



INSTITUTO TECNOLÓGICO DE SONORA
Educar para Trascender

Propuesta de Arquitectura en FPGA de un Detector de Símbolos Near-ML para un
receptor OFDM-V2V

TESIS

QUE PARA OBTENER EL GRADO DE
MAESTRO EN CIENCIAS DE LA INGENIERÍA

PRESENTA

Aarón Escoboza Villegas

CIUDAD OBREGÓN, SONORA
DICIEMBRE 2021

Índice

CAPÍTULO I. INTRODUCCIÓN	5
1.1 Antecedentes	5
1.2 Planteamiento del problema	10
1.3 Objetivo	11
1.4 Hipótesis	11
1.5 Justificación	11
1.6 Delimitaciones	12
1.7 Limitaciones	12
CAPÍTULO II. FUNDAMENTOS TEÓRICOS	13
2.1 El estándar IEEE 802.11p	13
2.2 Pilotos en el estándar 802.11p	14
2.3 Canal de comunicación V2V	15
2.4 Sistemas de comunicación V2V	16
2.5 Detección de datos	17
2.5.1 Modelo del receptor	18
2.5.2 Detector OSIC	20
2.5.3 Detector QR-ML convencional	21
2.5.4 Detector V2V Near ML	22
2.7 Redes de ordenamiento	24
2.7.1 Medio limpiador	25
2.7.2 Ordenador bitónico	26
2.7.3 Red fusionadora	26
2.7.4 Construcción de una red de ordenamiento	27
2.8 Ordenamiento secuencial	28
2.8.1 Máquina de ordenamiento de inserción	28
2.8.2 Odd-even sorting network	29
2.8.3 Ordenamiento basado en FIFOs	30
2.6 Diseño digital a nivel RTL en FPGA	31
2.6.1 Mapeo algoritmo a arquitectura	31
2.6.2 Modelo de oro	32
2.6.3 Adecuaciones al algoritmo	32

2.6.4 Análisis de punto fijo	33
2.6.5 Diseño de la arquitectura de hardware	35
2.6.6 Implementación y verificación	36
2.9 Diseño HLS en FPGA	36
CAPÍTULO III. MÉTODO	39
3.1 Sujeto	39
3.2 Procedimiento	39
3.3 Materiales y Herramientas	42
CAPÍTULO IV. DESARROLLO	43
4.1 Desarrollo del modelo algorítmico del sistema	43
4.1.1 Modelo de oro del detector	44
4.1.2 Sistema SISO V2V	45
4.1.3 Análisis de punto fijo	47
4.2 Diseño HLS de los ordenadores	50
4.2.1 Selección de algoritmos	50
4.2.2 Codificación	51
4.2.4 Programación del <i>testbench</i> en alto nivel	52
4.3 Propuesta de arquitectura	53
4.3.1 Bloque top	53
4.3.2 El bloque <i>error</i>	55
4.3.3 El bloque <i>addrToData</i>	56
4.3.4 El bloque <i>sorter</i>	57
4.3.5 El bloque <i>dataAndPointers</i>	63
4.3.6 El bloque <i>survivors</i>	64
4.3.7 El bloque <i>detection</i>	65
4.3.8 El bloque <i>topControlUnit</i>	66
CAPÍTULO V. RESULTADOS	68
5.1 Síntesis	68
5.1.1 Diseño HLS	68
5.1.2 Arquitectura	69
5.2 Latencia	70
5.2.1 Diseño HLS	70
5.2.2 Arquitectura	70
5.3 Verificación	71

5.3.2 Diseño HLS	71
5.3.3 Arquitectura	72
CAPÍTULO VI. CONCLUSIONES	74
Bibliografía	75
Anexo – Artículo publicado	79

CAPÍTULO I. INTRODUCCIÓN

El presente capítulo presenta una breve descripción sobre el área de investigación del trabajo, se contextualiza y se define el problema a resolver, además, se plantean tanto el objetivo general como los específicos, planteando una posible solución al problema. Por otro lado, se mencionan los beneficios del trabajo resaltando la contribución del mismo, se define el alcance del trabajo, así como las limitaciones del mismo que impiden ir más allá del objetivo definido.

1.1 Antecedentes

Con el incremento de los vehículos en carretera, el control de tráfico se ha convertido en un gran reto hoy en día. La tasa de vehículos crece exponencialmente diariamente, solo en el 2010 había más de mil millones de vehículos. Con esto, los accidentes aumentaron causando aproximadamente 1.25 millones de muertes alrededor del mundo en 2010 [1].

En Estados Unidos se perdieron 6.9 mil millones de horas en carreteras y 3.1 mil millones adicionales de galones de gasolina debido a embotellamientos, lo cual representó una pérdida anual de 160 mil millones [2]. En Europa alrededor de 40 mil

fallecieron y 1.7 millones fueron heridos en accidentes de tráfico [3]. Para tratar de resolver los problemas como los anteriores, las compañías, instituciones académicas, y agencias gubernamentales han incorporado tecnología de comunicación en vehículos y en infraestructura del transporte [4].

Se han integrado las comunicaciones vehículo a vehículo (V2V), vehículo a infraestructura(V2I), vehículo a peatones (V2P) y vehículo a red (V2N), en conjunto denominadas comunicaciones V2X. ~~V2X~~ junto con las capacidades de detección de vehículos provee un soporte para aplicaciones relacionadas con la seguridad, infoentretenimiento para pasajeros y optimización del tráfico de vehículos [4]. En la Figura 1 se muestra las distintas formas de V2X [5].

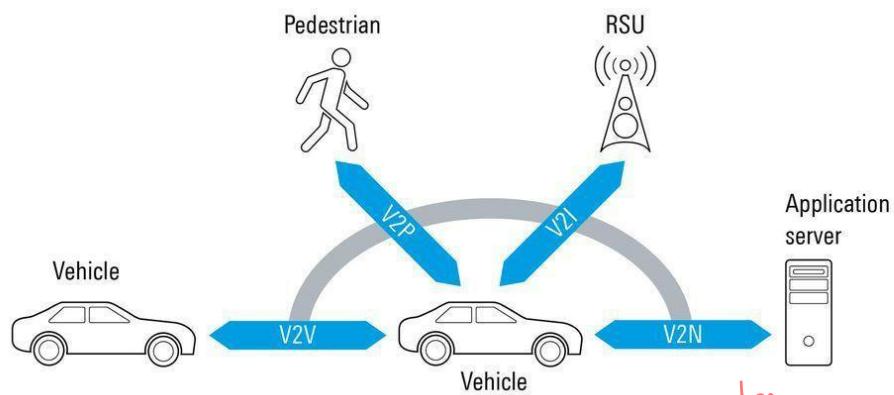


Figura 1. Una ilustración simple de las comunicaciones V2X.

V2V es una tecnología que presenta una comunicación en red para vehículos, de tal manera que puedan estar conectados entre sí. Fue desarrollado y demostrado en 2005 por General Motors y promete una comunicación confiable y fuerte entre usuarios [6]. V2V puede ser usada para mandar advertencias al conductor en situaciones consideradas críticas y de alto riesgo (embotellamiento, cambio de carril) o mandar mensajes de ayuda después de un accidente de tráfico severo [7], [8].

V2V incorpora las siguientes tecnologías relacionadas con la seguridad [9]:

- Sistema de prevención de colisiones.

- Sistema de advertencia al conductor.
- Sistema de estabilidad del vehículo.
- Asistencia automatizada de frenado de emergencia.

Algunas de las aplicaciones son las siguiente [10] :

- Advertencia de colisión frontal.
- Advertencia de no pasar.
- Advertencia de punto ciego.
- Advertencia de cambio de carril.
- Luz de freno de emergencia electrónica.
- Asistente de giro a la izquierda.

Para soportar las comunicaciones V2X se han creado dos soluciones: Comunicaciones dedicadas de rango corto (Dedicated Short Range Communications por sus siglas en inglés DSRC) y tecnologías de redes celulares [2]. DSRC consiste en transceptores de radiocomunicación en los vehículos y unidades colocadas en los laterales de la carretera. Fue desarrollada para cumplir con los requerimientos de un ambiente dinámico como alta movilidad, baja latencia, y un rango considerado bueno de comunicación [8].

DSRC tiene 70 MHz en la banda de 5.9 GHz y es la tecnología por defecto en las aplicaciones de seguridad en V2V porque presenta una latencia menor a otras existentes, se basa en el estándar IEEE 802.11p y se espera que todos los nuevos vehículos la soporten, de acuerdo ~~con~~ ^{o lo reportado por} la administración nacional de seguridad del tráfico en carreteras (National Highway Traffic Safety Administration, por sus siglas en inglés NHTSA) [11], [12].

Las tecnologías celulares incluyen tanto LTE (Long Term Evolution), 5G como los estándares antiguos. Con respecto a DSRC, estas tecnologías presentan un conjunto de ventajas como: mayor cobertura, infraestructura, seguridad, garantías de QoS

(Quality of service) y escalabilidad robusta. Sin embargo, esas ventajas conducen a una arquitectura centralizada, latencia significativa de un extremo a otro, dependencia de la conectividad con la infraestructura y un costo mayor por el uso de la [13].

la que?

Existen algunos retos en las comunicaciones vehiculares principalmente por el comportamiento estocástico del canal. El estándar IEEE 802.11p utiliza OFDM para DSRC, de tal manera que reduce el desvanecimiento que produce el movimiento de las terminales de comunicación y los objetos en el medio ambiente [14]. OFDM es un esquema de modulación multiportadora, los símbolos se transmiten en paralelo en múltiples subportadoras. La técnica reduce el ancho de banda en la subportadora y logra robustez contra el desvanecimiento en frecuencia, debido al retardo de las señales en las diferentes trayectorias de propagación [15].

Una de las actividades importantes a realizar en cualquier sistema de comunicación, como en un sistema SISO-OFDM para comunicaciones V2V, es la detección de símbolos en el receptor. El detector ML (Maximum-likelihood por sus siglas en inglés ML) presenta la mínima tasa de error de bit. Sin embargo, tiene una dependencia exponencial en la complejidad, por lo tanto, se necesitan algoritmos que alcancen tasas de error similares a la detección ML [16].

Para el desarrollo de tecnología en V2V es necesario la implementación de nuevos algoritmos de detección en plataformas que permitan su construcción física lo más rápido y eficiente posible con el objetivo de medir su viabilidad. El FPGA (Field-Programmable Gate arrays por sus siglas en inglés FPGA) es una tecnología en uso que provee grandes ventajas para implementar algoritmos y prototipos a nivel de hardware [17].

Un FPGA consiste de un arreglo de bloques lógicos programables de diferentes tipos, lógica general, memoria y bloques multiplicadores [18]. Alrededor del arreglo se encuentran los bloques de entrada y salida que permiten la comunicación hacia el mundo exterior como se muestra en la Figura 2.

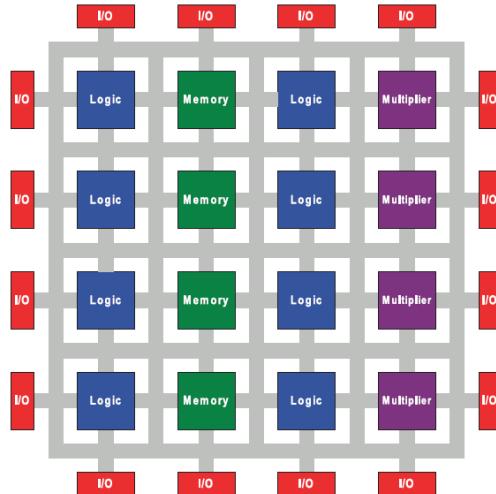


Figura 2. Estructura básica de un FPGA .

Algunos problemas en trabajar al nivel de registro de transferencia (Register Transfer Level, RTL) utilizando lenguajes de descripción de hardware como VHDL o Verilog para su posterior síntesis en un FGPA, radica en que no son tan poderosos como los lenguajes de alto de nivel. Esto provoca códigos largos, incremento en la probabilidad de errores y un gran consumo de tiempo al hacer modificaciones en el diseño. Por los problemas anteriores, las herramientas de síntesis de alto nivel (High-Level Synthesis, HLS) son utilizadas [19].

HLS crea un código en un lenguaje de descripción de hardware tomando la descripción del algoritmo o sistema hecho en un lenguaje de alto nivel como C o C++ y directivas que describen restricciones de la arquitectura [20]. El flujo de diseño se muestra en la Figura 3 [21], [22].

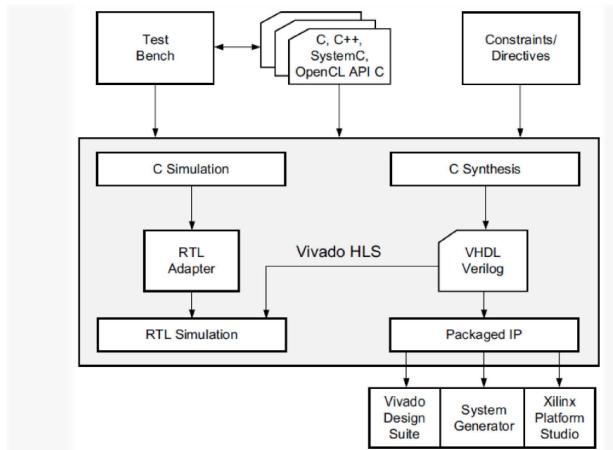


Figura 3. Flujo de diseño en Vivado.

Flujo de diseño HLS en Vivado

junto, sin guion

1.2 Planteamiento del problema

Los sistemas OFDM sufren de un fenómeno llamado interferencia inter-portadora (inter-carrier interference, ICI), el cual se presenta en mayor medida en la comunicación V2V por las altas frecuencias de dispersión Doppler producidas por la alta movilidad en los enlaces vehiculares. El rendimiento general de los sistemas OFDM-V2V es reducido debido a la presencia de ICI provocando que la tarea de detección de datos sea agotadora [23].

La detección de datos es un proceso que conlleva una complejidad computacional importante en el receptor, la factibilidad de cualquier sistema diseñado para funcionar en canales V2V estará comprometida con el grado de complejidad del detector utilizado. El detector Near-ML es de baja complejidad y mejora el desempeño de BER (Bit Error Rate) [24], sin embargo, no existen arquitecturas de hardware que permitan su implementación en un FPGA. Su factibilidad de implantación en un FPGA (Incluso a nivel propio).

1.3 Objetivo

Desarrollar una propuesta de arquitectura de hardware del algoritmo Near-ML e implementar en HDL y HLS la etapa de ordenación de datos para la detección de símbolos en el receptor de un sistema SISO-ODFM en ambientes vehiculares V2V con la finalidad de conocer la viabilidad de incorporarlo a los estándares de comunicación actuales.

1.4 Hipótesis

Es posible proponer una arquitectura digital de un detector de símbolos bajo el esquema de modulación OFDM y criterio de decisión Near-ML. Además, realizar la implementación del bloque más importante del algoritmo en HDL y HLS.

1.5 Justificación

En la literatura sólo se han reportado resultados de simulación sobre el rendimiento del algoritmo, por lo tanto, no existe una investigación que haya implementado en hardware el algoritmo Near-ML y obtenido métricas de eficiencia. Con esto, se podrá corroborar la factibilidad y la viabilidad de ser introducido en las comunicaciones vehiculares y en los distintos estándares de comunicaciones inalámbricas móviles contemporáneos.

Este trabajo tendrá un impacto en una de las áreas con mayor crecimiento en los últimos años, las comunicaciones vehiculares. Normalmente, los receptores en V2V requieren implementaciones en hardware eficientes, que operen a mayores velocidades de transmisión, alta calidad de servicio y sobre todo de baja complejidad. Una forma de cumplir con lo anterior es implementar mejores y nuevos algoritmos como el que se presenta en esta tesis.

Finalmente, una vez terminado el trabajo se podría continuar con la implementación de cada bloque de la arquitectura y posteriormente tenerlo en un circuito integrado de uso comercial. Lo anterior, permitirá agregarlo a las aplicaciones V2V actuales y emergentes, en consecuencia, la complejidad y el consumo de potencia en los equipos de recepción del vehículo disminuirá.

1.6 Delimitaciones

Este trabajo abarca únicamente la propuesta de la arquitectura del algoritmo de detección y la implementación de uno de los bloques más importantes y de mayor uso.

No se tocará el tema de diseñar y simular el algoritmo debido a que fue un trabajo previo de un ex alumno de maestría.

abordaré

ese tema ya
fue llevada a cabo
en un trabajo previo
de maestría a [REF]

1.7 Limitaciones

Los recursos institucionales como el laboratorio no están disponibles por lo que podría poner en riesgo la etapa de validación física del ordenador si es necesaria. Otra limitación es el tiempo, una parte del trabajo será realizada en C++ en una plataforma que soporta la síntesis de programas en alto nivel y la siguiente parte en un lenguaje de descripción de hardware.

CAPÍTULO II. FUNDAMENTOS TEÓRICOS

En este capítulo se abordan los conocimientos generales para la correcta comprensión del presente trabajo. Asimismo, se mencionan aquellos trabajos actuales similares al área de estudio planteada en esta investigación.

2.1 El estándar IEEE 802.11p

La comunicación V2V fue estandarizada con el nombre de comunicaciones dedicadas a corto alcance (Dedicated short-range communications) IEEE 802.11p en el año 2010 [25]. El estándar hace uso de la capa física correspondiente al estándar IEEE 802.11a, el cual opera en ambientes interiores y de baja movilidad. Debido a que el estándar 802.11a fue diseñado para entornos donde no existe alta movilidad del receptor y del transmisor, el rendimiento de los sistemas V2V no cumple con el desempeño deseable.

Una de las principales diferencias entre los estándares radica en el preámbulo y duración del símbolo. En la Figura 4 se muestra la trama de transmisión, consta de un preámbulo de dos símbolos de entrenamiento, un campo para señal señal y otro para los datos. La sincronía del sistema se realiza con los primeros 10 símbolos cortos de entrenamiento de $1.6 \mu s$ de duración. La estimación del canal y sincronización final se

realizan con dos símbolos de entrenamiento largos de $6.4 \mu s$ de duración. La última parte de la trama es para el campo de la señal que contiene información sobre longitud de la trama, modulación y esquemas de codificación en la carga útil.

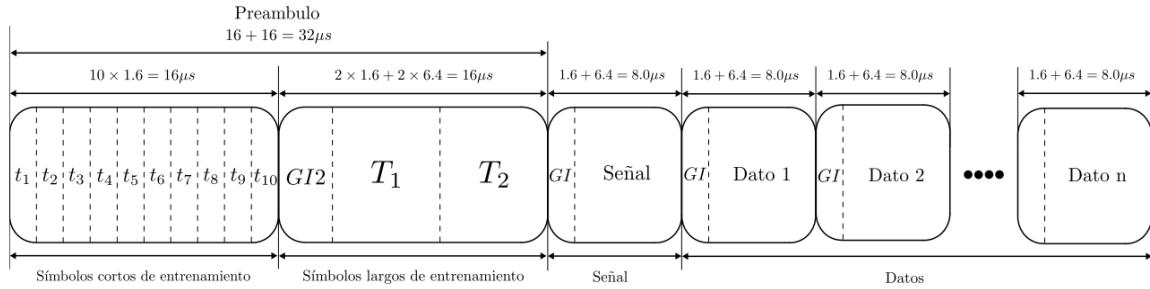


Figura 4. Trama del estándar 802.11p.

2.2 Pilotos en el estándar 802.11p

El escenario de alta movilidad en los ambientes V2V producen una respuesta al impulso del canal variante en el tiempo, por cada símbolo en OFDM se presentan variaciones temporales de los coeficientes del canal. Existen 64 subportadoras por símbolo OFDM, incluyendo 48 subportadoras de datos, 4 subportadoras pilotos en los índices -21, -7, 7 y 21 como se observa en la Figura 5.

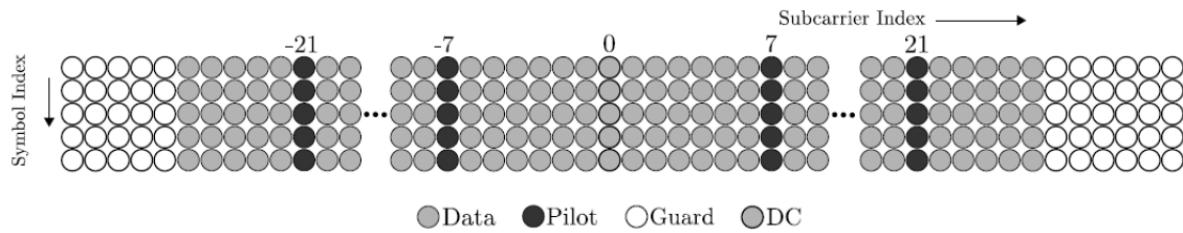


Figura 5. Asignación de pilotos en 802.11p.

En [2] se demuestra que la asignación de pilotos anteriormente descrita, no es la mejor para la estimación de un canal V2V. Sin embargo, al usar receptores con esquemas de estimación de canal iterativos y detecciones basadas en la técnica de mínimo error cuadrático medio (MMSE) se logra rendimientos aceptables después de la tercera iteración [26].

2.3 Canal de comunicación V2V

Existen una gran cantidad de investigaciones para clasificar y modelar los canales V2V. Algunas clasificaciones son expuestas en [27], [28], [29], [30], [31], [32], [33]. En estos dos últimos trabajos, se expone la clasificación de escenarios V2V posibles, dentro de los cuales destacan los siguientes: V2V-Autopista con dirección opuesta, V2V-Urbano en avenida principal con dirección opuesta, V2V sub-urbano, V2V en autopista. Lo anterior se resume en la Tabla 1, publicada por [28].

Tabla 1. Características de los diversos escenarios V2V.

Escenario	Velocidad (Km/h)	Desplazamiento Doopler (Hz)	Retardo máximo (μ s)
V2V autopista, dirección opuesta	104	1000-1003	0.3
V2V urbano avenida principal, dirección opuesta	32-48	300	0.5
V2V autopista	104	600-700	0.4
V2V urbano avenida principal, misma dirección	32-48	400-500	0.4
V2V suburbano calle, misma dirección	32-48	300-500	0.7
V2V autopista, misma dirección	104	900-1150	0.7

Tomando como referencia lo expuesto en los diversos trabajos de la literatura, se concluye que un modelo de canal simple es insuficiente para modelar con precisión el entorno adverso de un canal V2V. Por este motivo, se usan canales estacionarios basados en el modelo TDL (Tapped ~~Delay~~ Line) obteniendo la variación presente en los *Delay*

canales V2V. La respuesta al impulso del canal varía temporalmente en el periodo de símbolo OFDM, tal variación es la causante de la interferencia entre portadoras (ICI) degradando el rendimiento de la modulación OFDM [33].

2.4 Sistemas de comunicación V2V

En la Figura 6 se muestra la estructura básica de un sistema de comunicación V2V, en la cual se basan gran parte de trabajos presentes en el estado del arte. Las etapas de estimación y ecualización representan dos de los procesos más relevantes durante la recepción. El desempeño general de un sistema V2V va estar estrechamente relacionado al desempeño de los dos procesos anteriormente mencionados.

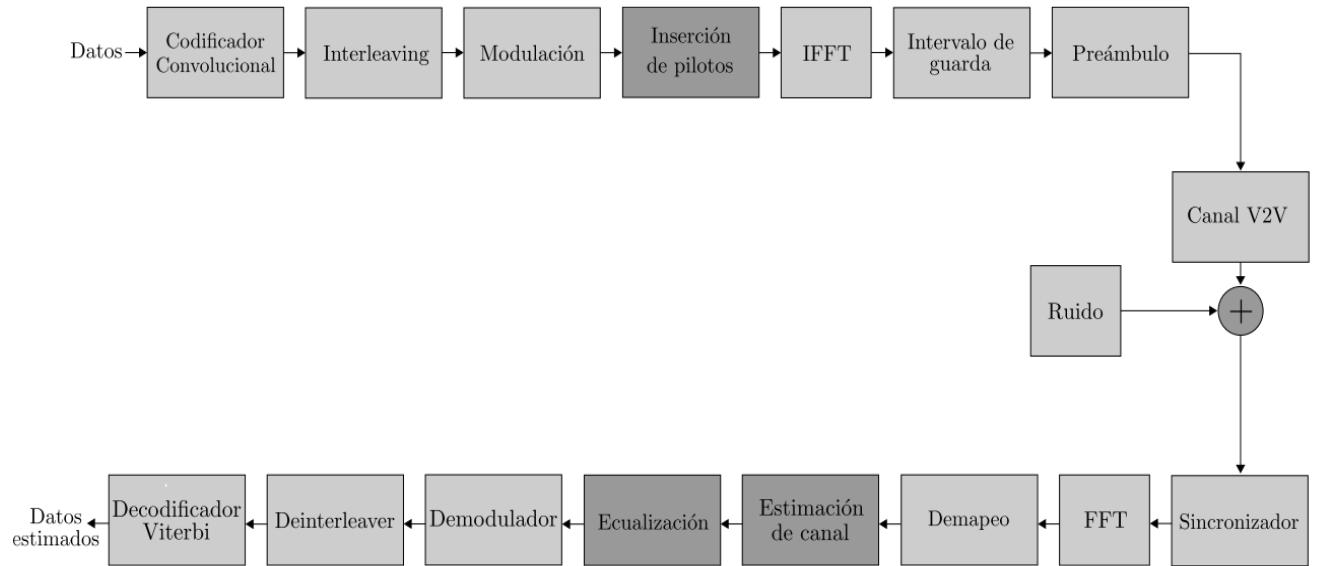


Figura 6. Diagrama a bloques de un sistema V2V típico.

Con la finalidad de afrontar y mitigar los inconvenientes producidos por la alta movilidad del entorno adverso V2V, se han presentado una gran cantidad de trabajos que abordan la problemática desde diversos enfoques, tales como:

- Algoritmos de asignación de pilotos.
- Estimadores de canal en el dominio temporal.
- Estimadores de canal en el dominio frecuencial.

