ULPGC Universidad de Las Palmas de Gran Canaria | Instituto Universitario de Microelectrónica Aplicada IUMA

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Análisis y diseño de un *beamformer* digital para constelaciones de satélites LEO

Autor: Karen Lyn García Suárez
Tutor(es): Dr. Sunil Lalchand Khemchandani
Irene Merino Fernández
José Luis Saiz Pérez
Fecha: Junio 2024

t +34 928 451 150
+34 928 451 086
f +34 928 451 083
e: iuma@iuma.ulpgc.es
w: www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 1 – 20 julio 2020

ULPGC Universidad de Las Palmas de Gran Canaria | Instituto Universitario de Microelectrónica Aplicada IUMA

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Análisis y diseño de un *beamformer* digital para constelaciones de satélites LEO

HOJA DE FIRMAS

Alumno/a:	Karen Lyn García Suárez	Fdo.:
Tutor/a:	Dr. Sunil Lalchand Khemchandani	Fdo.:
Tutor/a:	Irene Merino Fernández	Fdo.:
Tutor/a:	José Luis Saiz Pérez	Fdo.:

Fecha: Junio 2024

t +34 928 451 150
+34 928 451 086
f +34 928 451 083

e: iuma@iuma.ulpgc.es

w: www.iuma.ulpgc.es

Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 1 – 20 julio 2020

ULPGC
Universidad de Las Palmas de Gran Canaria

Instituto Universitario de Microelectrónica Aplicada

IUMA

Máster Universitario en Electrónica y Telecomunicación Aplicadas (META)

Trabajo Fin de Máster

Análisis y diseño de un beamformer digital para constelaciones de satélites LEO

HOJA DE EVALUACIÓN

Calificación: .....

Presidente	<<Nombre del Presidente>>	Fdo.:
Secretario	<<Nombre del Secretario>>	Fdo.:
Vocal	<<Nombre del Vocal>>	Fdo.:

Fecha: Junio 2024

t +34 928 451 150
+34 928 451 086
f +34 928 451 083

e: iuma@iuma.ulpgc.es

w: www.iuma.ulpgc.es

Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

Versión: 1 – 20 julio 2020

Agradecimientos

Quiero expresar mi gratitud a mi familia por su apoyo constante y comprensión durante todo este proceso. Su respaldo ha sido esencial para que pudiera llegar hasta aquí.

A mis tutores, les agradezco su orientación y valiosos consejos que han sido fundamentales para la realización de este trabajo. Su compromiso y disposición para ayudarme han sido de gran importancia.

Gracias a todos por su apoyo.

Resumen

Este trabajo aborda el control de fase de las agrupaciones o *arrays* de antenas para las comunicaciones con satélites de órbita terrestre baja o LEO (*Low Earth Orbit*) a partir de las propiedades tanto de los *arrays* de antenas como de las distintas arquitecturas de conformación de haz o *beamforming* existentes. Se aporta una comparación entre estas últimas, destacando el papel de la arquitectura *beamforming* digital que se implementa en este Trabajo Fin de Máster (TFM).

La arquitectura *beamforming* digital se caracteriza por emplear algoritmos programables para ajustar el haz de un *array* de antenas. Frente a la arquitectura analógica, el *beamforming* digital aporta una mayor precisión y control sobre el haz generado por un *array* de antenas, además permite reducir el número de componentes, llevando a un diseño más sencillo y versátil.

En este TFM se presenta el diseño de un algoritmo de *beamforming* digital, implementado sobre Matlab/Simulink. El algoritmo se ajusta para su posterior implementación sobre una plataforma de hardware programable o *Field Programmable Gate Array* (FPGA). Se comparan los resultados obtenidos mediante el modelo de rendimiento y los que se describe en el código en Matlab implementado, utilizando para ello la plataforma FPGA Arty z7-20. Finalmente, se genera el código HDL (*Hardware Description Language*) a través de Simulink, verificado a través de un banco de pruebas. A partir de este código, se genera el bloque IP (*Intellectual Property*), necesario para su implementación en Vivado, completando de este modo la síntesis. Estos resultados permiten presentar el diseño del *layout* de la síntesis, así como los recursos utilizados para su implementación física.

Abstract

This work deals with the phase control of antenna arrays for Low Earth Orbit (LEO) satellite communications based on the properties of both antenna arrays and the different beamforming architectures available. A comparison between the architectures is provided, highlighting the role of the digital beamforming architecture implemented in this TFM.

The digital beamforming architecture is characterized by the use of programmable algorithms to adjust the beam of an antenna array. Compared to the analog architecture, digital beamforming provides greater precision and control over the beam generated by an antenna array. In addition, it reduces the number of components, leading to a simpler and more versatile design.

This TFM presents the design of a digital beamforming algorithm, implemented in Matlab/Simulink. The algorithm is adjusted for its subsequent implementation on a programmable hardware platform or FPGA (Field Programmable Gate Array). The results obtained from the performance model and those described in the implemented Matlab code are compared using the Arty z7-20 FPGA platform. Finally, the HDL (Hardware Description Language) code is generated through Simulink, verified through a *testbench*. From this code, the IP (Intellectual Property) block is generated, necessary for its implementation in Vivado, thus completing the synthesis. These results allow us to present the *layout* design of the synthesis, as well as the resources used for its physical implementation.

Contenido

Índice de Figuras

Índice de Tablas

ACRÓNIMOS

ADC	*Analog to Digital Converter*
ASIC	*Application-Specific Integrated Circuit*
ASSP	*Application Specific Standard*
BRAM	*Block RAM*
BSP	*Bit Stream Processing*
BUFG	*Global Clock Simple Buffers*
CORDIC	*Coordinate Rotation Digital Computer*
CPU	*Central Processing Unit*
CTBPDSM	*Continuous-Time Delta-Sigma Modulator*
DAC	*Digital to Analog Converter*
DAS	*Delay-and-Sum*
DSP	*Digital Signal Processor*
FF	*Flip-Flops*
FFT	Transformada Rápida de Fourier
FI	Frecuencias intermedias
FPGA	*Field Programmable Gate Array*
GEO	*Geostationary Equatorial Orbit*
HDL	*Hardware Description Language*
HEO	*Highly Elliptical Orbit*
HLS	*High-Level Synthesis*
IoT	*Internet of Things*
IP	*Intellectual Property*
LEO	*Low Earth Orbit*
LUT	*LookUp Table*
LUTRAM	*Look-Up Table Random Access Memory*
MAC	*Multiply and Accumulate*
MEO	*Medium Earth Orbit*
MIMO	*Multiple-Input-Multiple-Output*
MMCM	*Mixed-Mode Clock Manager*
mMTC	*Machine-Type Communications*
PL	*Programmable Logic*
PRF	*Pulse Repetition Frequency*

PS	*Processing System*
RAM	*Random Access Memory*
RF	Radiofrecuencia
RTL	*Register Transfer Level*
RTT	*Round Trip Time*
SNR	*Signal-to-Noise Ratio*
SoC	*System-on-Chip*
TFM	Trabajo Fin de Máster
TTD	*True Time Delay*
TT&C	*Tracking, Telemetry and Commands*
ULA	*Uniform Linear Array*
URAM	*UltraRAM*
VHDL	*Very High-Speed Integrated Circuit Hardware Description Language*

Capítulo 1. Introducción

1.1. Introducción

En este capítulo se exponen los antecedentes que contextualizan el marco de investigación de este Trabajo de Fin de Máster (TFM). Partiendo del concepto de constelaciones de satélites de órbita terrestre baja (*Low Earth Orbit,* LEO), se analiza su utilidad en el ámbito de las telecomunicaciones y cómo el uso de arrays de antenas en fase pueden mejorar sus funcionalidades.

1.2. Antecedentes

1.2.1. Constelaciones LEO

Las constelaciones de satélites LEO son un conjunto de satélites de órbita terrestre baja que trabajan cooperativamente y orbitan relativamente cerca de la superficie terrestre, situándose en altitudes inferiores a 2000 km. Si bien existen otros tipos de órbitas situadas a mayor altura como son las *Medium Earth Orbit* (MEO) entre 2.000 y 36.000 km, *Geostationary Equatorial Orbit* (GEO) a 35.786 km o *Highly Elliptical Orbit* (HEO) más allá de los 36.000 km, las constelaciones LEO ofrecen distintas ventajas significativas que se citan a continuación [1], [2], [3], [4]:

- Suponen un menor tiempo de retardo en la trasmisión de datos debido a su proximidad a la superficie terrestre, lo que resulta de interés en aplicaciones en tiempo real como videoconferencias o adquisición de datos meteorológicos, entre otras.
- El tamaño y peso de los satélites suele ser menor, como ocurre con los *cubesat*, incurriendo en costes de fabricación y lanzamiento inferiores.
- Las pérdidas de espacio libre son menores que en otros tipos de constelaciones, debido a que se reduce la distancia entre los satélites y los usuarios en la superficie terrestre, lo que permite establecer enlaces fiables utilizando una menor potencia y con antenas de menor tamaño.

En contrapartida, las constelaciones LEO suponen una serie de desventajas:

- Su vida útil se encuentra limitada, así que deben ser reemplazados con relativa frecuencia y, por consiguiente, incurren en un aumento de los desechos espaciales.
- Las constelaciones requieren un mayor número de satélites para cubrir huellas de cobertura amplias. Además, la duración de su visibilidad es de aproximadamente 90 minutos, lo que implica que los satélites sean observables entre 5 y 20 minutos por órbita.

Pese a estas desventajas, las constelaciones de satélite LEO siguen siendo una opción atractiva para diversas aplicaciones como para el monitoreo del agua, gestión de energía, servicio de Internet global, observación terrestre, seguimiento de desastres, etcétera. [1], [5].

Una representación de sistema de comunicaciones con satélites LEO se puede observar en la Figura 1. Por un lado, la zona del espacio está formada por los satélites, mientras que la zona terrestre está formada por las estaciones de rastreo, telemetría y comandos (*tracking, telemetry and commands* o TT&C), las estaciones bases o *gateways*, y los usuarios. Las estaciones TT&C son responsables de monitorizar los satélites en órbita, mientras que los *gateways* proveen del servicio de Internet a los usuarios a través de los satélites.

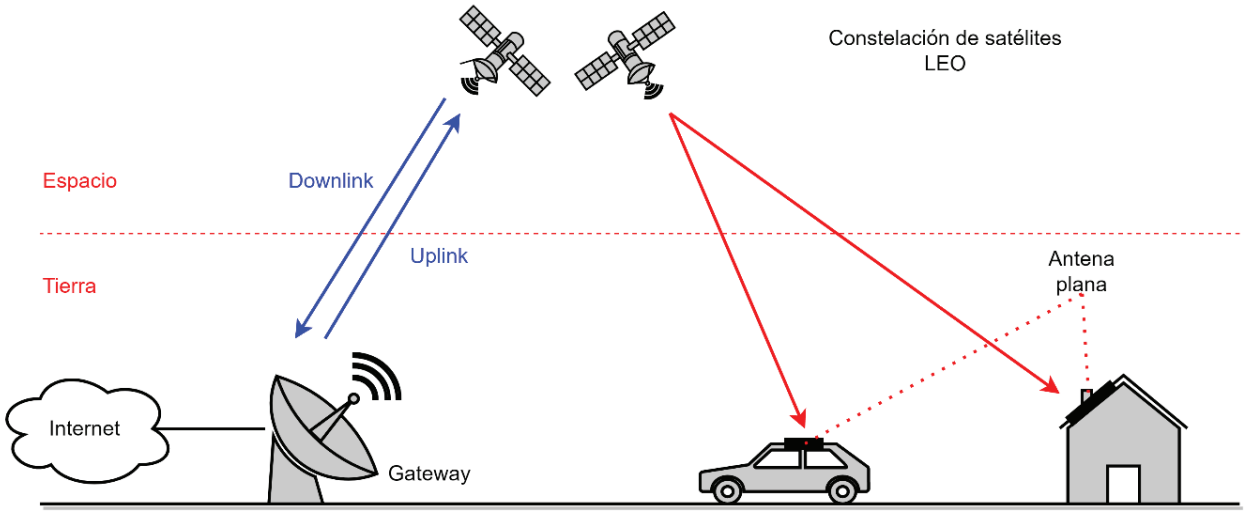


Figura 1: Sistema de comunicaciones con satélites LEO

Como la baja altitud de su órbita produce un retardo de propagación inferior al de otras constelaciones, también decrecerá el retardo de paquetes o *Round-Trip Time* (RTT), que se define como el tiempo medio que un paquete de datos tarda en regresar a su emisor tras haber llegada hasta su destino. Se trata de un parámetro especialmente útil para medir la calidad de los servicios de telecomunicaciones, siendo inferior a 100 ms en las constelaciones LEO. En otras constelaciones es bastante alto, llegando a más de 600 ms en los sistemas en órbita geoestacionaria [6]. Por tanto, las constelaciones LEO son una apuesta complementaria a 5G, ya que brindan cobertura global y dan soporte en diversos ámbitos, entre los que destaca el rendimiento de la telefonía móvil a través de comunicaciones *Machine-Type Communications* (mMTC), además de ampliar la gama de aplicaciones basadas en *Internet of Things* (IoT) al cubrir áreas geográficas extensas y aportar seguridad con latencias muy bajas, del orden de 30 ms, y con retardos de dispersión del orden de 2 ms entre la Tierra y los satélites. No obstante, el despliegue de esta tecnología es un desafío complejo, ya que sus costes se ven incrementados al requerirse un gran número de antenas para las comunicaciones [7], [8].

1.2.2. Array de antenas en fase

Una antena es un dispositivo diseñado con el objetivo de emitir y/o recibir ondas electromagnéticas hacia y desde el espacio libre. Se utilizan en las comunicaciones por satélite, radio, televisión, teléfonos móviles, estaciones bases de telefonía, routers inalámbricos, mandos a distancia, etcétera. Sus parámetros más importantes son el ancho de banda, la directividad, la ganancia, el rendimiento en la antena, la impedancia, la anchura de haz y el tipo de polarización [9][10].

Cuando varias antenas se agrupan para operar conjuntamente en la transmisión o recepción de señales, se dice que se forma una agrupación o *array* de antenas, cuyas propiedades van desde la disposición geométrica hasta el número de antenas del sistema [11]. En la recepción, estos *arrays* pueden permitir determinar la dirección de llegada de señales (radiogoniometría) y, en transmisión, pueden radiar las señales en una dirección específica del espacio

(*beamforming*). En las comunicaciones vía satélite son muy comunes las técnicas de beamforming para realizar un apuntamiento eficiente desde la superficie terrestre hasta un satélite. Para ello se suelen utilizar agrupaciones de antenas conocidas como *phased-arrays*, que tienen la capacidad de generar varios haces de radiación simultáneos sobre un área reducida de forma flexible. Estas agrupaciones son tratadas a lo largo de este TFM.

Como se observa en la Figura 2, los *phased-arrays* son una combinación de varias antenas que, al conectarse entre sí, actúan como una única antena. El ancho de haz y dirección de apuntamiento puede ser modificado mediante circuitos electrónicos, sin necesidad de mover físicamente ninguna de las antenas, lo que evita disponer de sistemas mecánicos para su apuntamiento [1][2]. Este apuntamiento se consigue desfasando la señal de entrada un cierto valor entre cada elemento radiante, lo que se traduce en un ángulo entre una referencia espacial dada y el lóbulo principal conformado. En frecuencias del orden de las microondas, son habituales los *phased-arrays* formados por antenas microstrip. Estas antenas son pequeñas y compactas, y su fabricación sobre un sustrato adecuado suelen tener costes asequibles en comparación con algunas antenas parabólicas y de bocina. Si bien las antenas microstrip presentan una eficiencia baja, su combinación en un *phased-array* permite conseguir resultados muy aceptables en el apuntamiento de antenas, y son especialmente interesantes en aplicaciones donde las dimensiones y el peso de las antenas sea crítico, como ocurre en el sector aeroespacial [12][13].

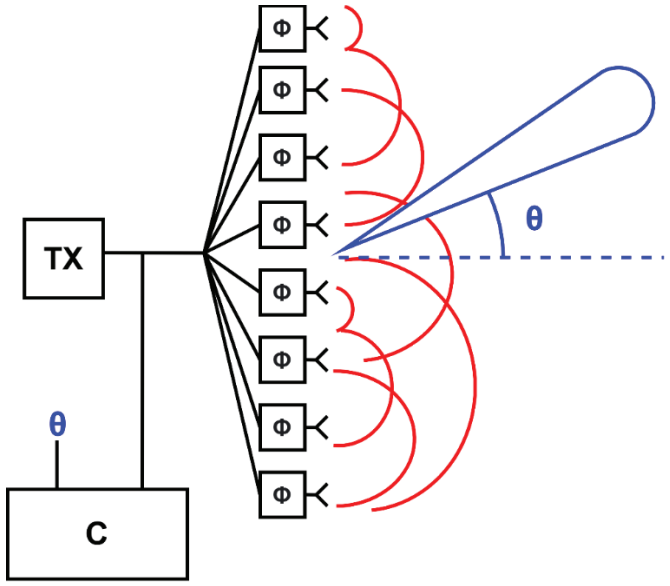


Figura 2. Phased array.

Las principales ventajas de utilizar *phased-arrays* son las siguientes [14], [15]:

- Puede controlarse la forma del haz, permitiendo dirigir la energía radiada de forma precisa sobre una dirección específica, sin que ello implique un movimiento físico del sistema.
- El fallo en una antena no compromete al resto de antenas de la agrupación, ya que podrá seguir operando a pesar de la degradación que ocasionaría que uno de sus elementos se encontrase deteriorado. Además, esta antena podría ser reemplazada con relativa sencillez.
- La agrupación permite operar con múltiples haces de forma simultánea, siendo especialmente relevante en el seguimiento de blancos móviles, como es el caso de los satélites LEO.
- Suponen un menor coste tanto de fabricación como despliegue, ya que no requieren componentes electrónicos especialmente complejos y se pueden adaptar a numerosas infraestructuras y terrenos.

No obstante, también presentan sus desventajas, como son las siguientes [14], [15] :

- No son adecuados para operar con múltiples frecuencias, ya que la dimensión de sus elementos se asocia a una longitud de onda específica. Algunas soluciones pueden encontrarse en diseños adecuados de la red de *beamforming* o en la incorporación de elementos de retardo eficientes.
- La incorporación de nuevas funcionalidades suele ser una capacidad limitada, pues implicaría en muchos casos una extensión en el rango de frecuencias permitido por la agrupación, lo que exigiría alterar todas las antenas que la componen.
- Si no hay un buen control en el acoplamiento mutuo entre antenas, se pueden producir comportamientos anómalos en el patrón de radiación del *array*, apareciendo numerosos lóbulos secundarios y puntos ciegos en el apuntamiento.

Actualmente existen tres arquitecturas *beamforming* de referencia para implementar un *phased array*, que se conocen como analógica, digital e híbrida. Estas arquitecturas varían su comportamiento en función de la tecnología utilizada, y se estudian en mayor profundidad en el Capítulo 2. Arquitecturas *beamforming* de este TFM.

Un *beamformer* analógico permite controlar la formación y dirección de haz de radiación de la antena a través de la manipulación de las fases y amplitudes de las señales de alimentación de cada antena del array. Esto se consigue por medio de componentes analógicos como redes de retardo (filtros), divisores de potencia, amplificadores de ganancia variable y desfasadores, entre otros. A partir de estos componentes se ajustan las características de fase y amplitud de cada antena para que, al combinarse, produzcan el patrón de radiación deseado.

Un *beamformer* digital aplica técnicas de procesamiento digital de señales para manipular las señales recibidas por cada antena del *array*. Requiere una conversión y muestreo de las señales analógicas en digitales por medio de un *Analog to Digital Converter* (ADC), para luego aplicar algoritmos de procesamiento digital de señales como la *Fourier Fast Transform* (FFT) y las técnicas de filtrado adaptativo. Con estas técnicas se consigue manipular las

amplitudes y fases de las señales digitales para formar un determinado haz de radiación.

Por último, un *beamformer* híbrido combina tanto componentes analógicos como digitales en su diseño. Esto permite aprovechar las ventajas de ambas tecnologías, proporcionando un equilibrio entre la flexibilidad y la eficiencia de las soluciones digitales y la simplicidad y bajo coste de las soluciones analógicas. En estos sistemas se emplean componentes analógicos para el preprocesamiento de las señales en cada antena y un procesamiento extenso de señales en formato digital.

1.3. Objetivos

El objetivo principal de este TFM consiste en diseñar un *beamformer* digital. Los objetivos específicos se describen a continuación:

O1. Análisis del sistema. Estudio y análisis para definir el concepto inicial del sistema y sus especificaciones. Además, se realiza una comparativa entre los *beamformers* analógicos, digitales e híbridos.

O2. Estudio de los *beamformers* digitales. Se exploran las técnicas aplicables al desfase entre señales y arquitecturas *beamforming*.

O3. Diseño de un *beamformer* digital. Se realiza el diseño de un *beamformer* digital, basado en las especificaciones indicadas por el análisis del sistema y el estudio de los *beamformers* digitales.

O4. Documentación y memoria. Se realiza una memoria detallada del TFM junto con los resultados alcanzados, especificada por capítulos.

1.4. Diagrama de flujo

En la Figura 3 se muestra el diagrama de flujo que se ha seguido para la realización de este TFM. El diagrama se divide en dos partes, en la primera parte se expone el diseño llevado a cabo en Matlab/Simulink, mientras que en la segunda parte se expone el modelado del sistema a través de Vivado.

En la parte de Matlab/Simulink se encuentra la creación del diseño del algoritmo que se desea implementar, se compara el modelo matemático diseñado con el modelo HDL para asegurar que ambos modelos sean equivalentes, se genera el bloque IP a partir del modelo matemático y se genera

el código HDL necesario para la implementación en hardware, junto con el *testbench* para verificar la funcionalidad del diseño.

