UAH

DISEÑO E IMPLEMENTACIÓN DE UN SOC PARA EL CONTROL DE UN LPS EXTENSO

Máster Universitario en Sistemas Electrónicos Avanzados: Sistemas Inteligentes

Departamento de Electrónica

Presentado por:

D. Francisco Pérez Fermoselle

Dirigido por:

Dr. D. Álvaro Hernández Alonso

Dr. D. Francisco Daniel Ruíz Pereda

Alcalá de Henares, a 23 de Enero de 2014

Agradecimientos

Me gustaría agradecer especialmente a mis padres por todo todo lo que han hecho por mí en mi vida, pero en particular por creer siempre en mí y estar siempre ahí. Han sido todo lo que se puede esperar de unos buenos padres y más.

A mis tutores Álvaro y Daniel por ayudarme durante todo el proceso de creación de este Trabajo Fin de Máster y por la cercanía que tienen con sus alumnos y en concreto conmigo.

A mis compañeros de la universidad y más en concreto a Elena, Rodrigo y Javi que se han convertido en unos amigos importantes y que han hecho más soportable este año y medio de Máster.

Y por último a todas esas personas que han estado en mi vida y me han hecho tanto bien.

Muchas gracias a todos.

Índice general

Índice de figuras

Parte I

Resumen

Capítulo 1

Resumen

1.1. Resumen

En el presente Trabajo Fin de Máster se propone el desarrollo de un sistema de control para un Sistema de Posicionamiento Local (LPS) configurable de forma remota; dicho sistema se ha realizado en la plataforma de desarrollo Genesys de Xilinx. Para conseguir el objetivo propuesto se realiza un System-on-Chip (SoC) sobre la FPGA, que además de utilizar otros recursos existentes de la plataforma de desarrollo, tendrá acceso a una serie de DACs externos que proporcionarán las señales analógicas a las balizas ultrasónicas del LPS. Además, para hacer el sistema mucho más versátil, se ha conectado un router a la plataforma de desarrollo que permite, mediante una conexión Wi-Fi, configurar una serie de parámetros de emisión. Dichos parámetros son:

- Tipo de modulación: BPSK (Binary Phase Shift Keying), APK (Amplitude Phase Keying), QPSK (Quadrature Phase Shift Keying), or QAM (Quadrature Amplitude Modulation).
- Secuencias utilizadas para codificar las emisiones ultrasónicas.
- Número de períodos de señal moduladora por código.
- Número de muestras por señal moduladora.
- Frecuencia de muestreo usada en la generación de la transmisión ultrasónica: sólo están permitidas las frecuencias 400kHz, 416.66kHz y 500kHz.
- Configuración del LPS como maestro o como esclavo.

Cabe destacar que la conexión entre el router Wi-Fi y la plataforma de desarrollo se realiza a través del protocolo Ethernet, siendo éste manejado por el chip de la plataforma de desarrollo que realiza el control de la capa MAC, y que la gestión de recepción/emisión se lleva a cabo por el MicroBlaze instanciado en la FPGA.

Palabras clave: LPS (Local Positioning System), SoC (System on Chip), FPGA (Field Programmable Gate Array), DAC (Digital-to-Analog Converter)

1.2. Abstract

In this Master Thesis the development of the control system for a Local Positioning System (LPS), remotely configurable, is proposed. The system has been based on the Genesys development platform. In order to achieve the proposal, a System-on-Chip has been deployed into the FPGA, which has access to some external DACs which provide the analogic signals to the ultrasonic beacons of the LPS. Furthermore, in order to make the system more flexible, a Wi-Fi router has been connected to the development platform to set some parameters of the emission.

These parameters are:

- Type of modulation: BPSK (Binary Phase Shift Keying), APK (amplitude Phase Keying), QPSK (Quadrature Phase Shift Keying), or QAM (Quadrature Amplitude Modulation).
- Idle temporal gap between successive emissions.
- Sequences used to encode the ultrasonic transmissions.
- Modulation carrier.
- Number of carrier periods per bit of the sequences.
- Number of samples per modulation carrier period.
- Sampling frequency used in the generation of the ultrasonic transmission: only 400kHz, 416.66kHz and 500kHz are allowed.
- Master or slave configuration
- IP address and connection port for the Ethernet link.

It is noteworthy that the connection between the Wi-Fi router and the development board is via Ethernet protocol. The Ethernet protocol is handled by the MAC Ethernet Controller chip, existing in the development board, and the reception/emission management is managed by the MicroBlaze processor.

Keywords: LPS (Local Positioning System), SoC (System on Chip), FPGA (Field Programmable Gate Array), DAC (Digital-to-Analog Converter)

Parte II

Memoria

Capítulo 2

Introducción

2.1. Contexto

Este proyecto se enmarca dentro del proyecto LORIS financiado por el Ministerio de Economía y Competitividad español que ha sido desarrollado dentro del grupo GEINTRA, *Grupo de Ingeniería Electrónica Aplicada a Espacios Inteligentes y Transporte*, y más concretamente en el subgrupo de ultrasonidos, perteneciente al departamento de electrónica de la Universidad Politécnica de Alcalá de Henares.

2.2. Objetivos y motivación

El objetivo principal de este Trabajo Fin de Máster es el diseño e implementación de un SoC (*System on Chip*) accesible de forma remota con el fin de controlar un LPS (*Local Positioning System*) que sea lo más versátil posible, de manera que cambiando una serie de parámetros mediante un controlador remoto se cambie la funcionalidad del sistema. Además se muestra como otro objetivo la posibilidad de conseguir una funcionalidad coordinada de diversos LPS en un mismo espacio físico para permitir una emisión de forma síncrona.

Se necesita un sistema de configuración lo más versátil posible debido a que en el grupo de investigación GEINTRA se realizan pruebas para mejorar el posicionamiento con diversas técnicas que afectan directamente a la forma de emisión de las balizas como son los códigos emitidos, el tipo de modulación, los tiempos muertos entre emisiones, etc ...

2.3. Estructura documento

La redacción del presente Trabajo Fin de Máster se ha dividido en siete partes atendiendo a la explicación asociada:

- **Capítulo 1:** se realiza un resumen del Trabajo Fin de Máster, tanto en español como en inglés.
- **Capítulo 2:** se presenta una pequeña introducción del trabajo atendiendo al contexto, los objetivos, motivación y la estructura del documento.
- **Capítulo 3:** se abordan los temas necesarios para la entendimiento del resto del documento, así como de sistemas parecidos ya implementados con anterioridad.

