

UNIVERSIDAD DE MÁLAGA

ESCUELA TÉCNICA SUPERIOR DE  
INGENIERÍA DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESARROLLO DE UN SISTEMA DE  
COMUNICACIONES VVLC CON  
IMPLEMENTACIÓN DE FILTRO  
ADAPTADO EN SOC FPGA

GRADO EN INGENIERÍA DE  
SISTEMAS ELECTRÓNICOS

Andrés Casasola Domínguez  
Málaga, 2020



E.T.S. DE INGENIERÍA DE TELECOMUNICACIÓN  
UNIVERSIDAD DE MÁLAGA

# DESARROLLO DE UN SISTEMA DE COMUNICACIONES VVLC CON IMPLEMENTACIÓN DE FILTRO ADAPTADO EN SOC FPGA

**Autor:** Andrés Casasola Domínguez

**Tutor:** Antonio García Zambrana

**Departamento:** Ingeniería de Comunicaciones (IC)

**Titulación:** Grado en Ingeniería de Sistemas Electrónicos

**Palabras clave:** FPGA, filtro adaptado, VLC, VVLC, Zynq, Red Pitaya, comunicaciones, DSP.

## Resumen:

En este proyecto se van a estudiar las herramientas proporcionadas por los dispositivos FPGA con la finalidad de diseñar e implementar un sistema de comunicación VVLC utilizando esta tecnología. Se va a estudiar e implementar un filtro adaptado con el objetivo de aumentar las capacidades del sistema de comunicación en cuanto a alcance y directividad. Finalmente se van a medir las capacidades del filtro adaptado implementado, así como el incremento de las capacidades del sistema de comunicación completo tras la integración del filtro adaptado.

Málaga, 19 de febrero de 2020



E.T.S. DE INGENIERÍA DE TELECOMUNICACIÓN  
UNIVERSIDAD DE MÁLAGA

# DEVELOPMENT OF A VVLC COMMUNICATIONS SYSTEM WITH MATCHED FILTER IMPLEMENTATION IN SOC FPGA

**Autor:** Andrés Casasola Domínguez

**Tutor:** Antonio García Zambrana

**Department:** Ingeniería de Comunicaciones (IC)

**Degree:** Grado en Ingeniería de Sistemas Electrónicos

**Key words:** FPGA, matched filter, VLC, VVLC, Zynq, Red Pitaya, communications, DSP.

## **Abstract:**

This project will study the different tools and capabilities provided by FPGA devices to design and implement a VVLC communication system based on this technology. A matched filter will be studied and implemented in order to increase the capabilities of the communication system in terms of scope and directivity. Finally, the capacities of the implemented matched filter will be measured, as well as the increase in the capabilities of the complete communication system after the integration of the matched filter.

Málaga, 19 de febrero de 2020



# Acrónimos

<b>ETSIT</b>	Escuela Técnica Superior de Ingeniería de Telecomunicación
<b>UMA</b>	Universidad de Málaga
<b>TFG</b>	Trabajo Fin de grado
<b>FPGA</b>	Field Programmable Gate Array
<b>V2V</b>	Vehicle to Vehicle
<b>V2I</b>	Vehicle to Infrastructure
<b>GPS</b>	Global Positioning System
<b>VLP</b>	Visible Light Positioning
<b>VLC</b>	Visible Light Communications
<b>DSRC</b>	Dedicated Short-Range Communications
<b>OOK</b>	On-Off Keying
<b>WHO</b>	World Health Organization
<b>SoC</b>	System On Chip
<b>RAM</b>	Random Access Memory
<b>IOB</b>	Input Output Block
<b>CLB</b>	Configurable Logic Block
<b>HDL</b>	Hardware Description Language
<b>PL</b>	Programmable Logic
<b>PS</b>	Processing System

<b>DLL</b>	Delay Locked Loop
<b>IDE</b>	Integrated Development Environment
<b>LED</b>	Light Emitting Diode
<b>VPPM</b>	Variable Pulse Position Modulation
<b>ADC</b>	Analog to Digital Converter
<b>DAC</b>	Digital to Analog Converter
<b>BRAM</b>	Block RAM
<b>DSP</b>	Digital Signal Processor
<b>ALU</b>	Unidad Aritmético Lógica
<b>MMCM</b>	Mixed-Mode Clock Manager
<b>PLL</b>	Phase Locked Loop
<b>DCM</b>	Digital Clock Manager
<b>CMT</b>	Clock Management Tile
<b>AWGN</b>	Additive White Gaussian Noise
<b>HDL</b>	Hardware Description Language
<b>PFD</b>	Phase Frequency Detector
<b>CP</b>	Charge Pump
<b>LF</b>	Loop Filter
<b>VCO</b>	Voltage Controlled Oscillator
<b>SSH</b>	Secure Shell
<b>RMS</b>	Root Mean Square

# Índice

<b>Acrónimos</b>	<b>vii</b>
<b>I Introducción</b>	<b>1</b>
<b>Introducción y visión general</b>	<b>3</b>
Objetivo . . . . .	3
Tecnologías de seguridad en automoción . . . . .	3
Estructura del documento . . . . .	8
<b>II Desarrollo del proyecto</b>	<b>9</b>
<b>1 Introducción tecnológica</b>	<b>11</b>
1.1 Plataforma hardware . . . . .	11
1.1.1 Conceptos generales . . . . .	11
1.1.2 Bloques DSP . . . . .	14
1.1.3 Bloques CMT . . . . .	18
1.1.4 Red Pitaya . . . . .	20
1.2 Plataforma software . . . . .	22
<b>2 Sistema de Comunicación</b>	<b>23</b>
2.1 Estándar IEEE 802.15.7 . . . . .	24
2.2 Transceptores ópticos . . . . .	25
2.3 Sistema de comunicación actual . . . . .	25
2.4 Mejoras sobre el sistema actual . . . . .	26
<b>3 Estudio del Filtro Adaptado</b>	<b>29</b>
3.1 Fundamento teórico del filtro adaptado . . . . .	29
3.2 Representación del filtro adaptado . . . . .	31
3.3 Recuperación de sincronización . . . . .	33
3.3.1 Early-late gate algorithm . . . . .	34

3.3.2	Gardner algorithm . . . . .	37
3.4	Comparativa en simulación de los algoritmos de sincronización . . . . .	39
<b>4</b>	<b>Implementación del Filtro Adaptado</b>	<b>45</b>
4.1	Sistema general . . . . .	46
4.2	Bloque integrador . . . . .	46
4.2.1	Acumulador . . . . .	47
4.2.2	Divisor . . . . .	48
4.2.3	Mapeo . . . . .	50
4.2.4	Simulación del bloque integrador . . . . .	51
4.3	Bloque Sampleador . . . . .	52
4.3.1	Limiter . . . . .	53
4.3.2	Counter . . . . .	54
4.3.3	Comparador . . . . .	54
4.3.4	Simulación del bloque Sampleador . . . . .	55
4.4	Bloque Error . . . . .	58
4.4.1	Simulación del bloque Error . . . . .	59
4.5	Simulación del Sistema . . . . .	60
<b>III</b>	<b>Pruebas</b>	<b>65</b>
<b>5</b>	<b>Pruebas</b>	<b>67</b>
5.1	Recuperación de tren de pulsos . . . . .	67
5.1.1	Comparativa transmisor prototipo en placa y faro de vehículo	68
5.1.2	Alcance del faro de vehículo . . . . .	70
5.1.3	Directividad . . . . .	72
5.1.4	Resumen de resultados . . . . .	74
5.2	Integración del filtro adaptado en el sistema de comunicación . . . . .	76
<b>IV</b>	<b>Conclusiones</b>	<b>79</b>
	<b>Conclusiones y líneas futuras</b>	<b>81</b>
<b>V</b>	<b>Apéndices</b>	<b>83</b>
<b>A</b>		<b>85</b>
A.1	Inicialización Red Pitaya . . . . .	85
A.1.1	Configuración de Vivado . . . . .	85
A.1.2	Configuración de Linux en ARM . . . . .	86

A.1.3	Programación de la FPGA . . . . .	88
A.1.4	Puertos de la Red Pitaya . . . . .	89
A.2	IP Cores de Xilinx . . . . .	91
A.2.1	Processing System . . . . .	91
A.2.2	Clock Wizard . . . . .	91
A.3	Periféricos de la placa . . . . .	92
A.3.1	Conversor Analógico a Digital (ADC) . . . . .	92
A.3.2	Conversor Digital a Analógico (DAC) . . . . .	92



# Índice de figuras

1	Número de fallecidos en accidentes de tráfico en vías interurbanas y urbanas por año en España. Fuente de datos: DGT . . . . .	5
2	Parque de vehículos por año en España. Fuente de datos: DGT . . . . .	6
3	Localización de emisores y receptores en un vehículo para VLC. Fuente: <i>Smart automotive lighting for vehicle safety, IEEE Communications Magazine</i> [1]. . . . .	7
1.1	Diagrama de bloques básico de FPGA Spartan-II. Fuente: Xilinx . . . . .	12
1.2	Diagrama simplificado del SoC Zynq 7010. . . . .	13
1.3	Diagrama del SoC Zynq-7010. Fuente: Xilinx . . . . .	14
1.4	Diagrama simple del slice DSP48E1. Fuente: Xilinx . . . . .	15
1.5	Diagrama avanzado del slice DSP48E1. Fuente: Xilinx . . . . .	16
1.6	Diagrama avanzado del slice DSP48. Fuente: Xilinx . . . . .	17
1.7	Diagrama de bloques de un CMT. Fuente: Xilinx . . . . .	18
1.8	Diagrama de bloques de un módulo MMCM y PLL. Fuente: Xilinx . .	19
1.9	Esquemático <i>Red Pitaya</i> . . . . .	20
1.10	Placa <i>Red Pitaya</i> . Fuente: Red Pitaya. . . . .	21
2.1	Modos de operación de OOK de la capa PHY I del estándar IEEE 802.15.7. Fuente: <i>Impact of IEEE 802.15.7 standard on visible light communications usage in automotive applications, IEEE Communications Magazine</i> [2]. . . . .	25
2.2	Diagrama del sistema de comunicación, sin filtro adaptado. . . . .	27
2.3	Diagrama del sistema de comunicación, con filtro adaptado. . . . .	28
3.1	Diagrama de bloques del filtro adaptado. . . . .	32
3.2	Señal de transmisión supuesta para todas las simulaciones. . . . .	32
3.3	Representación gráfica del comportamiento de cada bloque del filtro adaptado para una SNR en recepción de 25dB. . . . .	33
3.4	Representación gráfica del comportamiento de cada bloque del filtro adaptado para una SNR en recepción de 0dB. . . . .	34

3.5	Representación gráfica del comportamiento de cada bloque del filtro adaptado para una SNR en recepción de -10dB. . . . .	35
3.6	Diferentes casos de muestreo con el algoritmo <i>Early-late</i> . . . . .	36
3.7	Diferentes casos de muestreo con el algoritmo <i>Gardner</i> . . . . .	38
3.8	Sincronización del algoritmo Early-Late para tren de pulsos con una SNR en recepción de 25dB. . . . .	39
3.9	Sincronización del algoritmo de Gardner para tren de pulsos con una SNR en recepción de 25dB. . . . .	41
3.10	Sincronización del algoritmo Early-Late para tren de pulsos y trama pseudoaleatoria con una SNR en recepción de 25dB. . . . .	42
3.11	Sincronización del algoritmo de Gardner para tren de pulsos y trama pseudoaleatoria con una SNR en recepción de 25dB. . . . .	43
4.1	Esquemático del módulo top. . . . .	47
4.2	Esquemático del módulo integrador. . . . .	48
4.3	Esquemático del bloque acumulador. . . . .	48
4.4	Esquemático del slice DSP equivalente al bloque divisor. . . . .	49
4.5	Esquemático del bloque mapeo. . . . .	51
4.6	Simulación del bloque integrador. . . . .	53
4.7	Esquemático del bloque Sampleador. . . . .	54
4.8	Esquemático del bloque limiter. . . . .	55
4.9	Esquemático del bloque counter. . . . .	56
4.10	Esquemático del bloque cmp1. . . . .	57
4.11	Esquemático del bloque cmp2. . . . .	58
4.12	Primera simulación del bloque sampleador. . . . .	58
4.13	Valor de cuenta en la activación de CE1 y CE3 para la primera simulación del bloque sampleador. . . . .	59
4.14	Valor de cuenta en la activación de CE2 para la primera simulación del bloque sampleador. . . . .	59
4.15	Segunda simulación del bloque sampleador. . . . .	60
4.16	Esquemático del bloque error. . . . .	61
4.17	Esquemático del slice DSP del bloque error. . . . .	61
4.18	Simulación del bloque error. . . . .	62
4.19	Simulación del sistema completo, etapa transitoria. . . . .	62
4.20	Simulación del sistema completo, etapa estable. . . . .	63
5.1	Configuración de los equipos para la ejecución de la prueba 1: Recuperación de tren de pulsos. . . . .	68
5.2	Dispositivos transmisores y receptor. . . . .	69
5.3	Señal en transmisión, común a todos los experimentos de esta prueba. . . . .	70

5.4	Señal en recepción (canal superior) y salida del filtro adaptado (canal inferior). Distancia de 2 metros y desapuntamiento de 0°. . . . .	71
5.5	Señal en recepción (canal superior) y salida del filtro adaptado (canal inferior). Distancia de 2 metros y desapuntamiento de 60°. . . . .	71
5.6	Señal en recepción (canal superior) y salida del filtro adaptado (canal inferior) para diferentes distancias entre transmisor y receptor con 0° de ángulo de desapuntamiento. . . . .	72
5.7	Señal en recepción (canal superior) y salida del filtro adaptado (canal inferior) para diferentes ángulos de desapuntamiento y 2 metros de distancia entre transmisor y receptor. . . . .	73
5.8	Señal de ruido capturada por el osciloscopio. . . . .	74
5.9	Configuración de los equipos para la ejecución de la prueba 2: Integración del filtro adaptado en el sistema de comunicación. . . . .	76
5.10	Terminal de la sesión SSH en el PC remoto desde el que se controla el ARM de la placa, transmitiendo y recibiendo correctamente el mensaje <i>Hello World</i> . . . . .	77
A.1	Ventana de selección de <i>Boards</i> en la creación de un proyecto en Vivado. . . . .	87
A.2	Dirección MAC de la Red Pitaya. . . . .	89
A.3	Parte del esquemático eléctrico de la interfaz entre FPGA y ADC [3]. . . . .	90
A.4	Core Processing System para el SoC Zynq-7000 series. . . . .	92
A.5	Core Clock Wizard por defecto de Xilinx. . . . .	93
A.6	Diagrama de bloques del integrado LTC2145-14 [4]. . . . .	94
A.7	Rango bipolar de los integrados LTC2145-14 y DAC1401D125. . . . .	94
A.8	Diagrama de bloques del integrado DAC1401D125 [5]. . . . .	95



# Parte I

## Introducción



# Introducción y visión general

## Contenido

---

Objetivo . . . . .	3
Tecnologías de seguridad en automoción . . . . .	3
Estructura del documento . . . . .	8

---

## Objetivo

El objetivo de este TFG es desarrollar un sistema de comunicaciones VLC, orientado a aplicaciones vehiculares de exterior, basado en FPGA y cumpliendo con las especificaciones del estándar IEEE 802.15.7 para VLC, haciendo uso de la capa PHY I del estándar.

## Tecnologías de seguridad en automoción

Resulta común en el día a día encontrarse con noticias de graves accidentes de tráfico, fallecidos en carretera y titulares similares en prensa y medios de comunicación. La cantidad de muertes por accidentes de tráfico en relación al número de vehículos en circulación era muy alta con las primeras generaciones de vehículos que se fabricaron. El número de fallecidos aun así era bajo, 2.288 fallecidos en 1960 [6], puesto que no todo el mundo disponía de vehículo propio. Conforme más personas iban adquiriendo vehículo, el número de fallecimientos aumentaba cada año, llegando a un máximo de 9.344 fallecidos en 1989 [6]. A medida que pasa el tiempo y aumenta la tecnología, se intensifican los estudios y avances sobre seguridad en vehículos, lo que empieza a disminuir esta cifra de fallecidos. Como se muestra en la figura 1, el número de fallecidos en 1995 alcanza los 5.751 [6]. En los años siguientes, este valor sigue disminuyendo y en 2016 alcanza un valor de 1.810 [6].

La tecnología juega un papel importante en este descenso y tiene sentido pensar que a mayor avance tecnológico, más mecanismos de seguridad actuarán en caso de accidente, evitando muertes. La probabilidad de que una máquina falle es inferior a la probabilidad de que lo haga una persona. Si en cualquier caso no fuese así, probablemente será a causa de un mal diseño de la máquina.

Por otro lado en la figura 2 se observa como el número de vehículos no deja de crecer cada año en España, partiendo con 17 millones de vehículos en 1993 y llegando a los 32 millones de vehículos en 2016 [6]. Con esta cifra de vehículos en aumento, cabe pensar en el desarrollo de tecnologías que faciliten y acomoden al ser humano su uso y mejore su seguridad.

Existen también otras ventajas a parte de la seguridad por las que resulta beneficioso mejorar este campo, como lo es por ejemplo una selección automática de una ruta para llegar a un destino. Tomando información en tiempo real desde internet los vehículos pueden organizarse para no colapsar ciertas vías, evitar vías en las que haya podido producirse un accidente o algún desastre natural. Las personas ancianas incapacitadas para conducir un vehículo podrían usarlo al igual que cualquier otra persona [7].

Son varias las alternativas tecnológicas que se proponen para avanzar en este campo: redes de comunicación vehiculares, vehículos inteligentes con capacidad de reconocimiento del entorno, etc, todas ellas con el objetivo de reducir el control que debe realizar el conductor, de forma que este sea llevado a cabo por máquinas.

En la actualidad, muchos artículos e investigaciones se enfocan en el desarrollo e investigación de sistemas de comunicación entre vehículos. Estas investigaciones estudian el comportamiento de algunas técnicas, determinando así los sistemas más eficientes en función del escenario en el que se encuentran y otras posibles variantes. Actualmente para la comunicación vehicular se tienen en cuenta dos principales flujos de comunicación: comunicación vehículo-vehículo o V2V y comunicación vehículo-infraestructura o V2I. Las ventajas que estas comunicaciones pueden dar a la conducción son varias, a continuación se muestran algunas de ellas [1] [8]:

- La información que un vehículo obtiene sobre sí mismo es más aproximada a la realidad que la que los vehículos cercanos podrán medir de él. En caso de difundir esta información, los vehículos cercanos conseguirán calcular y realizar acciones mucho más precisas.
- Si un vehículo mantiene una comunicación constante en la que transmite y recibe posiciones y velocidades con los vehículos cercanos, las posibles pérdidas de visión por parte del conductor dejan de ser un problema crítico.

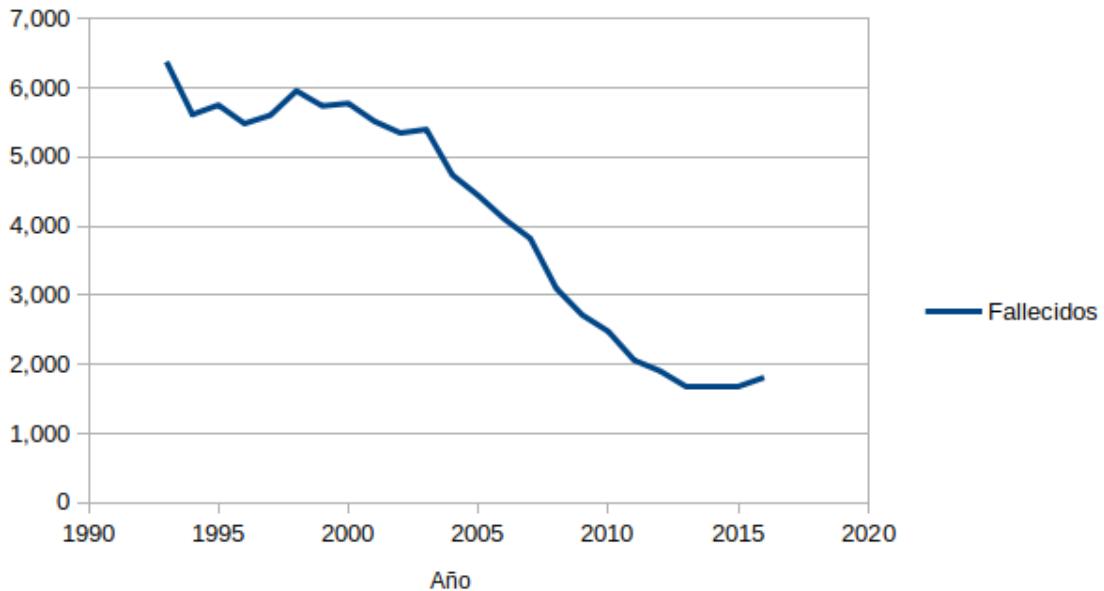


Figura 1: Número de fallecidos en accidentes de tráfico en vías interurbanas y urbanas por año en España. Fuente de datos: DGT

- Combinando las observaciones de varios vehículos cercanos se consigue reducir el error en las mediciones.
- Si existe una comunicación y un intercambio de información entre vehículos, la cantidad de sensores que cada vehículo deberá incorporar será menor, puesto que se aprovechará de los vehículos cercanos para adquirir información que no disponga. Esto resulta en un precio más bajo.

Para la implementación de estas comunicaciones existen varias opciones. La que parece ser mas prometedora es DSRC o *Dedicated Short-Range Communications*. Esta tecnología actualmente dispone de estándares desarrollados para la capa física y la capa de enlace de datos que utilizan el espectro de 5.9 GHz. A pesar de todo el desarrollo llevado a cabo sobre esta tecnología, aún no ha conseguido integrarse en el mercado y no se está seguro de si llegará a hacerlo [1].

Por otro lado las comunicaciones VLC resultan cada vez más atractivas a la hora de construir nuevas redes de comunicación, en este caso redes vehiculares. VLC es una tecnología moderna y muy poco utilizada por lo que el espacio que ésta ocupa está prácticamente libre. La gran ventaja de esta opción es que el interfaz físico necesario para transmitir y recibir información está casi integrado en los vehículos ya existentes. Para transmitir es necesario faros de tecnología LED, los cuales ya

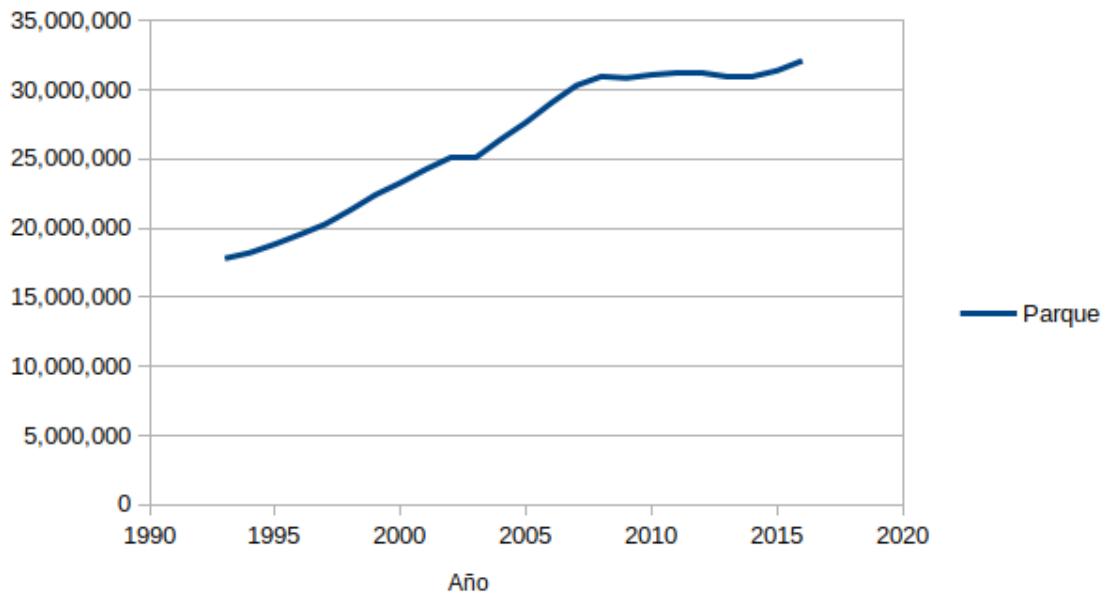


Figura 2: Parque de vehículos por año en España. Fuente de datos: DGT

son utilizados por los vehículos actuales. Para recibir información sería necesario incorporar sensores fotosensibles, los cuales tienen un bajo coste. En la figura 3 se representa un ejemplo de las posiciones donde se podrían integrar los emisores y receptores en un vehículo [1] [9] [8].

En comparativa, la tecnología VLC es mucho más barata que DSRC, y además, presenta multitud de ventajas en lo que respecta a aplicaciones de seguridad vehicular frente a otras comunicaciones basadas en radio frecuencia, como lo es DSRC.

Una ventaja ya mencionada es el menor grado de complejidad y coste, puesto que el diseño del interfaz emisor-receptor de un enlace por radiofrecuencia tendrá una complejidad y coste mayor. Además, como se ha dicho, los vehículos actuales ya disponen de la interfaz óptica transmisora estándar de la tecnología VLC [1].

En situaciones de alta densidad de tráfico, los sistemas V2V basados en radiofrecuencia, deberán controlar de algún modo el número de nodos con el que el vehículo se está comunicando puesto que, de no ser así, se correría el riesgo de procesar mucha información innecesaria. Este sistema, encargado de controlar la comunicación con los nodos cercanos, podría provocar una sobrecarga en el equipo dando lugar a una fiabilidad inferior en las comunicaciones. Los sistemas basados en VLC, sin embargo, carecen de esta desventaja, puesto que solo se mantendrá comunicación

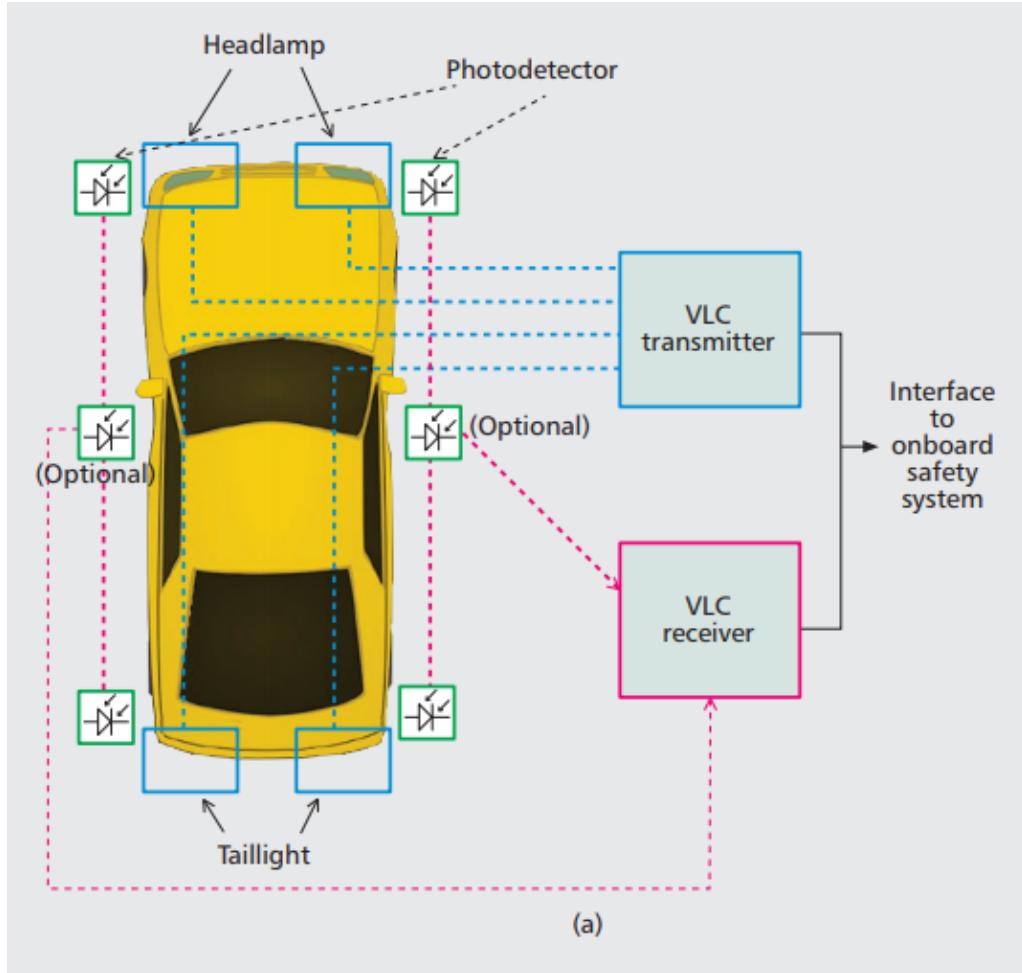


Figura 3: Localización de emisores y receptores en un vehículo para VLC. Fuente: *Smart automotive lighting for vehicle safety, IEEE Communications Magazine* [1].

con nodos con los que la interfaz óptica receptora mantenga visión directa [1].

Respecto a la precisión de posicionamiento, los sistemas implementados con tecnología GPS tienen una precisión máxima de 10 metros de exactitud, lo que supone un valor excesivo para un posicionamiento entre vehículos. El error de VLP o *Visible Light Positioning*, en cambio, es del orden de decenas de centímetros [1].

La tecnología DSRC, puesto que utiliza radiofrecuencia, está clasificada por la organización WHO o *World Health Organization* como una posible causa de cáncer en seres humanos [2]. VLC resulta seguro para las personas a la vez que seguro para el resto de equipos electrónicos que pudieran estar funcionando en vehículos e

infraestructuras.

Tras esta comparativa, VLC parece una tecnología más precisa, segura, más barata y más sencilla. Por lo que parece preferible su elección frente a las tecnologías de radiofrecuencia. A pesar de ello, VLC propone nuevos retos y problemas a los que se debe hacer frente para que esta pueda llegar a ser una tecnología fiable y rentable.