En la parte de Vivado se revisa el *testbench* generado para asegurar que el código HDL funcione correctamente y cumpla con los requisitos del diseño. Se genera el bloque IP necesario para su implementación en Vivado.

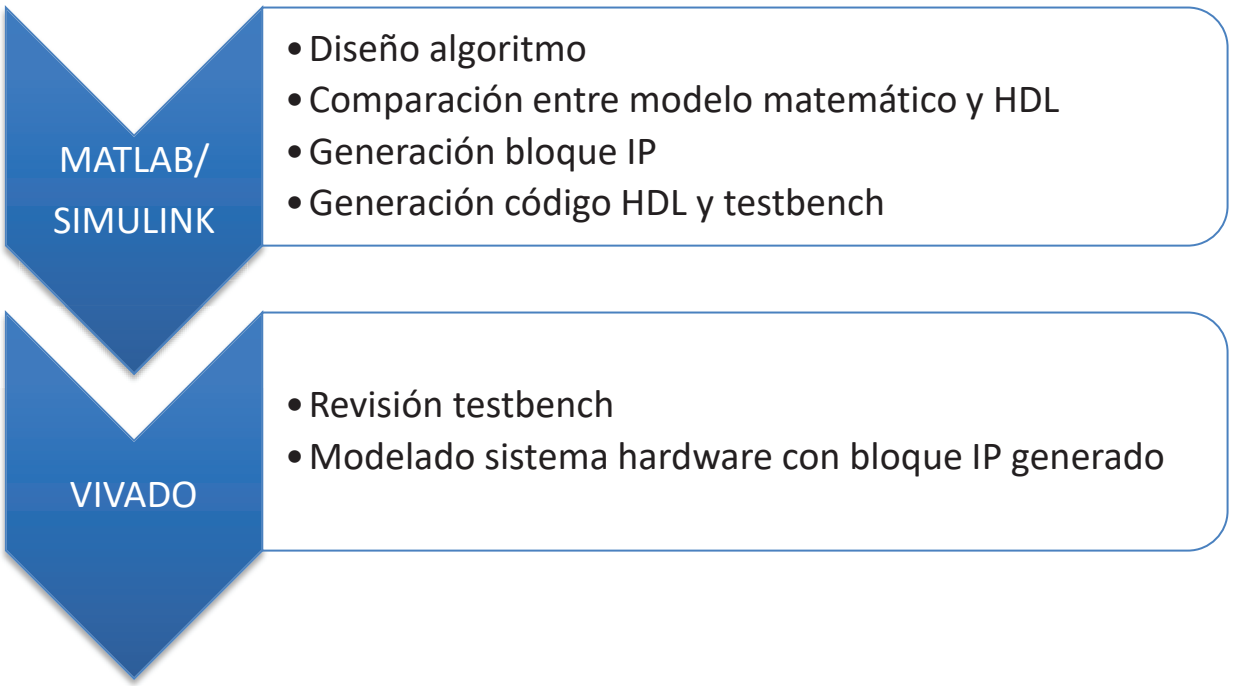


Figura 3: Diagrama de flujo

1.5. Estructura de la memoria

La estructura de la memoria de este TFM se organiza en los siguientes seis capítulos:

- **Capítulo 1:** se introducen los antecedentes y contextualización de este trabajo, así como los objetivos y la organización seguida en la memoria, dispuesta por capítulos.
- **Capítulo 2:** se detallan las diferentes arquitecturas de *beamforming* existentes (analógica, digital e híbrida).
- **Capítulo 3**: se muestra el diseño del algoritmo en Matlab/Simulink.
- **Capítulo 4**: se detalla la plataforma *Field Programmable Gate Array* (FPGA) utilizada, así como sus características.
- **Capítulo 5:** se describe el diseño del *beamformer* sobre FPGA.

- **Capítulo 6:** se exponen las conclusiones, limitaciones y líneas futuras de este trabajo.

Capítulo 2. Arquitecturas beamforming

2.1 Introducción

El beamforming o conformado de haz es una técnica utilizada en sistemas de comunicaciones para mejorar tanto la dirección como la intensidad de la señal transmitida o recibida. Permite focalizar la energía de una señal en una dirección específica, mejorando la eficiencia y calidad de la transmisión o recepción frente a factores limitantes como la atenuación o la presencia de interferencias y ruido. Se emplea en diversas aplicaciones, desde los radares hasta sónares, pasando por las radiocomunicaciones. Su arquitectura suele basarse en matrices de sensores (antenas), algoritmos de procesamiento de señales y métodos de ajuste en tiempo real. Asimismo, mejora la calidad de las señales y permite tener un uso más eficiente del espectro radioeléctrico.

2.2. *Beamformer* analógico

En el *beamformer* analógico el control de la amplitud y fase de la señal para cada antena se realiza en el dominio analógico, tanto en radiofrecuencia o RF como en frecuencias intermedias o FI [16]. Esta arquitectura, mostrada en la Figura 4, utiliza un número reducido de componentes, ya que solo necesita un ADC o un *Digital to Analog Converter* (DAC), en función de si se emplean en la cadena de transmisión o de recepción, respectivamente. Esto permite reducir tanto los costes como el consumo de potencia del sistema. No obstante, presentan un elemento crítico conocido como desfasador analógico en cada una de las antenas de la agrupación. Estos componentes suelen ser difíciles de implementar debido a que producen pérdidas y errores de fase importantes, sobre todo si se quiere cubrir anchos de banda amplios [17]. A este efecto se le conoce como *beam squint*, y ocurre cuando el haz de radiación principal se desvía de la dirección esperada cuando se producen diferencias en la fase de la señal transmitida en cada elemento del *array*.

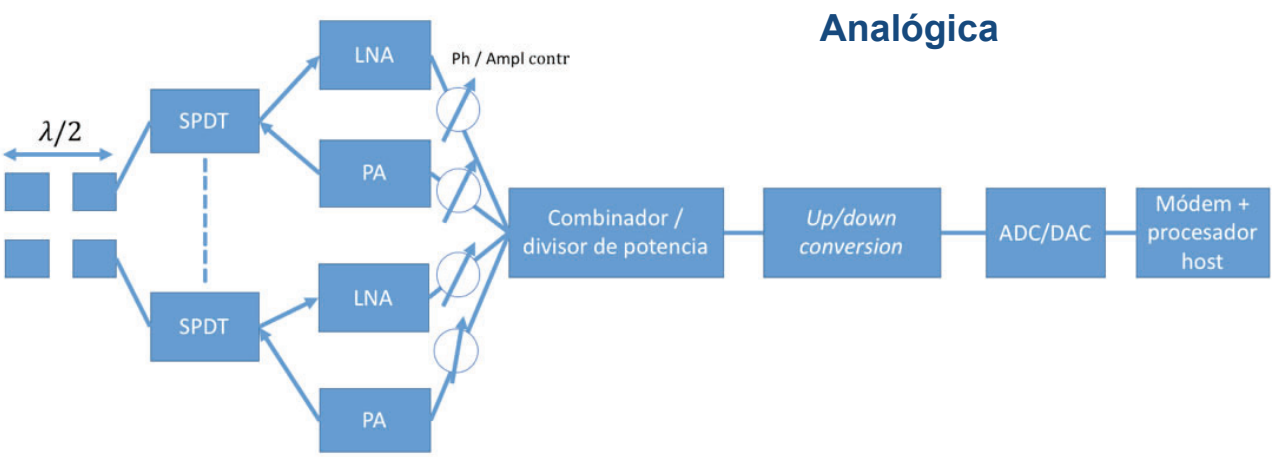


Figura 4: Arquitectura beamforming analógica

Para analizar este efecto, se considera un *array* formado por N antenas, donde cada una de ellas transmite una señal con una fase ligeramente diferente. La combinación de estas señales, con fases diferentes, genera un haz principal que se dirige hacia una dirección específica, pero si las fases de las señales no están perfectamente alineadas, el haz principal puede desviarse de la dirección deseada (Figura 5). En la práctica, debido a distintos factores relacionados con los componentes electrónicos o las condiciones ambientales, resulta complicado lograr una alineación de fase perfecta entre todas las antenas del *array*. La desviación producida puede afectar significativamente en el rendimiento de la agrupación de antenas, reduciendo su cobertura o mermando la calidad de la señal recibida, especialmente en aplicaciones que exigen anchos de banda elevados. Para contrarrestar este efecto se emplean distintas estrategias, entre las que destaca la incorporación de líneas de retardo, desplazadores de fase y diversos algoritmos de *beamforming*. Estas técnicas aseguran que el haz principal se mantenga alineado con la dirección deseada en todo el rango de frecuencias en el que opera el sistema, mitigando el impacto del *beam squint* [18], [19], [20], [21], [22], [23], [24].

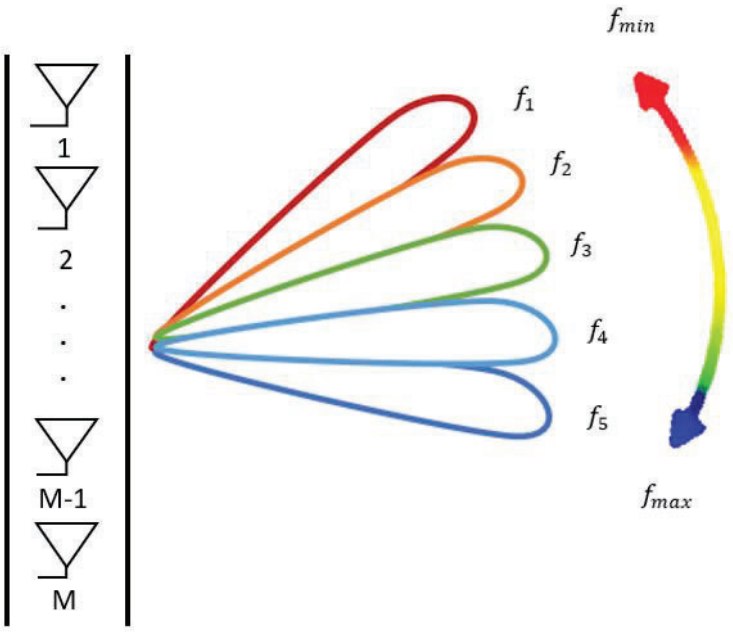


Figura 5: Beam squint

El *beam squint* se puede calcular a partir de la Ecuación (1), donde $\Delta\theta$ es el ángulo máximo de desfase, f_0 la frecuencia central, θ_0 el ángulo máximo del haz y f la frecuencia medida. Si θ es igual a cero no existiría desfase entre los elementos y, por lo tanto, tampoco *beam squint* [18], [21].

$$\Delta\theta = sin^{-1}\left(\frac{f_0}{f} * \sin\theta_0\right) - \theta_0 \quad (1)$$

En función del ángulo θ se puede conocer la variación de la frecuencia y la cantidad de *beam squint* que se ha producido. En la Figura 6 se muestra un ejemplo donde el ancho de banda de modulación es de 2 GHz, la frecuencia central es de 10 GHz, y el haz modifica su trayectoria dependiendo tanto del ángulo inicial con el que se emite como de la frecuencia. El *array* de antenas de este ejemplo se compone de 32 elementos, con una separación entre ellos igual a media longitud de onda, λ/2 [18], [21].

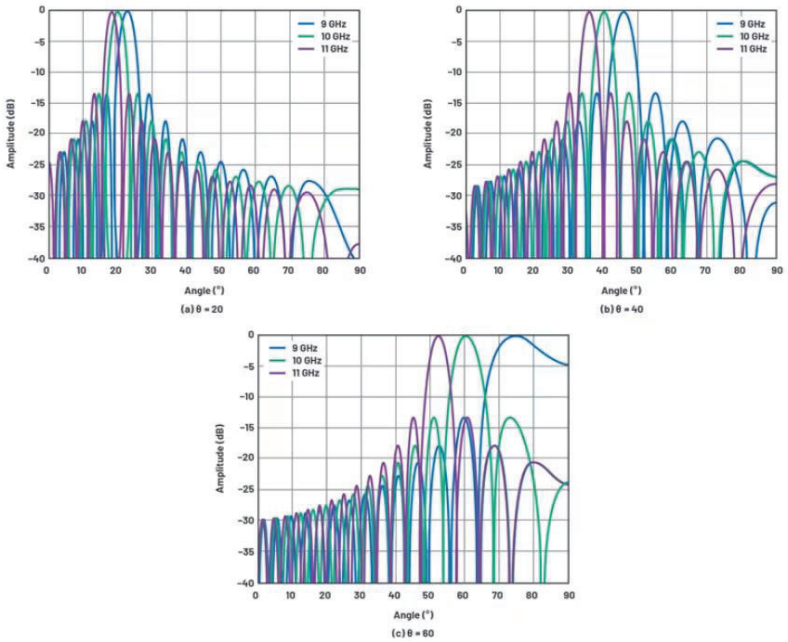


Figura 6: Beam squint según frecuencia y ángulo de elevación [18].

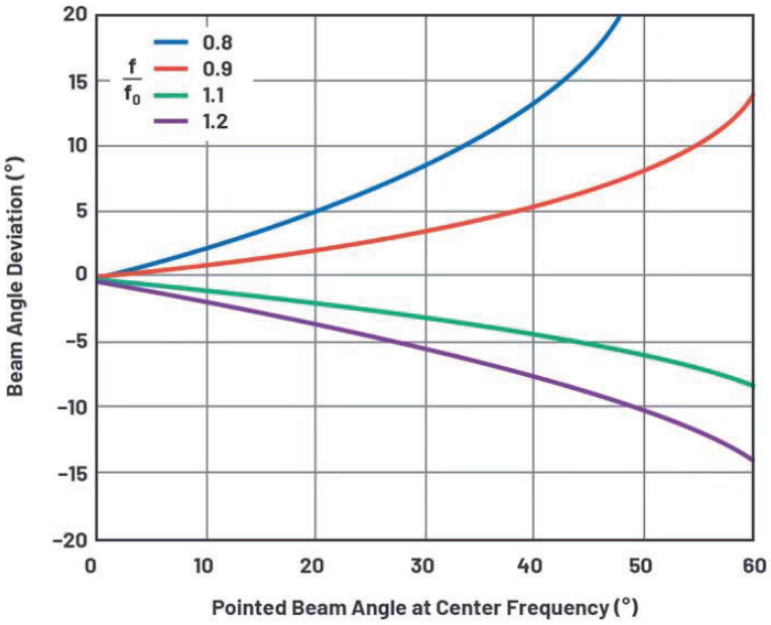


Figura 7: Beam squint en función del ángulo para varias desviaciones de frecuencia [18].

La Figura 7 permite obtener algunas conclusiones. Por un lado, en la medida en que el ángulo del haz se incrementa al alejarse del punto de observación de la agrupación, la desviación del ángulo del haz respecto a la frecuencia también aumenta. Por otro lado, una frecuencia inferior a la frecuencia central produce una desviación mayor que una frecuencia que sea superior a la frecuencia central, y por contra, una frecuencia inferior a la central desvía el haz del punto de observación [18].

Uno de los factores principales en la formación de un haz de radiación es la geometría del *array* de antenas. Se puede conocer la salida del *beamformer* en el tiempo mediante la combinación lineal de las muestras espaciales, $x_m(t), m = 0,1, \dots, M-1$, a partir de la Ecuación (2), donde M es el número de elementos radiantes (antenas) que se muestrean espacialmente en el campo de ondas, y(t) la salida del *beamformer*, $x_m(t)$ la salida de la antena y w_m^* el peso complejo para cada antena [25].

$$\mathrm{y(t)} = \sum_{m=0}^{M-1} w_m^* x_m(t) \tag{2}$$

En la Ecuación (3) se calcula el desplazamiento de fase, donde $\Delta\varphi$ es la diferencia de fase, θ el ángulo con respecto al eje de orientación de la antena, d la distancia entre antenas y λ la longitud de onda de operación.

$$\Delta\varphi = \frac{2\pi}{\lambda}\, dsen\, \theta \tag{3}$$

Se puede inclinar el lóbulo principal del *array* de antenas a una frecuencia dada. Como la relación de fase se calcula a una determinada frecuencia de portadora, según la frecuencia que se emplee en cada momento este lóbulo se desplazará un cierto ángulo, suponiendo pérdidas en la ganancia del sistema [25].

La Ecuación (3) puede independizarse de la frecuencia si en vez de considerar desplazamientos de frecuencia se emplean retardos de tiempo, como indica la Ecuación (4), donde c es la velocidad del medio de propagación, que en el vacío coincide con la velocidad de la luz.

$$\Delta\mathrm{t} = \frac{d\ \mathrm{sen}\ \theta}{c} \tag{4}$$

Las desventajas de este tipo de *beamforming* son las siguientes. Por un lado, encuentra importantes limitaciones tanto en la tasa de transferencia de datos como en la flexibilidad de la arquitectura, ya que las antenas del *array* solo son alimentadas por un único flujo de datos. Por otro lado, no es adecuado para aplicaciones en donde la potencia sea un factor importante, ni en aquellas que

exijan selección en frecuencia. Además, en implementaciones analógicas el número de *beams* creados no se puede variar dinámicamente [26][27].

2.3. *Beamformer* digital

El *beamformer* digital, mostrado en la Figura 8, realiza el desfase de la señal en el dominio digital, lo que permite generar y recibir haces de radiación más estrechos, comunicarse con varios haces simultáneamente y llevar a cabo una calibración más precisa y controlada del sistema [12][28][29]. Sin embargo, requieren un ADC o un DAC, así como un conversor de frecuencia o mezclador para cada antena. Esto repercute en el consumo de potencia, por lo que es un punto especialmente delicado en su diseño [30].

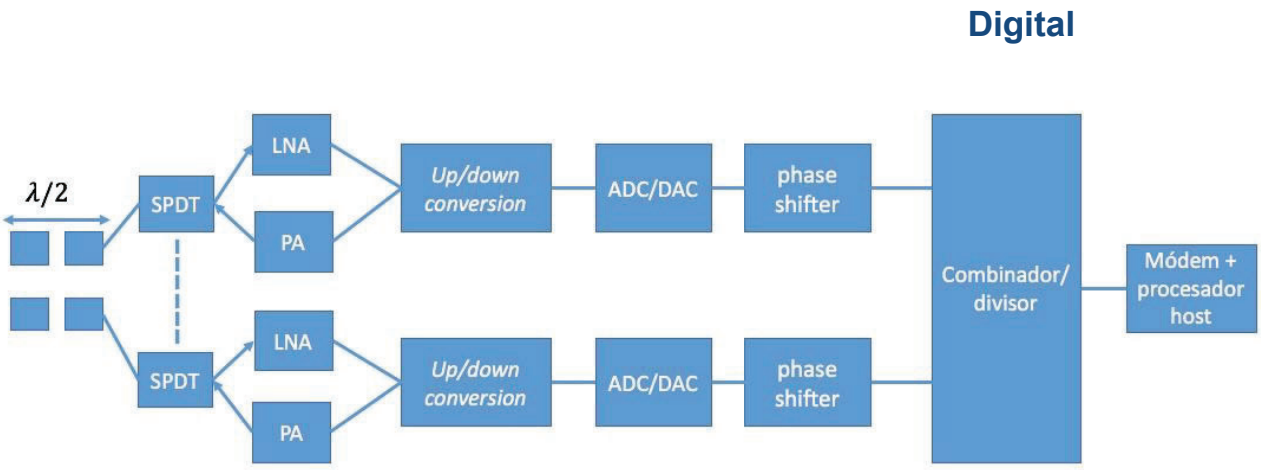


Figura 8: Arquitectura beamforming digital

En un *beamfomer* digital, la señal recibida se detecta y digitaliza para posteriormente procesarla, obteniendo el haz de radiación deseado. Este enfoque tiene dos características principales. En primer lugar, se conserva la información total disponible, expresada como N señales de un solo elemento frente a la generación de una señal analógica, que opera con una única señal. En segundo lugar, una vez digitalizadas correctamente las señales analógicas de entrada, se pueden manipular indefinidamente sin añadir más errores, ya que se emplea la representación digital de la señal en lugar de la magnitud real de la señal recibida. De esta manera, se puede generar cualquier número de haces [31].

El *beamformer* digital supone varias ventajas frente al *beamformer* analógico, como es la precisión en la generación de haces, la recepción de múltiples haces sin pérdida de calidad (expresada en función de la relación señal a ruido o *Signal-to-Noise,* SNR), la cancelación de ruido adaptable y, por último, la flexibilidad de generar un haz totalmente configurable. La generación de múltiples haces simultáneos en este *beamformer* avala el uso de la tecnología *Multiple-Input-Multiple-Output* (MIMO) para mejorar tanto la fiabilidad como acelerar la transmisión de datos. Cuando se recibe un único flujo de datos por ráfagas, la diversidad de transmisión mejora la resistencia al desvanecimiento por trayectos múltiples. La multiplexación espacial, donde se reciben diferentes flujos de datos en diferentes ráfagas, también puede lograr tasas de datos más altas o reducir aún más los requisitos de SNR del receptor. Sin embargo, el formato de *beamformer digital* se encuentra limitado por su alto consumo energético [32].

2.4. *Beamformer* híbrido

Los *beamformers* híbridos (Figura 9) combinan las dos tecnologías vistas anteriormente. El desfase se realiza tanto en el dominio digital como en el analógico [17], facilitando el diseño de los desfasadores con respecto al de otras arquitecturas. La configuración híbrida utiliza una combinación entre la formación del haz en el rango de frecuencias de RF desde la arquitectura analógica, y en banda base desde la arquitectura digital, con menos secuencias de RF en relación con el número de antenas transmisoras. El hecho de que se empleen menos cadenas de RF permite utilizar más antenas en la agrupación, al tiempo que reducen la potencia y la complejidad de diseño del sistema [33].

Con ello, la tecnología híbrida combina los *beamformers* analógicos y digitales para proporcionar un equilibrio entre el coste del hardware y el rendimiento de transmisión para las comunicaciones de ondas milimétricas, pero presenta problemas en la optimización [34].

