrama de bloques del envío de datos desde el satélite. La trama a enviarse debe ser codificada, en este caso se realiza mediante códigos BCH. El bloque opcional CLTU (del inglés Communication Link Transfer Unit, Unidad de transferencia de enlace de comunicación) es usado para sincronizar la trama, es decir, la secuencia de inicio con la de espera. Finalmente, el bloque PLOP (del inglés Physical Layer Operation Procedure, Procedimiento de operación de capa física) modula el código y envía las ondas de radio [10].

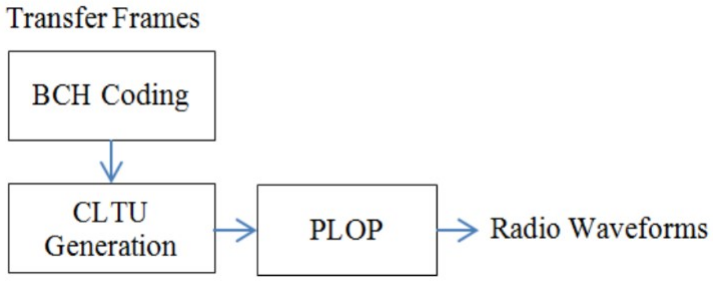

Figura 1.2: Diagrama de bloques del envío de datos [10]

La figura 1.3 representa la recepción de las ondas de radio de figura 1. El bloque Start Sequence Search (Iniciar búsqueda de secuencia) busca la secuencia de inicio generada anteriormente por el bloque CLTU. Esta secuencia ingresa a un decodificador BCH. El comando obtenido a la salida de este bloque ingresará a otro de decisión para ser aceptado o rechazado. En caso sea aceptado se envía la secuencia decodificada a la salida [10].

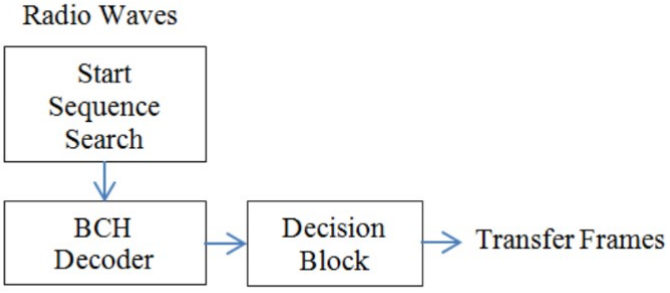

Figura 1.3: Diagrama de bloques de la recepción de datos en la estación base [10]

1.3. Objetivos de la tesis

1.3.1. Objetivo general

El objetivo general del trabajo de tesis es el diseño de la arquitectura de corrección de errores para un decodificador BCH, para ser aplicado en un nanosatélite.

1.3.2. Objetivos específicos

- Entender el funcionamiento interno del algoritmo de corrección de errores BCH.
- Diseñar el algoritmo de corrección de errores BCH en software.
- Realizar la Síntesis Comportamental del algoritmo para obtener un diseño que se rija por medio de una descripción mediante ASM-D (*Algorithmic State Machine* con *Datapath*).
- Realizar el diseño de cada uno de los bloques del decodificador para corrección de errores BCH, mediante técnicas que permitan obtener una mayor eficiencia en términos de throughput considerando un bajo consumo de energía.

Capítulo 2

Fundamentos teóricos de corrección de errores

En el presente capítulo se presentarán los fundamentos teóricos de la elección de los códigos BCH, según lo mencionado en la sección 1.2.1 del capítulo 1. Además, se realizará una revisión de campos de Galois necesaria para diseñar el codificador y decodificador BCH. Finalmente, se efectuará una revisión de arquitecturas implementadas en el decodificador con el objetivo de establecer una propuesta de diseño.

2.1. Comparación entre códigos BCH y RS

Los códigos BCH son los más eficientes y potentes para la corrección de errores aleatorios y potentes, debido a su amplio rango de velocidad y capacidad de corrección de múltiples errores [11]. Estos son también llamados códigos binarios ya que su detección y corrección se realiza bit por bit. Por otro lado, códigos RS son denominados no binarios puesto que la detección y corrección se realiza por símbolos, los cuales pueden contener más de 1 bit.

En la decodificación, para localizar la ubicación del error, ambos códigos son similares por sus 3 bloques principales: syndrome computation (cálculo de síndromes), key equation solver (solucionador de ecuaciones clave) y Chien Search (búsqueda de Chien). Sin embargo, como se ilustra en la figura 2.1, los códigos RS necesitan un bloque adicional, Forney's algorithm, para determinar la magnitud del error, debido a que no necesariamente es un bit [12]. Este bloque adicional genera un incremento en el tiempo de cálculo para el hardware.

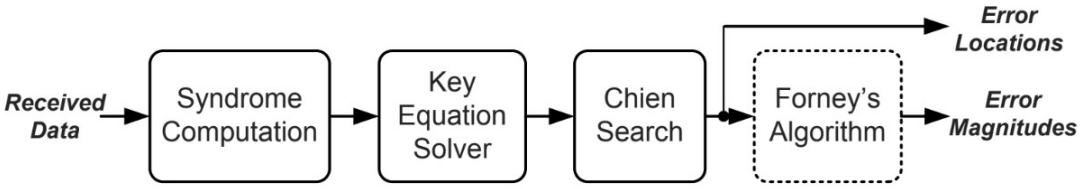

Figura 2.1: Localización de errores en decodificadores BCH/RS [12]

Las comunicaciones satelitales requieren un bloque de modulación con el que la información (datos) se transformen en ondas de radio y puedan ser enviadas. Esta modulación puede ser por medio de Quadrature Amplitude Modulation (Modulación de amplitud en cuadratura / QAM) o Phase Shift Keying (Modulación por desplazamiento de fase / PSK). Además, se debe considerar que en todo canal existe ruido [13].

En las figuras 2.2 y 2.3, se muestran la comparación de los códigos BCH y RS respecto a su rendimiento en un canal de desvanecimiento de Rayleigh para una modulación BPSK y QAM. Se puede observar que en ambos casos los códigos BCH son más eficientes que los RS por su bajo valor en BER (Bit error rate / Tasa de bit errados) [14].

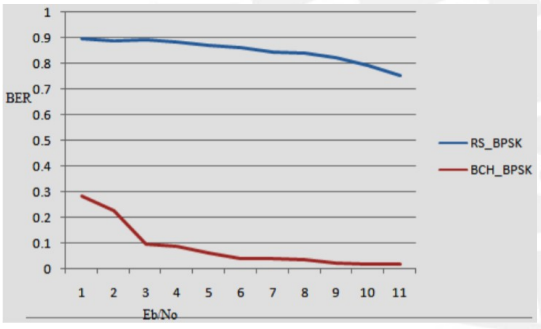

Figura 2.2: BCH y RS en modulación BPSK [14]

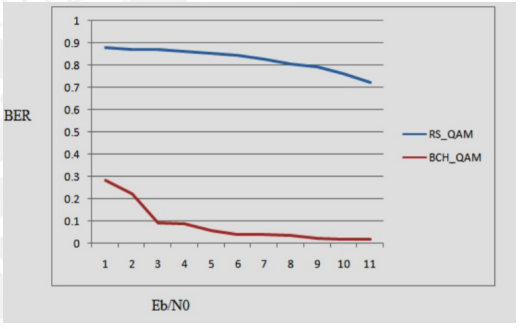

Figura 2.3: BCH y RS en modulación QAM [14]

En conclusión, debido a que las comunicaciones satelitales requieren minimizar el tiempo de cálculo y tener una eficiencia alta en cuanto a la tasa de bits errados, los códigos BCH son los adecuados y necesarios para la corrección de bits errados.

2.2. Campos de Galois, codificador y decodificador BCH

Los códigos BCH tienen como fundamento matemático a los campos finitos o campos de Galois. Para ello será necesario realizar una breve introducción.

2.2.1. Campos de Galois

Son representados como $GF(q)$, donde q es la potencia de un número primo, el número de elementos del campo y el orden del mismo. Se llama un campo de Galois binario cuando $q = 2$. $GF(2)$ son los campos más simples, aunque su importancia radica en que las comunicaciones digitales son binarias. Los elementos en el campo son $\{0, 1, 2, ..., q-1\}$. De manera general, los campos binarios son representados por $GF(2^m)$ [15].

Para efectuar las operaciones en campos de Galois, se debe considerar además la operación módulo de la siguiente manera. Se tomará el siguiente ejemplo:
- Sea el campo $GF(3)$

- Los elementos del campo son $\{0, 1, 2\}$.
- Se debe realizar las operaciones de suma y multiplicación entre sus elementos.
- Sumar o multiplicar los elementos de manera cotidiana.
- Dicho resultado parcial se debe dividir con el número de elementos del campo (3), de la cual se tendrá un residuo (operación módulo), este será el resultado final de la operación.

En la tabla 2.1, se muestran los resultados de la operación suma entre sus elementos, y se obtiene también que dicho resultado pertenece al campo. Esto es replicado en la tabla 2.2 para multiplicación.

Tabla 2.1: Operación suma

+	0	1	2
0	0	1	2
1	1	2	0
2	2	0	1

Tabla 2.2: Operación multiplicación

.	0	1	2
0	0	0	0
1	0	1	2
2	0	2	1

Los elementos pueden ser representados también de forma polinomial.

$$f(x) = f_{n-1}x^{n-2} + f_{n-2}x^{n-2} + ... + f_1x^1 + f_0 \tag{2.1}$$

Para el campo binario los valores de f_x solo pueden ser 0 o 1. Además, en este campo la suma coincide con la puerta lógica XOR bit a bit. Por ejemplo, para $1 + 1 = 2$, 2 mod 2 = 0, lo cual coincide con la puerta XOR $1 \oplus 1 = 0$, también par $1 + 0 = 1$, 1 mod 2 = 1, lo que es igual a $1 \oplus 0 = 1$ [16].