- **Capítulo 4:** se explica cómo se ha abordado la distribución de tareas entre los diversos grupos funcionales y su interconexión a nivel conceptual, para que el sistema se comporte adecuadamente.
- **Capítulo 5:** se detalla como se ha resuelto el problema en todas las etapas de desarrollo del sistema, con las simulaciones, diagramas de estado, esquemas y flujogramas necesarios.
- **Capítulo 6:** se analizan las pruebas realizadas sobre el sistema final real ya implementado, así como el porcentaje de ocupación de la FPGA.
- **Capítulo 7:** se muestran las conclusiones y los posibles trabajos futuros que se pueden realizar para mejorar el sistema.

Capítulo 3

Antecedentes

3.1. Sistemas SoC

El system on chip (SoC) es un circuito integrado en el cual se albergan todos los componentes que puede necesitar un sistema en un solo chip. La estructura básica de un SoC puede constar de los siguientes elementos:

- Un microcontrolador, un microprocesador o núcleo DSP.
- Módulos de memoria: ROM, RAM, E2PROM y/o FLASH.
- Componentes periféricos.
- Interfaces externos como USB, Ethernet, UART o SPI.
- Interfaces analógicos como ADCs o DACs.
- Reguladores de alimentación.

Los SoC se pueden desarrollar sobre plataformas hechas con las tecnologías ASIC (*Application-Specific Integrated Circuit*), ya sean full-custom o semi-custom, o con FPGAs (*Field Programmable Gate Array*). En los próximos puntos se muestran los elementos básicos y las distintas características de cada una de las tecnologías.

3.1.1. PLD

Los primeros dispositivos fueron los PLDs (*Programmable Logic Device*). Éstos son un conjunto de circuitos digitales integrados formados por puertas lógicas, cuyas conexiones pueden ser establecidas bien por el fabricante o por el usuario. Los fabricantes pueden realizar grandes tiradas de estos circuitos integrados lo que abarata sus costes de producción y los usuarios pueden utilizarlo para sus propios diseños. Los PLDs tienen las siguientes características:

- Consumos medios, aunque hay familias especializadas en bajo consumo.
- Velocidad intermedia.
- Alta fiabilidad.
- Tiempo de desarrollo bajo, sin dependencia de terceros.
- Metodología y equipamiento necesario sencillos.
- Aumentan la confidencialidad de las placas.

La figura 3.1 muestra la evolución y la escala de integración de los PLDs.

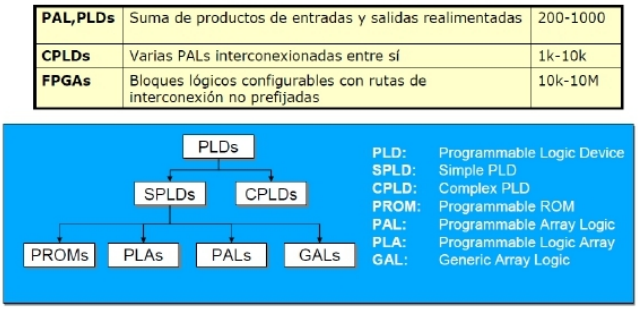


Figura 3.1: Evolución temporal y escala de integración.

En los siguientes puntos se realiza una explicación de los distintos dispositivos presentados en el diagrama de la figura 3.1. Debido a que los SPLDs son una tecnología muy antigua se pasará directamente a la explicación de los CPLDs.

3.1.1.1. CPLD

Un CPLD (*Complex Programmable Logic Device*) extiende el concepto de un SPLD a un mayor nivel de integración y permite implementar sistemas más eficaces. Los CPLDs utilizan menor espacio, mejoran la fiabilidad del diseño y reducen costos. Un CPLD se forma con múltiples bloques lógicos, cada uno similar a un PLD, y éstos se comunican entre sí utilizando una matriz programable de interconexiones. En la figura 3.2 se puede observar la estructura interna de un CPLD genérico.

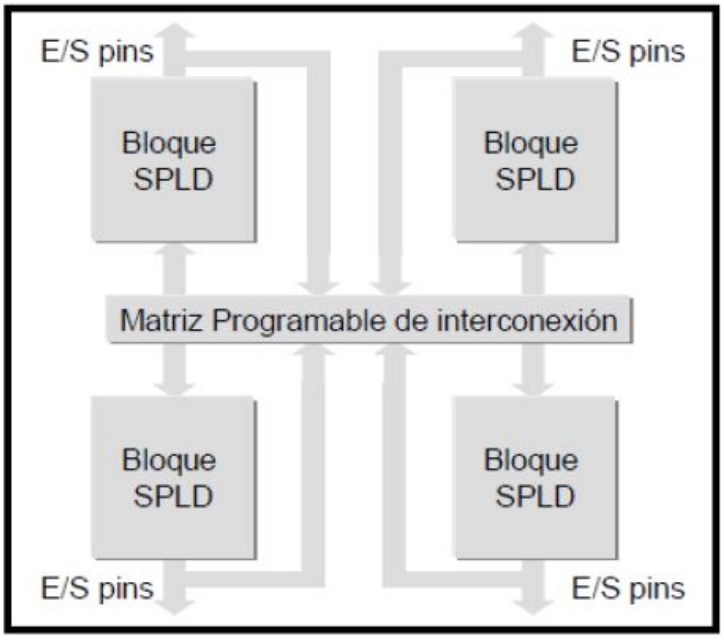


Figura 3.2: Esquema de un dispositivo CPLD[1].

[1]Imagen tomada de `http://www.scribd.com/doc/79177686/4-PLD`

Existen dos tipos de arquitectura de los CPLD:

Matriz de suma de productos. Se relacionan con un mayor número de interconexión de secciones formadas por puertas programables interconectados entre sí en un solo chip. Dichas secciones pueden ser otros PLD de baja densidad. En la Figura 3.3, se puede observar el diagrama de bloques de un CPLD con matriz de suma de productos.