- Detectores lineales.
- Detectores no lineales.

En el estado del arte, es conocido que la estimación de parámetros en el receptor se complica en un sistema V2V debido a la ICI, la cual afecta a las subportadoras piloto, necesarias para la correcta estimación del canal. El estimador de canal reportado en [34] es adecuado para operar en canales altamente variantes.

2.5 Detección de datos

Uno de los procesos con mayor grado de complejidad computacional en el receptor es la detección de datos, la factibilidad de un sistema diseñado para funcionar en tiempo real en un canal V2V dependerá principalmente del grado de complejidad del detector utilizado. El detector ML conocido como el detector óptimo tiene una complejidad de $O(N_D \Omega^{ND})$ [35]. Se puede apreciar que la complejidad aumenta de forma abrupta con el tamaño de la constelación Ω y número de subportadoras de datos N_D . En consecuencia, no resulta viable implementar tal detector. Actualmente, existen una serie de detectores con menor complejidad y rendimiento similar al detector ML. Algunos de ellos han sido diseñados para el sistema de espacio-tiempo de laboratorios Bell vertical (Bell Laboratories Layer Space-Time, V-BLAST) [36]. Uno de los más importantes y conocidos es el detector esférico [37].

Detectores no lineales basados en el algoritmo M en conjunto con la descomposición QR de la matriz de canal, ha comenzado a tener un gran auge [38], [39]. Recientemente, una gran contribución dentro de este, los OSIC (Ordering Successive Interference Cancellation), QR-ML y V2V Near-ML. Los resultados de simulación han comprobado que son adecuados para mitigar la ICI y obtienen desempeños en términos de BER favorables [24].

2.5.1 Modelo del receptor

Es posible representar la señal del k -ésimo símbolo que llega al receptor en su representación compleja banda base con la convolución discreta:

$$y^k[n] = \sum_{l=0}^{L-1} h^k[n, l] x^k[< n - l >_N] + w^k[n] \quad (1)$$

Donde $n = \{0, \dots, N - 1\}$, $l = \{0, \dots, L - 1\}$, L representa la CIR (Channel Impulse Response), $h^k[n, l]$ la CIR del k -ésimo bloque en el instante n para un impulso de entrada en l muestras pasadas, $x^k[n]$ el k -ésimo símbolo OFDM cuyo tamaño es N , por último $w^k[n]$ es el AWGN (Additive White Gaussian Noise) complejo.

La ecuación (1) puede ser vista en su forma matricial como:

$$y^k = H^k x^k + w^k \quad (2)$$

donde:

$$\begin{aligned} y^k &= [y^k[0] \ y^k[1] \ \dots \ y^k[N - 1]]^T, \\ x^k &= [x^k[0] \ x^k[1] \ \dots \ x^k[N - 1]]^T, \\ w^k &= [w^k[0] \ w^k[1] \ \dots \ w^k[N - 1]]^T \end{aligned}$$

el ruido w^k es de media cero y varianza $\sigma_w^2 = N_0/2$. Los coeficientes del CIR son representados por la matriz H^k de dimensión $N \times N$ de la siguiente forma:

$$[H^k]_{n,n'} = h^k[n, < n - n' >_N] \quad (3)$$

donde $n, n' = \{0, \dots, N - 1\}$, $< >_N$ representa el operador módulo N , la CIR es considerada cero para $< n - n' >_N > L - 1$. Para obtener el símbolo OFDM en FD (Frequency Domain) con CP (Cyclic Prefix) removido, se multiplica la ecuación (2) por

la DFT (Discrete Fourier Transform) normalizada:

$$[F]_{n,n'} = \frac{1}{\sqrt{N}} e^{(-j2\pi nn'/N)} \quad (4)$$

la multiplicación resulta en:

$$u^k = FH^k x^k + z^k \quad (5)$$

recibido
 u^k es el símbolo OFDM *recibido* en el FD y z^k es la DFT del vector de ruido. Utilizando las propiedades de la matriz F ortogonal; $F^{-1} = F^H$ y $F^H F = I$, donde I representa la matriz identidad de tamaño $N \times N$, entonces (5) puede ser formulada como:

$$\begin{aligned} u^k &= FH^k F^H F x^k + z^k \\ &= FH^k F^H s^k + z^k \end{aligned} \quad (6)$$

$$= G^k s^k + z^k \quad (7)$$

Matriz de Frecuencia del canal
Matriz de Frecuencia del canal
z₀
donde s^k se compone por el vector S_D^k ⊂ Ω de tamaño N_D, N_P símbolos pilotos y N_G símbolos de guarda. La CFM (Channel Frequency Matrix) G^k = FH^kF^H contiene información en dominio de la frecuencia del canal y la frecuencia Doppler de la representación circulante y dispersa de la CIR variante en el tiempo. G^k se convierte en una matriz diagonal cuando la propagación Doppler es insignificante, lo cual significa un sistema libre de ICI, sin embargo, los ambientes V2V presentan alta movilidad tanto en el transmisor y receptor, ocasionando una dispersión Doppler significativa. Lo anterior convierte a G^k en una matriz dispersa generando un sistema afectado por la ICI.

Debido a la selectividad del canal V2V, los sistemas OFDM son susceptibles a errores de detección, la potencia local de algunas subportadoras puede ser baja por causa de desvanecimientos profundos en la función de transferencia del canal imposibilitando la detección del dato transmitido en tales soportadoras. La técnica DFTS (Discrete Fourier Transform Spreading) se utiliza para reducir esta problemática.

Multiplicando la matriz de Fourier con el vector de símbolo OFDM transmitido es posible incorporar la dispersión DFT de los datos en el modelo de señal (7). Lo anterior, puede ser descrito con la siguiente ecuación:

$$\begin{aligned} u_D^k &= G_D^k F s_D^k + Z_D^k \\ &= G_D^k X_D^k + Z_D^k \end{aligned} \quad (8)$$

donde u_D^k es el vector de datos recibido, G_D^k la CFM de rango reducido $N_D \times N_D$, X_D^k el vector de datos transmitido con LP (Linear Precoding), Z_D^k el vector de ruido; cada término muestreado en las posiciones correspondientes de las subportadoras de datos.

2.5.2 Detector OSIC

Utilizando la descomposición V2V Sorted QR y el detector OSIC es posible implementar un detector subóptimo de baja complejidad, para un sistema multiportadora V2V con LP y que presente un excelente desempeño en términos de BER. La descomposición V2V Sorted QR de la matriz de canal G_D consiste en calcular una matriz triangular superior R , una matriz ortogonal de norma unitaria Q y una matriz de permutación P , de tal manera que $G_D P = Q R$, donde $G_D P$ es la matriz G_D con sus columnas ordenadas de acuerdo a P . Se obtiene un modelo reducido de la ecuación (8) :

$$u_D^k = Q R s_D^k + Z_D^k \quad (9)$$

$$Q^H u_D^k = R s_D^k + Q^H Z_D^k \quad (10)$$

$$\tilde{u} = R s_D^k + \tilde{z} \quad (11)$$

Debido a que Q es unitaria, las estadísticas del ruido no son alteradas. El nuevo modelo representado en la ecuación (11) es adecuado para la aplicación del detector

OSIC. La estructura triangular de la matriz R permite calcular el k -ésimo elemento de \tilde{u} como:

$$\tilde{u}_k = r_{kk}s_i + \sum_{i=k+1}^{N_D} r_{ki}s_i + \tilde{z} \quad (12)$$

2.5.3 Detector QR-ML convencional

La detección QR-ML convencional puede ser incorporado al modelo del sistema receptor anteriormente mostrado como:

$$\hat{S}_{ML} = \underset{s \in \Omega^{ND}}{\operatorname{argmin}} \| \tilde{y} - Rs \| \quad (13)$$

$$= \underset{s \in \Omega^{ND}}{\operatorname{argmin}} \left(\sum_{j=1}^{N_D} |\tilde{y}_j - \sum_{i=j}^{N_D} r_{j,i} - s_i|^2 \right) \quad (14)$$

La búsqueda del símbolo estimado \hat{S}_{ML} de la ecuación (14) se basa en la construcción de un árbol binario de búsqueda como se muestra en la Figura 7. Se define la métrica de distancia para cada rama del árbol.

$$d_i = | \tilde{y} - r_{i,i}\hat{S}_i - \sum_{j=i+1}^{N_D} r_{j,i} - \hat{S}_j |^2 \quad (15)$$

La distancia d_i representa el valor de rama de un nodo \hat{S}_i que tiene a los nodos $\hat{S}_{N_D}, \dots, \hat{S}_{i+1}$ como sus nodos antecesores. La distancia de un nodo que se encuentra en el k -ésimo nivel y el nodo de la raíz representa la distancia acumulada, esto es, la suma de todas las distancias de las ramas desde el nodo raíz hasta el nodo indicado. En un nivel n , la distancia acumulada se obtiene con la siguiente ecuación.

$$\sum_{i=1}^n d_{N_D} - i + 1 = \sum_{i=1}^{N_D} |\tilde{y}_{N_D} - i + 1 - \sum_{j=N_D-i+1}^{N_D} r_{N_D} - i + 1, \tilde{S}_j|^2 \quad (16)$$

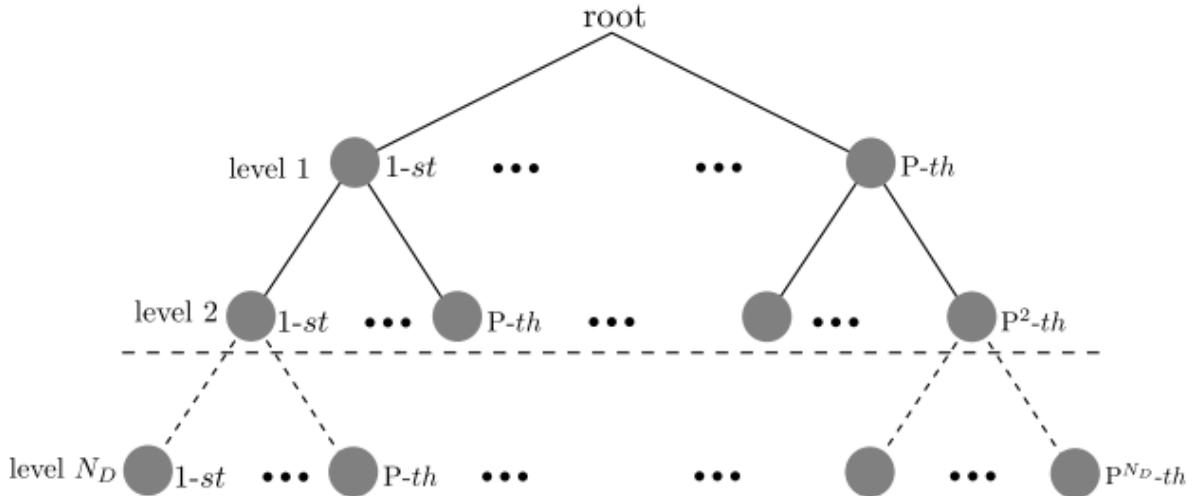


Figura 7. Estructura del árbol de búsqueda bajo el criterio ML.

Con base a la ecuación (14), la detección óptima del vector \hat{s} será la ruta que minimice a la ecuación (16), cuando $n = N_D$.

2.5.4 Detector V2V Near-ML

El detector V2V Near-ML incorpora el algoritmo M adaptativo a la detección QR-ML convencional, el cual realiza ajustes durante el proceso de búsqueda del árbol binario con el fin de reducir la complejidad computacional. Al igual que el detector OSIC hace uso de la descomposición Sorted-QR. Como se muestra en la Figura 8 el símbolo S_{N_D} se encuentra en el nodo raíz del árbol, y los nodos que surgen de el a través de las ramas son solución posible para $S_{N_D} \dots S_1$. El algoritmo M consiste en seleccionar en cada nivel del árbol binario de búsqueda un máximo de $M (M < P)$ candidatos para la detección del i -ésimo simbolo del vector \hat{s} estimado, descartando $P - M$ nodos restantes del nivel actual.

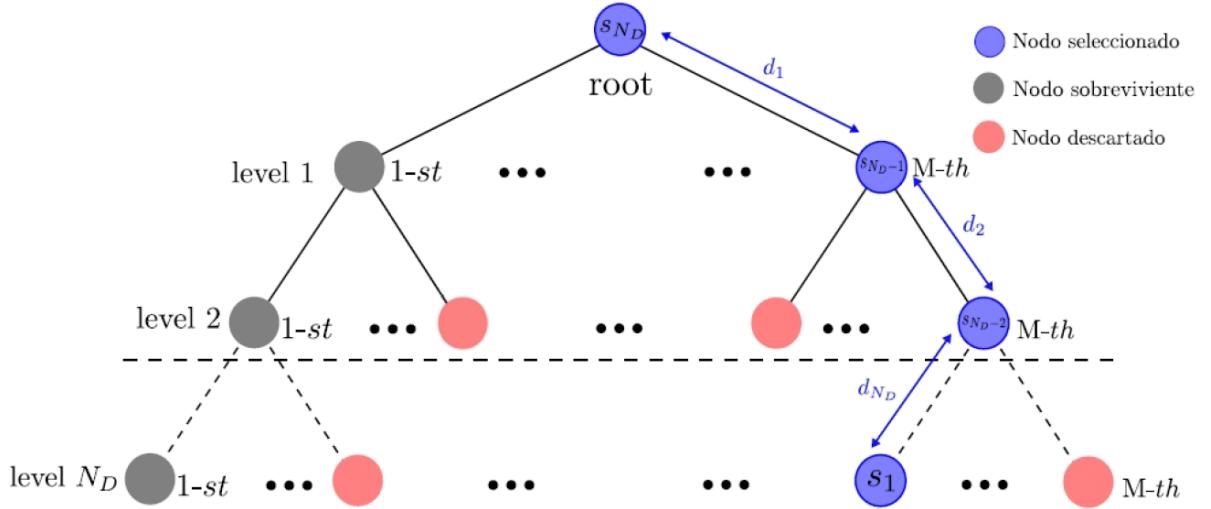


Figura 8. Estructura del árbol de búsqueda bajo el criterio ML con algoritmo M.

Cada rama del árbol de búsqueda contiene una métrica de distancia definida por:

$$D_k^2 = \| \tilde{y}_k - \sum_{i=k}^{N_D} R_{k,i} - X_j \|^2 \quad (17)$$

Se selecciona M nodos que mantienen una distancia menor entre cada nodo del k -esimo nivel y el nodo de la raíz, por lo tanto, al realizar la detección se obtiene M rutas posibles. La distancia correspondiente a cada ruta se define como :

$$D_T^2 = \sum_{k=1}^{N_D} D_k^2 \quad (18)$$

la solución \hat{s} esta dada por la ruta que cumpla con:

$$\hat{s} = \underset{s \in \Omega}{\operatorname{argmin}} \sum_{k=1}^{N_D} D_k^2 = \underset{s \in \Omega}{\operatorname{argmax}} D_T^2 \quad (19)$$

El algoritmo Near ML genera detectores de baja complejidad para valores de M pequeños, sin embargo, subóptimos en términos de tasa de error de bit, a medida que el valor de M incremente, el desempeño del algoritmo se acerca al del detector ML, con la penalidad del incremento en complejidad computacional.

Como se puede inferir del detector Near ML, una de las etapas importantes es el ordenamiento de un conjunto de distancias. Este proceso es tan relevante que el desempeño del detector y del receptor en general depende de que tan eficiente es el ordenamiento. A continuación, se muestran en las secciones 2.7 y 2.8 las técnicas actuales para el ordenamiento de datos.

2.7 Redes de ordenamiento

Las redes de ordenamiento representan una de las maneras más eficientes y tradicionales de ordenamiento en el contexto de los FPGAs. Son atractivas por dos razones principales: no requieren líneas de control y son sencillas de parallelizar debido a que tienen un patrón simple de flujo de datos. Las redes de ordenamiento son adecuadas cuando se tiene una secuencia relativamente pequeña de elementos cuya longitud se conoce con anterioridad.

La representación de este tipo de ordenamiento es la de Knuth (ver Figura 9 (a)), consiste en una serie de n líneas horizontales, donde n es la cantidad de elementos a ordenar y k líneas virtuales, donde k es la cantidad de comparadores. Estos últimos, los observamos en la Figura 9 (b).

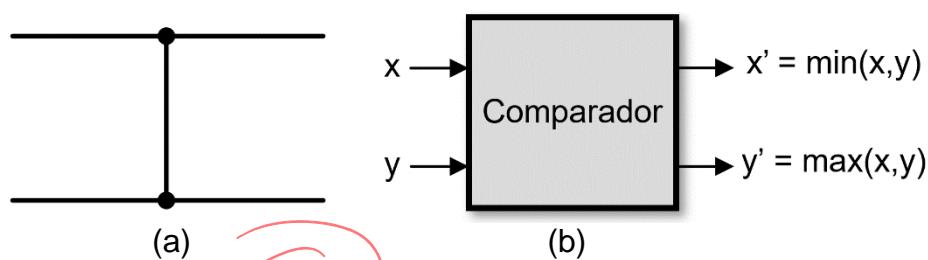


Figura 9. (a) Notación Knuth; (b) Comparador con entradas y salidas.
Notación de bloques

En la Figura 10 vemos un ejemplo de una red de ordenamiento de 4 entradas, entonces, existen 4 líneas horizontales en las cuales una cantidad de k comparadores están contactados.

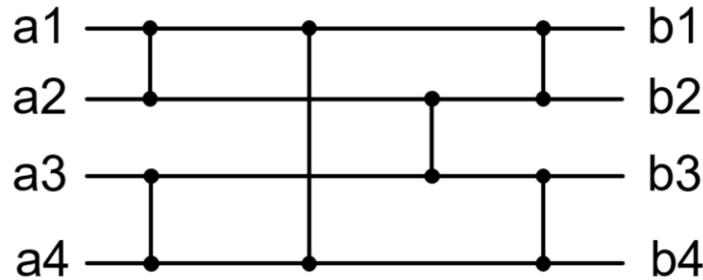


Figura 10. Ejemplo de red de ordenamiento.

En este punto, la red de ordenamiento está en una forma muy básica y sin la capacidad de ordenar, a continuación, veremos que los elementos medio limpiador (*half cleaner*), clasificador bitínico (*bitonic sorter*) y la red fusionadora (*merging network*) permiten el correcto funcionamiento de la red de ordenamiento.

2.7.1 Medio limpiador

El medio limpiador consiste en una red de comparación en donde la línea i es comparada con la línea $i + n/2$ para $i = 1, 2, \dots, n/2$. Las secuencias de entrada tienen la clasificación de bitónicas, estas tienen la forma $0^i, 1^j, 0^k$ ó $1^i, 0^j, 1^k$ para $i, j, k \geq 0$. Cuando una secuencia bitónica de 0's y 1's se aplica al medio limpiador obtenemos una secuencia de salida en donde los valores más pequeños están en la mitad superior y los más grandes en la mitad inferior. El comportamiento anterior, lo observamos en la Figura 11.

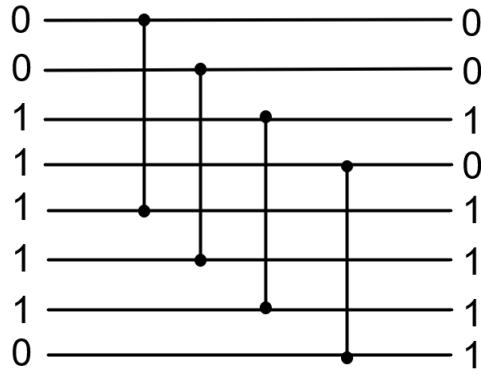
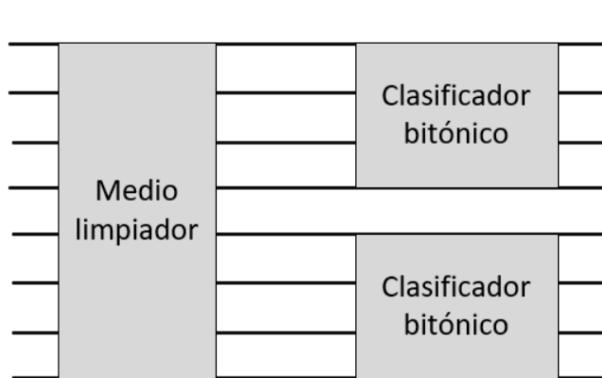


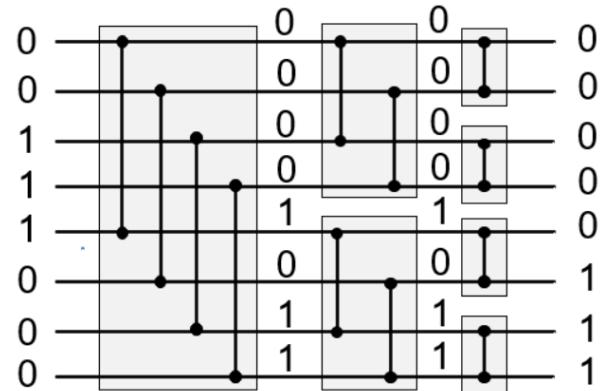
Figura 11. Medio limpiador.

2.7.2 Ordenador bitónico

El tercer elemento en la lista para construir una red de ordenamiento, es el ordenador bitónico, consiste en combinar medios limpiadores de manera recursiva. La Figura 12 (a) muestra la construcción recursiva, primero, el medio limpiador divide la secuencia de entrada en dos, posteriormente se construyen clasificadores bitónicos para ordenar las mitades restantes desde un enfoque recursivo. El desglose del proceso recursivo lo vemos en la Figura 12 (b).



(a)



(b)

Figura 12. (a) Construcción recursiva. (b) Recursión desglosada.

2.7.3 Red fusionadora

El último elemento para construir la red de ordenamiento es la red fusionadora, consiste en un circuito combinacional capaz de fusionar dos secuencias previamente ordenadas en una tercera que también está ordenada.

La red fusionadora es construida modificando el medio limpiador presente en el ordenador bitónico. La clase es realizar la inversión de la segunda mitad de las entradas implícitamente. Dada dos secuencias ordenadas $a_1, a_2, \dots, a_{n/2}$ y $a_{(\frac{n}{2})+1}, a_{(\frac{n}{2})+2}, \dots, a_n$ que van a ser fusionadas, se requiere tener el efecto de ordenar binómicamente la secuencia $a_1, a_2, \dots, a_{n/2}, a_n, a_{n-1}, \dots, a_{\frac{n}{2}+1}$. Debido a que el medio limpiador del ordenador binotico compara las entradas i y $\frac{n}{2} + i$ para $i = 1, 2, \dots, n/2$, la primera etapa de la red fusionadora se compara con las entradas i y $n - i + 1$. La Figura 13 muestra esta correspondencia.

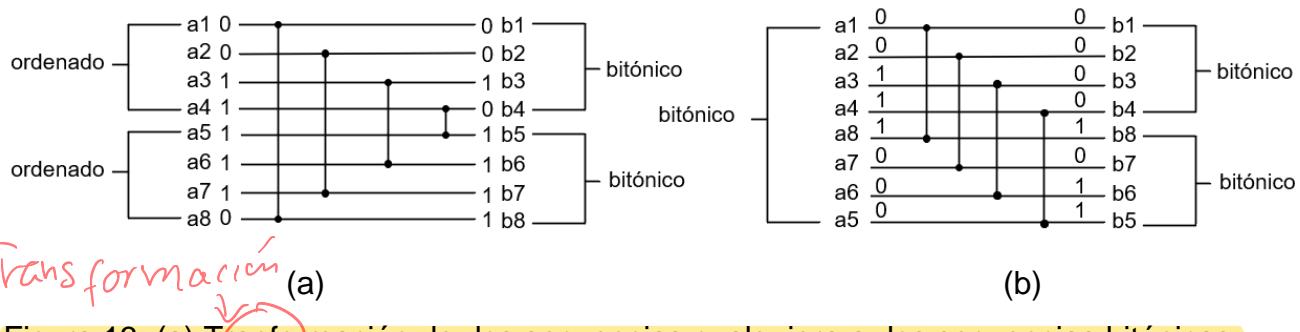


Figura 13. (a) Transformación de dos secuencias cualquiera a dos secuencias bitónicas.
 (b) Transformación de una secuencia bitónica a dos secuencias que también son bitónicas.

2.7.4 Construcción de una red de ordenamiento

En este punto tenemos todos los elementos para construir una red totalmente combinacional capaz de ordenar n elementos dados como entrada. La Figura 14 (a) representa la construcción de la red de ordenamiento, en esta figura observamos un nuevo elemento llamado ordenador[n], consiste en crear redes fusionadoras de manera recursiva hasta llegar a $n < 2$ (ver Figura 14 (b)). Finalmente, la Figura 14 (c) muestra la red actual desde el punto de vista de comparadores.

ella se observa
 consistente en

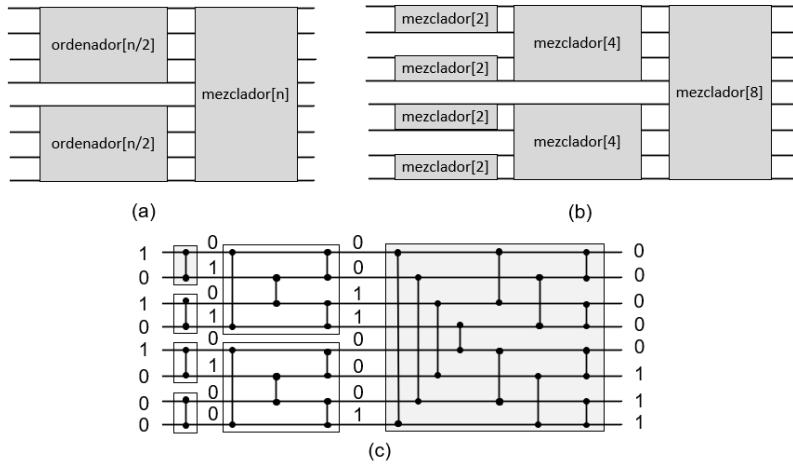


Figura 14. (a) Construcción recursiva. (b) Desglose de la recursión. (c) Desglose a nivel de comparadores.