2.2.2. Codificador BCH (n,k)

En el capítulo 1, se hizo referencia al Comité consultivo para sistemas de datos espaciales (CCSDS por sus siglas en inglés), el cual tenía como objetivos brindar un estándar para el diseño de sistemas de comunicación satelital. En la técnica de control de errores no solo se recomienda el uso de códigos BCH, sino también emplear 56 bits para datos o información y 63 bits para su codificación [17] [10] [4] [18].

Para los códigos BCH se tienen los siguientes parámetros:

- k es el tamaño (número de bits) del mensaje que se desea enviar.
- n es el tamaño (número de bits) del mensaje codificado.
- t es la máxima cantidad de errores que se pueden corregir en el decodificador sin importar la posición de estos.
- d_{min} es la distancia de Hamming, la cual debe cumplir $d_{min} \geq 2t + 1$.

Los códigos BCH pueden ser implementados mediante una codificación sistemática o no sistemática, esto depende de cómo se decida codificar el mensaje.

Se debe transmitir un mensaje, el cual puede ser denotado en su forma polinomial $m(x) = m_0 + m_1x + m_2x^2 + +m_{k-1}x^{k-1}$, donde $m_0, m_1, m_2, \ldots, m_{k-1}$ representan los dígitos del mensaje. Además, el polinomio $g(x) = g_0 + g_1x + g_2x^2 + +g_{n-k-1}x^{n-k-1}x^{n-k}$, es denominado el polinomio generador, el cual se obtiene de escoger un polinomio primitivo de grado m y contruir $GF(2^m)$.

1. Codificación no sistemática: En este tipo de codificación, el mensaje $m(x)$ es un factor de la palabra codificada $c(x)$ en su forma más simple, mutiplicando al polinomio generador $g(x)$, como se muestra en la ecuación 2.2.

$$c(x) = m(x)g(x) \tag{2.2}$$

2. Codificación sistemática: En este tipo de codificación, la obtención del polinomio de la

palabra código a enviarse $c(x)$ no se calcula directamente, sino mediante las operaciones de multiplicación y módulo, como se muestra en la ecuación 2.3

$$c(x) = p(x) + x^{n-k}m(x) \;\; donde \;\; p(x) = (x^{n-k}m(x))_{g(x)} \tag{2.3}$$

A pesar de la facilidad para hallar el polinomio de la palabra código $c(x)$ en la codificación no sistemática, se prefiere la implementación de codificación sistemática, debido a que el mensaje $m(x)$ es parte de $c(x)$ en sus últimos k bits [19].

En la figura 2.4, se muestra el diagrama de una codificación sistemática para el código propuesto por el Comité consultivo anteriormente mencionado. En este se puede observar que la información o mensaje es parte de la palabra o bloque de código [18].

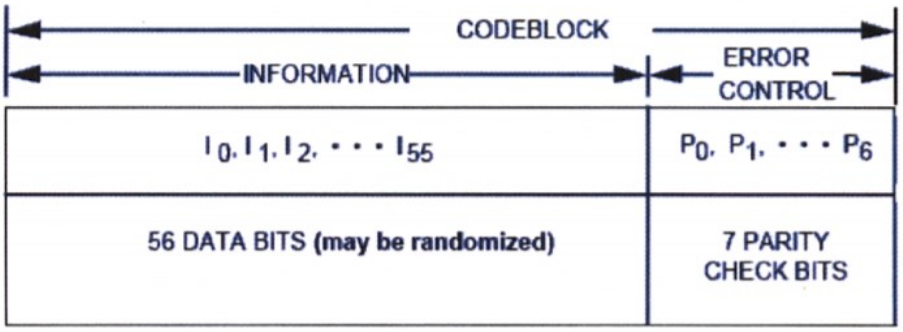

Figura 2.4: Diagrama de codificación sistemática para BCH (63,56) [18]

En la figura 2.5, se muestra el circuito de codifcación $BCH(n,k)$, donde $i(x)$ representa a $m(x), g_1, g_2, ..., g_{n-k-1}$ son los coeficientes de $g(x)$ y $b_0, b_1, ..., b_{n-k-1}$ representan registros de desplazamiento.

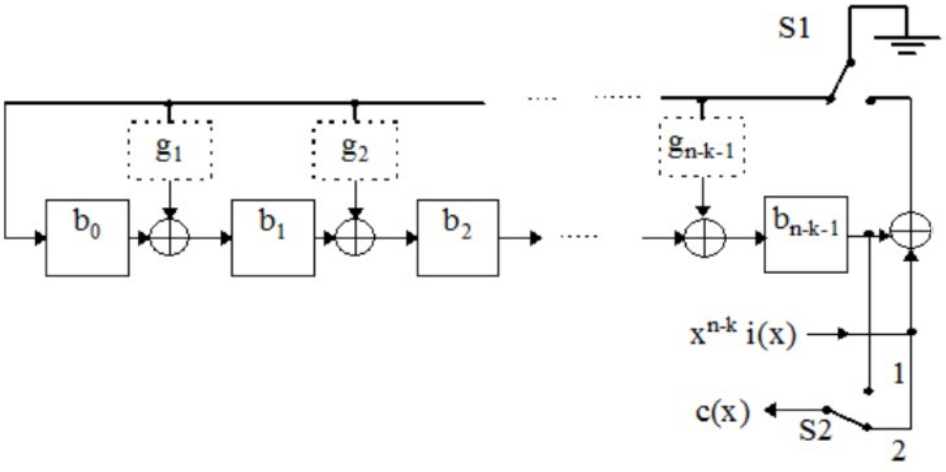

Figura 2.5: Circuito de codificación BCH (n,k) [20]

2.2.3. Decodificador BCH

Debido a los fenómenos mencionados en el capítulo 1, la palabra código transmitida por el codificador $c(x)$ no es la recibida en el decodificador, sino que a esta se le suma un error, el cual puede ser expresado en su forma polinomial $e(x)$. La ecuación 2.4 representa el mensaje recibido a la entrada del decodificador [21].

$$r(x) = c(x) + e(x) \tag{2.4}$$

Se deben seguir los siguientes pasos para decodificar $r(x)$:

1. Almacenar en un registro el mensaje recibido $r(x)$
2. Calcular los síndromes
3. Determinar el polinomio localizador de errores
4. Hallar las raíces del polinomio localizador de errores

Estos pasos son mostrados en la figura 2.6, donde $i_{62}, i_{61}, \ldots, i_0$ representan los bits de $r(x), s_1, s_2, \ldots, s_{2t}$ son los síndromes calculados y $\Lambda_1, \Lambda_2, \ldots, \Lambda_v$ con $v \geq t$ son los coeficientes del polinomio localizador de errores.

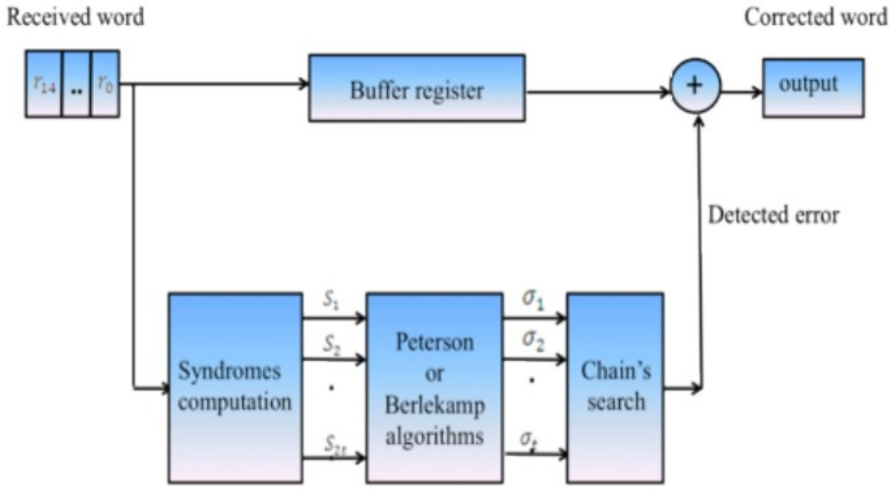

Figura 2.6: Diagrama de bloques del decodificador BCH [22]

- Se debe almacenar en un registro el mensaje recibido $r(x)$ como se muestra en la figura 2.6 con Buffer register, esto con el objetivo de que se pueda añadir a $r(x)$ el vector $e(x)$ con la operación módulo 2, y así obtener el mensaje original $c(x)$ [10].

- Calcular los síndromes.

  Si se evalúa $r(x)$ para $x = \alpha^i$, para $i = 1, 2, 3, \ldots, 2t$, donde $\alpha, \alpha^2, \alpha^3, \ldots, \alpha^{2t}$ son elementos del campo. Por lo tanto, se obtiene la ecuación $r(\alpha^i) = c(\alpha^i) + e(\alpha^i)$, en donde $c(\alpha^i) = 0$, ya que los elementos de campo son raíces del polinomio generador $g(x)$, y del cual $c(x)$ es divisible. La ecuación 2.5 muestra que los síndromes S_i solo dependen del error y no de la palabra código $c(x)$ [19].

$$S_i = r(\alpha^i) = e(\alpha^i) \tag{2.5}$$

Si todos los síndromes tienen como valor 0, esto significaría que no se introdujo error, y se debe pasar la entrada directamente a la salida [21].

- Determinar el polinomio localizador de errores.