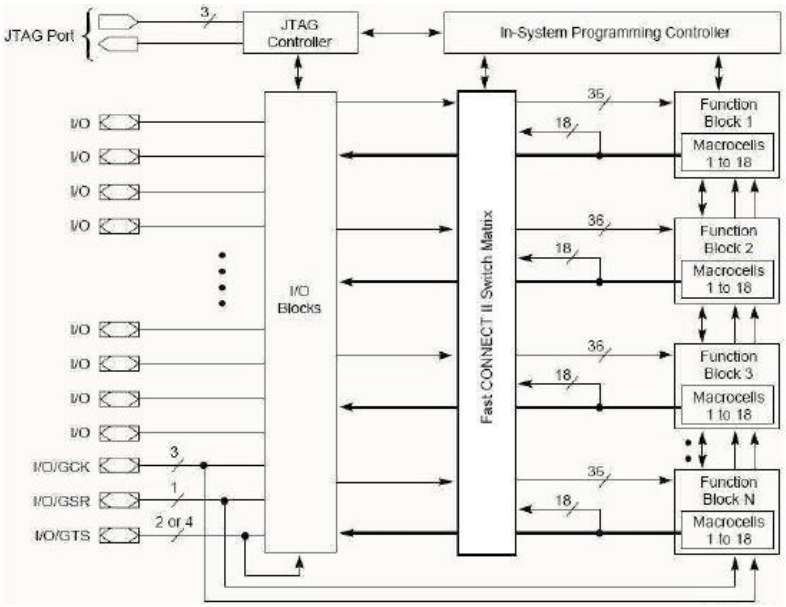


Figura 3.3: Esquema de un dispositivo CPLD con arquitectura Matriz suma de productos[2].

LUT (*Look-up table*). Esta arquitectura se basa en la implementación lógica de bus de interconexión de filas y columnas. Estas interconexiones también proveen conexión al LAB (*Large Array Block*). El LAB consiste en varios elementos lógicos que proporcionan una implementación eficiente de funciones lógicas definidas por el usuario. El Multitrack Interconnect proporciona una rápida conexión entre los LAB. En la Figura 3.4, se muestra la estructura interna de la arquitectura LUT.

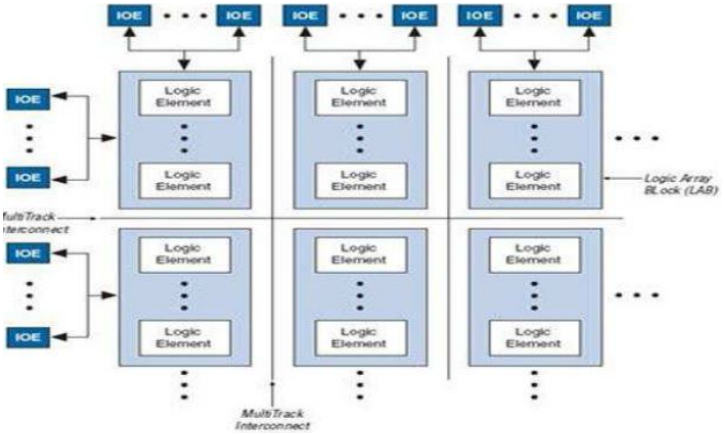
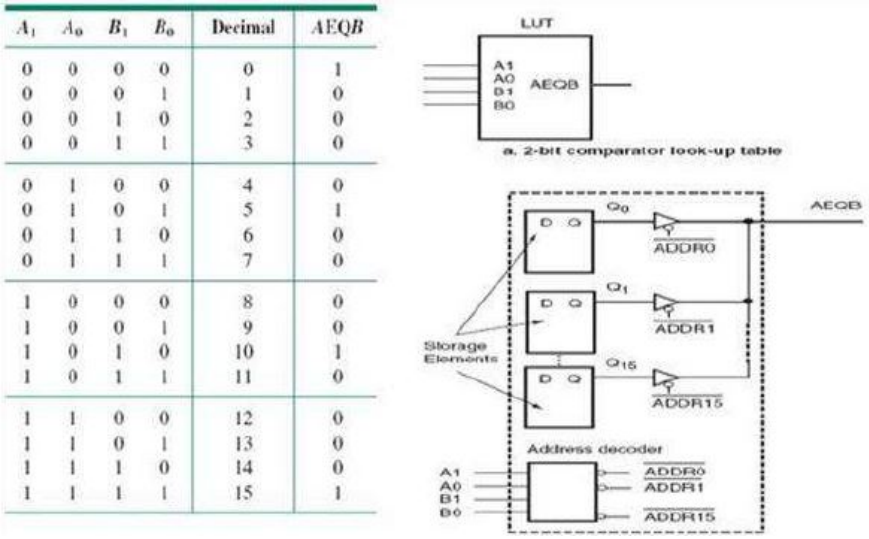


Figura 3.4: Esquema de un dispositivo CPLD con arquitectura LUT[3].

[2]Imagen tomada de `http://www.scribd.com/doc/79177686/4-PLD`

La LUT implementa una circuito de lógica combinacional para resolver la función especificada por el usuario, de manera que quedan definidas las salidas para cualquier combinación de valores de la entrada. En la figura 3.5 se muestra un ejemplo de funcionalidad de una LUT.



A_1	A_0	B_1	B_0	Decimal	$AEQB$
0	0	0	0	0	1
0	0	0	1	1	0
0	0	1	0	2	0
0	0	1	1	3	0
0	1	0	0	4	0
0	1	0	1	5	1
0	1	1	0	6	0
0	1	1	1	7	0
1	0	0	0	8	0
1	0	0	1	9	0
1	0	1	0	10	1
1	0	1	1	11	0
1	1	0	0	12	0
1	1	0	1	13	0
1	1	1	0	14	0
1	1	1	1	15	1

Figura 3.5: Ejemplo de funcionalidad de una LUT[4].

3.1.1.2. ASIC

Un circuito integrado de aplicación específica o ASIC (*Application-Specific Integrated Circuit*) al contrario que otros dispositivos, pueden contener funciones analógicas, digitales y combinaciones de ambas. Los chips ASIC son realizados con la funcionalidad completa en fábrica, el usuario, a partir de las especificaciones del fabricante, creará la funcionalidad deseada. Éstos dispositivos son de altas prestaciones pero tienen un costo inicial alto y son sólo rentables para largas tiradas de fabricación. Según la forma de diseño se distinguen:

Full-Custom: el diseño se realiza totalmente a medida y define la totalidad de las capas litográficas del dispositivo. Los beneficios de este método usualmente incluyen un área reducida (y consecuentemente costos por unidad menores), mejoras en el desempeño, la habilidad de integrar componentes analógicos y otros componentes pre-diseñados. Como contrapartida está su elevado coste y tiempo de desarrollo, costos fijos mayores, mayor complejidad del software CAD y la necesidad de habilidades mucho mayores por parte del equipo de diseño.

