INSTITUTO TECNOLÓGICO DE SONORA

Educar para Trascender

“Arquitectura en hardware de un detector OSIC para receptor SISO V2V”

Tesis

Que para obtener el título de

Maestro en Ciencias de la Ingeniería

Presenta

Dagoberto Alvarez Ibarra

Ciudad Obregón, Sonora; Octubre de 2020

Resumen

Índice

Lista de Figuras **ix**

Lista de Tablas **xi**

I Introducción **1**

1.1 Antecedentes 1
1.2 Planteamiento del problema 6
1.3 Objetivo 6
1.4 Hipótesis 7
1.5 Justificación 7
1.6 Alcances 7
1.7 Limitaciones 8

II Marco Teórico **9**

2.1 Estándar IEEE 802.11p 9
2.2 Asignación de pilotos 802.11p 11
2.3 Canal de comunicación V2V 12
2.4 Modelo del sistema de V2V 14
2.5 Detección en el receptor de los datos transmitidos 16
2.5.1 Modelo del sistema receptor 16
2.5.2 Modelo del detector OSIC basada en la descomposicion QR . . . 18
2.5.3 Detección QR-ML convencional 19
2.5.4 Detector V2V Near ML 21
2.6 Implementaciones en hardware 22
2.7 FPGA's y arquitectura 23
2.8 Métricas de desempeño 24
2.9 Modelo algorítmico 25
2.9.1 Procedimiento de mapeo de algoritmo a arquitectura 25

2.9.1.1 Generación del modelo de oro . . . 26
2.9.1.2 Adecuaciones al algoritmo . . . 26
2.9.1.3 Análisis de punto fijo . . . 27
2.9.1.4 Diseño de la arquitectura de hardware . . . 29
2.9.1.5 Implementación y verificación de la arquitectura . . . 30

III Método **33**
3.1 Sujeto . . . 33
3.2 Procedimiento . . . 33
3.3 Herramientas . . . 35

IV Modulación espacial en cuadratura (QSM) **37**

V Desarrollo **39**
5.1 Desarrollo del modelo algorítmico . . . 39
5.1.1 Análisis en punto fijo . . . 40
5.2 Arquitectura propuesta . . . 44
5.2.1 Parametrización . . . 44
5.2.2 Módulo *Top* . . . 45
5.2.3 **Div_Factor** . . . 47
5.2.3.1 **LUT_Factor** . . . 49
5.2.3.2 **Divisor** . . . 49
5.2.4 **Hard_Demapper** . . . 50
5.2.4.1 **Dec_Cuadrante** . . . 51
5.2.4.2 **ABS** . . . 52
5.2.4.3 **LUT_Compara** . . . 52
5.2.4.4 **Comparador** . . . 52
5.2.4.5 **Concat_Select** . . . 53
5.2.5 **Symbol_Detector** . . . 53
5.2.5.1 **Get** . . . 55
5.2.5.2 **ROM_Conste_real** y **ROM_Conste_imag** . . . 56
5.2.6 **Arit_Oper** . . . 56
5.2.6.1 **RAM** . . . 59

5.2.6.2 **Complex mult** . . . 59
5.2.6.3 **Reg** . . . 59
5.2.6.4 **Bloque menos** . . . 59
5.2.6.5 **Sel** . . . 60
5.2.7 **RA** . . . 60
5.2.8 **CTRL** . . . 61
5.2.8.1 **CTRL1** . . . 61
5.2.8.2 **CTRL2** . . . 61

VI Resultados **63**
6.1 Resultados de implementación . . . 63
6.2 Resultados de operación . . . 65
6.3 Resultados de verificación . . . 66

VII Conclusiones **69**

Lista de Figuras

1 Estructura del preámbulo y trama del estándar 802.11p 11
2 Esquema de asignación de pilotos en 802.11p 11
3 Diagrama a bloque simplificado, de un sistema de comunicación en V2V en banda base 15
4 Estructura del árbol de búsqueda bajo el criterio ML 20
5 Estructura del árbol de búsqueda bajo el criterio Near ML 21
6 Diagrama de flujo del procedimiento a seguir 34
7 Análisis de punto fijo del algoritmo. QPSK. 42
8 Análisis de punto fijo del algoritmo. 16-QAM. 43
9 Análisis de punto fijo del algoritmo. 64-QAM. 44
10 *Top* del detector 46
11 Arquitectura de **Div_Factor** 48
12 Arquitectura de **Hard_Demapper** 50
13 Arquitectura de **Symbol_Detector** 54
14 Arquitectura de **Arit_Oper** 57
15 Arquitectura de **RA** 60
16 Esquema de verificación para las arquitecturas de hardware 66
17 Comparación de BER en punto fijo vs arquitectura para QPSK 67
18 Comparación de BER en punto fijo vs arquitectura para 16-QAM 68
19 Comparación de BER en punto fijo vs arquitectura para 64-QAM 68

Lista de Tablas

1 Parámetros de configuración del estándar IEEE 802.11p 12
2 Características de seis escenarios V2V 13
3 Entradas y salidas del detector 46
4 Entradas y salidas de **Div_Factor** 48
5 Contenido de **LUT_Factor** 49
6 Entradas y salidas de **Div_Factor** 51
7 Contenido de **LUT_Compara** 52
8 Entradas y salidas de **Symbol_Detector** 54
9 Comportamiento de **Get** 55
10 Direcciones utilizadas en **ROM_Conste_real** y **ROM_Conste_imag** . . . 56
11 Entradas y salidas de **Arit_Oper** 57
12 Entradas y salidas de **RA** 60
13 Recursos utilizados para distintas constelaciones del detector 64

CAPÍTULO I

Introducción

1.1 Antecedentes

La industria automotriz es uno de los sectores que se encuentra en constante desarrollo, los vehículos automatizados brindan algunos beneficios, tales como mejor seguridad y comodidad, una menor gestión en tráfico, impacto ambiental, gastos, entre otros [1].

Bastantes aplicaciones de tecnología vehículo a vehículo ("Vehicle to Vehicle" por sus siglas en inglés V2V) se encuentran en desarrollo en la actualidad, a pesar de esto, los recursos utilizados para soportar estas aplicaciones no se encuentran disponibles y están progresando muy lentamente [2].

Entre algunas de las aplicaciones más utilizadas en los sistemas V2V están [3]

- Advertencias a vehículos alrededor acerca de pérdidas de control.
- Advertencias de emergencias vehiculares.

- Información acerca de las condiciones de los caminos.
- Advertencias de proximidad de vehículos.
- Acciones pre-colisiones.
- Asistencia de giros hacia la izquierda.
- Asistencia con movimientos en intersecciones.
- Advertencias sobre puntos ciegos.
- Advertencias de cambio de carril.

Las aplicaciones mencionadas necesitan que sean de alta seguridad, por medio de mensajes recibidos desde otros vehículos, estos deben de ser confiables, y con la menor demora posible para la transmisión e intercambio de información vehículo a vehículo.

En V2V se utilizan redes de comunicaciones directas de rango corto (Direct Short-Range Communications por sus siglas en inglés DSRC), que están relacionadas con el estándar IEEE 802.11p, el cual consiste en el uso de técnicas de radio para la transferencia de datos en distancias cortas entre unidades de radio fijas y móviles [4].

A continuación se muestran algunas de las fortalezas de las redes DSRC utilizadas en V2V:

- Enlaces y canales dedicados.
- Baja latencia.
- Alta seguridad.

En cuanto a las debilidades, se encuentran:

- Un despliegue limitado de infraestructura.
- Un cobro por el despliegue y mantenimiento de una infraestructura dedicada.
- La falta de un protocolo de transferencia IP (Internet Protocol).

Estas comunicaciones vehiculares conllevan problemas, puesto que las comunicaciones inalámbricas son complicadas per se, más aún cuando el factor de movimiento es agregado entre el transmisor y receptor a velocidades relativamente altas. Esto debido a que se tiene que tomar en cuenta los efectos de pérdida de trayectoria, retardo de dispersión, efecto Doppler, dispersión, entre otros, mientras la calidad de transmisión se mantiene [5].

Las características de un canal de comunicaciones V2V son significativamente diferentes a los canales celulares, más específicamente en términos de pérdidas de trayectoria y selectividad tanto en tiempo como en frecuencia, esto debido a ciertas particularidades de la propagación de radio en V2V [6].

En V2V el transmisor y receptor se encuentran a la misma altura y en ambientes similares. En sistemas celulares, la comunicación se realiza entre una estación base que se encuentra a una altura mayor con respecto a la estación móvil. Consecuentemente, los mecanismos de propagación dominantes en multi-trayectorias son distintos. Por ejemplo, la propagación de ondas sobre tejados en comunicaciones celulares es importante, mientras que en sistemas V2V se da en el plano horizontal. En comunicaciones celulares el área entre la estación base y la estación móvil se encuentra libre de dispersores, por otro lado en los sistemas V2V la dispersión ocurre entre el transmisor y receptor. Adicionalmente, la distancia en la que se lleva a cabo la transmisión es distinta, menor a 100 metros para V2V y alrededor de 1 kilómetro comunicaciones celulares.

Las comunicaciones V2V generalmente operan a 5.9 GHz, mientras que las comunicaciones celulares se lleva a cabo entre 700 y 2100 MHz, lo cual conlleva una mayor de atenuación de la señal en los sistemas V2V.

El uso de la modulación OFDM (por sus siglas en inglés “Orthogonal Frequency Division Multiplexing”) en los sistemas de comunicación de nueva generación, tales como V2V ha aumentado, debido a que proporcionan una mayor inmunidad a los desvanecimientos del canal por multitrayectoria. Un sistema OFDM convierte un canal con desvanecimiento selectivo en frecuencia, en múltiples canales con desvanecimientos con cierta banda de coherencia por medio del uso de subportadoras con frecuencias

múltiples. Estas portadoras deben de separarse por al menos el inverso de la frecuencia de muestreo para mantener la ortogonalidad [7].

Existen varias arquitecturas que definen técnicas de detección de símbolos en el sistema receptor, cada una con un desempeño particular y complejidad de implementación. Una forma de realizar la detección es mediante la anulación combinatoria lineal para satisfacer los criterios de MMSE y ZF (de sus siglas en inglés “Minimum Mean Square Error” y “Zero Forcing”, respectivamente) [8]. Uno de los algoritmos más utilizados en altos ordenes de modulación es la cancelación de interferencia sucesiva ordenada (OSIC). En ella, el proceso de cancelación busca al candidato más probable para la detección del símbolo recibido [9].

En este contexto, el algoritmo de detección de ML (por sus siglas en inglés “Maximum Likelihood”) proporciona una tasa de error de bit mínima (BER por sus siglas en inglés “Bit Error Rate”) al buscar todos los posibles candidatos, sin embargo, su complejidad computacional crece de manera exponencial a medida que aumenta la cantidad de símbolos OFDM o el nivel de modulación utilizada. Con la finalidad de disminuir la complejidad, se proponen técnicas de descomposición QR (ordenadas o no ordenadas), tales algoritma generalmente incluyen el prefijo QRD (QR Descomposition) [10].

Como puede notarse, las comunicaciones inalámbricas se encuentran en constante desarrollo, pero mucha de la investigación ha sido enfocada al plano teórico-algoritmico y son pocos los trabajos experimentales dedicados a la factibilidad de implementación en términos de rendimiento y consumo de área. De forma que la necesidad de investigación para logar prototipados rápidos y eficientes en tecnología de última generación para la evaluación en tiempo real de los algoritmos se ha convertido en tópico interesante de investigación. Así mismo, dentro del proceso de diseño es crucial el uso de herramientas computacionales para modelar sistemas en aritmética de precisión finita. Hoy en día, MATLAB es la herramienta de modelado preferida preferida por gran parte de los investigadores [11]. Como consecuencia, MATLAB implementa funciones y operaciones en punto fijo, lo que conlleva que se pueda emular el comportamiento de un sistema digital. Esto ha hecho que la simulación y comprobación de algoritmos utilizando aritmética de punto fijo y posterior implementación en una plataforma con

tecnología como lo son los FPGA´s (“Fiel Programmable Gate Array”) está promovido por el gran incremento y refinaciones de técnicas que se han desarrollado en la industria de los semiconductores [12].

La contribución de diferentes arquitecturas de algoritmos de detección es amplia, sin embargo, su modelado en hardware, simulaciones e implementaciones son una buena oportunidad de contribución. Por tal razón, los FPGAs son la plataforma principal para el desarrollo de sistemas embebidos [13].

Por otro lado, en la literatura existen varias técnicas para la detección de símbolo utilizando un detector OSIC, entre ellos están los más destacados que son los siguientes:

Rawal et al. Investigó el desempeño de la QR basado en un detector OSIC en un espacio libre óptico (FSO) bajo sistemas de comunicación por multiplexación espacial (SM) en un sistema de múltiples entradas y múltiples salidas (MIMO) [14]. Esta propuesta es un detector no lineal con una baja complejidad de detección, utilizando modulaciones BPSK y 16-QAM, comparando el desempeño en términos de BER con otras técnicas de detección QR-OSIC, pero en otros ambientes, ya que en MIMO FSO aún no han sido estudiados, al igual que la propuesta presentada en este documento, pero en sistemas SISO-OFDM.

Del Puerto-Flores et al presenta dos artículos relacionados con sistemas V2V utilizando la modulación OFDM, del cual se obtuvo el modelo del sistema para la cual fue diseñada la arquitectura planteada en el desarrollo de este documento. En uno de los artículos se evalúan los sistemas OFDM con portadoras virtuales (VC) bajo sistemas V2V, donde se muestra un detector OSIC con un mejor desempeño en términos de BER en comparación a la no utilización de VC [15]. En el segundo artículo, se evalúa un detector OSIC, pero en un sistema de dispersión por transformada rápida de Fourier (DFTS)-OFDM mostrando resultados favorables para dar pauta al trabajar con sistemas SISO-OFDM [16].

1.2 Planteamiento del problema

En un sistema de comunicación V2V SISO-OFDM, particularmente en el receptor, el uso del método ML de detección de símbolo, a pesar de ser óptimo, es exhaustivo y por ende con una complejidad computacional grande.

Entonces, ¿Que adecuaciones son necesarias al algoritmo original que permitan implementarlo de manera digital en un FPGA y que permitan evaluar métricas que definan su desempeño en términos de: consumo de recursos, SQNR y BER, al mismo tiempo con tasas de transmisión elevadas?

1.3 Objetivo

Desarrollar una arquitectura digital en FPGA con aritmética de precisión finita para un detector de símbolo basado en el algoritmo OSIC de un receptor V2V SISO-OFDM, evaluando métricas de desempeño funcional y de consumo área/potencia que permitan establecer su factibilidad de implementación.

Objetivos específicos:

- Modelar el esquema completo entre el receptor y transmisor en la plataforma de MATLAB.
- Realizar una análisis del algoritmo, para poder representar sus variables en punto fijo.
- Desarrollar una arquitectura digital e implementar los bloques requeridos en lenguaje de descripción de hardware (HDL) Verilog.
- Verificar la arquitecrua a traves de simulaciones que permitan evaluar su rendimiento bajo métricas de: consumo de área, funcionales y de desempeño; y comparar sus resultados respecto a los obtenidos con los modelos de simulación en punto flotante.
- Proponer una arquitectura integral *(full-hardware custom)* para nuestra diseño.

1.4 Hipótesis

Será posible implementar en FPGA un sistema de detección de símbolo compacto bajo el esquema de modulación OFDM y la técnica OSIC que presente un buen desempeño.

1.5 Justificación

El presente trabajo realizará la implementación en hardware de un algoritmo de detección de símbolos para sistemas vehículo a vehículo utilizado un sistema SISO-OFDM. El algoritmo de detección utilizado es OSIC, el cual se implementará en una tarjeta FPGA para comparar los resultados obtenidos con las simulaciones realizadas en aritmética de punto fijo.

EL algoritmo ya mencionado no ha sido tan investigado y además hasta el momento no existe un registro de su implementación en hardware ya aplicado en sistemas reales. Los investigadores de sistemas de telecomunicaciones y hardware, serán los beneficiados, ya que quedará un registro escrito del proyecto, etapas y resultados obtenidos, los cuales podrán servir como referencia para sistemas 5G que se deseen realizar en un futuro.

Finalmente, se dará pauta para que éste se pueda implementar en silicio, es decir, en un circuito integrado (C.I.) de uso comercial para aplicaciones V2V. Para que en consecuencia, la complejidad de los equipos utilizados hoy en día disminuya.

1.6 Alcances

El presente trabajo no busca realizar modificación al algoritmo original de detección de símbolo OSIC, más bien pretende exponer las adecuaciones al mismo para su translación a hardware, en este caso una plataforma de desarrollo FPGA. Así mimo, tampoco se pretende realizar su síntesis a un “chip” de silicio.

1.7 Limitaciones

Para la realización del proyecto como la mayoría de los trabajos el tiempo es una limitante apremiante. Así mismo, el costo de la tarjeta FPGA es elevado y la poca disponibilidad al laboratorio por consecuencia de la pandemía por Covid-19.

CAPÍTULO II

Marco Teórico

El proposito de este capítulo es presentar la investigación realizada en la literatura, conocimientos necesarios para la comprensión del presente trabajo, así como el análisis de trabajos de otros autores sobre el área de estudio. En el capítulo se presentan fundamentos de un canal de comunicación V2V, algoritmos de detección de símbolo, representaciones aritméticas en punto fijo, implementación en hardware, teoría sobre FPGA, y métodos para evaluar parámetros de desempeño.