Algunos de estos retos a los que se enfrenta VLC es el *dimming* u oscurecimiento. En OOK o *On-Off Keying* la intensidad de los niveles alto y bajo son fijos, por lo que resulta necesario añadir intervalos de tiempos de espera desde que se impone un cambio de nivel, hasta que el cambio termina de producirse. Esto es debido al tiempo de respuesta de los dispositivos LED, que como cabe esperar, no es ideal. Estos tiempos de espera, repercuten en la tasa de transmisión de datos o *throughput* de la comunicación.

Otro problema a tener en cuenta es el *flickering* o parpadeo. VLC utiliza la zona del espectro visible por el ojo humano, lo que significa que el parpadeo de un sistema mal diseñado, podría ser perceptible para una persona, afectando a su salud. El estándar define *flickering*, como la fluctuación de brillo capaz de producir cambios perceptibles en la fisiología del cuerpo humano, delimitando el máximo período de *flickering* en 5 ms, que representa el período de tiempo ante el cual el ojo humano no es capaz de apreciar los cambios de intensidad de la luz [2].

## Estructura del documento

La estructura de este documento se va a dividir en las siguientes secciones:

- **Introducción tecnológica.**
- **Sistema de comunicación.**
- **Estudio del filtro adaptado.**
- **Implementación del filtro adaptado.**
- **Pruebas.**
- **Conclusiones.**
- **Anexos.**

## Parte II

# Desarrollo del proyecto



# Capítulo 1

## Introducción tecnológica

### Contenido

---

<b>1.1</b>	<b>Plataforma hardware</b>	<b>11</b>
1.1.1	Conceptos generales	11
1.1.2	Bloques DSP	14
1.1.3	Bloques CMT	18
1.1.4	Red Pitaya	20
<b>1.2</b>	<b>Plataforma software</b>	<b>22</b>

---

Para este trabajo se ha decidido utilizar unas determinadas tecnologías y herramientas tanto hardware como software. A continuación, se van a mostrar cada una de las herramientas seleccionadas y el por qué de su elección.

### 1.1. Plataforma hardware

#### 1.1.1. Conceptos generales

En la parte hardware se van a utilizar dispositivos FPGA. Los dispositivos FPGA son chips que contienen bloques de lógica programable que se pueden interconectar según el programador lo deseñe, dando lugar a una configuración y rutado final de toda la lógica integrada en el chip, con la finalidad de implementar sistemas digitales, capaces de alcanzar una gran complejidad. Además de bloques de lógica programable de propósito general, existen diferentes bloques dedicados que se encargan de implementar funciones específicas, permitiendo una mayor eficiencia al estar ya

diseñados e integrados. A continuación, se mencionan algunos de los ejemplos más comunes:

- Bloques DCM o CMT: son bloques dedicados, especializados en la generación, distribución y control de las señales de reloj por todo el chip.
- Bloques RAM: son bloques dedicados que están compuestos por memoria RAM, se encuentran distribuidos por todo el chip y se pueden utilizar para almacenar, compartir datos, etc...
- Bloques IOB: son bloques dedicados que implementan la interfaz entre la lógica integrada en el chip y todos los periféricos externos.
- Bloques CLB: son bloques de propósito general, compuestos por bloques DRAM, capaces de configurarse como LUTs para implementar operaciones lógicas, como memoria RAM o como registros de desplazamientos, lógica específica para operaciones aritméticas, propagación eficiente de acarreo y *flip flops* de entrada/salida. Componen la lógica programable [10].

En la figura 1.1 se puede observar el esquemático de una FPGA Spartan-II. Se pueden ver los diferentes bloques mencionados anteriormente, en este caso los bloques DLL equivaldrían a los DCM anteriormente mencionados.

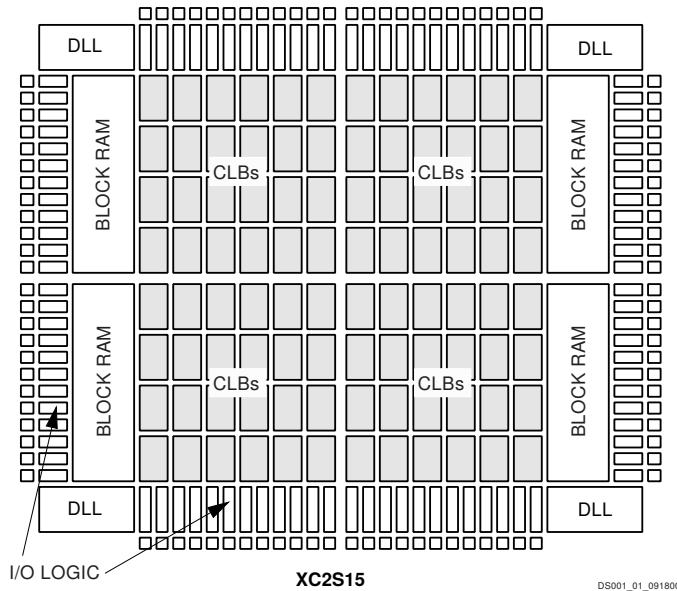


Figura 1.1: Diagrama de bloques básicos de FPGA Spartan-II. Fuente: Xilinx

Los lenguajes de programación que se utilizan para programar estos dispositivos se denominan HDL y actualmente los más conocidos son VHDL y Verilog.

Para este trabajo se va a hacer uso del *Zynq-7010* de Xilinx. Este SoC es mucho más avanzado que la FPGA Spartan-II. Si analizamos su arquitectura internamente se pueden diferenciar dos partes principales [11]:

- Sistema de procesamiento o PS, compuesto por un procesador ARM Cortex-A9 dual core con una frecuencia de funcionamiento máxima de 667 MHz [12].
- Lógica programable o PL, que integra CLBs, BRAM de 36Kb, bloques CTM, bloques DSP, interfaz JTAG y bloques de entrada/salida con niveles de tensión programables, entre otros [12].

En la figura 1.2 se puede ver un diagrama de bloques simplificado del *Zynq-7010*. En este diagrama aparecen los bloques PS y PL además de algunas de las interfaces de comunicación que se utilizarán en este trabajo.

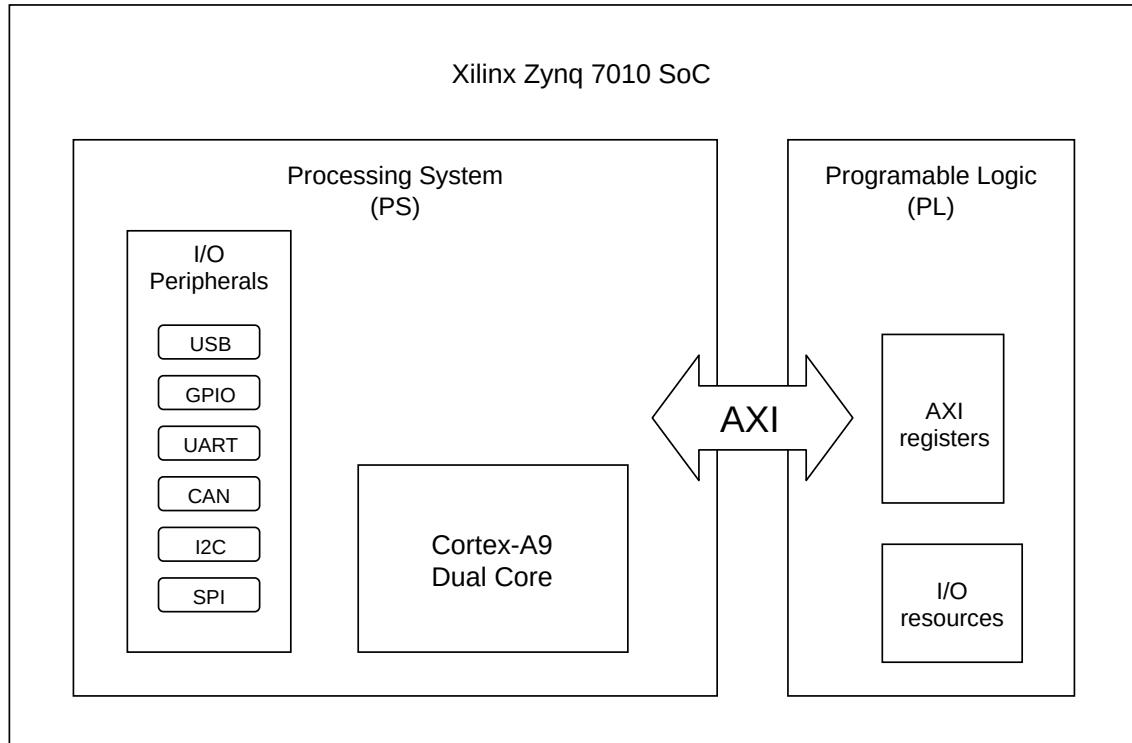


Figura 1.2: Diagrama simplificado del SoC Zynq 7010.

En la figura 1.3 se muestra el esquemático proporcionado por Xilinx. El esquemático es complejo y contiene una gran cantidad de interfaces de comunicación y

sistemas de procesamiento. Para este trabajo realmente se utilizará una pequeña parte de ellos.

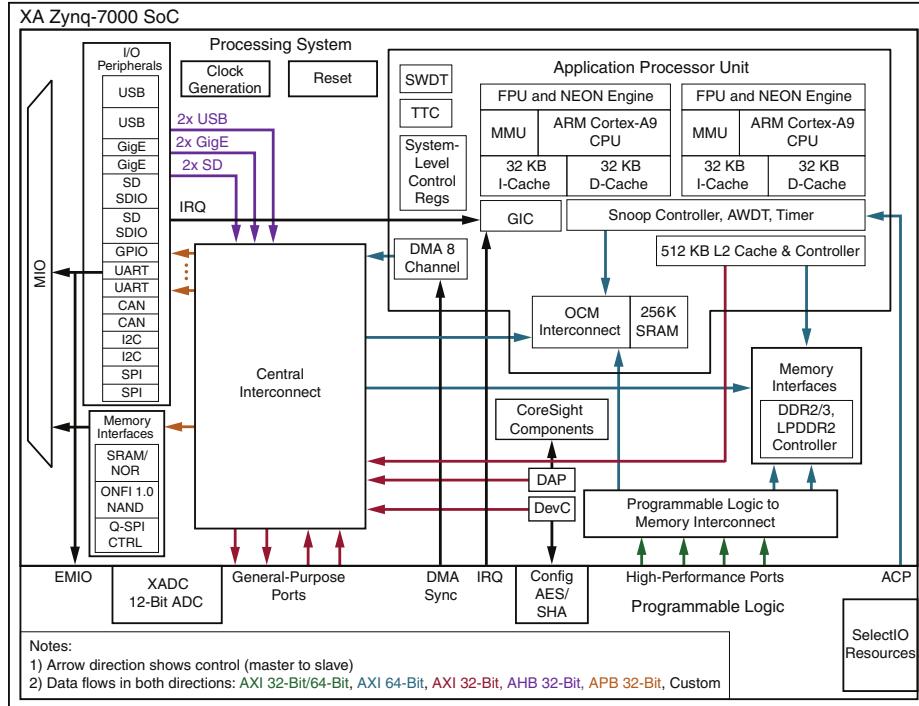


Figura 1.3: Diagrama del SoC Zynq-7010. Fuente: Xilinx

Para el sistema que se va a diseñar será importante el uso de los bloques CMT y DSP. Los bloques CMT serán necesarios para generar y distribuir de forma correcta las señales de reloj, y los bloques DSP para realizar operaciones de procesado de señal de forma eficiente y correcta. Estos dos bloques van a ser los más críticos a la hora de diseñar el sistema, por lo que se explicarán con mayor profundidad a continuación.

### 1.1.2. Bloques DSP

Los *slices* o bloques DSP48E1 de la Serie 7 de Xilinx están diseñados para implementar algoritmos de operaciones matemáticas muy concretas y configurables. Permiten alta eficiencia en cuanto a consumo y altas velocidades de funcionamiento [13].

En la figura 1.4 se muestra un diagrama de bloques simple de un slice DSP48E1. Se pueden diferenciar 4 datos de entrada, A, B, C y D. Los datos A y D van a un *Pre-adder* donde se pueden sumar o restar. A continuación, su salida va a un multiplicador de  $25 \times 18$  cuya otra entrada es el dato B. La salida del multiplicador y el dato C, multiplexado con la salida realimentada, van al módulo ALU donde se pueden realizar operaciones de suma, resta u operaciones bit a bit como XOR, AND, etc. Además existe un módulo comparador que compara la salida de la ALU con un patrón programado y activa o desactiva una señal binaria como resultado de dicha comparación.

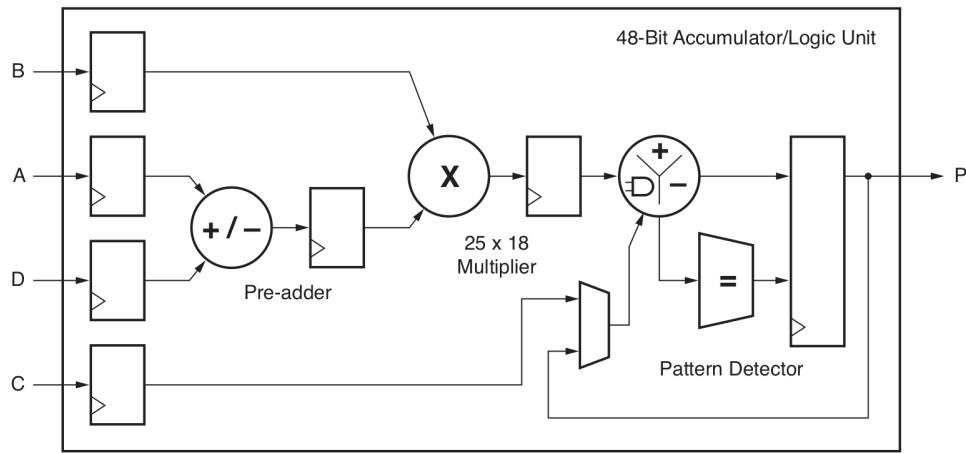


Figura 1.4: Diagrama simple del slice DSP48E1. Fuente: Xilinx

El esquemático previamente presentado es muy útil para entender, de forma conceptual, la capacidad que presentan estos bloques DSP. Realmente el esquemático circuital completo es más complejo y permite una gran versatilidad a la hora de diseñar algoritmos y operaciones concretas.

En la figura 1.5 se puede observar el esquemático avanzado del DSP48E1. En este esquemático aparecen una mayor cantidad de bloques. A continuación, se va a contrastar con el esquemático simplificado, visto anteriormente.

Sin prestar atención de momento a la configuración interna, se puede observar una gran cantidad de entradas y salidas que antes no aparecían. Todas las nuevas entradas y salidas marcadas con un asterisco son señales que, en caso de ser entradas, provienen de un slice DSP48E1 anterior o que, en caso de ser salidas, van al siguiente slice DSP48E1. Esto se debe a que los slices DSP están ubicados por el

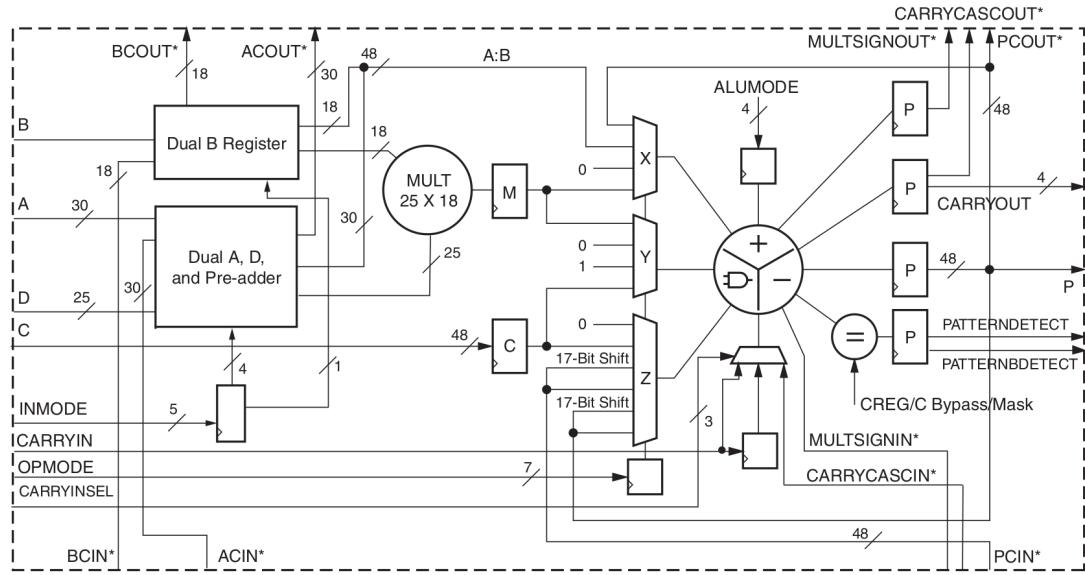


Figura 1.5: Diagrama avanzado del slice DSP48E1. Fuente: Xilinx

chip formando columnas para que la conexión de señales entre slices consecutivos sea mucho más eficiente y rápida, evitando el uso de lógica intermedia y recursos de interconexión. Además, la conexión en cascada de estas señales se implementa mediante un bus físico que no es configurable por el programador. Señales como PCIN, BCIN o ACIN son las señales PCOUT, BCOUT, ACOUT del anterior slice DSP48E1 utilizando la conexión en cascada.

Los bloques más importantes para configurar las operaciones que va a realizar un slice DSP48E1 se pueden dividir en tres etapas, son los presentados a continuación, y seguidos se muestran los registros de configuración de cada bloque.

- *Pre-adder* y multiplicador: Registro INMODE.
- Multiplexores X, Y, Z: Registro OPMODE.
- ALU: Registro ALUMODE.

La primera etapa está formada por el bloque *Dual B Register* que implementa un registro, o doble registro de forma opcional, de la entrada B. El bloque *Dual A, D and Pre-adder* que realiza una suma o resta entre las entradas A y D, implementando de nuevo, un doble registro opcional. Ambas salidas pueden ir al multiplicador en caso de utilizarse, o en caso de no utilizarse, se puede concatenar la salida del *Dual A, D and Pre-adder* con la salida de *Dual B Register*. Obteniendo así A:B,

que sería A concatenado con B. Nótese que en esta configuración la entrada D no tomaría parte. Más abajo aparece la entrada C únicamente registrada.

La segunda etapa son los multiplexores X, Y, Z. Estos multiplexores van a seleccionar las entradas que se deseen hacia la ALU. Si, por ejemplo, se desea utilizar A:B sin hacer uso del multiplicador se deberá configurar el multiplexor X para seleccionar A:B. O si se quiere utilizar la salida del multiplicador se deberán configurar los multiplexores tanto X como Y para seleccionar el registro M, en este caso el programador está obligado a seleccionar el registro M tanto para el multiplexor X como para el Y. Por el multiplexor Z, se puede seleccionar la entrada C, o por ejemplo se puede seleccionar la entrada P, de esta forma se realimenta la salida del propio slice DSP48E1, esta ultima opción es muy útil para implementar acumuladores. Si se selecciona la entrada PCIN se estaría operando con la salida del anterior DSP48E1, esto podría ser muy util para formar una estructura *pipeline*.

La tercera etapa está formada por la ALU, esta unidad tiene capacidad de realizar operaciones de suma, resta y operaciones bit a bit con las entradas proporcionadas por los multiplexores X, Y ,Z. Además, tiene la opción de implementar un detector de patrones. En la figura 1.6 se pueden ver las diferentes operaciones aritméticas que la ALU puede realizar en función del registro ALUMODE con las entradas provenientes de los multiplexores X, Y, Z.

<b>DSP Operation</b>	<b>OPMODE[6:0]</b>	<b>ALUMODE[3:0]</b>			
		<b>3</b>	<b>2</b>	<b>1</b>	<b>0</b>
Z + X + Y + CIN	Any legal OPMODE	0	0	0	0
Z - (X + Y + CIN)	Any legal OPMODE	0	0	1	1
-Z + (X + Y + CIN) - 1 = not (Z) + X + Y + CIN	Any legal OPMODE	0	0	0	1
not (Z + X + Y + CIN) = -Z - X - Y - CIN - 1	Any legal OPMODE	0	0	1	0

Figura 1.6: Diagrama avanzado del slice DSP48. Fuente: Xilinx

Para configurar estas 3 etapas se debe hacer uso de los registros INMODE, OPMODE y ALUMODE respectivamente, con el fin de implementar el algoritmo o parte de algoritmo que se deseé.

### 1.1.3. Bloques CMT

En las FPGA serie 7 de Xilinx los bloques CMT incluyen módulos MMCM y PLL. Realmente son muy similares, la diferencia es que los módulos PLL implementan un subconjunto de los MMCM [14] [15].

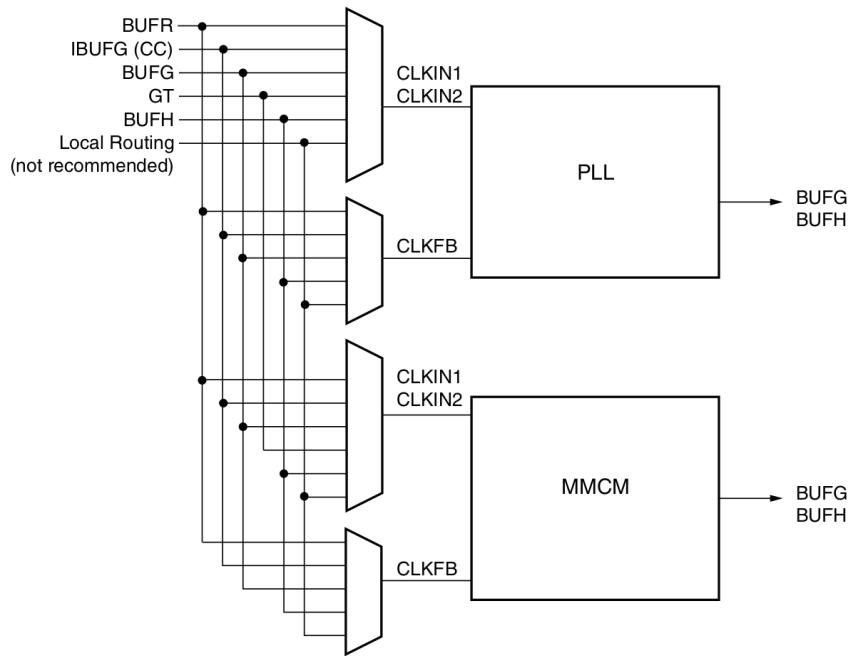


Figura 1.7: Diagrama de bloques de un CMT. Fuente: Xilinx

En la figura 1.7 se muestra un diagrama de bloques en el que se representa a un alto nivel la conexión entre fuentes de reloj y los módulos MMCM y PLL.

En la figura 1.8 se representa un diagrama de bloques de los módulos MMCM y PLL, las salidas marcadas como *MMCM only*, son salidas que únicamente están implementadas en los módulos MMCM, y no en los PLL. Se pueden ver las entradas CLKIN1 y CLKIN2 presentes en la figura 1.7, estas son las dos posibles fuentes de reloj seleccionables en el bloque CTM mediante el multiplexor que sigue a continuación. El bloque *D* representa un divisor común a todas las señales de reloj generadas por el módulo y se denomina *DIVCLK\_DIVIDE*. Los bloques PFD, CP, LF y VCO se encargan de monitorizar la frecuencia y voltaje de la fuente de reloj y ajustar la señal de reloj a su salida. Además, implementan un multiplicador de frecuencia denominado *CLKFBOUT\_MULT\_F*.

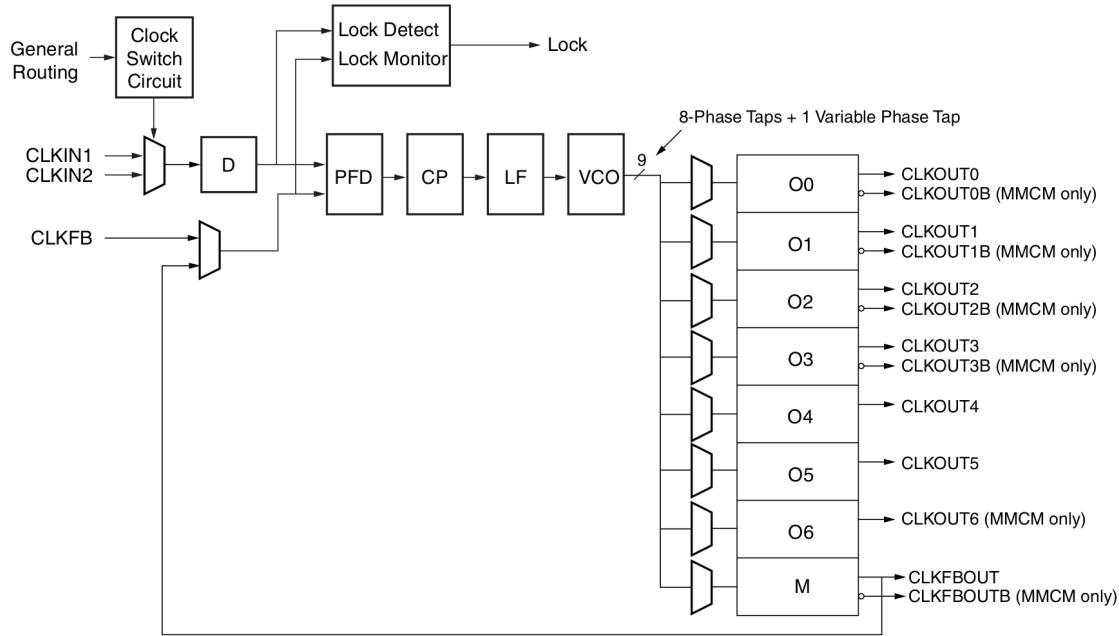


Figura 1.8: Diagrama de bloques de un módulo MMCM y PLL. Fuente: Xilinx

Tras el bloque VCO la frecuencia de la señal de reloj generada se puede calcular con la ecuación 1.1, donde M representa  $CLKFBOUT\_MULT\_F$  y D representa  $DIVCLK\_DIVIDE$ .

$$F_{VCO} = F_{CLKIN} \cdot \frac{M}{D} \quad (1.1)$$

A continuación, la salida del VCO entra en cada uno de los bloques  $O_0, O_1, O_2\dots$ , estos bloques implementan un divisor local para cada una de las salidas, denominado  $CLKOUTx\_DIVIDE$  y finalmente, la expresión de la frecuencia de reloj de salida para cada bloque es la mostrada en la ecuación 1.2, donde M y D representan los mismos parámetros explicados en la ecuación 1.1 y  $O_x$  representa el divisor local, denominado  $CLKOUTx\_DIVIDE$ .

$$F_{OUTx} = F_{CLKIN} \cdot \frac{M}{D \cdot O_x} \quad (1.2)$$

Para las ecuaciones 1.1 y 1.2:

- M equivale a *CLKFBOUT\_MULT\_F*.
- D equivale a *DIVCLK\_DIVIDE*.
- Ox equivale a *CLKOUTx\_DIVIDE*.

Más adelante se explicará el core *Clock Wizard*. Este core permite configurar el valor de los registros *DIVCLK\_DIVIDE*, *CLKFBOUT\_MULT\_F* y *CLKOUT\_DIVIDE* y por tanto, configurar cada una de las salidas del módulo MMCM o PLL. Una de las grandes ventajas que tiene el core *Clock Wizard* es que permite implementar una reconfiguración dinámica de algunos de estos registros, permitiendo así, modificar señales de reloj, cambiando su frecuencia, su fase y otros parámetros.

#### 1.1.4. Red Pitaya

Habiendo explicado ya las características principales del *Zynq-7010* de Xilinx, se va a estudiar la placa de desarrollo *Red Pitaya STEMLab 125-14*, que es sobre la que se va a desarrollar este trabajo. En la figura 1.10 se puede ver una imagen real de la placa de desarrollo. En la figura 1.9 se muestra un esquemático de la placa en el que se presentan los diferentes periféricos e interfaces que se van a utilizar [16].

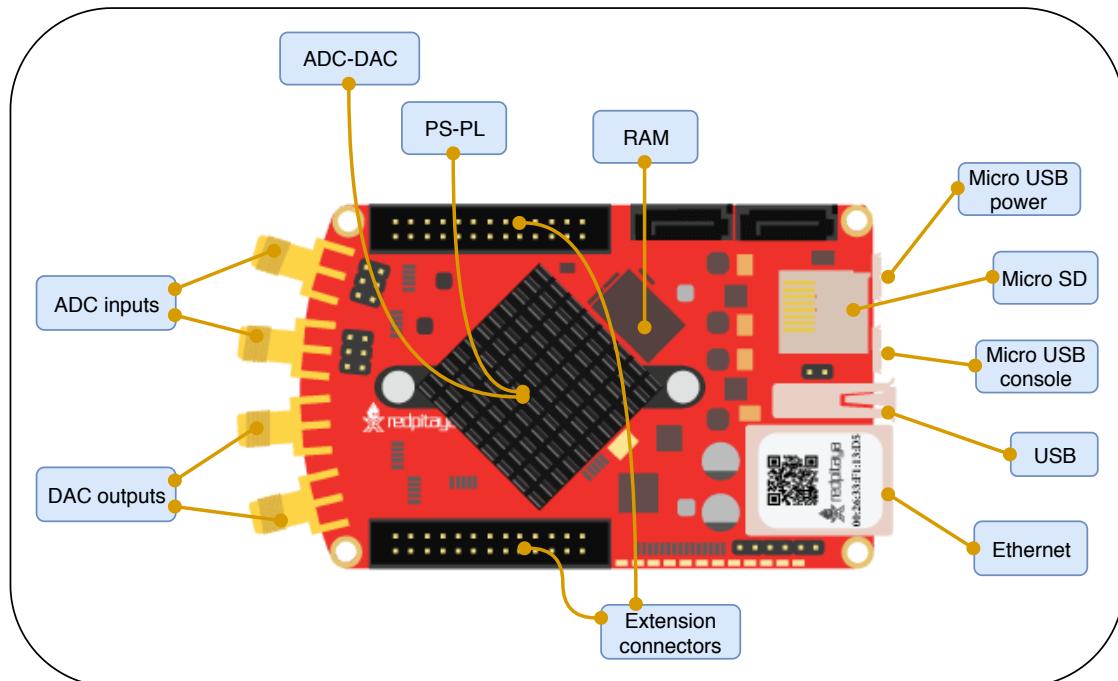


Figura 1.9: Esquemático *Red Pitaya*.