Semi-Custom: son sistemas construidos a partir de bloques funcionales existentes y la funcionalidad final se consigue mediante la interconexión entre ellos que no está definida a priori. Con esto se consigue abaratar los costes y acortar los tiempos de desarrollo. Existen diferentes tipos de sistemas semi-custom listados a continuación

- **Standard-Cells:** se utilizan células prediseñadas disponibles en librerías que dependen del fabricante y el diseño se realiza combinando dichas células. Las células pueden ser componentes como: inversores, puertas NAND, puertas NOR, puertas AOI y OAI, latches tipo D y flip-flops. Cada tipo de célula puede tener múltiples implementaciones que estén optimizadas para diferentes fan-outs,

[3]Imagen tomada de `http://www.scribd.com/doc/79177686/4-PLD`
[4]Imagen tomada de `http://www.scribd.com/doc/79177686/4-PLD`

por ejemplo, una puerta inversora puede estar construida a partir de transistores con tamaño estándar, doble o cuádruple. Para posibilitar la colocación automatizada de las células y el ruteado de las conexiones entre células, el layout de cada célula se diseña con una altura fija, de éste modo, las células son apiladas formando filas. La línea de alimentación y masa corren paralelas a los bordes superior e inferior de las filas, de tal modo que las células vecinas comparten las líneas de alimentación. Los pines de entrada y salida están localizados en los bordes superior e inferior de la célula. En la figura 3.6 se puede observar un esquema de cómo está formado interiormente un ASIC semi-custom Standard-Cells.

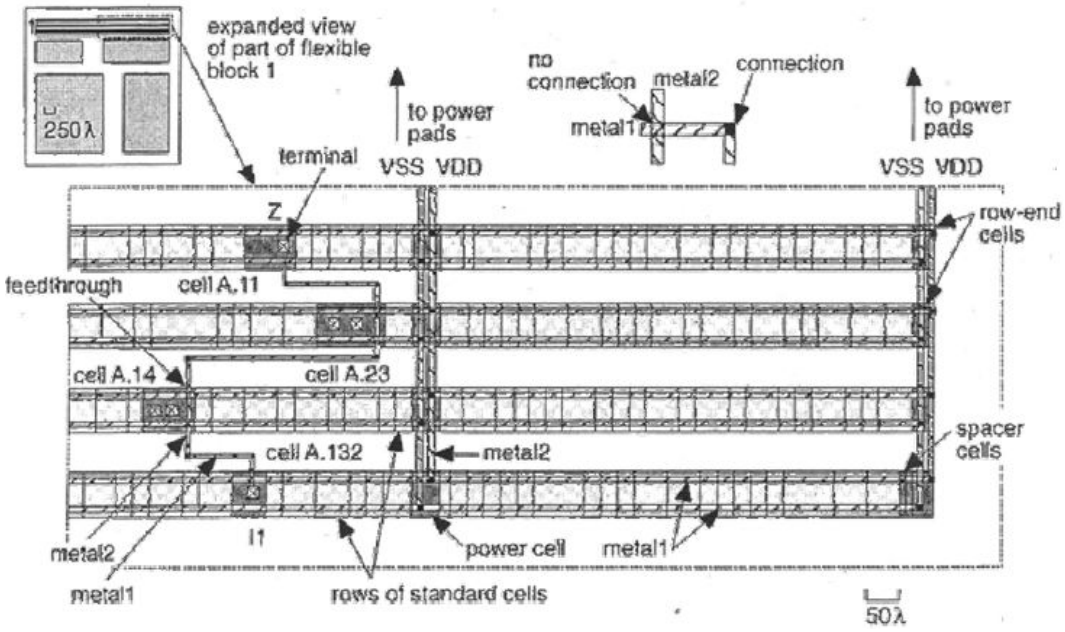


Figura 3.6: Esquema de sistema Semi-Custom Standard-Cells[5].

Las características más destacables de los semi-custom Standard-Cells son:

- Se utilizan sólo las células que se necesitan.
- El tamaño del chip y los canales de interconexión son variables según el diseño.
- La fabricación del C.I. se realiza a partir de la utilización de máscaras, lo que supone unos mayores costes de desarrollo.

- **Gate-Arrays:** Son circuitos lógicos programables realizados mediante máscara. Estos circuitos disponen de una parte predeterminada y otra parte realizada de acuerdo a la especificación del cliente. La parte estándar la constituyen una serie de componentes prefundidos en el sustrato como transistores, resistencias, etc ... y constituyen lo que se denomina el Master-Slice. La parte específica es la que forma las interconexiones entre los componentes de la parte estándar para conseguir una determinada función. El fabricante suministra una biblioteca de células que se pueden generar a partir de los componentes prefundidos, y que tienen funcionalidades como biestables, contadores, registros, etc .... La estructura interna de un C.I. del tipo Gate-Array se puede dividir en: células lógicas, células de Entrada/Salida y espacio para interconexiones.

[5]Imagen tomada de `http://www.scribd.com/doc/79177686/4-PLD`

Las características específicas más destacables de los tipos Gate-Arrays son las siguientes:

- La estructura básica es fija, varían las interconexiones.
- El tamaño y el número de celdas internas es fijo.
- Los canales de interconexión son fijos.

En la figura 3.8 se muestra el esquema interno de un ASIC semi-custom Gate-Arrays.

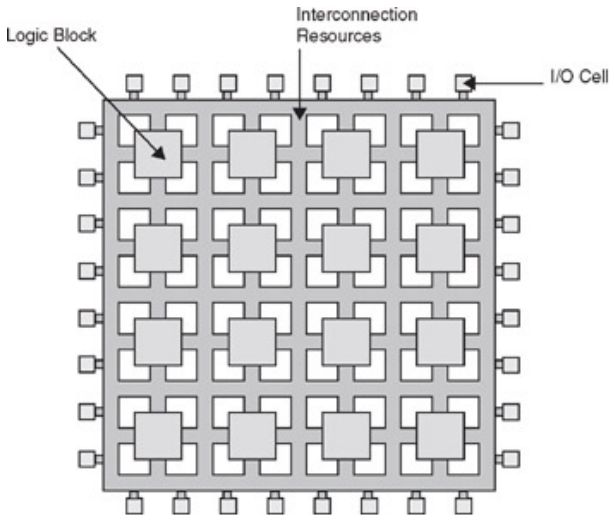


Figura 3.7: Esquema de los ASIC semi-custom Gate-Arrays[6].

Los Gate-Arrays ofrecen el sistema más rápido y económico de los diseños semicustom, pero su principal desventaja es que la implementación final puede contener bloques funcionales que no son utilizados al 100 % de sus posibilidades, con el consiguiente desaprovechamiento de Silicio. Los sistemas basados en Standard-Cells, al no ser una arquitectura tan cerrada como la del Gate-Array, permiten realizar un aprovechamiento óptimo del silicio.