  Para este bloque será necesario hacer una revisión de diferentes arquitecturas las cuales serán presentadas en la sección 2.3. Sin embargo, se debe tener la idea de hallar el polinomio localizador de errores por medio del cálculo anterior de síndromes.

- Hallar las raíces del polinomio localizador de errores (del inglés error locator polynomial, nemónico ELP).

  Finalmente, se hallan las raíces del ELP evaluando para $x = 1, \alpha, \alpha^2, \ldots, \alpha^{n-1}$. Entonces, si α^i es una raíz cuando $\Lambda(\alpha^i) = 0$, además, ya que la palabra código recibida tiene una longitud n, α^{n-i} corresponderá al número de posición del error de $r(x)$. El proceso por el cual se encuentran las raíces del ELP es denominado búsqueda de Chien [18] [21].

En la figura 2.7, se presenta el diagrama de flujo que corresponde al decodificador, en el cual se muestran los cálculos necesarios anteriormente descritos para encontrar la palabra código trasmitida por el codificador.

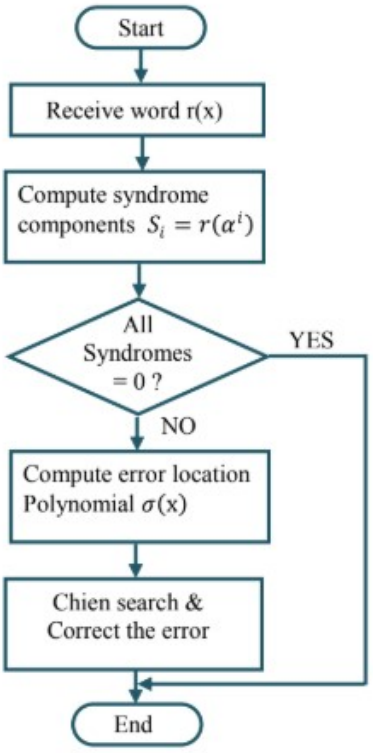

Figura 2.7: Diagrama de flujo del decodificador BCH [21]

2.3. Arquitecturas para implementación de decodificador BCH

Los códigos BCH operan en campos Galois, en donde se deben realizar operaciones que generan un mayor tiempo de cálculo. Estos pueden ser implementados en software o hardware; sin embargo, en las figuras 2.8 y 2.9 se corroboran las ventajas de una implementación en hardware respecto a una en software. En la figura 2.8, se tiene una comparación del área ocupada para software y hardware en función de la cantidad de errores corregidos. En la figura 2.9, se muestra un cuadro comparativo para el tiempo de cálculo en software y hardware del código BCH (15,5) tanto para un codificador como para un decodificador. Como se explicó en la sección 2.2, el código a implementarse será BCH (63,56) el cual tiene la capacidad de corregir 1 bit; por lo tanto, se tendría mayor ventaja en el área ocupada pues sería inferior a la de una implementación en software como se observa en la figura 2.8. Por otro lado, debido al procesamiento paralelo en hardware, el tiempo de cálculo es inferior al de software, además, la diferencia de estos tiempos se incrementa conforme el valor de n aumenta. En consecuencia, la implementación de los códigos BCH se realiza de manera eficaz en hardware [21].

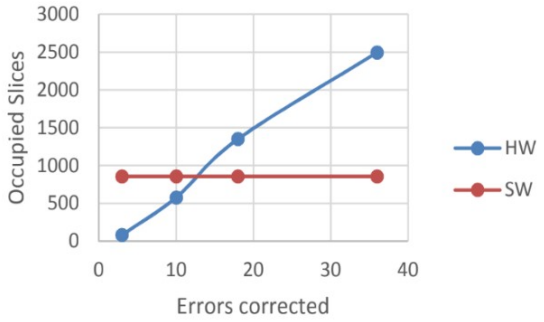

Figura 2.8: Área ocupada vs errores corregidos para software y hardware [23]

	$T_{hardware}$	$T_{software}$	n	k
Encoder	9.6106 µsec	1245.1 µsec	15	5
Decoder	40.22 µsec	13678.2 µsec		

Figura 2.9: Cuadro comparativo de tiempo de cálculo para BCH (n, k) [21]

La presente tesis tiene como propósito el diseño de una arquitectura de hardware para el decodificador BCH, por lo que es necesario conocer las diferentes arquitecturas que se tienen en la literatura y evaluarlas en referencia a parámetros como throughput, frecuencia, potencia y área. El decodificador se compone de 4 bloques, como se hizo referencia anteriormente; no obstante, no todos generan un retraso considerable o tiempo de cálculo elevado, como es el caso del bloque KES (del inglés Key Equation Solver, solucionador de ecuaciones clave) debido a su dificultad de implementación. Por ello se proponen diversas arquitecturas para el bloque KES.

Algunas de las arquitecturas a presentarse tienen como base el algoritmo Berlekamp-Massey (BM); sin embargo, en este se tienen desventajas que deben ser modificadas como la irregularidad de su arquitectura y el retraso de ruta que depende del número de errores que el código pueda corregir.

2.3.1. Arquitectura riBM

La arquitectura reformulada sin inversión Berlekamp-Massey (del inglés reformulated inversionless Berlekamp-Massey, nemónico riBM) fue propuesta en [24] con el objetivo de eliminar el cuello de botella debido al cálculo de discrepancias para códigos Reed-Solomon [19], [24]. Sin embargo, este puede ser utilizado en códigos BCH, además de ser simplificado debido a que estos son códigos binarios.

Con esta arquitectura se disminuye el retraso de la ruta crítica; no obstante, no se consiguen mejoras en la latencia y el número de componentes, es más, se incrementa el número de multiplicadores y multiplexores, y se mantiene el de sumadores.

2.3.2. Arquitectura RiBM

Esta arquitectura fue presentada como una reorganización de su precedente, riBM. El cambio fundamental es que la reorganización genera que los elementos de procesamiento (del inglés processing elements, nemónico PE) sean iguales con lo que se tiene una menor complejidad que en riBM [24], [25], [26]. Sin embargo, en esta arquitectura no se presenta una reducción en el uso de sumadores, multiplicadores y multiplexores de campos de Galois respecto a riBM, lo que significa una mayor área y por tanto mayor consumo de energía [25].

2.3.3. Arquitectura DcRiBM

La arquitectura RiBM sin cálculo de discrepancia (del inglés discrepancy computationless RiBM, nemónico DcRiBM) tiene como objetivo reducir la complejidad de hardware en sumadores y multiplicadores. Para ello, se elimina el bloque de control de cálculo de discrepancia, además de una reducción en el número de PE. No obstante, la cantidad de multiplexores se ve incrementada a más del doble, y por tanto el retraso de ruta crítica resulta mayor en el doble del periodo de los multiplexores al de RiBM [25]. Además de un ligero decremento en el troughput. Esto será expuesto en la tabla 2.3.

2.3.4. Arquitectura Peterson

Esta arquitectura es una de las más implementadas, debido a sus ventajas en la disminución de área de hardware, cuando el número de errores a corregir sea menor o igual a 4. Esto implica un menor consumo de energía, y mejoras en la velocidad de procesamiento. Además, esta arquitectura tiene un cálculo simple para encontrar los coeficientes del polinomio localizador de errores. En caso se requiera una capacidad de corrección de errores mayor, la complejidad y el área incrementan, y por tanto el consumo de energía se eleva [19], [21], [26].

2.3.5. Arquitectura I-Peterson

En la anterior arquitectura Peterson se tienen inversiones o divisiones que incrementan su complejidad. La arquitectura de Peterson sin inversión (del inglés, Inverssion-Less Peterson, abreviatura I-Peterson) elimina la operación de división mediante una multiplicación a los coeficientes, con ello se logra una reducción de área, consumo de energía y retraso de ruta crítico, además de un incremento de la velocidad de operación [26].

2.3.6. Análisis de arquitecturas

Como se presentó en la sección 2.2 el código que se implementará por recomendación de la CCSDS es el BCH (63,56), el cual posee una capacidad de corrección de error de 1 bit. Por lo tanto, el uso de una arquitectura iBM o alguno de sus derivados antes presentados (2.3.1, 2.3.2, 2.3.3) no serían recomendados, ya que se tendría un consumo de energía elevado por sus iteraciones, además de un retraso de ruta crítico elevado lo que incrementa el tiempo de decodificación. Una arquitectura Peterson o I-Peterson serían las adecuadas cuando se tenga como requerimiento una baja complejidad y consumo de energía. Para el caso de $t = 1$, no se tienen diferencias entre Peterson e I-Peterson, ya que no se cuenta con una división [21], [26]. En la tabla 2.3, se presenta la comparación de las arquitecturas expuestas en los parámetros latencia, retraso de ruta crítica y troughput.

Tabla 2.3: Comparación de diferentes arquitecturas

Arquitectura	Latencia	Retraso de ruta crítica	Throughput (GB/s)
riBM	$2t$	$T_{mul} + T_{add}$	-
RiBM	$2t$	$T_{mul} + T_{add}$	3 [3]
DcRiBM	$2t$	$T_{mul} + T_{add} + 2 \cdot T_{mux}$	2,9 [3]
Peterson	-	6.47 ns [2]	-
I-Peterson	$< 2t$	1.51 ns [2]	-

2.4. Propuesta de diseño

Debido al análisis en la sección 2.3, la arquitectura a implementarse para determinar el polinomio localizador de errores es Peterson. Como se presenta en la referencia [19], cuando el número de errores a corregir se encuentra en el rango de [1-4], el algoritmo de Peterson es eficiente, debido a que se pueden encontrar los coeficientes del polinomio por medio de reducciones y sin realizar un proceso iterativo como se hace en el algortimo de Berlekamp.