2.8 Ordenamiento secuencial

El ordenamiento secuencial es el segundo enfoque para ordenar un conjunto de elementos desordenados. El término “secuencial” refiere a que el algoritmo necesita una cantidad específica de iteraciones para lograr el objetivo de ordenar n elementos. Es el más lento en términos de latencia debido a que cada iteración consume un ciclo de reloj, sin embargo, es el más fácil de implementar. A continuación, se presentan algunos ejemplos de unidades de ordenamiento basados en el enfoque secuencial.

2.8.1 Máquina de ordenamiento de inserción

La siguiente unidad de ordenamiento llamado máquina de ordenamiento de inserción está basada en el conocido algoritmo de inserción, en el cual se busca en un arreglo línea la correcta posición del dato no ordenado. La Figura 15 (a) representa la máquina de ordenamiento. Cada nodo es una célula de inserción/comparación y se tienen tantos nodos como elementos a ordenar. Los nodos entre si se comparan de tal manera que cada dato encuentra su posición correcta en el arreglo lineal de nodos.

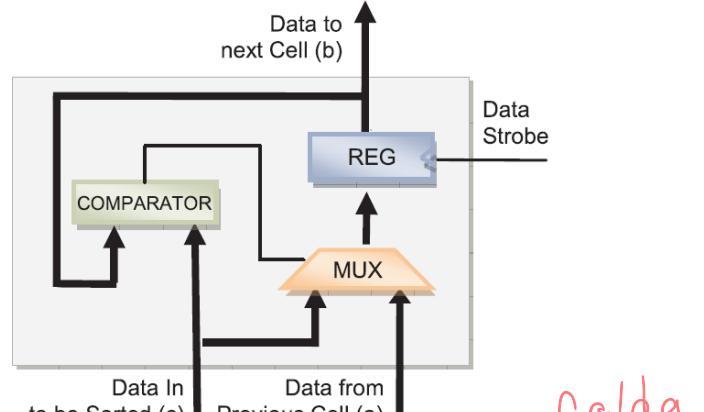
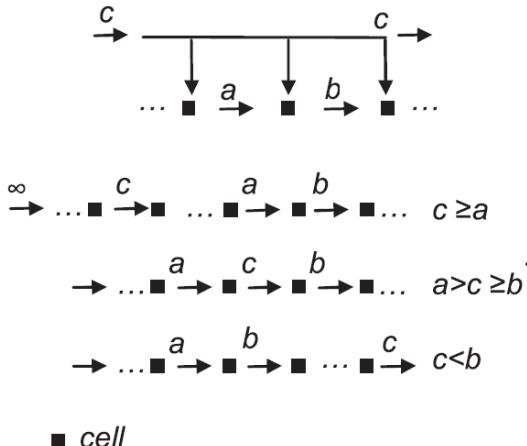


Figura 15. (a) Grafico de dependencias en modo ascendente, (b) Célula básica.

Los elementos que constituyen a la célula quedan plasmados en la Figura 15 (b), consiste en un comparador, multiplexor, registro de datos y algunas líneas de control. Si se conecta un conjunto de células entre sí se obtiene un arreglo línea y se agregan las líneas de control se obtiene la máquina de ordenamiento de inserción mostrada en la Figura 16.

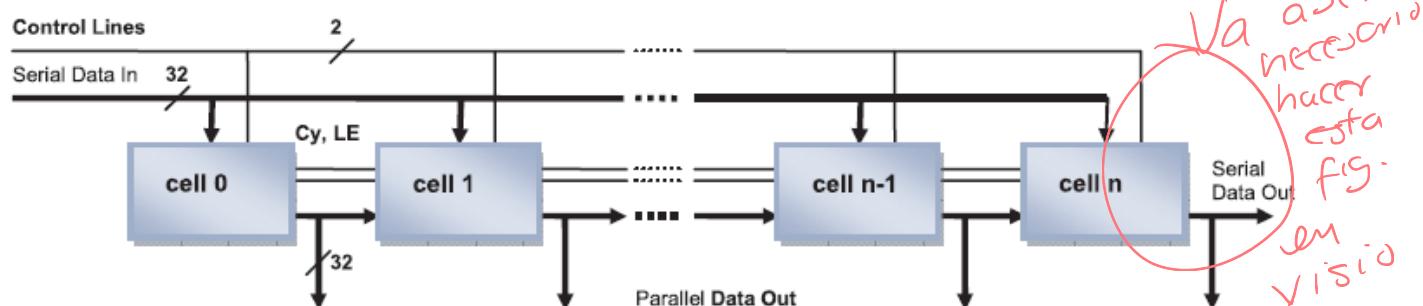


Figura 16. Diagrama de células para n - elementos.

2.8.2 Odd-even sorting network

Debido que la implementación de una red de ordenamiento implica una cantidad grande de recursos, el siguiente tipo de ordenamiento fue creado. La unidad de ordenamiento se reduce a solo una fila y la complejidad disminuye a $O(n)$. El nuevo enfoque utiliza la unidad básica del comparador vista en los anteriores enfoques de ordenamiento, sin embargo, agrega un multiplexor y registros en la salida como se

Este tipo de ordenamiento se creó debido que la implementación de una red implica una cantidad enorme de recursos.

a lineamiento
 (red de conmutación)
 muestra en la Figura 17 (a). Además, el nuevo enfoque ~~agregar~~ otro elemento, la ~~switch network~~ (Figura 17 (b)). Este último elemento tiene la responsabilidad de realizar un ~~alineamiento~~ de datos y llevarlos de nuevo hacia las unidades básicas de comparación. Cada alineamiento toma un ciclo de reloj, esto prosigue hasta lograr que todos los elementos estén ordenados, por lo que a este tipo de implementación es común verla como "red secuencial".

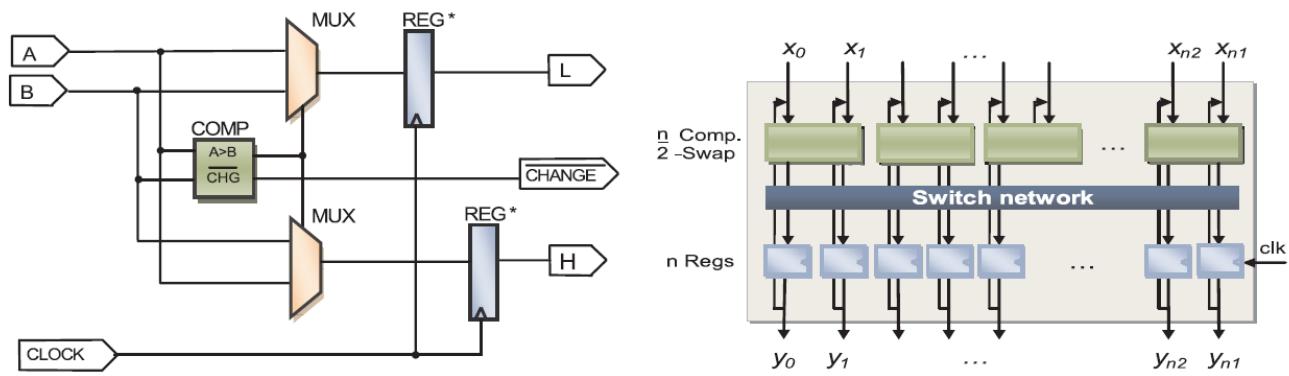


Figura 17. (a) Unidad de comparación. (b) Red de ordenamiento secuencial.

2.8.3 Ordenamiento basado en FIFOs

La Figura 18 representa el ordenamiento secuencial basado en tres colas: dos de ellas son colas de entrada y la tercera la cola de salida. Las primeras dos colas tienen una longitud de $n/2$ y la de salida n elementos. El nuevo enfoque se basa en el algoritmo *merge sort* donde dos conjuntos ordenados se fusionan para formar un ~~x~~ tercero que también está ordenado. El proceso de combinar las dos colas de entrada empieza comparando los dos primeros elementos que se encuentran en la cabeza de cada cola, el resultado de la comparación sirve de selector para un multiplexor, este elemento selecciona el dato apropiado para ser insertado en la cola de salida, por lo tanto, en cada ciclo de reloj un nuevo dato es ~~insertado~~ en la cola de salida. El proceso de inserción se sigue hasta obtener todos los datos ordenados.

~~inserto~~ *inserto*

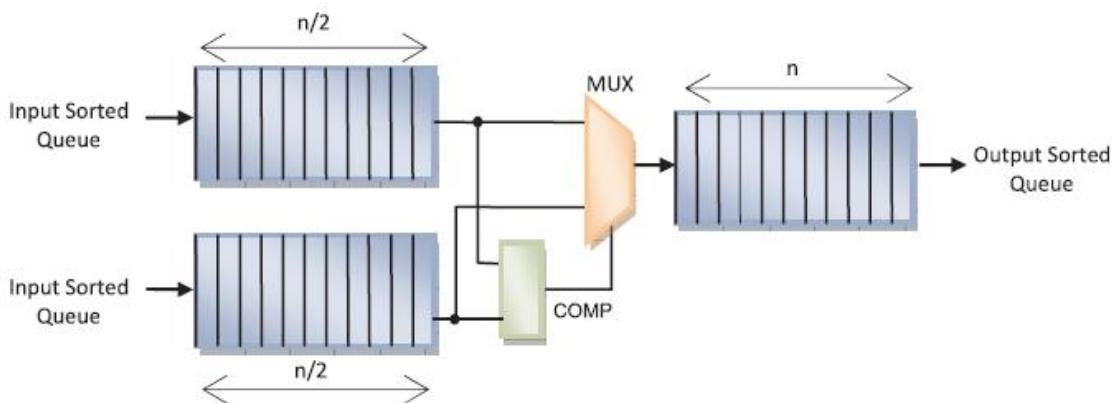


Figura 18. Diagrama a bloques del ordenamiento basado en FIFOs.

2.6 Diseño digital a nivel RTL en FPGA

~~se~~ ~~una metodología~~
En [40] proponen ~~un~~ ~~una metodología~~ que permite realizar la implementación de un algoritmo de forma eficiente en términos de velocidad o consumo de área y además, los diseños pueden ser replicados de manera sistemática. A continuación, se explica cada una de las etapas. Dicho método, llamado Mapo de algoritmo a arquitectura

2.6.1 Mapado de algoritmo a arquitectura

De manera general, el procedimiento se resume en el diagrama de flujo de la Figura

19. En las siguientes subsecciones se explican cada uno de los bloques correspondientes a cada etapa del modelo

Cabe resaltar, que tal diagrama tiene una naturaleza cíclica implícita en cada una de sus etapas.

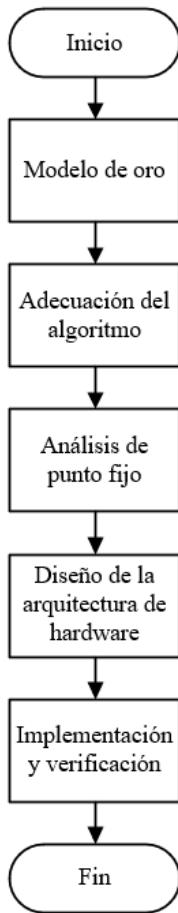


Figura 19. Algoritmo a arquitectura.

2.6.1

~~2.6.1~~ Modelo de oro

Se genera un modelo que sirva de referencia para el algoritmo a implementar, de tal manera que se pueda evaluar el **rendimiento de la implementación**. Para lograr esto, se programa el algoritmo bajo estudio empleando de preferencia un lenguaje de alto nivel, en aritmética de punto flotante y con sus ecuaciones originales.

2.6.2

~~2.6.2~~ Adecuaciones al algoritmo

En este paso ~~se modifica la secuencia de pasos o sus respectivas ecuaciones del~~ ^{funcional} algoritmo con el fin de reducir su complejidad sin alterar ~~la funcionalidad general~~. Es importante recalcar que no existe una metodología para aplicar ~~las~~ ^{tales} adecuaciones del algoritmo, solo el entendimiento profundo y experiencia sirve para ~~identificar las~~ ^{enumerar} posibles adecuaciones. A continuación, se mencionan algunas de ellas:

~~identificar las~~ ^{identificar las}

- Transformaciones o cambios de dominios.
- Sistolización de operaciones de cálculo intensivo.
- **Remplazo de divisiones** y multiplicaciones con operaciones como sumas y desplazamientos.
- Simplificación o propiedades matemáticas.
- Factorizaciones o aproximaciones matemáticas.

2.6.4 Análisis de punto fijo

Utilizando la representación de punto fijo es posible alcanzar desempeños cercanos a los obtenidos con la representación en punto flotante, por lo cual, esta última es comúnmente descartada en las arquitecturas de procesamiento de señales (PDS). Además, la representación en punto fijo reduce el tiempo de procesamiento y la complejidad de la implementación por ende el consumo de potencia y área de silicio.

El objetivo que se persigue al hacer un análisis de punto fijo es encontrar una longitud y formato de palabra que permita tener un desempeño similar al modelo de oro que está representado en punto flotante.

A continuación, se muestran los pasos a seguir para realizar el análisis de punto fijo:

- **Medición del rango dinámico:** El rango dinámico de una variable es la diferencia entre el valor máximo y mínimo que puede tomar durante la ejecución del algoritmo. Es importante obtener el rango dinámico para cada variable en punto flotante para después asignar una cantidad de fija de bits con su parte entera y fraccionaria.
- **Selección del formato de palabra:** La cantidad de bits para el parte entera (IP) de una variable se relacionada con su rango dinámico con la siguiente expresión:

$$n = \lceil \log_2(\max(|x_{\text{min}}|, |x_{\text{max}}|)) + 2 \rceil \quad (20)$$

donde α_{max} y α_{min} son los valores máximo y mínimo del rango dinámico de la variable en punto flotante. Para la parte fraccionaria se define la resolución ε de una variable en punto fijo, puede ser expresada como:

$$1 - \frac{1}{2^{FP}} \quad (21)$$

De (21) se despeja FP que representa la cantidad de bits de la parte fraccionaria. Para una resolución específica, se obtiene:

$$FP = \text{ceil} (\log_2 \frac{1}{\varepsilon}) \quad (22)$$

La resolución de la parte fraccionaria debe de ser escogida com base a las necesidades del sistema porque una mayor resolución generara un aumento en la anchura de la palabra (WL).

- Cuantificación de la relación señal a ruido de cuantización (SQNR): Cuando se realiza el proceso de cuantificación de un valor en punto flotante también se genera un error que puede ser representado como ruido agregado al valor original, por lo tanto, el SQNR es el parámetro para medir el nivel de calidad del proceso de cuantificación de una variable en su representación de punto flotante. La mayoría de los algoritmos operan con variables que representan matrices y vectores, por lo tanto, el SQNR se expresa como un promedio:



$$SQNR = 10 \log_{10} \frac{\sum_{n=0}^{N-1} a(n)^2}{\sum_{n=0}^{N-1} (b(n) - a(n))^2} \quad (23)$$

donde $a(n)$ es la variable original en punto flotante, $b(n)$ es la representación de $a(n)$ en punto fijo y N es la cantidad de elementos que conforman a la variable. Cuando una variable tiene un SQNR considerado bajo, se ajusta la longitud su

longitud de palabra y se sacrifican bits de aquellas palabras cuyo SQNR es considerado alto. El objetivo es conseguir un SQNR en la salida equilibrado con las variables que conforman el algoritmo.

- **Efectos del redondeo:** Una desventaja que se presenta al pasar un número en su presentación en punto fijo es que no todos los valores pueden ser representados de manera exacta. El método del redondeo es utilizado para obtener un valor representable a pesar de la perdida de precisión. Existe una gran variedad de tipos de redondeo, los más utilizados son el truncamiento, redondeo hacia cero, redondeo hacia más menos infinito y redondeo al más cercano. El tipo de redondeo a utilizar en el algoritmo tendrá un efecto importante en el desempeño en términos de SQNR, área y velocidad de procesamiento.

2.6.5 Diseño de la arquitectura de hardware

El objetivo en esta etapa es realizar un diagrama a bloques donde se visualice el recorrido completo que deben de seguir los datos desde la entrada hasta la salida del algoritmo, además, debe de permanecer con una esencia de abstracción de alto nivel.

Cada uno los de
Bloques de la arquitectura deberán de presentar un proceso funcional que termine en una transformación importante de datos. Después, el nivel de detalle es reducido a módulos más pequeños que en conjunto representan una jerarquía de bloques funcionales. La jerarquía se vuelva tan pequeña que resulta relativamente fácil diseñar una funcionalidad del bloque en cuestión. En este punto, se tiene una versión que es conocida como ruta de datos (*datapath*) de la arquitectura, la cual va acompañada con su correspondiente unidad de control.

De acuerdo con el criterio de optimización: velocidad o consumo de área; que se aplique a la arquitectura inicial, ésta se podrá mejorar a través de técnicas como: paralelismo, encauzamiento (*pipelining*), reconfigurabilidad, reusabilidad, etc.

Itálico

2.6.6 Implementación y verificación

Cuando finalmente se tiene la arquitectura de hardware a nivel de bloques funcionales, sigue describirla con un lenguaje de descripción de hardware e implementarla a nivel de lógica de transferencia de registros (RTL). El entorno de desarrollo que el fabricante del FPGA provee al arquitecto de hardware, juega un papel fundamental para facilitar e incrementar el proceso de diseño. Esto último, involucra estudiar una serie de comportamientos y condiciones que buscan cumplir con los siguientes objetivos:

- Verificar y asegurar que las funciones para las que fueron diseñadas sean realizadas con éxito. Lo anterior, se conoce como verificación estática.
- Verificar y asegurar que las secuencias en cada bloque funcional se ejecuten correctamente. Esto se le conoce como verificación por comportamiento dinámico, implica la generación de vectores de entradas variantes en el tiempo que se aplican a la arquitectura y así monitorear los cambios en las salidas.
- Comprobar el comportamiento temporal: Es posible crear pruebas para observar las variaciones que presenta las señales en puntos específicos.

2.9 Diseño HLS en FPGA

Actualmente, los **algoritmos son codificadores** en lenguajes de alto nivel como C y C++, lo que permite una abstracción de detalles de la plataforma de cómputo, además permiten una iteración rápida, mejoras incrementales y portabilidad del código, todas estas características son críticas en la industria del software. En las últimas décadas, la ejecución rápida de los algoritmos codificados en lenguajes de alto nivel ha sido lo que ha impulsado al desarrollo de **procesadores y copiladores**. Al principio, para mejorar el rendimiento de un programa en tiempo de ejecución se hacía uso de dos conceptos centrales: incrementar la frecuencia del reloj del procesador o usar procesadores dedicados. Por mucho tiempo, lo que se hacía era esperar un año para tener los nuevos procesadores y de esta manera acelerar la ejecución del software. Con cada incremento en la frecuencia de reloj se mejoraba el tiempo de ejecución del programa, sin embargo, para grandes aplicaciones y productos de **calidad no es viable**.

esta ya no es viable.

Por otro lado, los FPGAs tienen una gran diferencia con las aplicaciones basados en un procesador, principalmente el modelo de programación. En un FPGA se basa principalmente en la descripción de RTLs (Registers Transfer Level) sin utilizar los lenguajes de alto nivel como C o C++ anteriormente mencionados. La Figura 20 muestra en una línea de tiempo que representa el tiempo que tarda cada tipo de tecnología en alcanzar una primera versión de la aplicación y hasta lograr una versión final, como se observa, el modelo de programación basado en RTLs es el más lento.

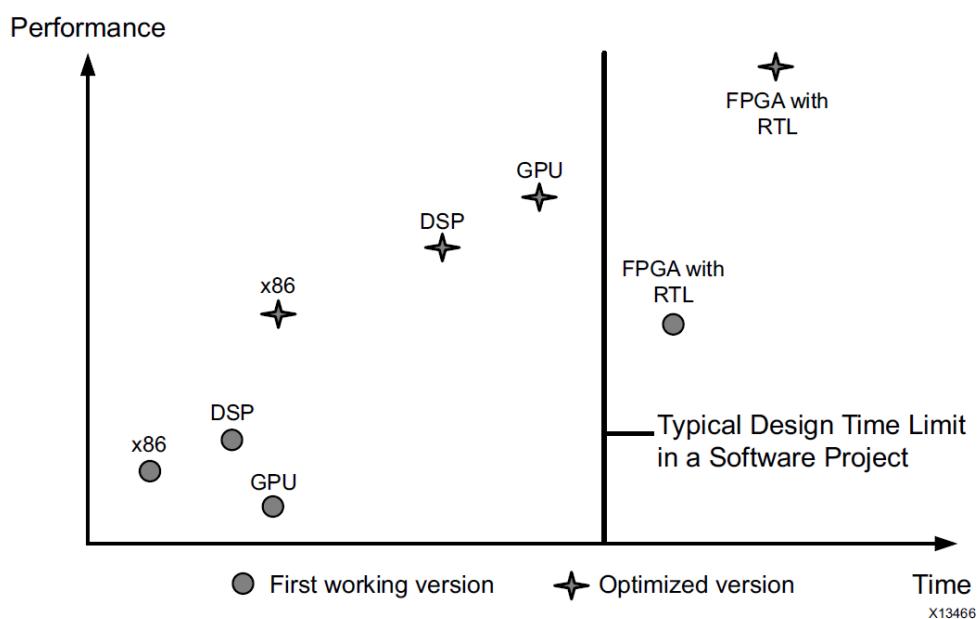


Figura 20. Tiempo de diseño vs rendimiento de la aplicación con diseño RTL.

La aplicación implementada en un FPGA presenta una mayor eficiencia en términos de consumo de potencia y tiempo de ejecución, sin embargo, el tiempo de desarrollo requerido para alcanzar la implementación esta más allá del alcance del esfuerzo típico de desarrollo de software. Por este motivo, normalmente los FPGAs son utilizados solo para aquellas aplicaciones que requieren un rendimiento que no puede ser logrado con la tecnología actual basada en procesadores.

Los recientes avances en la tecnología realizados por Xilinx® ~~rompen la diferencia entre los modelos de programación basados en un procesador y el del FPGA creando el copilador de~~

síntesis de alto nivel (*High Level Synthesis compiler*). El nuevo copilador permite convertir un algoritmo en un lenguaje de alto nivel en una implementación RTL, esto es, convierte un modelo de programación basado en procesador en un modelo de programación basado en niveles de trasferencia de registros. El proceso de conversión entre modelos permite reducir el tiempo de desarrollo de la aplicación (ver Figura 21) y conservar los beneficios de una implementación en hardware.

Chocar
también la otra

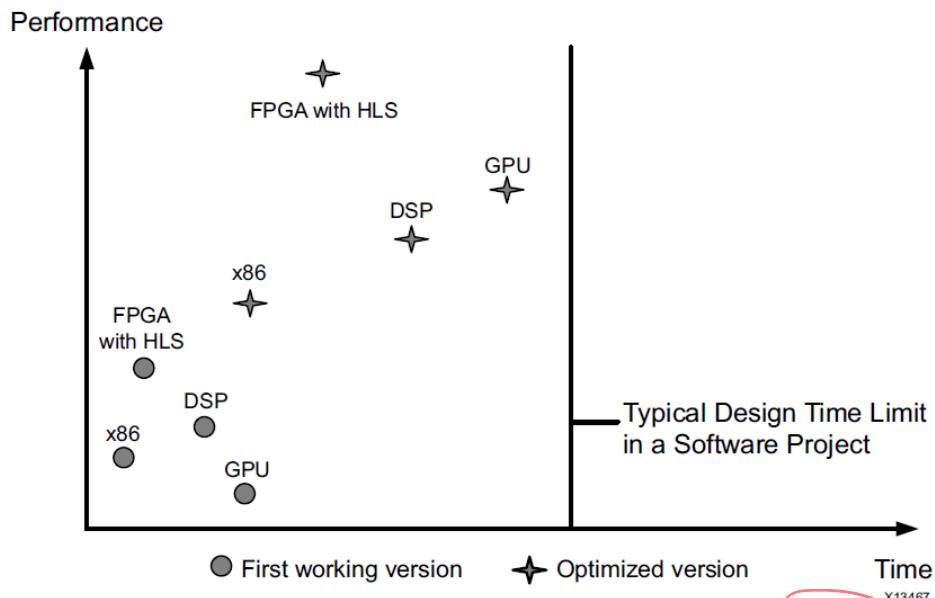


Figura 21. Tiempo de diseño vs rendimiento de aplicación con el copilador Vivado HLS.

CAPÍTULO III. MÉTODO

El objetivo de este capítulo es describir la metodología empleada para el desarrollo del presente trabajo, consta de los sujetos u objeto de estudio, el procedimiento que desglosa cada una de las etapas del proyecto y por último, la lista de materiales y herramientas que fueron necesarias.

3.1 Sujeto

El objeto de estudio es el detector Near-ML con esquema de transmisión SISO-OFDM en ambientes vehiculares. Este trabajo es desarrollado por un alumno de maestría y asesorado por un investigador del Dpto. de Ingeniería Eléctrica y Electrónica.

3.2 Procedimiento

La Figura 22 muestra el procedimiento a seguir para cumplir con los objetivos planteados en el presente proyecto. A continuación, se explica cada etapa con un mayor nivel de detalle.