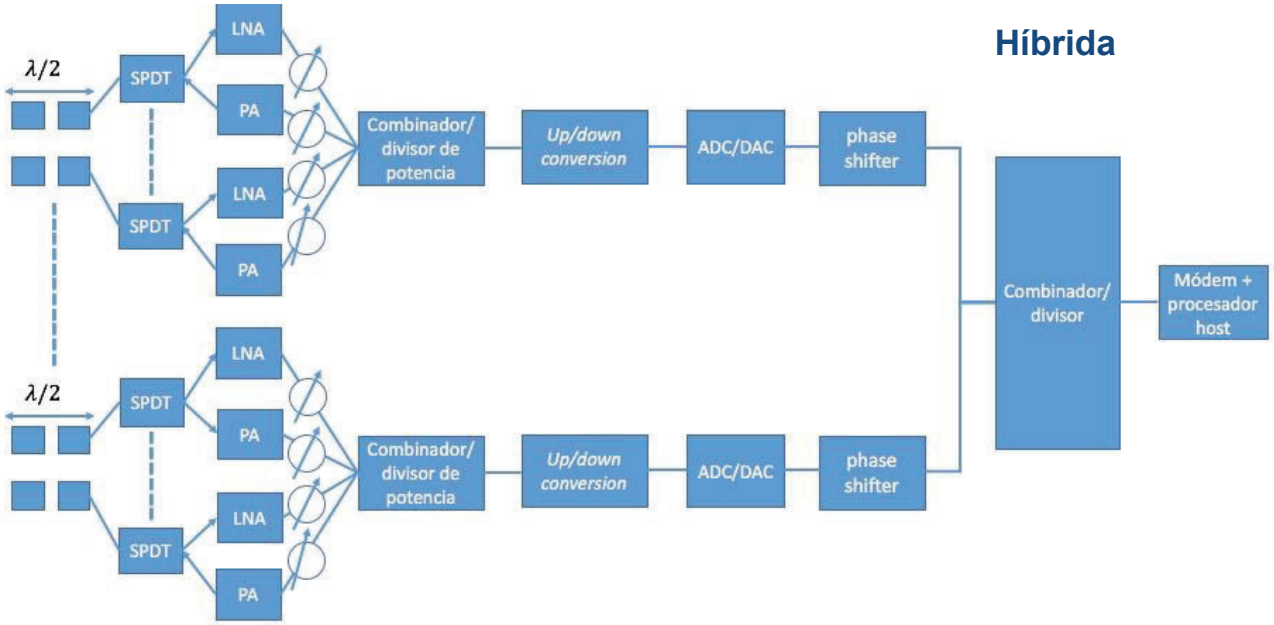


Figura 9: Arquitectura beamforming mixta

2.5. Comparativa entre arquitecturas *beamforming*

En la Tabla 1 se muestra una comparativa entre las tres estructuras existentes.

Tabla 1. Resumen de las diferentes arquitecturas para el beamforming.

***Beamformers* analógicos**	***Beamformers* digitales**	***Beamformers* híbridos**
• Menor número de componentes. • Menor disipación de energía. • Más sensible a la pérdida de interconexión. • Complejidad en el desfase. • Problema con la variación del ángulo para anchos de banda grandes (*beam squint)*.	• Gran cantidad de componentes. • Alta disipación de energía. • Más fácil de implementar. • Adecuado para grandes anchos de banda.	• Para grandes conjuntos en los que *el beamforming* analógico y digital es ineficiente y complejo. • Problema con la variación del ángulo para anchos de banda grandes (*beam squint*).

En la arquitectura analógica, donde las señales se manipulan electrónicamente, las variaciones en las fases pueden ser especialmente problemáticas. Esto se debe a que los componentes electrónicos analógicos pueden ser susceptibles a variaciones de temperatura y envejecimiento, además de otros cambios que pueden afectar las características de fase de las señales. Además, calibrar y mantener la coherencia de fase en sistemas analógicos puede resultar difícil y costoso en términos de recursos y tiempo. El efecto del *beam squint* es acuciante, y puede afectar la precisión del sistema de antenas [18], [19], [20], [21], [22], [23], [24].

En la arquitectura digital, las señales se procesan utilizando técnicas digitales, lo que ofrece varias ventajas significativas en términos de control y precisión. En sistemas digitales, las fases de las señales pueden ser ajustadas y controladas con mayor precisión, lo que permite mitigar el efecto del *beam squint*. Además, las técnicas de calibración digital pueden compensar automáticamente los errores de fase, lo que ayuda a mantener el rendimiento del sistema de antenas de manera más eficiente y confiable [18], [19], [20], [21], [22], [23], [24].

El *beam squint* tiende a ser más significativo en la arquitectura analógica debido a las limitaciones en el control y la calibración precisa de las fases de las señales. La formación del haz en sistemas analógicos se basa en el ajuste de fases y amplitudes de las señales sobre las antenas de la agrupación, y estas fases pueden ser difíciles de controlar con precisión para un rango amplio de frecuencias. En cambio, la arquitectura digital ofrece mejores capacidades de control y compensación, haciéndola menos vulnerable al efecto del *beam squint* y más adecuada para aplicaciones que requieren una alta precisión y fiabilidad en *el beamforming*. En sistemas digitales, el procesamiento de señales se realiza a nivel de bits, y puede incluir técnicas avanzadas de procesamiento que permiten ajustar dinámicamente las fases y amplitudes para corregir el desvío del haz [18], [19], [20], [21], [22], [23], [24].

2.6. Conclusiones

Los *beamformers* digitales evitan las limitaciones en la cantidad de *beams* que se pueden generar con desfasadores analógicos, así como las del receptor de RF. Las principales restricciones son el número de puertas y el consumo de energía. Uno de los atributos fundamentales de los *beamformers* digitales es la capacidad de generar o recibir una gran cantidad de *beams*. A continuación, se indican otras ventajas que incluye este tipo de tecnología [35]:

- Permite la conformación de haces de banda ancha mediante la implementación integrada de retardo en tiempo real (*True Time Delay,* TTD).
- Elimina los requisitos de coincidencia de longitud de línea en las redes de distribución de energía de RF, que también proporciona TTD.

- Mejora el rendimiento del perfil nulo y del lóbulo lateral con control preciso de amplitud y fase.
- El muestreo de RF directo y la generación de señales eliminan las etapas de conversión de frecuencia.
- Permite la formación de haces adaptables mediante control digital por cada elemento de la agrupación.
- Compensa las características de ancho de banda de los amplificadores de RF.
- Mejora de la linealidad mediante linealización digital previa, y posterior a la distorsión.
- Simplifica el diseño físico de los *arrays*, reduciendo su tamaño y peso [35].

Debido a estas ventajas, los *beamformers* digitales son escogidos para la realización de este TFM.

Capítulo 3. Diseño del algoritmo de conformación de haz en Matlab

3.1. Introducción

En este capítulo se presenta el desarrollo del *beamformer* en Simulink®. Se lleva a cabo una comparación entre un modelo basado en un algoritmo de MATLAB que realice el desfase temporal y un modelo generado en Simulink [36]. Para el modelado en Simulink se tiene en cuenta lo siguiente [36]:

1. Técnicas de procesamiento de datos eficientes en el diseño de hardware que permitan equilibrar los recursos y el rendimiento.
2. Para generar el código HDL a partir de un modelo, deben incluirse los bloques necesarios entre los subsistemas de Simulink, además de cumplir las restricciones de diseño.
3. Para comparar los resultados del modelo descrito en código Matlab y la implementación sobre FPGA, se necesita agregar un retardo al modelo descrito en código Matlab y alinear en el tiempo el resultado obtenido [36].

3.2. Matlab

Matlab es un entorno de programación de alto nivel orientado principalmente al cálculo numérico, visualización de datos y simulación. Fue desarrollado por MathWorks y se utiliza ampliamente en la industria, la investigación y la educación para resolver problemas científicos y de ingeniería. Matlab utiliza su propio lenguaje de programación y proporciona una interfaz de usuario con herramientas gráficas para facilitar la visualización y el análisis de datos. Además, incluye una amplia gama de funciones matemáticas y científicas, así como la creación de gráficos en 2D y 3D. Matlab permite la interacción con otros lenguajes de programación como C/C++, Java y Python. Además, ofrece numerosos *toolboxes* especializados para distintas áreas que van desde procesamiento de señales hasta el control de sistemas y procesamiento de imágenes, entre otros.

La *toolbox* de Matlab que se utiliza en este TFM es *Phased Array System Toolbox*. Se trata de una herramienta fundamental para la ingeniería de telecomunicaciones, y está orientado hacia el diseño y análisis de *arrays* de antenas en fase. Proporciona una amplia gama de funciones y algoritmos que son importantes para la simulación y modelado de sistemas de *arrays* de antenas en fase en una variedad de aplicaciones que van desde los radares hasta las comunicaciones inalámbricas. Una característica destacada es la capacidad para simular el comportamiento de antenas de *array* en fase en entornos complejos y dinámicos, permitiendo evaluar su rendimiento en distintas condiciones y optimizar el diseño para cumplir con requisitos específicos. Esta *toolbox* proporciona herramientas para la generación de diagramas de radiación, *beamforming*, seguimiento de objetivos, etcétera. Con ello se facilita el proceso de diseño y desarrollo de estos sistemas, ahorrando tiempo y recursos. En este trabajo se emplea para el diseño y verificación del algoritmo funcional de coma flotante, que proporciona el modelo de referencia de comportamiento. Se emplean funciones como las siguientes:

- *phased.ULA:* para crear un *array* lineal uniforme.
- *Arrayfactor:* para calcular el factor de *array* de una configuración de antenas en función de la dirección de la señal incidente.
- *phased.SteeringVector:* para calcular el vector de dirección de un *array* de antenas.
- *phased.ArrayResponse:* para calcular la respuesta de un sistema de antenas de matriz de fase para una señal de entrada.
- *phased.WidebandCollector:* para modelar un receptor de banda ancha que pueda recibir señales desde múltiples direcciones y a distintas frecuencias.

3.3. Simulink®

Simulink es una herramienta de modelado y simulación desarrollada por MathWorks. Su principal uso es en ingeniería, en áreas como el control automático, el procesamiento de señales y las comunicaciones. Permite diseñar y simular sistemas complejos mediante la creación de modelos visuales,

proporcionando un entorno gráfico basado en bloques. Las principales características son la construcción de modelos mediante conexiones por bloques gráficos, una biblioteca con una amplia variedad de bloques predefinidos, la simulación de sistema, herramientas para analizar y visualizar los resultados de las simulaciones y generar códigos ejecutables a partir de los modelos creados, facilitando la implementación de sistemas hardware específicos sobre FPGA. Al estar integrado con Matlab, permite una combinación de ambas herramientas de manera sencilla para los usuarios. Simulink se puede usar para distintos tipos de proyectos, como pueden ser comunicaciones inalámbricas, sistemas de control, procesamiento de señales, sistemas autónomos y robótica, sistemas avanzados de asistencia al conductor, inteligencia artificial y gemelos digitales, entre otros.

Simulink será la herramienta principal que se utilice en este TFM para la creación por bloques de un sistema de *beamforming* adecuado para su implementación en hardware. Estos bloques se comparan con las funcionalidades de distintas *toolboxes* de Matlab. Las herramientas de Simulink que se emplean en este TFM son HDL Coder, Fixed-Point Designer y HDL Verifier.

HDL Coder proporciona un entorno para diseñar, simular y desplegar algoritmos y sistemas en hardware programable, como FPGAs y *Application-Specific Integrated Circuit* (ASICs). Con esta herramienta se pueden convertir algoritmos y modelos desarrollados en Matlab y Simulink directamente en código HDL como Verilog o *Very High-Speed Integrated Circuit Hardware Description Language* (VHDL). Esto permite que se puedan implementar de manera sencilla algoritmos complejos en hardware digital, aprovechando la capacidad de procesamiento paralelo y flexibilidad de los dispositivos programables. En este caso se utiliza para generar automáticamente el código HDL de los bloques de diseño de alto nivel creados en Simulink, lo que simplifica el flujo de diseño y reduce el tiempo de desarrollo. Se emplean las siguientes funciones de esta herramienta:

- *Hdlsetup:* para configurar el entorno de HDL Coder en Matlab estableciendo las opciones de compilación y síntesis para el diseño HDL.
- *Codegen*: para generar código HDL a partir de modelos de Simulink. Esta función toma el modelo Simulink como entrada y produce código HDL optimizado para la implementación en hardware.
- *Makehdl:* para generar archivos HDL para componentes específicos dentro de un modelo de Simulink. Es de gran utilidad para depurar y analizar partes específicas del diseño de HDL.
- *Hdlcoder.WorkflowConfig:* para configurar y ejecutar flujos de trabajo de HDL *Coder*, que incluye pasos como la simulación de co-síntesis y la generación de código HDL.

Fixed-Point Designer permite diseñar, simular y analizar sistemas de punto fijo como los utilizados en este trabajo, permitiendo optimizar el rendimiento y la eficiencia de sus algoritmos en implementaciones de hardware y software. Es crucial para aplicaciones donde la precisión de los cálculos es crítica, como en sistemas de procesamiento de señales de audio y vídeo, comunicaciones inalámbricas y procesamiento de imágenes. Una de las características destacadas de esta herramienta es su capacidad para simular el comportamiento de sistemas de punto fijo con precisión, teniendo en cuenta efectos como el desbordamiento de datos o la pérdida de precisión, y evaluando su rendimiento en diferentes condiciones mediante reajustes para optimizar el diseño. Además, proporciona herramientas avanzadas que convierten algoritmos de punto flotante a punto fijo y optimizan automáticamente cómo se representan los datos y las operaciones en un diseño, reduciendo el tiempo necesario para desarrollar los sistemas enfocándose en un diseño funcional. En este trabajo se utiliza para la conversión de algoritmos de punto flotante a fijo y para llevar a cabo simulaciones precisas a nivel de bit, así como para examinar cómo afecta el rango y la precisión sin necesidad de implementar el diseño sobre hardware.

Las funciones utilizadas son:

- *Fi*: se utiliza para definir y manipular números de punto fijo en Matlab. Permite especificar la longitud de palabra y de fracción, así como realizar operaciones aritméticas con números de punto fijo.
- *Fimath*: se utiliza para especificar propiedades aritméticas de los objetos de punto fijo.
- *Numerictype*: permite especificar las propiedades numéricas de los objetos de datos, incluyendo los números de punto fijo.
- *Quantizer*: se utiliza en la cuantificación y redondeo en diseños de punto fijo.

HDL Verifier se emplea para la verificación y validación de diseños hardware descritos en lenguajes como VHDL y Verilog. Proporciona un entorno para la co-simulación entre Matlab/Simulink y las herramientas de simulación HDL, como ModelSim o Vivado, lo que permite realizar más pruebas de los diseños antes de su implementación física. Facilita la verificación conjunta del diseño hardware y software, y también la generación de bancos de pruebas, la verificación de prototipos, la integración y pruebas de componentes, el análisis de rendimiento y la depuración de diseño.

3.4. Algoritmo *delay and sum*

Una de las técnicas de *beamforming* más comunes y sólidas es el *beamforming* de retardo y suma, conocido como *Delay-and-Sum* o DAS. Esta técnica emplea un retardo y una ponderación de amplitud a la salida de cada antena y, posteriormente, combina las señales resultantes. Los retardos se seleccionan de manera que se maximice la sensibilidad del conjunto a las señales provenientes de una dirección específica. Ajustando estos retardos, se puede orientar la dirección hacia la que se enfoca o apunta la agrupación de antenas (fuente), y las formas de onda capturadas por las diferentes antenas se combinan de manera constructiva. Esto implica que las señales en ciertos ángulos experimenten interferencia constructiva, y destructiva en otros [37].

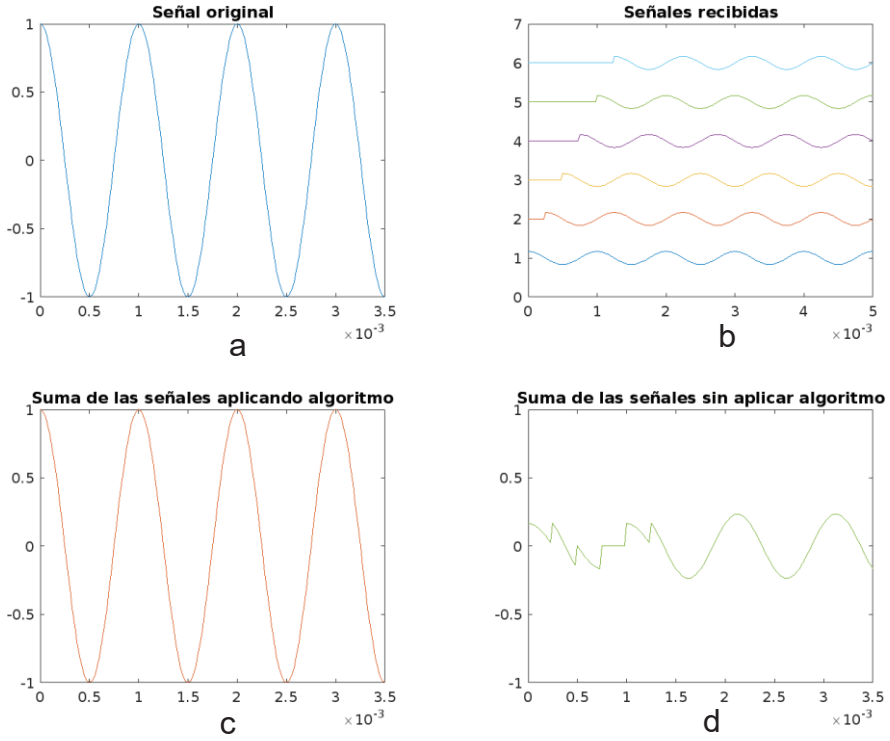


Figura 10: Explicación gráfica del delay-and-sum: a) Señal cosenoidal a transmitir en banda base. b) Señales recibidas ya pasadas también a banda base después de recibirlas. c) Señal reconstruida aplicando el algoritmo de delay-and-sum y sumando las señales. d) Si no se aplicará el algoritmo.

En la Figura 10 se utiliza un ejemplo ideal en el que no se considera ruido. Se observa que si no se utiliza el algoritmo DAS, no se recuperaría la señal original tras sumar las señales. En la Figura 10(a) aparece la señal cosenoidal original que se va a transmitir en banda base, en la Figura 10(b) la señal recibida en banda base, en la Figura 10(c) la suma de las señales recibidas aplicando el algoritmo DAS para reconstruirla y obtener la señal original, y en la Figura 10(d) se observa la suma de las señales sin aplicar el algoritmo DAS, donde no se obtendría la señal original.

El algoritmo DAS permite combinar las señales de salida de las antenas aplicando un retardo y un peso de amplitud w_m^* a cada una, para finalmente sumarlas y formar un haz único. Esto se observa en la Figura 11, donde se considera una agrupación compuesta por M antenas ubicadas en distintas posiciones del espacio $\vec{x}_m = [x_m, y_m, z_m]$, los cuales registran un frente de onda plano $f(\vec{x}, t)$.

La forma de onda espacialmente muestreada por la m-ésima antena se representa como $y_m(t) = f(\vec{x}_m, t)$. El proceso de *beamforming* DAS implica aplicar un retardo Δ_m y un factor de amplitud w_m a la salida de cada sensor, seguido por la suma de las señales resultantes [37].