- **Sea-Gates:** Los Sea Gates presentan una estructura similar a los Gate-Arrays, pero sin canales de ruteado. Para realizar las interconexiones se utilizan algunas de las células como rutas utilizando transistores de paso. Presentan una menor utilización final del silicio y están muy orientadas a la realización de RAMs y ROMs. En la figura 3.8 se muestra el esquema de un ASIC semi-custom Sea of Gates.

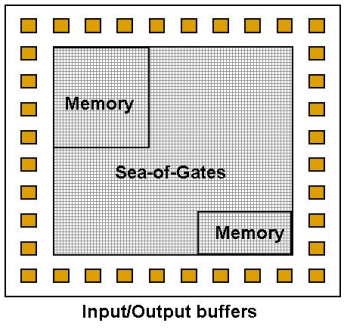


Figura 3.8: Esquema de los ASIC semi-custom Sea of Gates[7].

[6] http://images.books24x7.com/bookimages/id_32210/fig29-1.jpg
[7] http://www.fujitsu.com/img/MICRO/fme/microelectronics/asic/gate_array.gif

3.1.1.3. FPGA

Las FPGAs (*Field Programmable Gate Array*) son circuitos de aplicación específica de alta densidad programables por el usuario en un tiempo reducido y sin la necesidad de verificación de sus componentes, tarea ya realizada por el fabricante al tratarse de un producto estándar. La arquitectura de una FPGA consiste en conjuntos de celdas lógicas que se comunican mediante canales de conexiones verticales y horizontales realizado por un proceso de ruteado del que se encarga un software especializado. Una FPGA consta de tres tipos de elementos programables:

1. Bloques Lógicos Configurables, denominados CLB del acrónimo anglosajón *Configurable Logic Block*
2. Matrices de interconexión, denominados SM del acrónimo anglosajón *Switching Matrix*.
3. Bloques de Entrada/Salida, denominados IOB del acrónimo anglosajón *Input/Output Blocks*.

En la figura 3.9 se muestra la estructura interna de una FPGA.

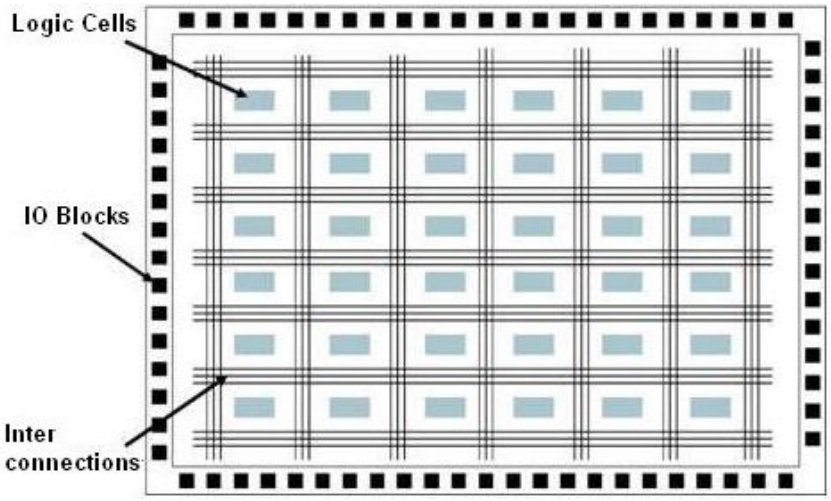


Figura 3.9: Esquema de una FPGA[8].

Cada celda lógica es similar a los bloques lógicos de un CPLD. La estructura de las celdas lógicas y las interconexiones varían de acuerdo al fabricante. En general, una celda lógica tiene menos funcionalidad que la combinación de sumas de productos y macroceldas de un CPLD, pero debido a que una FPGA tiene una gran cantidad de celdas lógicas, es posible implementar grandes funciones utilizando celdas lógicas en cascada. También dispone de células de memoria de configuración, denominados CMC del acrónimo anglosajón *Configuration Memory Cell*, distribuidas a lo largo de todo el chip, las cuales almacenan toda la información necesaria para configurar los elementos programables. Estas células de configuración suelen consistir en bloques de memoria RAM y son inicializadas en el proceso de carga del programa de configuración.

Bloques lógicos configurables (CLB): estos bloques constituyen el núcleo de una FPGA. Cada CLB presenta una sección de lógica combinacional programable y unos registros de almacenamiento. En la figura 3.10 se muestra el esquema interno de un CLB.

[8] `http://fpgacenter.com/fpga/images/fpga_structure_clip_image001.jpg`

[9] `http://upload.wikimedia.org/wikipedia/commons/thumb/1/1c/FPGA_cell_example.png/400px-FPGA_cell_example.png`

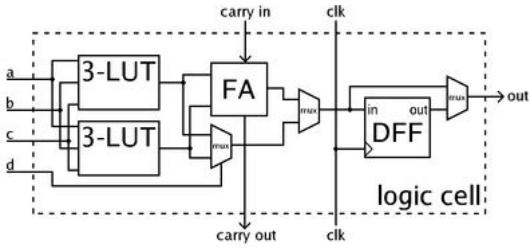


Figura 3.10: Esquema de un bloque CLB[9].

Los registros se utilizan en caso de que la funcionalidad del CLB sea de carácter secuencial. La lógica combinacional la lleva a cabo una LUT que permite implementar cualquier función booleana a partir de las variables de entrada. Su contenido se define mediante las células de memoria. Se presentan también multiplexores como elementos adicionales de direccionamiento de los datos del CLB, los cuales permiten variar el tipo de salidas (combinacionales o registradas), facilitan caminos de realimentación, o permiten cambiar las entradas de los biestables. Se encuentran controlados también por el contenido de las células de memoria.

Bloques de entrada salida (IOB): La periferia de la FPGA está constituida por bloques de entrada/salida configurables por el usuario. Cada bloque puede ser configurado de forma independiente para funcionar como entrada, salida o bidireccional, admitiendo también la posibilidad de control triestado. Los IOBs pueden programarse para trabajar con diferentes niveles lógicos (TTL, CMOS,...) y cada uno de los IOBs incluyen flip-flops que pueden utilizarse para registrar tanto las entradas como las salidas. En la figura 3.11 se puede observar la estructura típica de un bloque IOB.