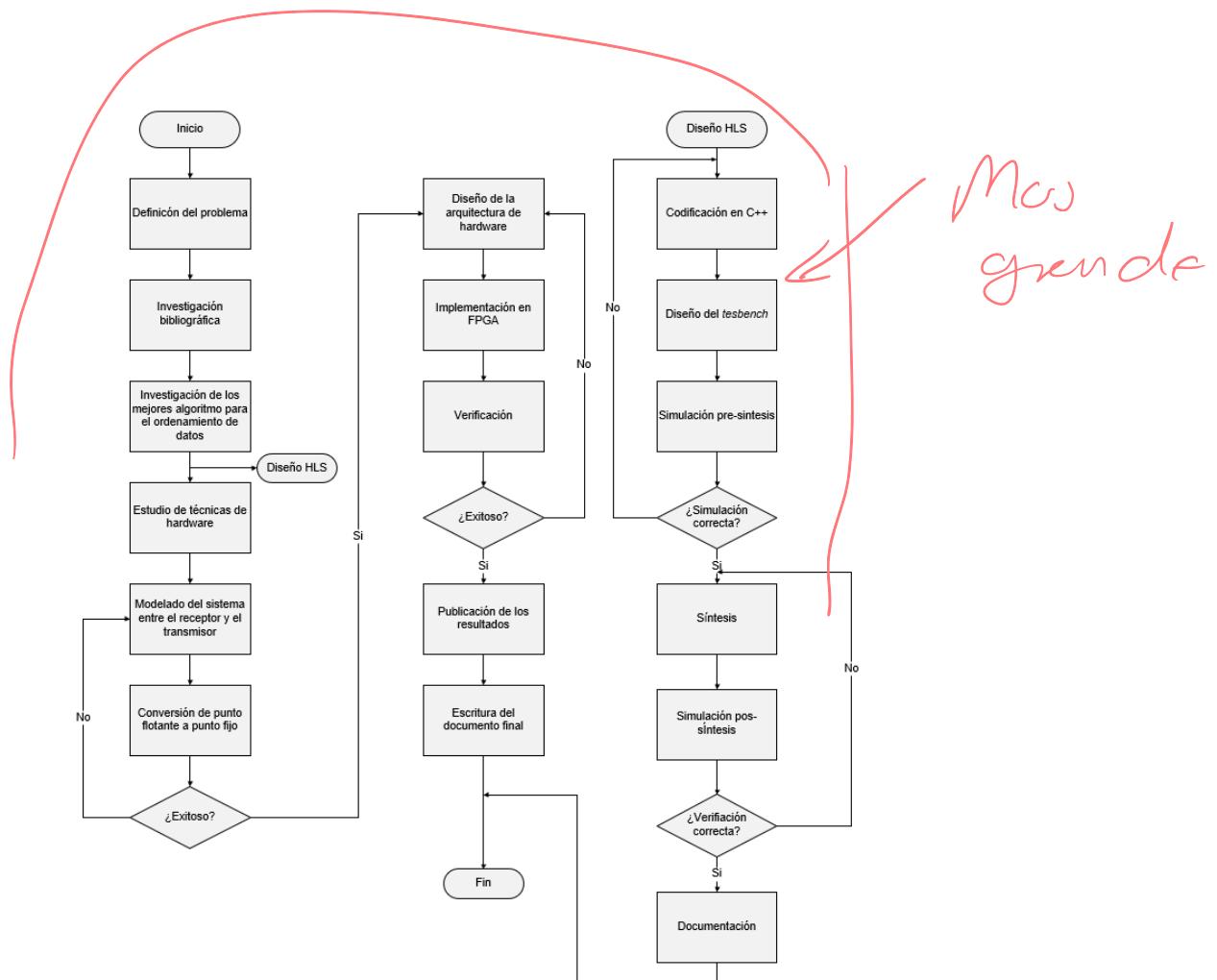


Figura 22. Diagrama de flujo del procedimiento.

Definición del problema: Se describen los antecedentes, objetivos, problema a resolver, hipótesis, alcances y limitaciones del proyecto.

Investigación bibliográfica: Se revisan documentos como artículos, libros y revistas cuya publicación sea reciente, con el objetivo de conocer el estado de arte de los detectores en sistemas SISO que operan en ambientes vehiculares.

Estudio de técnicas de diseño de hardware: Se estudian los métodos de diseño e implementación en hardware.

Modelado del sistema entre el receptor y transmisor: Se modela el sistema completo del transmisor y receptor en MATLAB con la finalidad de comprobar el

Mas grande

funcionamiento del detector.

Conversión de punto flotante a punto fijo: Despues de comprobar el funcionamiento del algoritmo en punto flotante, se convierte a aritmética de precisión finita para simular el comportamiento del algoritmo operando en un sistema digital real con recursos limitados.

Diseño de la arquitectura de hardware: La arquitectura propuesta del detector ML se codifica en un lenguaje de descripción de hardware, y se utiliza un simulador basado en software para comprobar el funcionamiento.

Implementación en FPGA: Una vez que la simulación de la arquitectura diseñada es exitosa se procede a implementarla en un FPGA disponible en el mercado.

Verificación: En la verificación se comprueba la frecuencia máxima de operación a la cual la implementación puede manejar, el hardware consumido en el FPGA. Si los resultados no cumplen con las expectativas se regresa a la etapa de diseño de la arquitectura.

→ Se corrobora a nivel funcional y estadístico la arquitectura de hardware.
Se ha finalizado

Publicación de resultados: Cuando la verificación está completa se procede a publicar los resultados en una revista o congreso.

Escritura del documento final: Se escribe el documento de tesis para cumplir con el requisito de titulación.

Codificación en C++: El algoritmo de ordenamiento se codifica utilizando el lenguaje de programación de alto nivel C++ como una función que recibe de parámetro un arreglo desordenado de tamaño N .

Diseño del testbench: Se genera un arreglo de N valores aleatorios en el rango de 0 a 65536, el cual son los valores posibles en decimal con una longitud de palabra de 16

Reescribe a nivel genérico "Se inyecta un conjunto de vectores de entrada

el diseño y - o o o //

bits.

Simulación pre-síntesis: En este paso se copia y se ejecuta el código en C++ para verificar el correcto funcionamiento del algoritmo de ordenamiento.

Síntesis: Se ejecuta la síntesis del algoritmo para obtener el reporte de síntesis, el cual ofrece una variedad de métricas de eficiencia de la arquitectura.

Simulación pos-síntesis: En esta etapa se ejecuta el tesbench en un ambiente de co-simulación con el objetivo de validar la arquitectura obtenida en la etapa de síntesis.

Documentación: Se documenta cada uno de los tipos de resultados dados por la herramienta Vivado HLS, tales resultados son: resultados de síntesis, rendimiento y pruebas funcionales.

consumo de recursos FPGA,
retardo de datapath, FREQ max,
consumo de bloques dedicados,
pruebas y resultados funcionales

3.3 Materiales y Herramientas

A continuación, se muestran los materiales y herramientas necesarios para cumplir con el desarrollo de la investigación.

Herramientas:

- MATLAB
- Vivado HLS 2016.1
- Laptop Acer con procesador Intel core i5-8265U, 8GB DDR4, 256GB PCIe NVMe SSD.

Materiales:

- Bases de datos institucionales de revistas internacionales.
- * Teses de posgrado afines.

CAPÍTULO IV. DESARROLLO

En este capítulo se describe la implementación del modelo algorítmico del detector Near ML, el diseño HLS de varios algoritmos de ordenamiento, los bloques que componen a la propuesta de la arquitectura y el diseño HDL de un ordenador híbrido (secuencial, paralelo). Cada bloque que conforma las arquitecturas se describe a detalle junto con sus entradas y salidas.

Sistema

4.1 Desarrollo del modelo algorítmico del sistema

El detector Near-ML que se definió de manera analítica en la sección 2.5.4 fue modelado utilizando la herramienta computacional MATLAB, posteriormente incorporado a un sistema completo de comunicaciones SISO-V2V en punto flotante. En las siguientes secciones, se muestra cada etapa de la implementación del detector.

QAM-4/16/64

4.1.1 Modelo de oro del detector

El modelo de oro que se implementó es el propuesto por [24]. La Figura 23 y Figura 24 muestran el modelo a nivel pseudocódigo, consiste en un algoritmo recursivo cuyas entradas son la matriz R de la descomposición QR, el vector \tilde{y} que contiene los datos que llegan al receptor, el vector $order$, el índice M que varía con base al tipo de modulación (QPSK, QAM-16, QAM-64), el tamaño de la constelación Ω y la cantidad de subportadoras de datos N_D de acuerdo con el esquema OFDM.

La siguientes notaciones se emplean dentro de las Figuras 23 y 24.

Las siguientes consideraciones son válidas en el pseudocódigo de la Figura 23 y la

Figura 24.

on denotan

- Variables con formato de negrita y en mayúscula representan matrices.
- Variables con formato de negrita y en minúscula representan vectores.
- Todas las variables son globales.
- El subíndice en las variables indica que se está refiriendo a un vector o una fila de una matriz.

Entrada: R , \tilde{y} , **order**, M , Ω , N_D .

Salida: Símbolo estimado \hat{s} .

detectorNearML(R , \tilde{y} , **order**, M , Ω , N_D)

1. $D_{min} = \infty$
2. $skip = true$
3. **level** = {}
4. **S** = {}
5. $l = N_D$
6. $d_l^2 = ||\tilde{y}_l - R_{ll}x_i||^2$ para todos los símbolos $x_i \in \Omega$
7. $dist_l, pos_l = d_l^2$, en orden ascendente
8. $S_l = \Omega(pos_l)$, en orden ascendente
9. $nearml(\tilde{y}, l - 1)$
10. $\hat{s}(\textbf{order}) = \textbf{sest}$ (símbolo estimado)
11. devolver \hat{s}

Pseudocódigo del detector

Figura 23. Detector Near-ML.

```

nearml( $\tilde{\mathbf{y}}, n$ )
1.   for ( $k = 1$  to  $M$ ) do
2.       if ( $\text{skip} = \text{true}$  and  $\mathbf{dist}_{n-1,k} > \mathbf{dist}_{n,1}$ ) then
3.           break
4.        $\mathbf{level}_{n+1} = k$ 
5.        $d_n^2 = ||\tilde{\mathbf{y}}_n - \mathbf{R}_{nn}\mathbf{x}_i - \sum_{i=n+1}^{N_D} \mathbf{R}_{ni}\mathbf{S}_n \mathbf{level}_n||^2$  para todo  $\mathbf{x}_i \in \Omega$ 
6.        $\mathbf{dist}_n = \mathbf{dist}_n + \mathbf{dist}_{n+1,\mathbf{level}_{n+1}}$ 
7.        $\mathbf{S}_n = \Omega(\mathbf{pos}_n)$ , en orden ascendente
8.       if ( $\text{skip} = \text{true}$  and  $\mathbf{dist}_{n,1} > D_{\min}$ ) then
9.           break
10.      if ( $n > 1$ ) then
11.          nearml ( $\tilde{\mathbf{y}}, n - 1$ )
13.      else
14.          if ( $\mathbf{dist}_{n,1} < D_{\min}$ ) then
15.               $D_{\min} = \mathbf{dist}_{n,1}$ 
16.               $\hat{\mathbf{s}} = \mathbf{S}_{n,1}$ 
17.              for ( $i = n + 1$  to  $N_D$ ) do
18.                   $\hat{\mathbf{s}} = \mathbf{S}_{i,\mathbf{level}(i)}$ 

```

Figura 24. Algoritmo Near ML.

Pseudocódigo del la función nearml()

Modelado del sistema SWO V2V en MATLAB

4.1.2 SISTEMA V2V

la ejecución

Con el fin de corroborar el funcionamiento del algoritmo mostrado en la sección anterior, fue codificado en MATLAB y posteriormente incorporado a un sistema de comunicaciones de una antena transmisora y una antena receptora, simulando su operación en ambientes vehiculares reales. La Figura 25 muestra el sistema V2V, cada bloque es seleccionado con base a las especificaciones del estándar IEEE V2V. A continuación, se describe a grandes rasgos la funcionalidad de cada bloque del sistema de comunicaciones V2V.

con modelos de canal de

uno de sus bloques

LLEVAR TODO
ESTO E INTEGRARLO
EN APARTADO 2.4 Y

AQUÍ SE PUEDE HACER
REFERENCIA A DICHO MODELO

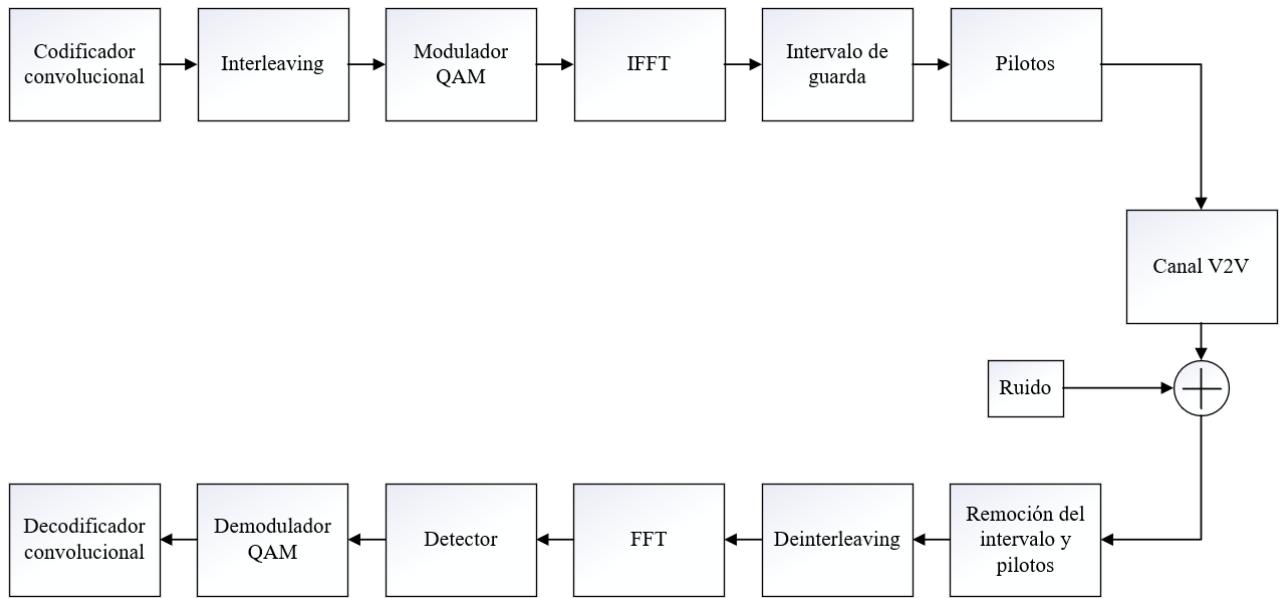


Figura 25. Sistema SISO V2V.

Codificador convolucional: Genera códigos lineales que se utilizan para proteger la información añadiendo redundancia a la secuencia de datos a transmitir. Con esto, se mejora el desempeño de la detección y corrección de errores de bits en el receptor.

Interleaving: Decorrelaciona la secuencia de bits generadas por el codificador convolucional con el fin de disminuir los errores que se puedan producir con bits por ráfaga.

Modulador QAM: Selecciona bloques de m bits para realizar un mapeo a un símbolo complejo $s \in \Omega$, por cada mT_s segundos, donde Ω es la constelación de acuerdo con el estándar 802.11p.

IFFT: El bloque IFFT se encarga de realizar la modulación OFDM por cada bloque de 64 muestras correspondientes a un símbolo.

Intervalo de guarda: Se agregan un conjunto finito de ceros a la izquierda y derecha de los datos en un símbolo OFDM, esto permite disminuir la interferencia Inter-

simbólica de las subportadoras adyacentes.

Inserción de pilotos: Se insertan un conjunto de elementos conocidos en cada símbolo OFDM cuya potencia suele ser mayor a las portadoras de datos, lo cual permite estimar el canal para posteriormente realizar la etapa de detección.

Canal V2V: Se generan matrices especiales que simulan los fenómenos físicos que se agregan a la señal de interés al pasar por el canal.

Ruido: Se genera un modelo de ruido básico conocido como ruido blanco gaussiano aditivo que representa el ruido en la electrónica del receptor.

Remoción del intervalo de guarda y pilotos: Remueve los intervalos de guarda y los pilos para solo dejar las subportadoras de datos.

Deinterleaving: Realiza el desentrelazado de la trama.

FFT: Realiza la demodulación OFDM.

Detector: Realiza la detección de los símbolos transmitidos utilizando el algoritmo Near-ML.

Demodulador QAM: Realiza la demodulación de los símbolos QAM provenientes del detector.

Decodificador Viterbi: Realiza la corrección de errores de bits de los datos detectados.

4.1.3 Análisis de punto fijo

Con el fin de obtener el rango dinámico de las variables que componen al algoritmo codificado en MATLAB, se ejecutó 1000 veces el script en punto flotante. En cada

Se llevó a cabo un análisis Montecarlo. Para esto se ejecutaron 1000 realizaciones del sistema. En cada realización se monitorearon

las variables críticas de algoritmo. La figura 26 muestra el resultado del experimento mencionado

iteración del programa los valores máximos y mínimos de las variables son guardadas, esto permite elegir al final de la ejecución del programa aquellos valores que representan el máximo y mínimo durante la vida del algoritmo. Estos últimos valores determinan la longitud de los buses. La Figura 26 muestra el resultado del procedimiento anteriormente descrito.

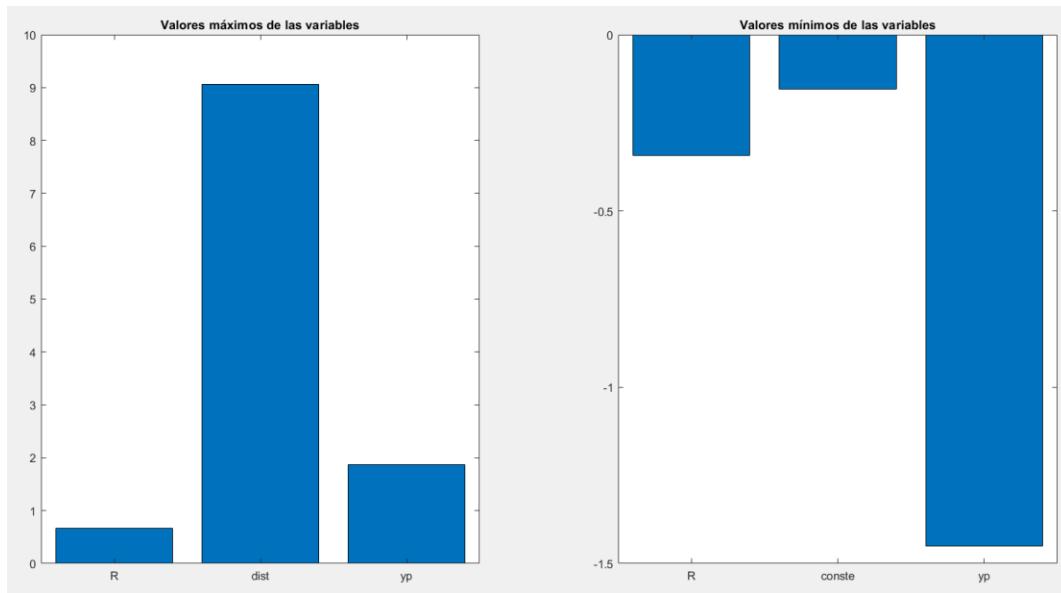


Figura 26. Valores máximos y mínimos. *de las variables críticas*

Una vez obtenido el valor máximo $\alpha_{max} = 9.06$ y el mínimo $\alpha_{min} = -1.08$ fueron sustituidos en la ecuación ~~+Floor(Numero que se encuentra en el origen de la referencia)~~ con el fin de obtener los bits de la parte entera que conformará el formato de la palabra.

mostrada
abajo

$$IP = \text{Floor}(\log_2(\max(|\alpha_{max}|, |\alpha_{min}|))) + 2$$

$$IP = \text{Floor}(\log_2(\max(9.06, 1.08))) + 2$$

$$IP = \text{Floor}(\log_2(9.06)) + 2$$

operador
floor

$$IP = 5$$

Para

Se obtiene que para poder representar en punto fijo el valor máximo es necesario por lo menos una parte entera de 6 bits. Para obtener la cantidad de bits de la parte fraccionaria se usó la ecuación $F = \frac{1}{10^{(-60/20)dB}}$ (No se encuentra el origen de la referencia.)

con una resolución de -60dB, → Se siguió el procedimiento que se muestra a continuación:

$$FP = cell\left(\frac{1}{10^{(-60/20)dB}}\right)$$

operación
técnica

$$FP = cell\left(\frac{1}{0.001}\right)$$

FP = cell(999)

$$FP = 10$$

De acuerdo con las operaciones anteriores, el formato de palabra resulta de la siguiente forma $WL = H + I + S + 1 = 15$ bits, el cual se extiende a $WL = 16$ porque es un formato estándar y permite tener una compatibilidad con otros módulos de la arquitectura.

májor con italicas
que qutu canilu

Para pasar de punto flotante a punto fijo se usa la herramienta "fixed-point toolbox" de MATLAB que permite convertir las variables en punto flotante a objetos que tienen propiedades de punto fijo y de esta manera comprar el funcionamiento de un algoritmo con longitudes de buses limitados. La Figura 27 muestra las líneas de código que definen el formato de la palabra, la capacidad de almacenar valores negativos y la creación del objeto `dataFormat` y `data`. Por otro lado, la palabra reservada `numerictype` crea un objeto con las longitudes del formato de palabra dadas como parámetros de la función y `dataFormat` y `data` contiene las relaciones entre variables de punto fijo.

```

signed = 1;
wordLength = 16;
integerPart = 5;
fractionalPart = wordLength - integerPart;

dataFormat = numerictype('Signed', signed, 'WordLength', wordLength, ...
    'FractionLength', fractionalPart...
);

dataFormat2 = fimath('RoundMode', 'Nearest', 'OverflowMode', 'Wrap', ...
    'ProductMode', 'TightPrecision', ...
    'ProductWordLength', wordLength, ...
    'ProductFractionLength', fractionalPart, ...
    'SumMode', 'SpecifyPrecision', ...
    'SumWordLength', wordLength, ...
    'SumFractionLength', fractionalPart...
);

```

Figura 27. Porción de código en MATLAB.

4.2 Diseño HLS de los ordenadores *de datos*

el

En esta sección se describe las etapas que se siguieron para realizar la parte de diseño HLS del ordenador. En general, consiste en tres pasos principales: selección de algoritmos de ordenamiento de acuerdo a un conjunto de criterios, codificación y su correspondiente *testbench*.

sistema de datos *pasos*

4.2.1 Selección de algoritmos

Actualmente existen una gran variedad de algoritmos de ordenamiento, cada uno de ellos tiene características bien definidas y áreas de aplicación. Para la selección de los algoritmos se tomó como base la métrica más representativa que presentan los algoritmos: el orden de complejidad. Los algoritmos que se elegieron fueron:

su selección

- Selection Sort (ss) *elegidos*
- Insertion Sort (IS)
- Bubble Sort (BS)
- Heap Sort (HS)

de cada uno de ellos.

En la Tabla 2 se muestran las ventajas y desventajas de los algoritmos seleccionados para ser sintetizados. Una de las características más presentativas de los algoritmos que se puede identificar en la tabla, es la facilidad de implementación y la eficiencia con arreglos pequeños y grandes.

disminuir Lento letre

Tabla 2. Ventajas y desventajas de algoritmos de ordenamiento seleccionados.

Algoritmo	Ventajas	Desventajas
Insertion Sort	<ul style="list-style-type: none">• Simple• Fácil de usar• Eficiente en listas pequeñas	<ul style="list-style-type: none">• Deficiente en grandes listas• Realiza numerosas comparaciones
Selection Sort	<ul style="list-style-type: none">• Rendimiento constante• Realiza pocos intercambios <p>con tar</p>	<ul style="list-style-type: none">• Su rendimiento es fácilmente influenciado por el orden inicial de los elementos• Realiza numerosas comparaciones
Bubble Sort	<ul style="list-style-type: none">• Fácil implementación• Fácil de comprender <p>con tar</p>	<ul style="list-style-type: none">• Deficiente en grandes listas• Requiere muchas lecturas y escrituras en memoria
Heap Sort	<ul style="list-style-type: none">• Es efectivo con datos desordenados <p>X</p>	<ul style="list-style-type: none">• Complejo• No es estable

4.2.2 Codificación

La codificación se realizó en el lenguaje de alto nivel C++. El programa de cada uno de los algoritmos de ordenamiento consiste en dos funciones principales. La primera función sirve de interfaz, en la cual se presentan variables de entrada como la operación (lectura o escritura) que se desea realizar, el dato de salida e índice de su posición en el arreglo lineal. La segunda función representa el algoritmo de ordenamiento. A continuación, se muestra a nivel de función como se compone la implementación de cada algoritmo .

letra lptu
menor
y negritas

```
void interfaz(operación, índice de dato de salida, dato de salida)  
{  
    //lógica del tipo de operación  
    Algoritmo();  
}
```

CENTRAR

La Figura 28 muestra como ejemplo la implementación de la interfaz del algoritmo de selección ~~donde~~ se puede ver los parámetros de entrada anteriormente descritos. Una condición if para escoger el tipo de operación. Si la variable `operation` es igual a la macro `SORT`, entonces, se realiza la operación de ordenamiento, de lo contrario, se accede a un elemento del arreglo de acuerdo con el índice dado en la variable `indexDout`.

```
void selectionSort(char indexDout, uint1 operation, data_out *dOut)  
{  
    #pragma HLS INTERFACE ap_none port = indexDout  
    #pragma HLS INTERFACE ap_none port = operation  
    #pragma HLS INTERFACE ap_none port = dOut  
  
    if(operation == SORT)  
    {  
        selectionAlgorithm();  
        *dOut = 0;  
    }  
    else  
    {  
        *dOut = A[indexDout];  
    }  
}
```

(a) Prototipo de función de interfaz y función del algoritmo

```
void selectionAlgorithm(void)  
{  
    for (short i = 0; i < N - 1; i++)  
    {  
        data_inp min = A[i];  
        data_inp index_min = i;  
        for (short j = i + 1; j < N; j++)  
        {  
            if (A[j] < min)  
            {  
                index_min = j;  
                min = A[j];  
            }  
        }  
        //Swap  
        data_inp temp = A[i];  
        A[i] = A[index_min];  
        A[index_min] = temp;  
    }  
}
```

Ponir Descripción
código del
SS

Figura 28. Implementación de la interfaz del algoritmo selection sort. (b)

Implementación de la lógica del algoritmo.