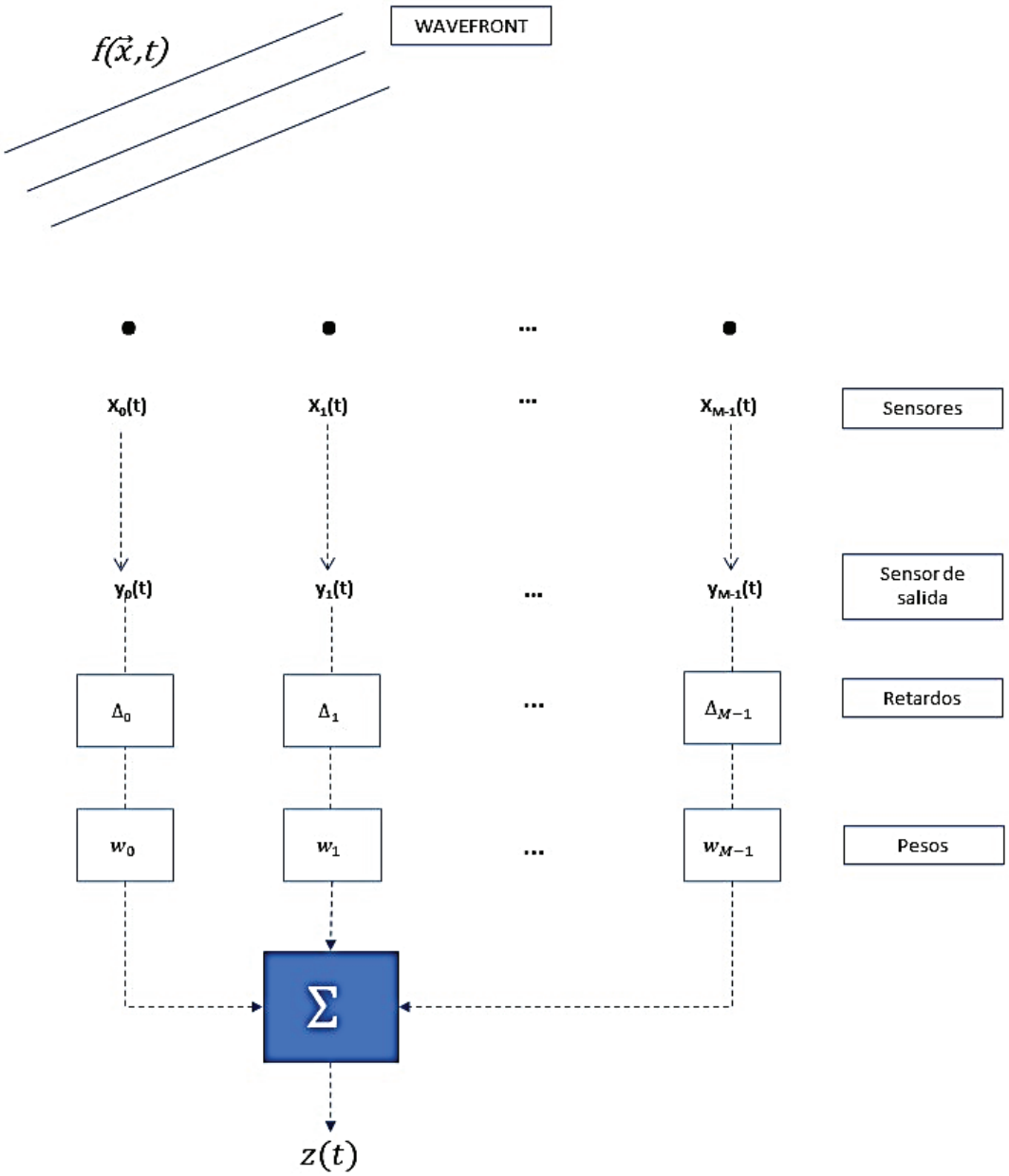


Figura 11: Delay-and-sum

A diferencia de los apuntamientos adaptativos, los coeficientes de ponderación de los sensores para el *beamformer* DAS se determinan previamente sin considerar la forma de onda recibida. Su salida en el dominio temporal se describe mediante la Ecuación (5) [37]:

$$\mathrm{z(t)} = \sum_{m=0}^{M-1} w_m * y_m(\mathrm{t} - \Delta_m) \tag{5}$$

La principal idea de este tipo de *beamformer* radica en emplear una serie de retardos para direccionar la agrupación hacia diversas direcciones o puntos dentro de un plano de exploración. Cuando la orientación coincide con una fuente, se registra una potencia de salida máxima. Al interpolar la potencia de salida medida en todos los puntos de exploración, se pueden asignar colores a la potencia espacial, es decir, la potencia en todo el plano de exploración, y generar una imagen acústica [37]. Si se define el grupo de ubicaciones de recepción en el plano de exploración como $\vec{x}_s = [x_s, y_s, z_s]$, los retardos Δ_m necesarios para dirigir el haz hacia un punto particular se determinan mediante la Ecuación (6), donde c es la velocidad del sonido [37]:

$$\Delta_m = \frac{|\vec{x}_s - \vec{x}_m|}{c} = \frac{\sqrt{(x_s - x_m)^2 + (y_s - y_m)^2 + (z_s - z_m)^2}}{c} \tag{6}$$

Cabe destacar que este algoritmo es bastante importante en el diseño de este TFM debido a que, si no se usara, no se podría obtener la misma señal de recepción. Al emplear conjuntos de retardos para dirigir espacialmente la dirección de la agrupación de antenas se observa que, si su dirección de orientación coincide con el de una fuente de radiación, la potencia de salida se maximiza. Si se mide la potencia de salida en todo el plano de exploración y se realiza una interpolación, se representa la potencia espacial para generar una imagen acústica. En la Figura 12 se ilustra cómo el *beamforming* permite a un *array* de antenas concentrar la señal en una dirección específica y ajustarla para explorar diferentes áreas en el espacio, mejorando la eficiencia y precisión tanto en la trasmisión como recepción de señales. En la figura, d es la distancia de separación entre el *array* de antenas y el plano de exploración, y el patrón de radiación es la forma con la que la señal se focaliza en una cierta dirección.

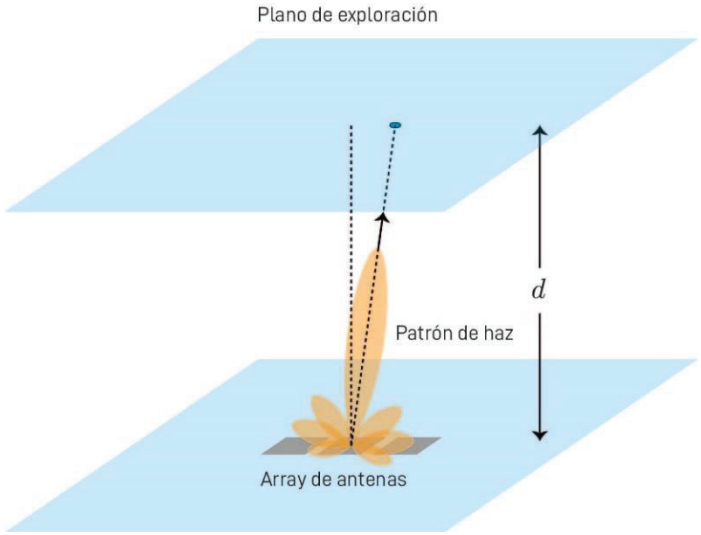


Figura 12: Imagen visual de la idea de beamforming

3.5. Algoritmo de conformación de haz en Matlab

A continuación, se explica el algoritmo funcional *beamformer* de cambio de fase que se ha reimplementado en un subsistema de algoritmo HDL, y que utiliza bloques de Simulink para la generación de código HDL. El *beamformer* realiza cálculos de la fase requerida para cada uno de los diez canales, con el objetivo de maximizar la potencia de la señal recibida a lo largo del ángulo de incidencia. La Figura 13 muestra un modelo de Simulink con el algoritmo operativo en Matlab y el algoritmo de implementación de FPGA [36].

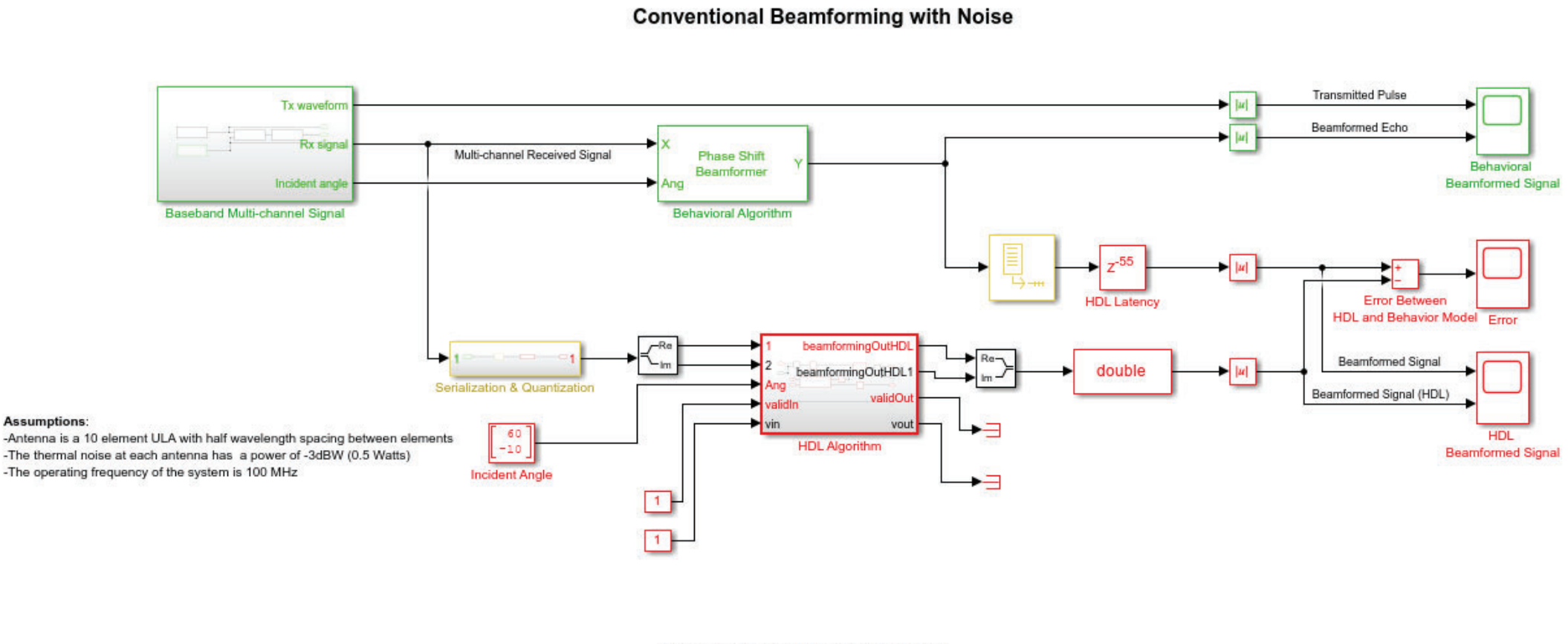


Figura 13: Modelo de Simulink con los algoritmos operativos y de implementación.

El modelo de Simulink tiene dos ramas. Las señales de entrada son un pulso de onda rectangular y el ángulo incidente obtenido a la salida, el pulso transmitido, el eco conformado por el haz, el error entre HDL y el modo de comportamiento, la señal formada por el haz y la señal HDL formada por el haz. El brazo superior es un modelo del comportamiento de punto variable del algoritmo y el brazo inferior es una versión de punto fijo funcionalmente equivalente que utiliza bloques de soporte de generación de código HDL. Aparte de representar gráficamente las salidas de las dos ramas para compararlas, también calcula y representa la diferencia o error entre ambas salidas.

La antena es un *array* linealmente uniforme o ULA (*Uniform Linear Array*) de diez elementos con una distancia de media onda entre ellos. La potencia de ruido térmico es de -3dBW (0,5 vatios) para cada antena, y la frecuencia de operación es de 100 MHz.

El primer elemento de la primera rama (*Baseband Multi-channel Signal*) es una señal multibanda que presenta tres canales: forma de onda transmitida, señal recibida y ángulo de incidencia, mientras que el segundo elemento es un *beamformer* basado en retardos. Las entradas de este bloque son un pulso de onda rectangular y el ángulo incidente, obteniendo a la salida la forma de onda de transmisión, la señal de recepción (señal multicanal de banda base) y el ángulo incidente. La principal función de este bloque es generar señales multicanal de banda base para su simulación.

El primer elemento amarillo (*buffer*) de la Figura 13, siguiendo las ramas que salen del elemento *Baseband Multi-channel Signal*, transforma rápidamente la gráfica en una salida muestreada escalar. La señal de entrada es la que sale del bloque de *Behavioral Algorithm* y la de salida en la señal escalar muestreada. Este bloque toma los sub-bloques en los que está dividida la señal (*buffers)* y los une para restaurar la señal original. El segundo elemento (HDL *Latency*), de color rojo, crea un retardo de la señal de entrada por un cierto número de muestras (55 en este caso). La señal de entrada es la que se obtuvo en el bloque anterior y la de salida, la señal con retardo. Este bloque permite especificar el retardo asociado con la implementación de un algoritmo en hardware.

El modelo tiene un bloque de latencia en la salida del algoritmo de movimiento. Este retardo es necesario porque el algoritmo de implementación se retarda 55 veces para implementar la canalización. Esto introduce un retardo que debe ser considerado. El cálculo de este retardo se denomina ecualización de retardo, siendo necesario organizar los tiempos de salida del modelo descrito en código Matlab y el modelo de rendimiento para que los resultados puedan compararse fácilmente [36]. A continuación, se explican los distintos bloques de manera interna.

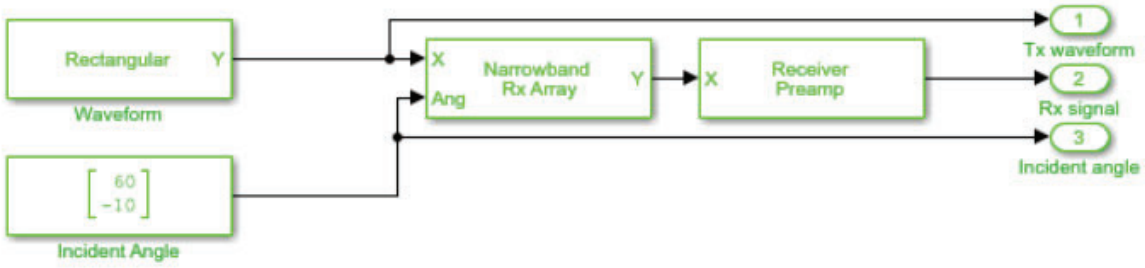


Figura 14: Señal de recepción multicanal.

Para agregar la señal recibida en el *array* de antenas en fase, el modelo incluye subsistemas que generan señales multicanal. Este modela las formas de onda transmitidas y los ecos de destino recibidos en ángulos de incidencia recogidos por un *array* de antenas de 10 elementos. El subsistema también incluye un modelo de preamplificador de receptor para calcular el ruido del receptor. Este subsistema genera estímulos de entrada para los patrones de comportamiento y rendimiento del modelo [36]:

1. *Rectangular Waveform*: genera una forma de onda de pulso rectangular con un ancho de pulso y un *Pulse Repetition Frequency* (PRF) definidos. La señal es una muestra o un número entero de pulsos.
2. *Narrowband Receive Array*: recibe las ondas planas de banda estrecha que atacan los elementos de la agrupación de antenas.
3. *Receiver Preamp:* modelo que considera el ruido del receptor. [36].

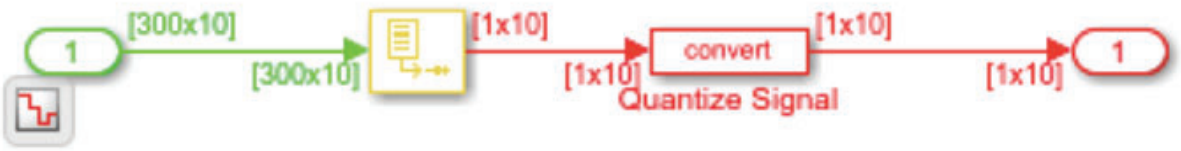


Figura 15: Serialización y cuantización.

Este modelo incluye subsistemas de serialización y cuantificación que convierten señales de punto flotante en señales de punto fijo para modelar datos de transmisión en hardware. Utiliza procesamiento de flujo escalar porque el algoritmo opera por debajo de 400 MHz. Las implementaciones de hardware se pueden optimizar para los recursos en lugar del rendimiento listo para usar.

La señal de entrada para el subsistema de serialización tiene 10 canales de 300 muestras cada uno, o un tamaño de señal de 300x10. El subsistema serializa y genera una señal basada en 1x10 muestras, es decir, una muestra por canal, cuantificada para cumplir con los requisitos del sistema. El tipo de datos de salida del bloque es un valor con signo con una longitud de palabra de 12 bits y una longitud de fracción de 9 bits.

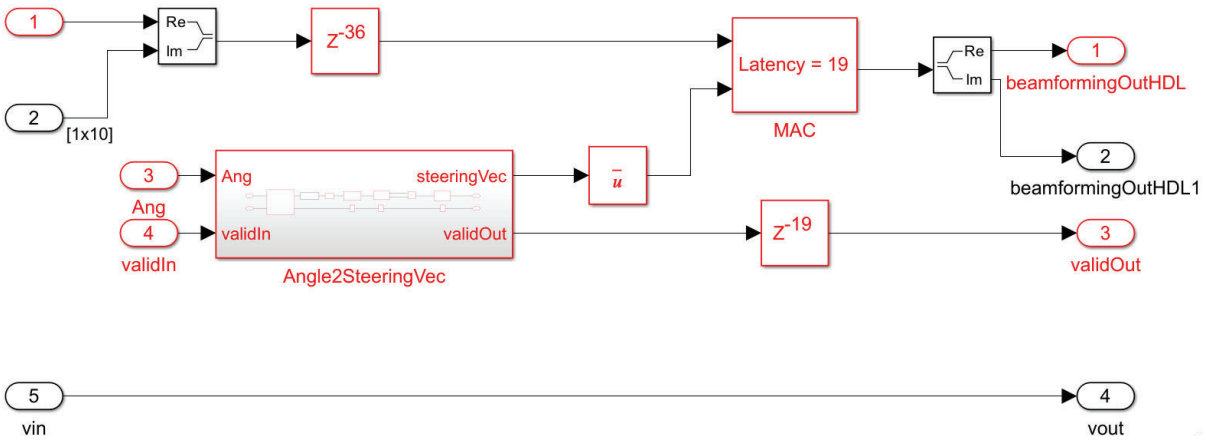


Figura 16: Diseño del subsistema de implantación.

El subsistema de algoritmos HDL para la generación de código implementa un *beamformer* diseñado con bloques *Simulink* adecuados a esta aplicación.

El subsistema Angle2SteeringVec calcula el retardo de la señal en cada elemento de la antena en una matriz lineal unificada. Los retardos se pasan al subsistema de multiplicación y acumulación (MAC) para el *beamforming*.

El algoritmo del subsistema (algoritmo HDL) es funcionalmente equivalente al algoritmo de DAS, pero además puede generar código HDL. Hay tres diferencias principales que permiten que este subsistema genere eficientemente código HDL [36]:

1. Diseña un proceso de entrada de datos en serie.
2. El subsistema utiliza tipos de datos de punto fijo para todos los cálculos.
3. La implementación incluye latencia para permitir el *pipelining* de la herramienta de síntesis HDL [36].

Para garantizar una sincronización de reloj adecuada, el retardo agregado a una rama del modelo de implementación debe coincidir con todas las demás ramas paralelas. El subsistema Angle2SteeringVec tiene un desplazamiento de 36 retardos, por lo que el brazo superior del subsistema del algoritmo HDL contiene un retardo de 36 muestras justo antes del subsistema MAC. Además, el subsistema MAC utiliza 19 retardos. Este retardo debe equilibrarse agregando 19 retardos a la salida del subsistema Angle2SteeringVec. A continuación, se muestra en la Figura 17 el interior del subsistema MAC para ilustrar estos 19 retardos [36].

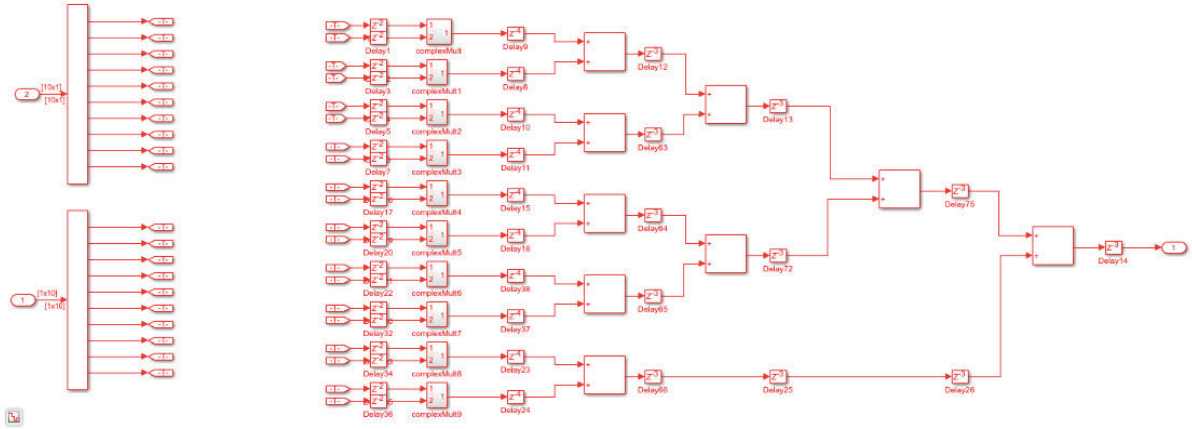


Figura 17: Interior del subsistema MAC.

El siguiente bloque, mostrado en la Figura 18, se emplea para calcular el vector de dirección.

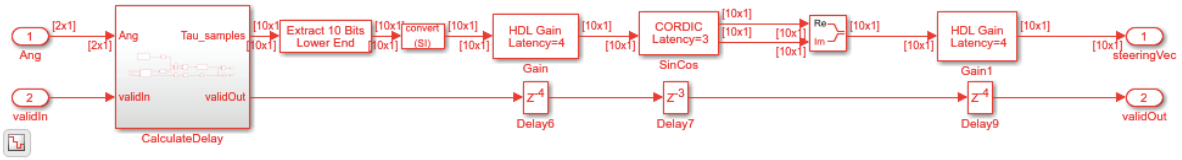


Figura 18: Cálculo del vector de dirección.

El subsistema Angle2SteeringVec calcula un vector direccional a partir del ángulo de llegada de la señal. Primero, se calcula el retardo de la señal de entrada en cada antena multiplicando la posición de la antena en el *array* por la dirección de propagación de la señal. Luego, los retardos se pasan al subsistema *SinCos*, que calcula las funciones trigonométricas de seno y coseno utilizando el algoritmo *Coordinate Rotation Digital Computer* (CORDIC) para realizar el *beamforming* final [36].

El algoritmo CORDIC, propuesto por Jack E. Volder, fue creado con el objetivo de calcular funciones trigonométricas mediante la rotación de vectores. El algoritmo fragmenta el ángulo de rotación en una suma de ángulos y ejecuta la rotación mediante una serie de micro-rotaciones a lo largo de estos ángulos. Para estas rotaciones utiliza únicamente operaciones de suma, resta y desplazamiento bit a bit, lo que lo hace eficiente en términos de hardware y fácil de implementar en sistemas digitales.

Este algoritmo tiene dos modos de funcionamiento: rotación y vectorización. En el modo de rotación se gira el vector de entrada según un ángulo específico proporcionado como parámetro, mientras que en el modo vectorización, el algoritmo rota el vector de entrada hacia el eje X, almacenando gradualmente el ángulo necesario para lograr esa rotación [38], [39].

Dado que este diseño consta de 10 elementos ULA espaciados a intervalos de media longitud de onda, la ubicación de las antenas se basa en la distancia entre cada una de ellas medida hacia afuera desde el centro de la red. El espaciado entre elementos se define como un vector de 10 números que van desde -6.7453 a 6.7453, correspondiente a media longitud de onda. El paso entre elementos de la antena sería aproximadamente de 1,498 media longitud de onda. En la aritmética de punto fijo, el tipo de datos que se utiliza para los vectores de distancia de los elementos es *fixdt* (1,8,4). Este es un valor con signo con una longitud de palabra de 8 bits y una longitud de fracción de 4 bits [36].