2.1 Estándar IEEE 802.11p

En el año 2010, la comunicación V2V fue estandarizada con el nombre de DSRC IEEE 802.11p [17]. Este estándar utiliza una capa física (PHY) basada en el estándar IEEE 802.11a, modificando el ancho de banda de la señal a 10 MHz y la frecuencia central

a 5,9 GHz, pero manteniendo el esquema de OFDM. El estándar 802.11a fue desarrollado originalmente para entornos en interiores, los cuales son relativamente estacionarios; por lo tanto, debido a que estas formas de onda se transmiten bajo entornos V2V con alta movilidad, el rendimiento (en términos de confiabilidad) de los sistemas de comunicación V2V se alejan de su desempeño deseable.

La estructura del preámbulo del 802.11p difiere del 802.11a, teniendo como principal diferencia la duración del símbolo, el cual es duplicado de 16 μs a 32 μs. Como se muestra en la Figura 1, cada trama transmitida consta de un preámbulo que incluye dos símbolos de entrenamiento, el campo de señal y el campo de datos. Los símbolos de entrenamiento cortos (10 símbolos, cada uno de 1,6 μs de duración) se encuentran al comienzo de la trama y se utilizan para realizar la sincronía del sistema. Posteriormente, se transmiten dos símbolos de entrenamiento largos (cada uno de 6,4 μs de duración) utilizados para la sincronización fina y en la estimación del canal. La parte restante de la trama se utiliza para el campo de SEÑAL (símbolo de encabezado), contiene información sobre la longitud de la trama, la modulación y los esquemas de codificación utilizados en los siguientes símbolos de carga útil; se codifica con el código más robusto y se modula usando la constelación BPSK. El receptor debe decodificar y analizar esta información antes de comenzar a decodificar el primer símbolo de carga útil. Por último, el campo de DATO (carga útil) de longitud variable contiene la información a transmitir y al igual que el campo de SEÑAL deben ir separados por un intervalo de guarda (GI). Dependiendo de la modulación de datos utilizada, el estándar IEEE 802.11p puede llegar a admitir varias velocidades de transmisión de datos que van desde los 3 a 27 Mbps.

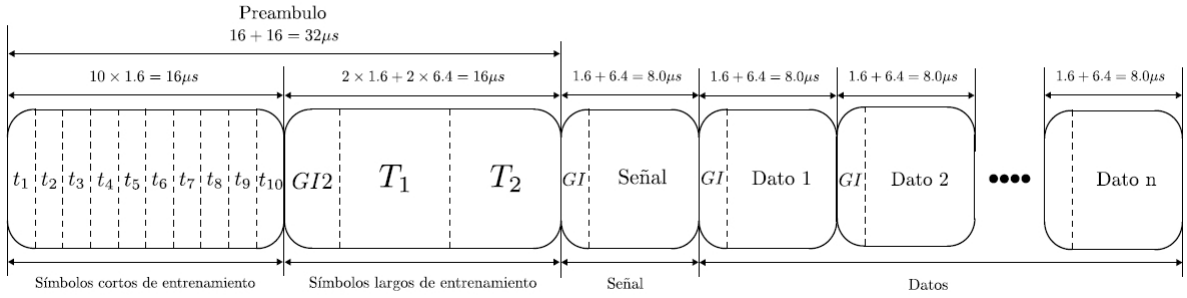

Figura 1: Estructura del preámbulo y trama del estándar 802.11p

2.2 Asignación de pilotos 802.11p

Las condiciones de alta movilidad presentes en ambientes V2V ocasionan una respuesta al impulso del canal (CIR) variante en el tiempo. Se asume que las variaciones temporales de los coeficientes del canal se darán por cada símbolo OFDM que componen a cada trama transmitida. La capa física de 802.11p utiliza 64 subportadoras por símbolo OFDM, incluyendo 48 subportadoras de datos, 4 subportadoras pilotos colocadas en los índices -21, -7, 7 y 21 como se observa en la Figura 2; pertenecientes al conjunto 0,1 utilizados para la estimación del canal, 11 subportadoras virtuales de valor nulo, utilizadas como guardas y una subportadora de DC, para realizar una transformada rápida de Fourier inversa (IFFT) de 64 muestras en la transmisión.

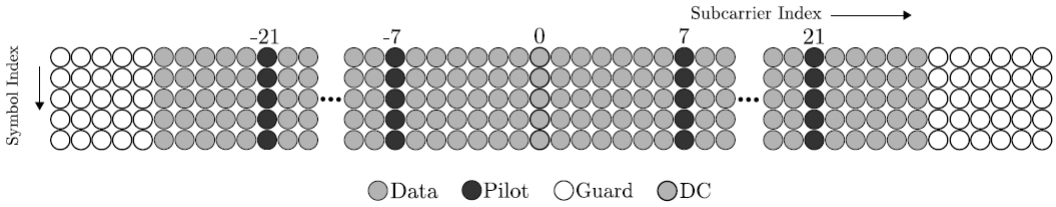

Figura 2: Esquema de asignación de pilotos en 802.11p

Como se demuestra en [18], la asignación de pilotos mencionada no es la más adecuada para la estimación del canal V2V; sin embargo, el uso de receptores con esquemas

de estimación de canal iterativos y detección de datos basadas en la técnica de mínimo error cuadrático medio (MMSE) en el dominio de la frecuencia (FD), logran tener un funcionamiento aceptable por arriba de la tercera iteración. Los parámetros utilizados por el estándar IEEE 802.11p se muestran en la Tabla 1.

Tabla 1: Parámetros de configuración del estándar IEEE 802.11p

Parámetro	**Valor**
BW(MHz)	10
Velocidad de transmisión (Mbit/s)	3, 4.5, 6, 9, 12, 18, 24, 27
Esquema de modulación	BPSK, QPSK, 16 QAM, 64QAM
Razón de código	1/2, 2/3, 3/4
Subportadoras de datos	48
Subportadoras pilotos	4
Tamaño de FFT	64
Periodo FFT (us)	6.4
Prefijo cíclico (us)	1.6

2.3 Canal de comunicación V2V

Los canales de comunicación, especialmente hablando para V2V, se han investigado bastante [2]. En [19] y [20] se expone una clasificación de los escenarios V2V posibles, dentro de los escenarios mas representativos encontramos: V2V-Autopista con dirección opuesta, V2V-Urbano sobre avenida principal con dirección opuesta, V2V en calle sub-urbano, V2V en autopista, V2V autopista en la misma dirección, V2V Urbano avenida principal. La Tabla 2 expone las características encontradas de cada uno de los escenarios reportados en [19].

Tabla 2: Características de seis escenarios V2V

Escenario	**Velocidad (Km/h)**	**Desplazamiento Doopler (Hz)**	**Retardo máximo (μs)**
V2V autopista, dirección opuesta	104	1000-1003	0.3
V2V urbano av. principal, dirección opuesta	32-48	300	0.5
V2V autopista	104	600-700	0.4
V2V urbano av. principa, misma dirección	32-48	400-500	0.4
V2V sub-urbano calle, misma dirección	32-48	300-500	0.7
V2V autopista, misma dirección	104	900-1150	0.7

Como se puede encontrar en la literatura, un modelo de canal simple no es capaz de modelar con precisión el entorno V2V. Por lo tanto, se utilizan modelos de canales estacionarios basados en el conocido modelo *Tapped Delay Line* (TDL) logrando replicar la variación temporal de los canales V2V. La CIR será variante durante el periodo de símbolo OFDM, es notable que la característica de variación temporal del canal dará lugar a la interferencia entre portadoras (ICI), lo que degradará en gran medida el rendimiento de la modulación OFDM [21].

Bajo el conocimiento del canal V2V doblemente selectivo, se debe tomar en cuenta que la colocación de símbolos piloto en la cuadrícula de tiempo-frecuencia OFDM, es de crucial importancia. Al diseñar un sistema inalámbrico, los símbolos de piloto adyacentes en la cuadrícula de tiempo-frecuencia deben cumplir algunos requisitos.

Específicamente, el espaciamiento entre pilotos máximo $\triangle f$ (numero de subportadoras) en el dominio de la frecuencia está determinado en función del retardo máximo τ_{max} del canal:

$$\triangle f \leq \frac{N}{\tau_{max} B}, \tag{1}$$

donde N es el número de subportadoras del símbolo OFDM igual a 64, y B es el ancho de banda del sistema igual a 10 MHz. Por otro lado, el espacio máximo $\triangle t$ (número de símbolos OFDM) debe de satisfacer lo siguiente:

$$\triangle t = \frac{B}{2 f_d (N + G)}, \tag{2}$$

donde G representa la duración del intervalo de guarda igual a 16 subportadoras. Por lo tanto, la asignación de pilotos de 802.11p (Figura 2) no resulta adecuada para realizar el proceso de estimación de canal.

2.4 Modelo del sistema de V2V

La estructura básica de todo sistema de comunicación V2V, se observa en la figura 3, de acuerdo a la Figura 3, el sistema recibe datos digitales de una fuente de información y ser procesados en el decodificador convolacional. El encargado de añadir redundancia a la secuencia de datos a transmitir y mejorar el desempeño de detección en el codificador de canal. El interleaver es el encargado de descorrelacionar la secuencia de bits generada para disminuir los errores de los bits. Posteriormente, el modulador selecciona bloques de *m* bits para el mapeo a un símbolo s $\in \Omega$, donde Ω es el conjunto de la constelación utilizada por el estándar 802.11p. Después, se insertan los símbolos pilotos pertinentes, realizando la formación del símbolo OFDM indicadas por el estándar. El bloque IFFT, modula cada bloque en 64 muestras. La siguiente etapa anexa el prefijo cíclico (CP) para mitigar la inferencia entre símbolos (ISI) OFDM. Finalmente, se añade el preámbulo de la trama V2V.

2.4. Modelo del sistema de V2V

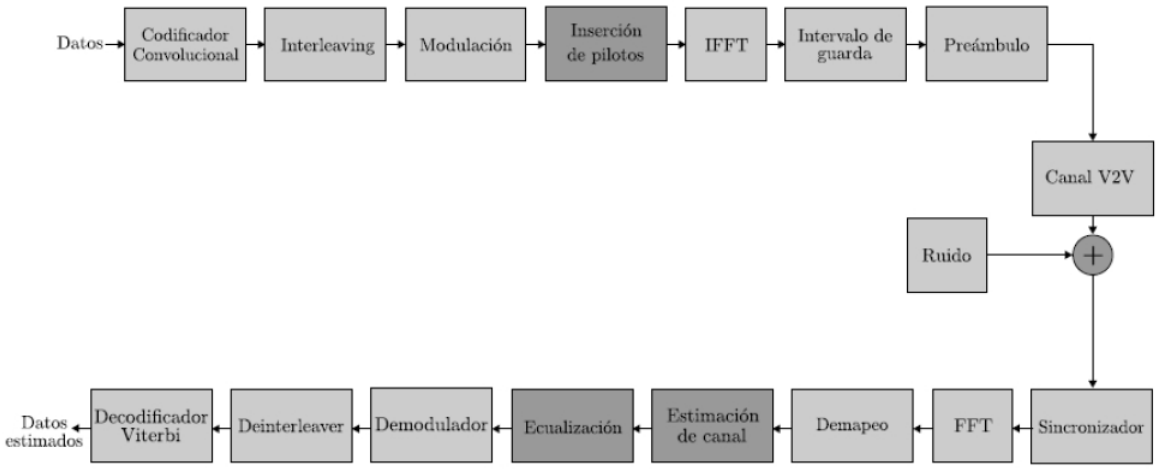

Figura 3: Diagrama a bloque simplificado, de un sistema de comunicación en V2V en banda base

En la parte del receptor, primeramente, se sincroniza la trama y el símbolo del sistema para ejecutar la extracción del CP y mitigar la interferencia intersimbólica (ISI). Después, el símbolo OFDM es procesado para el demapeador, el cual es el encargado de separar las subportadoras de datos de los pilotos. La estimación de canal y la ecualización, representan el proceso de mayor importancia en la recepción, el desempeño de estas dos etapas influye directamente con el desempeño del sistema de comunicación V2V. EL estimador de canal, estima la CIR variante con ayuda de los símbolos pilotos transmitidos dentro del símbolo OFDM. Como consecuencia, la ecualización realiza la igualación de canal con ayuda del canal estimado para reducir las distorsiones en los datos recibidos por el canal de comunicación, este bloque entrega símbolos complejos estimados que serán demodulados, ya sea por la decisión dura o suave. El deinterliver desentrelaza la trama y así por último el decodificador viterbi corrige los errores de bits de los datos detectados.

2.5 Detección en el receptor de los datos transmitidos

2.5.1 Modelo del sistema receptor

En un receptor V2V SISO-OFDM, la señal recibida para el *k-ésimo* símbolo en su representación compleja en banda base, queda expresada por la siguiente convolución circular discreta:

$$y^k[n] = \sum_{l=0}^{L-1} h^k[n,l]x^k[\langle(n-l)_N\rangle] + w^k[n] \tag{3}$$

donde, $n = \{0, ..., N-1\}$, $l = \{0, ..., L-1\}$, $x^k[n]$ es el *k-ésimo* símbolo OFDM transmitido de tamaño *N*, *L* es la longitud de CIR, $h^k[n,l]$ es la CIR del *k-ésimo* bloque en el instante *n* para un impulso como entrada en las *l* muestras previas y $w^k[n]$ es el ruido aditivo Gaussiano blanco (AWGN) complejo. La convolución circular en (3) entre el CIR y x^k se puede reescribir como:

$$\mathbf{y}^k = \mathbf{H}^k\mathbf{x}^k + \mathbf{w}^k, \tag{4}$$

donde:

$$\mathbf{y}^k = [y^k[0], y^k[1], ..., y^k[N-1]]^T,$$
$$\mathbf{x}^k = [x^k[0], x^k[1], ..., x^k[N-1]]^T,$$
$$\mathbf{w}^k = [w^k[0], w^k[1], ..., w^k[N-1]]^T,$$

es el vector de ruido AWGN circular y simétrico, con media cero y varianza $\sigma_w^2 = N_0/2$. $\mathbf{H}^k$ es una matriz de dimensión *N x N* cuyos elementos son formados por los coeficientes de la CIR con la siguiente asignación:

$$[\mathbf{H}^k]_{n,n'} = h^k[n, \langle n\text{-}n' \rangle_N], \tag{5}$$

donde *n,n' = {0, ..., N-1},* $\langle\ \rangle_N$ es el operador modulo *N* y la CIR se asume que es cero para $\langle n\text{-}n' \rangle_N > L\text{-}1$. El símbolo OFDM que se recibe en el dominio de la frecuencia (FD) y sin CP, se obtiene multiplicando la ecuación (4) por la matriz de la transformada discreta de Fourier (DFT) normalizada.

$$[\boldsymbol{F}]_{n,n'} = \frac{1}{\sqrt{N}} e^{-j2\pi nn'/N}, \tag{6}$$

lo cual da como resultado:

$$\boldsymbol{u}^k = \boldsymbol{F}\boldsymbol{H}^k \boldsymbol{x}^k + \boldsymbol{z}^k, \tag{7}$$

donde $\boldsymbol{u}^k$ es el símbolo OFDM recibido en el FD y $\boldsymbol{z}^k$ es la DFT del vector de ruido. Utilizando las propiedades de la matriz $\boldsymbol{F}$ ortogonal, $\boldsymbol{F}^{-1} = \boldsymbol{F}^H$ y $\mathbf{F}^H\ \mathbf{F} = \mathbf{I}$, donde **I** es la matriz indentidad de tamaño *N x N*, le ecuación 7 se puede reescribir como:

$$\boldsymbol{u}^k = \boldsymbol{F}\boldsymbol{H}^k \boldsymbol{F}^H \boldsymbol{F}\boldsymbol{x}^k + \boldsymbol{z}^k$$

$$= \boldsymbol{F}\boldsymbol{H}^k \boldsymbol{F}^H \boldsymbol{s}^k + \boldsymbol{z}^k \tag{8}$$

$$= \boldsymbol{G}^k \boldsymbol{s}^k + \boldsymbol{z}^k \tag{9}$$

donde $\boldsymbol{s}^k$ está compuesto por el vector $s_D^k \in \Omega$ de tamaño N_D, N_p símbolos pilotos y N_G símbolos de guarda. $\boldsymbol{G}^k = \boldsymbol{F}\boldsymbol{H}^k \boldsymbol{F}^H$ es la matriz de canal en la frecuencia que contiene la información en los dominios de la frecuencia y la frecuencia Doopler de la representación circulante y dispersa de la CIR variante en el tiempo. Cuando la propagación Doppler es insignificante, $\boldsymbol{G}^k$ es una matriz diagonal lo cual repercute a un sistema libre de ICI. En ambientes V2V debido a la gran movilidad tanto del transmisor como del receptor, la dispersión Doppler es muy significativa originando que la matriz $\boldsymbol{G}^k$ se convierta en una matriz dispersa generando un sistema afectado por ICI.