4.2.4 Programación del *testbench* en alto nivel

Con el fin de comprar el funcionamiento correcto de cada algoritmo de ordenamiento, una función externa fue creada, esta función comprueba que todos los elementos del arreglo estén ordenados. La Figura 29 muestra dicha función, consiste en iterar por cada elemento comparándolo con el elemento anterior, si existe un elemento en la posición i cuyo elemento adyacente $i - 1$ sea menor, entonces, quiere decir que hay un par de elementos que no están ordenados y por lo tanto el algoritmo no ha cumplido

se
desenv

con su objetivo.

```
uint8_t testSortAlgorithm(void)
{
    for(int i = 1; i < N;i++)
    {
        if(A[i] < A[i-1])
        {
            return 1;
        }
    }
    return 0;
}
```

dominir
Figura

Figura 29. Función para probar los algoritmos de ordenamiento.

corroborar funcionalidad de

4.3 Propuesta de arquitectura

En esta sección primero se empieza explicando la parte de la propuesta de la arquitectura y posteriormente en la sección 4.3.4 se presenta cada etapa del diseño HDL correspondiente al ordenador híbrido, que está basado en el algoritmo merge y redes de ordenamiento.

4.3.1 Bloque top

El nivel principal, también llamado *top level*, contiene una serie de bloques funcionales que en conjunto hacen posible la detección de símbolos. La Figura 30 muestra el diagrama a bloques de la propuesta de arquitectura y la Tabla 3 sus entradas y salidas.

A continuación,
se explica cada uno de los bloques.

los

MIPS

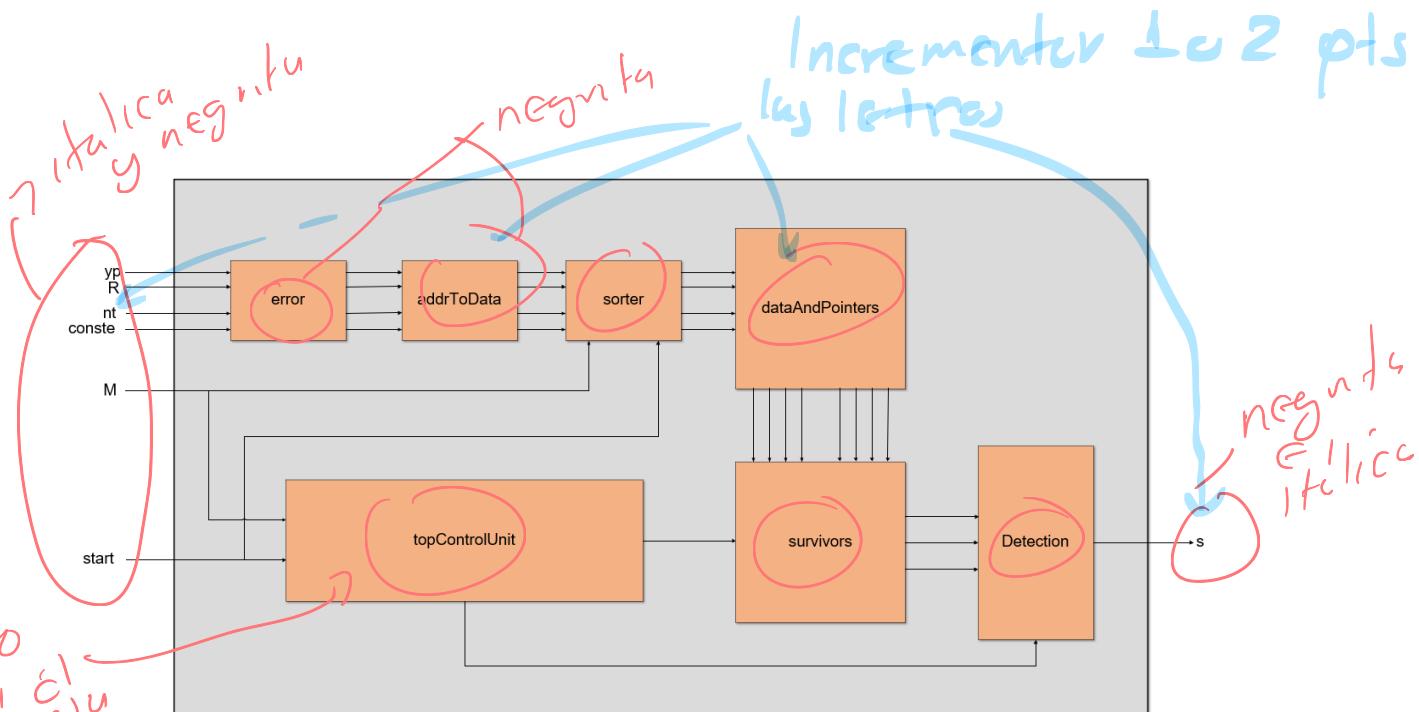


Figura 30. Propuesta de arquitectura. **RTL del detector**

Bloque Error

Genera la diferencia entre cada símbolo recibido y el símbolo real de la constelación.

Bloque addr To Data

~~addrToData~~: Accede a las direcciones de memoria que son dadas como entrada y las entrega como salida.

sorter: Realiza el ordenamiento para un conjunto de distancias cuyo número corresponde a las modulaciones QPSK, QAM16, QAM64 y QAM256.

dataAndPointers: Genera las direcciones de memoria de los datos que son dados como entrada.

survivors: Seleccionada a los nodos sobrevivientes en un árbol de búsqueda.

detection: Realiza la detección y genera un símbolo estimado.

ESTO SE QUEDA SUSTITUIR
POR LA EXPLICACION GENERAL
DE COMO FUNCIONIA LA ARQUITECTURA

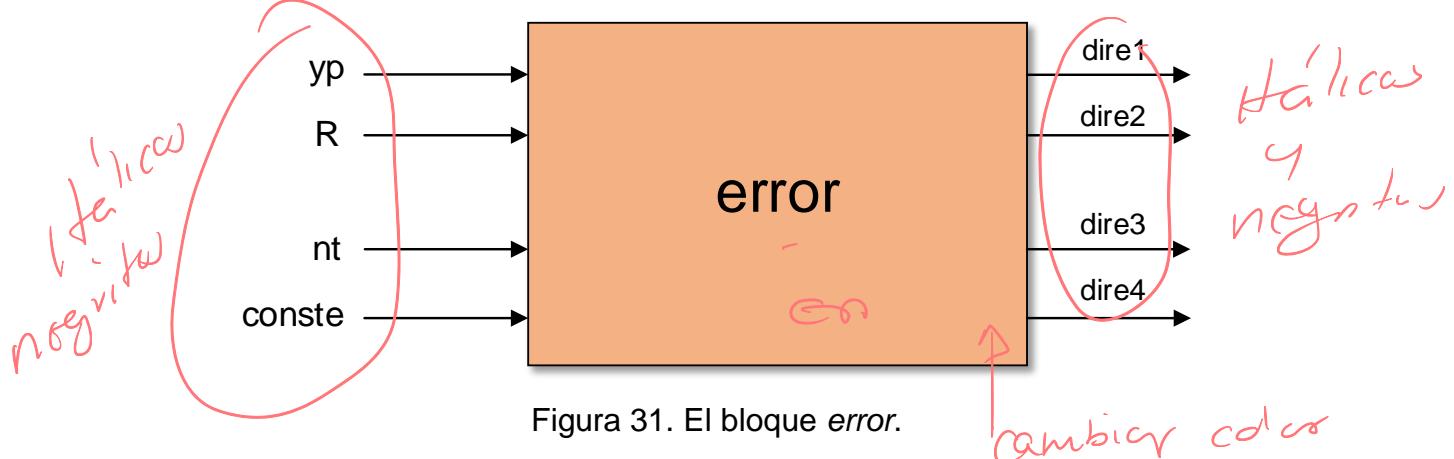
Tabla 3. Descripción de entradas y salidas del *top level*.

Entrada S	Longitud en bits	Descripción
yp	16	Vector recibido
R	16	Matriz R
nt	16	Tamaño de la matriz R
conste	16	Tipo de constelación utilizada
start	1	Señal que indica el inicio de la ejecución del módulo top
M	2	Índice de la modulación a utilizar
Salida S	Longitud en bits	Descripción
s	16	Símbolo estimado

4.3.2 El bloque error

El bloque *error* de la Figura 31 tiene el objetivo de generar el error entre los símbolos recibidos en el receptor y los verdaderos símbolos de la constelación, para este proceso se necesita el vector recibido, la matriz de canal *R*, el tamaño de la matriz *R* y la constelación que utilizada. Las entradas y salidas las observamos en la Tabla 4.

Se muestran



reduir 1 punto
letra

Tabla 4. Descripción de entradas y salidas del bloque *error*.

Entrada <i>S</i>	Longitud en bits	Descripción
yp	16	Vector recibido
R	16	Matriz R
nt	16	Tamaño de la matriz R
conste	16	Tipo de constelación utilizada
Salida <i>S</i>	Longitud en bits	Descripción
dire1,dire2,dire3,dire4	16	Direcciones de los errores

4.3.3 El bloque *addrToData*

El bloque *addrToData* de la Figura 32 tiene la función de acceder a los datos que se encuentran almacenados en las direcciones de memoria provenientes del bloque *error* y entregárselos al bloque *sorter* para ser ordenadas. Las entradas y salidas se muestran en la Tabla 5.



Figura 32. El bloque *addrToData*.

letra
-1 e 10

Tabla 5. Descripción de entradas y salidas del bloque *addrToData*.

Entrada	Longitud en bits	Descripción
dire1, dire2, dire3, dire4	16	Direcciones de los errores
Salida	Longitud en bits	Descripción
d1_un, d2_un, d3_un, d4_un	16	Distancias desordenadas

4.3.4 El bloque *sorter*

La Figura 33 presenta el diagrama a bloques de la arquitectura del bloque *sorter* y la Tabla 6 define sus entradas y salidas. De la Figura 33 se pueden identificar los siguientes elementos: *subsistemas*:

Sorting network: Consiste en una red de ordenamiento combinacional de 4 entradas basada en comparadores.

Merge unit: Realiza la combinación de conjuntos de 4 elementos para poder formar 16, 64 y 256 distancias ordenadas.

Mux: Realiza la multiplexación de las salidas de los bloques Merge unit y Sorting network.

Control unit: Controla la secuenciación necesaria para lograr que el bloque Merge unit combine subconjuntos de 4 entradas e indica cuando el proceso de ordenamiento ha finalizado.

En ruta

conformar

Supervisa y dirige

Finaliza

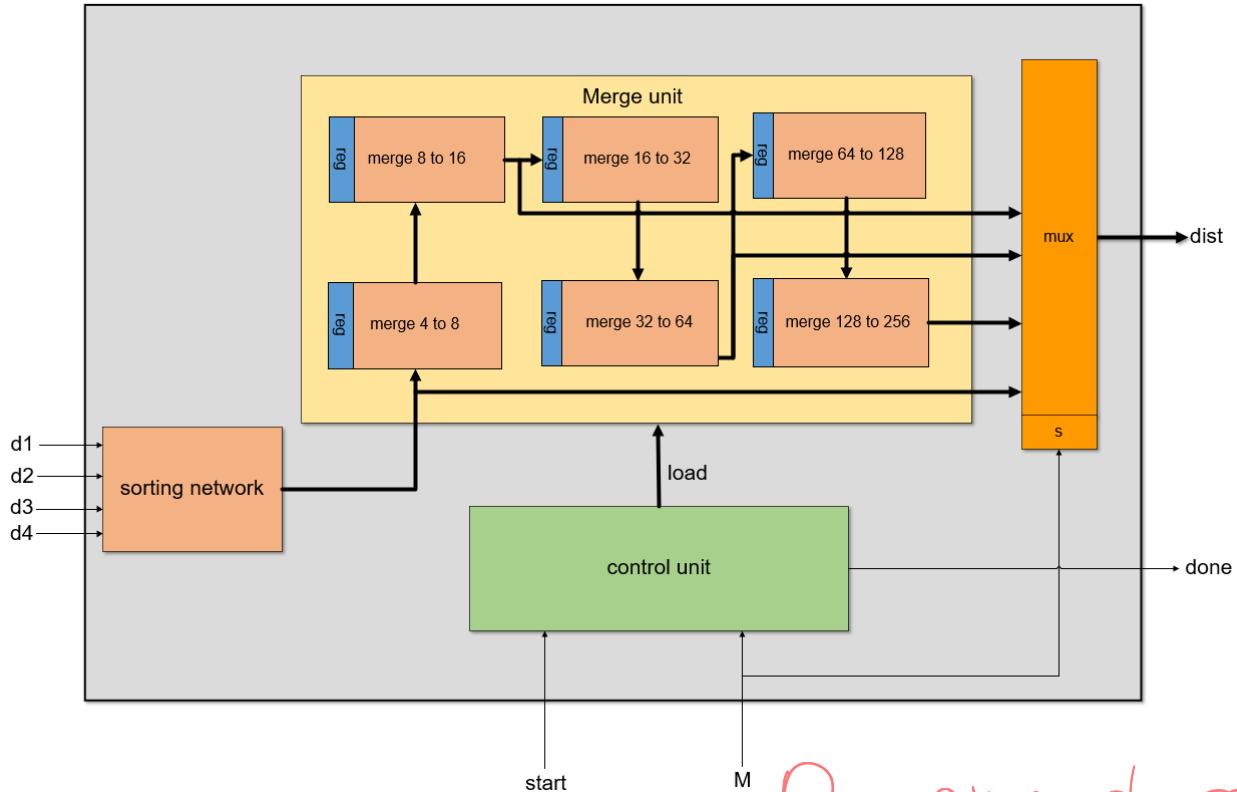


Figura 33. Bloque top.

Diagrama de caja blanca del bloque Top.

Tabla 6. Descripción de entradas y salidas del bloque top.

Entrada <i>S</i>	Longitud en bits	Descripción
d1, d2, d3, d4	16	Distancias generadas en el algoritmo Near-ML para posteriormente ser ordenadas
start	1	Señal que indica el inicio de la ejecución del módulo top
M	2	Índice de la modulación a utilizar
Salida <i>S</i>	Longitud en bits	Descripción
dist	64	Representa un vector con las distancias ordenadas
done	1	Indica que el ordenamiento ha finalizado.

- 1 pto letra

El inicio de la ejecución de bloque sorter comienza cuando la señal de start se vuelve un "1 lógico", entonces en cada ciclo de reloj se aceptan cuatro distancias desordenadas en las entradas d1 a d4 para formar la cantidad de distancias 16, 64 y 256, de acuerdo con las modulaciones QAM16, QAM64 y QAM256 respectivamente. En el caso de QPSK, no es necesario formar cantidades múltiples de 4, las salidas del bloque sorting network fluyen directamente al multiplexor de la salida. Al terminar el ordenamiento la bandera done se vuelve "1 lógico" indicando que el proceso de ordenamiento ha finalizado.

El bloque Sorting network es uno de los bloques fundamentales de la arquitectura, se compone solo de comparadores combinacionales cuyas salidas entregan el máximo y mínimo de las señales dadas de entrada. El funcionamiento es simple, dado 4 entradas a, b, c y d desordenadas, el bloque genera las salidas ordenadas f, g, h, i como se observa en la Figura 34.

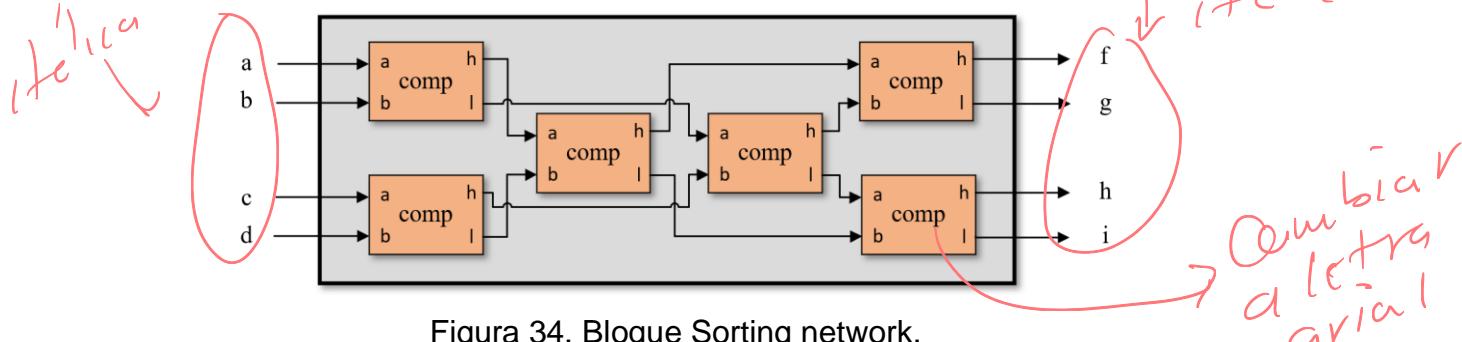


Figura 34. Bloque Sorting network.

Tabla 7. Descripción de entradas y salidas del bloque Sorting network.

Entrada	Longitud en bits	Descripción
a, b, c, d	16	Distancias desordenadas.
Salida	Longitud en bits	Descripción
f, g, h, i	16	Distancias ordenadas, tal que $f \leq g \leq h \leq i$.

El bloque Merge unit es el que presenta mayor complejidad de la arquitectura del ordenador híbrido, es el encargado de formar las distancias de las contestaciones

ya que

Esta desenada

QAM16, QAM64 y QAM256 tomando como base 4 entradas ordenadas que provienen del bloque *Sorting network*. Consiste en bloques basados en el algoritmo *merge sort*, en donde dos conjuntos ordenados de longitud n y t forman un vector ordenado de longitud $n + t$. *Merge unit* genera como salida las 4 distancias más pequeñas generadas por los bloques internos de acuerdo con el índice de modulación M . En la Figura 35 se observa a nivel de diagrama de bloque su composición y en la Tabla 8 las entradas y salidas.

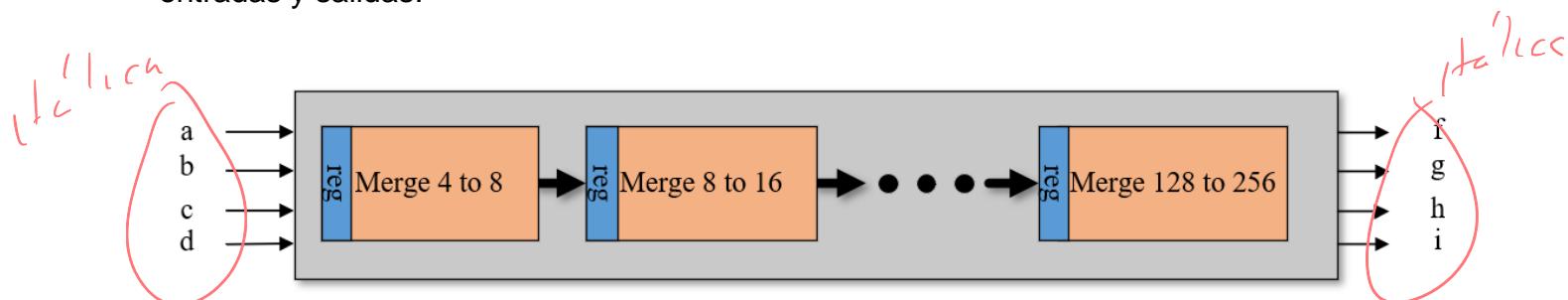


Figura 35. Bloque Merge unit.

Tabla 8. Descripción de entradas y salidas del bloque merge de 4 a 8.

Entrada	Longitud en bits	Descripción
a,b,c,d	16	Distancias ordenadas
Salida	Longitud en bits	Descripción
f,g,h,i	16	Las 4 distancias más pequeñas de las modulaciones QAM16, QAM64 y QAM256.

La Figura 36 representa el registro secuencial que tienen cada bloque Merge. Este bloque tiene la responsabilidad de formar un conjunto de salida de tamaño $2n$ tomando como base dos conjuntos de tamaño n . Para lograrlo, en el primer ciclo de reloj se guarda el vector de entrada a_1, a_2, \dots, a_n en el vector de salida $c_1, c_2, \dots, c_{n/2}$, durante el segundo ciclo se toma el vector de entrada $b_{\frac{n}{2}+1}, b_{\frac{n}{2}+2}, \dots, b_{2n}$ y lo guarda en $c_{\frac{n}{2}+1}, c_{\frac{n}{2}+2}, \dots, c_{2n}$. En la Tabla 9 se definen las entradas y salidas del registro.

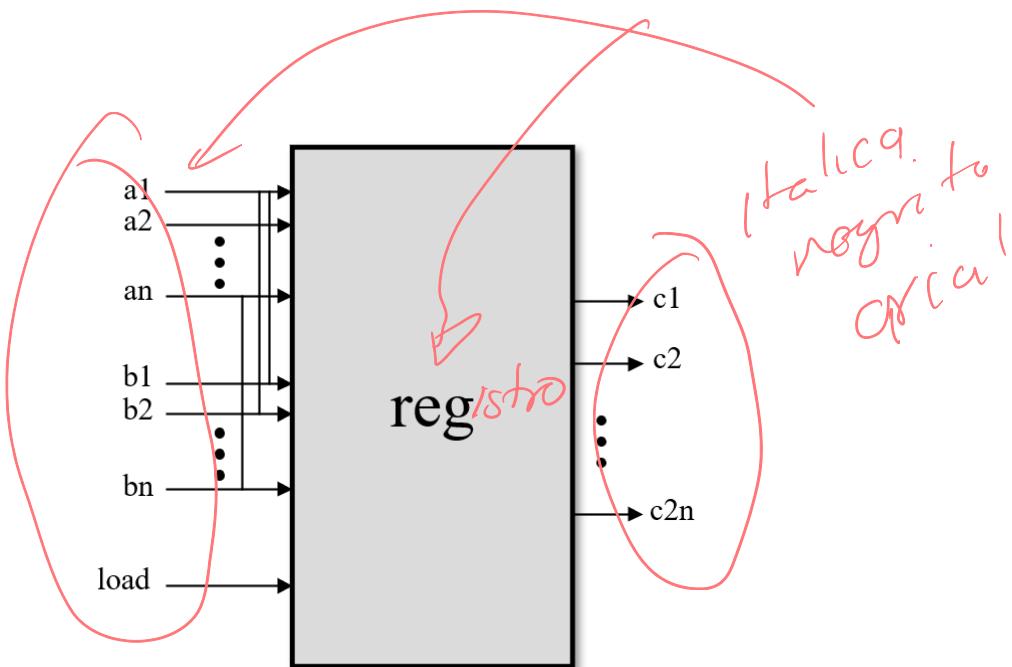


Figura 36. Registro de entrada de los sub bloques Merge.

Tabla 9. Descripción de entradas y salidas del bloque reg.

Entrada S	Longitud en bits	Descripción
a_1, a_2, \dots, a_n	16	Distancias ordenadas
b_1, b_2, \dots, b_n	16	Distancias ordenadas
Salida	Longitud en bits	Descripción
c_1, c_2, \dots, c_{2n}	16	Concatenación de los vectores de entrada a y b

En la Figura 37 se muestra el bloque merge de 4 a 8, de manera de ejemplo para ilustrar su estructura. Consiste en dos bloques Merge principales (bloques de color azul) de 2 a 4 y una columna de comparadores. El bloque de 4 a 8 toma dos conjuntos ordenados de 4 elementos y combina los dos conjuntos para dar una salida ordenada del doble de longitud. En la Tabla 10 se describen las entradas y salidas del bloque.

tal bloque.

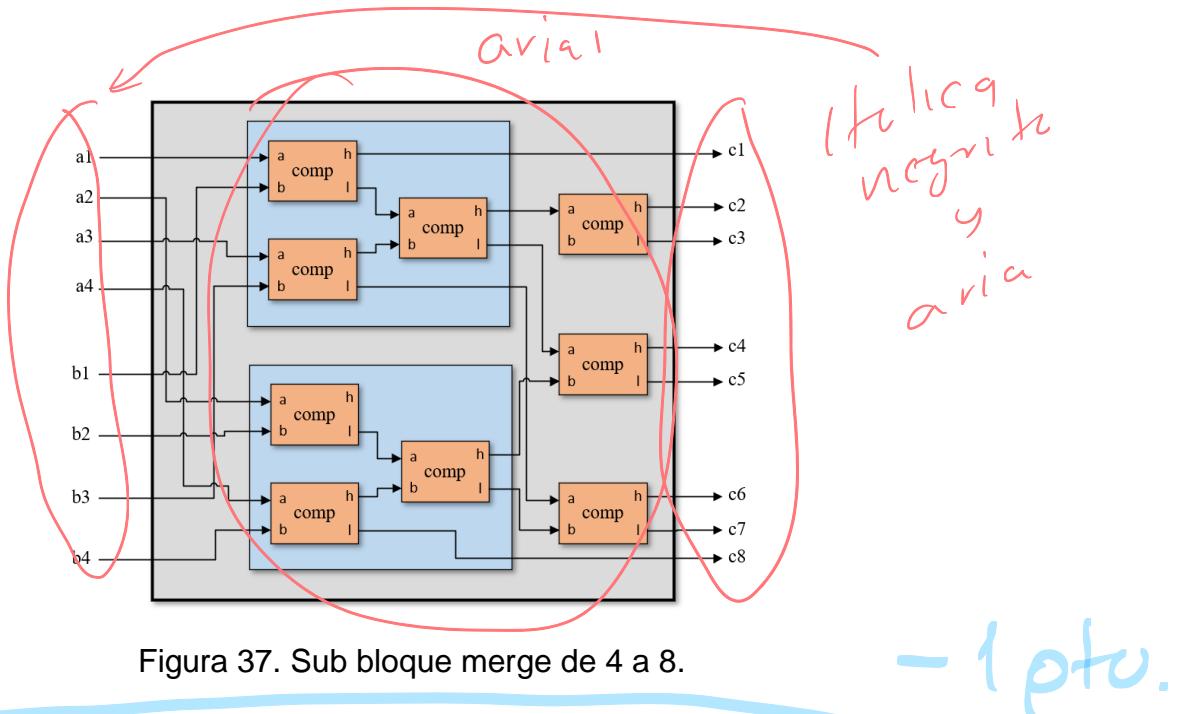


Figura 37. Sub bloque merge de 4 a 8.