Finalmente, se explican los últimos bloques que corresponden con la deserialización, mostrados en la Figura 19.

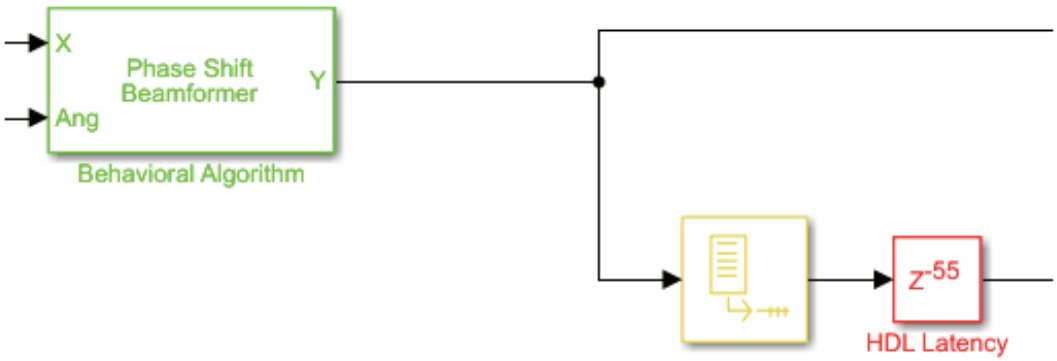


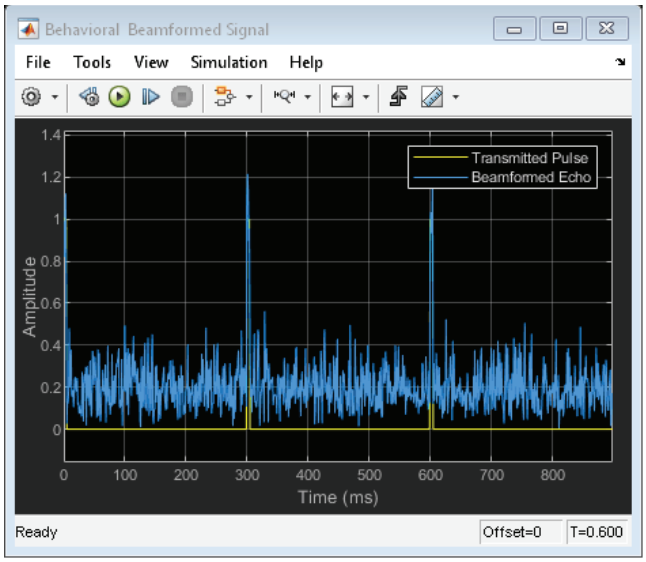
Figura 19: Deserialización

Para comparar la implementación de punto fijo basada en muestras con el diseño de comportamiento basado en marco de punto flotante, la salida del subsistema de implementación debe decodificarse y convertirse a un tipo de datos de signo. Los resultados también se pueden comparar directamente con las señales basadas en muestras, pero la salida del modelo descrito en código Matlab debe estar sin *buffer* para que coincida con la salida de la señal basada en muestras del algoritmo de implementación, como se muestra en la Figura 19.

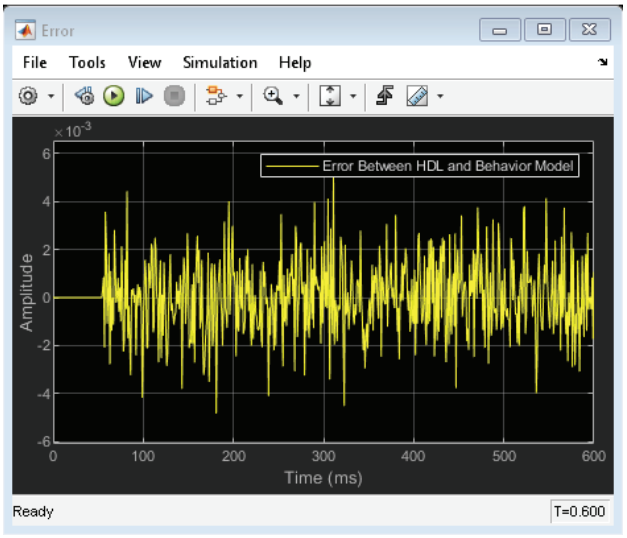
En este caso, se convierte la salida del subsistema del algoritmo HDL a punto flotante y se configura la salida del bloque de conversión de tipo de datos en doble [36].

3.6. Resultado

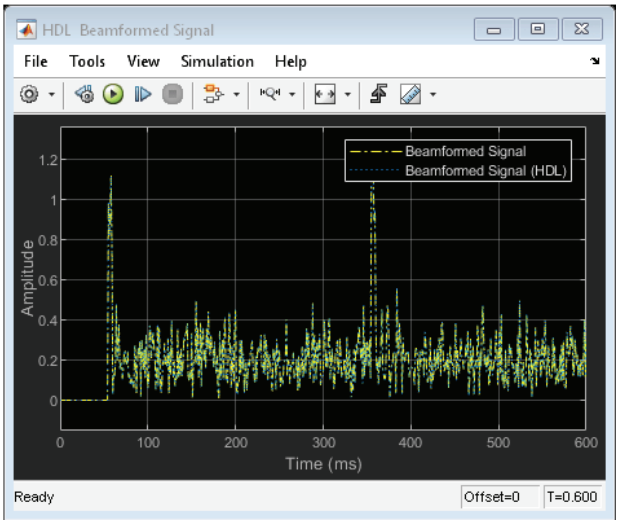
Si se ejecutan los bloques explicados anteriormente se obtienen las gráficas indicadas en la Figura 17 (a), (b) y (c).



(a)



(b)



(c)

Figura 20: Resultados de la simulación. (a) Behavioral Beamformed Signal (b) Error (c) HDL Beamformed Signal

En la Figura 20 (a) se muestra las señales *beamformed* tanto del modelo descrito en código Matlab como del modelo HDL a lo largo del tiempo. Permite una comparación visual de las señales para evaluar su similitud y alineación. La señal *beamformed* es el resultado de combinar varias señales de entrada con un retardo específico para enfocar el haz en la dirección deseada. Se muestra cómo se procesan y ajustan las señales basándose en los desplazamientos de fase calculados para maximizar la potencia de la señal recibida en la dirección del ángulo incidente.

En la Figura 20 (b) se muestra la diferencia o error entre las salidas del modelo descrito en código Matlab y del modelo HDL. Cuantifica la diferencia entre las dos salidas, normalmente en términos de error de cuantización u otros factores que afectan a la precisión de los modelos. Para el algoritmo *beamforming* por desplazamiento de fase utilizado, esta figura indica el nivel de error de cuantización o diferencias en las señales *beamformed* debido a la implementación en el modelo HDL.

En la Figura 20 (c) se representa la señal *beamformed* de salida del modelo HDL. Proporciona información sobre el rendimiento de la implementación HDL al mostrar la señal en comparación con la señal original. Esta figura puede mostrar cualquier retardo o diferencia en la señal *beamformed* generada por la implementación HDL en comparación con la señal original transmitida o *beamformed*.

El algoritmo utilizado en esta comparación es un formador de haces por desplazamiento de fases usando retardos, que calcula los ajustes de fase necesarios para cada canal con el fin de maximizar la potencia de la señal recibida en la dirección del ángulo incidente. Este algoritmo se implementa tanto en el modelo descrito en código Matlab como en el modelo HDL para permitir una comparación directa de sus resultados. Analizando estas graficas se puede evaluar el rendimiento y la precisión de la implementación HDL en comparación con el modelo descrito en código Matlab, garantizando que el formador de haces basado en FPGA funciona según lo previsto.

Las duraciones de las señales de *beamformed* y *beamformed* (HDL) muestran que las dos señales son casi idénticas. EL error es de 10^{-3}, lo que indica que la diferencia entre las dos señales es muy pequeña. Este resultado muestra que el subsistema del algoritmo HDL produce la misma salida, dentro del error de cuantificación, que el modelo descrito en código Matlab. Esta verificación es un primer paso importante antes de generar código HDL. El modelo HDL utiliza un retardo de 55 ms, por lo que el dominio etiquetado como *HDL Beamformed Signal* tiene un retardo de 55 ms con respecto a la forma original que se muestra en *Behavioral Beamformed Signal* [36].

El algoritmo de implementación en el modelo HDL utiliza 55 retardos para la canalización. Esto significa que el procesamiento se divide en 55 etapas, cada una de las cuales introduce una cierta cantidad de latencia. Para garantizar que las salidas del modelo descrito en código Matlab y del modelo HDL estén alineadas en el tiempo para la comparación, se añade un retardo correspondiente de 55 ms al modelo descrito en código Matlab. Así, las salidas de ambos modelos estarán sincronizadas, haciendo más fácil comparar los resultados y verificar el sistema.

3.7. Conclusiones

En este capítulo se explica el uso de Simulink para diseñar un *beamformer* adecuado para ser implementado en una FPGA. Se muestra el diseño, así como el proceso de comparación entre los resultados del modelo de rendimiento frente a los resultados del modelo descrito en código Matlab. También se explica la importancia del algoritmo DAS para recuperar la señal original y los bloques utilizados para este diseño. Además, se han explicado las herramientas y librerías utilizadas en la creación de este modelo.

Capítulo 4. Estudio FPGA

4.1. Introducción

En este capítulo se explica la plataforma FPGA que se emplea para la futura implementación del diseño *beamformer* creado, así como las herramientas utilizadas para el desarrollado de este TFM.

4.2 Estudio de la plataforma

Para el desarrollo de este proyecto se ha elegido la FPGA Arty z7-20 que usa el *System-on-Chip* (SoC) Zynq-7000. Esta plataforma se representa en la Figura 22. La familia SoC Zynq-7000 combina la capacidad de programación de software de un procesador basado en ARM® con la flexibilidad de programación de hardware de una FPGA. Esto posibilita la ejecución de análisis clave y la aceleración de hardware, al mismo tiempo que incorpora en un solo dispositivo *Central Processing Unit* (CPU), *Digital Signal Processor* (DSP), *Application Specific Standard* (ASSP) y funciones de señal mixta. Conformada por dispositivos Zynq 7000S de un solo núcleo y Zynq 7000 de doble núcleo, la familia Zynq 7000 proporciona una plataforma SoC completamente adaptable. Esta familia puede utilizarse en distintas aplicaciones como: asistencia al conductor, cámaras inteligentes, control de motores industriales, diagnóstico médico por imágenes, equipos de vídeo y de visión nocturna, y otras más.

La arquitectura Zynq-7000 posibilita la incorporación de lógica personalizada en el *Programmable Logic* (PL) y software adaptado en el *Processing System* (PS). Esto facilita la ejecución de funciones exclusivas y distintas. La fusión del PS con el PL posibilita niveles de rendimiento que las soluciones de dos chips, como un ASSP con una FPGA, no pueden alcanzar debido a sus restricciones en cuanto a ancho de banda de E/S, latencia y consumo de energía. Utiliza una interconexión ARM AMBA AXI que simplifica la creación de plataformas al admitir IP desarrollados por terceros. A continuación, en la Figura 21 se puede observar los bloques funcionales de la arquitectura [40].

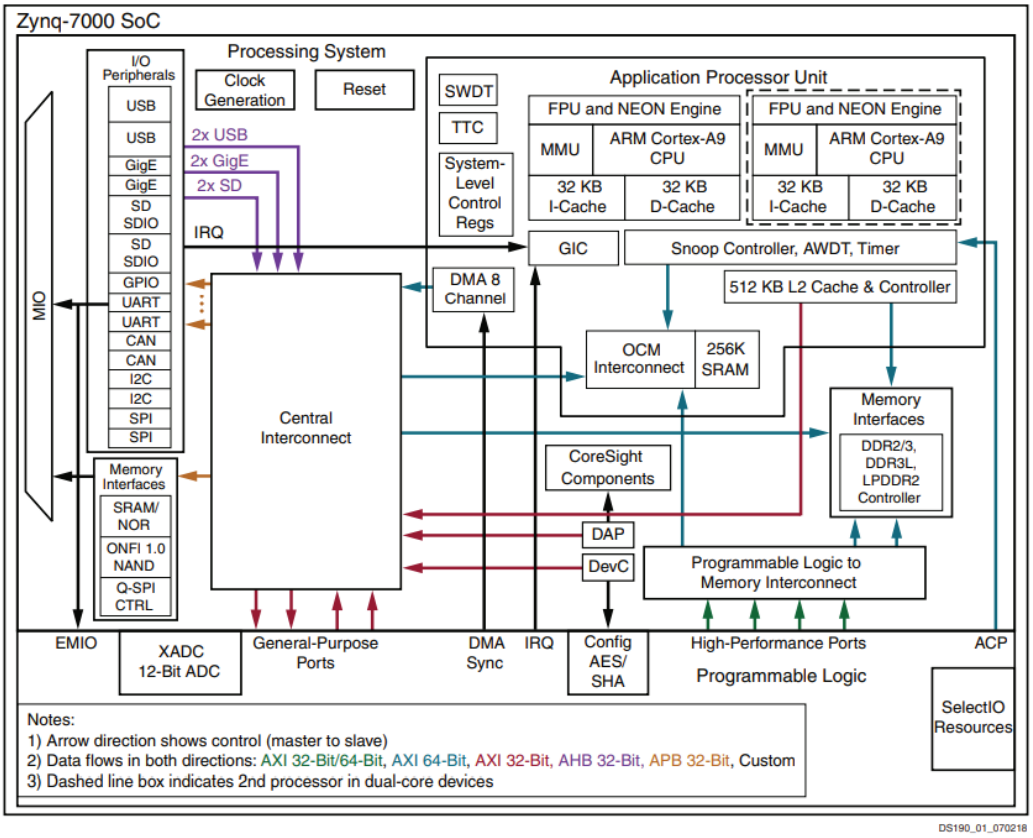


Figura 21: Bloques funcionales de la arquitectura Zynq-7000 [40]

La plataforma en la que se basa el desarrollo de este trabajo es la FPGA Arty z7-20, la cual se muestra en la Figura 22.

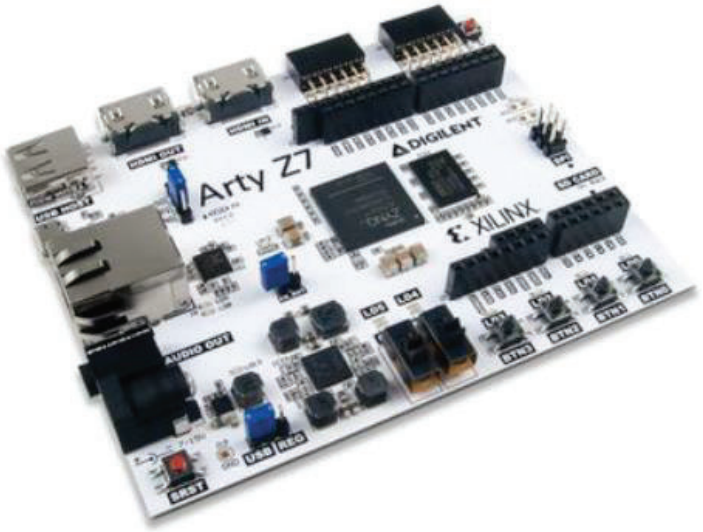


Figura 22: FPGA Arty z7-20

Las características de este tipo de FPGA incluyen un procesador Cortex-A9 de doble núcleo a 650MHz, un controlador de memoria DDR3 de 512 MB con 8 canales DMA y 4 puertos esclavos AXI3 de alto rendimiento. Contiene tanto controladores periféricos de alto ancho de banda (Ethernet 1G, USB 2.0, SDIO) como de bajo ancho de banda (SPI, UART, CAN, I2C). Se alimenta por USB o por una fuente externa de alimentación de 7V-15V.

Esta placa es compatible con Vivado Design Suite de alto rendimiento de Xilinx. Utiliza una Arquitectura Zynq APSoC que se subdivide en dos sistemas diferentes, la PL y PS. En la Figura 23 se expone una descripción genérica de esta arquitectura donde el PL se representa en color amarillo y en verde claro el PS. Estos dispositivos están configurados en torno al procesador, que funciona como el controlador principal en la gestión del tejido lógico programable y coordina todas las demás unidades periféricas dentro del chip del sistema de procesamiento.

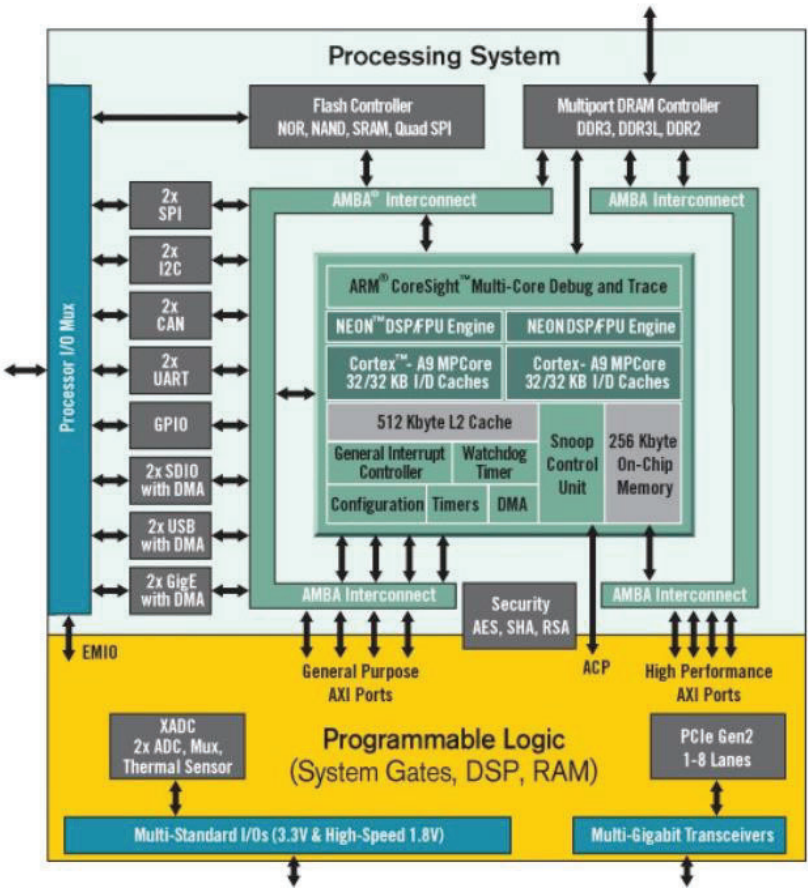


Figura 23: Arquitectura Zynq APSoC

Esta placa se utiliza por su compatibilidad con Vivado, alto rendimiento y disponibilidad en el laboratorio donde se lleva a cabo este TFM. Es adecuada para distintas aplicaciones, y posee capacidad para ejecutar sistemas operativos como Linux, lo que permite tener un entorno software completo para las distintas aplicaciones.

4.3 Vivado

En este TFM se emplea el conjunto de herramientas de desarrollo hardware proporcionada por Xilinx, conocido como Vivado Design Suite. Estas herramientas se emplean para diseñar, implementar y programar dispositivos FPGA y SoC en las distintas fases del flujo de diseño hardware:

- Diseño de Alto Nivel (*High-Level Synthesis*, HLS): eleva a un nivel más alto de abstracción la descripción hardware del sistema, reduciendo el ciclo de diseño. Utiliza lenguajes de programación C/C++ o SystemC [41].
- Simulación y síntesis lógica: basadas en herramientas como VHDL o Verilog [42].
- Implementación y generación del *Bitstream*, y posteriormente se lleva a cabo una depuración y análisis del rendimiento.

El flujo de Vivado HLS se muestra en la siguiente Figura 24.

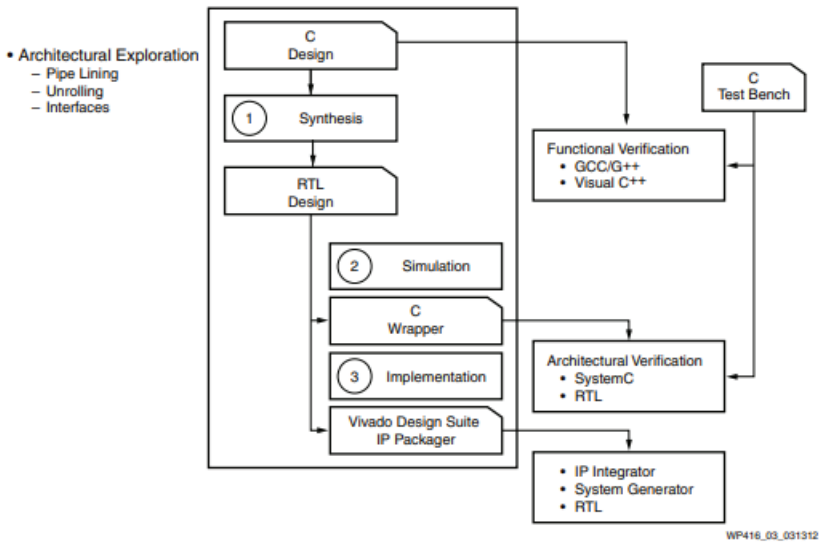


Figura 24: Flujo Vivado HLS

4.4. Conclusiones

En este capítulo se estudia tanto la arquitectura como las herramientas utilizadas para la FPGA escogida para este TFM. El uso de Vivado es fundamental para el diseño, ya que cubre desde la descripción de alto nivel hasta la implementación física del dispositivo. En el desarrollo de este TFM se trabaja principalmente con la simulación y síntesis lógica del diagrama de bloques generado con el diseño de Matlab.

Capítulo 5. Beamforming basado en FPGA

5.1. Introducción

En este capítulo se genera el código HDL a partir de los algoritmos previamente implementados en Matlab/Simulink. Además, se genera un modelo para un banco de pruebas de co-simulación que permite comprobar cómo se comporta el hardware generado en HDL.