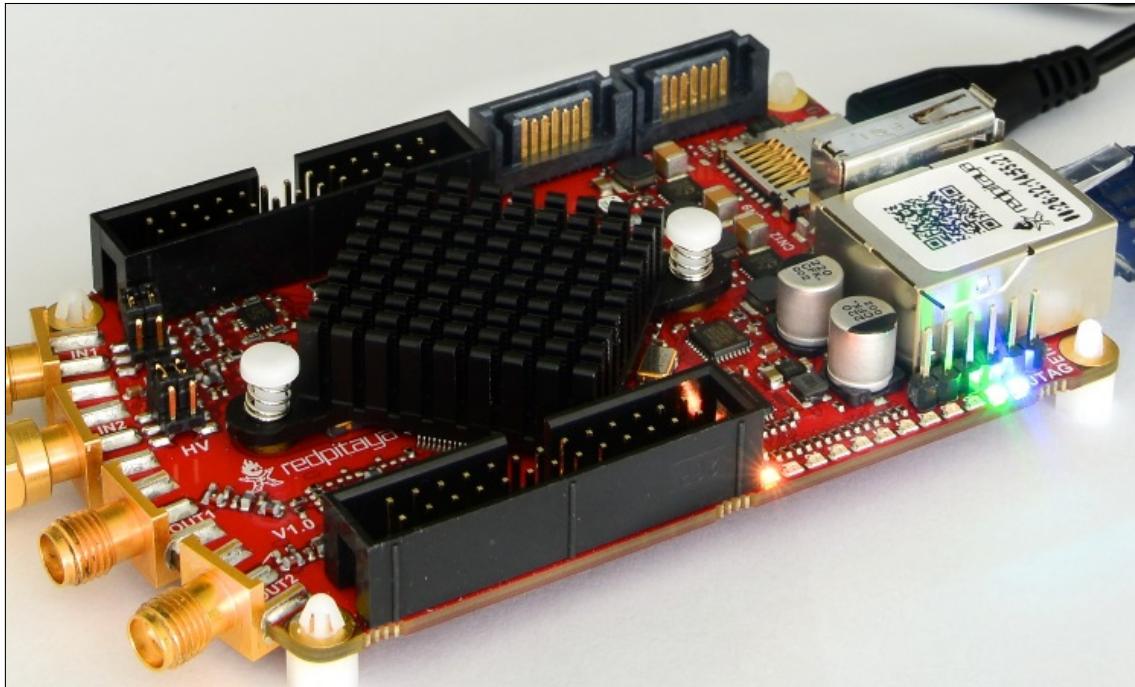


Figura 1.10: Placa *Red Pitaya*. Fuente: Red Pitaya.

La placa de desarrollo *Red Pitaya* es un proyecto *open source* que pretende implementar en un único dispositivo, un laboratorio de instrumentación de muy de bajo coste y de gran movilidad. Para ello, dispone del *Zynq-7010* de Xilinx, un ADC, un DAC, además de todo lo necesario para hacer funcionar el sistema basado en ARM, alimentación y periféricos estándares como *USB* o *Ethernet*. Para los conectores del ADC y el DAC la placa tiene soldados cuatro conectores SMA.

Puesto que este trabajo consiste en implementar un sistema de comunicación utilizando el ADC y DAC de la *Red Pitaya*, se mencionarán a continuación, las características más interesantes de estos periféricos.

- Frecuencia de muestreo: 125 MS/s.
- Resolución: 14 bits.
- Impedancia de carga:  $50\Omega$ .
- Rango de voltaje en modo LV o voltaje bajo:  $\pm 1V$ .
- Rango de voltaje en modo HV o voltaje alto:  $\pm 20V$  (sólo disponible para ADC).

- *Slew Rate* de salida:  $200V/\mu s$

También se presentan algunas características para los conectores del banco de extensión que integra la placa, puesto que en más de una ocasión se utilizarán para controlar o depurar algunas de las aplicaciones.

- Niveles de voltaje a nivel alto disponibles: +5V, +3.3V, -4V.
- Interfaces de comunicación disponibles: I2C, SPI, UART.

## 1.2. Plataforma software

Resulta interesante mencionar que tanto la compañía que ha desarrollado la placa *Red Pitaya*, como algunos desarrolladores de software libre, sustentan un software de alto nivel para el dispositivo, orientado a usuarios no expertos, que simplemente busquen utilizar esta placa como banco de instrumentación. Para este trabajo, en cambio, se deberán utilizar herramientas de desarrollo de forma que sea posible la implementación de diseños total o parcialmente creados por un desarrollador.

Para programar la FPGA se va a utilizar el software Vivado Design Suite 2018.2 proporcionado por Xilinx. El sistema operativo que se instalará en el procesador ARM de la placa, será una versión de Ubuntu 16.04, configurada para funcionar en la *Red Pitaya* y proporcionada por la propia compañía desarrolladora.

# Capítulo 2

## Sistema de Comunicación

### Contenido

---

2.1	Estándar IEEE 802.15.7 . . . . .	24
2.2	Transceptores ópticos . . . . .	25
2.3	Sistema de comunicación actual . . . . .	25
2.4	Mejoras sobre el sistema actual . . . . .	26

---

El objetivo de este trabajo es desarrollar un sistema de comunicaciones VLC basado en FPGA, seleccionando y cumpliendo con las especificaciones del estándar IEEE 802.15.7 que mejor se adapten a un sistema de comunicación orientado a aplicaciones vehiculares.

Para el sistema que se pretende diseñar e implementar, existen una serie de requisitos impuestos por el software y los dispositivos que se han acordado utilizar. Son los presentados a continuación.

- El *hardware* sobre el que se va a implementar el sistema será el dispositivo *STEMLab 125-14 Red Pitaya*, que tiene integrado el SoC, *Zynq 7010* de Xilinx.
- El software que se va a programar para ser integrado en la placa, se va a desarrollar en Vivado Design Suite, desarrollado por Xilinx.

En la sección 2.1, se presentan las especificaciones tomadas del estándar IEEE 802.15.7 para ser implementadas y se justificará el por qué de su elección.

En la sección 2.2, se detallan las características de los equipos transceptores ópticos. Estos equipos afectan de forma directa al diseño que se quiere desarrollar, ya

que algunos de ellos impondrán los extremos de funcionamiento a los que el diseño se podrá extender, como por ejemplo, la frecuencia de funcionamiento, la resolución de los conversores, la amplitud en recepción y otros.

En la sección 2.3, se explica la situación del sistema de comunicación desarrollado hasta el momento, así como sus capacidades, sus limitaciones y sus posibles mejoras.

En la sección 2.4, se profundiza en las diferentes soluciones posibles para mejorar el sistema, se decide cuál de ellas se va a implementar y el por qué de su elección.

## 2.1. Estándar IEEE 802.15.7

A continuación, se presentan algunas de las especificaciones que se han seleccionado de acuerdo con el estándar IEEE 802.15.7 para VLC, concretamente con la capa PHY I, diseñada para aplicaciones de exterior [2]:

- Se va a utilizar OOK como modulación para las comunicaciones.
- Se va a utilizar codificación Manchester.
- Se van a utilizar dispositivos LED, como elementos ópticos de emisión.
- En solución al fenómeno *flickering*, el máximo período de tiempo para comutar el estado del LED emisor será de 5ms.

La elección de OOK como modulación, viene dada debido a su simpleza respecto a otros tipos de modulaciones como puede ser VPPM, la cual también está incluida en la capa PHY I del estándar. El estándar, por otra parte, recomienda el uso de OOK para aplicaciones de exterior.

Se recogen en la figura 2.1, las tasas de transmisión de datos en bits, denominado *data rate*, para cada modo de operación del estándar IEEE 802.15.7 de la capa PHY I, utilizando OOK. En la última fila, aparece un ejemplo en el que no se implementan códigos convolucionales, ni códigos Reed-Solomon, por tanto, haciendo uso de un reloj óptico de codificación de 200KHz, la tasa de transmisión de datos queda dividido por 2 debido a la codificación Manchester, permitiendo así, 100kbps de tasa efectiva de transmisión de datos [2].

Modulation	RLL coding	Optical clock	FEC		
			Outer code (RS)	Inner code (CC)	Data rate ([kb/s])
OOK	Manchester	200 kHz	(15,7)	1/4	11.67
			(15,11)	1/3	24.44
			(15,11)	2/3	48.89
			(15,11)	None	73.3
			None	None	100

Figura 2.1: Modos de operación de OOK de la capa PHY I del estándar IEEE 802.15.7. Fuente: *Impact of IEEE 802.15.7 standard on visible light communications usage in automotive applications, IEEE Communications Magazine* [2].

## 2.2. Transceptores ópticos

Para posteriormente realizar pruebas a través de un canal óptico se utilizarán transceptores ópticos los cuales presentan las siguientes características y especificaciones a tener en cuenta para el diseño del sistema de comunicación.

### Características del transmisor

- Amplitud nivel alto: 1V
- Amplitud nivel bajo: 0V

### Características del receptor

- Amplitud nivel alto: 1V
- Amplitud nivel bajo: -1V

Las amplitudes del receptor están diseñadas para el conversor analógico digital de la placa Red Pitaya, cubren todo el rango del conversor, por tanto la resolución útil será la máxima (ver apéndice A.3).

## 2.3. Sistema de comunicación actual

El sistema de comunicación desarrollado hasta el momento se presenta en la figura 2.2 y el flujo de transmisión-recepción del sistema es el siguiente.

Un programa en lenguaje C genera una trama para transmitir a partir de un texto introducido, esta trama se almacena en una memoria RAM que posteriormente será serializada por el bloque serializador. Tras el serializador, se encuentra el modulador OOK y el codificador Manchester. Finalmente, el DAC para convertir los datos digitales a analógico y el transceptor óptico de transmisión para convertir la señal eléctrica a señal lumínica. La señal transmitida se propaga a través del canal y llega al transceptor óptico receptor. La señal recibida es muestreada por el ADC y posteriormente demodulada y decodificada por los siguientes bloques. Los datos se van apilando en una memoria RAM, que posteriormente los mandará a un programa en lenguaje C, funcionando en el ARM.

Este sistema está diseñado para operar a 200kHz o 100kbps, tal como sugiere el estándar para una modulación OOK y codificación Manchester. Y el método de decisión que implementa este sistema es un detector de umbrales, para decidir si el dato recibido es un nivel alto o bajo. La desventaja de este método es el corto alcance que permite debido a dos motivos principales, de los cuales se hace responsable el canal. Por un lado, la gran atenuación que se produce en la señal durante su propagación por el medio, hace que tal y como funciona el sistema sea muy difícil la recuperación de los datos a poco que se alejen los transceptores. Además de atenuar, el canal también introduce un ruido que en los posteriores análisis y cálculos se supondrá como ruido blanco gaussiano. Debido a estos dos motivos, las pruebas realizadas sobre este sistema no han permitido apenas conseguir transmisiones exitosas a más de 20 centímetros. En el capítulo de pruebas, también se testearán dos equipos transmisores con la intención de medir la mejoría de uno con respecto a otro ya que los transceptores ópticos también limitan las capacidades del sistema.

## 2.4. Mejoras sobre el sistema actual

Como se ha visto en la sección 2.3, el sistema actual no permite el alcance necesario para las aplicaciones vehiculares que se desean desarrollar, por tanto, es necesario integrar algún subsistema capaz de mejorar las prestaciones. El problema al que se debe hacer frente en el sistema montado actualmente, es la mala calidad y la pequeña relación señal a ruido con la que se consigue la señal en recepción. Las soluciones que se pueden adoptar son: añadir un filtro en recepción de forma que se pueda filtrar el ruido blanco gaussiano introducido por el canal y/o amplificar la señal de recepción. La amplificación en recepción se recomienda que sea implementada en el transceptor óptico mediante una amplificador de ganancia programable, proporcionando siempre una amplitud acondicionada al ADC y aprovechando su máxima resolución. El filtrado del ruido blanco gaussiano introducido por el canal,

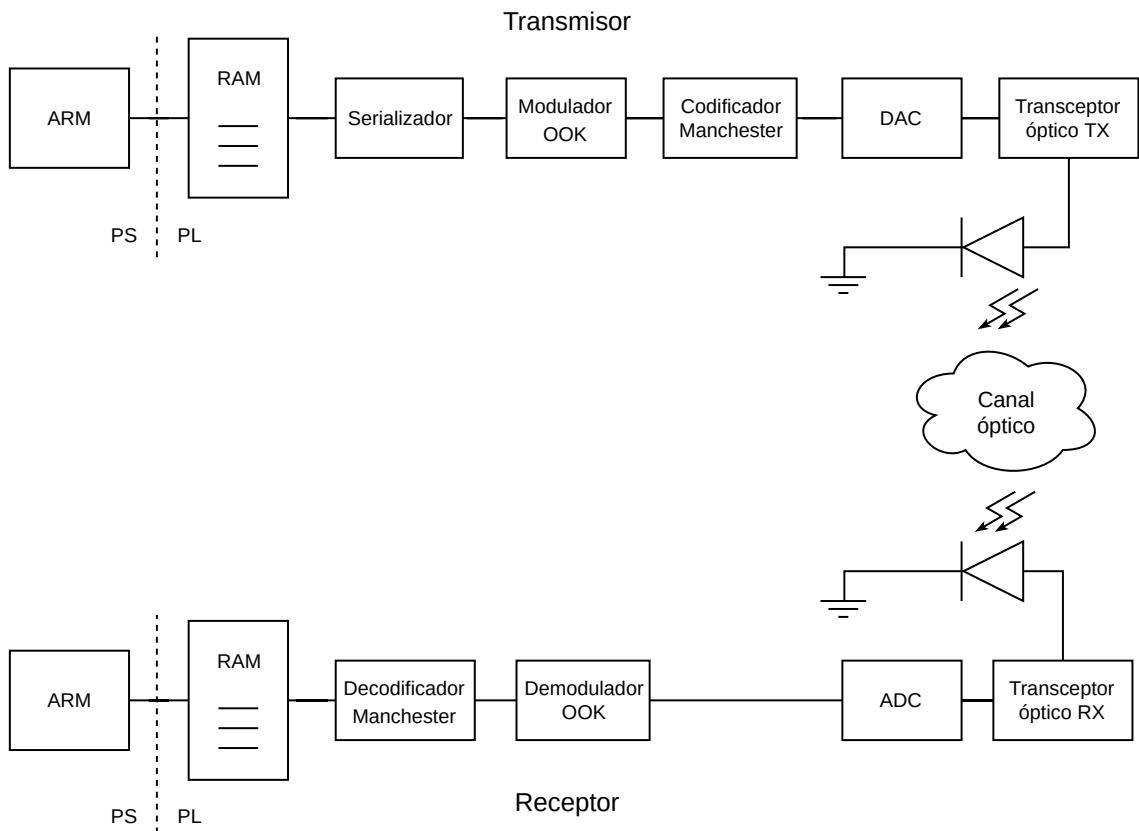


Figura 2.2: Diagrama del sistema de comunicación, sin filtro adaptado.

sin embargo, es una solución potente y eficaz para conseguir mejorar las prestaciones actuales del sistema.

Para realizar este filtrado se va a implementar un filtro adaptado, que de forma abreviada, se define como un sistema capaz de detectar un patrón conocido en una señal aleatoria y desconocida, maximizando la relación señal a ruido en un instante de muestreo determinado y cuya muestra será la que posteriormente habrá a la entrada de un decisor. Por ello, será muy útil para detectar los patrones, que en este caso serán los pulsos generados por el transmisor y emitidos al canal.

En la figura 2.3, se puede ver cómo queda el esquema del sistema con el filtro adaptado integrado. En los siguientes capítulos se estudiará el filtro adaptado tanto de forma teórica, como con simulaciones y con su implementación en la placa.

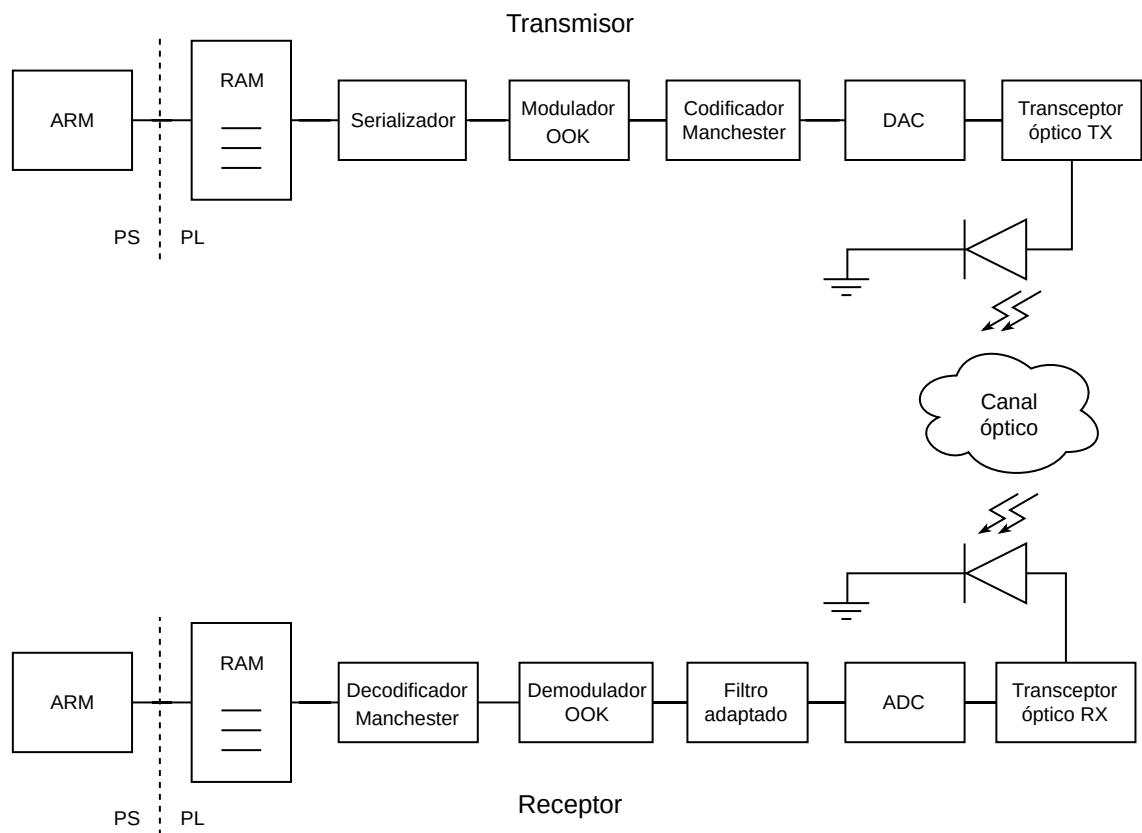


Figura 2.3: Diagrama del sistema de comunicación, con filtro adaptado.

# Capítulo 3

## Estudio del Filtro Adaptado

### Contenido

---

3.1	Fundamento teórico del filtro adaptado . . . . .	29
3.2	Representación del filtro adaptado . . . . .	31
3.3	Recuperación de sincronización . . . . .	33
3.3.1	Early-late gate algorithm . . . . .	34
3.3.2	Gardner algorithm . . . . .	37
3.4	Comparativa en simulación de los algoritmos de sincronización . . . . .	39

---

En este capítulo se van a estudiar de forma teórica y simulada los fundamentos del filtro adaptado, así como la eficacia y eficiencia de cada uno de los algoritmos presentados para la recuperación de sincronismo [17] [18].

Por orden, se demostrará de forma matemática el funcionamiento del filtro adaptado, se estudiarán simulaciones en las que se podrá comprobar de forma gráfica el funcionamiento de cada uno de los bloques que componen el filtro adaptado, se presentarán y explicarán dos algoritmos de sincronización y finalmente se simularán dichos algoritmos, obteniendo una comparativa en cuanto a su eficacia y robustez.

### 3.1. Fundamento teórico del filtro adaptado

El filtro adaptado o *matched filter* es un sistema capaz de detectar un patrón conocido en una señal aleatoria y desconocida. Para ello se correla el patrón que se quiere detectar invertido en el tiempo con la señal desconocida. Esta operación es

equivalente a convolucionar la señal desconocida con el patrón conocido conjugado e invertido en el tiempo.

De esta forma, si se define la forma de pulso en transmisión como:

$$h(t) \text{ para } 0 \leq t \leq T \quad (3.1)$$

Entonces, la respuesta ideal del filtro adaptado será:

$$h_m(t) = h(T - t) \text{ para } 0 \leq t \leq T \quad (3.2)$$

Suponiendo una señal recibida  $x(t)$ , y teniendo en cuenta que aplicar el filtro adaptado a dicha señal es equivalente a realizar una convolución entre la señal y el filtro, se puede calcular la expresión de la salida del filtro como:

$$y(t = T) = \int_0^T x(t) \cdot h_m(T - t) dt \quad (3.3)$$

Donde  $y(t)$  representa la salida del filtro adaptado en el instante  $T$ .

Se había definido que  $h_m(t) = h(T - t)$ , por tanto si se sustituye en la ecuación 3.3, se obtiene la siguiente equivalencia:

$$y(t = T) = \int_0^T x(t) \cdot h(T - (T - t)) dt = \int_0^T x(t) \cdot h(t) dt \quad (3.4)$$

La expresión 3.4 se ha derivado de las expresiones 3.2 y 3.3, y equivale a la correlación cruzada de la señal recibida  $x(t)$  y el pulso en transmisión en el instante  $T$ .

Este comportamiento del filtro adaptado proporciona dos grandes ventajas. Por un lado, las formas de pulso generalmente están compuestas por bajas frecuencias, por tanto, la aplicación de un filtrado paso bajo a la señal en recepción permite el paso a estas formas de pulso que contienen la información relevante y atenúa las altas frecuencias que únicamente contienen ruido. La otra ventaja es la correlación que realiza el filtro de la señal en recepción con la forma de pulso en transmisión,

detectando así, cuándo la señal recibida coincide con el patrón conocido.

El filtro adaptado es un filtro lineal muy eficaz para maximizar la SNR ante la presencia de ruido blanco, es decir, ruido cuyas muestras no mantienen correlación estadística en el dominio del tiempo, y por tanto su densidad espectral de potencia es constante. Debido a que se va a modelar y diseñar un sistema de comunicaciones basado en señales electromagnéticas, el ruido que se va a tomar para todos los estudios y simulaciones será ruido blanco gaussiano o AWGN.

Para la implementación del filtro adaptado completo será necesario el funcionamiento de 2 etapas adicionales.

### **Etapa 1: Control automático de ganancia**

Esta etapa se encarga de escalar de forma automática y adaptada, la señal en recepción a unos valores de potencia conocidos. Generalmente esta etapa se diseña en la parte analógica. El problema de no implementar esta etapa es que los conversores analógico-digital tienen un rango dinámico limitado, por tanto si la señal en recepción tiene una potencia demasiado alta, el conversor introducirá una distorsión conocida como *clipping*. Si por el contrario, la potencia de la señal en recepción es demasiado baja, se estará desaprovechando parte de la resolución del conversor, lo que implica una pérdida de precisión a la hora de recuperar la información relevante.

### **Etapa 2: Recuperación de sincronización**

Esta etapa es la más crítica y necesaria a la hora de implementar un filtro adaptado. El objetivo de esta etapa es sincronizar el muestreo sobre la salida del filtro adaptado, tanto a la frecuencia en recepción como a su fase.

## **3.2. Representación del filtro adaptado**

En la figura 3.1 se muestra un diagrama de bloques de un filtro adaptado, se puede diferenciar el conversor analógico-digital de entrada, el filtro adaptado a la forma de pulso  $h_m[n]$ , y el bloque sampleador, que se encarga de conseguir la recuperación de la sincronización.

A continuación, se muestra gráficamente el comportamiento de cada bloque en función a diferentes señales en recepción. La señal de transmisión se puede ver en la figura 3.2 y va a ser común a todas las simulaciones con un valor de  $\pm 1V$ .

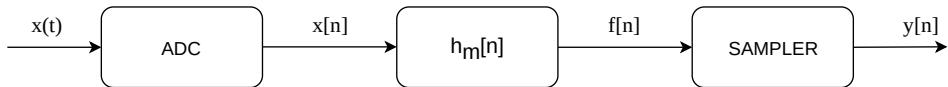


Figura 3.1: Diagrama de bloques del filtro adaptado.

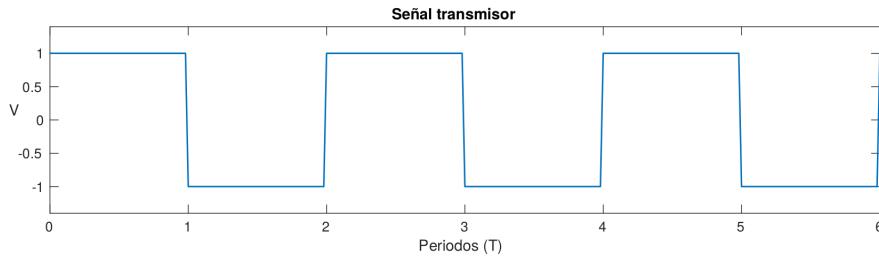


Figura 3.2: Señal de transmisión supuesta para todas las simulaciones.

En la figura 3.3, se ha simulado una SNR en recepción de 25dB. Se puede ver una ligera oscilación en la señal recibida  $x(t)$ , debido al ruido añadido. En la señal de salida del filtro adaptado  $f[n]$  se genera el resultado de la acumulación durante el período completo. Finalmente el sampleador captura las muestras en los instantes óptimos,  $nT$ , de máxima SNR. La recuperación de los datos es correcta.

En la figura 3.4, se muestra la misma simulación para una SNR en recepción de 0dB. Si se observa la señal recibida aparece muy distorsionada, sin embargo, el filtro adaptado recupera correctamente los datos originales, en los instantes de máxima SNR. La recuperación sigue siendo correcta.

En la figura 3.5, se muestra la misma simulación para una SNR en recepción de -10dB. La señal recibida ya es totalmente irreconocible de forma visual, el filtro adaptado recupera correctamente los datos originales, pero con mayor dificultad. Por ejemplo la muestra capturada en el instante  $t = 3T$ , podría tener un valor aproximado de 0.5V, lo que se interpretaría correctamente por el decisor como un nivel alto, pero la diferencia es mucho menor que en la simulación con una SNR de 25dB. La recuperación sigue siendo correcta.

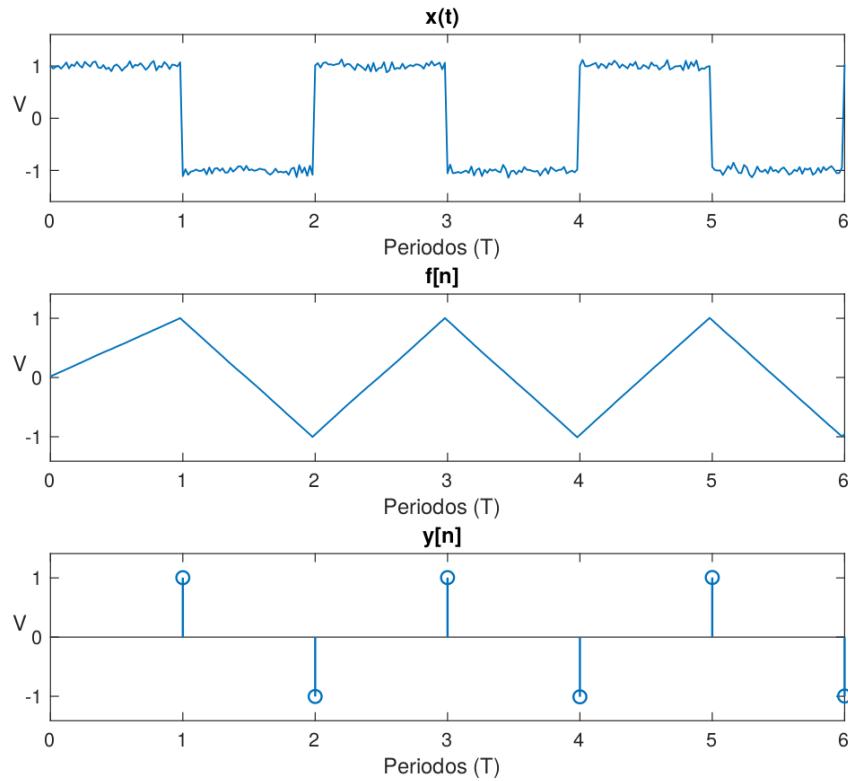


Figura 3.3: Representación gráfica del comportamiento de cada bloque del filtro adaptado para una SNR en recepción de 25dB.

### 3.3. Recuperación de sincronización

El bloque sampleador, presente en la figura 3.1, es el encargado de la recuperación de sincronización tanto en frecuencia, como en fase. Para el trabajo actual, la frecuencia en transmisión es conocida por el receptor y por tanto no se trata de detectar y recuperar. La sincronización en fase, en cambio, es un problema que se debe tener en cuenta.

El método para conseguir la sincronización en fase consiste en desplazar la fase de muestreo sobre la salida del filtro adaptado hasta que las capturas se realicen en los instantes de máxima relación señal a ruido o SNR. Para conseguir esto se necesitan dos cosas, por un lado va a ser necesario medir el error en fase que se está cometiendo entre la captura realizada y la captura óptima. Por otro lado va a ser necesario poder desplazar la fase de muestreo. De estas dos tareas se encarga el bloque sampleador.