Tabla 10. Descripción de entradas y salidas del bloque Merge 4 a 8.

Entrada <i>S</i>	Longitud en bits	Descripción
a_1, a_2, a_3, a_4	16	Distancias ordenadas
b_1, b_2, b_3, b_4	16	Distancias ordenadas
Salida <i>S</i>	Longitud en bits	Descripción
c_1, c_2, \dots, c_{2n}	16	Concatenación de los vectores de entrada <i>a</i> y <i>b</i> de manera ordenada.

a todos los

El bloque *control unit* de la Figura 38 es el elemento encargado de coordinar ~~los~~ *elementos* que componen a la arquitectura, de tal manera que en conjunto realicen el proceso de ordenamiento. Cuando la señal de *start* es 1 lógico, la unidad de control genera los valores necesarios en la señal *load* para poder coordinar al bloque *Merge unit* de acuerdo con el índice de modulación *M*. En la Tabla 11 se describen las ~~entradas y salidas~~ *entradas y salidas* del bloque.

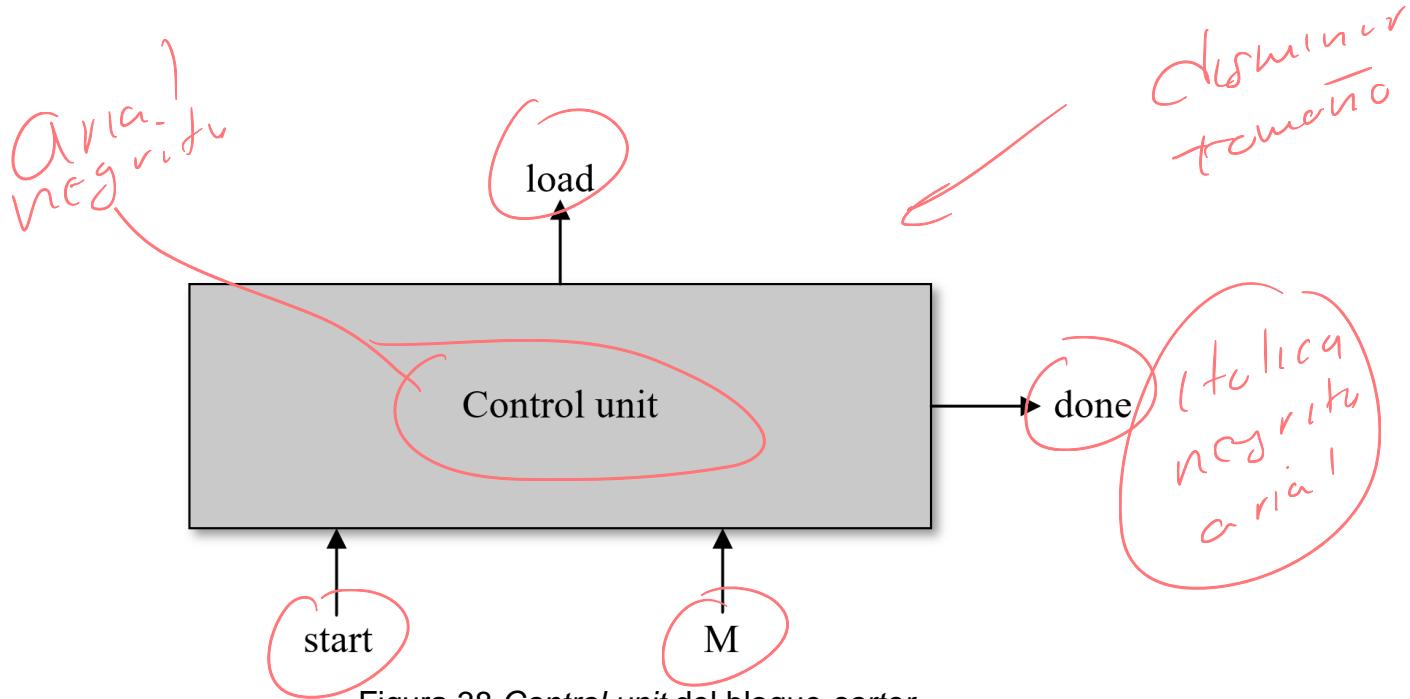


Figura 38. Control unit del bloque sorter.

Tabla 11. Descripción de entradas y salidas del bloque Control Unit.

Entrada <i>S</i>	Longitud en bits	Descripción
start	1	Señal que indica el inicio de la ejecución del módulo top
M	2	Índice de la modulación a utilizar
Salida	Longitud en bits	Descripción
load	16	señal de control para la unidad Merge
Done	1	Señal que indica cuando el proceso de detección ha finalizado

4.3.5 El bloque *dataAndPointers*

El bloque *dataAndPointers* de la Figura 39 tiene como objetivo obtener las direcciones de memoria de cada distancia ordenada que llega como entrada al bloque. Las entradas y salidas se muestran en la Tabla 12.

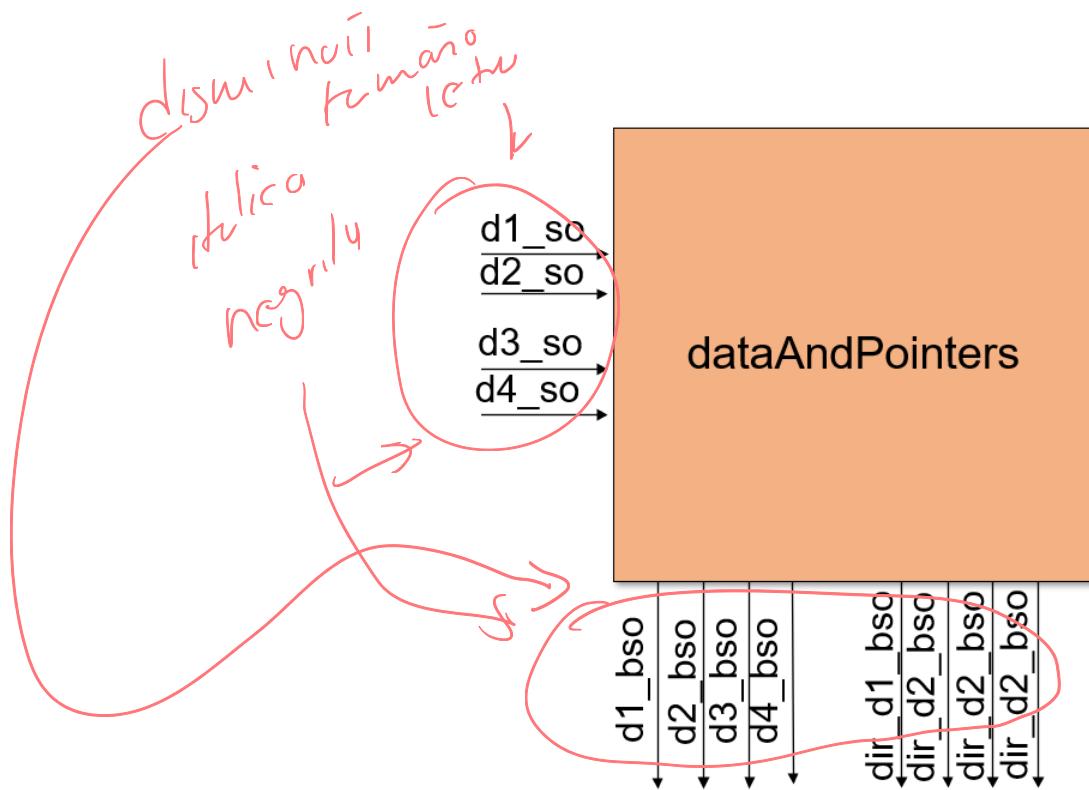


Figura 39. El bloque *addrAndPointers*.

Tabla 12. Descripción de entradas y salidas del bloque *dataAndPointers*.

Entrada	Longitud en bits	Descripción
d1_so, d2_so, d3_so, d4_so	16	Distancias ordenadas
Salida	Longitud en bits	Descripción
d1_bso, d2_bso, d3_bso, d4_bso	16	Distancias ordenadas
dir_d1_bso, dir_d2_bso, dir_d3_bso, dir_d4_bso	16	Direcciones de cada distancia ordenada

4.3.6 El bloque *survivors*

El bloque *survivors* mostrado en la Figura 40, tiene como objetivo escoger el número de sobrevivientes en cada nivel del árbol y generar tres vectores relacionados con las distancias. Las entradas y salidas se muestran en la Tabla 13.

Sus

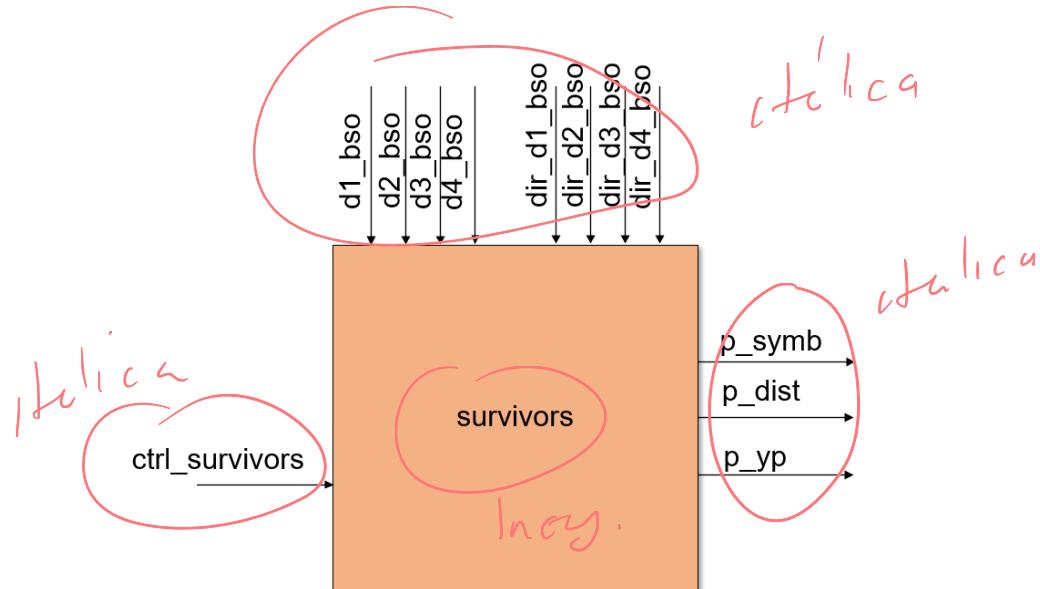


Figura 40. El bloque *survivors*.

Tabla 13. Descripción de entradas y salidas del bloque data *survivors*.

Entrada	Longitud en bits	Descripción
$d1, d2, d3, d4$	16	Las cuatro distancias más pequeñas
$addr1, addr2, addr3, addr4$	16	Direcciones de cada distancia ordenada
$ctrl_survivors$	16	Línea de control
Salida	Longitud en bits	Descripción
p_symb	16	Símbolo padre del símbolo sobreviviente
p_dist	16	Distancia del símbolo sobreviviente
p_yp	16	Vector recibido

4.3.7 El bloque *detection*

La función del bloque *detection* mostrado en la Figura 41 es estimar el símbolo recibido con la ayuda un conjunto de entradas que representan un árbol de búsqueda. Las entradas y salidas se muestran en la Tabla 14.

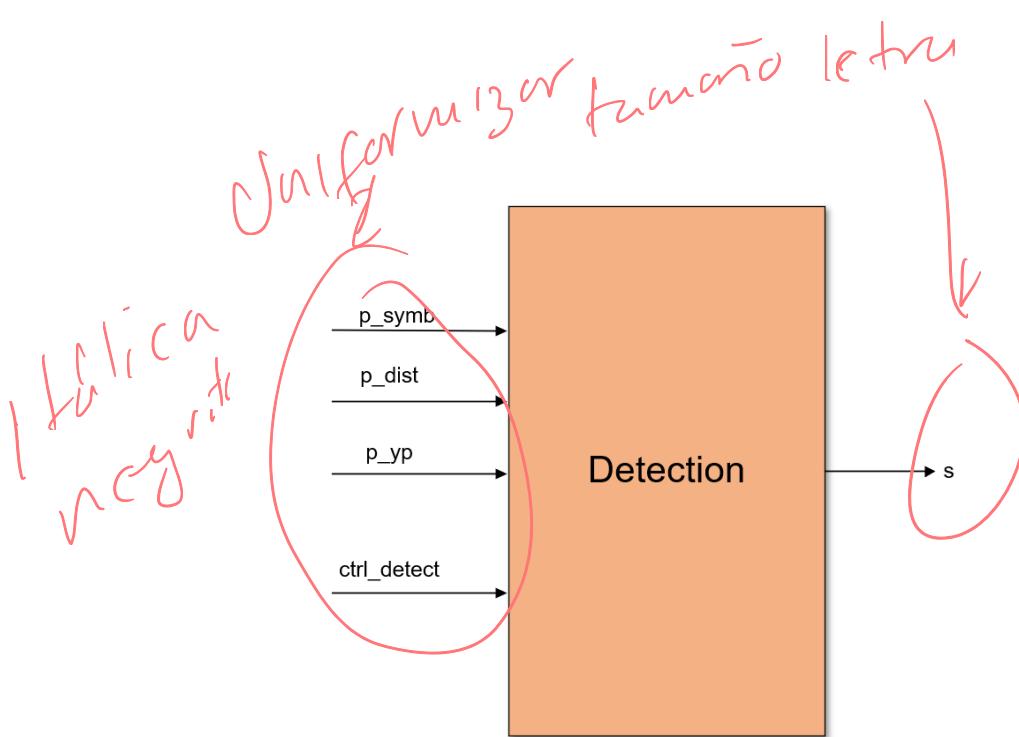


Figura 41. El bloque *detection*.

- I ptp petra

Tabla 14. Descripción de entradas y salidas del bloque *detection*.

Entrada	Longitud en bits	Descripción
p_symb	16	Símbolo padre del símbolo sobreviviente
p_dist	16	Distancia del símbolo sobreviviente
p_yp	16	Vector recibido
ctrl_detect	16	Línea de control
Salida	Longitud en bits	Descripción
S	16	Símbolo detectado

4.3.8 El bloque *topControlUnit*

Por último, en la Figura 42, el bloque que controla la secuencia necesaria para generar el símbolo estimado. Controla al bloque *survivors* y *detection* con el fin de generar el número de iteraciones necesarias para realizar la detección de datos. Las entradas y salidas se muestran en la Tabla 15.

Disminuir tamaño figura

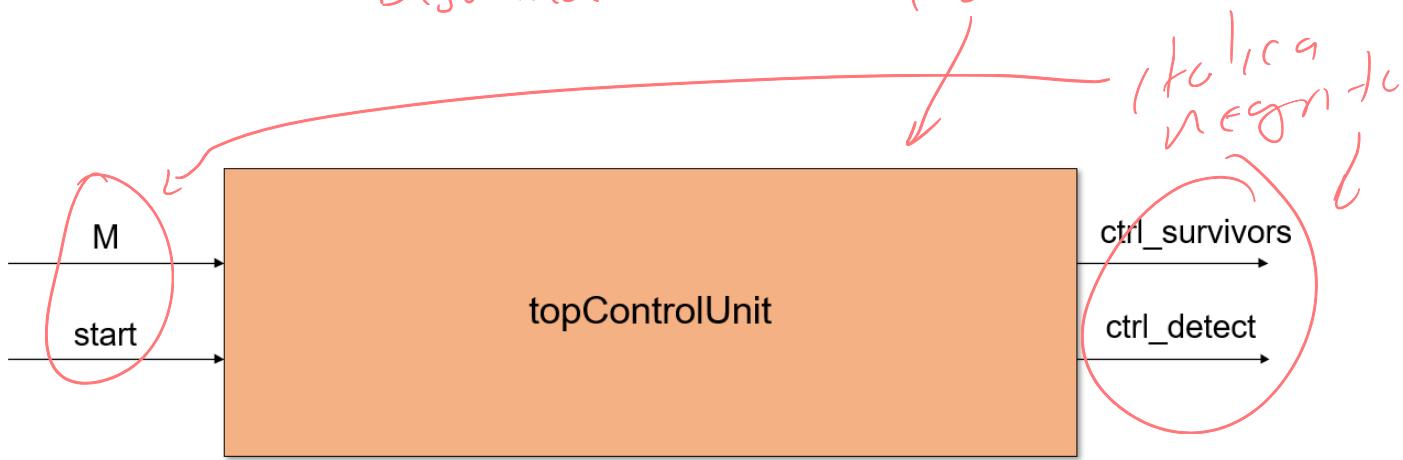


Figura 42. Unidad de control para el bloque *top*

Tabla 15. Descripción de entradas y salidas del bloque *top*.

Entrada	Longitud en bits	Descripción
start	1	Señal que indica el inicio de la ejecución del módulo top
M	2	Índice de la modulación a utilizar
Salida	Longitud en bits	Descripción
ctrl_survivors	16	Línea de control para el bloque <i>survivors</i>
ctrl_detect	16	Línea de control para el bloque <i>detection</i>

-1 ptos. letra

CAPÍTULO V. RESULTADOS

ambos ordenadores de datos:

RTL-HDL

En este capítulo se presentan resultados de la síntesis, latencia y la correspondiente verificación para el [redacted] diseñado con HLS y el [redacted] híbrido con [redacted]. En cada sección se agregan tablas o imágenes obtenidas en reportes de simulación con Vivado e ISE Design Suite 14.7, para ambos enfoques de diseño.

5.1 Síntesis

Resultados de síntesis

Síntesis a nivel HLS

5.1.1 Diseño HLS

En la Figura 43 se muestra (a manera de ejemplo) el módulo de hardware correspondiente a la síntesis del algoritmo de inserción y su correspondiente arquitectura RTL. Se observa los puertos de entrada y salida del bloque de hardware generado por Vivado HLS.

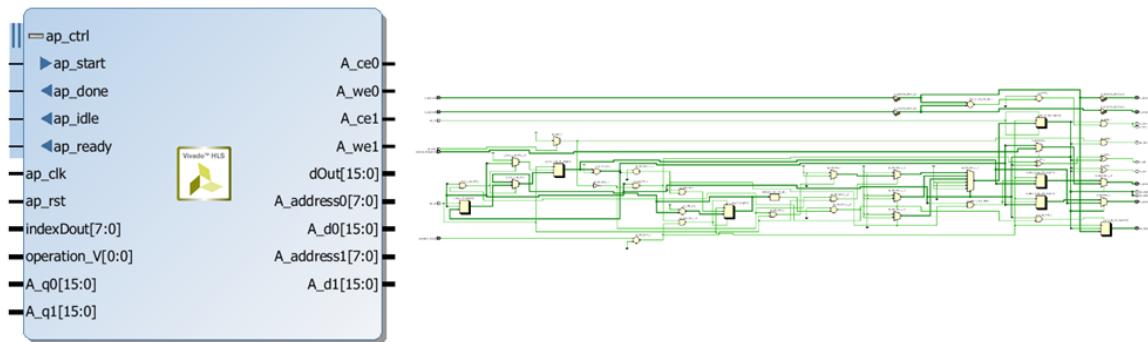


Figura 43. Bloque y RTL del algoritmo de inserción.

Módulo y arquitectura RTL
obtenidos en HLS por el alg. LS

En la Figura 44 se presentan de forma resumida, los resultados del consumo de hardware de cada algoritmo para cantidades de elementos a ordenar coincidente con los tamaños de constelación QAM ($N = 4, 16, 64$ y 256) definidas en el detector. Todas las arquitecturas de los algoritmos presentan consumos reducidos de *Flip-Flops* (FFs) y LUTs (*Lookup Tables*) en sus síntesis y ninguno de ellos requiere bloques dedicados de RAM o DSPs.

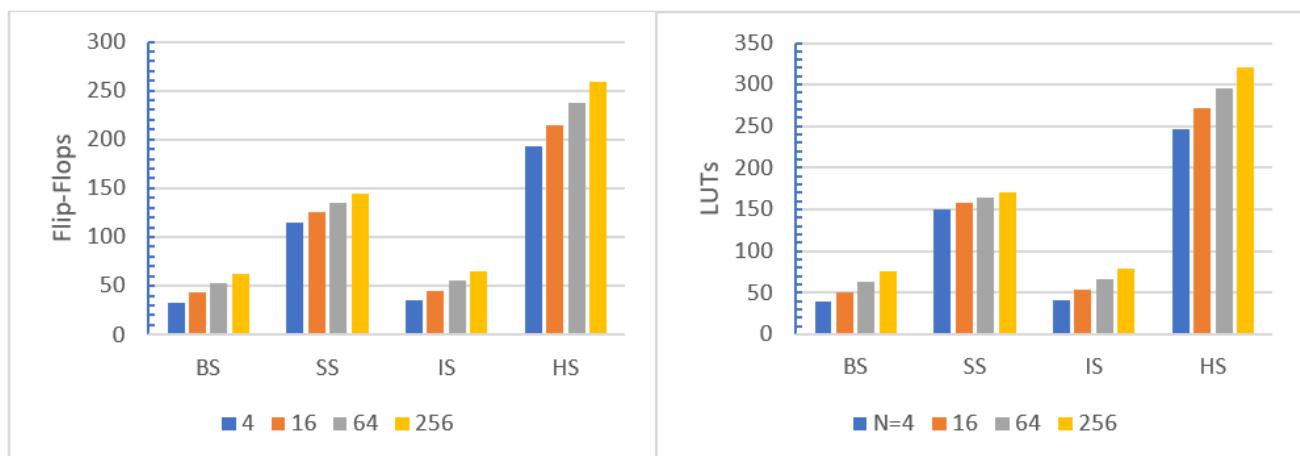


Figura 44. Resultados de síntesis; consumo de FFs (izquierda) y LUTs (derecha)

~~5.1.2 Arquitectura~~

Síntesis a nivel HDL - RTL

La síntesis del detector se llevó a cabo en el software de desarrollo ISE Design Suite 14.7 para el bloque sorter, el dispositivo seleccionado fue el FPGA de Xilinx de la familia Artix-7, modelo xc7a100t-3csg324. Los resultados de la síntesis quedan resumidos en la Tabla 16.

Se muestran en la tabla 16.

Tabla 16. Recursos consumidos por el bloque sorter.

Recurso consumido	Usado	Disponible	Utilización
Número de slices registros	10	126800	0.007%
Número de slices LUTs	19	63400	0.002%
Número de IOBs	70	210	33.33%
Número de BUFG/BUFGCTRLs	1	32	3.125%

- 1 pto *letra*

Resultados de Latencia en HLS

5.2 Latencia

5.2.1 Diseño HLS

En la Figura 45 se presenta el desempeño en términos de latencia (izquierda) y frecuencia de operación (derecha). La latencia máxima representa los ciclos de reloj necesarios para lograr ordenar los N elementos. Como puede notarse, la menor latencia la presenta el hardware del algoritmo *bubble sort* y la peor el del *heap sort*.

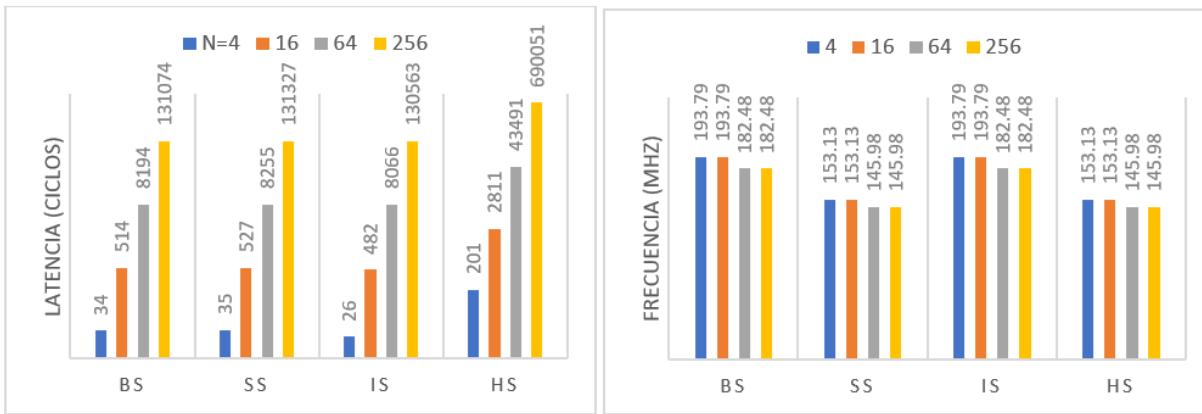


Figura 45. Resultados de rendimiento; latencia (izq.) y freq. max. de operación (derecha).

En lo que respecta a la velocidad, la Figura 4 (derecha) exhibe el comportamiento de la frecuencia máxima de operación de las arquitecturas sintetizadas para los valores de N ya mencionados. La arquitectura más veloz fue la del algoritmo de inserción con una $f_{max} = 193 \text{ MHz}$. El peor caso se tiene para la del *heap-sort* con una $f_{max} = 149 \text{ MHz}$.

5.2.2 Arquitectura La latencia en HDL-RTL

En la Tabla 17 se muestran los ciclos que necesita el bloque *sorter* para generar las distancias ordenadas en cada tipo de modulación, se observa que para la modulación QPSK no ocupa ningún ciclo, esto se debe a que el bloque es completamente combinacional.