Si CIR es variante en el tiempo, entonces $\boldsymbol{G}^k$ es una matriz no diagonal, originado ICI. La ecuación (9) puede ser reescrita para el modelo como:

$$\boldsymbol{u}_D^k = \boldsymbol{G}_D^k \boldsymbol{F} \boldsymbol{s}_D^k + \boldsymbol{z}_D^k$$

$$= \boldsymbol{G}_D^k \boldsymbol{x}_D^k + \boldsymbol{z}_D^k \tag{10}$$

donde $\boldsymbol{u}_D^k$, $\boldsymbol{G}_D^k$ y x_D^k son respectivamente el vector de datos recibido, la CFM de rango reducido $N_D \times N_D$, el vector de datos transmitido con precodificación lineal (LP) y el vector de ruido; cada uno muestreodo en las posiciones de subportadoras de datos.

2.5.2 Modelo del detector OSIC basada en la descomposicion QR

La combinación del detector de símbolo de cancelación de interferencia sucesiva ordenada (OSIC) con la descomposición V2V Sorted QR, permite la implementación de un detector de símbolo subóptimo de baja complejidad, para que, el sistema multipotadora V2V con LP presente un excelente desempeño en términos de BER. La descomposición V2V Sorted QR de la matriz de canal $\boldsymbol{G}_D$, calcula una matriz triangular superior $\boldsymbol{R}$, una matriz ortogonal de norma unitaria $\boldsymbol{Q}$ y una matriz de permutación $\boldsymbol{P}$, tal que, $\boldsymbol{G}_D\boldsymbol{P} = \boldsymbol{QR}$, donde $\boldsymbol{G}_D\boldsymbol{P}$ es la matriz $\boldsymbol{G}_D$ columnas ordenadas de acuerdo a $\boldsymbol{P}$ [22].

La idea básica es obtener un modelo reducido de la ecuación 10 utilizando:

$$\boldsymbol{u}_D^k = \boldsymbol{QR}\boldsymbol{s}_D^k + \boldsymbol{z}_D^k \tag{11}$$

$$\boldsymbol{Q}^H \boldsymbol{u}_D^k = \boldsymbol{R}\boldsymbol{s}_D^k + \boldsymbol{Q}^H \boldsymbol{z}_D^k \tag{12}$$

$$\widetilde{u} = \boldsymbol{R}\boldsymbol{s}_D^k + \widetilde{z} \tag{13}$$

las estadísticas del ruido no son alteradas debido a que la matriz $\boldsymbol{Q}$ es unitaria. El nuevo modelo descrito en la ecuación 13 es adecuado para la aplicación directa de la

detección OSIC. Debido a la estructura triangular de la matriz ***R***, el *k-esimo* elemento de $\widetilde{u}$ puede ser calculado por:

$$\widetilde{u}_k = \boldsymbol{r}_{kk}\boldsymbol{s}_i + \sum_{i=k+1}^{N_D} r_{ki}s_i + \widetilde{z} \tag{14}$$

donde r_{ab} denota el elemento de la matriz **R** en la *a-ésima* fila y *b-ésima* columna. La detección del *j-ésimo* símbolo recibido es encontrado secuencialmente en el orden $k = N_D, N_D\text{-}1, ..., 1$ utilizando la siguiente expresión:

$$\widehat{s}_k = Q[\frac{\widetilde{u}_k - \sum_{i=k+1}^{N_D} r_{ki}\widehat{s}_i}{r_{kk}}], \tag{15}$$

donde $\widehat{s}_k$ es el *k-ésimo* elemento del vector estimado de s_k y $Q[\cdot]$ es un dispositivo de decisión que mapea su argumento al punto más cercano de la constelación Ω. Finalmente, los símbolos decodificados son reordenados acorde a ***P***. Asumiendo que, todas las decisiones previas son correctas, la interferencia de los símbolos previos puede ser cancelada perfectamente en cada iteración del proceso, la contribución del ruido no se considera debido a que su varianza no fue afectada por la propiedad ortonormal de la matriz ***Q***.

La adaptación del detector OSIC al sistema multiportadora posee una muy baja complejidad computacional con respecto al detector óptimo ML, sin embargo debido a la cancelación secuencial que describe el algoritmo para la búsqueda del vector estimado $\widehat{s}$ ocasiona que el vector óptimo sea descartado en algunas realizaciones, por lo que su desempeño en términos de BER se aleja al logrado por el detector óptimo ML.

2.5.3 Detección QR-ML convencional

La adaptación de la detección QR-ML convencional al modelo del sistema receptor planteado, se puede reformular como:

$$\widehat{s}_{ML} = \arg \min_{s \in \Omega^{N_D}} ||\widetilde{y} - \mathbf{R}s|| \tag{16}$$

$$= \arg \min_{s \in \Omega^{N_D}} (\sum_{j=1}^{N_D} |\widetilde{y}_j - \sum_{i=j}^{N_D} r_{j,i} s_i|^2), \tag{17}$$

la busqueda de la solución ML basada en la ecuación (17) puede reflejarse en la construcción del árbol de búsqueda en la Figura 5. Para poder calcular (17) se define la siguiente métrica de rama:

$$d_i = |\widetilde{y} - r_{i,i}\widehat{s}_i - \sum_{j=i+1}^{N_D} r_{i,j}\widehat{s}_j|^2, \tag{18}$$

donde d_i es la valor de la métrica de rama de un nodo $\widehat{s}_i$, que tiene a $\widehat{s}_{N_D}$, ..., $\widehat{s}_{i+1}$ ($\widehat{s}_k \in \Omega$, $i+1$, $\leq k \leq N_D$) como sus nodos antecesores. La distancia entre cada nodo de un *k-ésimo* nivel y la raíz se define como el valor métrico acumulado, el cual representa la suma de todas la métricas de ramas desde la raíz hasta el nodo indicado. Para un nivel *n*, la métrica acumulada se obtiene a partir de:

$$\sum_{i=1}^{n} d_{N_D-i+1} = \sum_{i=1}^{n} |\widetilde{y}_{N_D-i+1} - \sum_{j=N_D-i+1}^{N_D} r_{N_D-i+1}\widehat{s}_j|^2, \tag{19}$$

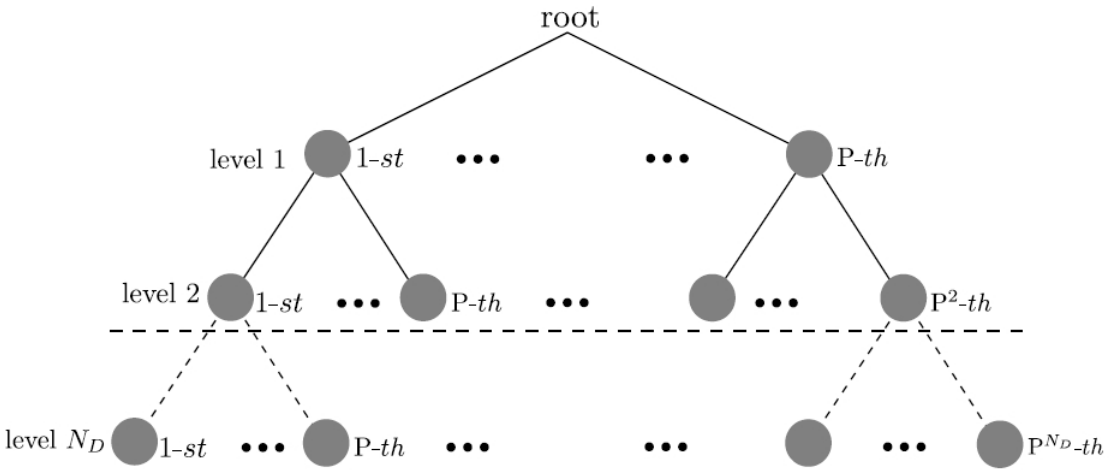

Figura 4: Estructura del árbol de búsqueda bajo el criterio ML

de acuerdo con la ecuación (17), la detección óptima del vector $\widehat{s}$ será la ruta que minimice a (19), cuando $n = N_D$.

Nuestra propuesta expuesta a continuación, se basa en la incorporación del algoritmo M adaptativo a la detección **QR-ML** convencional, realizando ajustes en el proceso de búsqueda de árbol con el objetivo de reducir la complejidad computacional.

2.5.4 Detector V2V Near ML

Es importante aclarar que al igual que la detección OSIC se hace uso de la descomposición Sorted-QR. Como se ilustra en la Figura 5 el simbolo s_{N_D} se ubica en el nodo raíz del árbol, y los nodos hijos que emanan del mismo son una solución posible para $s_{N_{D-1}} \cdots s_1$, la apliación del algoritmo M radica en seleccionar en cada nivel del árbol un máximo de $M(M < P)$ candidatos para la detección del *i-ésimo* símbolo del vector $\widehat{s}$ estimado, descartando los $P - M$ nodos restantes del nivel actual.

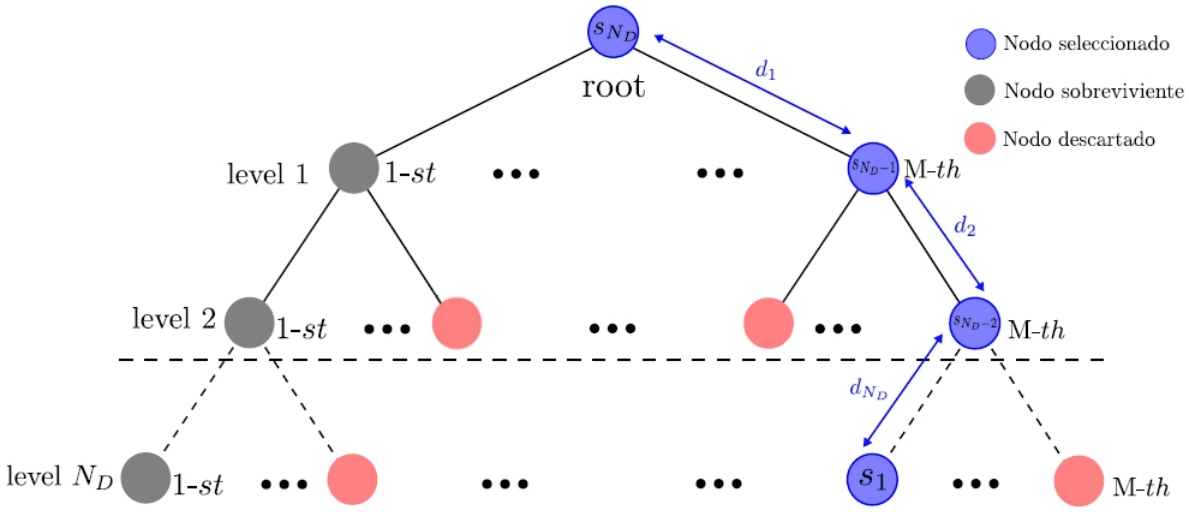

Figura 5: Estructura del árbol de búsqueda bajo el criterio Near ML

A cada rama se le asigna una métrica de distancia definida por:

$$D_k^2 = ||\widetilde{y}_k - \sum_{i=k}^{N_D} \mathbf{R}_{k,i}\mathbf{X}_i||^2, \tag{20}$$

seleccionando los M nodos que mantengan una distancia menor entre cada nodo de un *k-ésimo* nivel y el nodo raíz, por lo tanto al finalizar la detección se tendrá tan solo M rutas posibles, cada una con una distancia total igual a:

$$D_T^2 = \sum_{k=1}^{N_D} D_k^2, \tag{21}$$

la solución $\widehat{s}$, sera dada por la ruta que cumpla con:

$$\widehat{s} = \arg \min_{s \in \Omega} \sum_{k=1}^{N_D} D_k^k = \arg \max_{s \in \Omega} D_T^2. \tag{22}$$

Valores pequeños de M generan como resultado detectores de baja complejidad pero subóptimos en términos de BER, a medida que el valor de M es incrementado, el desempeño del algoritmo propuesto se acerca al desempeño del detector ML, con la penalidad de un incremento en la complejidad computacional del mismo [23].

2.6 Implementaciones en hardware

Una implementación en hardware es mucho más rápida que una de software, ya que el diseño es particular a la ejecución de un algoritmo. Los diferentes dispositivos para realizar una implementación en hardware se encuentran los dispositivos lógicos programables complejos (CPLD), FPGA, los cuales se utilizan para la implementación de prototipos y los circuitos integrados de aplicación específica (ASIC), los cuales son circuitos integrados hechos para un uso en particular.

Los FPGA comparados con los CPLD, es que contienen un gran número de bloques para realizar operaciones sencillas y de alto nivel. Los FPGA comparado con ASIC tienen la principal desventaja de ser más lentos, y un mayor consumo de potencia, pero a un menor costo de adquisición y la característica de ser reprogramables.

En los últimos años, los FPGA's han tenido un gran avance creando soluciones reprogramables, y con la implementación, los diseñadores se pueden enfocar más en la optimización de la arquitectura.

2.7 FPGA's y arquitectura

Los FPGA fueron realizados gracias a la combinación del control del diseñador y el tiempo de desarrollo de los PLD's con el bajo costo y la densidad de los arreglos de compuertas, por lo que rápidamente, los FPGA's se fabricaron. Como los ASIC's necesitan una gran cantidad de lógica combinacional y secuencial y además costosos, los FPGA's se fueron insertando en el mercado rápidamente en el mercado.

Un FPGA, independientemente del fabricante, tiene ciertos elementos en común, tales como:

- *Look-up table* (LUT): elementos básicos que realizan operaciones lógicas de *N* variables booleanas. Este elemento, es básicamente una tabla de verdad, donde a diferentes entradas combinacionales se obtienen diferentes salidas.
- *Flip-Flop* (FF): elementos básicos de un registro, su estructura básica tiene una entrada, una salida, una entrada de reloj, un habilitador, y un *reset*.
- Wire: elementos que realizan las conexiones entre elementos.
- Bloques entrada salida (I/O): puertos disponibles ingresar datos de entrada y asignar valores de salida.

Un bloque típico de un FPGA, consiste en una LUT, en la cual van las entradas lógicas y en la salida un FF con otra entrada para la señal de reloj. Algunos FPGA's cuentan con bloques especializados tales como:

- Módulo DSP48: elemento complejo disponible como una unidad aritmética lógica (ALU), que se encuentra dentro de los FPGA's, que se componen principalmente de una cadena de tres bloques diferentes; un sumador/restador conectado a un multiplicador y finalmente, conectado a un sumador/restador/acumulador.

- BRAM: memoria RAM de doble puerto dentro del FPGA para proveer el almacenamiento *on-chip*. Existen dos tipos de BRAM de 18 Kbits o 36 Kbits. En este bloque se pueden implementar memorias de tipo RAM y ROM.

2.8 Métricas de desempeño

Existen diversas formas para analizar y evaluar las arquitecturas digitales de hardware, independientemente de cual de ellas se use, la mayoría tienen en común las siguientes métricas de desempeño:

- Latencia: Se define como la cantidad de ciclos de reloj que se toma para realiza una instrucción o un grupo de instrucciones. Este parámetro se puede mejorar si es aplicada la técnica de *pipeline*, que significa que la siguiente instrucción en ser realizada puede ser iniciada antes de que la actual ejecución esté completa, permitiendo la superposición de etapas requeridas en el proceso de instrucciones, y así reducir la latencia para el grupo total de instrucciones.
- Error absoluto: Para caso de hardware establece la diferencia entre el valor exacto (punto flotante)de la magnitud y el valor obtenido (punto fijo), el cual puede dar como resultado un número positivo o negativo. El resultado proporciona una idea de la calidad de la medida.
- Error relativo: Define el cociente entre la magnitud en punto fijo y el valor en punto flotante [24].
- Relación señal ruido de cuantificación (SQNR): Es una métrica que sirve para evaluar el desempeño en punto fijo de una arquitectura. La SQNR expresa la razón entre la potencia deseada y la señal de ruido cuantificada, y así, obtener el valor mínimo de precisión computacional y especificaciones de la aritmética de punto fijo ajustadas a la aplicación. Entre mayor sea el SQNR, es más preciso el resultado, es decir, será más cercano al resultado que se obtiene con la precisión proporcionada por la aritmética de punto flotante. SQNR es una métrica estadística, la cual dice si él algoritmo es confiable o no [25].

- Consumo de recursos: Se consideran como recursos del FPGA, los elementos que pertenecen a su estructura básica, mismos que fueron descritos en el apartado 2.7. Dependiendo del algoritmo que se esté implementando se consumirá un porcentaje de los recursos, una vez realizada la síntesis del mismo, se despliega la cantidad de recursos que son utilizados. Generalmente se busca mantener el menor consumo posible.
- Frecuencia máxima: Es la frecuencia máxima del reloj principal a la que puede funcionar correctamente el sistema, es decir, con una frecuencia menor o igual se garantiza el correcto funcionamiento. Para calcular la frecuencia máxima, primero se debe de calcular el periodo mínimo del sistema, el cual de compone por retardos máximos de los componentes síncronos, retardo de los componentes computacionales, y retardo de los conexiones o ruteo. Al trasladar una aquitectura al FPGA, se realizan conexiones, con esto múltiples caminos síncronos, siendo el camino de mayor retardo el de más importancia. Con el tiempo de camino crito es medido el periodo mínimo y el inverso de esto se tiene la frecuencia máxima de muestreo.

2.9 Modelo algorítmico

El modelo algorítmico es planteado por Romero en [26], que permite al algoritmo en cuestión pueda ser implementado en forma eficiente ya sea en términos de velocidad o área consumida. Asimismo, facilita que los diseños puedan ser replicados por cualquiera que requiera de su uso.