En la ecuación 2.6, se tiene el ELP, donde el coeficiente $\Lambda_0 = 1$.

$$\Lambda(x) = \prod_{l=1}^{v}(1 - \beta_l x) = \Lambda_0 + \Lambda_1 x + ... + \Lambda_v x^v \tag{2.6}$$

Los síndromes anteriormente calculados se relacionan con β_l como se muestra en la ecuación 2.7,

[2] Implementación en el campo $GF(2^{14})$ [26].

[3] Implementado en el código BCH (2040,1930) [25].

y por tanto con ellos se puede encontrar Λ_v

$$S_i = \sum_{l=1}^{v} \beta_l^i \tag{2.7}$$

En tabla 2.4 se indican los coeficientes en función de los síndromes para diferentes t.

Tabla 2.4: Coeficientes de ELP en función de síndromes

Número de errores	Coeficientes del ELP
$t = 1$	$\Lambda_1 = S_1$
$t = 2$	$\Lambda_1 = S_1$ $\Lambda_2 = \frac{S_3 + S_1^3}{S_1}$
$t = 3$	$\Lambda_1 = S_1$ $\Lambda_2 = \frac{S_5 + S_1^2 S_3}{S_1^3 + S_3}$ $\Lambda_3 = (S_1^3 + S_3) + S_1\Lambda_2$
$t = 4$	$\Lambda_1 = S_1$ $\Lambda_2 = \frac{S_1(S_7 + S_1^7) + S_3(S_1^5 + S_5)}{S_3(S_3 + S_1^3) + S_1(S_5 + S_1^5)}$ $\Lambda_3 = (S_1^3 + S_3) + S_1\Lambda_2$ $\Lambda_4 = \frac{(S_5 + S_1^2 S_3) + (S_1^3 + S_3)\Lambda_2}{S_1}$

Capítulo 3

Diseño de la propuesta

El presente capítulo tiene como objetivo el diseño de la arquitectura del decodificador BCH (63,56). Por lo que será necesario realizar una primera implementación en el software Matlab. Sin embargo, con el objetivo de corroborar la capacidad de corrección de 1 bit de dicho código, se implementará también el codificador BCH (63,56) y un canal que pueda introducir bits errados.
En el diseño de bloques del decodificador se utilizará una descripción por medio de una máquina de estados algorítimica con datapath (del inglés Algorithmic State Machine con Datapath, nemónico ASM-D).

3.1. Requerimientos y consideraciones

La presente tesis consta del diseño de 3 bloques: cálculo de síndromes, polinomio localizador de errores y búsqueda de Chien. Como se mencionó anteriormente, se debe realizar el código de dichos bloques en software.
La síntesis comportamental consiste en la conversión de un código a una descripción RTL (del inglés Register Transfer Level, nivel de transferencia de registros). El objetivo de realizar esta síntesis es obtener una descripción de una máquina de estados algorítmica con datapath.

En la figura 3.1 se muestra el diagrama de bloques de una máquina de estados, la cual posee la siguiente estructura [27]:

- Función del estado siguiente, dada por una función (g) de transformación combinacional
- Estado presente (provista por una función de memoria (secuencial))
- Función del salida, regida por una función (f) de transformación combinacional.

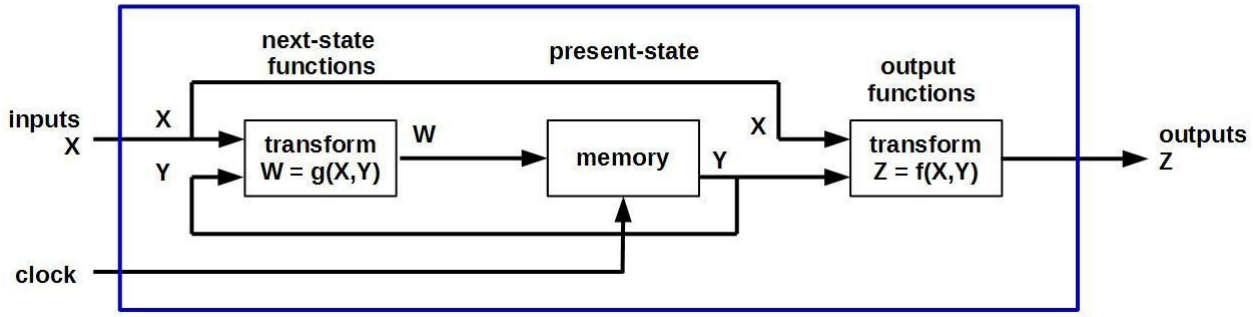

Figura 3.1: Diagrama de bloques de una máquina de estados [28]

Tipos de máquinas de estado [27]

- Tipo Moore: En esta máquina las salidas dependen únicamente del estado actual f(Y).
- Tipo Mealy: En esta máquina las salidas dependen no solo del estado actual, sino también de las entradas (z = f(x,y)).

Debido a que las máquinas tipo Mealy requieren menos estados que las tipo Moore, estas serán las que se implementen. Sin embargo, si se tuviera una variación no deseada en las entradas, esta se vería reflejada en una variación en las salidas. Este problema es solucionado registrando las salidas, evitando así la presencia de glitches o fallas [27].

Para el diseño se empleará un FPGA Cyclone IV E debido a que son fácilmente reprogramables y, como se mostró en la figura 2.9, se tiene un tiempo de cálculo bajo por su procesamiento paralelo.

3.2. Diagrama de bloques

El diseño del decodificador BCH (63,56) se basará en el siguiente diagrama de bloques, en donde se muestran las señales de entrada y salida.

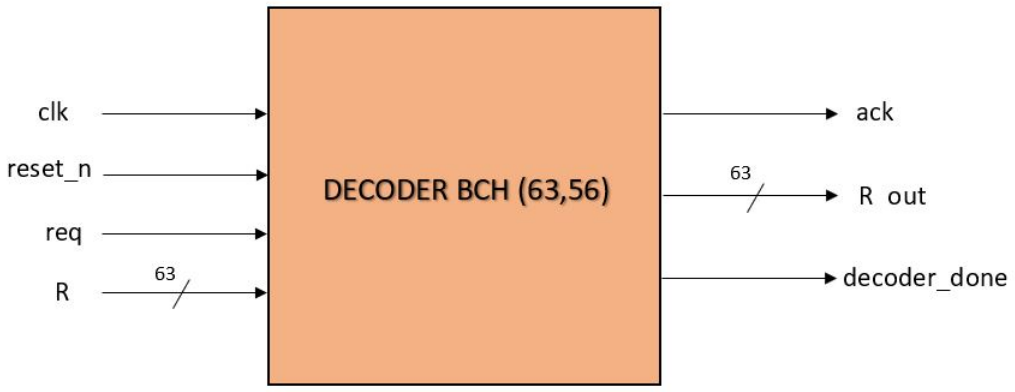

Figura 3.2: Diagrama de bloques del decodificador BCH (63,56)

En la tabla 3.1 se mostrarán las señales, su longitud y una breve descripción de su funcionamiento.

Tabla 3.1: Señales del decodificador BCH (63,56)

Señal	Longitud en bits	Descripción
clk	1	Reloj del sistema
reset_n	1	Reinicio del sistema
req	1	Solicitud para el inicio de decodificación cuando req = '1'
R	63	Palabra a decodificar con o sin error
ack	1	Reconocimiento de la solicitud cuando ack = '1'
R_out	63	Palabra decodificada
decoder_done	1	Señal que permite validar que la decodificación fue correcta cuando decoder_done = '1'

3.3. Bloques de la máquina de estados algorítmica

La máquina de estados algorítmica se dividirá en 9 bloques o estados que permitan la decodificación de la palabra, estos serán explicados a continuación en conjunto con resultados de Matlab que fueron útiles para su elaboración.

3.3.1. Cálculo de síndromes

En el capítulo 1, se obtuvo que el cálculo de síndromes se realiza evaluando la palabra recibida en α^i, con lo que se obtendría el patrón de errores, lo cual es sustancial para poder corregirlos. Se debe recalcar que los síndromes son independientes de la palabra a decodificar R", es decir, se tienen valores fijos de síndrome para cada posición del error (1, 2, 3, . . . , 62, 63).

En la figura 3.3 se muestra el circuito para generar los síndromes del código BCH (63,56). Los bloques $x^0, x^1, x^2, ..., x^6$ representan registros de desplazamiento, la palabra recibida es representada por IN y el síndrome es obtenido en OUT.

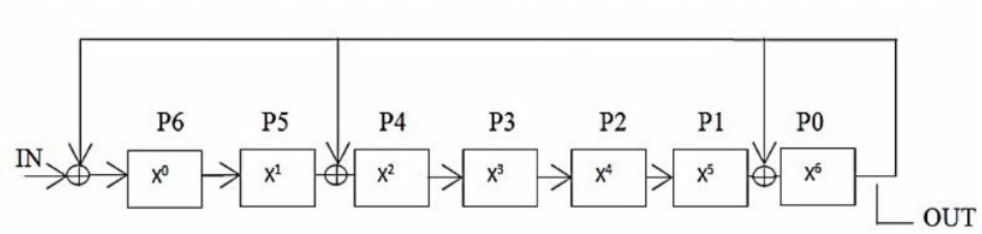

Figura 3.3: Diagrama de bloques del cálculo de síndromes [10]

Para la implementación en MatLab del cálculo de síndrome anterior, se deben considerar vectores de 7 ceros que permitan simular la acción de los 7 registros de desplazamiento. Como se puede observar en la figura 3.3, se tiene una realimentación, lo que implica que la salida debe ser registrada. Además, las operaciones son realizadas bit a bit, es decir, en IN ingresa el primero de los 63 bits de la palabra recibida. Por lo tanto, como se expuso en la sección 2.2.1, la suma es representada por una función XOR.