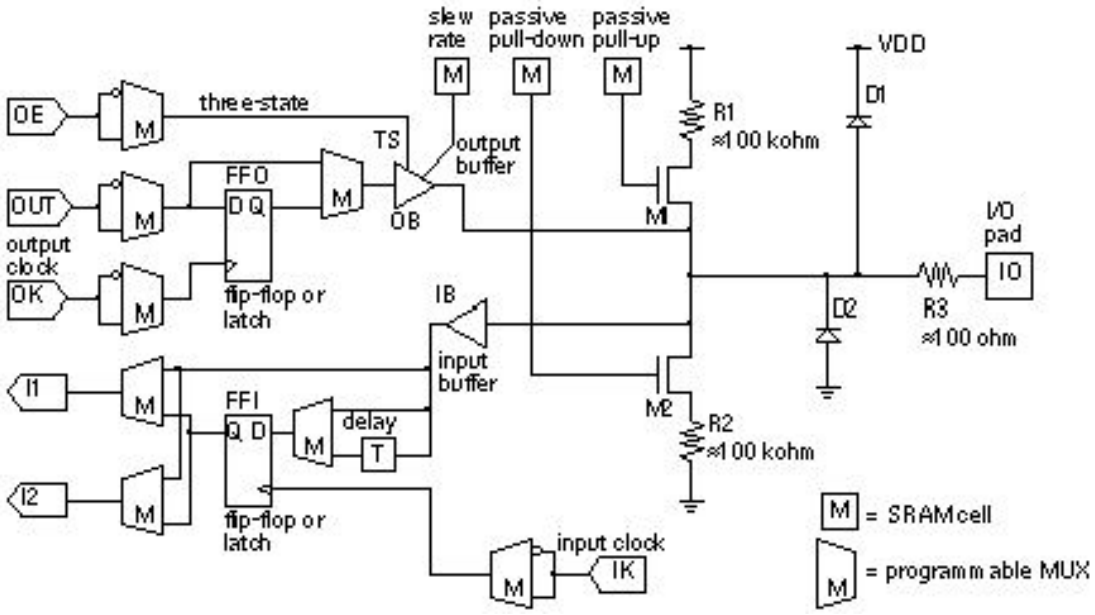


Figura 3.11: Esquema de un bloque IOB[10].

[10] http://iroi.seu.edu.cn/books/asics/Book2/CH06/CH06-21.gif

Líneas de interconexión: constituyen los caminos que permiten interconectar las entradas y salidas de los diferentes bloques. Están formadas por líneas metálicas de dos capas que recorren horizontal y verticalmente las filas y columnas existentes entre los CLBs. Dos elementos adicionales participan activamente en el proceso de conexión:

- **Puntos de Interconexión Programable** (PIP, Programmable Interconnection Point): permiten la conexión de CLBs e IOBs a líneas metálicas cercanas. Consisten en transistores de paso controlados por un bit de configuración.
- **Matrices de interconexión** (SW, Switch Matriz o Magic Box): son dispositivos de conmutación distribuidos de forma uniforme por la FPGA formados internamente por transistores que permiten la unión de las denominadas líneas de propósito general, permitiendo conectar señales de unas líneas a otras.

Familias FPGAs de Xilinx: antes de 2010 Xilinx ofrecía dos familias de FPGAs; la familia Spartan de bajo coste y la familia Virtex de altas prestaciones. Con la introducción de su FPGA de 28nm en junio de 2010 Xilinx reemplazó la familia Spartan por la familia Kintex y la familia Artix de bajo coste. En marzo de 2011 se introduce la familia Zynq-7000 que integra el sistema basado en procesador ARM Cortex-A9 MPCore.

- **Spartan:** el objetivo de la serie Spartan son aplicaciones de bajo consumo, bajo coste y gran volumen de producción, por ejemplo displays, routers ...
- **Kintex:** la familia Kintex es la primera familia de rango medio, situadas entre la familia Spartan de bajo coste y la familia Virtex de alto rendimiento. La familia Kintex incluye: conexiones series de altas prestaciones de velocidad 12.5Gbit/s o y optimizadas para menor coste con velocidad de 6.5Gbit/s, memoria y lógica con rendimiento suficiente para aplicaciones como comunicaciones ópticas de alto volumen y proveen un balance entre rendimiento, consumo y coste de desarrollo para despliege de redes inalámbricas con evolución a largo plazo.
- **Artix:** la familia Artix-7 consigue un 50 % menos de consumo y un coste inferior de 35 % comparado con la familia Spartan-6, y se basa en la arquitectura de la serie Virtex. Xilinx asegura que la familia Artix-7 ofrece: el rendimiento necesario para realizar proyectos sensibles al coste y que es capaz de abarcar el mercado de alto volumen antes realizado sobre ASSPs, ASICs, y low-cost FPGAs. La familia Artix ha sido diseñada para conseguir pequeños factores de forma y bajo consumo.
- **Zynq:** la familia Zynq-7000 está pensada para aplicaciones de sistemas embebidos de gama alta tales como video vigilancia, asistencia automoción-conductor, comunicaciones inalámbricas de última generación y automatización de fábricas. Zynq-7000 integra un sistema completo ARM Cortex-A9 MPCore-processor-based de 28 nm. La arquitectura Zynq difiere de arquitecturas anteriores entre la lógica programable y los procesadores integrados al pasar de una plataforma centrada en FPGA a un modelo centrado en el procesador. Para los desarrolladores software, Zynq-7000 aparece como un SoC basado en el procesador ARM estándar, arrancando directamente tras el encendido y que soporta variedad de sistemas operativos, independientemente de la lógica programable. En 2013, Xilinx introdujo el Zynq-7100, que integra DSP para cumplir con los requisitos emergentes de integración de sistemas programables de la tecnología inalámbrica, difusión, aplicaciones médicas y militares.
- **Virtex:** la serie Virtex de FPGAs tiene características integradas que incluyen lógica FIFO y ECC, bloques DSPs, controladores de PCI-Express, bloques de Ethernet MAC y transceptores de alta velocidad. Además de la lógica de la FPGA, las series Virtex incluye funciones hardware embebidas como multiplicadores, memorias, transceptores serie y microprocesadores. Algunos de los miembros de la familia Virtex están disponibles con embalaje resistente a radiación, pensados para

operar en el espacio, donde ráfagas de partículas de alta energía pueden causar estragos en los semiconductores.

3.1.2. Comparativa entre tecnologías.

En la figura 3.12 se muestra una comparativa de entre coste de ingeniería y tiempo de desarrollo del producto contra velocidad, densidad, complejidad, volumen de mercado y necesidad por el producto.