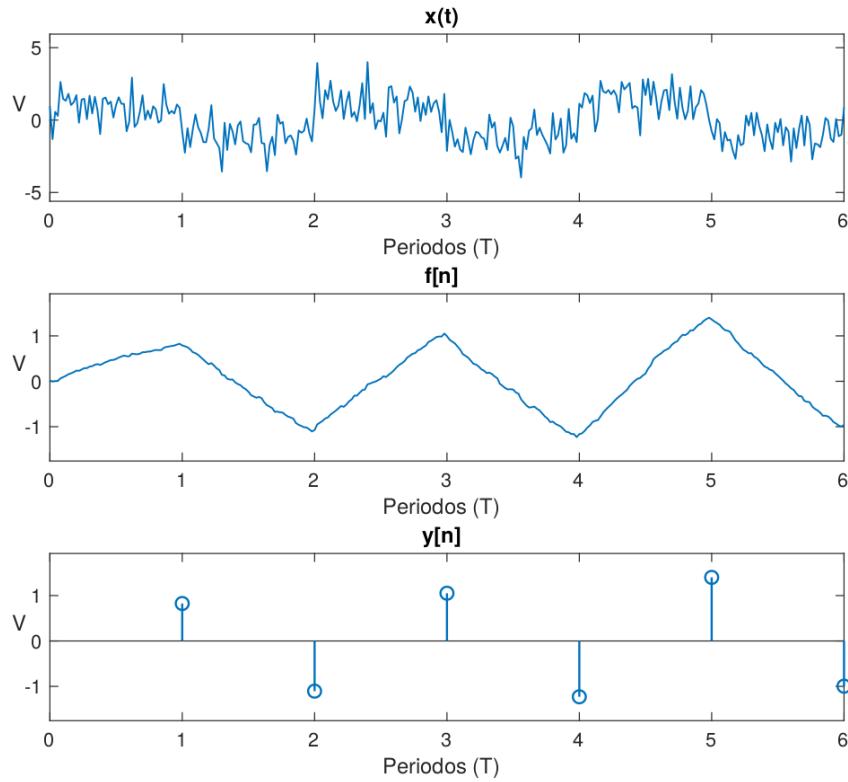


Figura 3.4: Representación gráfica del comportamiento de cada bloque del filtro adaptado para una SNR en recepción de 0dB.

Para medir el error en fase que se está cometiendo en el muestreo se estudiarán y compararán dos algoritmos de recuperación de sincronización denominados *Early-late gate algorithm* y *Gardner algorithm*.

### 3.3.1. Early-late gate algorithm

El algoritmo *Early-late* calcula el error a partir de muestras que están adelantadas y atrasadas respecto al punto de muestreo ideal. Para implementar este algoritmo es necesario tomar 3 muestras en cada período de símbolo, la muestra adelantada, la atrasada y la salida del filtro adaptado que es la que cae en medio. Para calcular el error se puede utilizar la siguiente ecuación:

$$e_n = (y_n - y_{n-2}) \cdot y_{n-1} \quad (3.5)$$

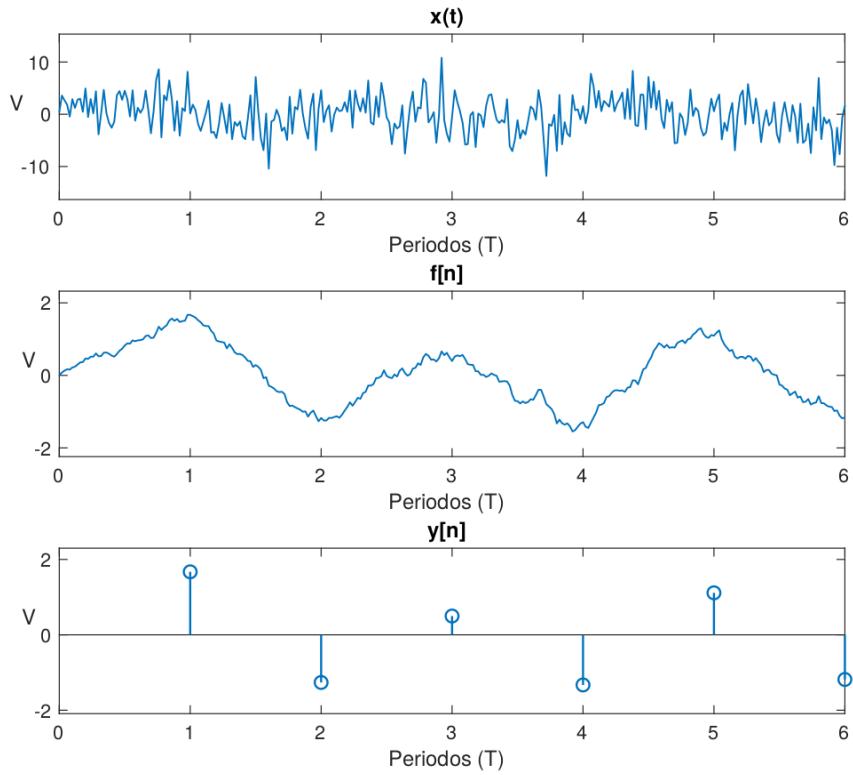


Figura 3.5: Representación gráfica del comportamiento de cada bloque del filtro adaptado para una SNR en recepción de -10dB.

La salida de este algoritmo es la muestra  $y_{n-1}$ .

En la figura 3.6, aparecen 6 casos de muestreo con el algoritmo *Early-late*. Los 3 casos superiores son casos en los que el cálculo del error se realizara sobre un pico positivo, los 3 inferiores sobre un pico negativo. Para estos ejemplos se ha supuesto una señal bipolar de  $\pm 1V$ . Si se calcula el error que devuelve la ecuación del algoritmo *Early-late* para cada uno de los ejemplos en las figuras, se obtienen los siguientes resultados:

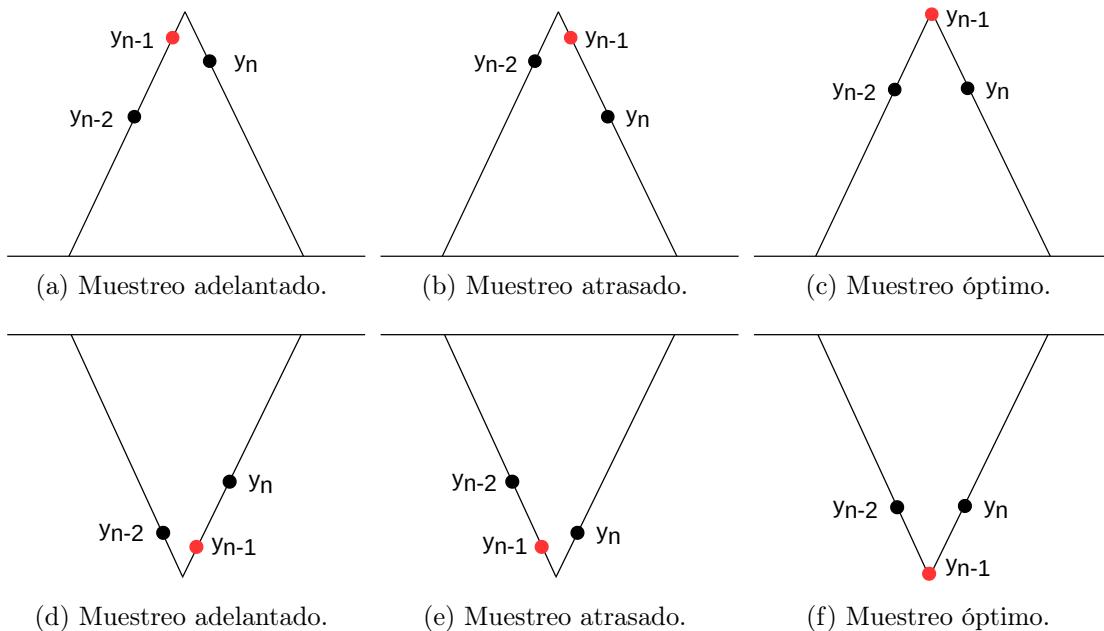


Figura 3.6: Diferentes casos de muestreo con el algoritmo *Early-late*.

$$\begin{aligned}
 e_{n3,6a} &= (0,8 - 0,6) \cdot 0,9 = 0,18 \\
 e_{n3,6b} &= (0,6 - 0,8) \cdot 0,9 = -0,18 \\
 e_{n3,6c} &= (0,7 - 0,7) \cdot 1 = 0 \\
 e_{n3,6d} &= (-0,6 + 0,8) \cdot (-0,9) = -0,18 \\
 e_{n3,6e} &= (-0,8 + 0,6) \cdot (-0,9) = 0,18 \\
 e_{n3,6f} &= (-0,7 + 0,7) \cdot (-1) = 0
 \end{aligned}$$

A continuación, se analizará el resultado de algunos de los ejemplos representados y calculados.

En la figura 3.6a, la ecuación devuelve  $e_{n3,6a} = 0,18$ . Este valor es positivo y como se puede apreciar en la figura, la corrección de fase necesaria para converger al momento óptimo de muestreo es positiva.

En la figura 3.6d, la ecuación devuelve  $e_{n3,6d} = -0,18$ . Este valor es negativo y como se puede apreciar en la figura, la corrección de fase necesaria para converger

al momento óptimo de muestreo es negativa.

De esta forma, se puede concluir que haciendo uso del algoritmo *Early-Late*, la corrección que se debe realizar a la fase de muestreo, es el valor devuelto por la ecuación 3.5, aplicando la correspondiente normalización en amplitud.

### 3.3.2. Gardner algorithm

El algoritmo de Gardner está muy extendido y es frecuente su uso en sistemas de recuperación de sincronismo. Gardner utiliza 2 muestras por período de símbolo y para calcular el error utiliza la ecuación 3.5, al igual que el algoritmo *Early-late*.

La salida de este algoritmo es la muestra  $y_n$ .

La separación temporal entre las muestras exteriores,  $y_n$  y  $y_{n-2}$  es de un período de bit, y la separación temporal entre las muestras consecutivas,  $y_n$  y  $y_{n-1}$  es de medio período de bit. Es importante destacar que la ecuación de Gardner necesita muestrear sobre una señal bipolar para que el error calculado sea correcto.

En la figura 3.7, se pueden observar 6 casos de muestreo con el algoritmo de Gardner. Los 3 casos superiores son casos en los que el cálculo del error se realizará sobre una pendiente negativa, los 3 inferiores son casos con pendiente positiva. Para estos ejemplos se ha supuesto una señal bipolar de  $\pm 1V$ . Si se calcula el error que devuelve la ecuación de Gardner para cada uno de los ejemplos en las figuras, se obtienen los siguientes resultados:

$$\begin{aligned} e_{n3,7a} &= (-0,8 - 0,8) \cdot (0,2) = -0,32 \\ e_{n3,7b} &= (-0,8 - 0,8) \cdot (-0,2) = 0,32 \\ e_{n3,7c} &= (-1 - 1) \cdot 0 = 0 \\ e_{n3,7d} &= (0,8 + 0,8) \cdot (-0,2) = -0,32 \\ e_{n3,7e} &= (0,8 + 0,8) \cdot (0,2) = 0,32 \\ e_{n3,7f} &= (1 + 1) \cdot (0) = 0 \end{aligned}$$

El resultado de las ecuaciones aporta dos datos importantes, uno es la amplitud del error que se está cometiendo en el muestreo y el otro es el signo del error, que revelará la dirección en la que se deberá corregir la fase de muestreo. En el caso de Gardner, el signo que devuelve la ecuación del error es contrario al signo del

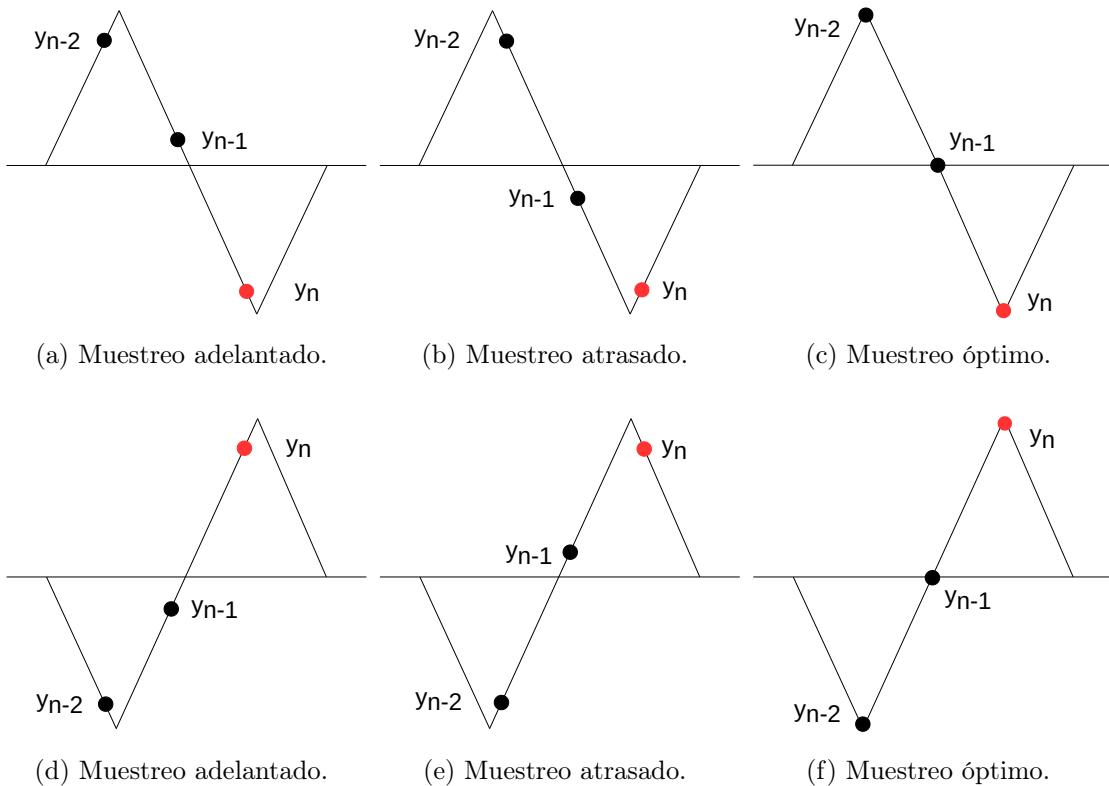


Figura 3.7: Diferentes casos de muestreo con el algoritmo *Gardner*.

incremento a realizar sobre la fase de muestreo.

A continuación, se analizará el resultado de algunos de los ejemplos representados y calculados.

En la figura 3.7a, la ecuación devuelve  $e_{n3,7a} = -0,32$ . Este valor es negativo y como se puede apreciar en la figura, la corrección de fase necesaria para converger al momento óptimo de muestreo es positiva.

En la figura 3.7e, la ecuación devuelve  $e_{n3,7e} = 0,32$ . Este valor es positivo y como se puede apreciar en la figura, la corrección de fase necesaria para converger al momento óptimo de muestreo es negativa.

De esta forma, se puede concluir que haciendo uso del algoritmo de Gardner, la corrección que se debe realizar a la fase de muestreo, es la negación del valor devuelto por la ecuación 3.5, aplicando la correspondiente normalización en amplitud.

### 3.4. Comparativa en simulación de los algoritmos de sincronización

Para comprobar la eficacia y eficiencia de los algoritmos presentados anteriormente, se van a realizar simulaciones en Matlab con diferentes tramas de transmisión y diferentes SNR en recepción.

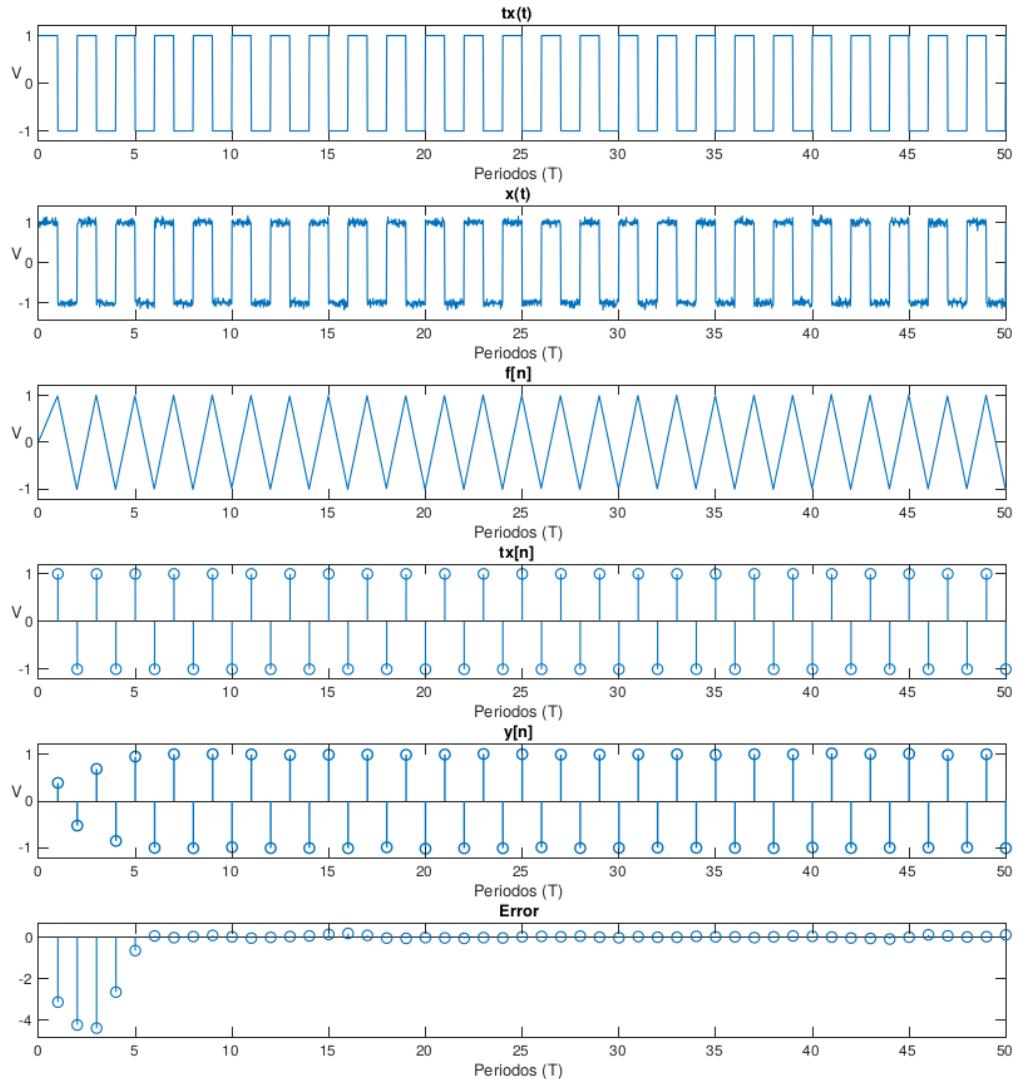


Figura 3.8: Sincronización del algoritmo Early-Late para tren de pulsos con una SNR en recepción de 25dB.

Todas las gráficas mencionadas a continuación contienen, por orden: transmisión, recepción, filtrado, transmisión discretizada, salida del sampleador y el error. El nombre de las señales se puede identificar en la figura 3.1 y la discretización del eje horizontal es en períodos de bit, T.

En la figura 3.8 aparece una simulación de sincronización del algoritmo *Early-Late* para tren de pulsos con una SNR en recepción de 25dB, la separación entre las muestras exteriores,  $y[n - 2]$ ,  $y[n]$  es del 20 % del período de bit, T. El desfase inicial aplicado es del 30 % del período de bit, T. Se puede observar como la señal  $y[n]$ , al principio consigue una menor amplitud, debido a un alto error en la fase de muestreo. Conforme avanza el tiempo, el algoritmo va ajustando la fase de muestreo y el error converge a cero.

La figura 3.9 muestra una simulación de sincronización del algoritmo de Gardner para tren de pulsos con una SNR en recepción de 25dB, el desfase inicial aplicado es del 30 % del período de bit, T. El comportamiento es el mismo que el algoritmo Early-Late excepto por el valor del error, que debido a la negación necesaria para Gardner, aparece con signo contrario. Aún así, la convergencia es equivalente.

La figura 3.10 es una simulación del algoritmo *Early-Late* bajo las mismas condiciones que la figura 3.8 pero esta vez además del tren de pulsos, simulando la trama de sincronización como cabecera del paquete, también contiene una trama pseudoaleatoria, que representa la información del paquete. La recuperación de los datos es correcta, pero cuando la señal es pseudoaleatoria el algoritmo presenta mayor inestabilidad y se pierde la sincronización. Se puede observar como el error comienza a oscilar.

La figura 3.11 es una simulación del algoritmo de Gardner bajo las mismas condiciones que la figura 3.9 pero de nuevo, se ha simulado trama de sincronización y trama pseudoaleatoria. La recuperación de los datos es correcta, en este caso, cuando la señal es pseudoaleatoria, el algoritmo presenta estabilidad y se mantiene la sincronización. Se puede observar cómo el error converge a cero.

Con las simulaciones presentadas, se puede afirmar que ambos algoritmos responden de forma efectiva ante un tren de pulsos, consiguiendo sincronizarse y manteniendo la sincronización. Sin embargo, cuando la entrada deja de ser un tren de pulsos y se presenta una trama pseudoaleatoria, resulta ser más estable y robusto el algoritmo de Gardner, sincronizándose y manteniendo la sincronización con un error que converge a cero, a diferencia del algoritmo *Early-Late*, cuyo error comienza a oscilar.

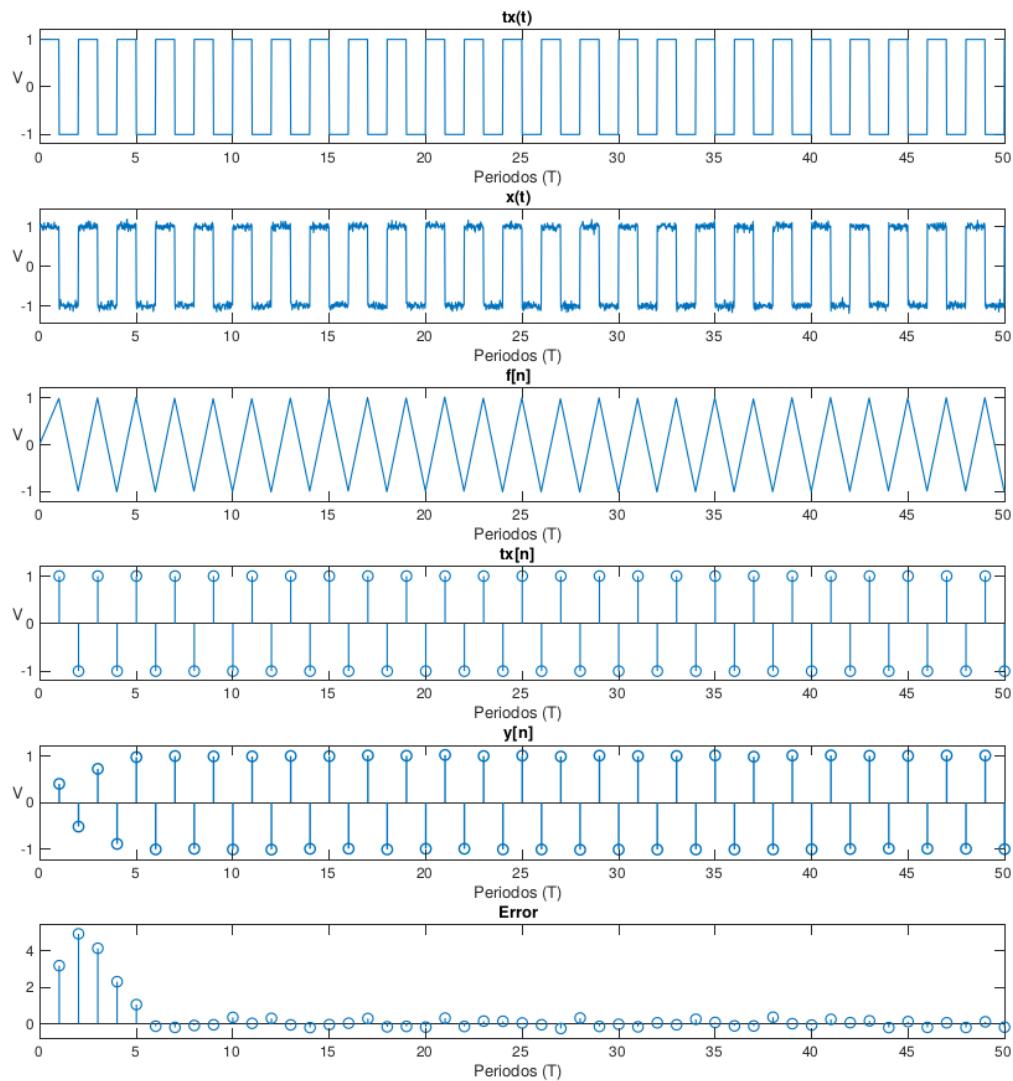


Figura 3.9: Sincronización del algoritmo de Gardner para tren de pulsos con una SNR en recepción de 25dB.

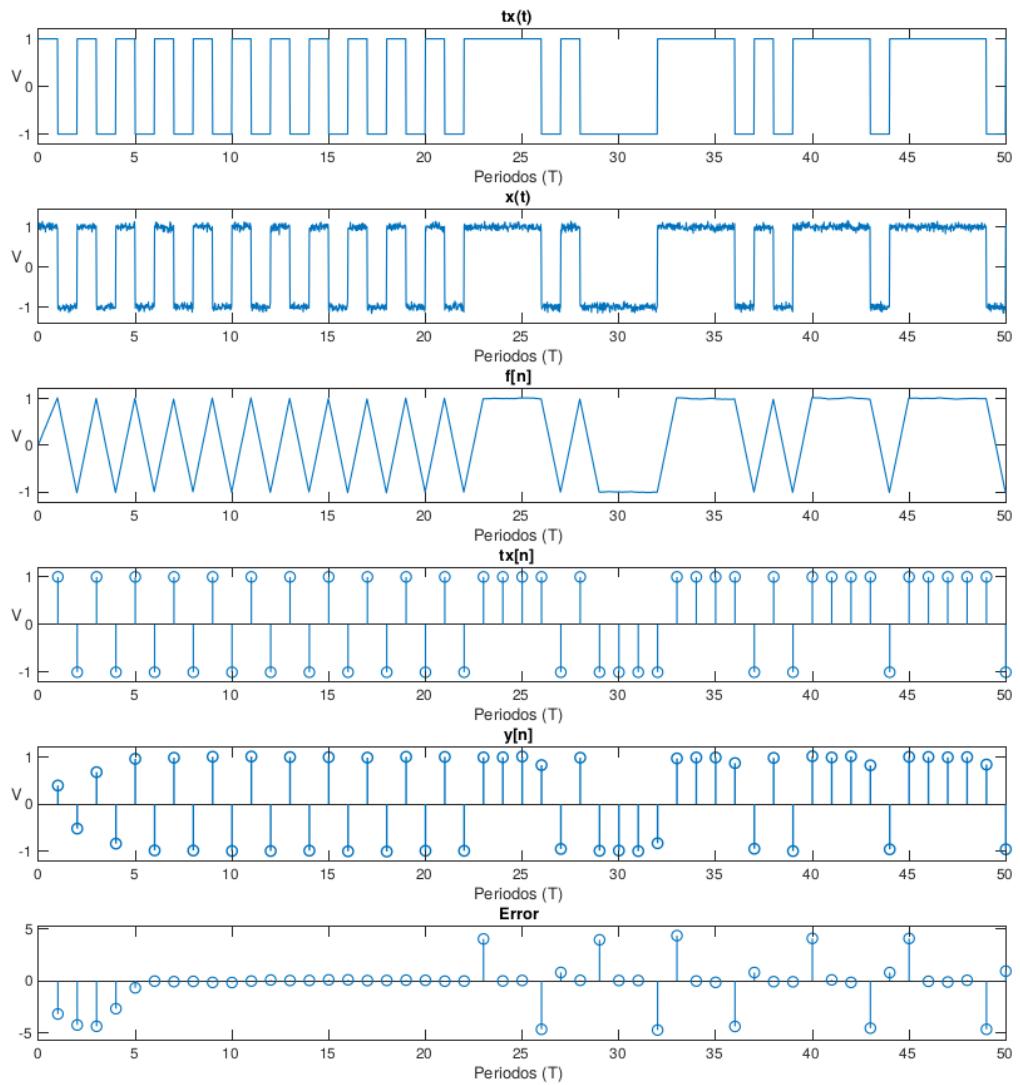


Figura 3.10: Sincronización del algoritmo Early-Late para tren de pulsos y trama pseudoaleatoria con una SNR en recepción de 25dB.

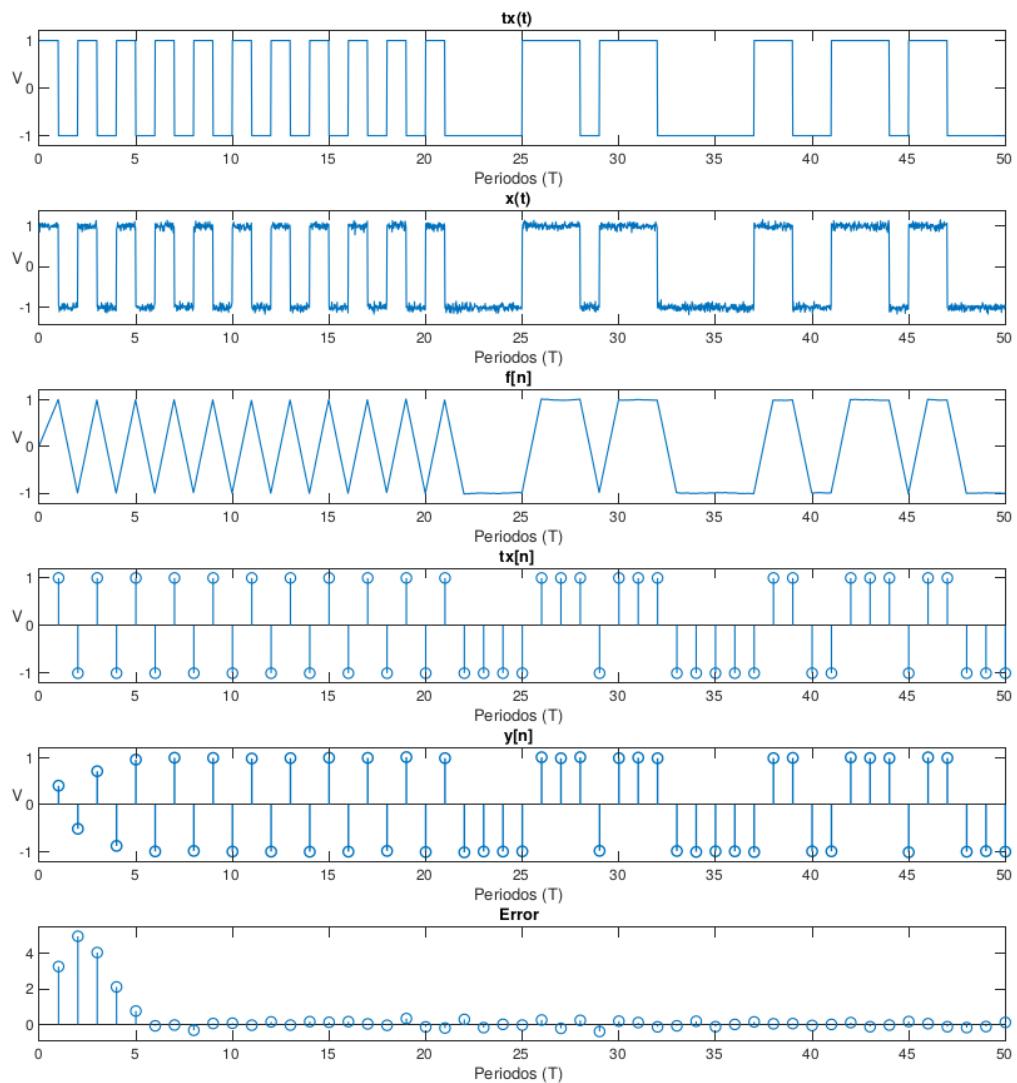


Figura 3.11: Sincronización del algoritmo de Gardner para tren de pulsos y trama pseudoaleatoria con una SNR en recepción de 25dB.