Para el inicio del cálculo de síndromes, y la decodificación en general, se tiene un primero estado, mostrado en la figura 3.4, en donde se necesitará que la señal req tenga un valor de 1 lógico para inicializar los valores de los registros de desplazamiento (y, yp) en 0, al igual que las señales ack y decoder_done. Además, se necesitará un contador j cuyo valor inicial sea 63 que permita registrar la palabra como se había expuesto.

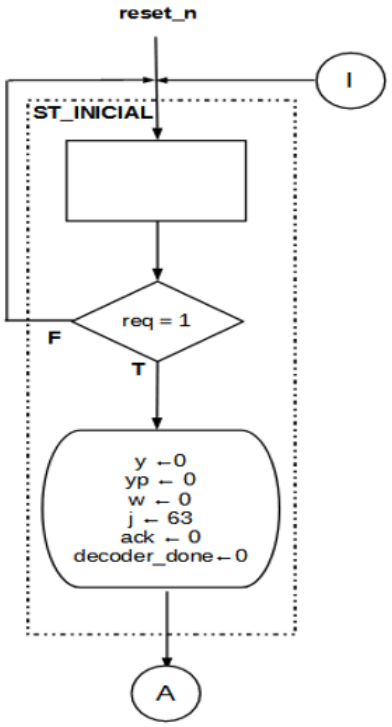

Figura 3.4: Estado inicial

En la figura 3.5, se muestra el estado GENERA_SÍNDROME, en el cual se realiza el cálculo de síndromes[4] por medio de la función XOR y los registros de desplazamientos. El contador j será evaluado desde su valor inicial 63 hasta 0. La condición j>0 permitirá el retorno al estado para continuar con los siguientes bits o el salto al siguiente estado.

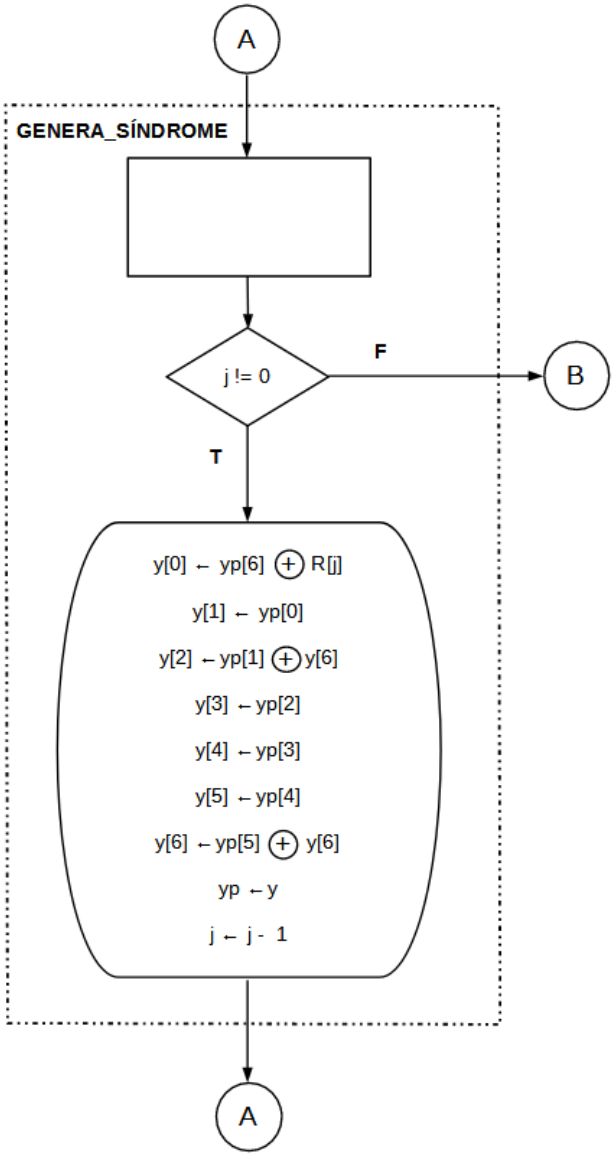

Figura 3.5: Estado GENERA_SÍNDROME

En el estado GENERA_W_HAMMING se registra el valor final de y (registro de desplazamiento) en S (síndrome). Además, se cálcula el peso de Hamming del síndrome, el cual será útil para determinar si es posible decodificar una palabra o no, si no posee error o la localización del mismo. En la figura 3.6, se muestra el cálculo de w (peso de Hamming de S) como la suma de todos los

[4]Cálculo de generación de síndromes tomado de [10]

bits de S; por lo tanto, w se encuentra en el rango [0-7].

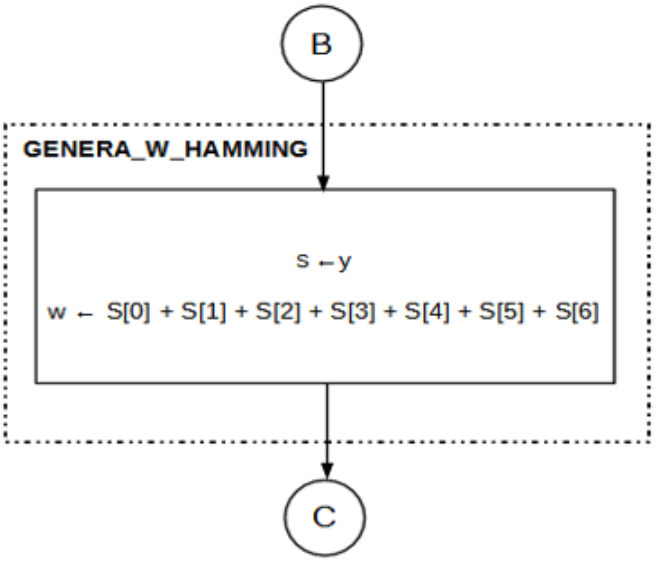

Figura 3.6: Estado GENERA_W_HAMMING

3.3.2. Localización del error

En el bloque EVALÚA_W, se tienen las siguientes 3 evaluaciones:

- La palabra R a decodificar tiene 1 o más errores.
- La palabra R a decodificar no posee error.
- El bit errado de la palabra se encuentra en los bits de paridad o en los de información.

Para el primer caso, mediante el código de MatLab, se obtuvo que siempre que w tome los valores 2, 4 y 6 será posible decodificar la palabra. Por otro lado, como se menciona en [10] el código BCH (63,56) puede corregir 1 bit y detectar 2 bits errados; sin embargo, en caso de introducir una palabra que contenga 3 bits errados, el error no será detectado por la propiedad cíclica de los códigos BCH [10].

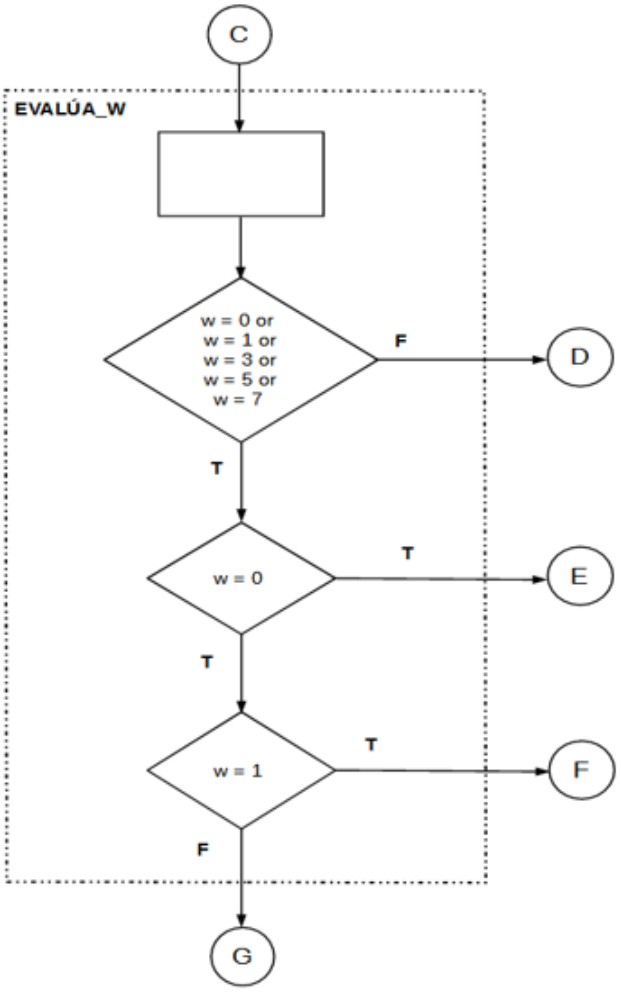

Figura 3.7: Estado EVALÚA_W

3.3.2.1. Palabra con 2 bits errados

Cuando R posea 2 errores se pasará al estado NO_CORRECCIÓN mostrado en la figura 3.8. En este la señal decoder_done toma un valor de 0 para mostrar que la decoficación no puede realizarse, y la señal ack toma valor de 1 lógico, ya que la solicitud (req) fue reconocida.