2.9.1 Procedimiento de mapeo de algoritmo a arquitectura

En sentido amplio, este proceso está conformado por los siguientes fases:

1. Generación del modelo de oro *(Golden model)*.

2. Adecuación del algoritmo *(Algorithm transformation)*.

3. Análisis de punto fijo *(Fixed-point analisys)*.

4. Dise~no de la arquitectura de hardware *(Hardware architecture design)*.

5. Implementación y verificación de la arquitectura *(Architecture implementation and verification)*.

Vale la pena resaltar, que este proceso puede tener una naturaleza iterativa en cualquiera de sus fases, ya que ha menudo se requieren hacer ajustes para obtener los resultados deseados.

2.9.1.1 Generación del modelo de oro

El objetivo que se persigue aquí es el de generar una referencia para el algoritmo que pueda emplearse en pasos posteriores para evaluar otras figuras de mérito. Para esto, se debe programar el algoritmo objeto de estudio, empleando preferentemente un lenguaje de alto nivel, en aritmética de punto flotante y con sus ecuaciones originales.

2.9.1.2 Adecuaciones al algoritmo

La finalidad de este punto es modificar las ecuaciones originales o secuencia de pasos del algoritmo para reducir su complejidad sin alterar el resultado final. No existe una regla de dedo para aplicarlo, únicamente el entendimiento profundo y un análisis exhaustivo del algoritmo ayudará a identificar si los siguientes artificios (por citar algunos) pueden ayudar a replantearlo.

- Factorizaciones o aproximaciones matemáticas.
- Simplificaciones o propiedades matetéticas.
- Transformaciones o cambios de dominios.
- Sistolización de operaciones de cálculo intensivo.
- Sustitución de divisiones y multiplicaciones con operaciones conjuntas de desplazamientos y sumas/restas.

2.9.1.3 Análisis de punto fijo

La elección natural de emplear aritmética de punto flotante en las arquitecturas de procesmiento digital de señales (PDS) regularmente es desechada. Esto se debe a que es posible alcanzar desempe~nos similares si se sustituyen los cálculos con su representación de punto fijo. También, éstos últimos ocupan menos tiempo de procesamiento y son menos complejos de implementar. En consecuencia, su consumo de área y potencia son menores.

Bajo la luz de lo expuesto, en este trabajo se ha optado por incluir aritmética de punto fijo en las arquitecturas de procesamiento digital de se~nales a dise~nar, lo cual requiere que un análisis del algoritmo sea llevado a cabo en esos términos para encontrar la longitud y el formato de palabra correcto. Para cuantificar los efectos de la aritmética de punto fijo en un algoritmo, se usa la figura de mérito SQNR, que mide la calidad de una se~nal analógica después de ser sometida a un proceso de cuantización.

El análisis de punto fijo se divide en los pasos que a continuación se enumeran:

- Medición del rango dinámico: Se define como rango dinámico a la diferencia entre los valores máximos y mínimos que toma la variable. De tal forma, que resulta imprescindible dimensionar el rango dinámico de todas las variables en punto flotante que son manipuladas por el algoritmo, para después calcular la cantidad de bits que se le deben asignar tanto a la parte entera como a la fraccionaria para representarlas en punto fijo.
- Selección del formato de palabra: Existen diversas formas para obtener el formato de palabra. La cantidad de bits IP para la representación entera de una palabra en punto fijo se relacionada directamente con el rango dinámico de la variable en cuestión, por medio de la expresión siguiente:

$$IP = \lfloor log_2(max(abs(\alpha_{max}), abs(\alpha_{min}))) \rfloor + 2 \tag{23}$$

  donde α_{max} y α_{min} son los valores máximo y mínimo del rango dinámico de la variable en punto flotante y $\lfloor \cdot \rfloor$ es el operador *floor*. La parte fraccional define la resolución ϵ de una variable en punto fijo. La resolución es el cambio más

peque~no que se puede tener debido a los niveles de cuantización y queda expresada como:

$$\epsilon = \frac{1}{2^{FP}} \tag{24}$$

Por lo que, al despejar *FP* que representa el número de bits fraccionales para una resolución dada, se tiene que:

$$\textit{FP} = \lceil log_2 \frac{1}{\epsilon} \rceil \tag{25}$$

donde $\lceil \cdot \rceil$ denota el operador *ceil*.

Cabe señalar que la resolución de la parte fraccional depende de las necesidades del sistema. En general, se debe hacer una elección cuidadosa debido a que una mayor resolución desembocará en un aumento en la anchura de la palabra (*WL*).

- Cuantificación de la relación señal a ruido de cuantización (SQNR): El proceso de cuantizar un valor de punto flotante genera un error que puede ser visto como ruido que contamina al valor original, por ende el SQNR es la figura de mérito preferida para medir la calidad del proceso de cuantificación de una variable de punto flotante. Debido que la mayoría de los algoritmos operan con datos expresados como matrices y vectores, la SQNR vendrá expresada como un promedio dado por:

$$\textit{SQNR} = 10 log_{10} \frac{\sum_{n=0}^{N-1} a(n)^2}{\sum_{n=0}^{N-1} (b(n) - a(n))^2} \tag{26}$$

donde *a(n)* es una señall original en punto flotante, *b(n)* es su contraparte en punto fijo y *N* es la cantidad de elementos que las conforman. En particular, se debe monitorear el SQNR en todas las variables del algoritmo y ajustar la longitud de la palabra en aquellas donde el SQNR sea muy pobre y sacrificar bits en donde sea muy elevado. La finalidad es alcanzar un SQNR objetivo en los resultados de salida con un equilibrio en las longitudes de palabras de las variables.

- Efectos del redondeo: El inconveniente de usar aritmética de punto fijo es que no todos los valores pueden ser representados de manera exacta. Cuando esto sucede, el método de redondeo se utiliza para convertir el valor a un número representable con la consabida pérdida de precisión. La elección del tipo de rendondeo que se pretenda usar en el algoritmo impactará directamente en su desempeño en términos de SQNR, área y velocidad de procesamiento. Así, un redondeo sencillo implicará un diseño más rápido, con menor consumo de área y con un SQNR menor que si se utilizará un redondeo más complejo. Los tipos de redondeo más empleados son el truncamiento, redondeo hacia cero, redondeo hacia más/menos infinito y el redondeo al más cercano.

2.9.1.4 Diseño de la arquitectura de hardware

Toda vez que se han dimensionado las longitudes y los formato de palabra correspondientes a las variables del algoritmo, lo siguiente es realizar un diagrama a bloques donde se visualice el recorrido que deben seguir los datos desde la entrada hasta la salida de algoritmo y que además capture su esencia en un nivel alto de abstracción. Cada bloque deberá representar un proceso funcional que derive en una transformación de los datos. Posteriormente, se tiene que ir incrementando el nivel de detalle inicial subdividiendo cada bloque en módulos, estableciendo una jerarquía. Cada módulo a su vez se tendrá que subdividir cuantas veces sea necesario hasta que sea sencillo de dise~nar o su funcionalidad se relacione con algún componente u operación primitiva. De esta forma se tendrá una primera versión de lo que se conoce como la ruta de datos *(datapath)* de la arquitectura, la cual se tendrá que complementar con su correspondiente unidad de control.

Dependiendo del criterio de optimización: velocidad o consumo de área; que se desee aplicar a la arquitectura inicial, ésta se podrá refinar a través de las técnicas que a continuación se mencionan.

- Paralelismo: Permite la ejecución simulténea de varias operaciones, siempre y cuando no existan una dependencia de datos entre ellas. Esto permite incrementar significativamente la velocidad de operación de la arquitectura pero el consumo de área también se vera afectado en la misma proporción.

- Encauzamiento *(pipelining)*: Cuando en un proceso de la arquitectura se ejecuta una secuencia de operaciones para obtener un resultado, es posible introducir una etapa de *pipeline* que permita traslapar la ejecución de tales operaciones de forma que, ya no será necesario esperar a que se termine de procesar dicha secuencia para poder introducir un nuevo dato al proceso. Vale señalar que la etapa de *pipeline* no mejorará el tiempo para procesar cada dato, pero sí incrementará la cantidad de operaciones que se ejecutan, con lo que el rendimiento de la arquitectura será mayor con un costo mínimo en el consumo de área.
- Reconfigurabilidad: No existe una definición concreta para éste término, pero esté estrechamente relacionado con la capacidad que se le puede otorgar a una arquitectura para modificar en tiempo de ejecución su ruta de datos. Esto con el objetivo de cambiar su funcionalidad y usando los mismos bloques o módulos de hardware. Esto disminuirá el consumo del área en la arquitectura pero las funcionalidades con las que cuente ahora serán excluyentes.
- Reusabilidad: Cuando alguno de los bloques o módulos que cumplan una misma función aparece más de una vez dentro de una arquitectura, se recomienda dise~nar solo uno y usarlo las veces que sea necesario, en lugar de dise~nar varios de ellos idénticos. Esto disminuirpa drásticamente el consumo de área en la arquitectura, pero depreciará la velocidad de operación.
- Portabilidad: A menos que se indique lo contrario, la descripción de la arquitectura no debe asociarse con opciones o componentes de alguna tecnología específica de un fabricante. Esto garantizará su independencia y asegurará el que pueda ser implementada en varias plataformas de dise~no *(EDA tools)* de cualquier fabricante sin realizar cambios en la descripción.

2.9.1.5 Implementación y verificación de la arquitectura

Toda vez que se tiene definida la arquitectura de hardware definitiva, lo que procede es describirla e implementarla a nivel de lógica de transferencia de registros (RTL) usando un lenguaje de descripcionn de hardware. El entorno de desarrollo que el fabricante de la tecnología pone a disposición del arquitecto juega un papel fundamental al facilitar

estos procesos. Después, la arquitectura ya sintetizada tiene que pasar por un proceso de verificación antes de que sea declarada lista para usarse. Esto involucra estudiar su comportamiento en condiciones controladas para:

- Verificar y asegurar que todas las funciones para las que fue diseñada sean llevadas a cabo. Esto se conoce como verificación estática.
- Verificar y asegurar que todas las secuencias involucradas en cada una de las funciones se ejecuten correctamente. Esto se conoce como verificación por comportamiento dinámico, e implica la generación de vectores de entradas específicos variantes con el tiempo, que se aplican a la arquitectura para monitorear las salidas generadas partir de ellos.
- Comprobar el comportamiento temporal: De manera adicional, es posible elaborar pruebas para observar las variaciones de las señales en puntos seleccionados, medir retardos entre eventos específicos, ancho de pulsos, etc.

La verificación se fundamenta casi siempre en bancos de pruebas *(test-benchs)*, que básicamente son un entorno de simulación alrededor de la arquitectura. Su labor consiste en inyectar un conjunto de estímulos específicos en las entradas y verificar que el resultado obtenido en las salidas sea el esperado. Por otro lado, es importante resaltar que a pesar de lo exhaustiva que pueda llegar a ser una verificación, nunca se logra al 100 % cuando se trata de arquitecturas complejas.

CAPÍTULO III

Método

El propósito de este capítulo es mostrar, a grandes rasgos, la metodología a seguir durante el desarrollo del trabajo de tesis; dígase, el sujeto de estudio, el procedimiento paso a paso, y las herramientas y materiales utilizados.

3.1 Sujeto

El objeto de estudio de esta investigación es un bloque detector de símbolo para el esquema de transmisión SISO-OFDM en ambientes vehiculares; implementado en un FPGA Xilinx.

3.2 Procedimiento

El procedimiento a seguir se puede observar en el diagrama de flujo de la Figura 6.

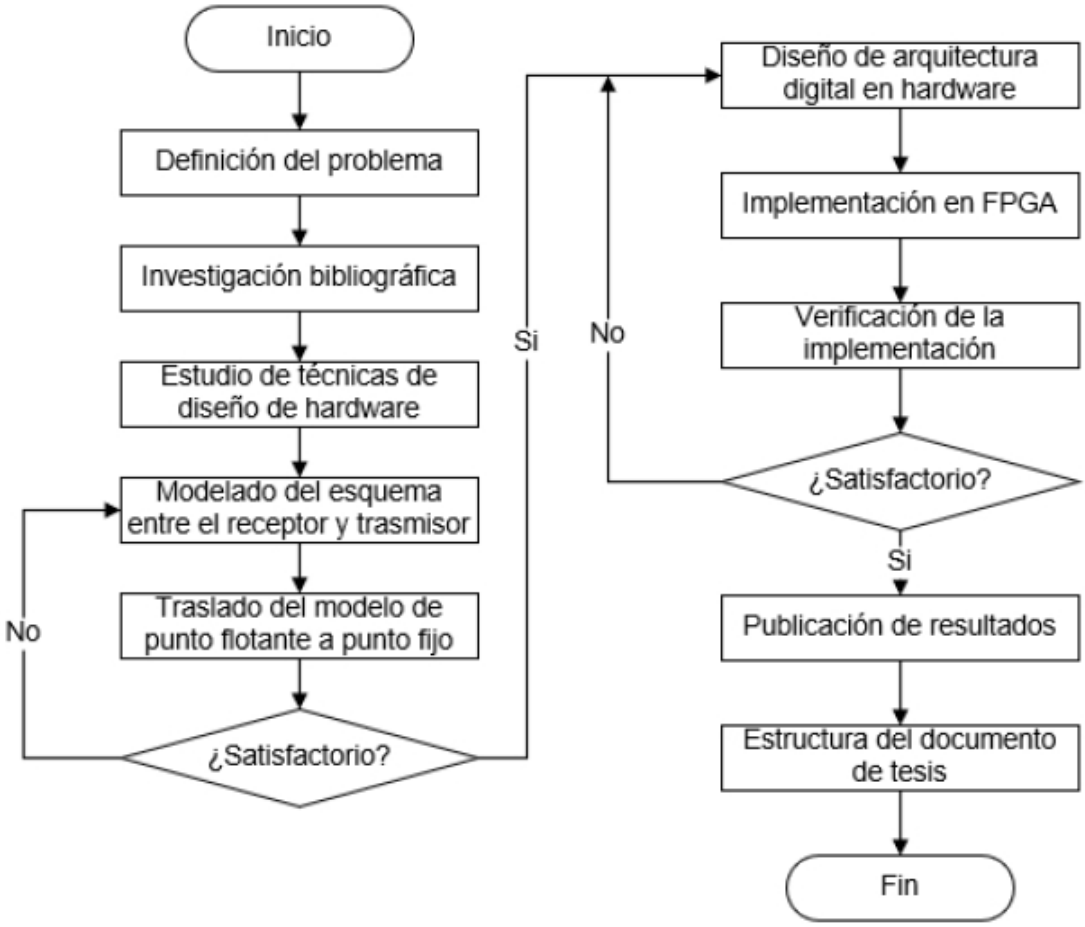

Figura 6: Diagrama de flujo del procedimiento a seguir

- *Definición del problema*: se establecen los antecedentes, objetivos, definición del problema a resolver, hipótesis planteadas, alcances y limitaciones del proyecto.
- *Investigación bibliográfica*: se revisan artículos, libros, revistas, etc. con la finalidad de conocer el estado del arte de detectores SISO en FPGA y así poder identificar las áreas de oportunidad.
- *Estudio de técnicas de diseño de hardware*: una vez "empapado" de información acerca de detectores de símbolo, se realiza un estudio sobre los métodos de diseño e implementación en hardware.

- *Modelado del esquema entre el receptor y transmisor*: se comprueba el funcionamiento del algoritmo del detector en MATLAB, utilizando el modelado completo entre el transmisor y receptor.
- *Traslado del modelo de punto flotante a punto fijo*: una vez comprobado el funcionamiento del algoritmo en punto flotante, se traslada a aritmética de precisión finita para emular su comportamiento en un sistema digital real.
- *Diseño de arquitectura digital en hardware*: se procede a trabajar con la propuesta para el detector de símbolo OSIC y codificación en lenguaje de descripción de hardware como verilog, y así utilizar un simulador basado en software para comprobar el funcionamiento de la arquitectura ya codificada.
- *Implementación en FPGA*: ya que la simulación en software fue exitosa, se procede a implementar la propuesta de diseño de la arquitectura ya codificada en FPGA.
- *Verificación de la implementación*: se comprueba la frecuencia máxima de operación que la implementación puede manejar, la cantidad de hardware requerido en el FPGA; si no hay resultados satisfactorios, se regresa a la etapa del diseño de la arquitectura.
- *Publicación de resultados*: una vez verificada la implementación, se publican los resultados en artículos, ya sean de revista o congresos.
- *Escritura del documento de tesis*: se escribe el documento de tesis para el proceso de titulación.

3.3 Herramientas

- Tarjeta FPGA Artix-7 AC701.
- Monitor y teclado.
- PC (Intel i7-7700HQ a 2.8 GHz, 8 GB RAM, 1 TB HDD) con Windows 10, y software Vivado 2016.1, System Generator 2016.1 y MATLAB 2015a instalados.

CAPÍTULO IV

Desarrollo

En este capítulo se realiza el procedimiento mostrado en la sección 2.9 del documento, así como la arquitectura para el desarrollo del algoritmo de detección OSIC, con figuras y tablas que ayudan a un mejor entendimiento del trabajo desarrollado.