5.2. Generación de códigos

En este apartado se explica el proceso para genera código HDL para el diseño creado mediante HDL Coder. En el Capítulo 3. Diseño del algoritmo de conformación de haz en Matlab se detalla el desarrollo de un algoritmo en Simulink adecuado para su implementación hardware en la FPGA Arty z7-20, estudiada en el capítulo anterior y que se utiliza a lo largo del TFM, así como la comparación entre la salida del modelo de implementación y la del modelo descrito en Matlab. Mediante el diseño de la Figura 13 totalmente configurado, se verifica y actualiza el modelo original para que sea compatible con la generación de código HDL.

Se emplea HDL Coder™ para generar código HDL a partir del modelo creado en el Capítulo 2, y HDL Verifier™ para verificar su funcionamiento. Esta herramienta genera un banco de pruebas de co-simulación para evaluar el comportamiento del código HDL producido. Se utiliza el modelo descrito en código Matlab para verificar los resultados de la implementación del algoritmo para hardware y el código HDL generado.

HDL Verifier es una herramienta que facilita la prueba y verificación de diseños Verilog y VHDL para FPGAs, ASICs y SoCs.

Una vez verificado que los resultados del modelo de implementación son iguales a los del modelo descrito en código Matlab, se procede a la generación del código HDL y la creación del banco de pruebas. En primer lugar, hay que ajustar los parámetros de generación de código HDL en el diálogo de configuración de parámetros de Simulink.

En este caso los ajustes son:

- Target: Synthesis tool: Xilinx Vivado; Family: Zynq-700; Device: xc7z020; Package: clg400; Target frequency: 200 MHz.
- Optimization: Uncheck all optimizations except Balance delays.
- Global Settings: Set the Reset type to Asynchronous.
- Test Bench: Select HDL *testbench* and Cosimulation model.

La desactivación de las optimizaciones en este modelo se justifica porque algunos de los bloques utilizados en esta implementación ya están optimizados para HDL, lo que podría entrar en conflicto con las optimizaciones realizadas por HDL Coder.

Seguidamente, se procede a la generación del código HDL y la creación del banco de pruebas utilizando HDL Coder para el subsistema de algoritmo de HDL con los siguientes comandos:

```
makehdl([modelname '/HDL Algorithm']);   % Generate HDL code
makehdltb([modelname '/HDL Algorithm']); % Generate Cosimulation test
bench
```

La generación del código HDL y la creación del banco de pruebas resultan en la creación de un modelo de Simulink adicional que se muestra en la Figura 25.

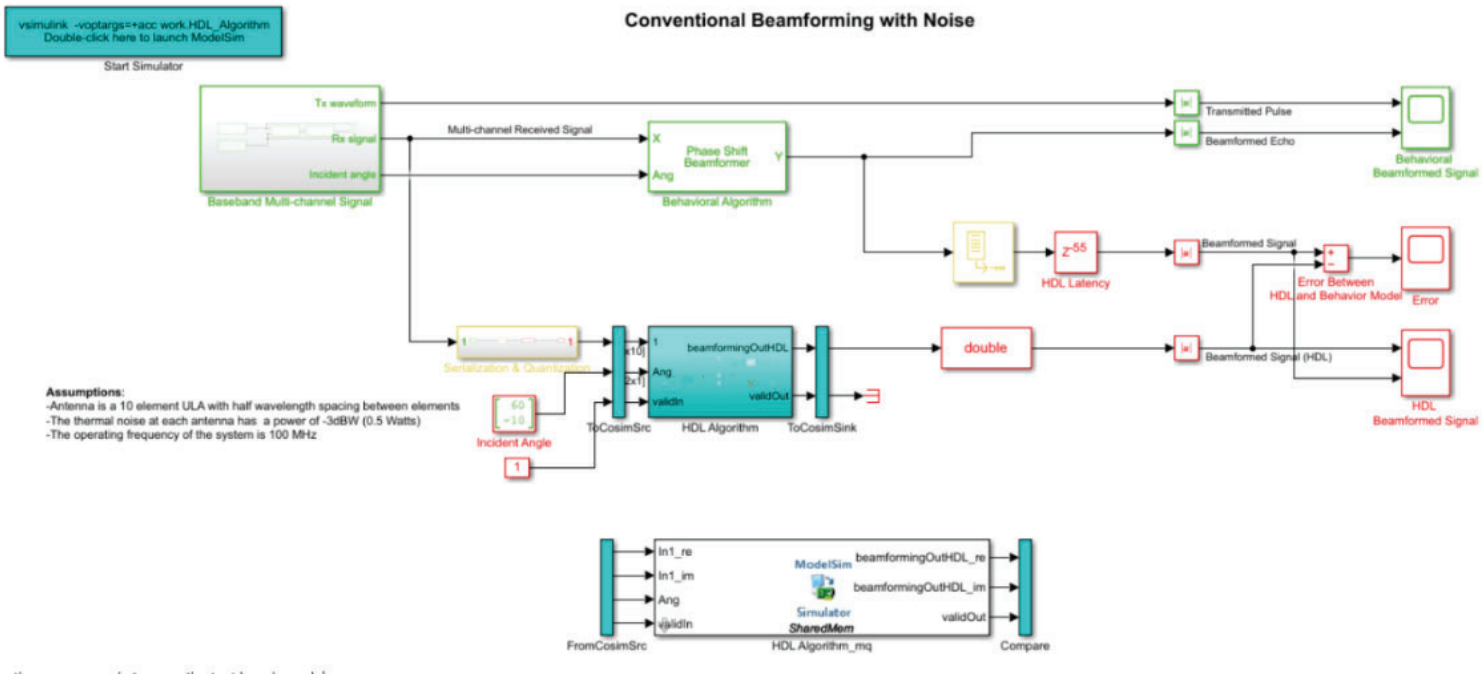


Figura 25: Nuevo modelo Simulink tras la generación del código HDL

Se comprueba si el modelo es compatible con la generación de código verificando los ajustes de configuración del modelo, la configuración de puertos y subsistemas, los bloques y su configuración, el soporte nativo de coma flotante y las directrices estándar del sector, como se observa en la Figura 26.

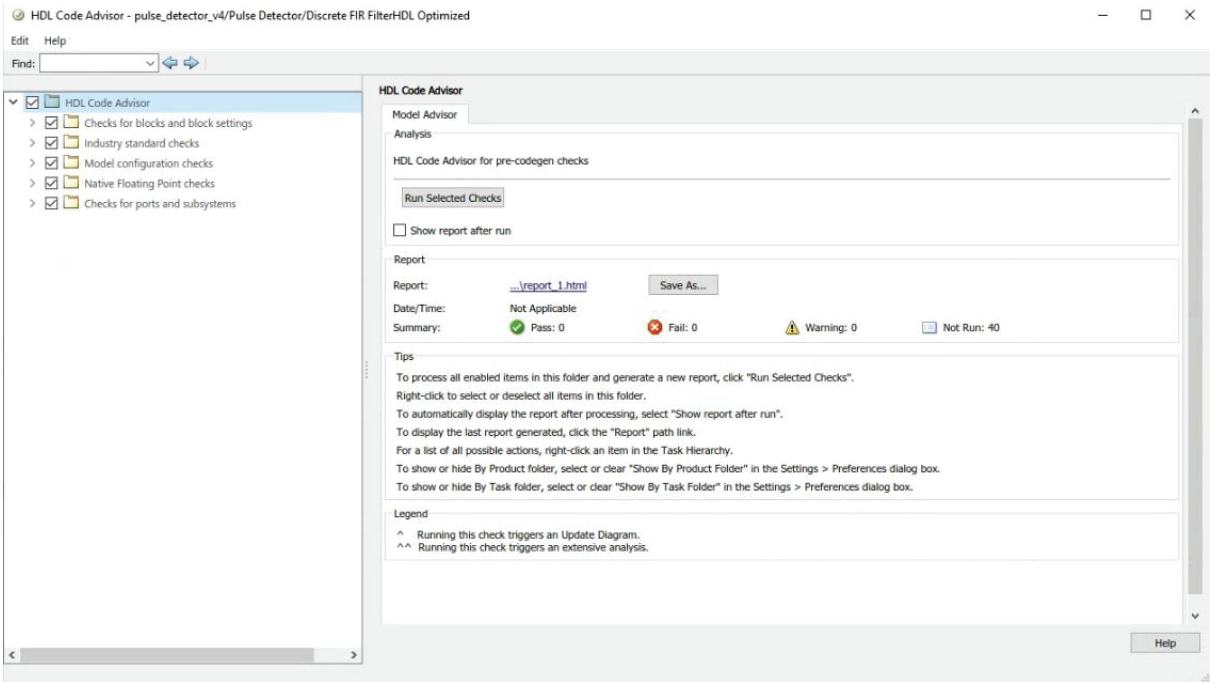


Figura 26: HDL Code Advisor

Una vez se verifica que el diseño es compatible con la generación de código HDL se pasa a generar código HDL, verificar el código y desplegarlo en la plataforma de destino (Vivado). Para ello, se abre en Simulink el asesor de flujo de trabajo HDL *Workflow Advisor.* El primer paso consiste en establecer varios ajustes de destino para la generación de código HDL como el dispositivo de destino y la herramienta de síntesis para la generación de código HDL, el diseño de referencia de destino, la interfaz de destino y la frecuencia de destino, que en este caso será de 50 MHz (Figura 27).

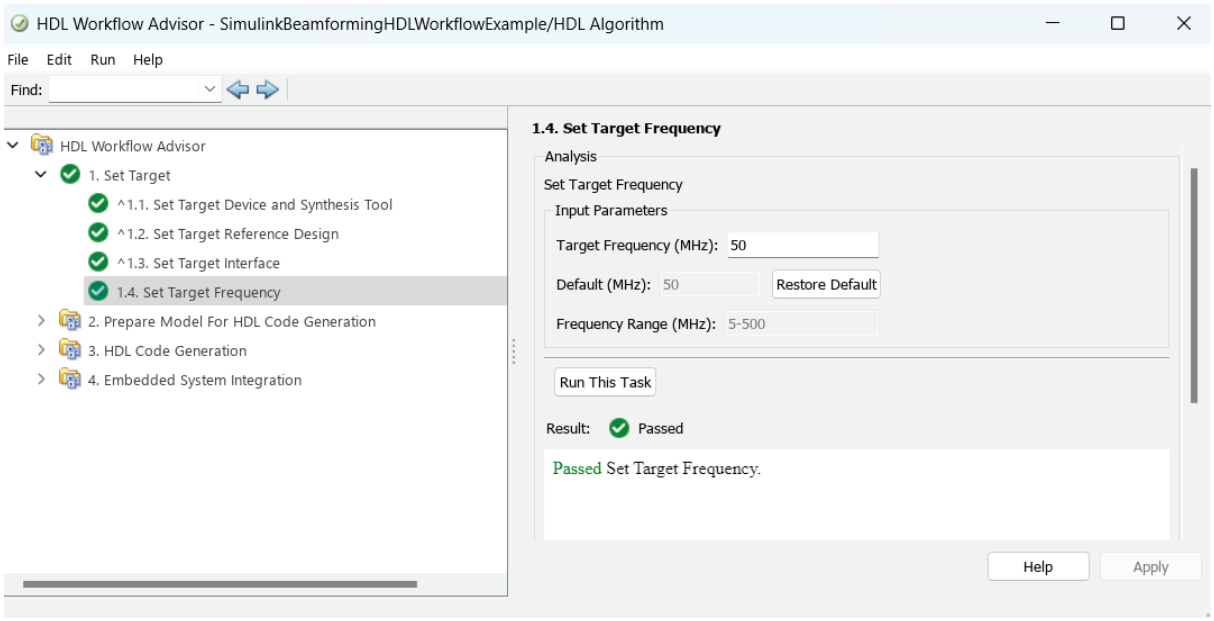


Figura 27: Establecimiento de ajustes de destino

Verificado el primer paso, se procede a comprobar y preparar el modelo para la generación de código HDL. Esto incluye la revisión de los ajustes básicos del modelo, como se ve en la Figura 28.

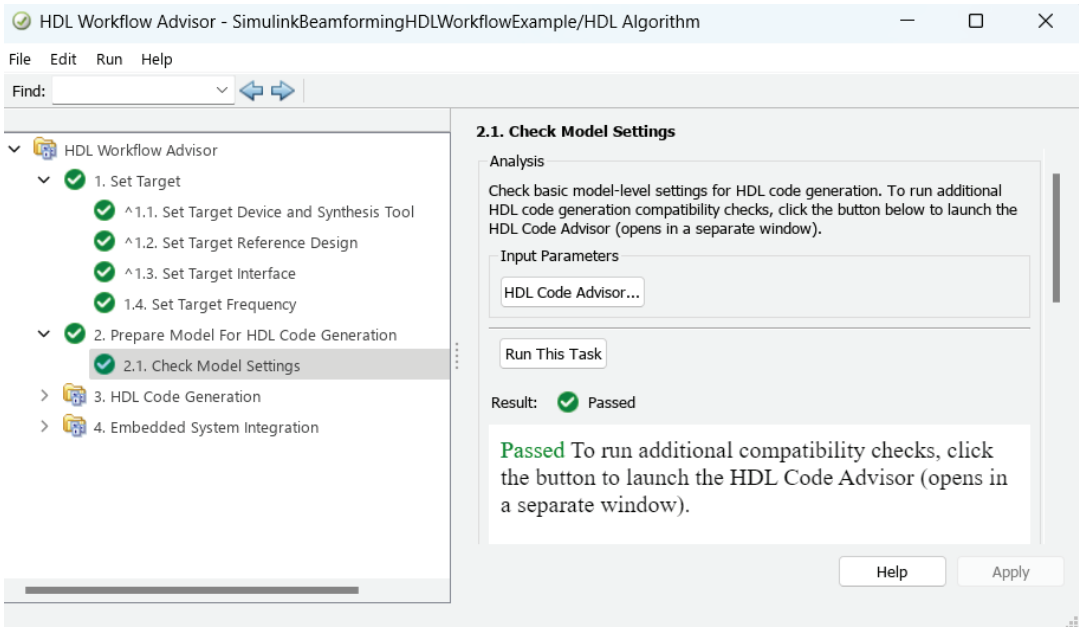


Figura 28: Comprobación los ajustes básicos

Realizada la comprobación de los ajustes básicos, se genera el código VHDL/Verilog a partir del subsistema seleccionado. Esto incluye la creación del modelo de simulación, el bloque de co-simulación y los archivos de trazabilidad. Además, se genera el *testbench*, el código *Register Transfer Level* (RTL) y el núcleo IP. Esto se muestra en la Figura 29.

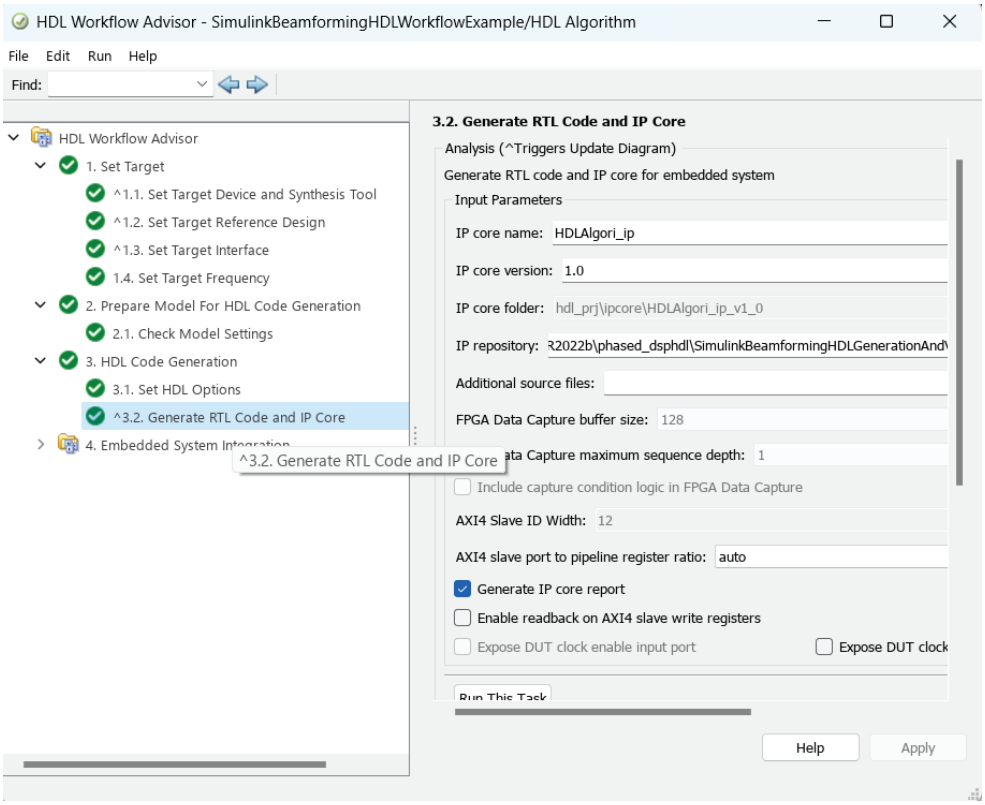


Figura 29: Generación de código HDL

Por último, se pasa a la integración del núcleo IP HDL con el procesador integrado en FPGA (Figura 30).

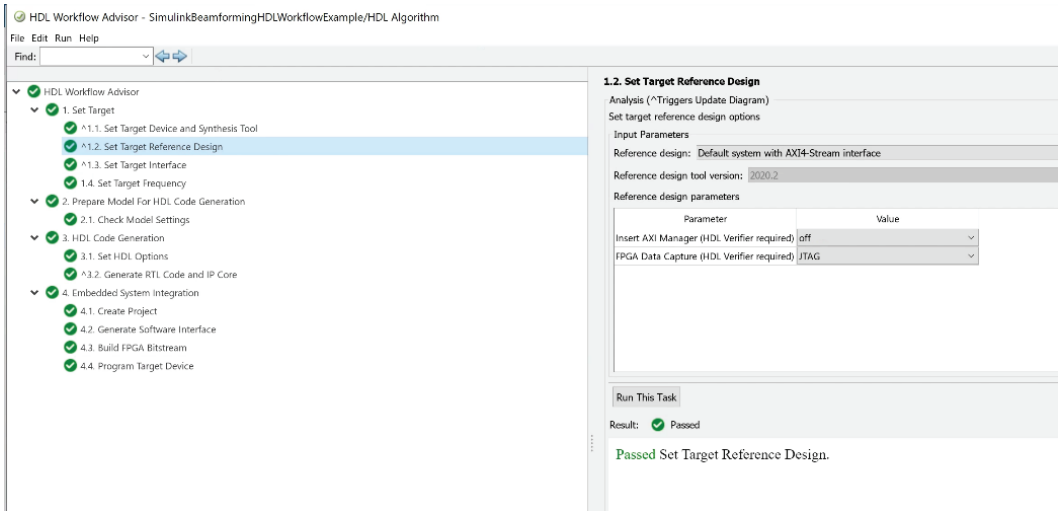


Figura 30: Integración del sistema

Se ejecuta el modelo de prueba de Simulink para visualizar los resultados de la simulación. Dentro de este modelo de prueba se encuentran bloques de *Time Scope* que permiten contrastar la salida de la cosimulación efectuada con ModelSim y la del subsistema HDL en Simulink como se muestra en Figura 31.

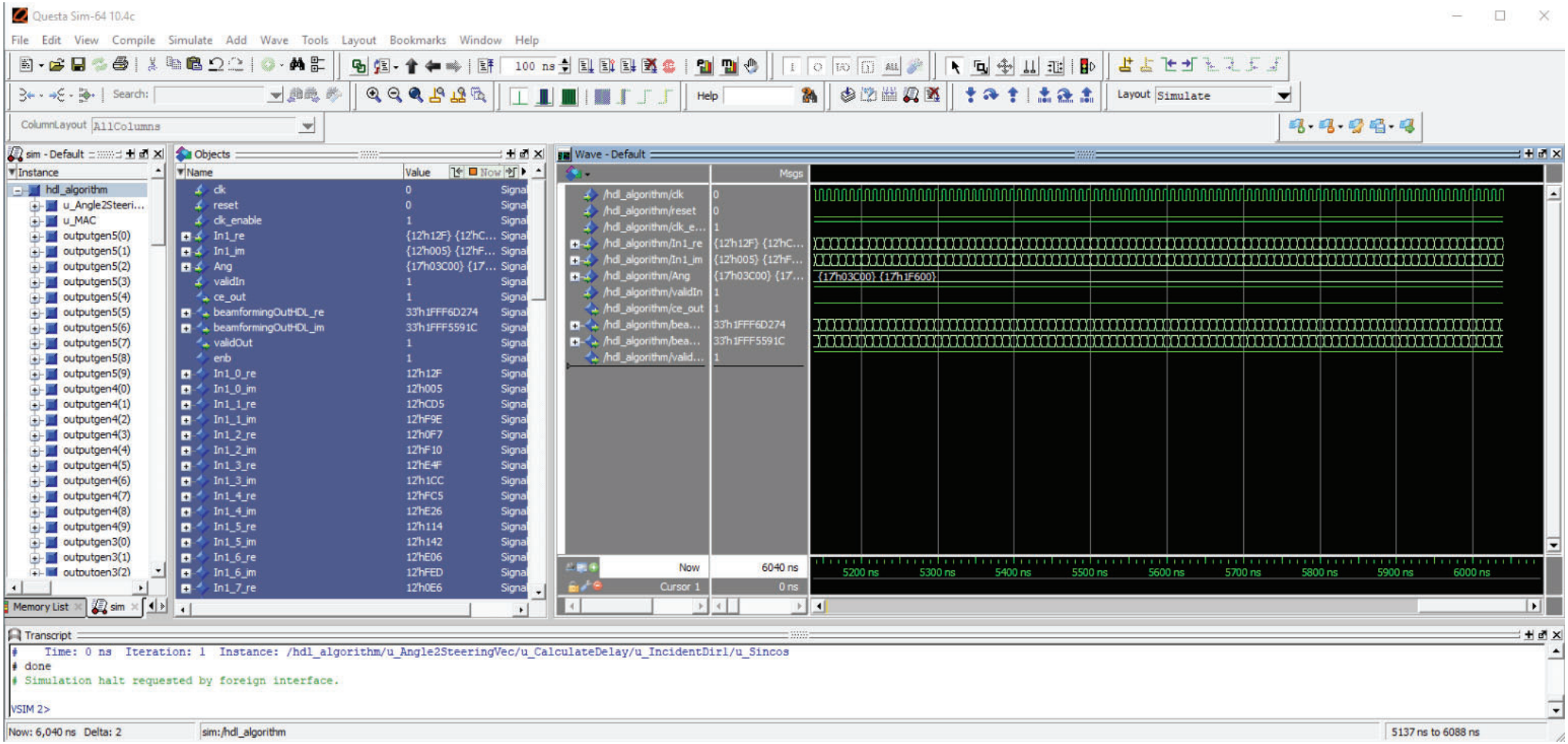


Figura 31: Comparación de resultados de cosimulación en los ámbitos Questa Sim y Simulink

La Figura 31 representa ejemplos de los resultados en los scopes de Questa Sim y Simulink. Muestra tanto la cosimulación como el modelo HDL produciendo una versión retardada de 79 ms de la señal original producida por el modelo descrito en código Matlab, como era de esperar, sin diferencia entre las dos formas de onda. El retardo se debe al retardo original añadido en el subsistema Algoritmo y un retardo adicional debido al equilibrado del retardo añadido por la generación del código HDL. Esto también se pude observar en la Figura 32, que muestra los resultados de la cosimulación comparando la salida del subsistema HDL en Simulink con la salida de la cosimulación realizada con ModelSim. Esto ayuda a comparar visualmente las formas de onda y los resultados del modelo HDL con la salida de la cosimulación, asegurando que se alinean correctamente y que muestran el comportamiento esperado.