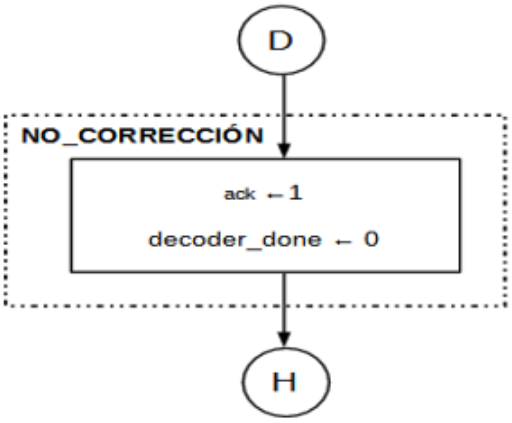

Figura 3.8: Estado NO_CORRECCIÓN

3.3.2.2. Palabra de entrada sin error

En la sección 3.3.1, se mencionó que los síndromes dependen directamente del patrón de errores; por lo tanto, si este patrón tiene como valor 0, el síndrome también será 0 y su peso de Hamming (w) de igual forma. En la figura 3.9, se muestra el estado NO_ERROR, en donde la palabra decodificada R_out será la palabra ingresada R. Además, la señal decoder_done será 1 lógico, ya que la decodificación se realizó con éxito y se reconoció la solicitud.

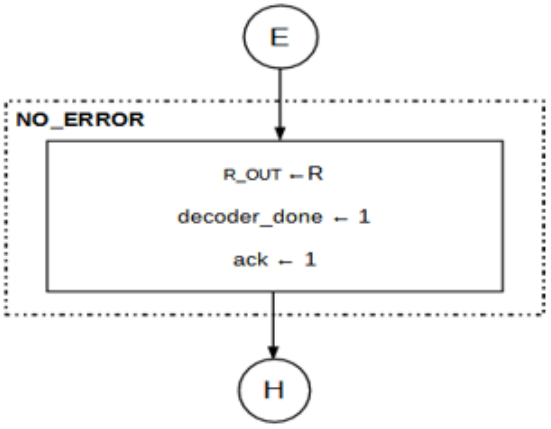

Figura 3.9: Estado NO_ERROR

3.3.2.3. Error en los bits de paridad

Luego de haber descartado que la palabra sea no corregible y que no posea errores, se tiene que R contiene 1 bit errado. Este bit se puede ubicar en los bits de paridad o en los de información.

El circuito de codificación y el del cálculo de síndromes poseen la misma estructura como se mostró en la sección 2.2.2. En la codificación se deben de agregar 7 bits de paridad para el control de errores, por tanto estos tienen una relación directa con la posición del mismo y el síndrome.

Mediante el código de MatLab, se obtuvo la siguiente relación, mostrada en la tabla 3.2, de la posición del bit errado (en los últimos 7 bits de la palabra R) y el valor del síndrome. Como se muestra en la tabla, solo se tiene 1 bit con valor '1' por cada posición, entonces el peso de Hamming de S tendrá siempre un valor de 1 cuando el error se encuentre en los bits de paridad.

Tabla 3.2: Valor de síndromes en relación a la posición del bit errado en los bits de paridad

SÍNDROME							**Posición del error en los bits de paridad**
1	0	0	0	0	0	0	1
0	1	0	0	0	0	0	2
0	0	1	0	0	0	0	3
0	0	0	1	0	0	0	4
0	0	0	0	1	0	0	5
0	0	0	0	0	1	0	6
0	0	0	0	0	0	1	7

Debido a que se conoce la posición del error en referencia a los últimos 7 bits, es necesario tener completar los 56 bits restantes con 0 lógico. Finalmente, se debe realizar la función XOR con la palabra de entrada R par así corregir el error; además, las señales decoder_donde y ack deben tomar el valor de 1 lógico ya que la decodificación fue exitosa y la solicitud fue reconocida. En la figura 3.10, se muestra lo anteriormente descrito.

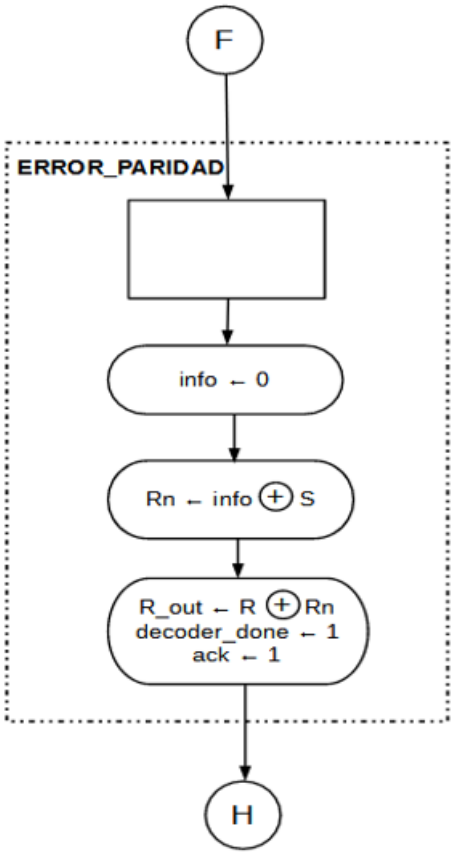

Figura 3.10: Estado ERROR_PARIDAD

3.3.2.4. Error en los bits de información

Cuando el peso de Hamming sea mayor a 1, significará que el error se encuentra en los bits de información, ya que se descartó en el bloque anterior el que se encuentre en los bits de paridad. En la tabla 3.3 se muestran los valores de síndrome para cada valor de posición del error en los bits de información, los cuales son los primeros 56 bits del total de 63.

Tabla 3.3: Valor de síndromes en relación a la posición del bit errado en los bits de información

Posición del error en los bits de información	**Síndrome**
1	1 1 0 0 0 1 0
2	0 1 1 0 0 0 1
3	1 1 1 1 0 1 0
4	0 1 1 1 1 0 1
5	1 1 1 1 1 0 0
6	0 1 1 1 1 1 0
7	0 0 1 1 1 1 1
8	1 1 0 1 1 0 1
9	1 0 1 0 1 0 0
10	0 1 0 1 0 1 0
11	0 0 1 0 1 0 1
12	1 1 0 1 0 0 0
13	0 1 1 0 1 0 0
14	0 0 1 1 0 1 0
15	0 0 0 1 1 0 1
16	1 1 0 0 1 0 0
17	0 1 1 0 0 1 0
18	0 0 1 1 0 0 1
19	1 1 0 1 1 1 0
20	0 1 1 0 1 1 1
21	1 1 1 1 0 0 1
22	1 0 1 1 1 1 0
23	0 1 0 1 1 1 1
24	1 1 1 0 1 0 1
25	1 0 1 1 0 0 0
26	0 1 0 1 1 0 0
27	0 0 1 0 1 1 0
28	0 0 0 1 0 1 1
29	1 1 0 0 1 1 1
30	1 0 1 0 0 0 1
31	1 0 0 1 0 1 0
32	0 1 0 0 1 0 1
33	1 1 1 0 0 0 0
34	0 1 1 1 0 0 0
35	0 0 1 1 1 0 0
36	0 0 0 1 1 1 0
37	0 0 0 0 1 1 1
38	1 1 0 0 0 0 1
39	1 0 1 0 0 1 0
40	0 1 0 1 0 0 1
41	1 1 1 0 1 1 0
42	0 1 1 1 0 1 1
43	1 1 1 1 1 1 1

Posición del error en los bits de información	Síndrome
44	1 0 1 1 1 0 1
45	1 0 0 1 1 0 0
46	0 1 0 0 1 1 0
47	0 0 1 0 0 1 1
48	1 1 0 1 0 1 1
49	1 0 1 0 1 1 1
50	1 0 0 1 0 0 1
51	1 0 0 0 1 1 0
52	0 1 0 0 0 1 1
53	1 1 1 0 0 1 1
54	1 0 1 1 0 1 1
55	1 0 0 1 1 1 1
56	1 0 0 0 1 0 1

Será necesario obtener un patrón de bits que permita corregir el error. Por ejemplo, si se tiene un error en la posición 27, el síndrome tendrá como valor 0010110; además, el patrón correspondería a 00000000000000000000000000010000000000000000000000000000. Es decir, se colocará un 1 en la posición del error y el resto serán 0 de manera similar a los síndromes obtenidos en la sección 3.3.2.3. Entonces colocando 1 por cada posición de error se obtendría una matriz identidad de 56x56 como se muestra en la figura 3.11

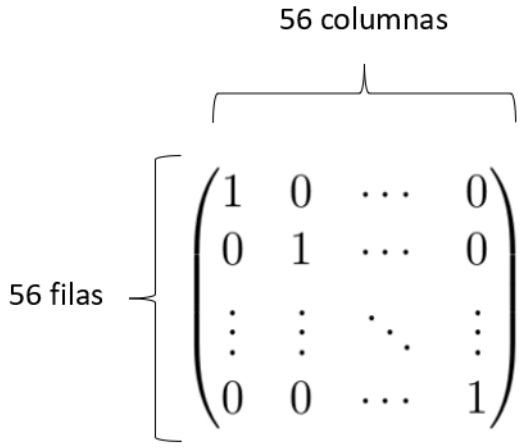

Figura 3.11: Matriz de patrón de bits a corregir

El valor de síndrome calculado en GENERA_SÍNDROME será comparado con los 56 valores de la tabla 3.3 y de acuerdo a este se le asociará un patrón de errores, es decir, se leerá la matriz fila por fila. Además, se deben agregar los últimos 7 bits con 0 lógico para realizar la corrección de R mediante la función XOR. Las señales decoder_done y ack tomarán el valor de 1 por la decodificación exitosa y el reconocimiento de la solicitud, esto es indicado en la figura 3.12.