# Capítulo 4

## Implementación del Filtro Adaptado

### Contenido

---

<b>4.1 Sistema general . . . . .</b>	<b>46</b>
<b>4.2 Bloque integrador . . . . .</b>	<b>46</b>
4.2.1 Acumulador . . . . .	47
4.2.2 Divisor . . . . .	48
4.2.3 Mapeo . . . . .	50
4.2.4 Simulación del bloque integrador . . . . .	51
<b>4.3 Bloque Sampleador . . . . .</b>	<b>52</b>
4.3.1 Limiter . . . . .	53
4.3.2 Counter . . . . .	54
4.3.3 Comparador . . . . .	54
4.3.4 Simulación del bloque Sampleador . . . . .	55
<b>4.4 Bloque Error . . . . .</b>	<b>58</b>
4.4.1 Simulación del bloque Error . . . . .	59
<b>4.5 Simulación del Sistema . . . . .</b>	<b>60</b>

---

En este capítulo, se explicarán los bloques que componen la implementación del filtro adaptado, así como algunas decisiones de diseño que se han debido tomar. Se mostrarán esquemáticos circuitales equivalentes al diseño implementado y se comprobará el correcto comportamiento lógico de cada uno de ellos mediante simulaciones en vivado.

## 4.1. Sistema general

Para tener una visión alejada del bajo nivel de cada uno de los bloques que componen el diseño, se presenta a continuación, la jerarquía de los bloques principales.

```

Sistema (Top)
    → Integrador
    → Sampleador
    → Error

```

En la figura 4.1, se puede ver la representación del módulo top, *Sistema*.

El funcionamiento del módulo *Sistema* es el siguiente: los datos de entrada capturados por el ADC entran al integrador, éste genera la señal de acumulación. El bloque *Sampleador* se encarga de generar durante un período de reloj, tres diferentes señales de clock enable, que habilitarán, independientemente, cada uno de los 3 registros que aparecen en paralelo para realizar las diferentes capturas de la salida del integrador. Como se puede observar, del bloque *Sampleador* salen 3 señales de clock enable, cada una para capturar las 3 muestras necesarias durante el período de bit y posteriormente calcular el error. El bloque de error únicamente aplica la ecuación 3.5, y su salida es capturada por un registro cuando se activa CE2, este registro es útil para evitar los *glitches* que se producen mientras se capturan los datos y se calcula el resultado de la operación. Una vez el error es capturado por el registro, el bloque *Sampleador* realiza un desfase de las tres señales de clock enable para, iteración tras iteración, converger a la fase óptima de sincronización.

Finalmente, la señal *MODE* sirve para implementar el algoritmo de Gardner o el algoritmo *Early-Late*. En el *Sampleador*, alterna entre sumar o restar el error calculado y registrado. En el multiplexor que aparece en la parte superior derecha, alterna la salida del filtro, pudiendo seleccionar entre la muestra S2 o la muestra S3. Como bien se vió en el capítulo anterior, estas son las dos diferencias entre ambos algoritmos.

## 4.2. Bloque integrador

El bloque integrador es el equivalente al bloque  $h_m[n]$  de la figura 3.1, es decir, el filtro adaptado al pulso.

Este bloque se encarga de acumular la señal de entrada de forma discreta para un orden definido, normalizar esta acumulación y finalmente realizar un mapeo de

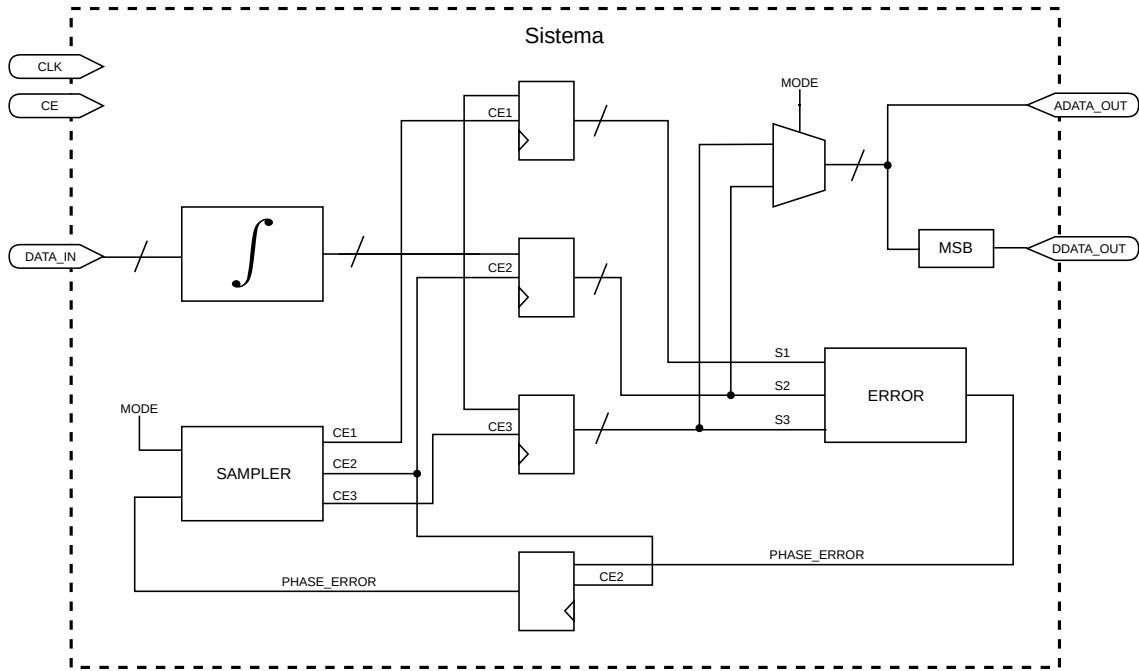


Figura 4.1: Esquemático del módulo top.

los datos de salida para acondicionarlos a los bloques posteriores. Estas tres operaciones están divididas en tres diferentes bloques, tal y como se muestra en la figura 4.2. A su vez, estos bloques están implementados con slices DSP, que son la forma óptima y mas eficiente de implementar operaciones matemáticas.

A continuación, se detallan con mayor profundidad, cada uno de los bloques que componen el integrador.

#### 4.2.1. Acumulador

El primer bloque debe realizar la ecuación 4.1, que es la expresión equivalente a un acumulador de orden  $N$ , donde  $x[n]$  es la entrada,  $acc[n]$  es la acumulación realimentada y  $y[n]$  la salida.

$$y[n] = x[n] + acc[n] - x[n - N] \quad (4.1)$$

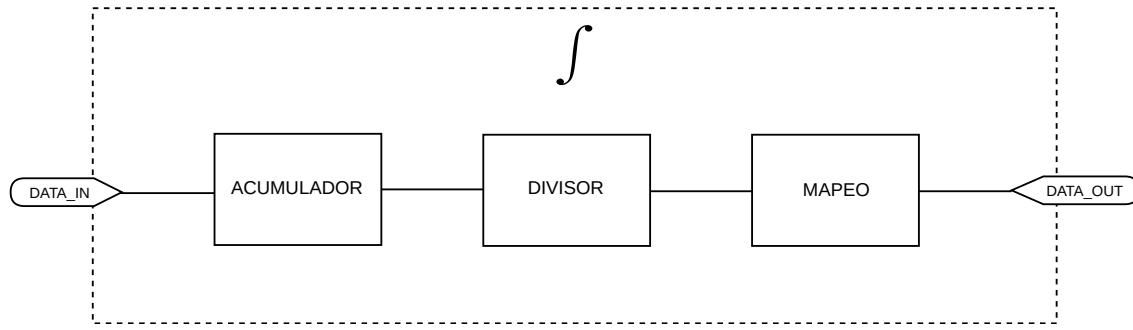


Figura 4.2: Esquemático del módulo integrador.

La implementación de dicha expresión matemática da como resultado el esquemático que se muestra en la figura 4.3. Para implementar este esquemático, se debe hacer uso de un registro de desplazamiento capaz de almacenar las muestras retrasadas en el tiempo, además de un slice DSP que se encargará de realizar la suma y resta necesaria.

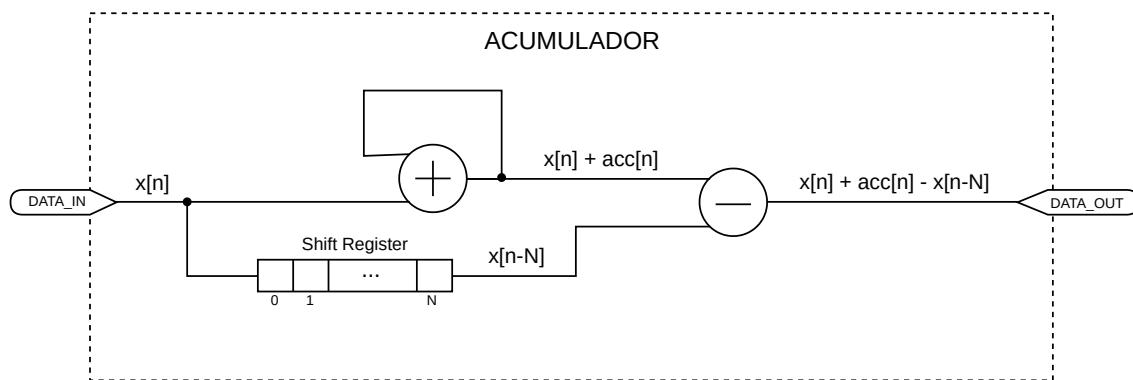


Figura 4.3: Esquemático del bloque acumulador.

#### 4.2.2. Divisor

El segundo bloque es un divisor para normalizar la acumulación. Su funcionamiento se basa en un slice DSP, por lo que se representará el esquemático de la configuración del DSP directamente. Dicho esquemático se puede ver en la figura 4.4.

Los recursos que ofrecen los slices DSP no permiten realizar una división como tal, por ello, es necesario realizar una multiplicación decimal en punto flotante para

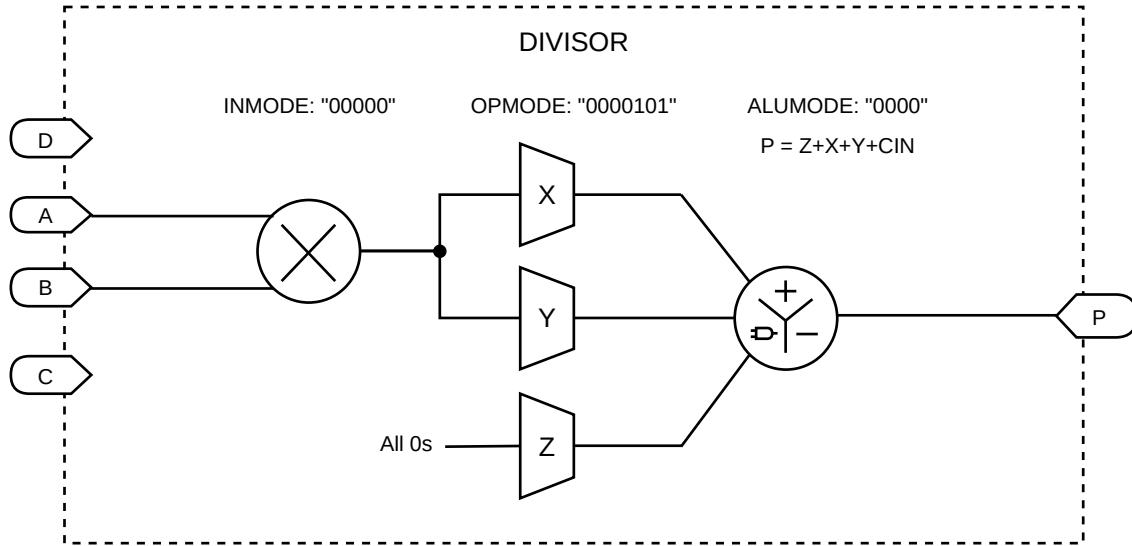


Figura 4.4: Esquemático del slice DSP equivalente al bloque divisor.

poder normalizar.

Suponiendo la expresión 4.2, como la salida del acumulador en la que  $acc$  representa la acumulación,  $N$  el orden del filtro y  $x$  el dato de entrada, el valor que se desea obtener tras el bloque divisor sería el representado por la expresión 4.3, donde  $div$  equivale a la salida del divisor.

$$acc[n] = \sum_{i=0}^{N-1} x[n-i] \quad (4.2) \qquad div[n] = \frac{1}{N} \cdot \sum_{i=0}^{N-1} x[n-i] \quad (4.3)$$

Para ello se multiplica la salida del acumulador por un valor constante definido como el inverso del orden del filtro, desplazado 17 bits a la izquierda.

Por ejemplo, si se supone un orden  $N = 625$  y una acumulación  $acc = 100$ , el resultado evidente sería:

$$div[n] = \frac{1}{N} \cdot acc[n] = 0,16$$

Pero la operación realizada por el DSP es la siguiente:

$$div[n] = \frac{1}{N} \cdot 2^{17} \cdot acc[n] = 20971,52$$

De esta forma, al desplazar 17 bits hacia la derecha, se obtiene el valor dividido:

$$\frac{div[n]}{2^{17}} = 0,16$$

Tanto la división directa, como la multiplicación en punto flotante coinciden en el resultado.

Si se observa la figura 4.4, se puede ver como la operación equivalente que realiza el slice DSP es la mostrada en la ecuación 4.4.

$$P = A \cdot B \quad (4.4)$$

### 4.2.3. Mapeo

El tercer bloque es el mapeo, encargado de acondicionar los valores de salida del divisor para que sean interpretables de forma correcta por los bloques posteriores.

Esta etapa es necesaria debido a que el ADC que se está utilizando ofrece valores entre [0, 16383], y tal como se mostró en el estudio teórico de los algoritmos de sincronización, la señal de acumulación debía ser bipolar para que las ecuaciones que proponían los algoritmos aportasen resultados correctos. Por tanto, el mapeo debe realizarse desde un dominio [0, 16383] hacia un dominio [8192, -8191], obteniendo así un escalado bipolar y aprovechando toda la resolución del conversor.

La operación matemática que debe realizar este bloque aparece reflejada en la ecuación 4.5, donde  $map$  representa la salida del bloque mapeo y  $div$  la salida del divisor.

$$map[n] = 8192 - div[n] \quad (4.5)$$

La configuración del bloque mapeo y, por tanto, del DSP asociado a este, se puede ver en la figura 4.5. Se puede observar como la entrada del multiplexor Z es PCIN, es decir, el bus dedicado para interconectar los slices DSP de forma directa sin rutado adicional, además, está seleccionada con un desplazamiento de 17 bits

hacia la derecha, este desplazamiento es necesario para rectificar la operación realizada previamente por el bloque divisor.

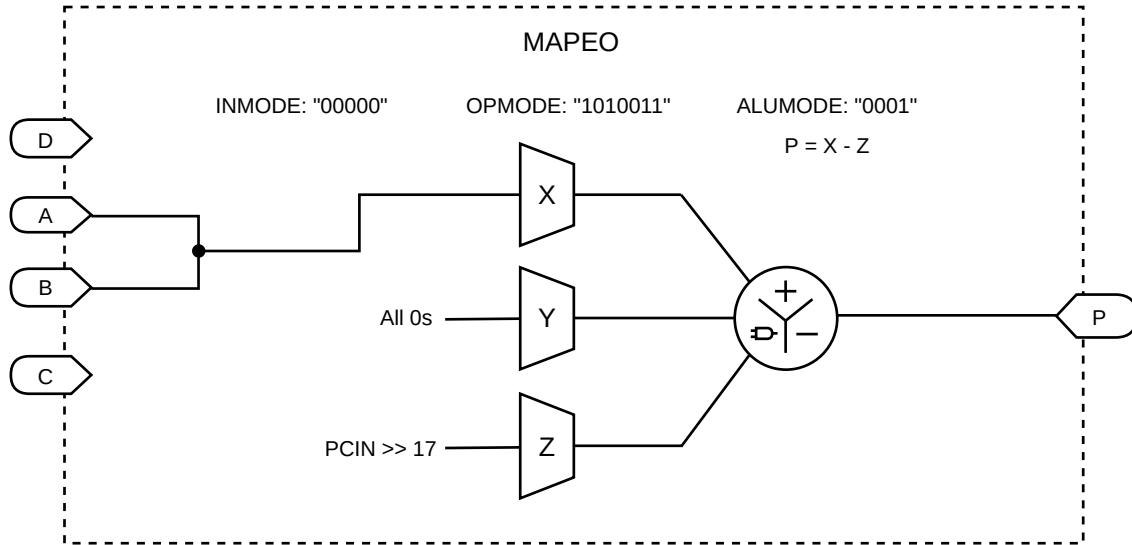


Figura 4.5: Esquemático del bloque mapeo.

#### 4.2.4. Simulación del bloque integrador

Para simular y visualizar gráficamente el funcionamiento del bloque integrador se va a preparar el siguiente escenario:

- $f_{CLK}$ , frecuencia de funcionamiento: 125MHz
- Orden del acumulador: 625

Esta configuración está diseñada para unos datos en recepción de  $T_{BIT} = 5\mu s$ , de forma que si el bloque acumulador funciona a 125MHz, el número de acumulaciones que realiza en un período de bit es de 625, de ahí el orden seleccionado. La operación utilizada para hallar el orden aparece en la ecuación 4.6.

$$orden = \frac{T_{BIT}}{T_{CLK}} = \frac{5\mu s}{8ns} = 625 \quad (4.6)$$

Configurado el escenario, se analiza la simulación. En la figura 4.6 se pueden diferenciar las siguientes señales en orden: CLK, CE, RST, entrada del acumulador,

salida del acumulador, salida del divisor y salida del bloque mapeo. Aparecen dos marcadores azules y uno amarillo. El amarillo corresponde a los valores asociados a cada señal en la parte izquierda, para el instante donde se encuentra situado el marcador.

La entrada es un tren de pulsos de  $T_{BIT} = 5\mu s$ . La señal RST comienza activa y al desactivarse comienza a funcionar el sistema. Inicialmente se observa un transitorio y una vez pasado, comienza a generarse la acumulación de los pulsos. A continuación, se van a utilizar los valores correspondientes al marcador amarillo.

A la salida del acumulador, el cursor marca un valor de 10173843, que es aproximadamente  $16383 \cdot 625 = 10239375$ , equivalente a la amplitud del pulso, 16383, multiplicado por el orden del acumulador. La salida del divisor marca un valor de 2126382336, equivalente a la siguiente expresión, explicada anteriormente:

$$\frac{1}{625} \cdot 2^{17} \cdot 10173843$$

La entrada del bloque mapeo es la salida del divisor desplazada 17 bits a la derecha, por tanto es 16233. Al realizar el mapeo, queda  $8192 - 16233 = -8031$ . El valor en simulación es de -8058, el error se debe a la precisión de la multiplicación en punto decimal.

Se puede comprobar, por tanto, que el funcionamiento en simulación del bloque integrador es correcto y concuerda con todo lo estudiado de forma teórica anteriormente.

### 4.3. Bloque Sampleador

El bloque *Sampleador* implementa la corrección de fase sobre la señal de muestreo. Esta funcionalidad es una de las dos que debía implementar el bloque Sampleador de la figura 3.1, la otra funcionalidad era el cálculo del error.

Este bloque, toma un valor de cuenta límite, implementa un contador que se incrementa hasta alcanzar dicho valor límite y una vez alcanzado, se reinicia a cero, siendo éste, el período del ciclo de cuenta. Durante cada ciclo de cuenta, se encarga de generar 3 señales activas durante un período de reloj, denominadas CE1, CE2 y CE3, que capturan cada una de las muestras necesarias para calcular el error.



Figura 4.6: Simulación del bloque integrador.

Observando la figura 4.7, se puede ver de forma más detallada el funcionamiento del Sampleador. El bloque *limiter* se encarga de calcular el valor límite de cuenta para cada ciclo, el bloque *counter* simplemente implementa un contador unitario que se resetea cuando alcanza el valor límite proporcionado por el bloque *limiter*. A través de las señales COUNT1 y COUNT2 se transmite el valor de cuenta a los bloques CMP1 y CMP2, estos dos bloques son internamente idénticos y son simples comparadores que activan su salida cuando sus dos valores de entrada coinciden.

A continuación, se profundiza en cada uno de los esquemáticos de los bloques del Sampleador.

### 4.3.1. Limiter

El bloque limiter se encarga de modificar el valor de cuenta límite para cada ciclo, pudiendo así contar un valor mayor o menor al período de bit. El límite de cuenta equivalente al período de bit se le proporciona por la entrada COUNT\_LIMIT, de esta forma se consigue desplazar la fase de muestreo hacia delante o hacia atrás. La corrección de fase realizada, será el error calculado por el módulo ERROR. La entrada MODE sirve para sumar o restar el error de entrada sobre el valor de COUNT\_LIMIT, por tanto permite alternar entre el algoritmo *Early-Late* y *Gardner*.

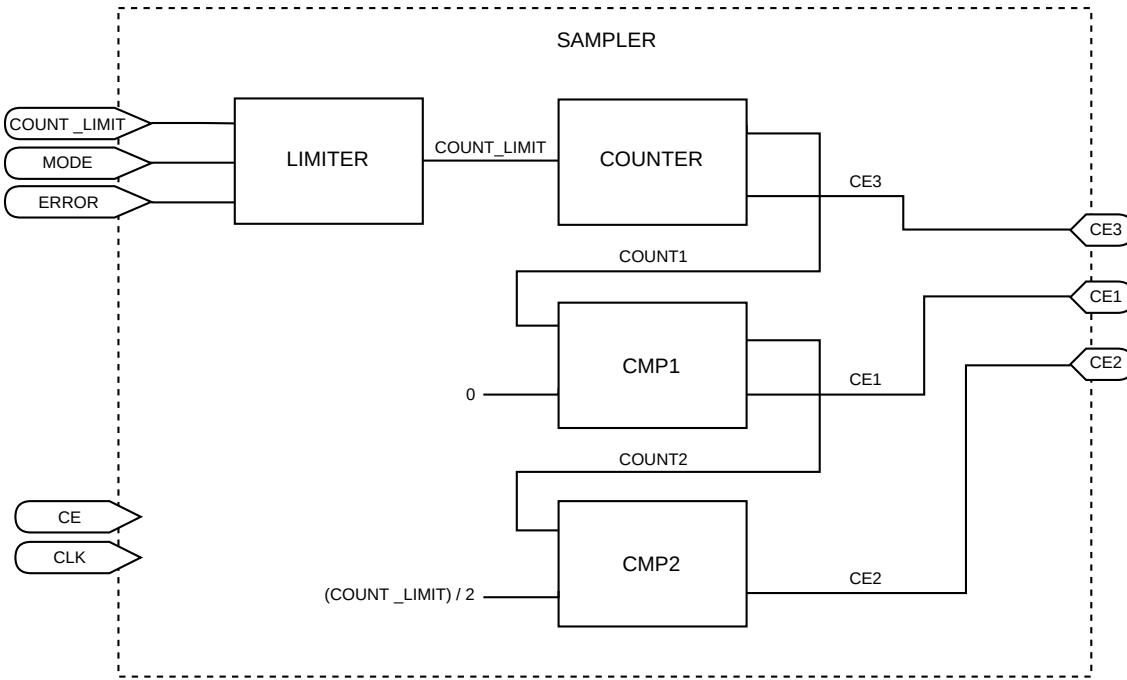


Figura 4.7: Esquemático del bloque Sampleador.

En la figura 4.8 se puede observar la configuración del slice DSP correspondiente al bloque limiter.

#### 4.3.2. Counter

El bloque *counter* se encarga de incrementar de forma unitaria un valor de cuenta y ofrecer dicho valor de cuenta a su salida. Además, tiene el detector de patrones configurado para que la cuenta se reinicie a cero cuando alcance el valor COUNT\_LIMIT. La propia señal PATTERNDETECT que ofrece el slice DSP es útil para detectar cuando la cuenta ha llegado al límite, precisamente en ese instante es cuando interesa capturar la tercera muestra, por lo tanto, la salida PATTERNDETECT será CE3.

La configuración del slice DSP que implementa el bloque *counter* se puede ver en la figura 4.9.

#### 4.3.3. Comparador

Los bloques comparadores CMP1 y CMP2 están implementados también sobre slices DSP. El bloque CMP1 transmite el valor de cuenta a través del bus PCIN

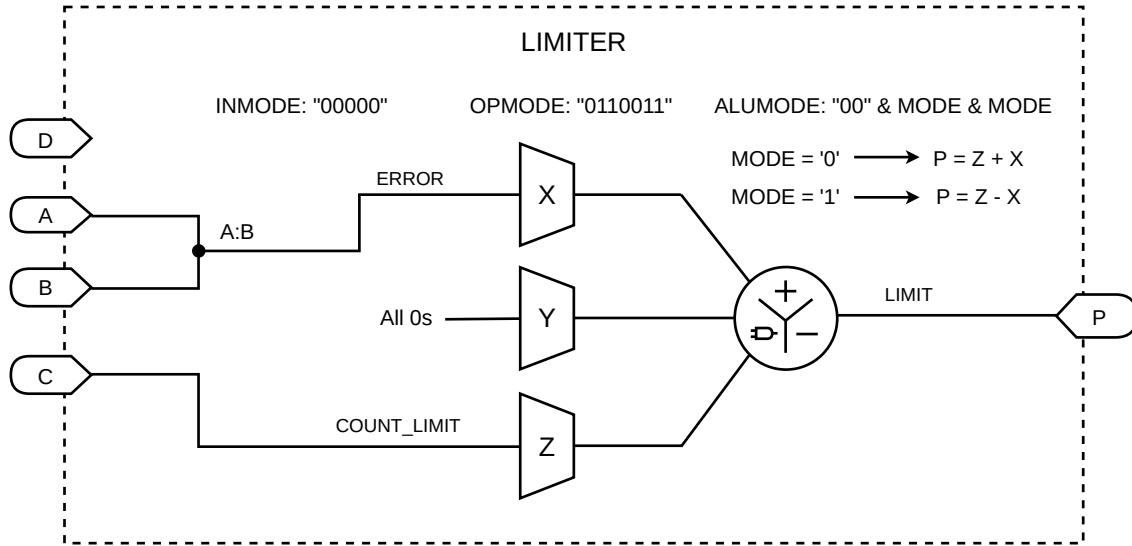


Figura 4.8: Esquemático del bloque limiter.

interno, al bloque CMP2, de forma que ambos conocen el valor de cuenta. Internamente, cada comparador compara el valor de cuenta con el valor de *threshold* o límite que tiene a su entrada, de esta forma, si se diseñan los valores de *threshold*, tal y como se muestra en la figura 4.7, se consigue que el comparador CMP1 se active justo al inicio de cada ciclo y el comparador CMP2 se active justo en la mitad. Estas señales serán CE1 y CE2 respectivamente.

En las figuras 4.10 y 4.11 se pueden ver las configuraciones de los slices DSP correspondientes a los bloques CMP1 y CMP2 respectivamente.

#### 4.3.4. Simulación del bloque Sampleador

Para simular y visualizar gráficamente el funcionamiento del bloque *Sampleador* se van a realizar dos simulaciones, en la primera de ellas el error se mantendrá a cero para poder comprobar de manera más sencilla, la correcta activación de las señales CE1, CE2 y CE3. En la segunda simulación, se asignará un valor diferente de cero a ERROR, para comprobar que la corrección de fase se aplica correctamente.

#### Primera simulación

Para la primera simulación se va a configurar el siguiente escenario:

- $f_{CLK}$ , frecuencia de funcionamiento: 125MHz

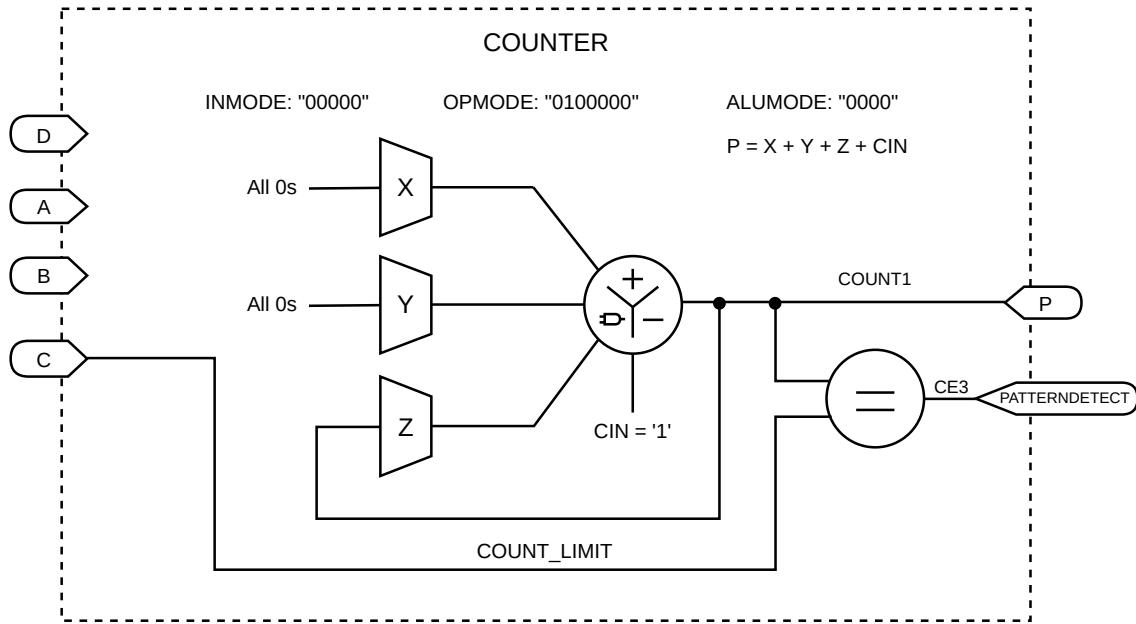


Figura 4.9: Esquemático del bloque counter.