4.1 Desarrollo del modelo algorítmico

Como se observó en la sección 2.9.1.1, el algoritmo original visto en la sección 2.5.2 fue modelado en la herramienta computacional MATLAB para comprobar su correcto funcionamiento en términos de BER. También se adecua este algoritmo como se ve en la sección 2.9.1.2 para lograr que las operaciones que se realizan sean implementables en FPGA.

4.1.1 Análisis en punto fijo

Con base en lo visto en la sección 2.9.1.3, se realiza la simulación en punto flotante para obtener $\alpha_{max} = 8$ y $\alpha_{min} = -8$. Los valores mencionados anteriormente, fueron obtenidos con la funcion max y min de MATLAB.

Con ayuda de la ecuación (23), se procedió a calcular la cantidad de bits de la parte entera sustituyendo los valores máximos y mínimos.

$$IP = \lfloor log_2(max(abs(7{,}88), abs(-7{,}92))) \rfloor + 2,$$

$$IP = \lfloor log_2(max(7{,}88, 7{,}92)) \rfloor + 2,$$

$$IP = \lfloor log_2(7{,}92) \rfloor + 2,$$

$$IP = \lfloor 2{,}98 \rfloor + 2,$$

$$IP = 2 + 2,$$

$$IP = 4,$$

Por cuestiones de compatibilidad con otros módulos utilizados posteriormente, y para estandarizar nuestro tamaño de palabra a 16 bits, se optó por utilizar un $IP = 5$.

En nuestra implementación, se busca tener un error de resolución representado en -60 dB, por lo cual se sustituye éste valor en la ecuación 25, con su respectiva conversión de dB a magnitud, dando como resultado:

$$FP = \lceil log_2 \frac{1}{10^{(-60/20)} dB} \rceil$$

$$FP = \lceil log_2 \frac{1}{0{,}001} \rceil$$

$$FP = \lceil log_2 9{,}98 \rceil$$

$$FP = 10$$

Como en el caso anterior, al momento de calcular IP, hacemos un ajuste a los resultados de FP para utilizar los bits que faltan para acompletar la longitud de palabra de 16 bits, por lo que nuestra $FP = 11$.

Para comprobar el correcto funcionamiento del algoritmo en precisión finita, se adaptó el algoritmo en MATLAB para hacer las simulaciones en punto fijo, y así corroborar la longitud de bits adecuada para implementar el algoritmo.

Se utilizó el siguiente formato en el *script* de MATLAB, con el cual se hacen las transformaciones a punto fijo de las variables necesarias:

```
Signed = 1;
Word_Length = 16;
Integer_Part = X;
Fractional_Part = Word_Length - Integer_Part;

Data_Format = numerictype('Signed',Signed,...
'WL',Word_Length,...
'FractionLength',Fractional_Part);

Data_OP = fimath('RoundMode','floor',...
'OverflowMode','saturate',...
'ProductMode','SpecifyPrecision',...
'ProductWordLength',Word_Length,...
'ProductFractionLength',Fractional_Part,...
'SumMode','SpecifyPrecision',...
'SumWordLength',Word_Length,...
'SumFractionLength',Fractional_Part);
```

En la porción de código mostrada anteriormente, la variable *Integer_Part* se evaluó para los valores 4,5 y 6 (representado como $Integer\_Part = X$) para así lograr observar su desempeño en términos de BER, como es mostrado en las Figuras 7, 8 y 9, donde

se observan los resultados de estas pruebas para los ordenes de modulación QPSK, 16-QAM y 64-QAM, respectivamente.

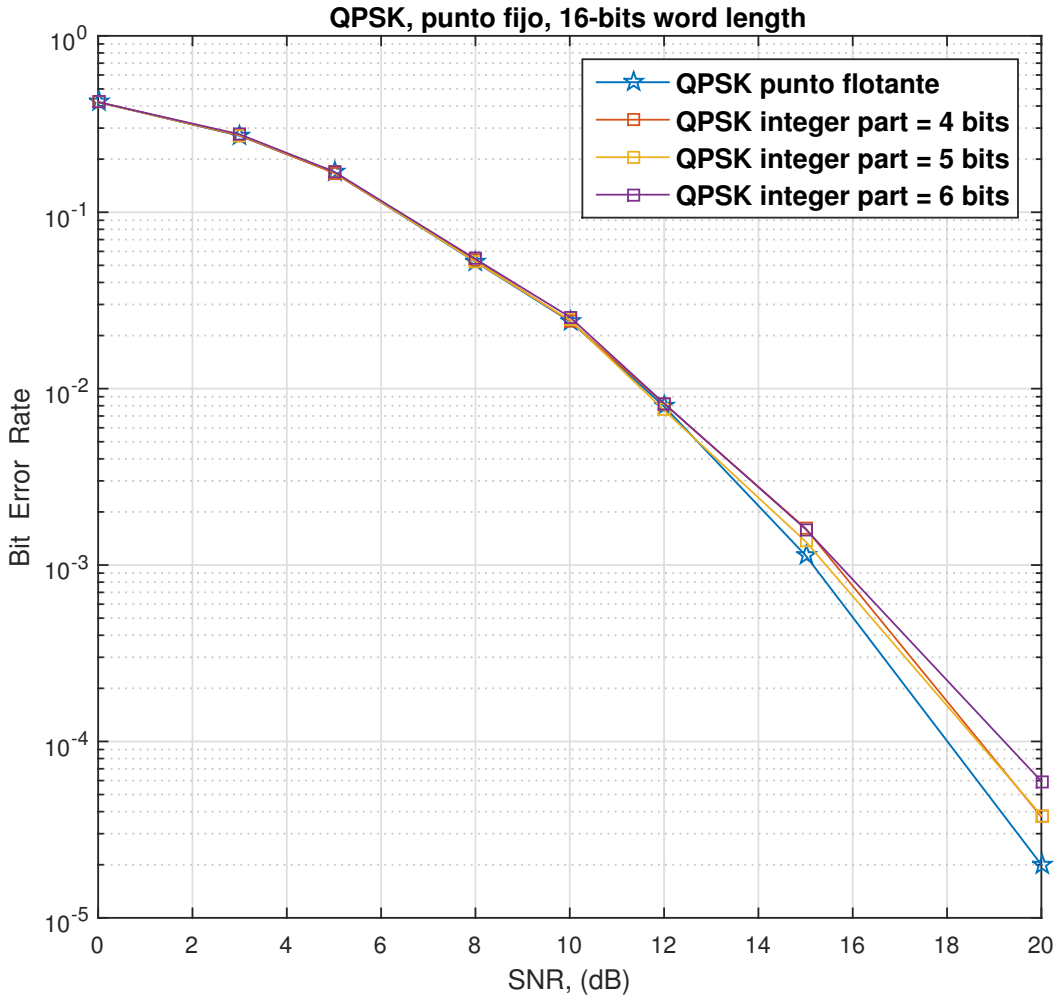

Figura 7: Análisis de punto fijo del algoritmo. QPSK.

Cabe destacar que las gráficas generadas no son exhaustivas, solo son un conjunto de simulaciones promediadas con base en matrices de canal especiales para canales de comunicación V2V, pues las operaciones en punto fijo en MATLAB son extremadamente tardadas, pero los resultados ayudan a tener un estimado del comportamiento general del algoritmo con diferentes configuraciones de bits.

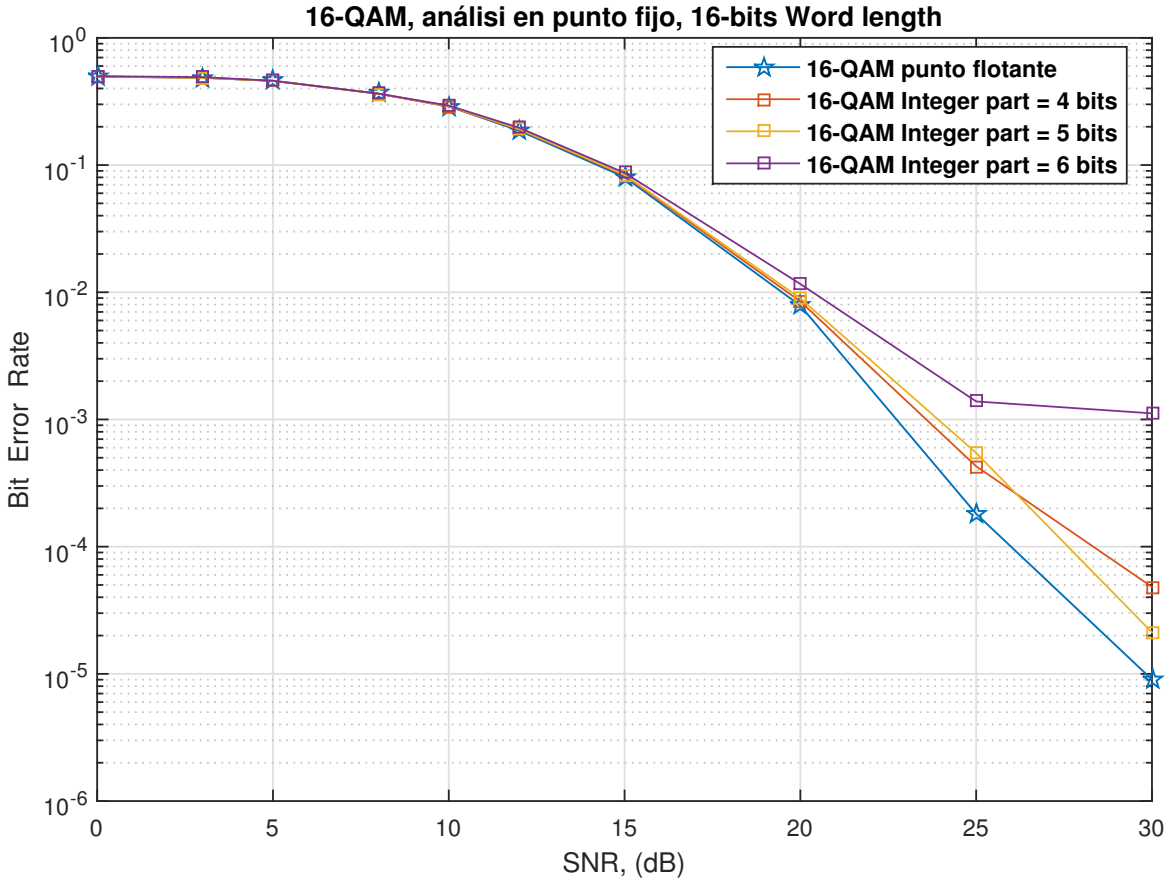

Figura 8: Análisis de punto fijo del algoritmo. 16-QAM.

Los resultados en las tres gráficas indican que tener una parte entera de 4, o incluso 5 bits es aceptable, pues son las que mejor siguen a la línea de punto flotante. En este trabajo, son utilizados 5 bits para la parte entera, dados los casos de *overflow* que se presentaban al utilizar solo 4 bits y además, es el valor calculado anteriormente.

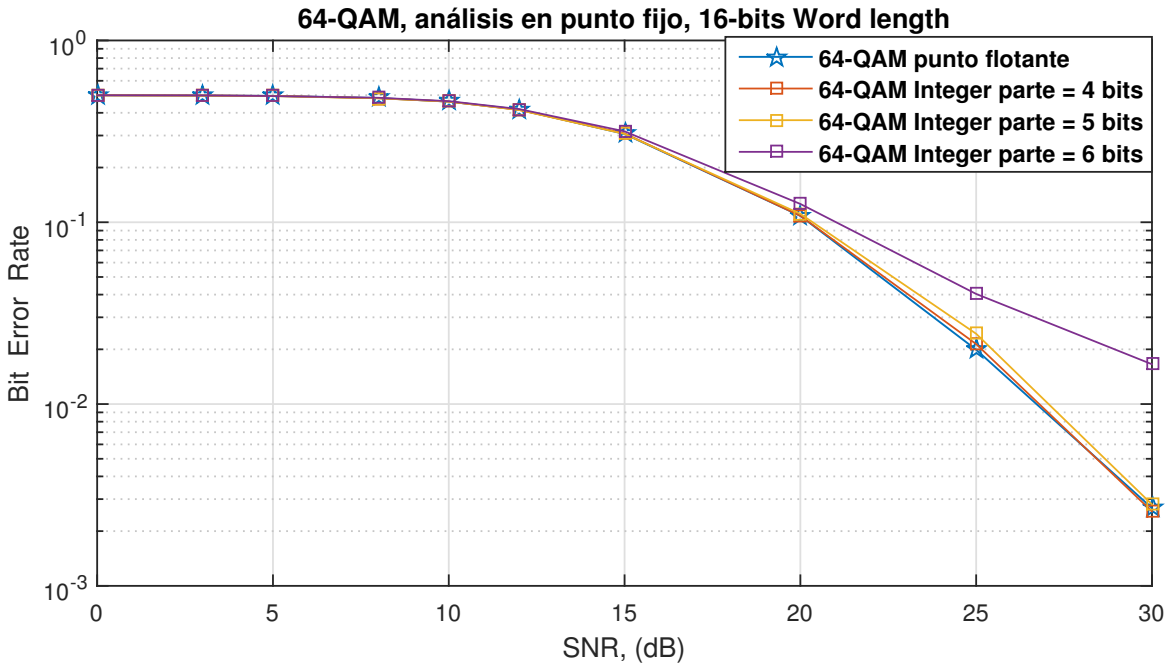

Figura 9: Análisis de punto fijo del algoritmo. 64-QAM.

4.2 Arquitectura propuesta

Para explicar la composición de la arquitectura, primeramente, se muestra el modulo *TOP LEVEL* el cual es el grado más alto de abstracción, posteriormente, describir la composición de los módulos internos.

Consecuentemente, los vectores de bits, se representar con líneas más gruesas que las de un solo bit.

4.2.1 Parametrización

Para la codificación de la arquitectura fueron utilizados los siguientes parámetros principales:

- *WL*: longitud de palabra utilizada para la representación en punto fijo.
- *IP*: cantidad de bits utilizados para la parte entera de *WL*.

- *IF*: cantidad de bits utilizados para la parte fraccionaria de *WL*. Siendo el resultado *WL* menos *IP*.
- *NP*: número de subportadoras por símbolo OFDM.
- *NV*: número de subportadoras de datos por cada símbolo OFDM.
- *M*: índice de modulación utilizada (BPSK, QPSK, 16-QAM o 64-QAM).
- *MAP*: valor máximo de bits para representar el número 255 en binario (8 bits).

4.2.2 Módulo *Top*

La figura 10, muestra el módulo *TOP LEVEL*, siendo la parte más alta de la arquitectura, donde se observa los bloques principales para la detección de símbolo. Así mismo, en la Tabla 3, son presentadas las entradas y salidas del detector completo.

Componentes:

- **Div_Factor**: primera parte del proceso de cancelación de la detección..
- **Hard_Demapper**: genera cadena de bits correspondiente a un punto de la constelación.
- **Symbol_Detector**: mapea un punto de la constelación.
- **Arit_Oper**: realiza operaciones aritméticas.
- **CTRL**: módulo de control.
- **RA**: arreglo de registros.

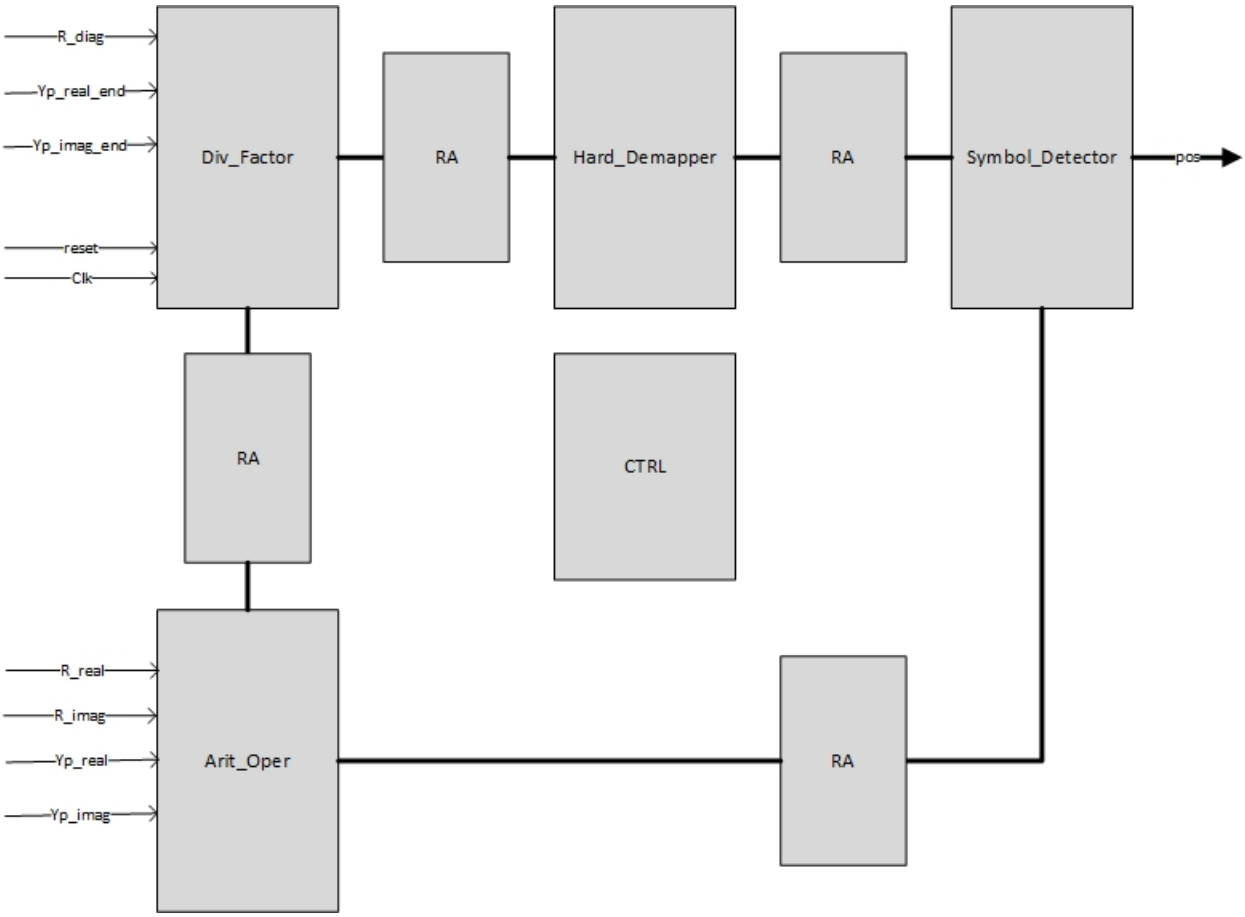

Figura 10: *Top* del detector

Tabla 3: Entradas y salidas del detector

Entrada	Tamaño (bits)	Entra a	Descripción
R_diag	*WL*	**Div_Factor**	Diagonal principal de la matriz **R**.
Yp_real_end	*WL*	**Div_Factor**	*K-ésimo* valor de la parte real del vector recibo.
Yp_imag_end	*WL*	**Div_Factor**	*K-ésimo* valor de la parte imaginaria del vector recibo.