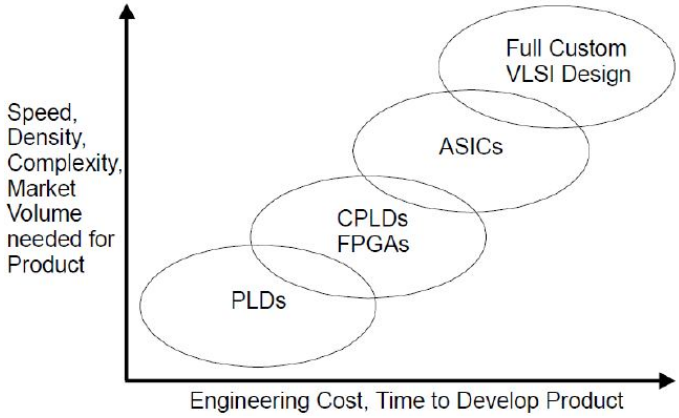


Figura 3.12: Comparativa entre tecnologías.

En la tabla 3.1 se muestra una comparativa de características entre FPGAs y ASIC basadas en Gate-Arrays. Mientras los dispositivos ASIC poseen mejor densidad de integración y mejores prestaciones, las FPGAs son reconfigurables y se ahorra tiempo de diseño.

	Gate-Array	**FPGA**
Tipo de producto	Específico	Estándar
Impacto en la producción	Retrasos en la producción	Rapidez de distribución en mercado
Programación	Sólo en proceso de fabricación	Por el usuario (reprogramable)
Simulación	Complicada	Fácil
Verificaciones en fases previas	Imposible	Posible
Cambios en el diseño	Muy costosos	Posible en cualquier momento
Comprobación del dispositivo	Específica para cada diseño	Comprobada por el fabricante

Tabla 3.1: Diferencias entre Gate-Array y FPGA.

Por último se muestra la figura 3.13 que presenta con mayor detalle el tiempo de diseño necesario para realizar el desarrollo en una FPGA y en una ASIC.

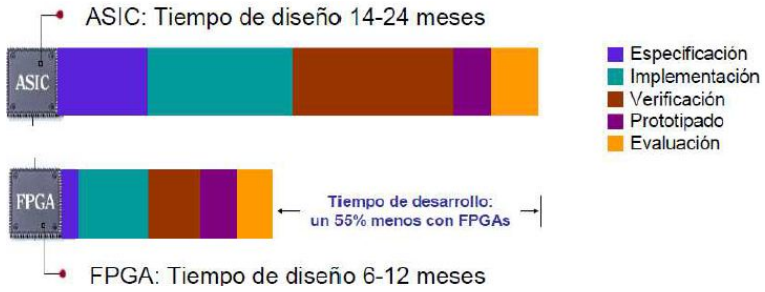


Figura 3.13: Diferencias de tiempo de desarrollo entre un ASIC y una FPGA[11].

Además de los tiempos inferiores, el desarrollo en una ASIC es mucho más costoso. Por estas dos razones se ha llevado a cabo el desarrollo del sistema en una FPGA.

3.2. Modulaciones digitales

Según la *American National Standard for Telecommunications*, *"la modulación es el proceso, o el resultado del proceso, de variar una característica de una portadora de acuerdo con una señal que transporta información"*. Un sistema de modulación, está compuesto principalmente por una señal portadora y una señal moduladora. La señal portadora es una onda eléctrica modificada en alguno de sus parámetros, frecuencia, fase, amplitud o una combinación de los anteriores, por la señal de información y que se transporta por el canal de comunicaciones. En el sistema propuesto se utilizan las modulaciones digitales, en dichas modulaciones una portadora analógica es modulada por una señal discreta. Existen diversas modulaciones digitales atendiendo a los parámetros que se modifican en la señal portadora. En las siguientes subsecciones se presentan las modulaciones utilizadas en el sistema.

3.2.1. Modulación binaria por desplazamiento de fase

La modulación binaria por desplazamiento de fase o BPSK, del término anglosajón *"Binary Phase Shift Keying"*, consiste en hacer variar la fase de la portadora entre dos valores discretos. La figura 3.14 muestra una posible constelación para una modulación BPSK con los valores seno para un código y -seno para el otro código.

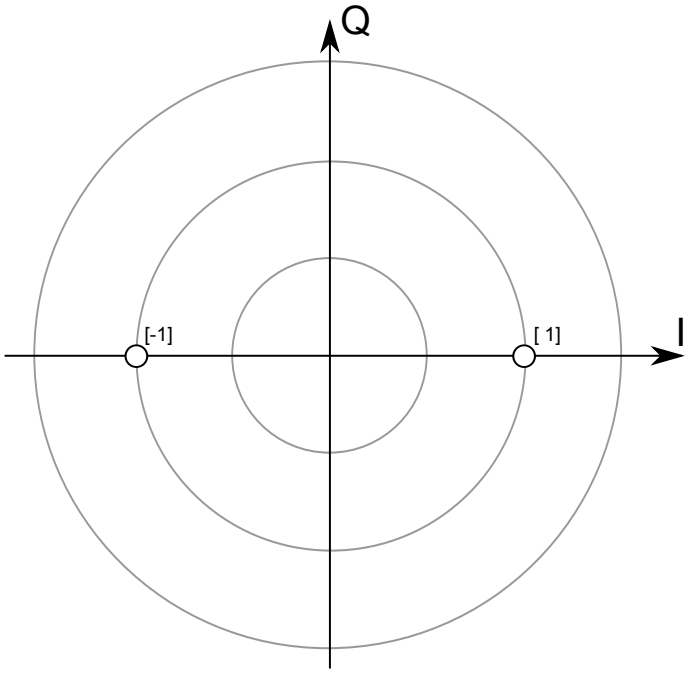


Figura 3.14: Ejemplo de constelación BPSK.

En la figura 3.15 se muestra el código [1,-1,-1, 1,-1, 1, 1, 1, 1,-1] modulado con una señal senoidal y siguiendo la constelación mostrada en la figura 3.14.

[11] Imagen tomada de `http://laimbio08.escet.urjc.es/assets/files/docencia/EDII/Tema3FPGAs.pdf`

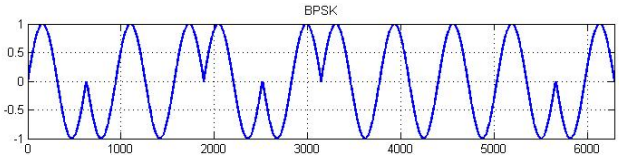


Figura 3.15: Ejemplo de modulación con la constelación de la figura 3.14.