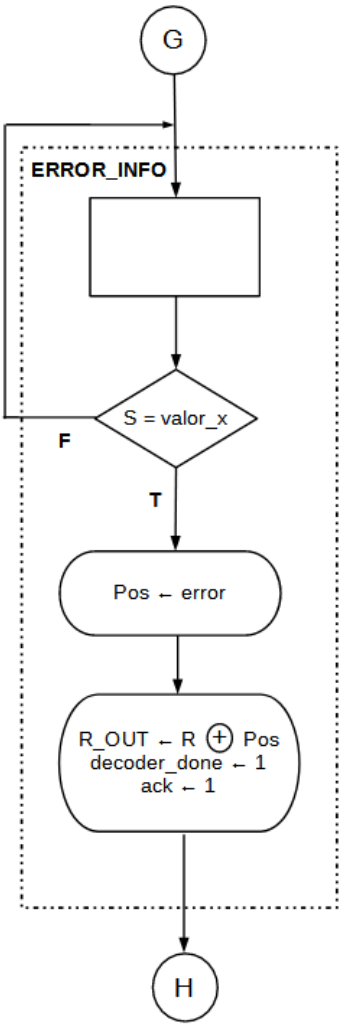

Figura 3.12: Estado ERROR_INFO

3.3.2.5. Estado de salida

Finalmente, luego de los bloques NO_CORRECCIÓN, NO_ERROR, ERROR_PARIDAD y ERROR_INFO, se pasará al estado ESPERA, mostrado en la figura 3.13, en donde se necesitará una nueva solicitud (req = 1 lógico) para iniciar la decodificación de otra palabra en el estado ST_INICIAL.

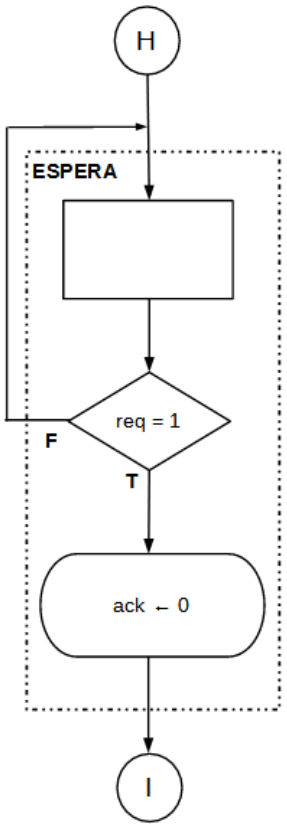

Figura 3.13: Estado ESPERA

3.4. Resultados de la simulación de bloques

En la tabla 3.4 se muestran los resultados de la simulación de los bloques funcionales del decodificador BCH (63,56)

Tabla 3.4: Resultados de simulación de bloques

Bloque	Máxima frecuencia de operación	Elementos lógicos	Registros
Cálculo de síndrome	280.82MHz	67	13
Localización del error	-	4	3
Error en bits de paridad	-	191	56
Error en bits de información	-	160	63

Capítulo 4

Simulaciones y resultados

En el presente capítulo se mostrarán los resultados del diseño de propuesta del decodificador BCH(63,56) basado en una máquina de estados algorítmica con datapath (ASM-D) para FPGA. Se corroborarán los mismos por medio del software Matlab, en el cual se elaboró un programa centrado en el pseudocódigo mostrado en [10].

4.1. Simulaciones

Para poder validar el correcto funcionamiento de la decodificación se deben de analizar los siguientes 4 casos:

- No se haya introducido error en la palabra a decodificar.
- Se introduzca 1 error en los bits de información.
- Se introduzca 1 error en los bits de paridad.
- Se introduzcan 2 errores en cualquier posición.

4.1.1. Palabra a decodificar sin error

Como se mencionó en el capítulo anterior, se consideran como entradas las señales clk, reset_n, req y R la cual es la palabra a decodificar, y como salidas ack, dc_dn (decoder_done) y R_out (palabra decodificada). En la figura 4.1 se muestran los resultados de esta primera simulación, donde se encerró en un recuadro amarillo la palabra de entrada y en un recuadro naranja la palabra decodificada. Ya que no se tienen errores, estas son iguales como se puede observar.

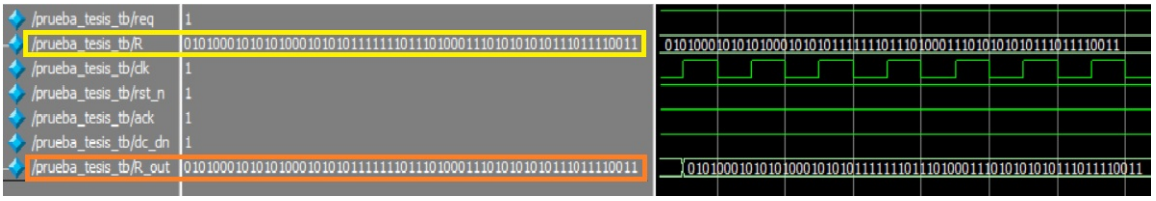

Figura 4.1: Decodificación de una palabra sin error

En el programa de Matlab al ingresar 0 errores se obtiene lo mismo, lo cual comprueba su correcto funcionamiento.

4.1.2. Error en los bits de información

En este caso se seleccionaron de forma aleatoria 2 posiciones de error en los bits de información para corroborar la decodificación. Se tendrá un error en el bit 14 (siendo el bit 1 el primero de la izquierda) y en el bit 49.

4.1.2.1. Error en el bit 14

En la figura 4.2, se muestra mediante el recuadro rojo el cambio del bit 14, es decir, la corrección del mismo en la palabra decodificada; además, esto fue corroborado también en Matlab, así como el resto de simulaciones.

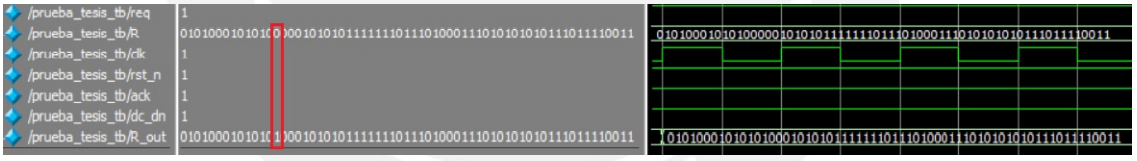

Figura 4.2: Decodificación de error en la posición 14

4.1.2.2. Error en el bit 49

En la figura 4.3, se muestra el cambio del bit 49 en R_out respecto a R, es decir, la corrección de la palabra de entrada con error.

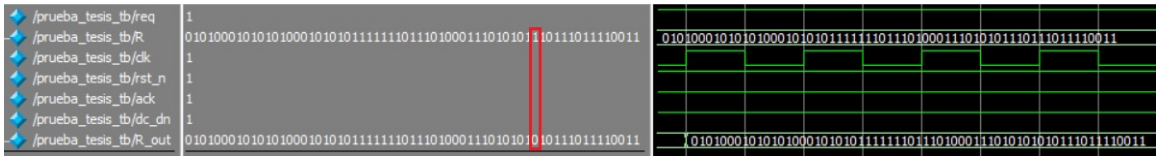

Figura 4.3: Decodificación de error en la posición 49

4.1.3. Error en los bits de paridad

En este caso se validará la corrección cuando el error se encuentre en los últimos 7 bits de la palabra de entrada, es decir, en los bits de paridad, donde el rango es [57-63]. Se tomaron aleatoriamente 2 valores: posición 58 y 63.

4.1.3.1. Error en el bit 58

En la figura 4.4 se muestra el recuadro azul donde se tiene el cambio de bit de 0 a 1 en R_out respecto a R, lo que indica la corrección del bit.

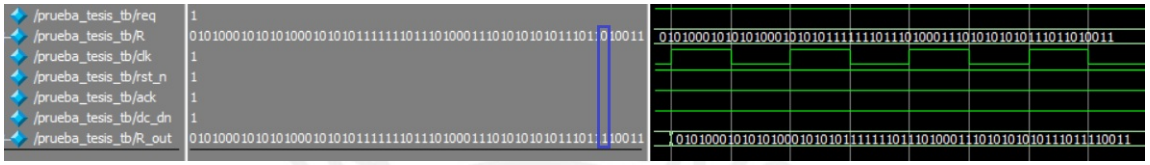

Figura 4.4: Decodificación de error en la posición 58

4.1.3.2. Error en el bit 63

En la figura 4.5, se observa el cambio del último bit de la palabra con 0 lógico a 1 lógico en la salida, obteniendo así la palabra codificada sin error.

Figura 4.5: Decodificación de error en la posición 63

4.1.4. 2 Errores en la palabra de entrada

Finalmente, se tiene el caso donde la palabra posea 2 errores, entonces el código BCH(63,56) es capaz de realizar su detección más no corrección. La señal dc_dn, mostrada en el recuadro celeste de la figura 4.6, tiene el valor de 0 lógico, lo que indica que la decodificación no fue realizada con éxito. Por otro lado, se muestra en el recuadro naranja el valor de la salida R_out, donde se observan 63 ceros. Lo anterior se realizó para indicar que la palabra de salida no fue decodificada, y por tanto, los errores no fueron corregidos.

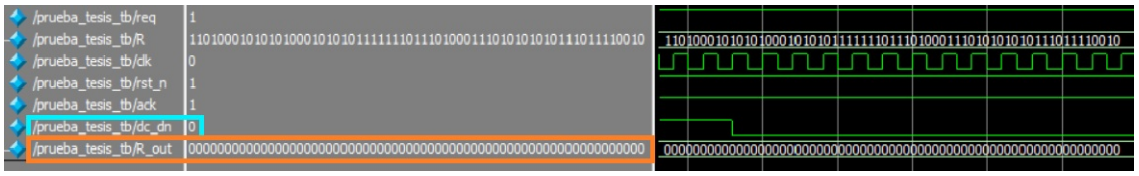

Figura 4.6: 2 errores introducidos en la palabra de entrada

4.2. Resultados del diseño de la arquitectura ASM-D

El diseño de la máquina de estados algorítmica con datapath (ASM-D) fue realizado en el software Quartus Prime 15.1. En la figura 4.7 se muestra el resultado de la síntesis de la arquitectura. El total de elementos lógicos son 360, equivalente a menos del 1 % de los elementos lógicos totales con lo que se comprueba el funcionamiento de la arquitectura diseñada. Además, en la parte inferior se observa la máxima frecuencia de operación obtenida.