R_real	$WL \times NV$	**Arit_Oper**	Valor real de la *k-ésima* columna de la matriz **R**.
R_imag	*WL*	**Arit_Oper**	Valor imaginario de la *k-ésima* columna de la matriz **R**.
Yp_real	$WL \times NV$	**Arit_Oper**	Parte real del vector recibo.
Yp_imag	$WL \times NV$	**Arit_Oper**	Parte imaginaria del vector recibo.
Salida	**Tamaño (bits)**	**Sale a**	**Descripción**
pos	*log2(M)*	**Símbolo detectado**	Índice detectado.

EL módulo **Div_Factor** con ayuda del *k-ésimo* elemento del vector recibido, el *k-ésimo* elemento de la diagonal principal de la matriz **R**, se realiza la parte de la cancelación para así obtener un punto en la configuración de la constelación que se esté trabajando. Una vez realizada la cancelación, el punto obtenido es procesado por medio de comparadores por el módulo **Hard_Demapper** teniendo como resultado una cadena de 8 bits. El módulo **Symbol_Detector** mapea la posición de la coordenada que representa el símbolo estimado la *k-ésima* subportadora de datos. El módulo **Arit_Oper** con ayuda de la *k-ésima* columna de la matriz **R** de la descomposición **QR** y el vector recibió (**Yp**), realiza operaciones aritméticas para la siguiente cancelación. El módulo **CTRL**, como su nombre lo indica, se encarga de controlar y sincronizar la operatividad de la arquitectura. **RA**, son únicamente arreglo de registros entre cada uno de los módulos.

4.2.3 Div_Factor

El algoritmo de detección comienza con **Div_Factor**, con su arquitectura en la Figura 11 y una explicación de las entradas y salidas en la tabla 4, este módulo se compone

de la siguiente manera:

- **Divisor**: realiza división aritmética entre números en punto fijo.
- **LUT_Factr**: contiene constantes a multiplicar por cada índice de modulación.

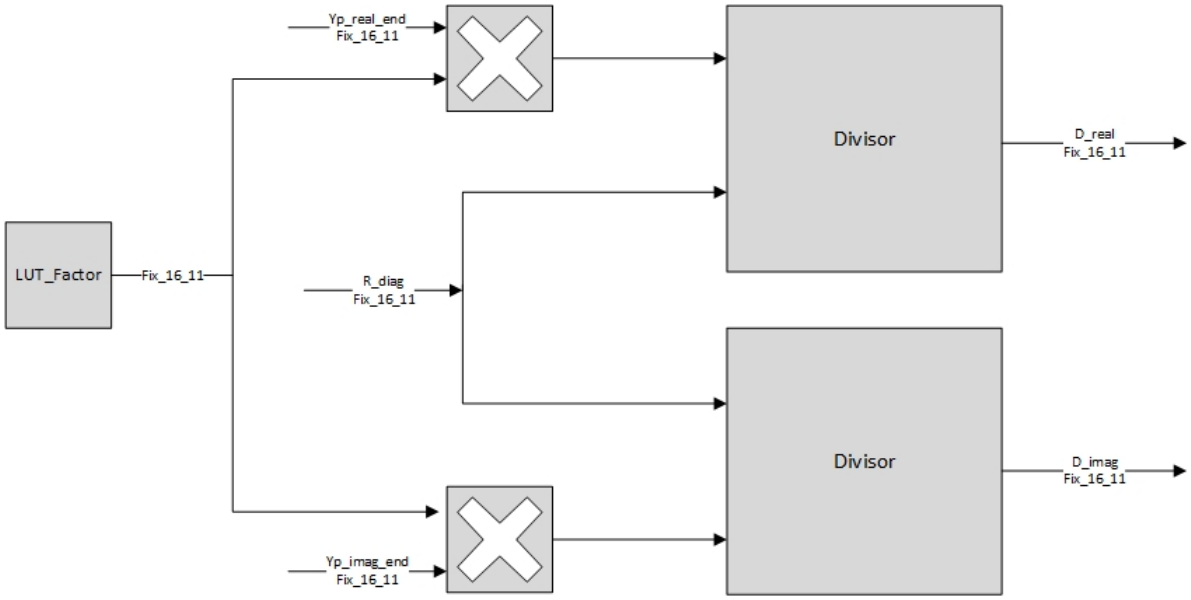

Figura 11: Arquitectura de **Div_Factor**

Tabla 4: Entradas y salidas de **Div_Factor**

Entrada	**Tamaño (bits)**	**Proviene de**	**Descripción**
R_diag	*WL*	**Div_Factor**	Diagonal principal de la matriz **R**.
Yp_real_end	*WL*	**Div_Factor**	*K-ésimo* valor de la parte real del vector recibo.
Yp_imag_end	*WL*	**Div_Factor**	*K-ésimo* valor de la parte imaginaria del vector recibo.
Salida	**Tamaño (bits)**	**Sale a**	**Descripción**
D_real	*WL*	**Hard_Demapper**	Valor real contaminado con ruido.

D_imag	*WL*	**Hard_Demapper**	Valor imaginario contaminado con ruido.

Para iniciar las operaciones aritméticas mostradas, se necesita que **LUT_Factor** muestre el valor del factor de normalización de la constelación utilizada. Este valor es multiplicado tanto por la parte real e imaginaria del *k-ésimo* valor del vector recibido, siendo el resultado de esta multiplicación dividido entre el *k-ésimo* valor de la diagonal principal de la matriz **R**, para que las salidas del módulo pasen al módulo **Hard_Demapper**.

4.2.3.1 LUT_Factor

Como ya se mencionó, el módulo **LUT_Factor** se encarga por medio de una memoria ROM de proveer el dato necesario para el factor de normalización utilizada por cada índice de modulación. La tabla 5, presenta el contenido de este.

Tabla 5: Contenido de **LUT_Factor**

LUT	Contenido (HEX)
00	0800h
01	0b50h
10	194ch
11	33d8h

4.2.3.2 Divisor

El módulo **Divisor**, como su nombre lo dice, realiza la división en aritmética de punto fijo entre dos valores. Como hay dos divisores, se realiza dos divisiones de manera simultánea entre el vector recibido multiplicado por contenido en **LUT_Factor** sobre

el valor *k-ésimo* de la diagonal principal de la matriz **R**. El resultado válido para este módulo es de 3 ciclos de reloj basándose en [27].

4.2.4 Hard_Demapper

Una vez obtenidos los datos **D_real** y **D_imag** se necesita tomar la decisión de que símbolo de la constelación es el que se está detectando, para esto, se necesita una cadena de bit asociada a un punto de la constelación. En la Figura 12, se muestra la arquitectura del módulo **Hard_Demapper** y en la Tabla 6, una explicación de sus entradas y salidas. Este módulo se compone de la siguiente manera:

- **Dec_cuadrante**: detector de signo.
- **ABS**: valor absoluto.
- **LUT_Compara**: constantes necesarias para comparación.
- **Comparador**: establece posición del símbolo.
- **Concat_Select**: concatena bits de la comparación.

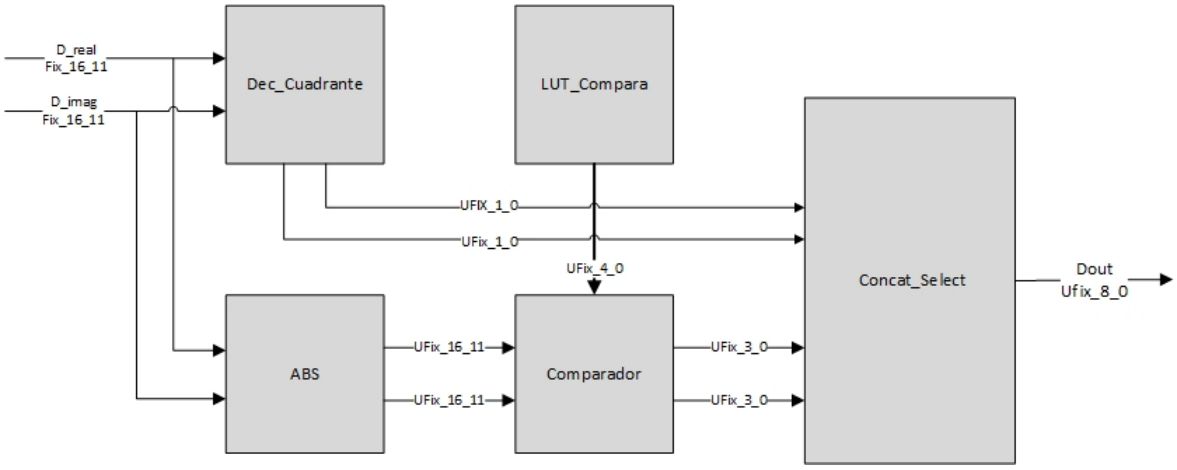

Figura 12: Arquitectura de **Hard_Demapper**

Tabla 6: Entradas y salidas de **Div_Factor**

Entrada	**Tamaño (bits)**	**Proviene de**	**Descripción**
D_real	*WL*	**Div_Factor**	Valor real contaminado con ruido.
D_imag	*WL*	**Div_Factor**	Valor imaginario contaminado con ruido.
Salida	**Tamaño (bits)**	**Sale a**	**Descripción**
Dout	*MAP*	**Symbol_Detector**	Bits relacionados a un punto de la constelación.

Al llegar los datos del resultado de la división al módulo, a los bloques **Dec_Cuadrante** y **ABS** entran las señales contaminadas con ruido, para así verificar la posición del símbolo en Ω y la magnitud presente en las entradas. El bloque **comparador** la señal es comparada con las constantes almacenadas en **LUT_Compara** para así, poder definir la cadena de bits asociado a un punto de la constelación Ω. **Concat_Select** recibe la información procesada por el bloque **comparador** y **Dec_Cuadrante**, para concatenar los bits salientes de los bloques y seleccionar las posiciones de los bits que se utilizaran de acuerdo al índice de modulación utilizado.

4.2.4.1 Dec_Cuadrante

Dec_Cuadrante es un bloque común para las modulaciones utilizadas en el detector, el cual es el encargado de detectar el signo de las señales de entrada, es decir, permite establecer el cuadrante en el que se encuentra el símbolo recibido, por medio de la ecuación 27 y 28.

$$\textit{Dout(7)} = \begin{cases} 1 & si \qquad D\_real \geq 0 \\ 0 & \qquad cualquier\ otro\ caso \end{cases} \tag{27}$$

$$\textit{Dout(3)} = \begin{cases} 1 & si \quad D\_imagl \geq 0 \\ 0 & cualquier\ otro\ caso \end{cases} \tag{28}$$

4.2.4.2 ABS

El bloque **ABS** obtiene el valor absoluto de las coordenadas **D_real** y **D_imag** recibidas por el modulo anterior, para realizar el procesamiento con solo números reales positivos. El funcionamiento de esto bloque se encuentra definido por la ecuación 29.

$$|y| = \begin{cases} y & si \quad y \geq 0 \\ -y & cualquier\ otro\ caso \end{cases} \tag{29}$$

donde y representa el valor de **D_real** o **D_imag**.

4.2.4.3 LUT_Compara

El bloque **LUT_Compara** es una memoria ROM que contiene las constantes necesarias para realizar las comparaciones en el bloque **comparador**. La tabla 7, presenta los valores contenidos en este bloque.

Tabla 7: Contenido de **LUT_Compara**

LUT	Contenido (HEX)
00	0800h
01	0b50h
10	194ch
11	33d8h

4.2.4.4 Comparador

El bloque **Comparador** establece la posición del símbolo de la constelación utilizada por medio de comparaciones entre el bloque **ABS** y **LUT_Compara**. La ecuación (4)

modela el comportamiento para 16-QAM y la ecuación (5) y (6) para 64-QAM.

$$\textit{Dout(6,2)} = \begin{cases} 1 & si \quad y \leq 2 \\ 0 & cualquier\ otro\ caso \end{cases} \tag{30}$$

$$\textit{Dout(6,2)} = \begin{cases} 1 & si \quad y \leq 4 \\ 0 & cualquier\ otro\ caso \end{cases} \tag{31}$$

$$\textit{Dout(5,1)} = \begin{cases} 1 & si \quad 2 \leq y \leq 6 \\ 0 & cualquier\ otro\ caso \end{cases} \tag{32}$$

donde "*y*" representa las salidas del bloque **ABS**.

4.2.4.5 Concat_Select

El bloque **Concat_Select** recibe las comparaciones realizadas por el bloque **Comparador** para concatenar los bits de la palabra **Dout**, la cual es la salida del bloque de 8 bits, seleccionando las posiciones de los bits que serán utilizados de acuerdo al índice de modulación.

4.2.5 Symbol_Detector

Una vez obtenida la cadena de bits correspondiente a un punto de la constelación respecto a la modulación utilizada, se procede a realizar la detección del símbolo por medio del módulo **Symbol_Detector**, el cual es el bloque encargado de decodificar la cadena de bits para obtener la posición correspondiente al valor estimado por el detector. En la Figura 13, se muestra la arquitectura del módulo **Symbol_Detector** y en la Tabla 8, una explicación de sus entradas y salidas. Este módulo se compone por los siguientes elementos:

- **Get**: decodificador.
- **ROM_Conste_real**: almacenamiento de los valores reales de la constelación.

- **ROM_Conste_imag**: almacenamiento de los valores imaginarios de la constelación.

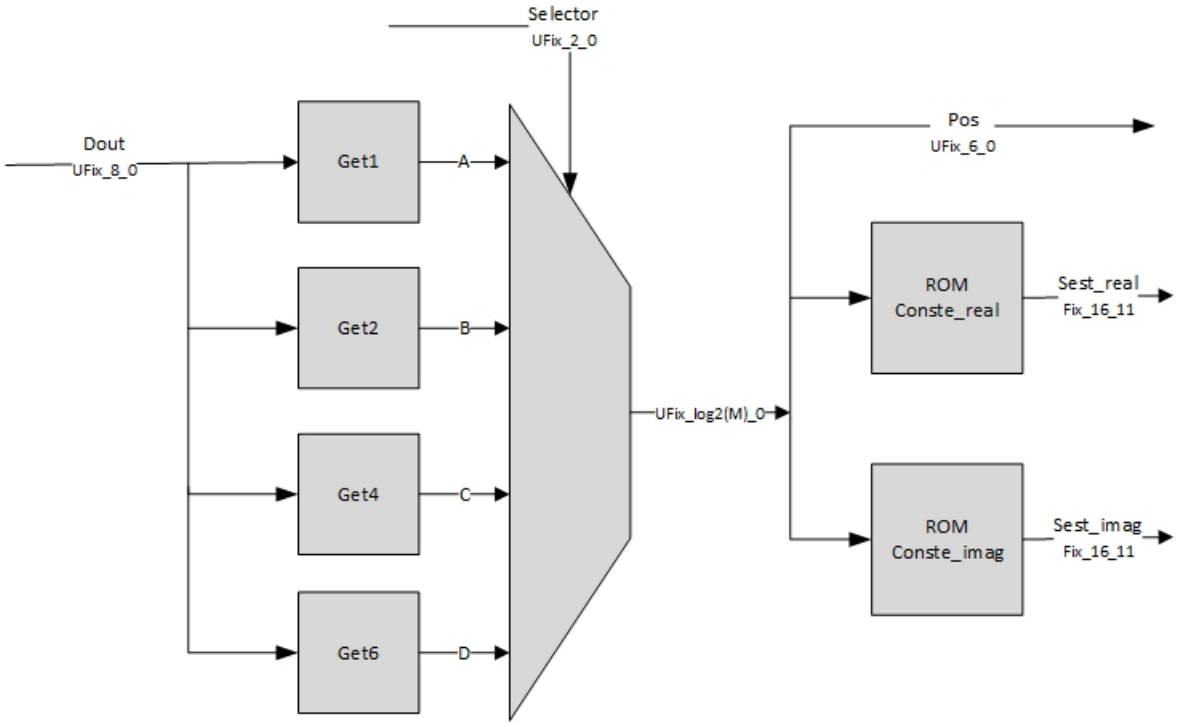

Figura 13: Arquitectura de **Symbol_Detector**

Tabla 8: Entradas y salidas de **Symbol_Detector**

Entrada	**Tamaño (bits)**	**Proviene de**	**Descripción**
Dout	*MAP*	**Hard_Demapper**	Bits relacionados a un punto de la constelación.
Salida	**Tamaño (bits)**	**Sale a**	**Descripción**
Pos	*Log2(M)*	**Salida del detector**	índice de símbolo detectado.
Sest_real	*WL*	**Arit_Oper**	Valor estimado real de la constelación.