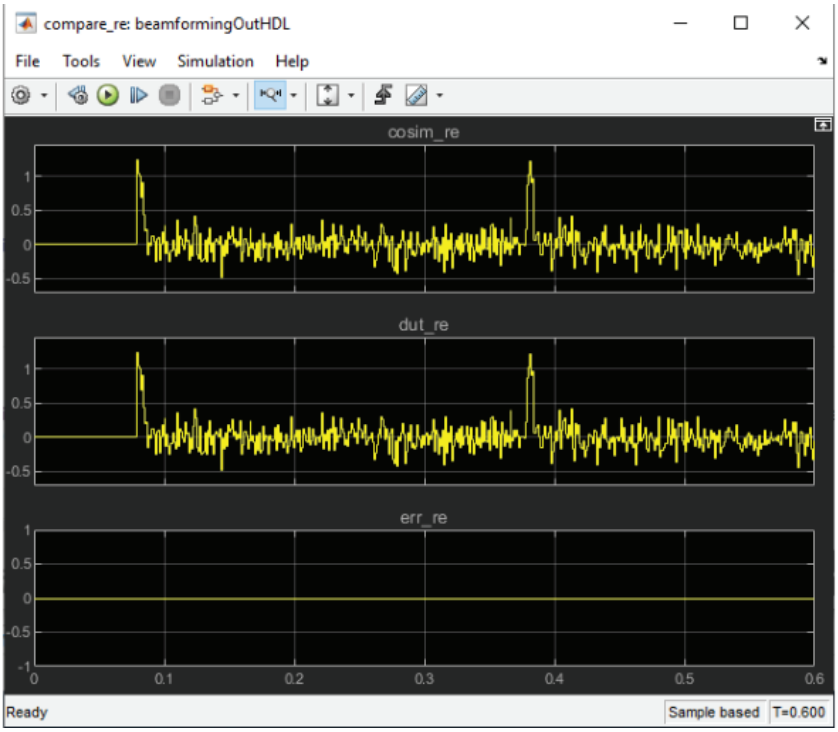


Figura 32: Comparación de la salida de los subsistemas

Una vez se ha terminado con todo este proceso se abre el diseño de bloques implementado en Vivado. El diseño de bloques generado es el que se muestra en la Figura 33.

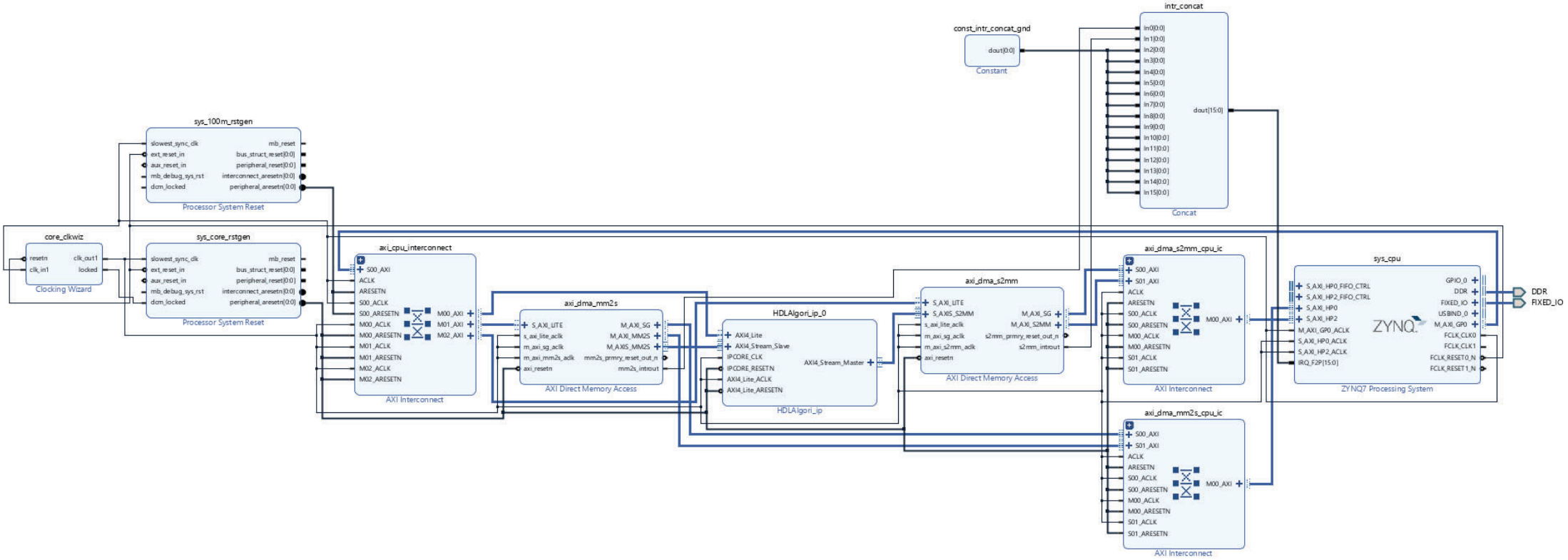


Figura 33: Diseño de bloque en Vivado

En este diseño presenta los siguientes bloques:

- Figura 34(a): Es el bloque *Clocking Wizard* utilizado para generar las señales de reloj con los requisitos de sincronización del diseño.
- Figura 34(b): Es el bloque *Processor System Reset* utilizado para las funciones de *reset*.
- Figura 34(c): Es el bloque *AXI Direct Memory Access* utilizado para transferir datos entre los dispositivos periféricos y la memoria principal del sistema.
- Figura 34(d): Es el bloque *HDLAlgori_ip*, es decir, el bloque IP generado del algoritmo Matlab. Utiliza el protocolo AXI para la comunicación con el resto de los bloques. Como entrada tenemos AXI4_Lite y AXI4_Stream_Slave que son protocolos de comunicación para la transferencia de datos entre los distintos bloques del diseño. AXI4_Lite se utiliza para el control de periféricos siendo ideal para operaciones de lectura y escritura de registros de configuración. AXI4_Stream_Slave se utiliza para la transferencia de datos. IPCORE_CLK y IPCORE_RESETM son las señales de reloj y reset de este bloque. La señal AXI4_Lite_ACLK representa la señal de reloj del protocolo AXI4 Lite y la señal AXI4_Lite_ARESETN es la señal de reinicio asincrónico del protocolo AXI4 Lite. En la salida está la AXI4_Stream_Master, que es una interfaz de salida que sigue el protocolo AXI y es compatible con el resto de los bloques. Se utiliza AXI 4 por su estandarización, su alto rendimiento y que permite la compatibilidad del bloque generado por Matlab con el resto de los bloques.
- Figura 34(e): Es el bloque *AXI Interconnect* que actúa como switch para facilitar la comunicación entre múltiples maestros y esclavos dentro del sistema.
- Figura 34(f): Es el bloque *Concat*, operación de concatenación del diseño. Este bloque combina las señales para formar un único flujo continuo de datos.

- Figura 34(g): Es el bloque *ZYNQ7 Processing System* que es la parte del SoC Zynq-7000 que incluye el procesador ARM Cortex-A9 y los componentes periféricos necesarios para soportarlo.

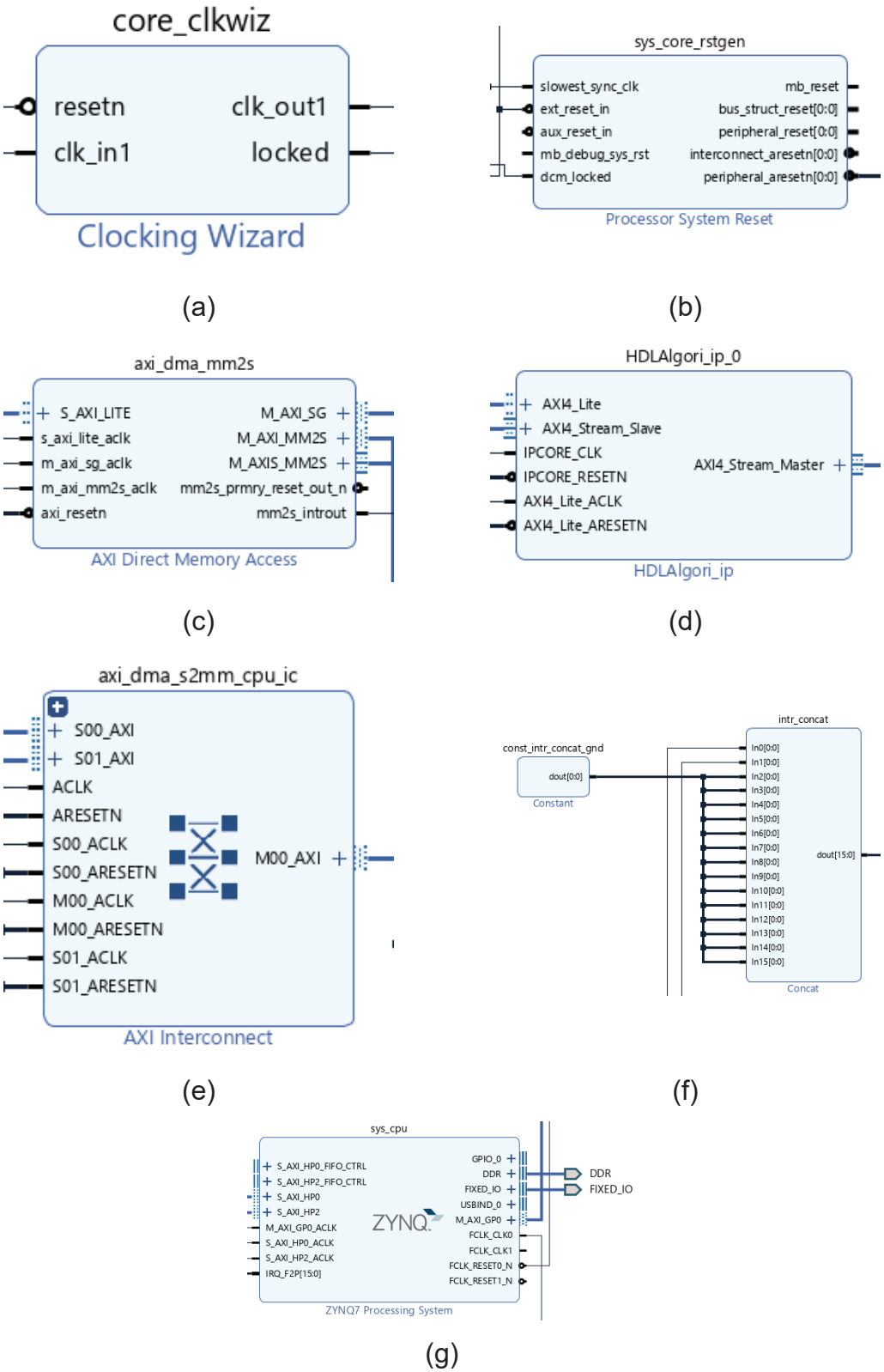


Figura 34: Bloques que componen el diseño final en Vivado: (a) Clocking Wizard, (b) Processor System Reset, (c) AXI Direct Memory Access, (d) HDLAlgori_ip, (e) AXI Interconnect, (f) Concat, (g) ZYNQ7 Processing System

Por último, se realiza la síntesis del diseño creado y se muestran los resultados. El diseño sintetizado es el que se muestra en la Figura 35.

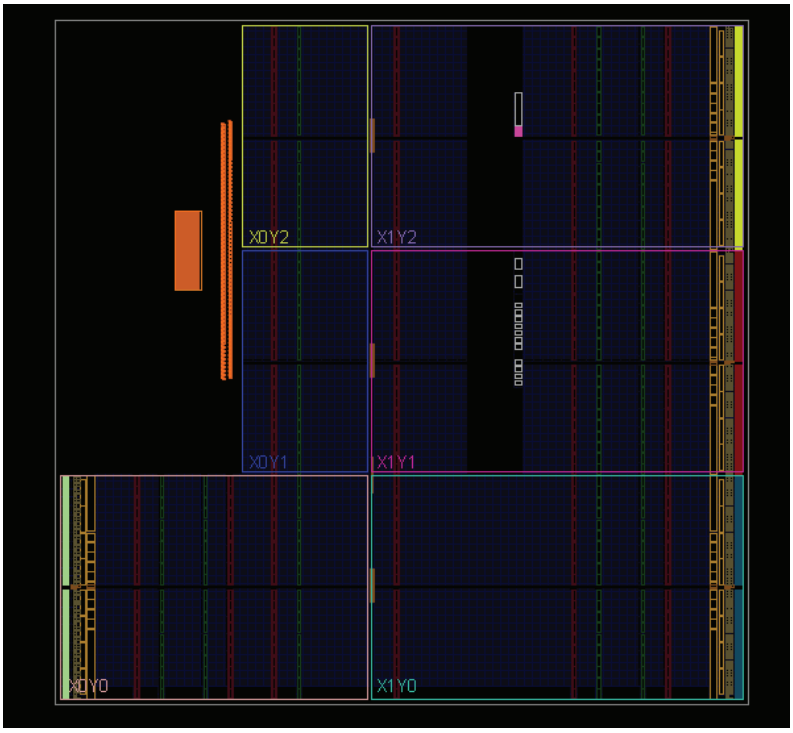


Figura 35: Diseño sintetizado (Layout)

Las ejecuciones de diseño dentro de Vivado muestran el consumo de área expresado en *LookUp Table* (LUT), *Flip-Flops* (FF), *Block RAM* (BRAM), *UltraRAM* (URAM) y DSP (Figura 36).

LUT	FF	BRAM	URAM	DSP
13471	19303	9.0	0	105

Figura 36: Consumo de área

El LUT es un componente fundamental en una FPGA y tiene la capacidad de realizar cualquier operación lógica de N variables booleanas. Este componente se asemeja a una tabla de verdad, donde diversas combinaciones de las entradas ejecutan distintas funciones para generar valores de salida.

FF es un componente de circuito secuencial que incorpora un registro de 1 bit y cuenta con la capacidad de restablecimiento.

BRAM son los bloques de memoria *Random Access Memory* (RAM) disponibles en las FPGA, que constan de varias celdas de memoria para el almacenamiento de los datos.

URAM es una memoria síncrona de dos puertos y un único reloj, en la que se crea una memoria flexible y de alta densidad.

DSP es un procesador que ha sido optimizado para el procesamiento de las señales.

A modo de resumen se puede ver en la Figura 35 la utilización estimada post-síntesis de estos recursos y de otros más como *Look-Up Table Random Access Memory* (LUTRAM), *Global Clock Simple Buffers* (BUFG) *o Mixed-Mode Clock Manager* (MMCM).

LUTRAM es una implementación de LUTs como recurso de RAM síncrono. Se puede utilizar para resolver operaciones complejas con eficiencia energética, claridad y versatilidad mínimas. También, contribuye a simplificar el diseño del sistema. Se puede aplicar en situaciones que demandan diseños complejos para garantizar tiempos de respuesta eficientes y confiables. Además, ofrece características significativas de propagación de señal, convirtiéndose en una elección destacada para aplicaciones que necesiten sincronización precisa o integración de la señal.

BUFG es un buffer de alto *fanout* que conecta las señales a los recursos de enrutamiento global con el fin de lograr una distribución equitativa de la señal. Comúnmente, se emplean en redes de reloj y otras redes con una variabilidad significativa, como aquellas que involucran *set*/*reset* y habilitación de reloj.

MMCM se emplea para producir varios pulsos de reloj con relaciones específicas de fase y frecuencia en relación con un reloj de entrada dado. Desempeñan un papel de generadores de frecuencia para abarcar una amplia variedad de frecuencias, al mismo tiempo que funcionan como dispositivos para estabilizar oscilaciones en relojes externos o internos y para sincronizar relojes.

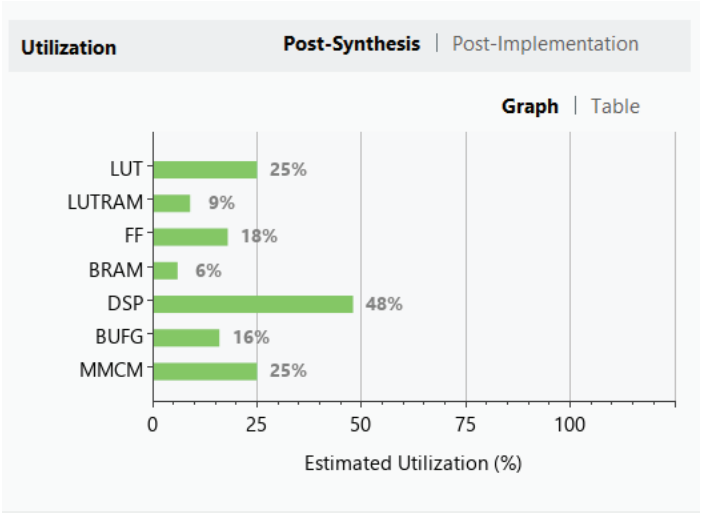


Figura 37: Estimación de utilización

Una vez realizada la síntesis del diseño de bloques se pasa a realizar su implementación. En la Figura 38 se observa el resumen de los tiempos de diseño donde, en la parte de configuración, el margen de tiempo negativo más desfavorable es 3,621 ns, este valor representa el caso más desfavorable de retardo entre la llegada de una señal a su destino y el tiempo requerido para cumplir con la restricción de temporalización en ese camino. Mientras, el número total de terminaciones es 56.858, que representa el número total de terminaciones en el diseño que están sujetos a restricciones de temporización. En la segunda columna *hold* tenemos el peor valor, que es 0,011 ns, que representa cuánto tiempo, sin infringir la restricción de temporización de retención, se puede capturar el dato en el receptor antes de que ocurra el cambio en la señal de retención. Lo mismo pasa en la columna de ancho de pulso donde la anchura de pulso más desfavorable es 3,000 ns e indica cuánto se está vulnerando la restricción de temporización de la duración del pulso en el peor caso del diseño.

En el contexto de la temporización, la columna *hold* se refiere al margen de tiempo adicional o a la capacidad de respuesta que se reserva en un plan o cronograma para acomodar posibles retardos, imprevistos o variaciones en la duración de las tareas. Esencialmente, es un término utilizado para describir la flexibilidad o espacio extra incorporado en un programa para garantizar que se cumplan los plazos incluso si surgen obstáculos inesperados.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	3,621 ns	Worst Hold Slack (WHS):	0,011 ns	Worst Pulse Width Slack (WPWS):	3,000 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	56858	Total Number of Endpoints:	56608	Total Number of Endpoints:	20880

All user specified timing constraints are met.

Figura 38: Resumen de los tiempos de diseño

En la Figura 39 se expone un análisis de potencia a partir del diseño de red implementado. La potencia total en el chip es de 1.906 W. El gráfico "*On-Chip Power*" muestra la distribución del consumo de energía entre los distintos componentes del diseño, lo que ayuda a identificar qué áreas del diseño consumen mayor energía.

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**1.906 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**47,0°C**
Thermal Margin:	38,0°C (3,2 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

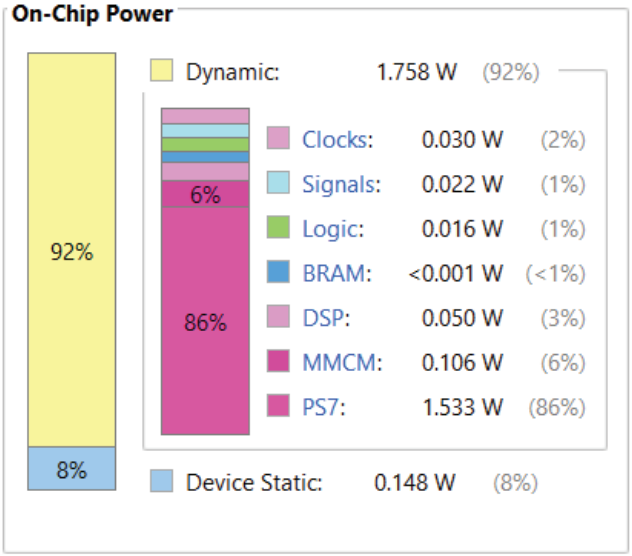


Figura 39: Resumen de Potencia

En la Figura 40 se expone detalladamente información sobre cómo se utilizan los recursos específicos de la FPGA en el diseño. En la primera columna, se encuentran los distintos recursos de la FPGA que se emplean en el diseño. En la segunda, la cantidad de recursos de cada tipo , y en la tercera, la cantidad total de recursos de cada tipo que están disponibles en la FPGA para ser utilizados en este diseño. La última columna representa el porcentaje de utilización de cada recurso en relación con la cantidad total disponible.