- COUNT \_ LIMIT: 625
- ERROR = 0

Esta configuración está diseñada para un ciclo de cuenta de partida de  $5\mu s$ , equivalente al período de bit de los pulsos que se esperan recibir,  $T_{BIT} = 5\mu s$ .

Configurado el escenario, se analiza la simulación. En la figura 4.12 se pueden diferenciar las siguientes señales en orden: CLK, CE, RST, datos de entrada, CE1, CE2, CE3, y error.

En la simulación se ha forzado un dato de entrada de  $T_{BIT} = 5\mu s$  con la única finalidad de poder apreciarse de forma visual, el período de cuenta con respecto a una señal periódica y de fase constante. Como se puede ver, nada mas desactivarse la señal RST, comienza el ciclo de cuenta y se activa CE1 durante un período de reloj, a mitad de ciclo de cuenta se activa CE2 y finalmente al final del ciclo de cuenta y se activa CE3.

A continuación, se va a comprobar el valor de cuenta para cada una de las activaciones de las señales CE.

En la figura 4.13, se puede observar la activación de CE1 en el instante en que

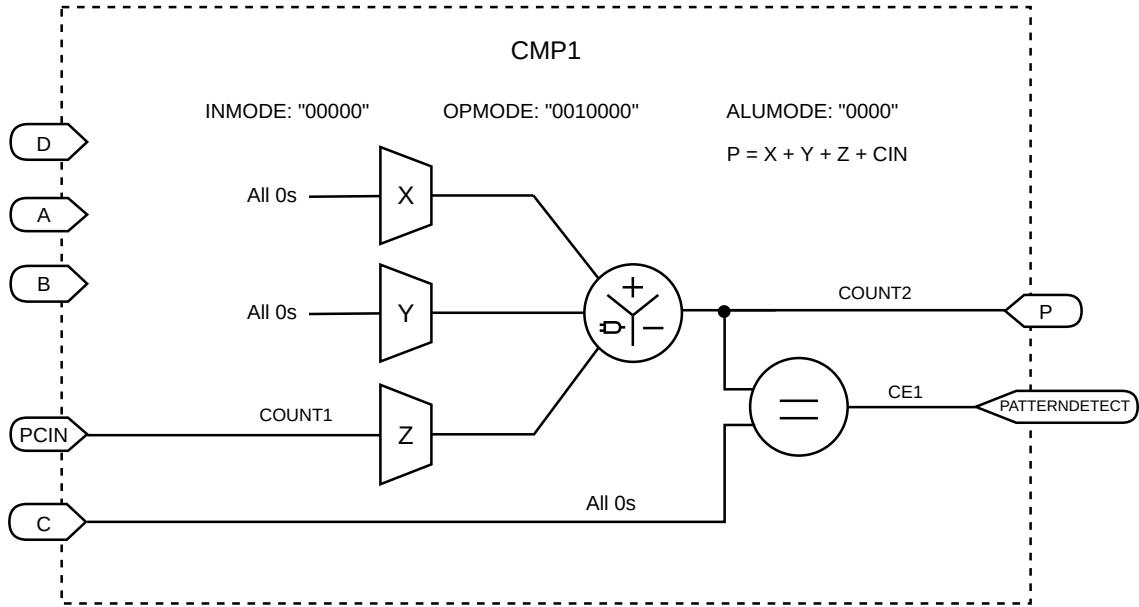


Figura 4.10: Esquemático del bloque cmp1.

el valor de cuenta es cero. La activación de CE3 se produce cuando la cuenta vale 625. Y en la figura 4.14, se puede ver como la activación de CE2 sucede cuando el valor de cuenta es de 312.

## Segunda simulación

Para la segunda simulación se va a configurar el siguiente escenario:

- $f_{CLK}$ , frecuencia de funcionamiento: 125MHz
- COUNT\_LIMIT: 625
- ERROR = 50
- MODE = Gardner

En esta simulación, se va a comprobar la correcta corrección de fase sobre los ciclos de cuenta. Para ello se va a imponer un error constante de 50 y se va a implementar el algoritmo de Gardner, de esta forma, la corrección de fase será negativa.

Observando la figura 4.15, se puede ver como la activación de la señal CE1, se desfase hacia la izquierda en cada ciclo de cuenta y a su vez CE2, CE3. Se ha comprobado que el valor de cuenta en el que se reinicia el contador es de 575, equivalente a  $625 - 50$ , cuyos valores son el límite de cuenta configurado y el error.

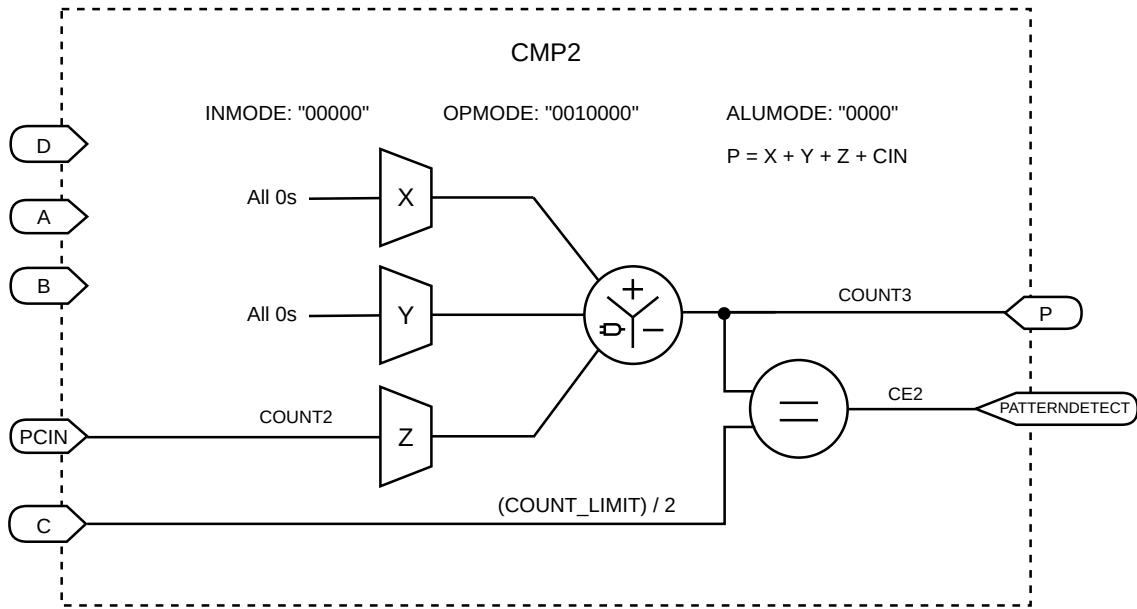


Figura 4.11: Esquemático del bloque cmp2.

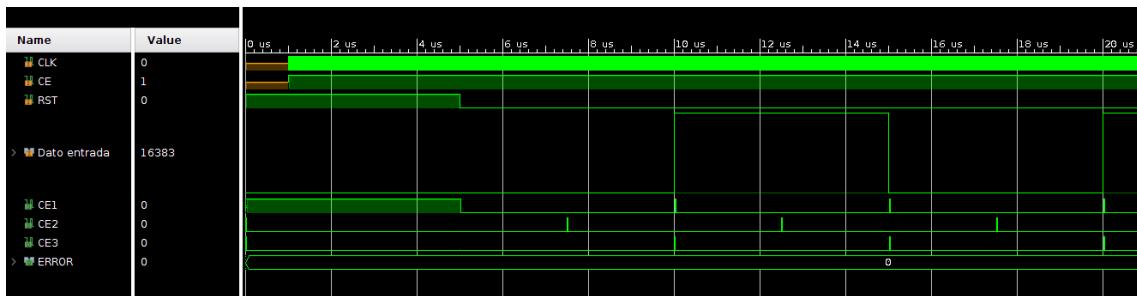


Figura 4.12: Primera simulación del bloque sampleador.

## 4.4. Bloque Error

El bloque error está formado por un único slice DSP y se encarga de calcular el error a partir de las tres muestras capturadas sobre la señal de acumulación del pulso. Implementa la ecuación 3.5, y ofrece a su salida, el resultado de dicha ecuación dividido por un valor potencia de 2 configurable. Este factor, es potencia de 2 para que la división se pueda implementar de forma sencilla como desplazamiento de bits.

En la figura 4.16 aparece el esquemático del bloque error, tan solo presenta las tres muestras como entrada y el error calculado como salida. Las operaciones que realiza son la ecuación del error en el slice DSP y posteriormente el desplazamiento

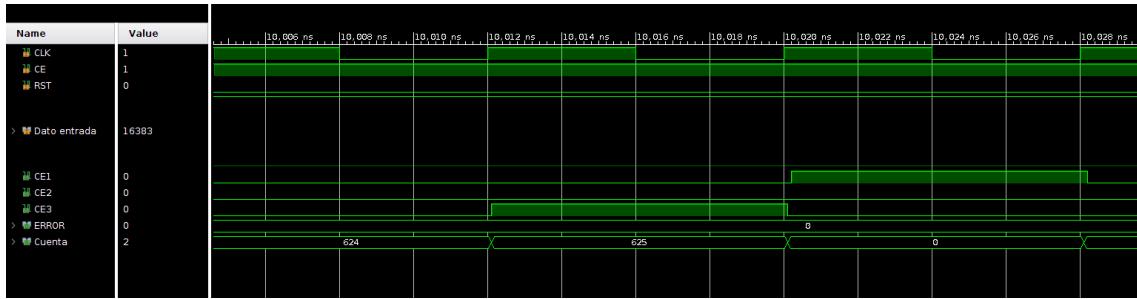


Figura 4.13: Valor de cuenta en la activación de CE1 y CE3 para la primera simulación del bloque sampleador.

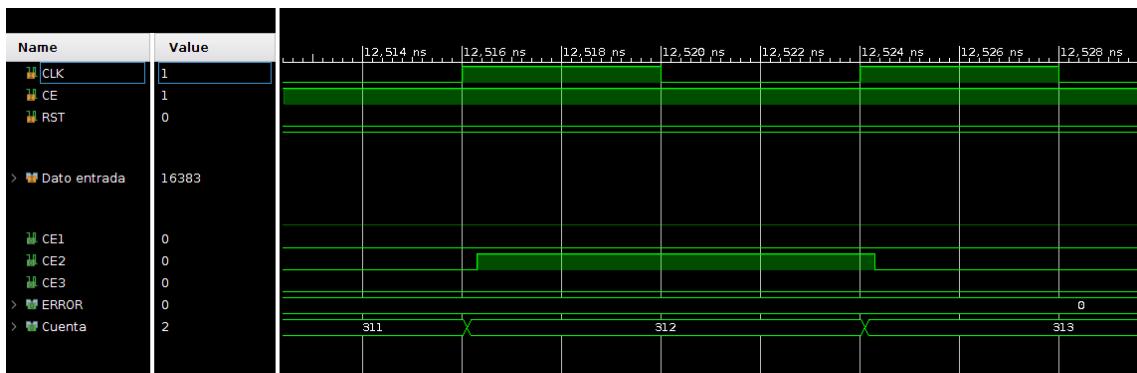


Figura 4.14: Valor de cuenta en la activación de CE2 para la primera simulación del bloque sampleador.

para dividir. Es interesante explicar, que para generalizar el valor de division  $DIV$ , se ha utilizado la ecuación 4.7. Esta ecuación se ha deducido mediante prueba y error, en diferentes simulaciones para datos en recepción de 200KHz y 5MHz, que son las frecuencias mínima y máxima establecidas por el estándar.

$$DIV = 26 - \log_2(\text{orden}/4); \quad (4.7)$$

#### 4.4.1. Simulación del bloque Error

En la figura 4.18, aparece un ejemplo de simulación del bloque *Error*. Se pueden ver los valores de las entradas S1, S2 y S3, a los que si se les aplica la ecuación 3.5, devuelve el valor que presenta la salida P. Si esta salida P se divide por  $2^{26-7}$ , se obtiene 6, que es el error normalizado para ser capturado a la entrada del *Sampleador*.

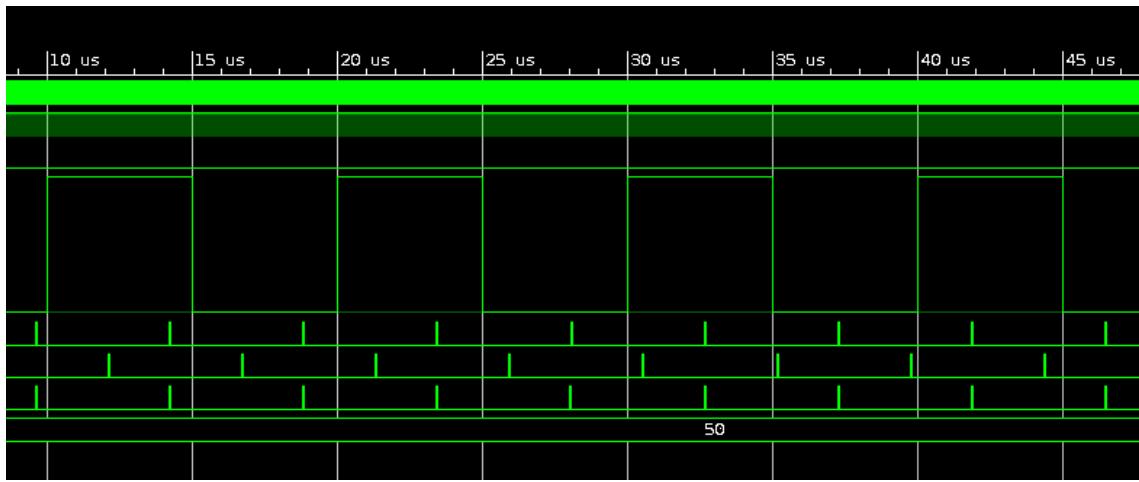


Figura 4.15: Segunda simulación del bloque sampleador.

## 4.5. Simulación del Sistema

Para la simulación del sistema completo se simulará el mismo escenario que para las simulaciones anteriores, pero esta vez, se mostrarán las señales más importantes de cada bloque al mismo tiempo, para representar la relación entre cada una de ellas. Las señales que se van a representar son, por orden: CLK, CE, RST, dato entrada, acumulación, CE1, CE2, CE3, salida digital, salida analógica y error.

En la figura 4.19, aparece el transitorio de la simulación nada mas desactivarse la señal RST. Se puede observar la correcta acumulación de los pulsos y se puede observar un claro desfase entre los picos de la señal acumulada y el trio de señales de captura, CE1, CE2 y CE3. Conforme el sistema va iterando, se va corrigiendo el error de fase, y se puede observar como la amplitud de la salida analógica, que representa la salida del filtro de forma directa, es decir, sin el decisor, va aumentando. El error válido es el capturado en los flancos de CE2, por tanto, comienza en  $-121$ , y va sucediendo:  $-42, -29, 14, 43$ . Se puede apreciar una ligera convergencia, aunque un poco oscilante.

En la figura 4.20, ya han pasado unos  $100\text{ }\mu\text{s}$ , equivalente a unos 20 bits de  $T_{BIT} = 5\text{ }\mu\text{s}$ . Se puede observar como el trio de señales de captura CE1, CE2 y CE3 ya están correctamente sincronizados con la señal de acumulación de salida del integrador. Los valores de error que se calculan ahora son:  $-3, -6, 0, 1, 5, 0, 0$ . El error ha convergido a cero y se ha conseguido satisfactoriamente la sincronización.

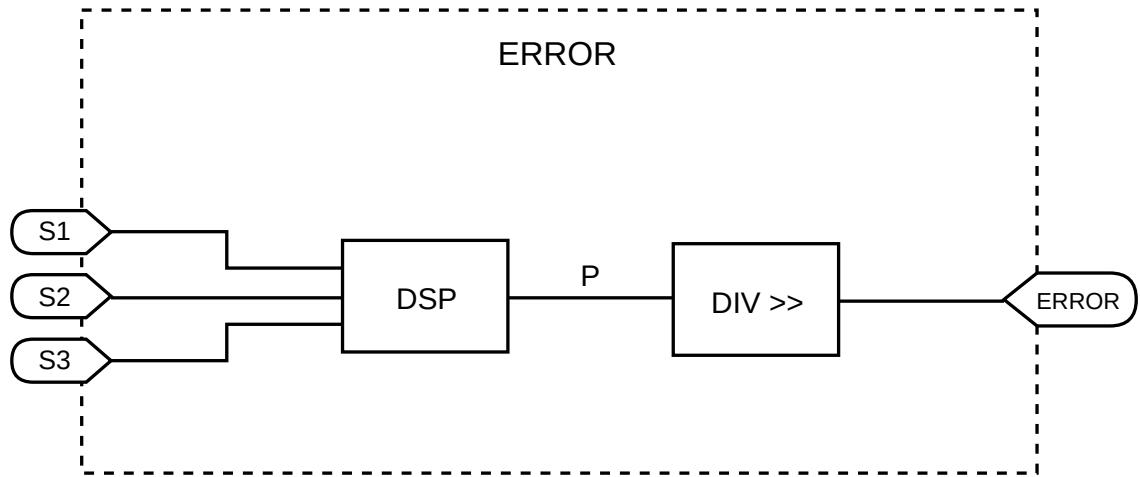


Figura 4.16: Esquemático del bloque error.

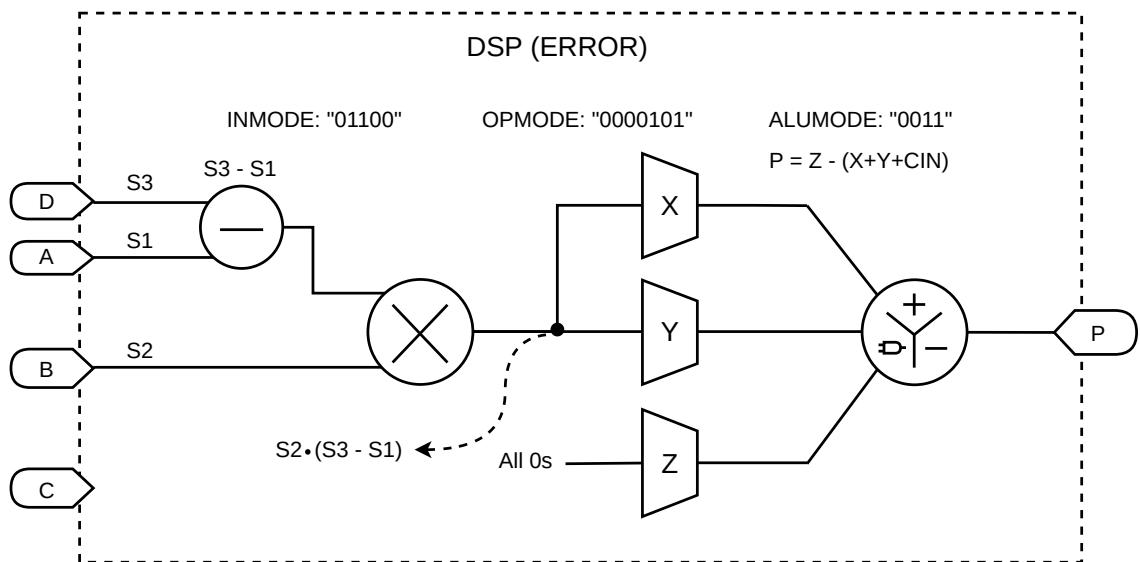


Figura 4.17: Esquemático del slice DSP del bloque error.



Figura 4.18: Simulación del bloque error.

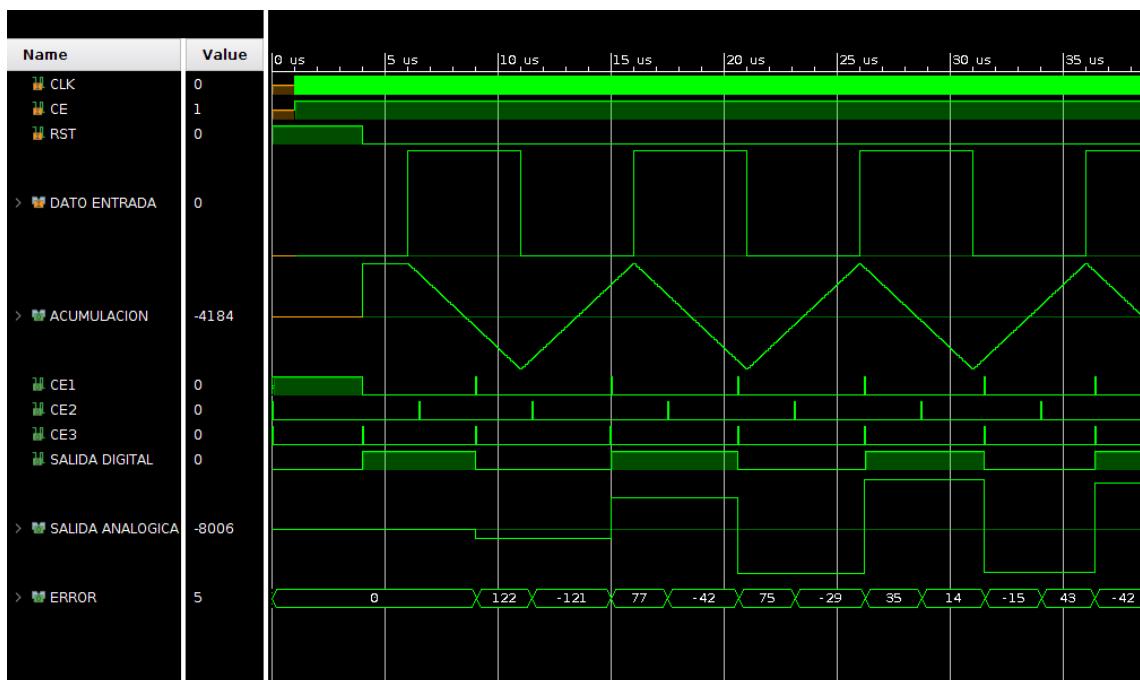


Figura 4.19: Simulación del sistema completo, etapa transitoria.

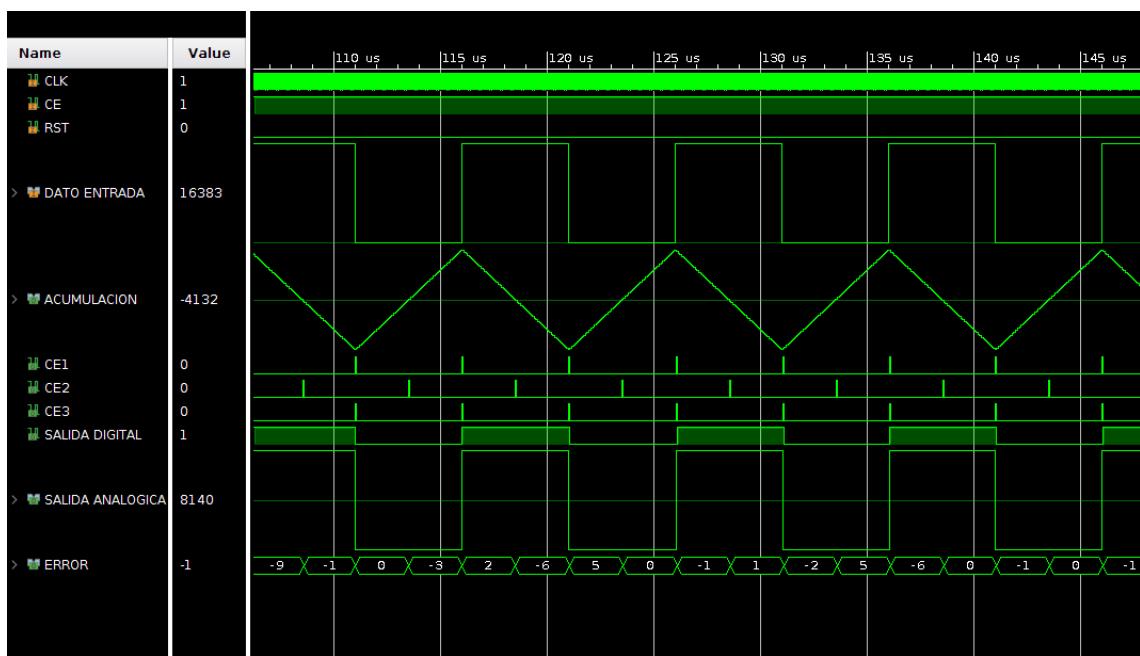


Figura 4.20: Simulación del sistema completo, etapa estable.



# Parte III

## Pruebas



# Capítulo 5

## Pruebas

### Contenido

---

<b>5.1 Recuperación de tren de pulsos . . . . .</b>	<b>67</b>
5.1.1 Comparativa transmisor prototipo en placa y faro de vehículo . . . . .	68
5.1.2 Alcance del faro de vehículo . . . . .	70
5.1.3 Directividad . . . . .	72
5.1.4 Resumen de resultados . . . . .	74
<b>5.2 Integración del filtro adaptado en el sistema de comunicación . . . . .</b>	<b>76</b>

---

En este capítulo se van a presentar diferentes pruebas para comprobar el correcto funcionamiento del filtro adaptado de forma independiente, así como sus capacidades en cuanto a alcance y directividad. Además se integrará y probará su funcionamiento conjuntamente en el sistema de comunicación.

### 5.1. Recuperación de tren de pulsos

Esta prueba consiste en transmitir de forma constante un tren de pulsos cuadrados de frecuencia 100kHz y amplitud  $\pm 1V$  equivalente una trama de sincronización frecuencia de bit 200kHz y comprobar en el extremo de recepción que la señal de salida del filtro adaptado coincide con la señal en transmisión. Para ello, será necesario mostrar por osciloscopio las señales de transmisión, recepción y salida del filtro adaptado.

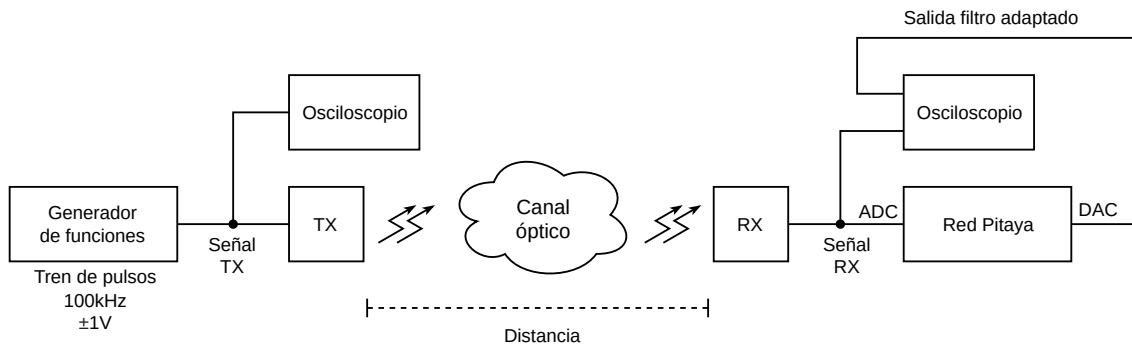


Figura 5.1: Configuración de los equipos para la ejecución de la prueba 1: Recuperación de tren de pulsos.

Se van a configurar los equipos tal como se representa en la figura 5.1.

Además de esto, se compararán dos transmisores diferentes, uno montado en placa diseñado para pruebas, mostrado en la figura 5.2b y 5.2c y otro integrado en un faro de vehículo, mostrado en la figura 5.2d. El receptor utilizado se muestra en la figura 5.2a.

Para todos los experimentos de esta prueba, la señal en transmisión va a ser la señal mostrada en la figura 5.3.

### 5.1.1. Comparativa transmisor prototipo en placa y faro de vehículo

En esta sección se van a medir y comparar los dos dispositivos transmisores en cuanto a amplitud de señal recibida en el receptor.

#### Primer experimento

En el primer experimento se han colocado los transceptores a una distancia de 2 metros y con un ángulo de desapuntamiento de  $0^\circ$ . En la figura 5.4a, se muestra la recepción con el transmisor de pruebas montado y la amplitud media recibida es de 95mV. En la figura 5.4b, se muestra la recepción con el transmisor integrado en el faro y la amplitud media recibida es de 878mV.

La amplitud recibida es aproximadamente 9 veces mayor con el transmisor integrado en el faro.