Sest_imag	*WL*	**Arit_Oper**	Valor estimado imaginario de la constelación.

Una vez obtenidos la cadena de bits relacionada a un punto de la constelación utilizada, hace falta identificar cual símbolo es el estimado y la posición de este. La señal **Dout** llega a los bloques **Get** los cuales son los encargados de tomar los bits necesarios para decodificar la cadena de bits a una dirección de memoria ROM en donde se encuentran el símbolo estimado, sin antes pasar por un multiplexor para seleccionar la cantidad de bits de salida, por medio de un **Selector**. Esta dirección dada por el multiplexor, es la posición de salida para la estimación de símbolo. Las señales **Sest_real** y **Sest_imag**, son los valores estimados de la constelación para la parte real e imaginaria, respectivamente.

4.2.5.1 Get

Como ya se mencionó, los bloques **Get**, con la cadena de bits proveniente del módulo **Hard_Demapper**, toma los bits necesarios decodificar la dirección de memoria ROM donde se encuentra el símbolo estimado. La Tabla 9, muestra el comportamiento del bloque.

Tabla 9: Comportamiento de **Get**

LUT	Tamaño de salida (bits)	Bits obtenidos de Dout
Get1	1	7
Get2	2	7, 3
Get4	4	7, 6, 3, 2
Get6	5	7, 6, 5, 3, 2, 1

4.2.5.2 ROM_Conste_real y ROM_Conste_imag

Los bloques **ROM_Conste** son direcciones de memorias ROM en donde se encuentran almacenados los valores de la constelación sin normalizar, para que, por medio de la dirección de memoria entregada por el bloque anterior, dar como resultado los valores estimados al módulo **Arit_Oper**. La tabla 10, muestra la cantidad de direcciones de memoria utilizadas por la ROM.

Tabla 10: Direcciones utilizadas en **ROM_Conste_real** y **ROM_Conste_imag**

Modulación	Direcciones de memoria
BPSK	2
QPSK	4
16-QAM	16
64-QAM	64

4.2.6 Arit_Oper

Una vez obtenido el símbolo estimado representado por las señales textbfSest_real y textbfSest_imag provenientes del módulo textbfSymbol_Detector, llegan a módulo textbfArit_Oper, el cual es el encargo de realizar la operación matemática de multiplicación compleja entre el símbolo estimado y el vector recibido, para restar el resultado con los elementos de la columna texttitk-ésima de la matriz textbfR, y así obtener el siguiente valor del vector recibido (textbfYp_real y textbfYp_imag) y el valor para la cancelación siguiente (textbfYp_real_end y textbfYp_imag_end). En la Figura 14, se muestra la arquitectura del módulo **Arit_Oper** y en la Tabla 11, una explicación de sus entradas y salidas. Este módulo contiene los siguientes elementos:

- **RAM**: guardado de los valores del vector recibido.
- **Comple mult**: multiplicación compleja entre dos valores.

- **Reg**: registro.
- **menos**: operación resta entre valores complejos.
- **Sel**: selección de datos para la siguiente cancelación.

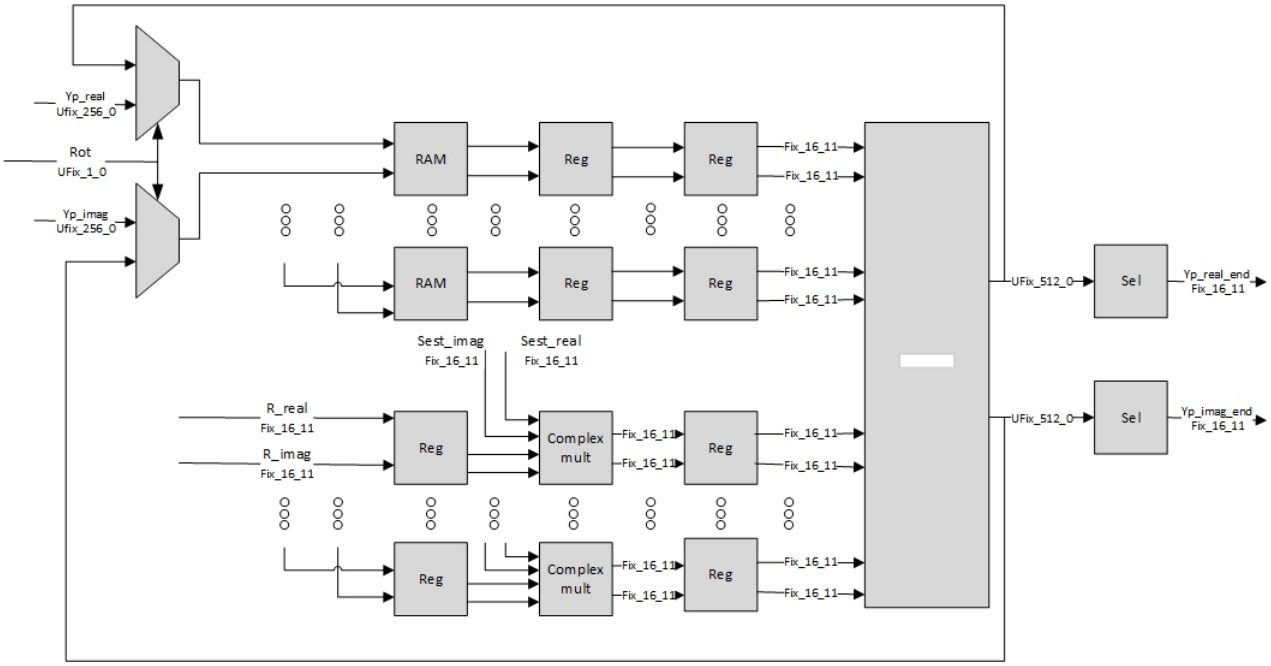

Figura 14: Arquitectura de **Arit_Oper**

Tabla 11: Entradas y salidas de **Arit_Oper**

Entrada	Tamaño (bits)	Proviene de	Descripción
Yp_real	$WL \times WL$	**Entrada del sistema** / **Arit_Oper**	Parte real del vector recibido.
Yp_imag	$WL \times WL$	**Entrada del sistema** / **Arit_Oper**	Parte imaginaria del vector recibido.
R_real	$WL \times WL$	**Entrada del sistema**	Parte real de la *k-ésima* columna de la matriz **R**.

R_imag	$WL \times WL$	**Entrada del sistema**	Parte imaginaria de la *k-ésima* columna de la matriz **R**.
Sest_real	*WL*	**Symbol_Detector**	Valor estimado real de la constelación.
Sest_imag	*WL*	**Symbol_Detector**	Valor estimado imaginaria de la constelación.
Rot	*1*	**CTRL**	Selector de vector.
Salida	**Tamaño (bits)**	**Sale a**	**Descripción**
Yp_real_end	*WL*	**Div_Factor**	*K-ésimo* elemento real del vector recibido.
Yp_imag_end	*WL*	**Div_Factor**	*K-ésimo* elemento imaginario del vector recibido.

En este módulo, primeramente, llega el vector recibido (**Yp_real** y **Yp_imag**), el cual es distribuido en 16 memorias RAM de puerto sensillo, para guardar un dato por cada memoria, ya que del vector recibido llegan 16 datos de 48 por cada ciclo de reloj, por lo que la cantidad de localidades requeridas en estas memorias RAM son 3. Estos datos, pasan por diferentes registros para dar un retardo de 3 ciclos para así operar con los valores estimados del módulo anterior y la *k-ésima* columna de la matriz **R**, la cual ingresa al módulo pasando por diferentes registros. Una vez obtenidos el resultado de la multiplicación compleja entre estos datos, se resta este valor del vector recibido, y así obtener el valor para la siguiente cancelación por medio del bloque **Sel**, el nuevo vector recibido que es guardado en la memoria RAM para operar cuando se lo indique la unidad de control.

4.2.6.1 RAM

El bloque **RAM** es una memoria RAM de puerto sensillo con 3 localidades de memoria, en la cual son guardados los datos del nuevo vector recibido, una memoria RAM por cada 3 valores de este vector, tardando 3 ciclos de reloj para entregar los datos por cada que se realice una operación.

4.2.6.2 Complex mult

El bloque **Comple mult** es el encargado de realizar la operación matemática de multiplicación compleja entre los valores de la *k-ésima* columna de la matriz **R** y el valor estimado obtenido por el módulo anterior. La ecuación 33 y 34 describe el funcionamiento de este módulo.

$$V_{real} = R_{real} * Sest_{real} - R_{imag} * Sest_{imag} \tag{33}$$

$$V_{real} = R_{real} * Sest_{imag} - R_{imag} * Sest_{real} \tag{34}$$

Donde las operaciones de multiplicación en este bloque son realizadas primero, para esperar en un registro para realizar las operaciones sumas y restas en el siguiente ciclo de reloj.

4.2.6.3 Reg

El bloque **Reg** es el encargado de retener los datos de las operaciones realizadas para cuando estas sean indicadas por medio de la unidad de control, además de tener un flujo de *pipeline* en la operación de la arquitectura.

4.2.6.4 Bloque menos

Este bloque se encarga de realizar la operación de resta compleja entre los valores del vector recibido y los obtenidos en el bloque de **Complex mult**, para así tener los datos para la siguiente cancelación e ir pasando los datos al bloque **RAM** para la siguiente

operación. La salida de este bloque lleva los 16 datos reales y 16 imaginarios del vector recibido (**Yp_real** y **Yp_imag**).

4.2.6.5 Sel

El bloque **Sel** selecciona el elemento *k-ésimo* del vector recibido para realizar la siguiente constelación por medio de las señales **Yp_real_end** y **Yp_imag_end**.

4.2.7 RA

El módulo **RA** es simplemente un arreglo de registros de almacenamiento para ir de un módulo a otro, el cual es habilitado por medio de la unidad de control por del pin *enable*. La Figura 15, se muestra la arquitectura del módulo **RA** y la Tabla 12, una explicación de sus entradas y salidas. Este módulo contiene el siguiente elemento:

- **Reg**: registro.

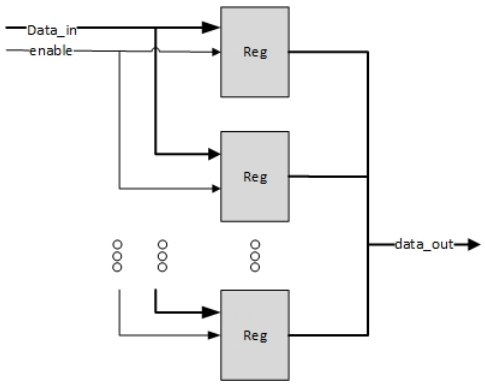

Figura 15: Arquitectura de **RA**

Tabla 12: Entradas y salidas de **RA**

Entrada	**Tamaño (bits)**	**Proviene de**	**Descripción**
Data_in	*Variable*	**Módulo anterior**	Datos del módulo anterior.
Enable	*1*	**CTRL**	Habilitador de registros.
Salida	**Tamaño (bits)**	**Sale a**	**Descripción**

Data_ou	*Variable*	**Módulo siguiente**	Datos del módulo siguiente.

Los datos son ingresados al módulo para colocardos en registros (Reg), **Data_in** son datos variables, que dependen de la cantidad de entradas de cada módulo anterior de donde se obtienen los datos, al igual que la salida **Data_out**. El bloque **Reg** ya se explicó en la sección 5.2.6.3.

4.2.8 CTRL

Para el diseño de la arquitectura, se optó por separar el control del sistema en dos procesos diferentes, uno para la parte de la cancelación (**CRL1**) y otro control para la parte de las operaciones aritméticas (**CTRL2**).

4.2.8.1 CTRL1

CTRL1 es el encargado del control de los datos de llegada del módulo **Div_Factor**, para que pasen por los registros correspondientes y además sean obtenidos en el momento que sean necesarios, la propagación de los valores de activación de este control se propaga por medio de registros para módulos posteriores.

4.2.8.2 CTRL2

CTRL2 es el encargado del control de los datos de llegada del módulo **Arit_Oper** y activado por al momento de que el módulo **Div_Factor** arroja un dato valido. Este control hace que los datos pasen por los registros correspondientes, y sean obtenidos en el momento necesario para que sean almacenados en las localidades de memorias RAM correspondientes al vector recibido, la propagación de los valores de activación de este control se propaga por medio de registros para módulos posteriores.

CAPÍTULO V

Resultados

En este capítulo se presentan los resultados más relevantes acerca del trabajo realizado, junto con análisis y comentarios sobre los mismos.

5.1 Resultados de implementación

La arquitectura fue descrita a nivel RTL usando Verilog HDL, utilizando una longitud de palabra WL = 16, parte entera IP = 5 y parte fraccionaria FP = 11, Sintetizado en un FPGA Xillinx AC701 bajo las siguientes condiciones:

- La síntesis fue habilitada por defecto (*default settings*) y no se seleccionó alguna restricción de usuario (*user constraints*).

  item No se usó ningún componente pre-diseñado proveniente de alguna librería de módulos.

- Se aplicó un esquema de redondeo de truncamiento simple al no requerir hardware adicional para su implementación.
- Todas las señales se presentaron en aritmética de punto fijo signada en complemento a dos.

La síntesis del detector se realizó con la ayuda del software vivado 2016.1, el dispositivo utilizado para sintetizar el diseño fue FPGA Xillinx Artix-7, dispositivo xc7a200t-2fbg676.

Los resultados de la síntesis del detector propuesto se resumen en la tabla 13. En ella se exponen el total disponible de cada uno de los recursos del FPGA Artix-7, la cantidad consumida por la arquitectura y el porcentaje que esto representa.

Tabla 13: Recursos utilizados para distintas constelaciones del detector

Constelación	***Slices LUTs***	***Slices Registers***	**DSPs48**	**Frecuencia máxima**
QPSK	4276 (3.17 %)	4157 (1.54 %)	64 (8.65 %)	21.433 MHz
16-QAM	4583 (3.40 %)	4159 (1.54 %)	64 (8.65 %)	21.433 MHz
64-QAM	4378 (3.25 %)	4162 (1.54 %)	64 (8.65 %)	21.433 MHz

Analizándo la tabla anterior, se observa una frecuencia de operación superior a los 20 MHz con un consumo de recursos muy reducido (menor al 10 %), con lo que queda de manifiesto que la arquitectura presenta una excelente relación velocidad–área. Por otro lado, no fue posible comparar los resultados de síntesis del detector, debido a que no existe en la literatura un proceso de síntesis o implementación como tal en escenarios similares a los reportados en este documento. Todo lo anterior, permite al bloque detector ser incorporado a cualquier sistema de recepción que trabaje con los estándares actuales de comunición.

5.2 Resultados de operación

Ahora, es importante determinar un aproximado de los ciclos necesarios en cada uno de los módulos presentados en el capítulo V. Por lo que, analizando cada uno de los módulo, primeramente se analiza **Div_Factor**. Este módulo, idependientemente la cantidad de operaciones que realiza, tarda 3, por lo tanto:

$$ciclos\_Div\_Factor = 3. \tag{35}$$

Posterior, se analiza el modulo **Hard_Demapper**, aunque este módulo es completamente combinacional, contiene un registro en la entrada y otro en la salida, por lo que:

$$ciclos\_Hard\_Demapper = 2. \tag{36}$$

Ya demapeado en símbolo, se necesita la estimación del mismo en el módulo **Symbol_Detector**. Aquí solamente, se realiza una busqueda de en una memoria ROM, por lo tanto:

$$ciclos\_Symbol\_Detector = 1. \tag{37}$$

Finalmente, se tiene el módulo **Arit_Oper**, que tiene dos registros intermedios, uno de salida y una memoria RAM, además la primera mitad tarda un ciclo más que la segunda mitad del proceso, por lo que:

$$ciclos\_Arit\_Oper = \begin{cases} 4 & si \quad N_D \leq 24 \\ 3 & cualquier\ otro\ caso \end{cases} \tag{38}$$

Con base en el análisis anterior y considerenado que se tienen N_D subportadoras de datos, el módulo **Arit_Oper** es necesario que se ejecute $N_D - 1$ veces y son N_D ciclos para obtener los datos, por lo tanto, el número de de ciclos de necesarios para obtener en su totalidad los índices estimados es:

$$ciclos\_totales = (ciclos\_Div\_Factor + ciclos\_Hard\_Demapper + ciclos\_Symbol\_Detector + ciclos\_Arit\_Oper + 1) \times N_D + N_D - 4$$

$$ciclos\_totales = (3 + 2 + 1) \times N_D + 4 \times \frac{N_D}{2} + 3 \times (\frac{N_D}{2} - 1) + N_D - 4$$

$$ciclos\_totales = (6 + 2 + 3/2 + 1) \times N_D - 4 - 3.$$

$$ciclos\_totales = \lfloor (10{,}5) \times N_D \rfloor - 7. \quad (39)$$

5.3 Resultados de verificación

Para la obtención de las gráficas de BER, se utiliza la plataforma de Co-simulación System Generator 2016.1, sustituyendo la función original de MATLAB por la arquitectura propuesta con el uso de matrices de canal específicas para un canal de comunicación V2V. El proceso de verificación del detector queda contestualizado en la Figura 16.