Tabla 17. Latencia del bloque sorter.

-1pto letra

Modulacion	Ciclos
QPSK	0
QAM16	6
QAM64	24
QAM256	96

Resultados de
5.3 Verificación

5.3.2 Diseño HLS

Test-bench en HLS

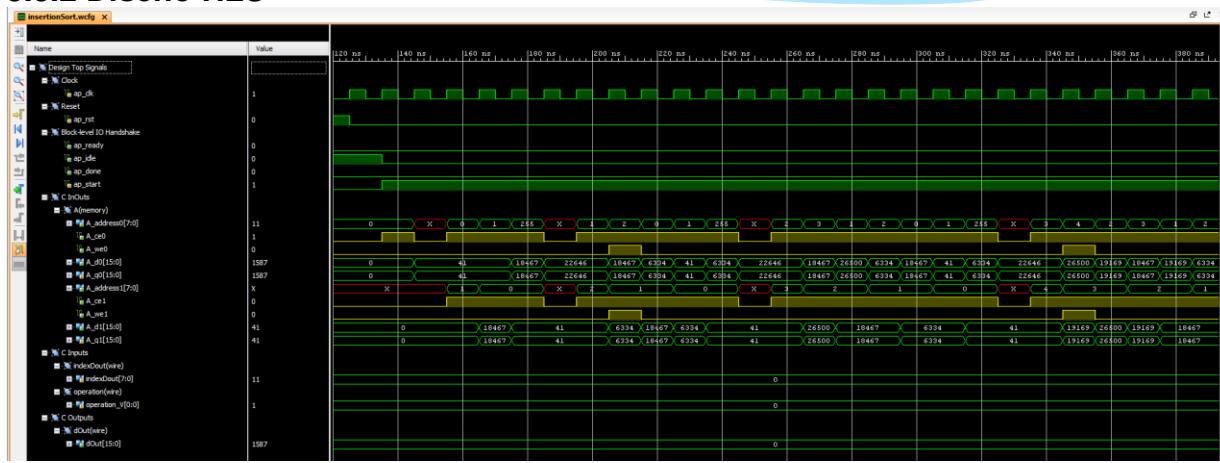


Figura 46 se presenta el diagrama de tiempos del algoritmo de inserción. La ejecución empieza cuando la señal *ap_start* se vuelve un uno lógico, en ese momento el bloque de hardware inicia con el proceso de ordenamiento generando direcciones para leer o escribir en la memoria externa. Después de una cantidad de ciclos de reloj, la bandera *ap_done* indica que los elementos están completamente ordenados y es posible seguir con la etapa de lectura como se muestra en la Figura 47 con la señal de salida *dOut*.

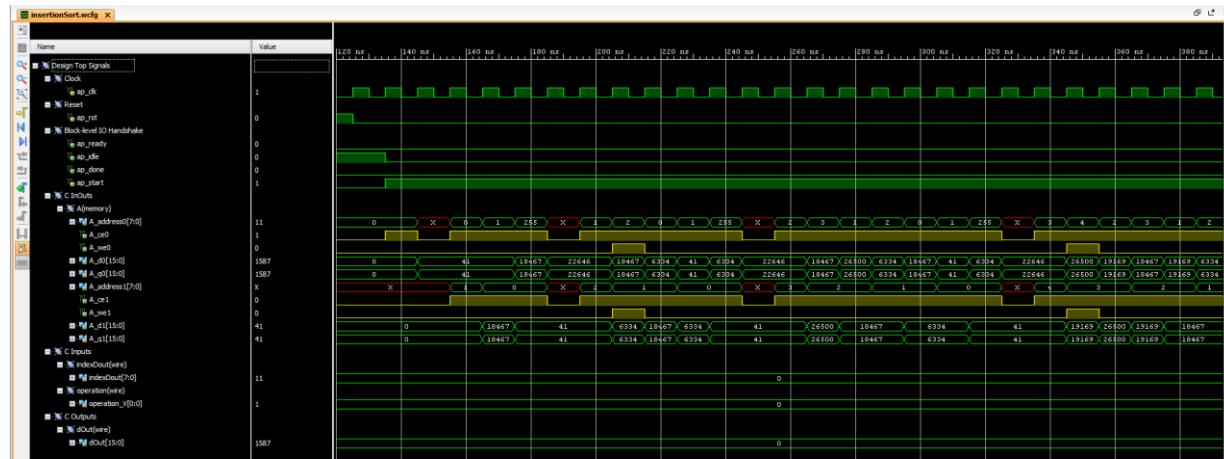


Figura 46. Cronograma de operación del módulo durante el inicio de ordenamiento para

→ acercar aca

el algoritmo de inserción.



Figura 47. Cronograma de operación del módulo en la fase final de ordenamiento para el algoritmo de inserción.

5.3.3 Arquitectura

Fast-Launch o nivel HDL-RTL

Como ejemplo, todas las simulaciones del detector corresponden a la modulación QAM16. A continuación, se muestra la verificación correspondiente al módulo *sorting network, merge unit* de 4 a 8 y el módulo *Top (sorter)*

El bloque *sorting network* realiza el ordenamiento de cuatro distancias en cada ciclo de reloj, ya que en cada ciclo existe un nuevo dato de entrada. Al comportamiento anterior, se observa en la Figura 48.

Entradas	Msgs
+ /sorter_tb/UUT/sortingNetwork/d1	8'd122
+ /sorter_tb/UUT/sortingNetwork/d2	8'd86
+ /sorter_tb/UUT/sortingNetwork/d3	8'd91
+ /sorter_tb/UUT/sortingNetwork/d4	8'd15
Salidas	
+ /sorter_tb/UUT/sortingNetwork/y1	8'd15
+ /sorter_tb/UUT/sortingNetwork/y2	8'd86
+ /sorter_tb/UUT/sortingNetwork/y3	8'd91
+ /sorter_tb/UUT/sortingNetwork/y4	8'd122

Figura 48. Prueba funcional de la red de ordenamiento con QAM16.

El bloque *Merge unit* combina cuatro distancias en cada ciclo de reloj. Como ejemplo, se muestra el su comportamiento con el bloque de 4 a 8, esto es, toma dos entradas de 4 para formar una salida de 8. En la modulación QAM16, se necesita formar dos

conjuntos de 8 distancias para posteriormente juntarlas y tener una salida de 16 distancias, tal comportamiento lo observamos en la Figura 49.

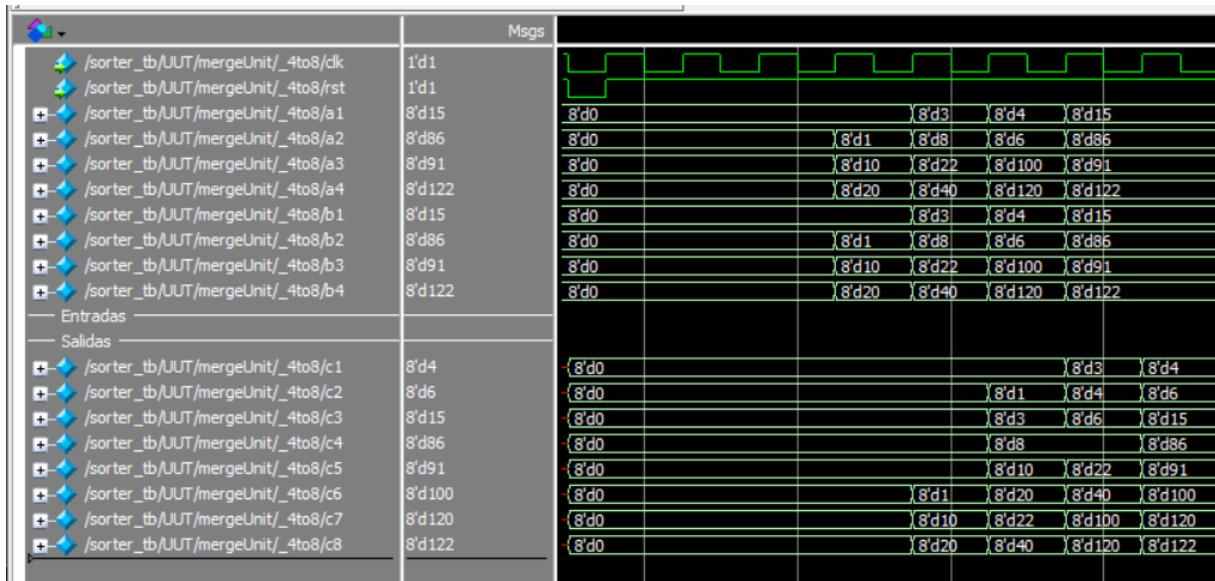


Figura 49. Prueba funcional de la *merge unit* de 4 a 8.

La Figura 50 corresponde a la verificación funcional del bloque *sorter*, se observa que el tipo de modulación está seleccionada a QAM16 ($M = 01$). Cuando la señal de entrada se vuelve uno lógico empieza la operación de ordenamiento, en cada ciclo de reloj se ordenan cuatro distancias para formar 16 y seleccionar las cuatro más pequeñas, después, se transfieren a la salida. El comportamiento anterior, lo observamos en la Figura 50.

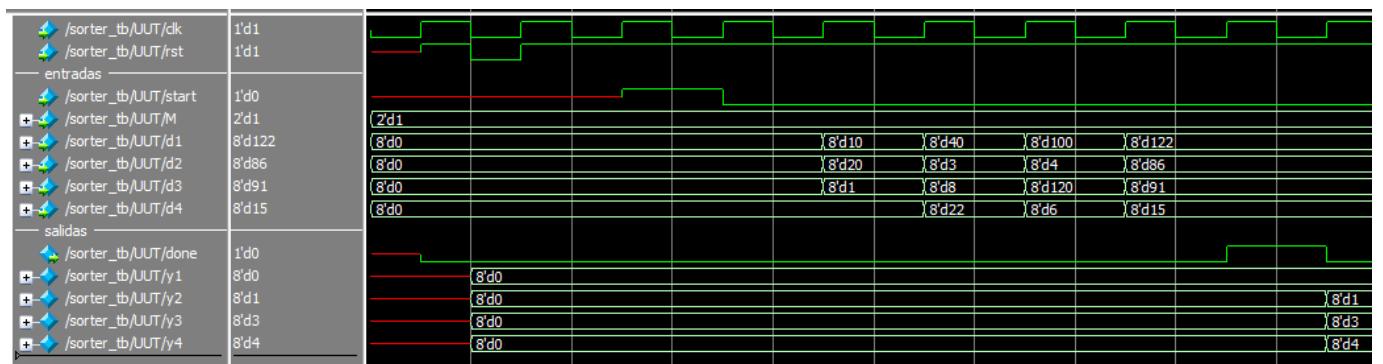


Figura 50. Verificación funcional del boque *sorter* cuando $M = \text{QAM16}$.

secuencial / paralelo

cuello de botella del detector y el ordenamiento de datos
ordenamiento de datos
submódulo de

CAPÍTULO VI. CONCLUSIONES

En este trabajo se propuso una arquitectura digital del algoritmo de detección de símbolos Near-ML para un sistema de comunicaciones SISO-OFDM en ambientes V2V. Se hizo énfasis en implementar el sub bloque ordenador de la arquitectura debido a que es el más relevante de mayor uso durante la ejecución del algoritmo. El ordenador fue diseñado utilizando un enfoque híbrido secuencial y paralelo, logrando así, un diseño eficiente en latencia y recursos consumidos del FPGA. Al mismo tiempo, se utilizó HLS para realizar el prototipado rápido de los algoritmos de ordenamiento más empleados actualmente. Logrando en ellos, una uniformidad en el consumo reducido de hardware, bajas latencias y velocidad elevadas. Así, con base a lo anterior se comprobó que, la arquitectura obtenida para el algoritmo de ordenamiento de inserción y la arquitectura híbrida se perfilan como las opciones con mayor viabilidad para ser incorporadas en el diseño de sistemas receptores OFDM de última generación con detección Near-ML. Como futuro trabajo se pueden implementar los bloques restantes de la propuesta de arquitectura y así obtener las métricas de eficiencia del algoritmo de detección a partir de una implementación en hardware.

del desarrollo final
del sistema receptor

dejar un espacio entre ítems

Bibliografía

- [1] S. Kulkarni, A. Darekar, and S. Shirol, "Proposed framework for V2V communication using Li-Fi technology," in *2017 International Conference on Circuits, Controls, and Communications (CCUBE)*, Bangalore, Dec. 2017, pp. 187–190. doi: 10.1109/CCUBE.2017.8394163.
- [2] K. Abboud, H. A. Omar, and W. Zhuang, "Interworking of DSRC and Cellular Network Technologies for V2X Communications: A Survey," *IEEE Trans. Veh. Technol.*, vol. 65, no. 12, pp. 9457–9470, Dec. 2016, doi: 10.1109/TVT.2016.2591558.
- [3] J. Lianghai, A. Weinand, B. Han, and H. D. Schotten, "Multi-RATs support to improve V2X communication," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, Barcelona, Apr. 2018, pp. 1–6. doi: 10.1109/WCNC.2018.8377432.
- [4] S. Gyawali, S. Xu, Y. Qian, and R. Q. Hu, "Challenges and Solutions for Cellular Based V2X Communications," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 1, pp. 222–255, 2021, doi: 10.1109/COMST.2020.3029723.
- [5] "The details of V2X communication," presented at the <https://www.epdtonthenet.net/article/178093/Under-the-hood-of-v2x-communications.aspx>.
- [6] K. Eshteiwi, K. Ben Fredj, G. Kaddoum, and F. Gagnon, "Performance analysis of peer-to-peer V2V wireless communications in the presence of interference," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Montreal, QC, Oct. 2017, pp. 1–6. doi: 10.1109/PIMRC.2017.8292202.
- [7] D. Boehmlaender, S. Hasirlioglu, V. Yano, C. Lauerer, T. Brandmeier, and A. Zimmer, "Advantages in Crash Severity Prediction Using Vehicle to Vehicle Communication," in *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, Rio de Janeiro, Brazil, Jun. 2015, pp. 112–117. doi: 10.1109/DSN-W.2015.23.
- [8] T. Bey and G. Tewolde, "Evaluation of DSRC and LTE for V2X," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, Jan. 2019, pp. 1032–1035. doi: 10.1109/CCWC.2019.8666563.
- [9] H. P. Dai Nguyen and R. Zoltan, "The Current Security Challenges of Vehicle Communication in the Future Transportation System," in *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*, Subotica, Sep. 2018, pp. 000161–000166. doi: 10.1109/SISY.2018.8524773.
- [10] "vehicle_to_vehicle_communications_readiness_of_V2V_technology_for_application."
- [11] R. Sabouni and R. M. Hafez, "Performance of DSRC for V2V communications in urban and highway environments," in *2012 25th IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, Montreal, QC, Apr. 2012, pp. 1–5. doi: 10.1109/CCECE.2012.6335027.
- [12] "V2V for Vehicular Safety Applications."
- [13] Z. MacHardy, A. Khan, K. Obama, and S. Iwashina, "V2X Access Technologies:

Incapacitada

- Regulation, Research, and Remaining Challenges," *IEEE Commun. Surv. Tutorials*, vol. 20, no. 3, pp. 1858–1877, 2018, doi: 10.1109/COMST.2018.2808444.
- [14] Md. S. Hossen, M.-S. Bang, Y. Park, and K.-D. Kim, "Performance analysis of an OFDM-based method for V2X communication," in *2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*, Shanghai, China, Jul. 2014, pp. 238–242. doi: 10.1109/ICUFN.2014.6876789.
- [15] T. Maehata *et al.*, "DSRC using OFDM for roadside-vehicle communication system," in *VTC2000-Spring. 2000 IEEE 51st Vehicular Technology Conference Proceedings (Cat. No.00CH37026)*, Tokyo, Japan, 2000, vol. 1, pp. 148–152. doi: 10.1109/VETECS.2000.851435.
- [16] "near_ML_detection_using_dikstras_algorithm_with_bounded_list_size_over_MI_MO_channels."
- [17] V. Kiray, S. Demir, and M. Zhaparov, "Improving Digital Electronics Education with FPGA technology, PBL and Micro Learning methods," in *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, Bali, Indonesia, Aug. 2013, pp. 445–448. doi: 10.1109/TALE.2013.6654479.
- [18] I. Kuon, R. Tessier, and J. Rose, "FPGA Architecture: Survey and Challenges," *FNT in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2007, doi: 10.1561/1000000005.
- [19] K. Georgopoulos *et al.*, "An evaluation of vivado HLS for efficient system design," in *2016 International Symposium ELMAR*, Zadar, Croatia, Sep. 2016, pp. 195–199. doi: 10.1109/ELMAR.2016.7731785.
- [20] F. Callaly, D. O'Loughlin, D. Lyons, A. Coffey, and F. Morgan, "Xilinx Vivado High Level Synthesis: Case studies," in *25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, Limerick, Ireland, 2014, pp. 352–356. doi: 10.1049/cp.2014.0713.
- [21] "ug871-vivado-high-level-synthesis-tutorial."
- [22] "wp416-Vivado-Design-Suite."
- [23] J. A. Del Puerto-Flores, J. Cortez, C. A. Gutierrez, C. Del Valle-Soto, R. Velazquez, and L. J. Valdivia, "Performance of MRC Detection in OFDM System with Virtual Carriers over V2V Channels," in *2019 IEEE 39th Central America and Panama Convention (CONCAPAN XXXIX)*, Guatemala City, Guatemala, Nov. 2019, pp. 1–6. doi: 10.1109/CONCAPANXXXIX47272.2019.8976929.
- [24] Hector Eduardo Aldrete Vidrio, "Sistema de comunicación multiportadora para el estándar 802.11p utilizando precodificación frecuencial y cancelación no lineal de interferencia," 2019.
- [25] "IEEE draft standard for information technology."
- [26] T. Zemen, L. Bernado, N. Czink, and A. F. Molisch, "Iterative Time-Variant Channel Estimation for 802.11p Using Generalized Discrete Prolate Spheroidal Sequences," *IEEE Trans. Veh. Technol.*, vol. 61, no. 3, pp. 1222–1233, Mar. 2012, doi: 10.1109/TVT.2012.2185526.
- [27] X. Cheng, C.-X. Wang, D. Laurenson, S. Salous, and A. Vasilakos, "An adaptive geometry-based stochastic model for non-isotropic MIMO mobile-to-mobile channels," *IEEE Trans. Wireless Commun.*, vol. 8, no. 9, pp. 4824–4835, Sep. 2009, doi:

10.1109/TWC.2009.081560.

- [28] Cheng-xiang Wang, Xiang Cheng, and D. Laurenson, "Vehicle-to-vehicle channel modeling and measurements: recent advances and future challenges," *IEEE Commun. Mag.*, vol. 47, no. 11, pp. 96–103, Nov. 2009, doi: 10.1109/MCOM.2009.5307472.
- [29] X. Cheng *et al.*, "An Improved Parameter Computation Method for a MIMO V2V Rayleigh Fading Channel Simulator Under Non-Isotropic Scattering Environments," *IEEE Commun. Lett.*, vol. 17, no. 2, pp. 265–268, Feb. 2013, doi: 10.1109/LCOMM.2013.011113.121535.
- [30] X. Cheng *et al.*, "Cooperative MIMO Channel Modeling and Multi-Link Spatial Correlation Properties," *IEEE J. Select. Areas Commun.*, vol. 30, no. 2, pp. 388–396, Feb. 2012, doi: 10.1109/JSAC.2012.120218.
- [31] I. Sen and D. W. Matolak, "Vehicle–Vehicle Channel Models for the 5-GHz Band," *IEEE Trans. Intell. Transport. Syst.*, vol. 9, no. 2, pp. 235–245, Jun. 2008, doi: 10.1109/TITS.2008.922881.
- [32] G. Acosta-Marum and M. A. Ingram, "Six time- and frequency- selective empirical channel models for vehicular wireless LANs," *IEEE Veh. Technol. Mag.*, vol. 2, no. 4, pp. 4–11, Dec. 2007, doi: 10.1109/MVT.2008.917435.
- [33] 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 J. Select. Areas Commun.*, vol. 31, no. 9, pp. 434–448, Sep. 2013, doi: 10.1109/JSAC.2013.SUP.0513039.
- [34] F. Pena-Campos, R. Carrasco-Alvarez, O. Longoria-Gandara, and R. Parra-Michel, "Estimation of Fast Time-Varying Channels in OFDM Systems Using Two-Dimensional Prolate," *IEEE Trans. Wireless Commun.*, vol. 12, no. 2, pp. 898–907, Feb. 2013, doi: 10.1109/TWC.2013.010413.120624.
- [35] M. O. Damen, H. El Gamal, and G. Caire, "On maximum-likelihood detection and the search for the closest lattice point," *IEEE Trans. Inform. Theory*, vol. 49, no. 10, pp. 2389–2402, 2003.
- [36] D. Wübben, R. Böhnke, J. Rinas, V. Kühn, and K. D. Kammeyer, "Efficient algorithm for decoding layered space-time codes," *Electron. Lett.*, vol. 37, no. 22, p. 1348, 2001, doi: 10.1049/el:20010899.
- [37] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm I. Expected complexity," *IEEE Trans. Signal Process.*, vol. 53, no. 8, pp. 2806–2818, 2005.
- [38] H. Moroga, T. Yamamoto, and F. Adachi, "Iterative Overlap TD-QRM-ML Block Signal Detection for Single-Carrier Transmission without CP Insertion," in *2012 IEEE Vehicular Technology Conference (VTC Fall)*, 2012, pp. 1–5.
- [39] K. Temma, T. Yamamoto, and F. Adachi, "Improved 2-Step QRM-ML Block Signal Detection for Single-Carrier Transmission," in *2011 IEEE Vehicular Technology Conference (VTC Fall)*, 2011, pp. 1–5.
- [40] "Arquitecturas digitales de procesamiento de señales para sistemas de comunicacion con entrenamiento."
- [41] A. Farmahini-Farahani, H. J. Duwe, M. J. Schulte, and K. Compton, "Modular Design of High-Throughput, Low-Latency Sorting Units," *IEEE Trans. Comput.*, vol. 62, no. 7, pp. 1389–1402, Jul. 2013.
- [42] D. Chen, J. Cong, and P. Pan, "FPGA Design Automation: A Survey," *FNT in Electronic Design Automation*, vol. 1, no. 3, pp. 195–334, 2006.

- [43] I. Skliarova and V. Sklyarov, *FPGA-BASED Hardware Accelerators*, vol. 566. Cham: Springer International Publishing, 2019.
- [44] Y.-K. Choi, Y. Chi, J. Wang, and J. Cong, "FLASH: Fast, Parallel, and Accurate Simulator for HLS," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Dec. 2020, vol. 39, pp. 4828–4841.
- [45] M. M. Kamruzzaman, "Performance of Turbo coded wireless link for SISOOFDM, SIMO-OFDM, MISO-OFDM and MIMO-OFDM system," in *14th International Conference on Computer and Information Technology (ICCIT 2011)*, 2011, p. 6.
- [46] I. Baig and V. Jeoti, "A new ZCT precoded OFDM system with pulse shaping: PAPR analysis," in *2010 IEEE Asia Pacific Conference on Circuits and Systems*, 2010, pp. 1131–1134.

CHECA EN LA TESIS
DE AERTO Y DEL
DAKO CINC CITAR
LINKS, PDF'S, MANCHAS
Y CORRIGE

Anexo – Artículo publicado

DISEÑO HLS DE ALGORITMOS DE ORDENAMIENTO PARA RECEPTORES OFDM CON DETECTOR DE SÍMBOLOS NEAR-ML

Aarón Escoboza-Villegas, Eduardo Romero-Aguirre

Instituto Tecnológico de Sonora

[*aaron.villegas@hotmail.com](mailto:aaron.villegas@hotmail.com), eduardo.romero@itson.edu.mx

Resumen

En un sistema receptor OFDM, la etapa de ordenamiento de datos es esencial para detección de símbolos mediante el algoritmo *Near-ML* ya que condiciona su desempeño y por ende de todo el sistema de comunicación. Por dicha razón, la motivación detrás de esta investigación es el diseño de arquitecturas de hardware en FPGA; bajo el enfoque HLS, de un conjunto de algoritmos de ordenamiento de datos y evaluar su desempeño en términos de métricas tales como: latencia, tiempo de ejecución, consumo de recursos de hardware y frecuencia de operación, para la cantidad de datos requerida por un detector de símbolos *Near-ML*. Esto permitirá establecer la factibilidad de implementación de las arquitecturas digitales de receptores OFDM de esta índole en sistemas de comunicación inalámbricos contemporáneos.

Palabras clave: Diseño digital, FPGA, Ordenamiento de datos, Telecomunicaciones.