Flow Summary

Flow Status	Successful - Sat Dec 05 01:23:45 2020
Quartus Prime Version	15.1.0 Build 185 10/21/2015 SJ Lite Edition
Revision Name	prueba_tesis
Top-level Entity Name	prueba_tesis
Family	MAX 10
Device	10M50DAF484C7G
Timing Models	Preliminary
Total logic elements	360 / 49,760 (< 1 %)
Total combinational functions	298 / 49,760 (< 1 %)
Dedicated logic registers	149 / 49,760 (< 1 %)
Total registers	149
Total pins	131 / 360 (36 %)
Total virtual pins	0
Total memory bits	0 / 1,677,312 (0 %)
Embedded Multiplier 9-bit elements	0 / 288 (0 %)
Total PLLs	0 / 4 (0 %)

Slow 1200mV 85C Model Fmax Summary

	Fmax	Restricted Fmax	Clock Name
1	160.54 MHz	160.54 MHz	clk

Figura 4.7: Resultados del diseño del decodificador BCH(63,56) por medio de ASM-D y máxima frecuencia de operación

Conclusiones

- Se realizó el diseño de una arquitectura en hardware para el decodificador BCH(63,56) basado en los bloques de cálculo de síndrome, búsqueda del error y corrección del mismo. En las figuras 4.1-4.6 se demuestra el cumplimiento de los objetivos del decodificador, los cuales son corregir 1 bit errado y detectar 2 bits errados.

- De acuerdo con el pseudocódigo de la referencia [10], se obtuvo un programa en el software Matlab que permitió adquirir la lista de síndromes mostrada en las tablas 3.2 y 3.3 con la cual se realizó la localización del error en los 63 bits de la palabra de entrada.

- Se realizó la síntesis comportamental del algoritmo de Matlab obteniendo una descripción basada en estados de una ASM-D, mostrados en las figuras 3.4 a la 3.13 con lo cual se buscó una optimización del diseño. Lo anterior se debe a que dependiendo del valor del síndrome y peso de Hamming se realiza la decodificación, caso contrario mediante la señal dc_dn se notifica la no corrección de la palabra. Con esto se evitan cálculos que generan mayor consumo de energía como el de la búsqueda de la posición del error.

- Adicionalmente, se utilizó el protocolo Handshake que permitió por medio de la señal req (requerimiento) realizar la decodificación, así como también mediante la señal ack el reconocimiento de la solicitud.

Recomendaciones y trabajo futuro

- Con el objetivo de obtener más simulaciones que permitan validar lo realizado en el software Matlab, se recomienda colocar en un archivo de texto cadenas de palabras de entrada (R) en las cuales se haya introducido error en las 63 posiciones. Este debe ser enviado como generación de estímulos para la simulación.

- La verificación del diseño de la arquitectura del decodificador se realizó por medio de Testbench, en donde se introdujeron distintas palabras tanto sin error como con 1 y 2 errores; sin embargo, es recomendable la verificación funcional del diseño mediante SystemVerilog. Lo anterior debido a que es comúnmente usado en la industria del diseño electrónico como una mejora respecto a Verilog.

- Si bien se verificó el funcionamiento de la arquitectura por medio de simulaciones, ya que la tesis se encuentra orientada a una implementación para nanosatélites, se considera como trabajo futuro la implementación de un ASIC. Esto puede ser realizado con el uso del software Cadence realizando además las verificaciones de reglas de diseño (del inglés Design Rule Check, nemónico DRC) y layout versus esquemático (del inglés Layout Versus Schematic, nemónico LVS).

- Se recomienda la implementación de decodificador BCH en un FPGA antifusible, debido a que estos poseen una baja sensibilidad a la radiación. Sin embargo, estos FPGA no son de fácil acceso, por lo que también se pueden implementar técnicas de tolerancia a fallos como los TMR Flip Flops (del inglés Triple Modular Redundancy) la cual presenta mayor confiabilidad en caso un rayo gamma impacte el FPGA. Con esto último se evitaría un cambio de estado involuntario en la ASM-D.

Bibliografía

[1] K. Hambleton, "Artemis I Map — NASA." [Online]. Available: https://www.nasa.gov/image-feature/artemis-i-map

[2] K. F. Robinson, S. F. Spearing, and D. Hitt, "NASA's Space Launch System: Opportunities for Small Satellites to Deep Space Destinations," *32nd Annual Small Satellite*, no. August, pp. 1–9, 2019. [Online]. Available: https://ntrs.nasa.gov/search.jsp?R=20180006368

[3] Instituto de radioastronomía INRAS - PUCP, "Satélites: PUCP-Sat-1 - PUCP — Instituto de Radioastronomía." [Online]. Available: https://inras.pucp.edu.pe/proyectos/pucp-sat-1/

[4] Balta, "ERROR CORRECTION ALGORITHMS IN SATELLITE COMMUNICATION – BCH ENCODING / DECODING IMPLEMENTATION on VHDL."

[5] M. Hatamian, H. Barati, S. Berenjian, A. Naghizadeh, and B. Razeghi, "Error Control Coding in Optical Fiber Communication Systems : An Overview," vol. 4, no. 2, pp. 70–80, 2015.

[6] R. Mehra, G. Saini, and S. Singh, "FPGA based high speed BCH encoder for wireless communication applications," *Proceedings - 2011 International Conference on Communication Systems and Network Technologies, CSNT 2011*, no. 2, pp. 576–579, 2011.

[7] Specialists Group on Coding for Visual Telephony, "Comparison Bose-Chaudhuri-Hocquenghem BCH and Reed Solomon."

[8] C. Engineering, "VHDL IMPLEMENTATION OF REED-SOLOMON CODING."

[9] CCSDS, "The Consultative Committee for Space Data Systems (CCSDS)." [Online]. Available: https://public.ccsds.org/default.aspx

[10] S. Arunkumar and T. Kalaivani, "FPGA implementation of CCSDS BCH (63, 56) for satellite communication," in *International Conference on Electronic Devices, Systems, and Applications*, 2012, pp. 248–253.

[11] C. P. S, “Instroduction to Bose Chaudhuri Hocquenghem codes,” 1967.

[12] B. Chen, “Hardware Implementation of Error Control Decoders,” 2008. [Online]. Available: http://rave.ohiolink.edu/etdc/view?acc_num=case1209531418

[13] L. Alejos, “Introducción a Tecnologías de Redes vía Satélite Cobertura geográfica,” 2011.

[14] F. R. Lone, A. Puri, and S. Kumar, “Performance Comparison of Reed Solomon Code and BCH Code over Rayleigh Fading Channel,” *International Journal of Computer Applications*, vol. 71, no. 20, pp. 23–26, 2013.

[15] F. D. E. El and D. B. D. Carvalho, “Dois Códigos Corretores de Erro BCH para Comunicacoes Ópticas: Implementacao em FPGA e Comparacoes,” 2015.

[16] Standford, “GF(2m) arithmetic: summary,” Tech. Rep.

[17] Consultative Committee for Space Data Systems (CCSDS), “TC Synchronization and Channel Coding Recommendation for Space Data System Standards,” *Ccsds 231.0-B-3*, no. 3, 2017.

[18] M. Z. Hasan, M. A. Akbar, and I. Mahmood, “Performance Analysis of (63,56) Bch Code Using Multipath Rayleigh Fading Channel on Spartan-3 FPGA,” vol. 2, p. 5, 2008.

[19] X. Zhang, *VLSI Architectures for Modern Error-Correcting Codes*, 2015.

[20] V. P. Mahadevaswamy, S. L. Sunitha, and B. N. Shobha, “Implementation of Fault Tolerant Method Using BCH Code on FPGA,” no. 4, pp. 180–182, 2012.

[21] S. JasamMohammed and H. Fadhil Abdulsada, “FPGA Implementation of 3 bits BCH Error Correcting Codes,” *International Journal of Computer Applications*, vol. 71, no. 7, pp. 35–42, 2013.

[22] H. F. Abdulsada, “Design and Implementation of 2 bits BCH Error Correcting Codes using FPGA,” no. October, 2017.

[23] B. Jarvis, “FPGA Implementation and Analysis of Error Correction Codes for Physical Unclonable Functions,” vol. 2, pp. 1–6.

[24] D. V. Sarwate and N. R. Shanbhag, “High-Speed Architectures for Reed – Solomon,” vol. 9, no. 5, pp. 641–655, 2001.

[25] S. Yoon and H. Lee, “A discrepancy-computationless RiBM algorithm and its architecture for BCH decoders,” *2008 IEEE International SOC Conference, SOCC*, pp. 379–382, 2008.

[26] S. An, H. Tang, and J. Park, “A inversion-less peterson algorithm based shared KES architecture for concatenated BCH decoder,” *ISOCC 2015 - International SoC Design Conference: SoC for Internet of Everything (IoE)*, pp. 281–282, 2016.

[27] M. Raffo, “Material del curso de la PUCP 1IEE04 - Diseño Digital,” 2019.

[28] D. Green, *Modern Logic Design*, 1986.