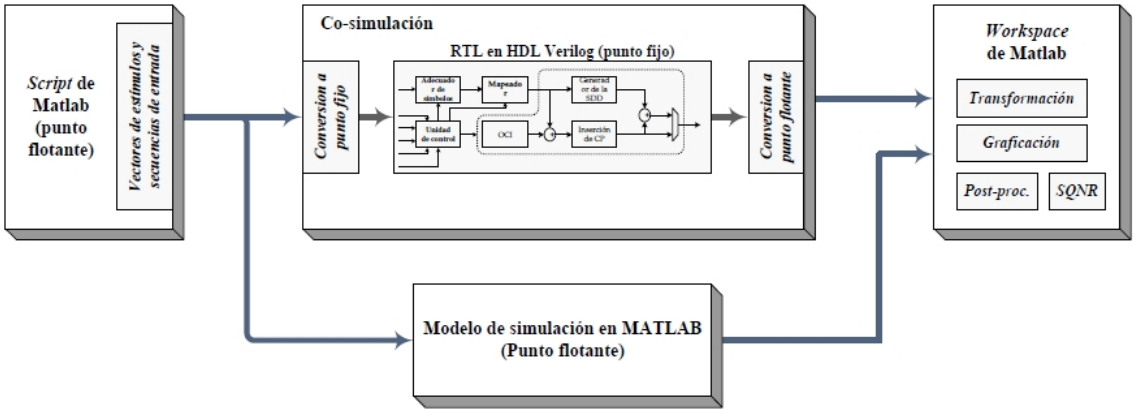

Figura 16: Esquema de verificación para las arquitecturas de hardware

5.3. Resultados de verificación

El desempeño de la arquitectura comparado con la simulación en aritmetica en punto fijo es mostrado en las Figuras 17, 18 y 19.

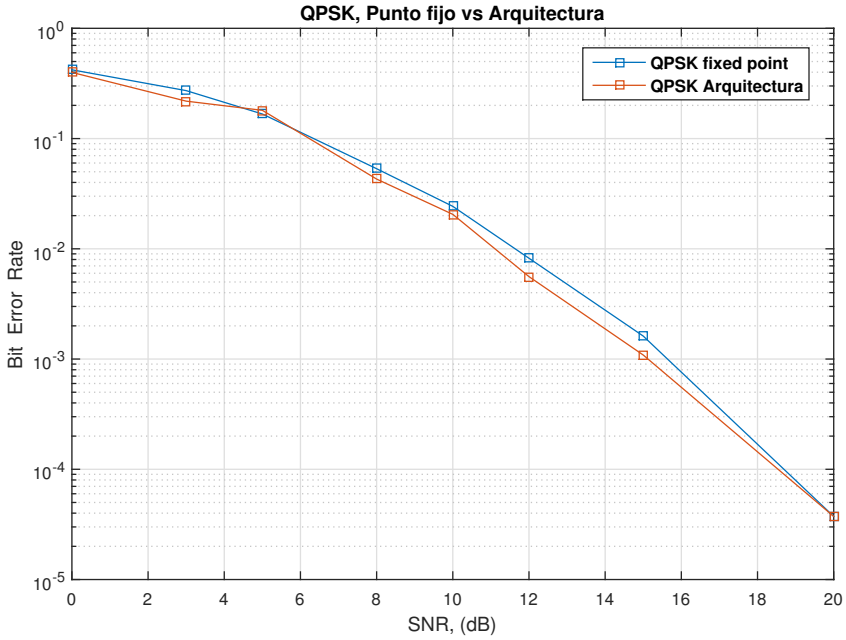

Figura 17: Comparación de BER en punto fijo vs arquitectura para QPSK

Para QPSK y 16-QAM, se muestra un rendimiento similar al rendimiento comparado con el analizado en punto fijo, lo que quiere decir que el dimencionamiento en la parte entera y fraccionaria de la arquitectura es correcta, aunque para el caso de 64-QAM en 30 SNR, se pierde el seguimiento de la gráfica en punto fijo, esto debido a la cuantizacion, al tener números muy pequeños en las matrices de canal.

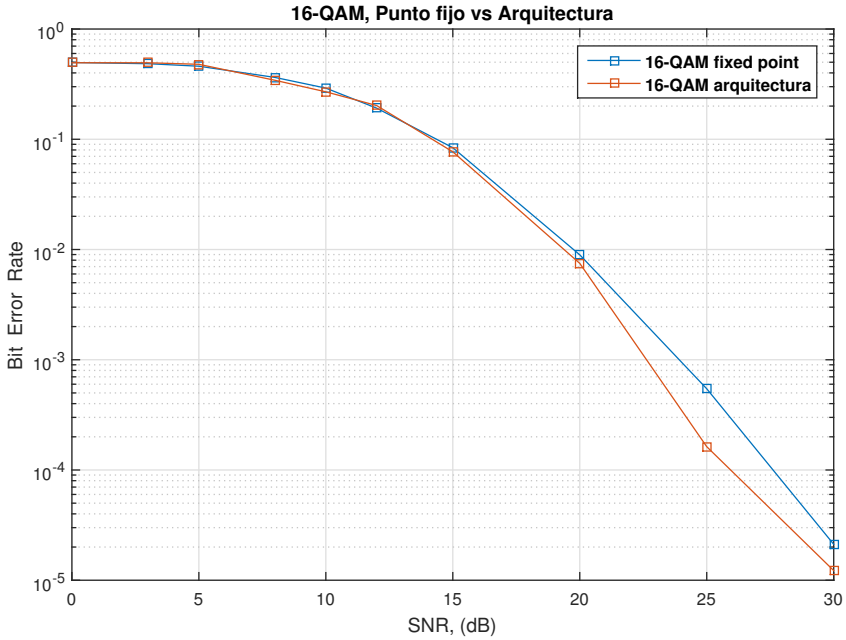

Figura 18: Comparación de BER en punto fijo vs arquitectura para 16-QAM

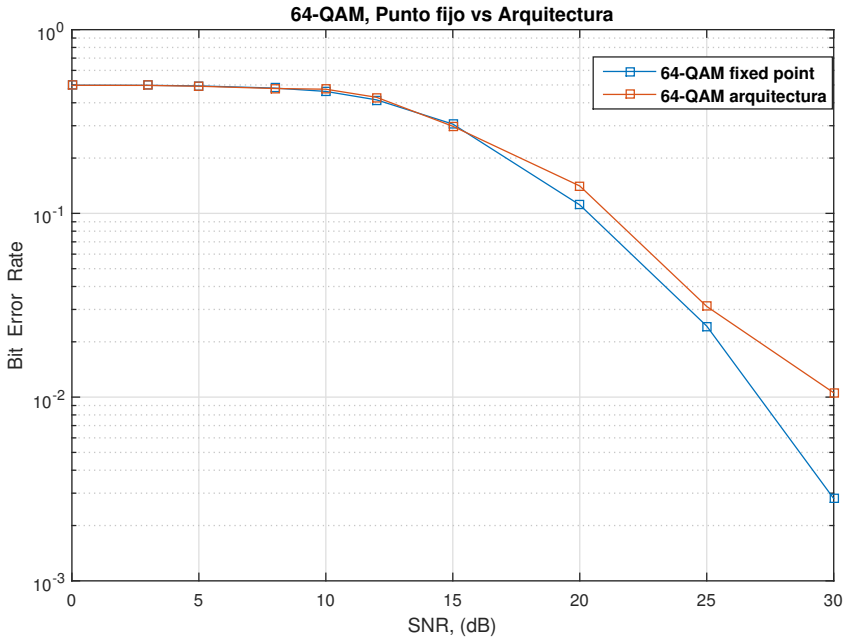

Figura 19: Comparación de BER en punto fijo vs arquitectura para 64-QAM

CAPÍTULO VI

Conclusiones

En este documento se propuso una arquitectura digital en hardware implementado en FPGA Artix-7 del algoritmo de detección de símbolo OSIC, con una baja complejidad de operaciones para un sistema de transmisión SISO-OFDM en sistemas V2V. A conocimiento del autor, no se ha presentado una arquitectura ya documentada de este estilo que implemente este algoritmo, por lo que hace esta tesis la primera en su clase.

Se presentaron los modulos necesarios que conforman la arquitectura del detector, con una descripción tanto de sus entradas y salidas, además de señalar la interacción entre ellas. Los resultados muestran que la implementación en punto fijo mantiene el rendimiento del algoritmo en punto flotante, es decir, las gráficas de BER de ambos modelos se comportan de manera muy similar.

Se observó que el número de recursos utilzados en la implementación en FPGA es menor al 10 % del total de la capacidad de la tarjeta en el caso más grande considerado, lo cual muestra que la arquitectura no necesita de una gran cantidad de recursos

para tener una detección de símbolo adecuada. Por lo anterior, permite al bloque detector ser incorporado a cualquier sistema de recepción que trabaje con los estándares actuales de comunición.

El máximo retardo o ruta crítica de la implementación se encuentra definida por el módulo **Arit_Oper** ya que aunque no es completamente combinacional, se tienen grandes recorridos de los datos. Lo cual afecta de manera negativa el desempeño general de la arquitectura en la frecuencia máxima de operación.

Los resultados obtenidos en este documento son considerados buenos en términos de funcionamiento y cantidad de recursos utilizados, considerando que es una primera implementación a un detector OSIC para un receptor SISO V2V implementado en una plataforma FPGA.

Como trabajo a futuro a investigar, se tienen optimizaciones típicas que se le pueden realizar a cualquier diseño, ya sea en hardware o software. Analizar las consecuencias que se tendrían, tanto positivas como negativas, en la mejora de desempeño contra complejidad de implementación al cambiar la forma en la que es realizado el módulo **Arit_Oper** es una buena opción de partida para mejorar la arquitectura.

Bibliografía

[1] Giovanni Nardini, Antonio Virdis, Claudia Campolo, Antonella Molinaro, and Giovanni Stea. Cellular-V2X Communications for Platooning: Design and Evaluation. *Sensors*, 18(5):1527, May 2018.

[2] Tom Glenn McGiffen, Sven Beiker, and Arogyaswami Paulraj. Motivating Network Deployment: Vehicular Communications. *IEEE Vehicular Technology Magazine*, 12(3):22–33, September 2017.

[3] Gregory ; Yoon Rebecca ; Fikentscher Joshua ; Doyle Charlene ; Sade Dana ; Lukuc Mike ; Simons Jim ; Wang Jing Harding, John ; Powell. Vehicle-to-Vehicle Communications: Readiness of V2V Technology for Application. Tech Report DOT HS 812 014, U.S. Departament of Transportation, United States, August 2014.

[4] Innovation Science and Economic Development Canada. Intelligent Transportation Systems — Dedicated Short Range Communications (DSRC) — On-Board Unit (OBU). Technical report, 2017.

[5] David W. Matolak. V2V Communication Channels: State of Knowledge, New Results, and What's Next. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Moni Naor, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Madhu Sudan, Demetri Terzopoulos, Doug Tygar, Moshe Y. Vardi, Gerhard Weikum, Marion Berbineau, Magnus Jonsson, Jean-Marie Bonnin, Soumaya Cherkaoui, Marina Aguado, Cristina Rico-Garcia, Hassan Ghannoum, Rashid Mehmood, and Alexey Vinel, editors, *Communication Technologies for Vehicles*, volume 7865, pages 1–21. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[6] Andreas Molisch, Fredrik Tufvesson, Johan Karedal, and Christoph Mecklenbrauker. A survey on vehicle-to-vehicle propagation channels. *IEEE Wireless Communications*, 16(6):12–22, December 2009.

[7] Naima Sofi, Fethi Tarek Bendimerad, and Fatima Debbat. Compromise between spectral efficiency and interference cancellation in OFDM system. In *2017 International Conference on Engineering & MIS (ICEMIS)*, pages 1–7, Monastir, May 2017. IEEE.

[8] P.W. Wolniansky, G.J. Foschini, G.D. Golden, and R.A. Valenzuela. V-BLAST: an architecture for realizing very high data rates over the rich-scattering wireless channel. In *1998 URSI International Symposium on Signals, Systems, and Electronics. Conference Proceedings (Cat. No.98EX167)*, pages 295–300, Pisa, Italy, 1998. IEEE.

[9] Adnan Saifullah, Linbo Zhang, Ayoob Muhammad, and Irshad Muhammad. Low Complexity MIMO Detection Algorithm by Combining Modified OSIC and ML Detection. volume 8, pages 192–195, Harbin Engineering University, 2016.

[10] D. Wubben, R. Bohnke, V. Kuhn, and K.-D. Kammeyer. MMSE Extension of V-BLAST Based on Sorted QR Descomposition. volume 1, pages 508–512, Orlando, FL, USA, 2003.

[11] Tejas M. Bhatt and Dennis McCain. Matlab as a development environment for FPGA design. In *Proceedings of the 42nd annual conference on Design automation - DAC '05*, page 607, San Diego, California, USA, 2005. ACM Press.

[12] MathWorks. Fixed-Point Basics in MATLAB, 2019.

[13] Anuj Vaishnav, Khoa Dang Pham, and Dirk Koch. A Survey on FPGA Virtualization. In *2018 28th International Conference on Field Programmable Logic and Applications (FPL)*, pages 131–1317, Dublin, Ireland, August 2018. IEEE.

[14] Divyang Rawal and Nikhil Sharma. MIMO Free Space Optical Communication Systems with Low Complexity QR-OSIC Detector. volume 14, pages 1–5, 2017.

[15] J.A. Del Puerto-Flores, R. Parra, F. Peña, J. Cortez, and E. Romero. Evaluation of OFDM Systems With Virtual Carriers Over V2V Channels. pages 882–886, Vancuber, 2018.

[16] J.A. Del Puerto-Flores, R. Parra, F. Peña, and J. Cortez. Performance Evaluation of Turbo Decoding in DFTS-OFDM Systems over V2V Channel. pages 1–5, Guadalajara, 2018.

[17] IEEE Draft Standard for Information Technology - Telecommunications and information exchange between systems - Local and metropolitan area networks - Specific requirements - Part 11: Wireless LAN Medium Access control (MAC) and Physical Layer (PHY) specifications Amendment : Wireless Access in Vehicular Environments. 2010.

[18] Lang Tong, B.M. Sadler, and Min Dong. Pilot-assisted wireless transmissions - General model, design criteria, and signal processing. *IEEE Signal Processing Magazine*, 21(6):12–25, November 2004.

[19] Guillermo Acosta-Marum and Mary Ann Ingram. Six time- and frequency- selective empirical channel models for vehicular wireless LANs. *IEEE Vehicular Technology Magazine*, 2(4):4–11, December 2007.

[20] I. Sen and D.W. Matolak. Vehicle–Vehicle Channel Models for the 5-GHz Band. *IEEE Transactions on Intelligent Transportation Systems*, 9(2):235–245, June 2008.

[21] Xiang Cheng, Qi Yao, Miaowen Wen, Cheng-Xiang Wang, Ling-Yang Song, and Bing-Li Jiao. Wideband Channel Modeling and Intercarrier Interference Cancellation for Vehicle-to-Vehicle Communication Systems. *IEEE Journal on Selected Areas in Communications*, 31(9):434–448, September 2013.

[22] D. Wubben, R. Bohnke, V. Kuhn, and K.-D. Kammeyer. MMSE extension of V-BLAST based on sorted QR decomposition. In *2003 IEEE 58th Vehicular Technology Conference. VTC 2003-Fall (IEEE Cat. No.03CH37484)*, pages 508–512 Vol.1, Orlando, FL, USA, 2003. IEEE.

[23] Jose Alberto Del Puerto Flores. *Sistema de comunicación multiportadora para el estándar 802.11p utilizando precodificación frecuencial y cancelación no lineal de interferencia*. Tesis, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Enero 2019.

[24] Juan M Bueno. *Introducción a la óptica instrumental*. EDITUM, 1999.

[25] Daniel Menard, Daniel Chillet, and Olivier Sentieys. Floating-to-fixed-point conversion for digital signal processors. *EURASIP Journal on Advances in Signal Processing*, 2006(1):096421, 2006.

[26] Eduardo Romero Aguirre. *Arquitecturas digitales de procesamiento de señales para sistemas de comunicación con entrenamiento implicito*. Tesis, Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, Diciembre 2012.

[27] Alberto Rodriguez-Garcia, Luis Pizano-Escalante, Ramón Parra-Michel, O Longoria-Gandara, and J Cortez. Fast fixed-point divider based on newton-raphson method and piecewise polynomial approximation. In *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–6. IEEE, 2013.