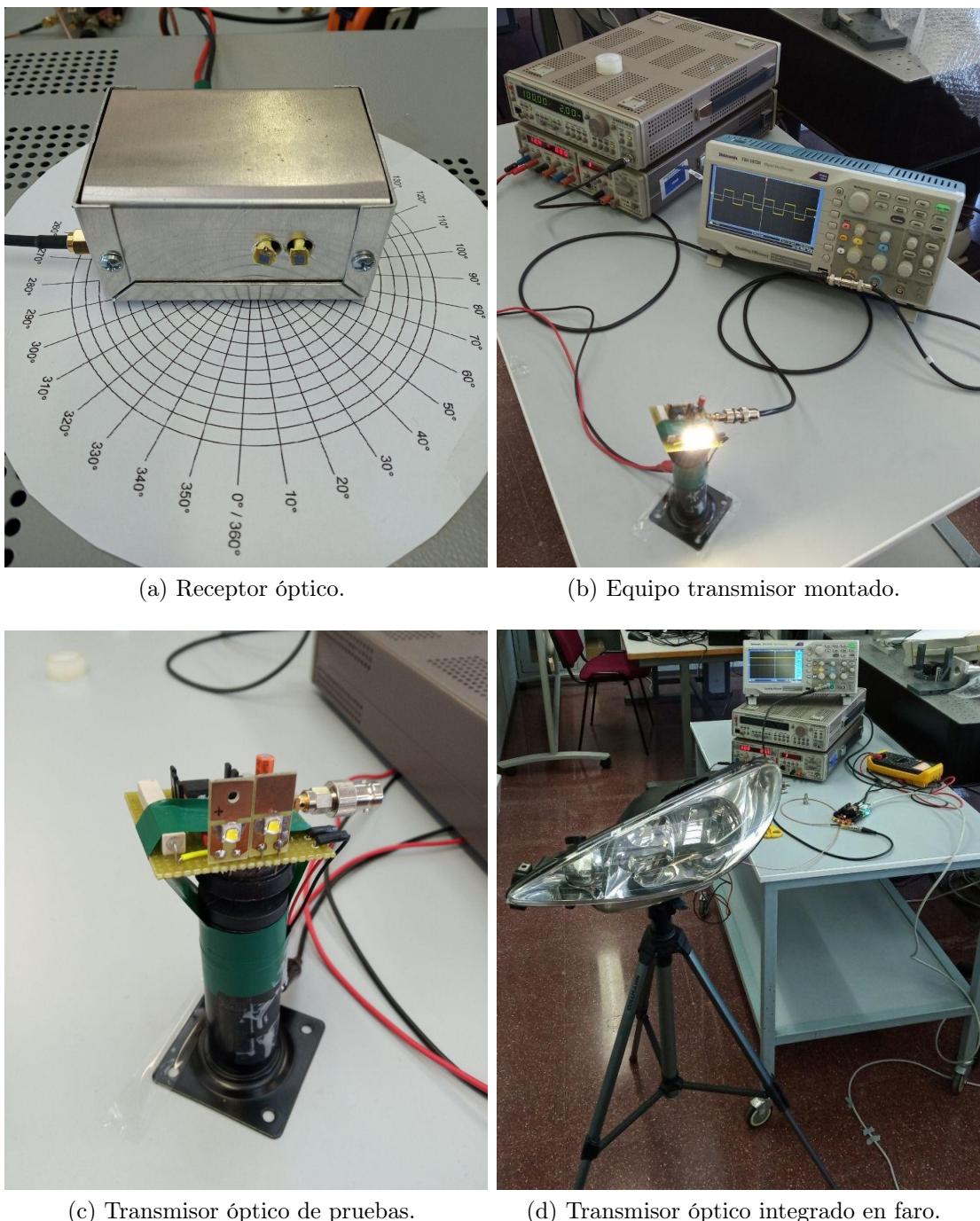


Figura 5.2: Dispositivos transmisores y receptor.

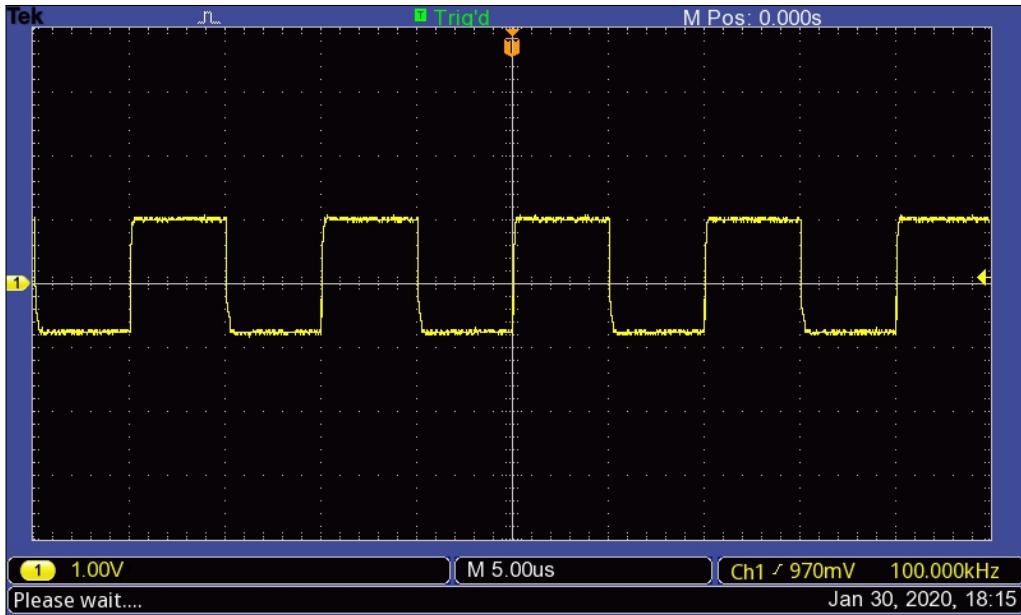


Figura 5.3: Señal en transmisión, común a todos los experimentos de esta prueba.

## Segundo experimento

En el segundo experimento se han colocado los transceptores a una distancia de 2 metros y con un ángulo de desapuntamiento de  $60^\circ$ . En la figura 5.5a, se muestra la recepción con el transmisor de pruebas montado y la amplitud media recibida es de 40mV. En la figura 5.5b, se muestra la recepción con el transmisor integrado en el faro y la amplitud media recibida es de 368mV.

La amplitud recibida vuelve a ser aproximadamente 9 veces mayor con el transmisor integrado en el faro.

Con esta prueba se ha podido comprobar como el transmisor integrado en faro mejora en un factor de 9 la amplitud recibida respecto del transmisor de pruebas montado en placa. Esto permitirá aumentar notablemente las capacidades del sistema de comunicación.

### 5.1.2. Alcance del faro de vehículo

En esta sección se va a montar el transmisor integrado en el faro y se va a medir la amplitud de la señal recibida en el receptor para diferentes distancias de enlace. El ángulo de desapuntamiento va a ser siempre  $0^\circ$  puesto que únicamente se pretende

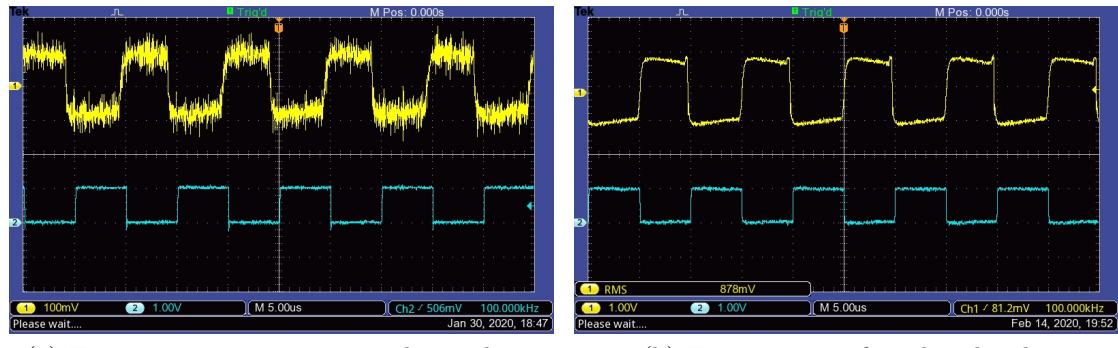


Figura 5.4: Señal en recepción (canal superior) y salida del filtro adaptado (canal inferior). Distancia de 2 metros y desapuntamiento de  $0^\circ$ .

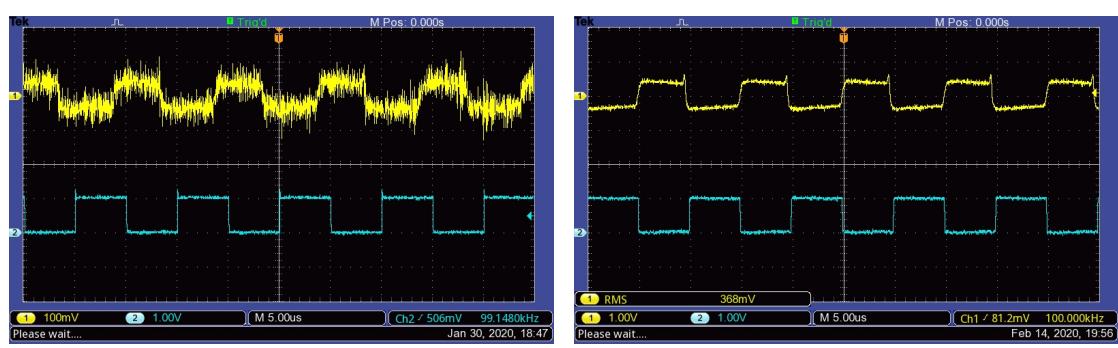


Figura 5.5: Señal en recepción (canal superior) y salida del filtro adaptado (canal inferior). Distancia de 2 metros y desapuntamiento de  $60^\circ$ .

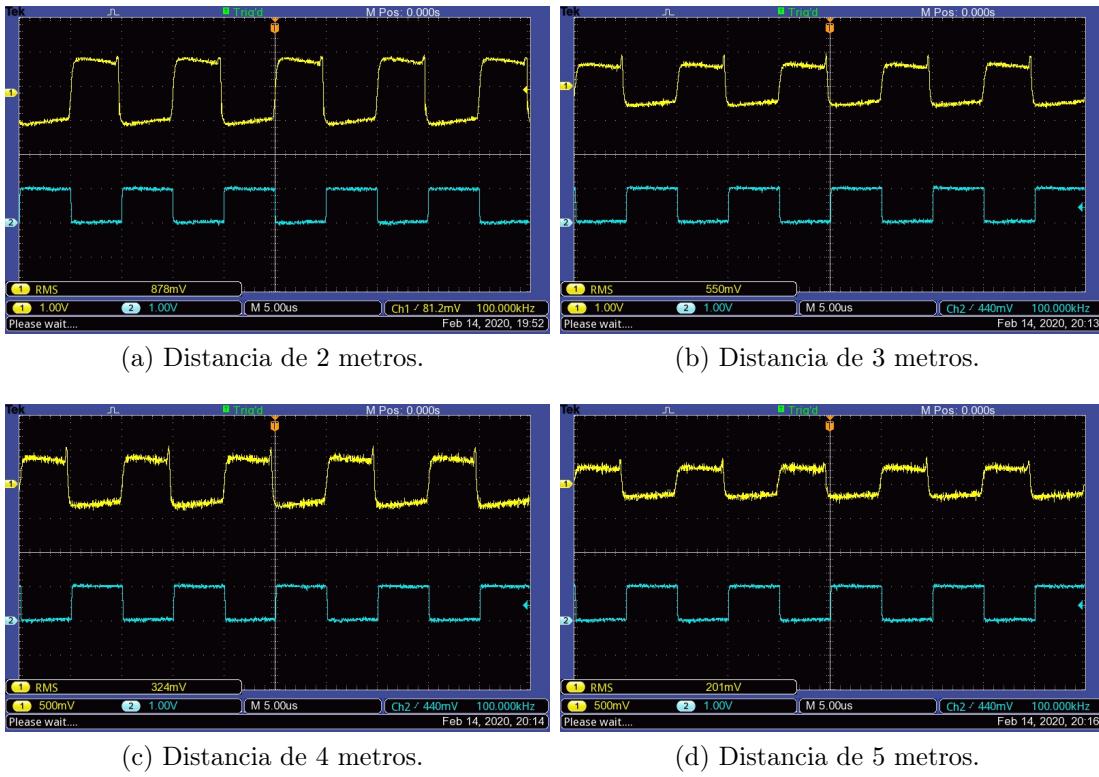


Figura 5.6: Señal en recepción (canal superior) y salida del filtro adaptado (canal inferior) para diferentes distancias entre transmisor y receptor con  $0^\circ$  de ángulo de desapuntamiento.

medir el alcance.

### Tercer experimento

En el tercer experimento se han colocado los transceptores a distancias de 2, 3, 4 y 5 metros. En la figura 5.6, se pueden ver las diferentes señales recibidas en recepción.

Las amplitudes de señal recibida para 2m, 3m, 4m y 5m respectivamente son de 878mV, 550mV, 324mV y 201mV.

#### 5.1.3. Directividad

En esta sección se va a montar el transmisor integrado en el faro y se va a medir la amplitud de la señal recibida en el receptor para diferentes ángulos de desapuntamiento. La distancia va a ser siempre 2 metros puesto que únicamente se pretende

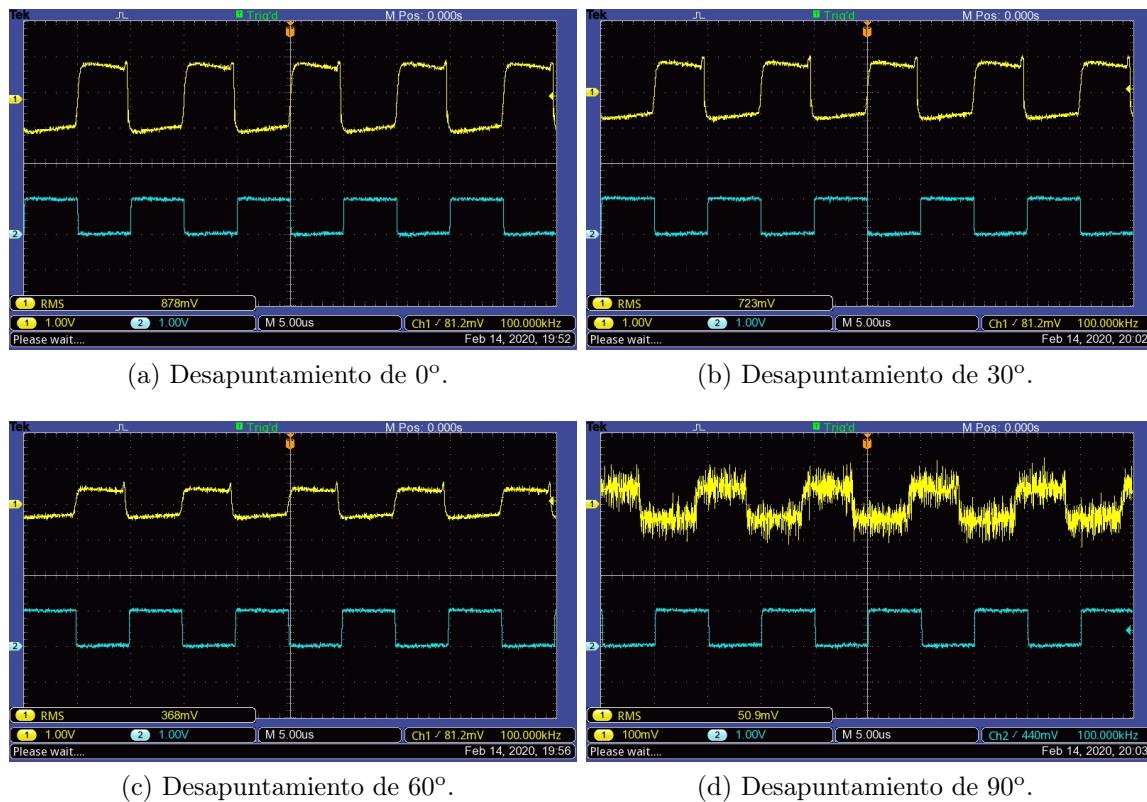


Figura 5.7: Señal en recepción (canal superior) y salida del filtro adaptado (canal inferior) para diferentes ángulos de desapuntamiento y 2 metros de distancia entre transmisor y receptor.

medir la directividad.

## Cuarto experimento

En el cuarto experimento se han colocado los transceptores con ángulos de desapuntamiento de  $0^\circ$ ,  $30^\circ$ ,  $60^\circ$  y  $90^\circ$ . En la figura 5.7, se pueden ver las diferentes señales recibidas en recepción.

Las amplitudes de señal recibida para  $0^\circ$ ,  $30^\circ$ ,  $60^\circ$  y  $90^\circ$  respectivamente son de 878mV, 723mV, 368mV y 50.9mV.

### 5.1.4. Resumen de resultados

En esta sección, se muestran de forma resumida los resultados de los cuatro experimentos realizados anteriormente y además se van a calcular las relaciones señal a ruido para cada uno de ellos.

Para calcular la relación señal a ruido se debe medir la amplitud media de ruido en cada uno de los experimentos sin embargo, independientemente de la configuración de cada experimento, la amplitud de ruido permanece constante y entre unos valores aproximados de 20mV y 40mV. En la figura 5.8, se puede apreciar la señal de ruido y su amplitud media calculada por el osciloscopio. Se ha optado por tanto, por utilizar 40mV como valor de referencia de amplitud de ruido para todos los experimentos.

De esta forma, el cálculo de la relación señal a ruido o SNR se puede hacer con la ecuación 5.1.

$$SNR = 20 \cdot \log_{10} \left( \frac{V_{RMS_{señal}}}{V_{RMS_{ruido}}} \right) \quad (5.1)$$

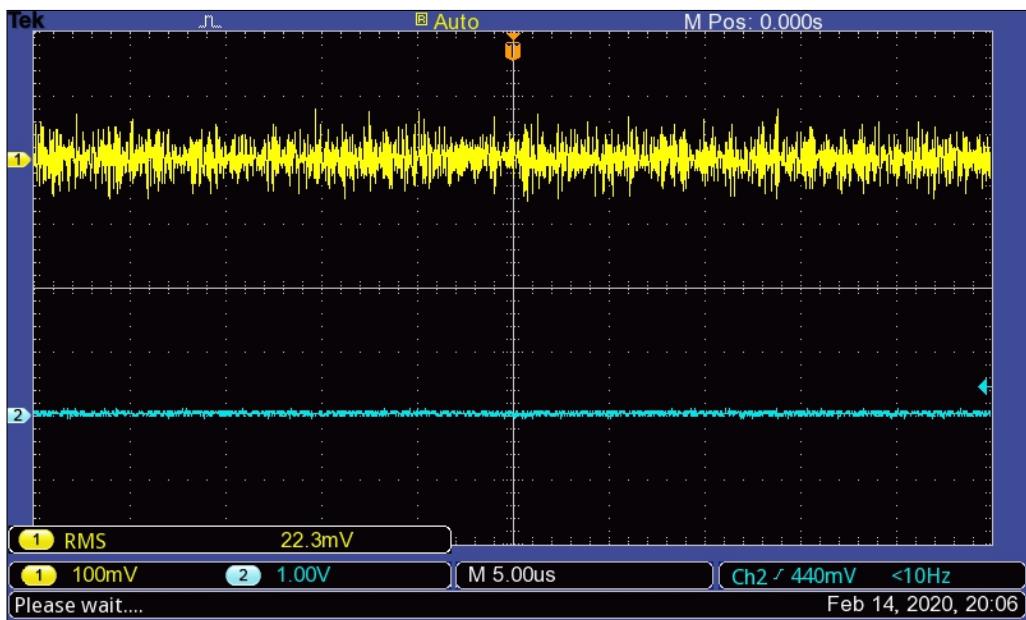


Figura 5.8: Señal de ruido capturada por el osciloscopio.

Distancia / Desapuntamiento	2m / 0°	2m / 60°	Unidad
Amplitud RMS en recepción con transmisor en placa	95	40	mV
Relación señal a ruido en recepción	7.5	0	dB
Amplitud RMS en recepción con transmisor en faro	878	368	mV
Relación señal a ruido en recepción	26.8	19.3	dB

Tabla 5.1: Resultados obtenidos por los experimentos 1 y 2.

Distancia	2m	3m	4m	5m	Unidad
Amplitud RMS en recepción	878	550	324	201	mV
Relación señal a ruido en recepción	26.8	22.8	18.2	14.0	dB

Tabla 5.2: Resultados obtenidos por el experimento 3.

Desapuntamiento	0°	30°	60°	90°	Unidad
Amplitud RMS en recepción	878	723	368	51	mV
Relación señal a ruido en recepción	26.8	25.1	19.3	7.1	dB

Tabla 5.3: Resultados obtenidos por el experimento 4.

## 5.2. Integración del filtro adaptado en el sistema de comunicación

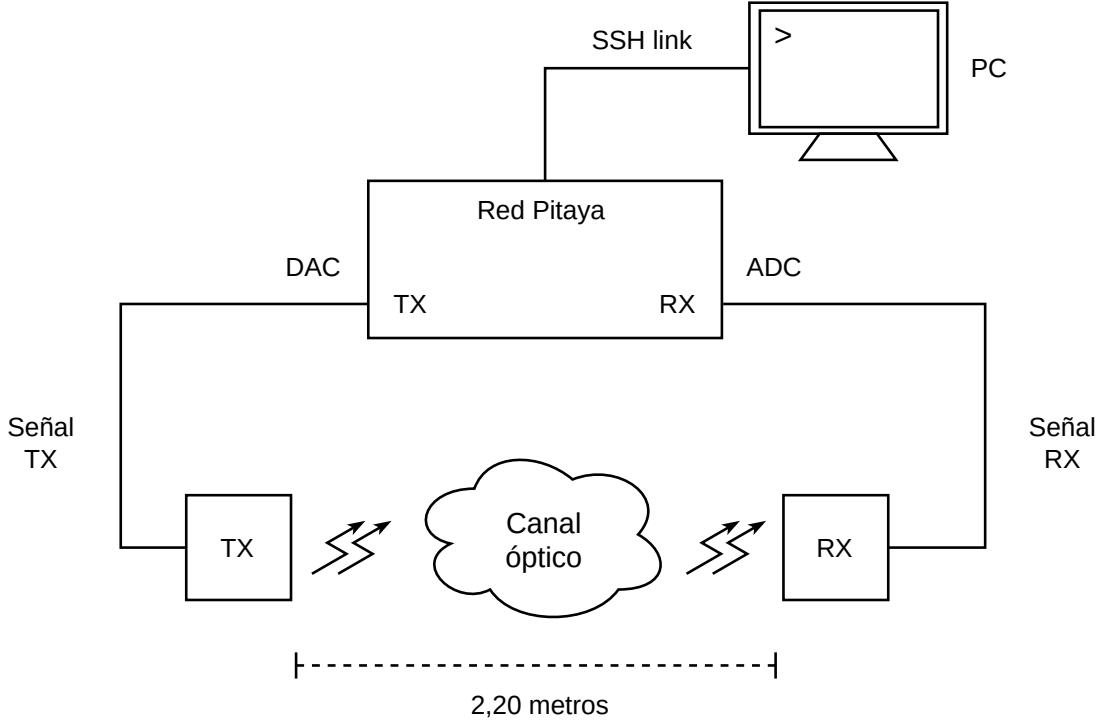


Figura 5.9: Configuración de los equipos para la ejecución de la prueba 2: Integración del filtro adaptado en el sistema de comunicación.

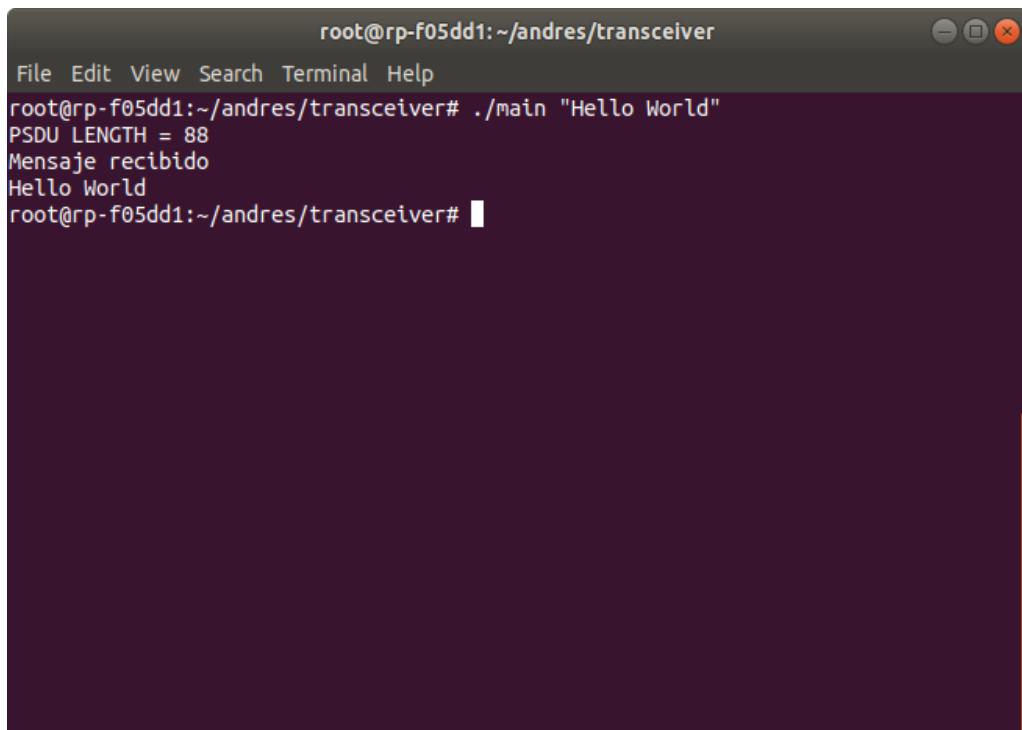
Esta prueba consiste en integrar el filtro adaptado en el sistema de comunicación ya existente y realizar transmisiones de mensajes de texto para comprobar que ambos sistemas funcionan conjuntamente. En la figura 5.9, se puede ver la configuración de los equipos para la ejecución de esta prueba.

El objetivo de la prueba únicamente es comprobar que ambos sistemas funcionan correctamente de forma conjunta y que en recepción se recibe el mensaje transmitido, por lo tanto, no se harán pruebas de alcance ni directividad.

En la figura 5.10, aparece el terminal de una sesión SSH desde un PC remoto en el que se controla el ARM de la placa y se ejecuta un programa en lenguaje C, con un mensaje como parámetro. El programa comunica al transmisor tanto el mensaje, como la orden de transmitir y además lee posteriormente lo que se ha recibido. De esta forma, en la figura se ve como se ha ejecutado el programa con el mensaje *Hello*

*World*, y el mensaje recibido es correcto.

Con esta prueba se ha verificado la correcta integración del filtro adaptado en el sistema de comunicación y además se ha conseguido realizar transmisiones exitosas a una distancia de 2.20 metros.



The screenshot shows a terminal window titled "root@rp-f05dd1: ~/andres/transceiver". The window has a dark background and a light-colored title bar. The terminal menu bar includes "File", "Edit", "View", "Search", "Terminal", and "Help". The command entered is "root@rp-f05dd1:~/andres/transceiver# ./main \"Hello World\"". The output shows "PSDU LENGTH = 88", "Mensaje recibido", and "Hello World". The terminal prompt "root@rp-f05dd1:~/andres/transceiver# " is visible at the bottom.

```
root@rp-f05dd1:~/andres/transceiver
File Edit View Search Terminal Help
root@rp-f05dd1:~/andres/transceiver# ./main "Hello World"
PSDU LENGTH = 88
Mensaje recibido
Hello World
root@rp-f05dd1:~/andres/transceiver# 
```

Figura 5.10: Terminal de la sesión SSH en el PC remoto desde el que se controla el ARM de la placa, transmitiendo y recibiendo correctamente el mensaje *Hello World*.



# Parte IV

## Conclusiones



# Conclusiones y líneas futuras

Tras la realización de este trabajo se han comprendido en profundidad los fundamentos del filtro adaptado para comunicaciones digitales, se ha profundizado enormemente en la estructura interna de los dispositivos FPGA, a la vez que se ha aprendido a programarlos en un mayor grado de complejidad, utilizando y estudiando módulos como los DSP o MMCM.

Se han aumentado notablemente las capacidades a la hora de trabajar en entornos basados en sistemas empotrados, así como la propia FPGA y el ARM del chip Zynq de Xilinx. También se ha mejorado en el uso de dispositivos conversores analógico-digital y digital-analógico y a su vez se ha aprendido a implementar sobre FPGA, múltiples operaciones aritméticas a nivel de bit como por ejemplo, operaciones en punto flotante o tratamiento de variables con dominios bipolares, como lo son las señales que se han debido tratar. La principal fuente de problemas y errores a la que se ha tenido que hacer frente ha sido la programación del dispositivo FPGA y la comprobación del correcto funcionamiento de cada uno de los bloques que se han diseñado, sin duda ha sido el trabajo más duro.

Tras este trabajo, el sistema de comunicación queda con unas muy buenas capacidades, utilizables para ser integrado en vehículos y ser testeado. Pese a ello, hay numerosas mejoras que pueden añadirse y que se llevarán a cabo en el futuro. Como puede serlo un filtrado paso alto de la señal capturada por los conversores. Los transceptores ópticos están en pleno desarrollo y cada vez son mejores pero deben seguir mejorando, por ejemplo, el transceptor receptor aun no implementa el control de ganancia programable que se estudiaba en el capítulo del estudio del filtro adaptado. Además el sistema de comunicación debe ser mejorado con el objetivo de permitir diferentes tipos de comunicación, a diferentes velocidades y de simplificar su programación y abstraerla del bajo nivel de la lógica transmisora y receptora que implementaría el nivel más bajo de la comunicación.

Andrés Casasola Domínguez  
19 de febrero de 2020



# Parte V

## Apéndices



# Apéndice A

## Contenido

---

<b>A.1 Inicialización Red Pitaya . . . . .</b>	<b>85</b>
A.1.1 Configuración de Vivado . . . . .	85
A.1.2 Configuración de Linux en ARM . . . . .	86
A.1.3 Programación de la FPGA . . . . .	88
A.1.4 Puertos de la Red Pitaya . . . . .	89
<b>A.2 IP Cores de Xilinx . . . . .</b>	<b>91</b>
A.2.1 Processing System . . . . .	91
A.2.2 Clock Wizard . . . . .	91
<b>A.3 Periféricos de la placa . . . . .</b>	<b>92</b>
A.3.1 Conversor Analógico a Digital (ADC) . . . . .	92
A.3.2 Conversor Digital a Analógico (DAC) . . . . .	92

---

## A.1. Inicialización Red Pitaya

### A.1.1. Configuración de Vivado

Para poder trabajar con la placa *Red Pitaya* en Vivado, deberemos añadir una carpeta con archivos *xml* que describen el hardware de la *Red Pitaya*. Esta carpeta se encuentra en el repositorio *Github* oficial de *Red Pitaya*. Se puede obtener con el siguiente comando:

```
$ git clone https://github.com/RedPitaya/RedPitaya
```

Última comprobación: Marzo 2019

A continuación, se busca la carpeta con la siguiente ruta:

```
RedPitaya/fpga/brd/redpitaya
```

Y se copia en la siguiente ruta de los archivos de Vivado:

```
/opt/Xilinx/Vivado/2018.2/data/boards/board_files
```

Se puede utilizar el siguiente comando para realizar la operación:

```
sudo cp -r <ruta origen> <ruta destino>
```

Es necesario en este caso, ejecutar como *sudo*, puesto que la ruta donde está instalado Vivado solo es accesible por usuarios administradores. Además la opción *-r* ejecuta la operación de copiado de forma recursiva, permitiendo así, copiar todos los archivos contenidos.