Introducción

El ordenamiento es una de las operaciones fundamentales en la computación y tiene un amplio rango de aplicaciones tales como: procesamiento digital de señales, base de datos, procesamiento de imagen y video [41]. Dicho proceso puede modelarse como, dada una secuencia de N números $\langle a_1, a_2, \dots, a_N \rangle$, el problema de ordenamiento se puede definir como la permutación $\langle a'_1, a'_2, \dots, a'_N \rangle$ tal que $a'_1 \leq a'_2 \leq \dots \leq a'_N$. Los algoritmos *bubble sort*, (BS), *insertion sort* (IS), *selection sort* (SS) y *heap sort* (HS) son de los más usados. Por otro lado, en las últimas décadas la industria semiconductora ha crecido de manera exponencial. Los dispositivos de tecnología programable y en

particular los FPGA (*Field Programmable Gate Array*) ha destacado por su mayor densidad y velocidad de operación en comparación con otras tecnologías [42]. Dado el vasto campo de aplicación de algoritmos PDS que requieren etapas de ordenamiento y el crecimiento de los FPGAs como tecnología de desarrollo digital; el trabajo conjunto de estas dos se ha convertido en tema actual de investigación. Pese a lo anterior, los algoritmos de ordenamiento tienen la peculiaridad de ser difícilmente modelables usando HDLs (*Hardware Description Languages*), lo cual conlleva a códigos pocos eficientes, largos y engorrosos [43]. Herramientas de síntesis de alto nivel (HLS), tal como Vivado HLS han surgido para acortar los tiempos de diseño [44]. En HLS, los algoritmos son descritos en lenguajes de alto nivel y posteriormente son_sintetizados automáticamente por la herramienta en cuestión. En otro orden de ideas, la técnica OFDM (*Orthogonal Frequency Division Multiplexing*) ha llegado a ser el común denominador en sistemas de comunicación inalámbricos actuales [45]. Su idea básica es dividir el canal en varios subcanales ortogonales, los cuales se transmiten en paralelo mejorando el manejo del ancho de banda, reduciendo la ISI (*Intersymbol Interference*) y facilitando el trabajo del sistema receptor [46]. También para incrementar el desempeño de un sistema de comunicación es necesario reducir la complejidad de la etapa de detección de datos. Así, el detector óptimo de máxima verosimilitud (*Maximum Likelihood - ML*) presenta una complejidad de $O(N_D \Omega_D^3)$, donde Ω es el tamaño de la constelación Ω y la cantidad de subportadoras es N_D . Su implantación es poco viable en comparación con los detectores lineales cuya complejidad es acotada por $O(N_D^3)$ [35]. Una de las recientes propuesta es el detector *Near-ML*, el cual presenta baja complejidad y un desempeño en términos de tasa de error de bit (BER) que se acerca al ML [35]. Una de las etapas cruciales en el funcionamiento del detector es la del ordenamiento de una cantidad N de distancias acorde al tamaño de la constelación. Consecuentemente, el algoritmo de ordenamiento y su implementación en hardware *determina* en gran medida el rendimiento y consumo del bloque detector.

Objetivo

Diseñar arquitecturas en FPGA, bajo el enfoque HLS, de un conjunto de algoritmos de ordenamiento candidatos para un detector de símbolos *Near-ML*; evaluando su

desempeño en términos de métricas tales como: latencia, tiempo de ejecución, consumo de recursos de *hardware* y frecuencia de operación; a fin de establecer la factibilidad de su implantación en receptores OFDM modernos.

Metodología

En la Figura 1 se muestra el diagrama de flujo la metodología utilizada en este trabajo. Por principios de cuentas, el algoritmo de ordenamiento se codifica en lenguaje de programación C como una función que recibe de parámetro un arreglo desordenado de valores de tamaño N . Para efectos del diseño del *testbench*, los valores del arreglo se generan aleatoriamente y se introducen al algoritmo para corroborar su correcto funcionamiento. Hecho lo anterior, se procede a la traslación del código a su correspondiente arquitectura digital en FPGA con la ayuda de Vivado HLS (Xilinx, 2003) y se monitorean los resultados de métricas que definen la eficiencia de síntesis. Posteriormente se realiza una verificación a nivel funcional y se documentan los hallazgos junto a todo lo anterior para su difusión.

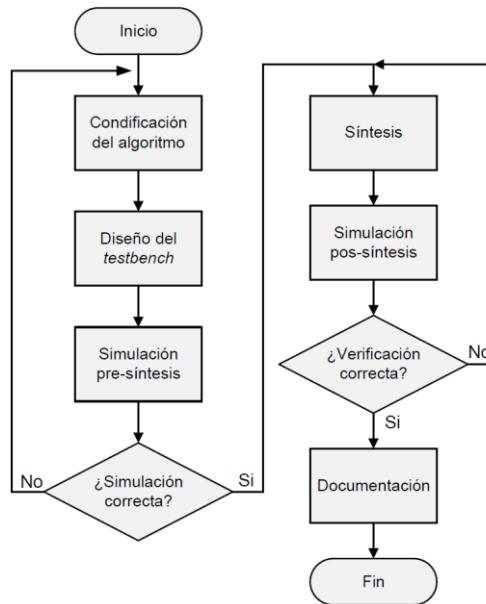


Figura 1: Flujograma de la metodología empleada.

Resultados y discusión

Cada una de las arquitecturas de *hardware* obtenidas por medio del diseño HLS, fueron sometidas a diversos *testbenches*, antes y después de su síntesis en la tarjeta de

desarrollo Nexys-4 que tiene un FPGA Artix-7 modelo xc7a100tcs324 de Xilinx. Los valores correspondientes recabados se pueden englobar en resultados de: síntesis, rendimiento y funcionales.

Resultados de síntesis

En la Figura 2 se muestra (a manera de ejemplo) el módulo de *hardware* correspondiente a la síntesis del algoritmo de inserción y su correspondiente arquitectura RTL. Se observa los puertos de entrada y salida del bloque de *hardware* generado por Vivado HLS.

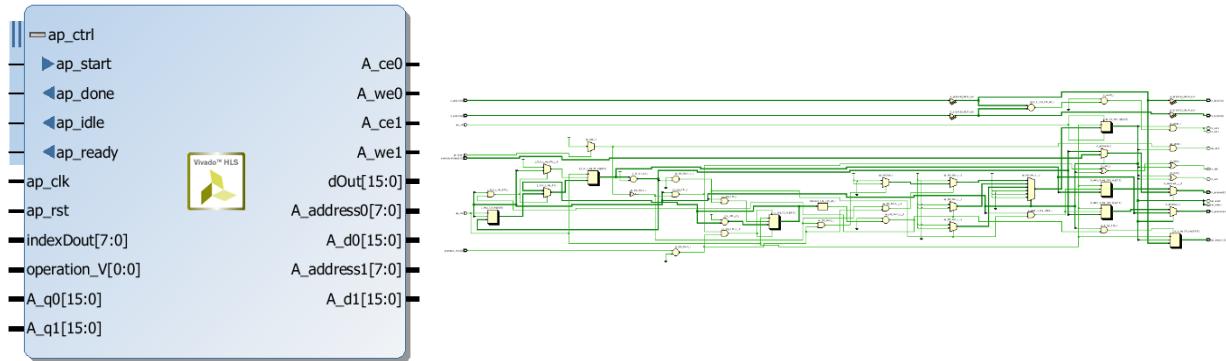


Figura 2: Bloque y RTL del algoritmo de inserción.

En la Figura 3 se presentan de forma resumida, los resultados del consumo de *hardware* de cada algoritmo para cantidades de elementos a ordenar coincidente con los tamaños de constelación QAM ($N=4, 16, 64$ y 256) definidas en el detector. Todas las arquitecturas de los algoritmos presentan consumos reducidos de *Flip-Flops* (FFs) y LUTs (*Lookup Tables*) en sus síntesis y ninguno de ellos requiere bloques dedicados de RAM o DSPs.

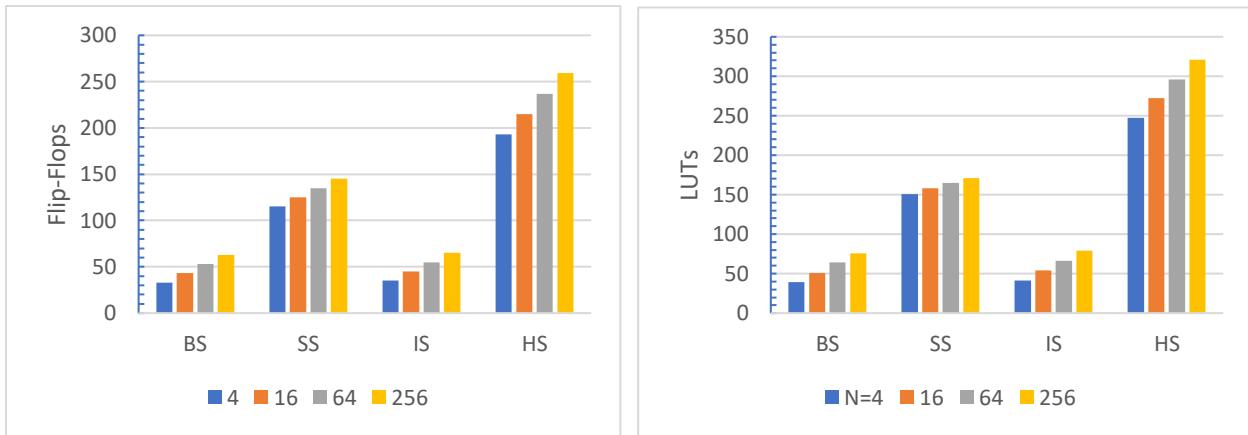


Figura 3: Resultados de síntesis; consumo de FFs (izquierda) y LUTs (derecha).

Resultados de rendimiento

En la Figura 4 se presenta el desempeño en términos de latencia (izquierda) y frecuencia de operación (derecha). La latencia máxima representa los ciclos de reloj necesarios para lograr ordenar los N elementos. Como puede notarse, la menor latencia la presenta el hardware del algoritmo *bubble sort* y la peor el del *heap sort*.

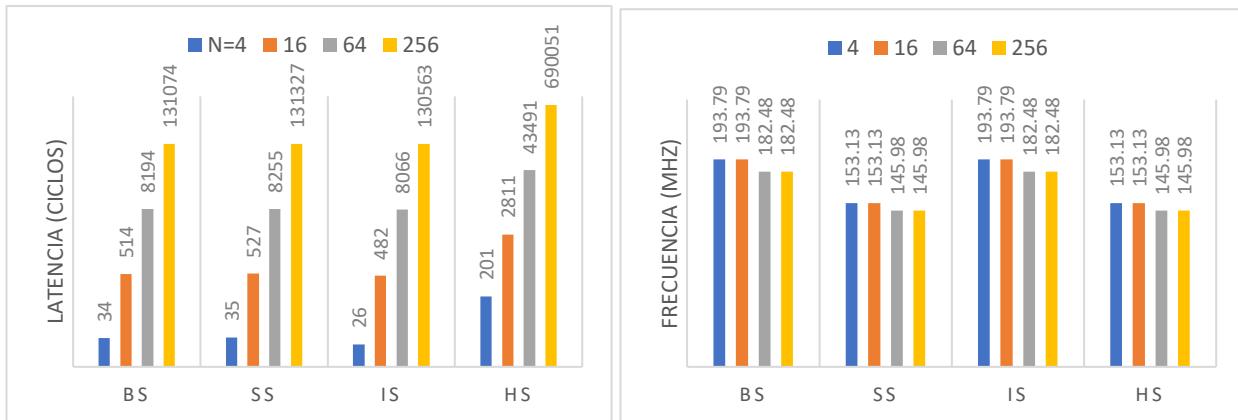


Figura 4: Resultados de rendimiento; latencia (izq.) y freq. max. de operación (derecha).

En lo que respecta a la velocidad, la Figura 4 (derecha) exhibe el comportamiento de la frecuencia máxima de operación de las arquitecturas sintetizadas para los valores de N ya mencionados. La arquitectura más veloz fue la del algoritmo de inserción con una $f_{max} = 193$ MHz. El peor caso se tiene para la del *heap-sort* con una f_{max} de 149 MHz.

Pruebas funcionales

En la Figura 5 se presenta el diagrama de tiempos de operación de la arquitectura sintetizada del algoritmo de inserción. La ejecución empieza cuando se activa la señal *ap_start*, en ese instante el *hardware* inicia con el proceso de ordenamiento generando direcciones para leer o escribir en la memoria externa. Después de una cantidad de ciclos de reloj la señal de salida *ap_done*, indica que los elementos están ordenados y listos para su lectura,

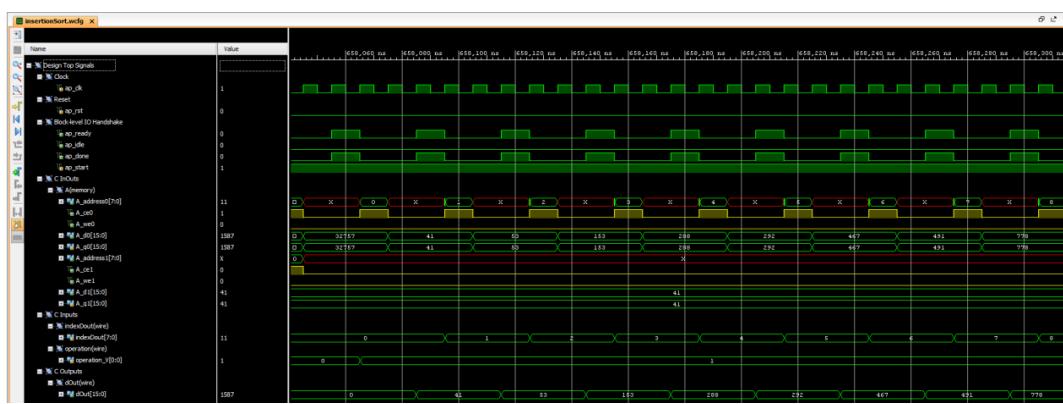


Figura 5: Cronograma de operación de la arquitectura del algoritmo de inserción.

Conclusiones

Uno de los procesos más importantes en el área de telecomunicaciones es el ordenamiento de datos, por ello, la correcta implementación en hardware impacta de manera significativa en el desempeño del detector y a su vez en el receptor. En este trabajo, el diseño HLS hizo posible el prototipado rápido en *hardware* de los algoritmos de ordenamiento más empleados actualmente. Logrando en ellos, una uniformidad en el consumo reducido de *hardware*, bajas latencias y velocidad elevadas, no obstante, el incremento de la cantidad de datos a ordenar. Así, con base a lo anterior se comprobó que, la arquitectura obtenida para el algoritmo de inserción se perfila como la opción con mayor viabilidad para ser incorporada en el diseño de sistemas receptores OFDM de última generación con detección *Near-ML*. Además de la flexibilidad adicional de las arquitecturas que les permite operar en modo *stand-alone* o como un coprocesador.

Agradecimientos

Este proyecto se llevó a cabo por el bloque de sistemas digitales del ITSON participante del proyecto Jalisco On-Chip (JoC).

Referencias

- CORREGIR*
- panor tu!*
cavo estu'
en el portico 10
- [1] S. Kulkarni, A. Darekar, and S. Shirol, "Proposed framework for V2V communication using Li-Fi technology," in *2017 International Conference on Circuits, Controls, and Communications (CCUBE)*, Bangalore, Dec. 2017, pp. 187–190. doi: 10.1109/CCUBE.2017.8394163.
 - [2] K. Abboud, H. A. Omar, and W. Zhuang, "Interworking of DSRC and Cellular Network Technologies for V2X Communications: A Survey," *IEEE Trans. Veh. Technol.*, vol. 65, no. 12, pp. 9457–9470, Dec. 2016, doi: 10.1109/TVT.2016.2591558.
 - [3] J. Lianghai, A. Weinand, B. Han, and H. D. Schotten, "Multi-RATs support to improve V2X communication," in *2018 IEEE Wireless Communications and Networking Conference (WCNC)*, Barcelona, Apr. 2018, pp. 1–6. doi: 10.1109/WCNC.2018.8377432.
 - [4] S. Gyawali, S. Xu, Y. Qian, and R. Q. Hu, "Challenges and Solutions for Cellular Based V2X Communications," *IEEE Commun. Surv. Tutorials*, vol. 23, no. 1, pp. 222–255, 2021, doi: 10.1109/COMST.2020.3029723.
 - [5] "The details of V2X communication," presented at the <https://www.epdtonthenet.net/article/178093/Under-the-hood-of-v2x-communications.aspx>.
 - [6] K. Eshteiwi, K. Ben Fredj, G. Kaddoum, and F. Gagnon, "Performance analysis of peer-to-peer V2V wireless communications in the presence of interference," in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, Montreal, QC, Oct. 2017, pp. 1–6. doi: 10.1109/PIMRC.2017.8292202.
 - [7] D. Boehmlaender, S. Hasirlioglu, V. Yano, C. Lauerer, T. Brandmeier, and A. Zimmer, "Advantages in Crash Severity Prediction Using Vehicle to Vehicle Communication," in *2015 IEEE International Conference on Dependable Systems and Networks Workshops*, Rio de Janeiro, Brazil, Jun. 2015, pp. 112–117. doi: 10.1109/DSN-W.2015.23.
 - [8] T. Bey and G. Tewolde, "Evaluation of DSRC and LTE for V2X," in *2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA, Jan. 2019, pp. 1032–1035. doi: 10.1109/CCWC.2019.8666563.
 - [9] H. P. Dai Nguyen and R. Zoltan, "The Current Security Challenges of Vehicle Communication in the Future Transportation System," in *2018 IEEE 16th International Symposium on Intelligent Systems and Informatics (SISY)*, Subotica, Sep. 2018, pp. 000161–000166. doi: 10.1109/SISY.2018.8524773.
 - [10] "vehicle_to_vehicle_communications_readiness_of_V2V_technology_for_application."
 - [11] R. Sabouni and R. M. Hafez, "Performance of DSRC for V2V communications in urban and highway environments," in *2012 25th IEEE Canadian Conference on*

Electrical and Computer Engineering (CCECE), Montreal, QC, Apr. 2012, pp. 1–5. doi: 10.1109/CCECE.2012.6335027.

[12] “V2V for Vehicular Safety Applications.”

[13] Z. MacHardy, A. Khan, K. Obama, and S. Iwashina, “V2X Access Technologies: Regulation, Research, and Remaining Challenges,” *IEEE Commun. Surv. Tutorials*, vol. 20, no. 3, pp. 1858–1877, 2018, doi: 10.1109/COMST.2018.2808444.

[14] Md. S. Hossen, M.-S. Bang, Y. Park, and K.-D. Kim, “Performance analysis of an OFDM-based method for V2X communication,” in *2014 Sixth International Conference on Ubiquitous and Future Networks (ICUFN)*, Shanghai, China, Jul. 2014, pp. 238–242. doi: 10.1109/ICUFN.2014.6876789.

[15] T. Maehata *et al.*, “DSRC using OFDM for roadside-vehicle communication system,” in *VTC2000-Spring. 2000 IEEE 51st Vehicular Technology Conference Proceedings (Cat. No.00CH37026)*, Tokyo, Japan, 2000, vol. 1, pp. 148–152. doi: 10.1109/VETECS.2000.851435.

[16]

“near_ML_detection_using_dikstras_algorithm_with_bounded_list_size_over_MI_MO_channels.”

[17] V. Kiray, S. Demir, and M. Zhaparov, “Improving Digital Electronics Education with FPGA technology, PBL and Micro Learning methods,” in *Proceedings of 2013 IEEE International Conference on Teaching, Assessment and Learning for Engineering (TALE)*, Bali, Indonesia, Aug. 2013, pp. 445–448. doi: 10.1109/TALE.2013.6654479.

[18] I. Kuon, R. Tessier, and J. Rose, “FPGA Architecture: Survey and Challenges,” *FNT in Electronic Design Automation*, vol. 2, no. 2, pp. 135–253, 2007, doi: 10.1561/1000000005.

[19] K. Georgopoulos *et al.*, “An evaluation of vivado HLS for efficient system design,” in *2016 International Symposium ELMAR*, Zadar, Croatia, Sep. 2016, pp. 195–199. doi: 10.1109/ELMAR.2016.7731785.

[20] F. Callaly, D. O’Loughlin, D. Lyons, A. Coffey, and F. Morgan, “Xilinx Vivado High Level Synthesis: Case studies,” in *25th IET Irish Signals & Systems Conference 2014 and 2014 China-Ireland International Conference on Information and Communications Technologies (ISSC 2014/CIICT 2014)*, Limerick, Ireland, 2014, pp. 352–356. doi: 10.1049/cp.2014.0713.

[21] “ug871-vivado-high-level-synthesis-tutorial.”

[22] “wp416-Vivado-Design-Suite.”

[23] J. A. Del Puerto-Flores, J. Cortez, C. A. Gutierrez, C. Del Valle-Soto, R. Velazquez, and L. J. Valdivia, “Performance of MRC Detection in OFDM System with Virtual Carriers over V2V Channels,” in *2019 IEEE 39th Central America and Panama Convention (CONCAPAN XXXIX)*, Guatemala City, Guatemala, Nov. 2019, pp. 1–6. doi: 10.1109/CONCAPANXXXIX47272.2019.8976929.

[24] Hector Eduardo Aldrete Vidrio, “Sistema de comunicación multiportadora para el estándar 802.11p utilizando precodificación frecuencial y cancelación no lineal de interferencia,” 2019.

[25] “IEEE draft standard for information technology.”

[26] T. Zemen, L. Bernado, N. Czink, and A. F. Molisch, “Iterative Time-Variant Channel Estimation for 802.11p Using Generalized Discrete Prolate Spheroidal Sequences,” *IEEE Trans. Veh. Technol.*, vol. 61, no. 3, pp. 1222–1233, Mar. 2012, doi:

10.1109/TVT.2012.2185526.

- [27] X. Cheng, C.-X. Wang, D. Laurenson, S. Salous, and A. Vasilakos, "An adaptive geometry-based stochastic model for non-isotropic MIMO mobile-to-mobile channels," *IEEE Trans. Wireless Commun.*, vol. 8, no. 9, pp. 4824–4835, Sep. 2009, doi: 10.1109/TWC.2009.081560.
- [28] Cheng-xiang Wang, Xiang Cheng, and D. Laurenson, "Vehicle-to-vehicle channel modeling and measurements: recent advances and future challenges," *IEEE Commun. Mag.*, vol. 47, no. 11, pp. 96–103, Nov. 2009, doi: 10.1109/MCOM.2009.5307472.
- [29] X. Cheng *et al.*, "An Improved Parameter Computation Method for a MIMO V2V Rayleigh Fading Channel Simulator Under Non-Isotropic Scattering Environments," *IEEE Commun. Lett.*, vol. 17, no. 2, pp. 265–268, Feb. 2013, doi: 10.1109/LCOMM.2013.011113.121535.
- [30] X. Cheng *et al.*, "Cooperative MIMO Channel Modeling and Multi-Link Spatial Correlation Properties," *IEEE J. Select. Areas Commun.*, vol. 30, no. 2, pp. 388–396, Feb. 2012, doi: 10.1109/JSAC.2012.120218.
- [31] I. Sen and D. W. Matolak, "Vehicle–Vehicle Channel Models for the 5-GHz Band," *IEEE Trans. Intell. Transport. Syst.*, vol. 9, no. 2, pp. 235–245, Jun. 2008, doi: 10.1109/TITS.2008.922881.
- [32] G. Acosta-Marum and M. A. Ingram, "Six time- and frequency-selective empirical channel models for vehicular wireless LANs," *IEEE Veh. Technol. Mag.*, vol. 2, no. 4, pp. 4–11, Dec. 2007, doi: 10.1109/MVT.2008.917435.
- [33] 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 J. Select. Areas Commun.*, vol. 31, no. 9, pp. 434–448, Sep. 2013, doi: 10.1109/JSAC.2013.SUP.0513039.
- [34] F. Pena-Campos, R. Carrasco-Alvarez, O. Longoria-Gandara, and R. Parra-Michel, "Estimation of Fast Time-Varying Channels in OFDM Systems Using Two-Dimensional Prolate," *IEEE Trans. Wireless Commun.*, vol. 12, no. 2, pp. 898–907, Feb. 2013, doi: 10.1109/TWC.2013.010413.120624.
- [35] M. O. Damen, H. El Gamal, and G. Caire, "On maximum-likelihood detection and the search for the closest lattice point," *IEEE Trans. Inform. Theory*, vol. 49, no. 10, pp. 2389–2402, 2003.
- [36] D. Wübben, R. Böhnke, J. Rinas, V. Kühn, and K. D. Kammeyer, "Efficient algorithm for decoding layered space-time codes," *Electron. Lett.*, vol. 37, no. 22, p. 1348, 2001, doi: 10.1049/el:20010899.
- [37] B. Hassibi and H. Vikalo, "On the sphere-decoding algorithm I. Expected complexity," *IEEE Trans. Signal Process.*, vol. 53, no. 8, pp. 2806–2818, 2005.
- [38] H. Moroga, T. Yamamoto, and F. Adachi, "Iterative Overlap TD-QRM-ML Block Signal Detection for Single-Carrier Transmission without CP Insertion," in *2012 IEEE Vehicular Technology Conference (VTC Fall)*, 2012, pp. 1–5.
- [39] K. Temma, T. Yamamoto, and F. Adachi, "Improved 2-Step QRM-ML Block Signal Detection for Single-Carrier Transmission," in *2011 IEEE Vehicular Technology Conference (VTC Fall)*, 2011, pp. 1–5.
- [40] "Arquitecturas digitales de procesamiento de señales para sistemas de comunicacion con entrenamiento."
- [41] A. Farmahini-Farahani, H. J. Duwe, M. J. Schulte, and K. Compton, "Modular

Design of High-Throughput, Low-Latency Sorting Units," *IEEE Trans. Comput.*, vol. 62, no. 7, pp. 1389–1402, Jul. 2013.

[42] D. Chen, J. Cong, and P. Pan, "FPGA Design Automation: A Survey," *FNT in Electronic Design Automation*, vol. 1, no. 3, pp. 195–334, 2006.

[43] I. Skliarova and V. Sklyarov, *FPGA-BASED Hardware Accelerators*, vol. 566. Cham: Springer International Publishing, 2019.

[44] Y.-K. Choi, Y. Chi, J. Wang, and J. Cong, "FLASH: Fast, Parallel, and Accurate Simulator for HLS," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Dec. 2020, vol. 39, pp. 4828–4841.

[45] M. M. Kamruzzaman, "Performance of Turbo coded wireless link for SISOOFDM, SIMO-OFDM, MISO-OFDM and MIMO-OFDM system," in *14th International Conference on Computer and Information Technology (ICCIT 2011)*, 2011, p. 6.

[46] I. Baig and V. Jeoti, "A new ZCT precoded OFDM system with pulse shaping: PAPR analysis," in *2010 IEEE Asia Pacific Conference on Circuits and Systems*, 2010, pp. 1131–1134.

Xilinx (2015), *Vivado Design Suite User Guide v2015.1*, Xilinx Inc, 2015.