Resource	Utilization	Available	Utilization %
LUT	12485	53200	23.47
LUTRAM	1142	17400	6.56
FF	19179	106400	18.03
BRAM	9	140	6.43
DSP	105	220	47.73
MMCM	1	4	25.00

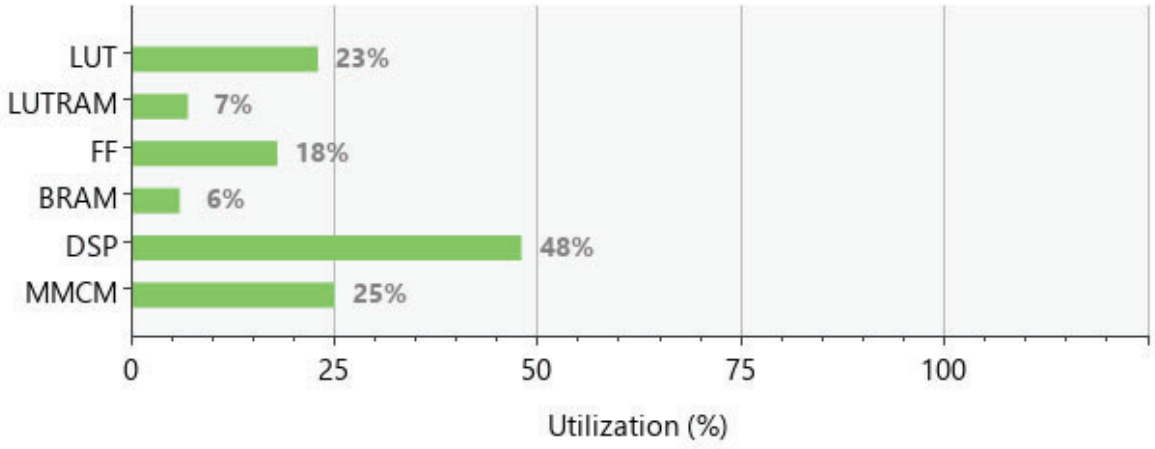


Figura 40: Área utilizada.

5.3. Conclusiones

En este apartado se genera el código HDL correspondiente para el diseño presentado, se crea el *testbench* para su comprobación, se genera el bloque IP necesario para su implementación en Vivado y se realiza la síntesis. Se muestra el diseño de *Layout* resultante de esta síntesis, así como los recursos que se utiliza para su implementación como LUTs, FFs, BRAMs o DSPs. Un 25% de los recursos de la FPGA serán destinados para LUT, un 9% para LUTRAM, un 18% para FF, un 6% para BRAM, un 48% para DSP, un 16% para BUFG y un 25% para MMCM.

Capítulo 6. Conclusiones y líneas futuras

6.1. Conclusiones

Este TFM consta de dos partes, donde se ha trabajado simultáneamente con las herramientas de Matlab, Simulink y Vivado.

En la primera parte, se describe el flujo de trabajo necesario para desarrollar un *beamformer* en Simulink destinado a la implementación en FPGA, destacando la importancia de comparar los resultados del modelo de implementación con el modelo descrito en código Matlab, para así garantizar su corrección. Se detallan las distintas herramientas dentro de Matlab y Simulink que se emplean para la creación de este diseño. Se crea el diseño de un *beamformer* adecuado para ser implementado en una FPGA utilizando las herramientas necesarias. Se muestra la implementación en Simulink del algoritmo de beamforming de desplazamiento de fase que maximiza la potencia de la señal recibida en una dirección específica. Se incluyen los resultados de este diseño y se comparan los resultados del modelo de rendimiento frente a los del modelo descrito en Matlab. Se comprueba que el subsistema del algoritmo HDL produce el mismo resultado que el modelo descrito en Matlab. Esta comparación es un paso importante antes de generar código HDL para garantizar la precisión de la implementación en hardware.

En la segunda parte se genera y verifica el código HDL a partir del modelo diseñado. Se destaca la importancia de comparar los resultados de la implementación del modelo con el modelo descrito en Matlab para garantizar el correcto funcionamiento del código generado. Se hace uso de un banco de pruebas para la verificación de este código. Se describe el proceso de cosimulación con ModelSim para la verificación a través de un *testbench*. Además, se verifica y se crea un diseño de bloques en Vivado sobre el que se lleva a cabo la síntesis e implementación, comprobando los recursos de la FPGA que se emplean.

Se concluye que este diseño de *beamformer* digital se puede implementar sobre una FPGA, debido a que ha superado las distintas pruebas. Un 25% de los recursos de la FPGA serán destinados para LUT, un 9% para LUTRAM, un 18% para FF, un 6% para BRAM, un 48% para DSP, un 16% para BUFG y un 25% para MMCM.

6.2. Líneas futuras

Se plantean las siguientes líneas futuras a partir de los resultados y conclusiones alcanzadas en este TFM:

- Implementación en Vivado, generación del *Bitstream* y volcado sobre la FPGA.
- Investigar técnicas para optimizar el rendimiento del diseño y reducir recursos utilizados en el hardware.
- Incluir algoritmos más complejos y avanzados que requieran más adaptaciones en la generación del código HDL.
- Explorar el uso de técnicas de aprendizaje automático en el proceso de generación de código HDL para optimizar tanto el diseño como su eficiencia [43], [44].
- Utilizar este diseño en una arquitectura híbrida e investigar otras formas de implementar el *beamforming* digital como con la arquitectura Flynn [45], [46]. Estas arquitecturas se fundamentan en la utilización de Moduladores Delta-Sigma de Paso Banda de Tiempo Continuo (Continuous-Time Delta-Sigma Modulator, CTBPDSM) combinados con *Bit Stream Processing* o BSP. Los CTBPDSMs digitalizan señales analógicas de banda estrecha directamente desde la FI o desde la RF sin requerir *downconverters*. Gracias a la integración de sobremuestreo y modelado de ruido propio de esta modulación, se obtienen moduladores con una alta relación SNR utilizando cuantificadores de baja resolución. Las ventajas de esta arquitectura son múltiples:
    - Reducción del consumo de energía al eliminar los *downconverters* analógicos.
    - Mayor eficiencia en las conversiones al evitar señales no deseadas provocadas por la conversión, lo que implica un menor consumo energético.
    - Posibilidad de alcanzar frecuencias de muestreo del orden de los gigahercios y anchos de banda de decenas de megahercios.

Referencias

[1] Z. Qu, G. Zhang, H. Cao, and J. Xie, “LEO Satellite Constellation for Internet of Things,” *IEEE Access*, vol. 5, pp. 18391–18401, Aug. 2017, doi: 10.1109/ACCESS.2017.2735988.

[2] I. Leyva-Mayorga *et al.*, “LEO Small-Satellite Constellations for 5G and Beyond-5G Communications,” *IEEE Access*, vol. 8, pp. 184955–184964, 2020, doi: 10.1109/ACCESS.2020.3029620.

[3] K. M. Peterson, “Satellite Communications,” in *Encyclopedia of Physical Science and Technology*, Elsevier, 2003, pp. 413–438. doi: 10.1016/B0-12-227410-5/00673-6.

[4] R. Perez, “Introduction to Satellite Systems and Personal Wireless Communications,” 1998, pp. 1–30. doi: 10.1016/S1874-6101(99)80014-3.

[5] G. Curzi, D. Modenini, and P. Tortora, “Large Constellations of Small Satellites: A Survey of Near Future Challenges and Missions,” *Aerospace*, vol. 7, no. 9, p. 133, Sep. 2020, doi: 10.3390/aerospace7090133.

[6] Z. Qu, G. Zhang, H. Cao, and J. Xie, “LEO Satellite Constellation for Internet of Things,” *IEEE Access*, vol. 5, pp. 18391–18401, Aug. 2017, doi: 10.1109/ACCESS.2017.2735988.

[7] I. Leyva-Mayorga *et al.*, “LEO Small-Satellite Constellations for 5G and Beyond-5G Communications,” *IEEE Access*, vol. 8, pp. 184955–184964, 2020, doi: 10.1109/ACCESS.2020.3029620.

[8] “ESA - Low Earth orbit.” Accessed: Mar. 02, 2022. [Online]. Available: https://www.esa.int/ESA_Multimedia/Images/2020/03/Low_Earth_orbit

[9] J. Mendez, W. Walt er Lupia, E. Pérez, and C. Alan López Mendoza, “Capítulo 2 Antenas 2.1 Antecedentes de electromagnetismo y álgebra vectorial para antenas 2.1.1 Ec…,” 2013.

[10] J. M. Huidobro, “Antenas de telecomunicaciones,” 2013.

[11] C. A. Balanis, *Antenna theory: analysis and design*. John wiley & sons, 2016.

[12] “Diseño y análisis de una patch (parche) antenna_Sistemas Avanzados de Comunicaciones_Máster META_ULPGC.” Accessed: Mar. 01, 2022. [Online]. Available: https://aep22.ulpgc.es/pluginfile.php/767614/mod_resource/content/9/practica1-Diseño de una patch antena.pdf

[13] J. Del, P. Despacho, S. Lalchand, and K. Despacho, “Propagación de ondas de radio-Sistemas Avanzados de Comunicación - Máster META - ULPGC”.

[14] V. Jamnejad, J. Huang, and R. J. Cesarone, “A study of phased array antennas for NASA’s Deep Space Network,” 2001.

[15] G. He, X. Gao, L. Sun, and R. Zhang, “A Review of Multibeam Phased Array Antennas as LEO Satellite Constellation Ground Station,” *IEEE Access*, vol. 9, pp. 147142–147154, 2021, doi: 10.1109/ACCESS.2021.3124318.

[16] “Trabajo Fin de Máster”.

[17] B. Chandhoke, S. Product, and L. Manager, “Design innovations in 5G mmWave FEMs & phased arrays”.

[18] “Phased-Array Antenna Patterns (Part 5)—Beam Squint | Microwaves & RF.” Accessed: Apr. 29, 2024. [Online]. Available: https://www.mwrf.com/technologies/embedded/systems/article/21142402/analog-devices-phased-array-antenna-patterns-part-5beam-squint

[19] M. Q. Abdalrazak, A. H. Majeed, and R. A. Abd-Alhameed, “A Critical Examination of the Beam-Squinting Effect in Broadband Mobile Communication: Review Paper,” *Electronics (Basel)*, vol. 12, no. 2, p. 400, Jan. 2023, doi: 10.3390/electronics12020400.

[20] “Beam-Squinting Effect in Broadband Mobile Communication | Encyclopedia MDPI.” Accessed: Apr. 29, 2024. [Online]. Available: https://encyclopedia.pub/entry/40837

[21] M. Q. Abdalrazak, A. H. Majeed, and R. A. Abd-Alhameed, “A Critical Examination of the Beam-Squinting Effect in Broadband Mobile Communication: Review Paper,” *Electronics (Switzerland)*, vol. 12, no. 2, Jan. 2023, doi: 10.3390/ELECTRONICS12020400.

[22] M. A. Antoniades and G. V. Eleftheriades, “A CPS Leaky-Wave Antenna With Reduced Beam Squinting Using NRI-TL Metamaterials,” *IEEE Trans Antennas Propag*, vol. 56, no. 3, pp. 708–721, Mar. 2008, doi: 10.1109/TAP.2008.916965.

[23] S.-H. Park, B. Kim, D. K. Kim, L. Dai, K.-K. Wong, and C.-B. Chae, “Beam Squint in Ultra-Wideband mmWave Systems: RF Lens Array vs. Phase-Shifter-Based Array,” *IEEE Wirel Commun*, vol. 30, no. 4, pp. 82–89, Aug. 2023, doi: 10.1109/MWC.007.2100530.

[24] “Phased-array antenna beam squinting related to frequency dependency of delay circuits | IEEE Conference Publication | IEEE Xplore.” Accessed: Apr. 29, 2024. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/6101846?casa_token=2vAG6Q

mvia4AAAAA:4X6kroYpnRtnnkHiTdRG6-oUdtok8VWGFbBpq8Cb0e_-BRXPVEI_7u8-Ob2Sxzsybe9LqmwJOg

[25] C. Engineering, “Design and Analysis of an Analog Beamforming Transmit System for 5G Midband Frequencies Design and Analysis of an Analog Beamforming Transmit System for 5G Midband,” 2019.

[26] “Advantages of Analog Beamforming | disadvantages of Analog Beamforming.” Accessed: Mar. 29, 2022. [Online]. Available: https://www.rfwireless-world.com/Terminology/Advantages-and-Disadvantages-of-Analog-Beamforming.html

[27] F. Sohrabi and W. Yu, “Hybrid Digital and Analog Beamforming Design for Large-Scale Antenna Arrays,” *IEEE Journal on Selected Topics in Signal Processing*, vol. 10, no. 3, pp. 501–513, 2016, doi: 10.1109/JSTSP.2016.2520912.

[28] S. Dutta, C. N. Barati, D. Ramirez, A. Dhananjay, J. F. Buckwalter, and S. Rangan, “A Case for Digital Beamforming at mmWave,” *IEEE Trans Wirel Commun*, vol. 19, no. 2, pp. 756–770, Feb. 2020, doi: 10.1109/TWC.2019.2948329.

[29] S. Pulipati *et al.*, “A Direct-Conversion Digital Beamforming Array Receiver with 800 MHz Channel Bandwidth at 28 GHz using Xilinx RF SoC,” in *2019 IEEE International Conference on Microwaves, Antennas, Communications and Electronic Systems (COMCAS)*, IEEE, Nov. 2019, pp. 1–5. doi: 10.1109/COMCAS44984.2019.8958039.

[30] A. M. Dixit *et al.*, “A CMOS Digital Beamforming Receiver,” *Anal Biochem*, vol. 11, no. 1, pp. 1–5, 2018.

[31] H. Steyskal, “COMPUTER],” pp. 49–57.

[32] A. M. Dixit *et al.*, “A CMOS Digital Beamforming Receiver,” *Anal Biochem*, vol. 11, no. 1, pp. 1–5, 2018.

[33] “The Case for Hybrid Beamforming in 5G mmWave Prototypes - IEEE Spectrum.” Accessed: Mar. 29, 2022. [Online]. Available: https://spectrum.ieee.org/the-case-for-hybrid-beamforming-in-5g-mmwave-prototypes

[34] J. Chen, J. Tao, S. Luo, S. Li, C. Zhang, and W. Xiang, “A deep learning driven hybrid beamforming method for millimeter wave MIMO system,” *Digital Communications and Networks*, Jul. 2022, doi: 10.1016/j.dcan.2022.07.005.

[35] P. K. Bailleul, “A New Era in Elemental Digital Beamforming for Spaceborne Communications Phased Arrays,” *Proceedings of the IEEE*, vol. 104, no. 3, pp. 623–632, Mar. 2016, doi: 10.1109/JPROC.2015.2511661.

[36] “FPGA-Based Beamforming in Simulink: Algorithm Design - MATLAB & Simulink - MathWorks España.” Accessed: Nov. 10, 2022. [Online]. Available: https://es.mathworks.com/help/phased/ug/design-an-hdl-beamforming-algorithm-in-simulink.html

[37] J. Grythe, “Beamforming algorithms-beamformers.”

[38] M. Garrido, P. Kallstrom, M. Kumm, and O. Gustafsson, “CORDIC II: A New Improved CORDIC Algorithm,” *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 63, no. 2, pp. 186–190, Feb. 2016, doi: 10.1109/TCSII.2015.2483422.

[39] W. J. Peréz-Holguín and R. A. Limas-Sierra, “Pipeline Implementation of the Unified CORDIC Algorithm in FPGA,” *Revista Facultad de Ingeniería Universidad de Antioquia*, Apr. 2022, doi: 10.17533/udea.redin.20220474.

[40] Xilinx and Inc, “Zynq-7000 SoC First Generation Architecture,” 2012. [Online]. Available: www.xilinx.com

[41] F. Winterstein, S. Bayliss, and G. A. Constantinides, “High-level synthesis of dynamic data structures: A case study using Vivado HLS,” in *2013 International Conference on Field-Programmable Technology (FPT)*, IEEE, Dec. 2013, pp. 362–365. doi: 10.1109/FPT.2013.6718388.

[42] M. Wegrzyn, “Implementation of safety critical logic controller by means of FPGA,” *Annu Rev Control*, vol. 27, no. 1, pp. 55–61, Jan. 2003, doi: 10.1016/S1367-5788(03)00007-5.

[43] I. M. Elfadel, D. S. Boning, and X. Li, *Machine learning in VLSI computer-aided design*. Springer, 2019.

[44] S. N. Shahrouzi and D. G. Perera, “HDL Code Optimizations: Impact on Hardware Implementations and CAD Tools,” in *2019 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, IEEE, Aug. 2019, pp. 1–9. doi: 10.1109/PACRIM47961.2019.8985074.

[45] B. Zheng, L. Jie, J. Bell, Y. He, and M. P. Flynn, “A Two-Beam Eight-Element Direct Digital Beamforming RF Modulator in 40-nm CMOS,” *IEEE Trans Microw Theory Tech*, vol. 67, no. 7, pp. 2569–2579, Jul. 2019, doi: 10.1109/TMTT.2019.2916803.

[46] J. Jeong, N. Collins, and M. P. Flynn, “A 260 MHz IF Sampling Bit-Stream Processing Digital Beamformer With an Integrated Array of Continuous-Time Band-Pass Modulators,” *IEEE J Solid-State Circuits*, vol. 51, no. 5, pp. 1168–1176, May 2016, doi: 10.1109/JSSC.2015.2506645.

Presupuesto

En este apartado se realizará el cálculo del presupuesto para este trabajo, se tendrán en cuenta los gastos por recursos *software*, recursos *hardware*, recursos humanos, material fungible, gastos de tramitación y envío, impuestos y redacción de la memoria.

1.1. Recursos *software*

En esta sección se mostrarán los costes de recursos *software*. Se realizará una tabla con cada uno de los *softwares* utilizados para este TFM, el tipo de licencia de cada *software* y su coste (Tabla 2). En este caso, el coste por recursos *software* es de cero euros.

Tabla 2: Recursos software

Recursos software		
Softwares	**Tipo de licencia**	**Costes (€)**
Matlab R2022b	Universitaria	0,00
Vivado 2020.2	Universitaria	0,00
Microsoft Office 365	Universitaria	0,00
Mendeley	Gratuito	0,00
	Coste total:	0,00

1.2. Recursos *hardware*

En este apartado se tratarán los costes por recursos *hardware*. Se realizará una tabla con cada uno de los elementos *hardware* utilizados para este TFM, su coste por unidad y su coste de utilización en términos de amortización (Tabla 3). Para obtener el coste de utilización de los recursos hardware se ha usado un porcentaje de amortización de 6,7% que corresponde con el porcentaje de vida útil del recurso en el periodo de tiempo establecido para este TFM. El coste total de los recursos *hardware* será de ciento veintinueve euros y sesenta y siete céntimos.

Tabla 3: Recursos hardware

Recursos hardware				
Hardware	**Cantidad**	**Costes por unidad (€)**	**Amortización (%)**	**Costes de utilización (€)**
Ordenador de sobremesa	1	1.599	6,7	107,13
FPGA Arty z7-20	1	336,46	6,7	22,54
			Coste total:	129,67

1.3. Recursos humanos

Para este apartado se debe tener en cuenta el salario que gana un ingeniero técnico en Tecnologías de la Telecomunicación con un Máster Universitario. Según la tabla de retribuciones publicada por la Universidad de Las Palmas de Gran Canaria el salario por 20 horas semanales es de 831,43 € al mes.

Teniendo en cuenta que el TFM se realizó en cuatro meses a media jornada, el costo total por recursos humanos sería de tres mil ciento veinte euros (Tabla 4).

Tabla 4: Recursos humanos

Recursos humanos			
Hardware	**Coste (€) /horas**	**Horas**	**Coste total (€)**
Ingeniero técnico	10,4	300	3120,00

1.4. Otros gastos

En este apartado se tratan los gastos de tramitación y envío del documento, así como los gastos de redacción de memoria.

En este apartado se calculará el coste que supone la redacción de este TFM con la siguiente fórmula (4):

$$R=0{,}07*P*Cn \quad (4)$$

donde la letra R corresponde con el coste total de redacción, la letra P con el presupuesto total de ejecución del TFM y C_n es el coeficiente de ponderación en función del presupuesto. Para presupuestos menores de 30.050 € el coeficiente de ponderación será de 1,00.

Los gastos totales (Tabla 5) de este apartado serán de setecientos cuarenta y seis euros.

Tabla 5: Otros gastos

Otros gastos	
Gasto	**Coste total (€)**
Tramitación y envío del documento	50,00
Redacción de la memoria	696,00
Coste total:	746,00

1.5. Presupuesto total

En este apartado se hará la suma de los anteriores apartados para obtener el presupuesto total de este TFM. Además, se le aplicará el Impuesto General Indirecto Canario (IGIC) del 7%. El presupuesto total de este TFM ascenderá a cuatro mil doscientos setenta y cinco euros y treinta y cinco céntimos (Tabla 6).

Tabla 6: Presupuesto total

Presupuesto total	
Recursos	**Costes (€)**
Recursos software	0,00
Recursos hardware	129,67
Recursos humanos	3120,00
Otros gastos	746,00
Total antes de impuestos:	3995,67
IGIC (7%)	279,70
Presupuesto total:	4275,35

Firma:

GARCIA SUAREZ KAREN LYN - 46251559P

Firmado digitalmente por GARCIA SUAREZ KAREN LYN - 46251559P
Fecha: 2024.06.19 17:43:04 +01'00'