En este caso Vivado estaba instalado en la ruta /opt/Xilinx.

A partir de este momento, cuando se ejecute Vivado y se cree un proyecto, aparecerá la placa Red Pitaya en la sección de *Boards*, tal y como se puede apreciar en la figura A.1.

### A.1.2. Configuración de Linux en ARM

Para cargar el sistema operativo en la Red Pitaya será necesario una tarjeta micro SD de al menos 4GB. El proceso descrito a continuación, se puede realizar con una gran cantidad de sistemas operativos diferentes, pero en este caso se utilizará el proporcionado por la propia compañía Red Pitaya, que es una versión de Ubuntu 16.04 configurada para funcionar en la placa. Se puede descargar desde su página oficial.

Antes de nada, se debe formatear la tarjeta SD.

A continuación, se desmonta la unidad con el siguiente comando:

```
sudo umount /dev/mmcblk0p1
```

En este caso, el dispositivo tenía el nombre *mmcblk0p1*. Se puede visualizar el uso del espacio en disco con el comando *df -h*, para poder ver la partición de la micro SD.

Ahora, se debe localizar el archivo con formato *img*, que contiene la imagen del sistema operativo que se quiere instalar. Una vez encontrado se ejecuta el siguiente comando para cargarlo en la micro SD:

```
sudo dd bs=1M if=<image_name.img> of=</dev/device_name>
```

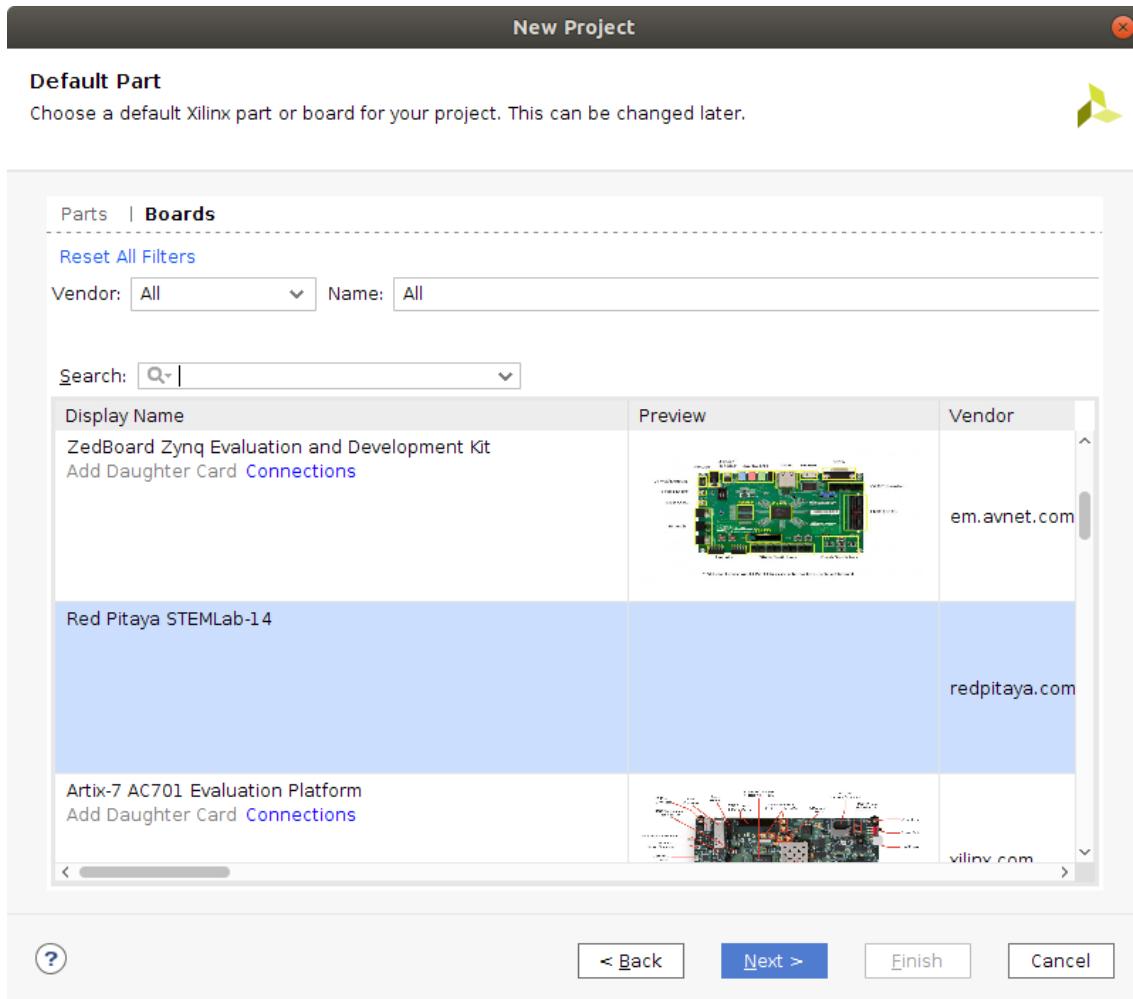


Figura A.1: Ventana de selección de *Boards* en la creación de un proyecto en Vivado.

Este comando funciona en *Linux* y ejecuta un copiado del archivo introducido en *if* (*Input file*) al destino introducido como *of* (*Output file*). La opción *bs=1M* realiza la copia en paquetes de 1MB.

Una vez realizados estos pasos, la micro SD estará lista para ser introducida en la Red Pitaya, arrancándose así el sistema operativo cada vez que se alimente la placa. En este punto, hay dos formas de poder comunicarse con el sistema operativo, mediante UART y mediante sesión SSH.

### Comunicación mediante UART

En la figura 1.9 se puede ver el puerto micro USB denominado *USB console*, que es el que se debe usar para la comunicación serie. Para establecer la comunicación se puede utilizar cualquier programa que pueda establecer una conexión serie. Por ejemplo *gtkterm*:

```
$ gtkterm -s 115200 -p <puerto>
```

Por defecto, la Red Pitaya se comunica a una velocidad de 115200 baudios.

### Comunicación mediante SSH

Para iniciar una sesión por SSH, será necesario conectar la Red Pitaya mediante ethernet a la misma red local del ordenador desde el que se quiera acceder. Una vez hecho esto, deberá ejecutarse un comando similar al que procede, pero cambiando la dirección del dispositivo:

```
$ ssh root@rp-f05dd1.local
```

*rp-f05dd1.local* es un ejemplo de dirección MAC, esta dirección se puede ver en la pegatina de la Red Pitaya ubicada en la posición marcada de la figura A.2. Una vez ejecutado, la Red Pitaya exigirá una contraseña para acceder al usuario *root*, la contraseña es *root*.

### A.1.3. Programación de la FPGA

La placa Red Pitaya tiene una pequeña desventaja, y es que no permite programar la FPGA mediante el protocolo JTAG a través del puerto micro USB, como si lo permiten otras placas de Digilent o Xilinx. El principal contratiempo de esto, es que no se puede utilizar Vivado, como herramienta para cargar el *bitstream* a la FPGA sin la utilización de algún programador intermedio. Por ello se ha debido buscar un método alternativo para poder cargar el *bitstream* a la lógica programable.

La solución ha sido por tanto, cargar los *bitstream* a través del sistema operativo funcionando en el ARM. El sistema operativo que proporciona Red Pitaya tiene configurado un dispositivo llamado */xdevcfg* que representa la interfaz de programación de la FPGA desde el sistema operativo.

Para cargar por tanto, un *bitstream* en la FPGA, se puede ejecutar el siguiente comando:

```
$ cat bitstream.bit > /dev/xdevcfg
```

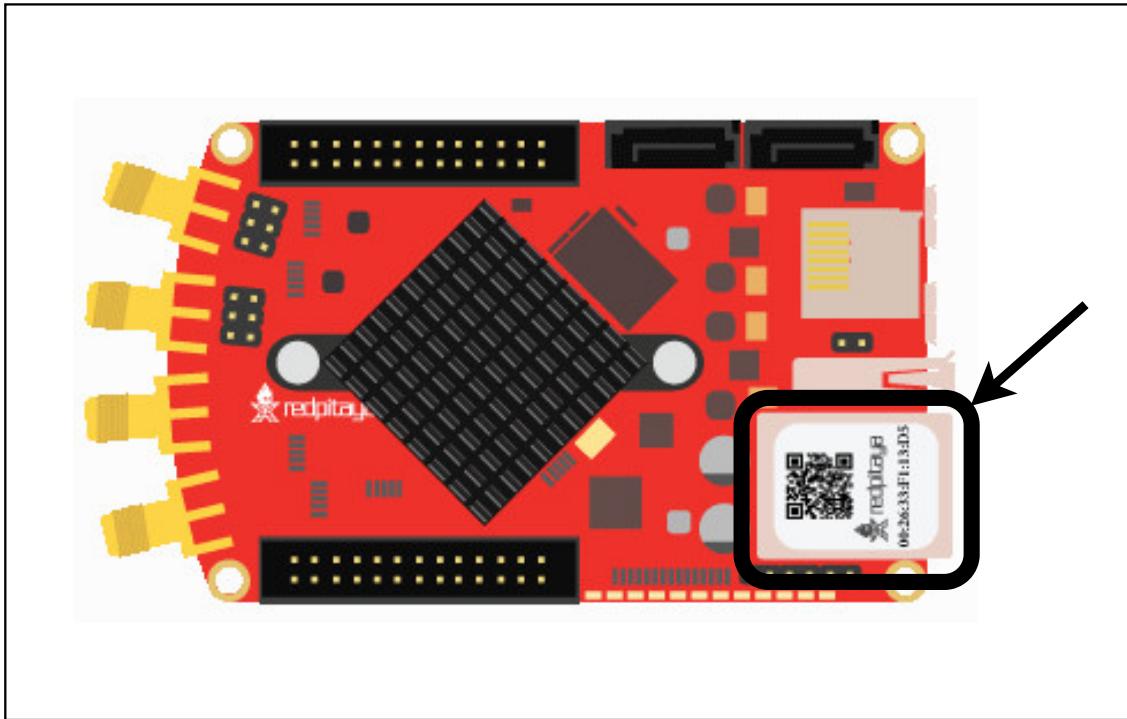


Figura A.2: Dirección MAC de la Red Pitaya.

El comando *cat* imprime o manda un fichero hacia una salida, que por defecto es la pantalla, pero si volcamos la operación sobre el fichero */dev/xdevcfg* con el operador *>*, se estaría volcando en el *driver* o controlador de la FPGA.

Hay varias formas de extraer el bitstream de un proyecto compilado con Vivado. Una forma es configurando Vivado para que genere un bitstream de salida en una ruta especificada por el usuario. La otra es copiandolo directamente desde la carpeta donde Vivado guarda los archivos que genera cuando compila el proyecto. La carpeta donde se encuentran estos archivos se llama *nombre\_proyecto.runs* y se encuentra en la carpeta donde se ha guardado el proyecto.

#### A.1.4. Puertos de la Red Pitaya

Para conectar los pines de interfaz E/S de la FPGA a los periféricos externos que hay en la placa, Vivado necesita un archivo de extensión *xdc* denominado *constraints* o restricciones, que asigna a cada pin creado en el diseño, un pin físico del chip y además le asigna un estándar de tensión. El nombre de los pines de cada periférico se puede encontrar en los esquemáticos eléctricos de la placa, proporcionados por Red Pitaya. Sin embargo, algunos desarrolladores *open-source* han creado y com-

partido estos archivos *xdc* en *github* con todos los periféricos de la placa descritos [3].

En el código que se muestra a continuación, se definen los puertos y estandáres de tensión para el reloj de entrada del periférico ADC de la placa. En la figura A.3 se puede observar una parte del documento de los esquemáticos eléctricos en el que aparecen los pines del reloj del ADC y sus correspondientes nombres, que a su vez, aparecen definidos en el código. En el código, además, aparecen definidos con los estándares de tensión *DIFF\_HSTL\_I\_18* que representa una conexión diferencial del estándar HSTL (*High Speed Transceiver Logic*) de 1.8V.

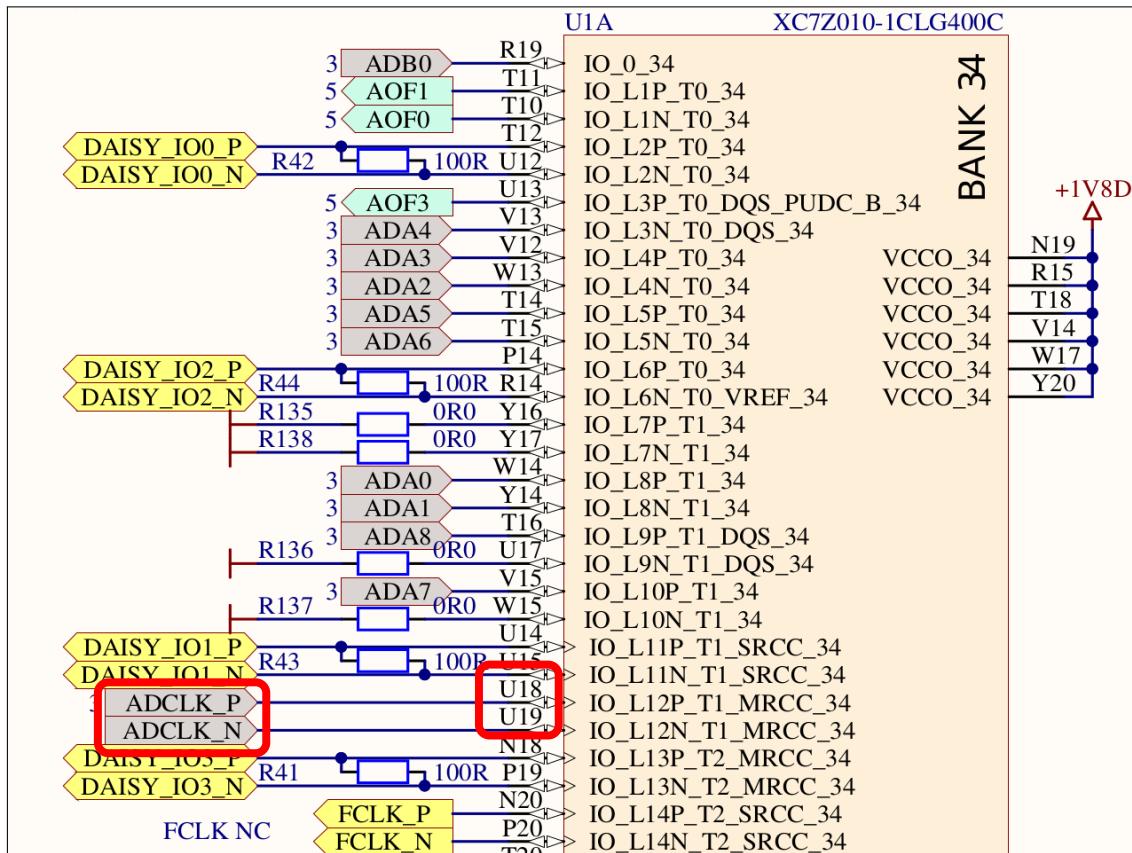


Figura A.3: Parte del esquemático eléctrico de la interfaz entre FPGA y ADC [3].

```
# Clock Input
set_property IOSTANDARD DIFF_HSTL_I_18 [get_ports adc_clk_p_i]
set_property IOSTANDARD DIFF_HSTL_I_18 [get_ports adc_clk_n_i]
set_property PACKAGE_PIN U18 [get_ports adc_clk_p_i]
set_property PACKAGE_PIN U19 [get_ports adc_clk_n_i]
```

## A.2. IP Cores de Xilinx

En esta sección se repasarán algunos de los cores de Xilinx más útiles, que serán utilizados en este trabajo.

### A.2.1. Processing System

El core *Processing System* implementa el interfaz entre el sistema de procesamiento (PS) y la lógica programable (PL), junto con el resto de periféricos de la placa.

Este core debe estar presente en todos los diseños orientados a la Zynq, puesto que de no estarlo al cargar el bitstream sobre la FPGA, el sistema de procesamiento pierde la interfaz de comunicación con la lógica programable y con todos los periféricos de la placa.

De hecho, se ha comprobado de forma experimental, que si se carga un bitstream que no contiene instanciado este core, se pierde la comunicación tanto por UART, como por sesión SSH, quedando el procesador aislado, hasta que se vuelve a reiniciar la alimentación de la placa y el procesador vuelve a reiniciar la interfaz con los periféricos de la placa desde la tarjeta SD.

Este core además proporciona una interfaz de comunicación entre el sistema de procesamiento y la lógica programable a través del protocolo de comunicación AXI. Se puede utilizar por tanto, para controlar la lógica combinacional y todo el hardware desde el sistema de procesamiento en el que se encuentra en funcionamiento un sistema operativo *Linux*.

### A.2.2. Clock Wizard

El core *Clock Wizard* configura las señales de reloj de la FPGA. Proporciona síntesis de frecuencia, lo que permite generar múltiples frecuencias de salida a partir de un mismo reloj de entrada. Permite trabajar con varios relojes de entrada y puede trabajar con relojes diferenciales.

Clock wizard utiliza los *hard cores* MMCM y PLL de la FPGA, estos son bloques hardware que ya están físicamente implementados en el chip. También proporciona una opción llamada *dynamic reconfiguration* o reconfiguración dinámica, que permite modificar algunos valores del clock wizard como divisores, desfases o ciclos de trabajo de cada reloj de salida, a través del interfaz AXI, permitiendo configurar los relojes en tiempo real desde el sistema operativo funcionando en el procesador.

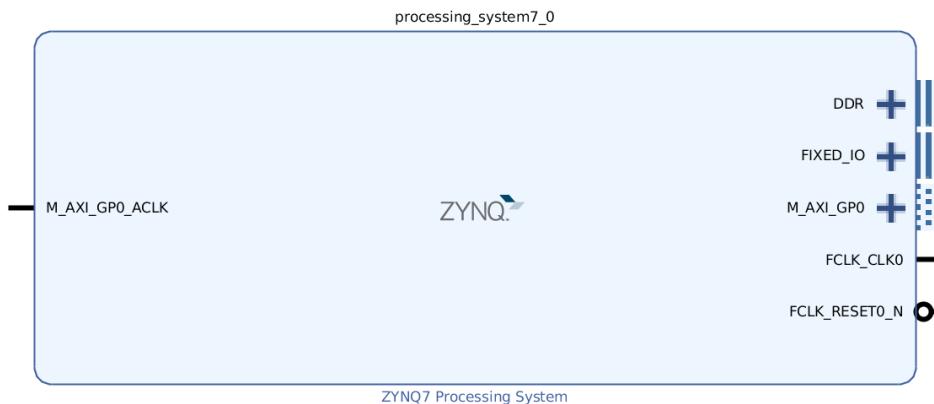


Figura A.4: Core Processing System para el SoC Zynq-7000 series.

## A.3. Periféricos de la placa

### A.3.1. Conversor Analógico a Digital (ADC)

El ADC de la placa *Red Pitaya* está basado en el chip LTC2145-14. En la figura A.6 se muestra el diagrama de bloques del integrado y se pueden diferenciar la señal de reloj de entrada, dos canales, cada uno con su registro S/H de entrada, seguido del conversor analógico a digital de 14 bits. Finalmente, el *driver* o controlador de salida [4].

Para la aplicación que se está desarrollando, el ADC de la placa se va a configurar en modo *low voltage* o voltaje bajo, es decir,  $\pm 1V$  de entrada analógica. El mapeo analógico-digital que se implementará será por tanto el mostrado en la figura A.7.

### A.3.2. Conversor Digital a Analógico (DAC)

El DAC de la placa *Red Pitaya* está basado en el chip DAC1401D125. En la figura A.8 se puede observar el diagrama de bloques del integrado y se pueden diferenciar señales de entrada de control, señal de reloj, y datos de 14 bits de entrada para cada canal. Tras una etapa de dos *latches* se encuentra el conversor digital a analógico y finalmente la salida analógica. También existe un pequeño interfaz de control, pero para el proyecto no ha sido necesario cambiar la configuración que el integrado trae de fábrica [5].

Los conversores digital a analógico funcionan en un rango de  $\pm 1V$  de salida. Y

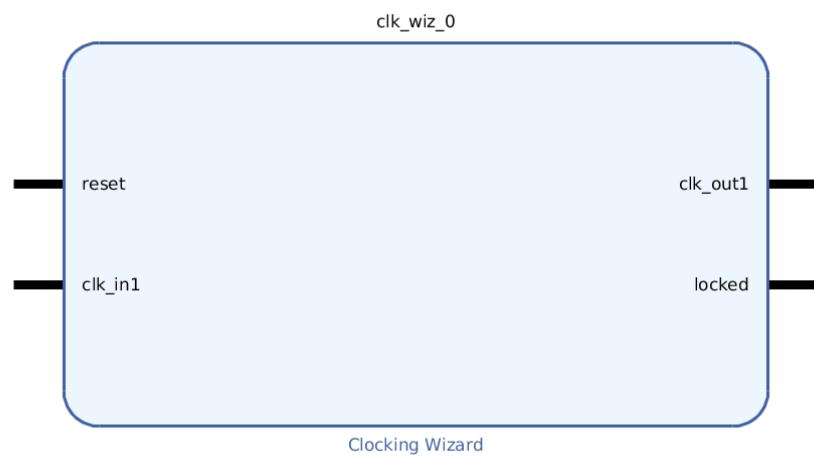


Figura A.5: Core Clock Wizard por defecto de Xilinx.

el mapeo digital-analógico es equivalente al mostrado en la sección anterior para el ADC en la figura A.7.

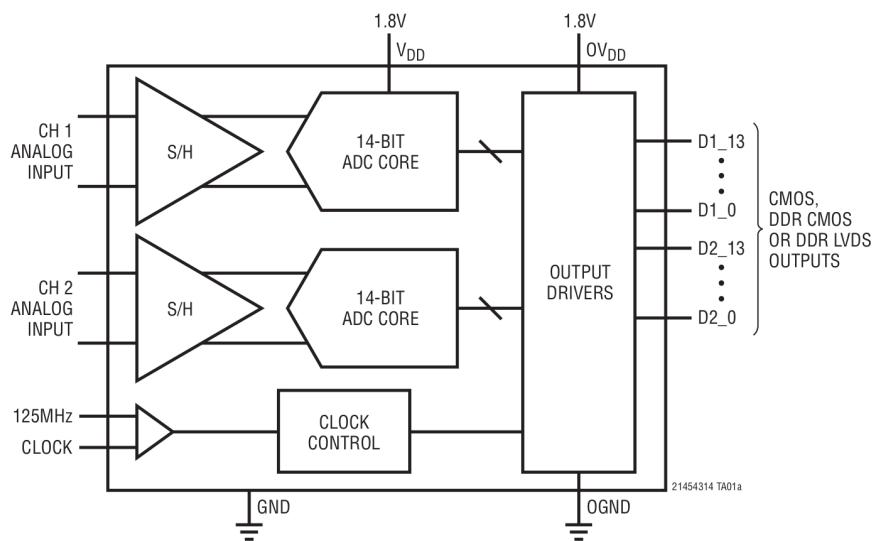


Figura A.6: Diagrama de bloques del integrado LTC2145-14 [4].

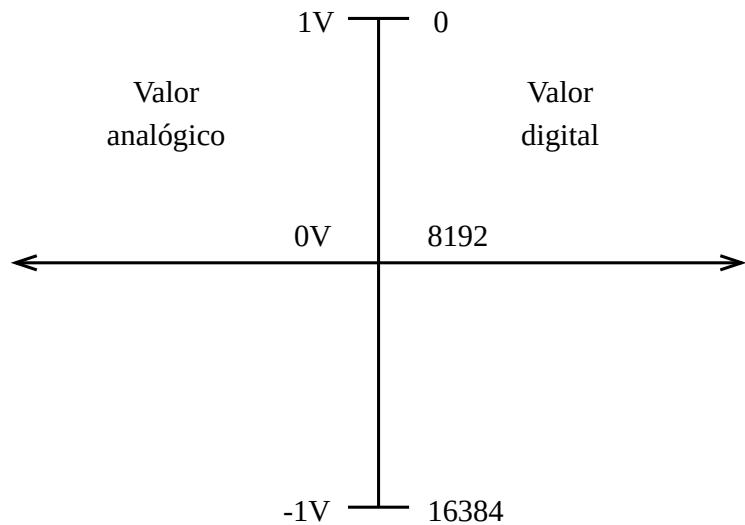


Figura A.7: Rango bipolar de los integrados LTC2145-14 y DAC1401D125.

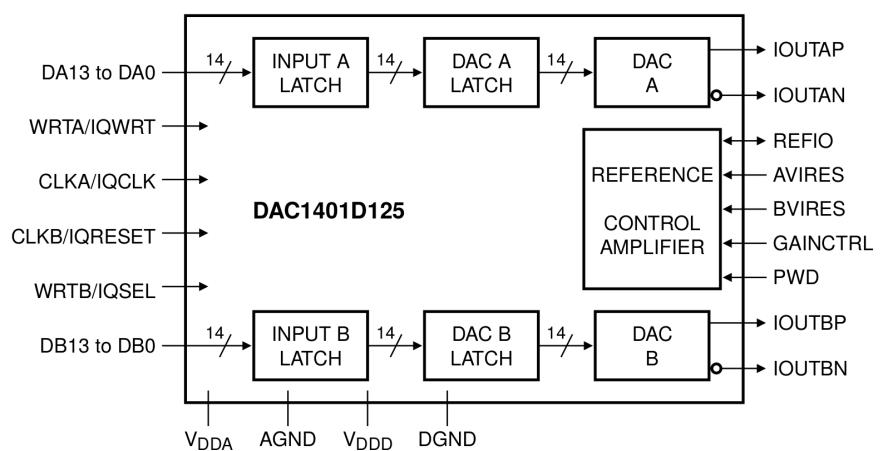


Figura A.8: Diagrama de bloques del integrado DAC1401D125 [5].



# Bibliografía

- [1] S. H. Yu, O. Shih, H. M. Tsai, and R. D. Roberts, “Smart automotive lighting for vehicle safety,” *IEEE Communications Magazine*, vol. 51, no. 12, pp. 50–59, 2013.
- [2] A. M. Căilean and M. Dimian, “Impact of IEEE 802.15.7 standard on visible light communications usage in automotive applications,” *IEEE Communications Magazine*, vol. 55, no. 4, pp. 169–175, 2017.
- [3] Red Pitaya Corporation, “Electrical schematics for Red Pitaya STEM 125-10,” 2017.
- [4] Linear Technology, “LTC2145-14/ LTC2144-14/LTC2143-14 14-Bit, 125Msps 105Msps/ 80Msps Low Power Dual ADCs datasheet,” pp. 96–126, 2004.
- [5] Integrated Device Technology, “DAC1401D125 datasheet,” no. July, pp. 1–25, 2012.
- [6] Ministerio de Interior: Dirección General de Tráfico, “Anuario Estadístico de Accidentes 2016,” 2016. [Online]. Available: <http://www.dgt.es/Galerias/seguridad-vial/estadisticas-e-indicadores/publicaciones/anuario-estadistico-de-accidentes/Anuario-accidentes-2016.pdf>
- [7] DGT, “Siniestralidad Vial,” *Ministeri d'interior*, pp. 5–12, 2016. [Online]. Available: <http://www.dgt.es/Galerias/seguridad-vial/estadisticas-e-indicadores/publicaciones/principales-cifras-siniestralidad/Las-principales-cifras-2016.pdf>
- [8] A. M. Cailean and M. Dimian, “Current Challenges for Visible Light Communications Usage in Vehicle Applications: A Survey,” *IEEE Communications Surveys and Tutorials*, vol. 19, no. 4, pp. 2681–2703, 2017.
- [9] L. Grobe, A. Paraskevopoulos, J. Hilt, D. Schulz, F. Lassak, F. Hartlieb, C. Kottke, V. Jungnickel, and K. D. Langer, “High-speed visible light communication systems,” *IEEE Communications Magazine*, vol. 51, no. 12, pp. 60–66, 2013.

- [10] Xilinx Corporation, “Spartan-II FPGA Family Data Sheet,” pp. 1–99, 2008.
- [11] S. Scholl, “The Xilinx Zynq : A Modern System on Chip for Software Defined Radios,” 2016. [Online]. Available: [https://kluedo.ub.uni-kl.de/frontdoor/deliver/index/docId/4442/file/zynq{\\_\\_}software{\\_\\_}defined{\\_\\_}radio.pdf](https://kluedo.ub.uni-kl.de/frontdoor/deliver/index/docId/4442/file/zynq{__}software{__}defined{__}radio.pdf)
- [12] Xilinx Inc., “XA Zynq-7000 SoC Data Sheet : Overview,” vol. 188, 2018.
- [13] Xilinx Inc, “DSP48E1 Slice User Guide,” *Xilinx*, vol. 479, pp. 1–56, 2017.
- [14] Xilinx, “7 Series FPGAs Clocking Resources - User Guide [UG472 v1.14],” vol. 472, pp. 1–114, 2018.
- [15] Xilinx Inc., “MMCM and PLL Dynamic Reconfiguration,” vol. 888, pp. 1–19, 2015.
- [16] Red Pitaya, “Web Page, Red Pitaya STEMlab, Documentation, Developers Guide,” 2017.
- [17] L. Litwin, “Matched filtering and timing recovery in digital receivers,” *RF Design*, vol. 24, no. 9, pp. 32–49, 2001. [Online]. Available: <http://www.teilar.gr/dbData/ProfAnn/profann-44e04051.pdf>
- [18] Uwe Meyer-Baese, *Digital Signal Processing with Field Programmable Gate Arrays*, 2005.