3.2.2. Modulación por desplazamiento de fase en cuadratura

La modulación por desplazamiento de fase en cuadratura o QPSK, del término anglosajón *"Quadrature Phase Shift Keying"*, consiste en realizar la codificación con cuatro fases distintas. Con estas cuatro fases, QPSK puede codificar dos bits por cada símbolo. La asignación de cada par de bits a cada símbolo es realizada por el diseñador pero suele hacerse mediante el código Gray de manera que entre dos símbolos adyacentes, los símbolos sólo se diferencien en un bit, con lo que se logra reducir la tasa de bits erróneos. Con respecto a BPSK tiene la ventaja de que usando el mismo ancho de banda transmite el doble de datos, pero en contrapartida es un sistema menos inmune al ruido. La figura 3.16 muestra una constelación QPSK codificada con códigos Gray.

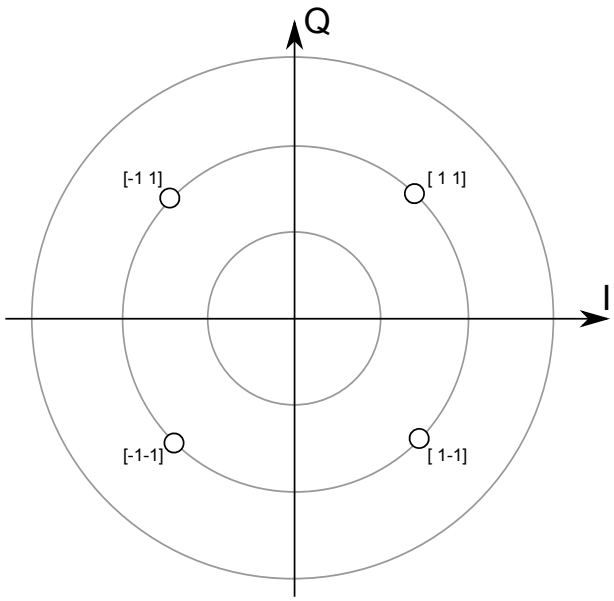


Figura 3.16: Ejemplo de constelación QPSK.

En la figura 3.17 se muestra el código [1 1, 1 -1, 1 1,-1 1,-1 -1, 1 1,-1 -1, 1 -1,-1 1,-1 1] modulado con una señal senoidal (*I=sin(x)*, *Q=cosine(x)*) y siguiendo la constelación mostrada en la figura 3.16.

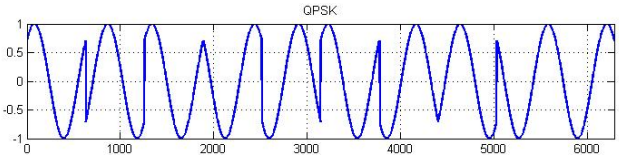


Figura 3.17: Ejemplo de modulación con la constelación de la figura 3.16.

3.2.3. Modulación por desplazamiento de amplitud y fase

La modulación por desplazamiento de amplitud y fase o APK, del término anglosajón *Amplitude Phase Keying*, consiste en realizar la codificación con dos fases y N niveles de amplitud. Se pueden codificar $log_2(Nniveles \cdot 2)$ por cada símbolo. Por ejemplo, para la constelación mostrada en la figura 3.18 con dos niveles de amplitud se pueden codificar dos bits por símbolo, en este caso además se ha usado una codificación Gray por las mismas razones comentadas para QPSK.

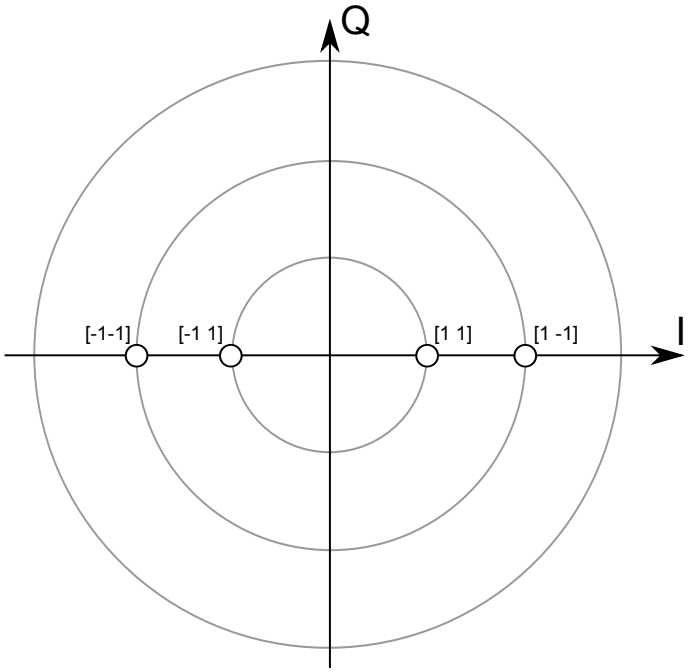


Figura 3.18: Ejemplo de modulación APK de dos amplitudes.

En la figura 3.19 se muestra el código [1 -1,-1 -1,-1 1, 1 1, -1 1,-1 -1, 1 -1, 1 1,-1 1] modulado con una señal senoidal (*I=sin(x)*) y siguiendo la constelación mostrada en la figura 3.18.

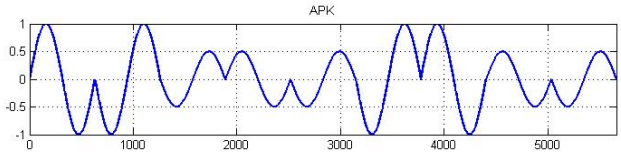


Figura 3.19: Ejemplo de modulación con la constelación de la figura 3.18.

3.2.4. Modulación por desplazamiento de amplitud en cuadratura

La modulación por desplazamiento de amplitud en cuadratura o QAM, del término anglosajón *Quadrature Amplitude Modulation*, es una técnica que transporta datos, mediante la modulación de la señal portadora, tanto en amplitud como en fase. De forma general se llama N-QAM a esta modulación debido a que se pueden tener *N* símbolos. Las formas más comunes son de 16-QAM, 64-QAM y 256-QAM. Al cambiar a una constelación de orden superior es posible transmitir más bits por segundo con el mismo ancho de banda, pero en contrapartida el sistema es más susceptible al ruido y a la distorsión

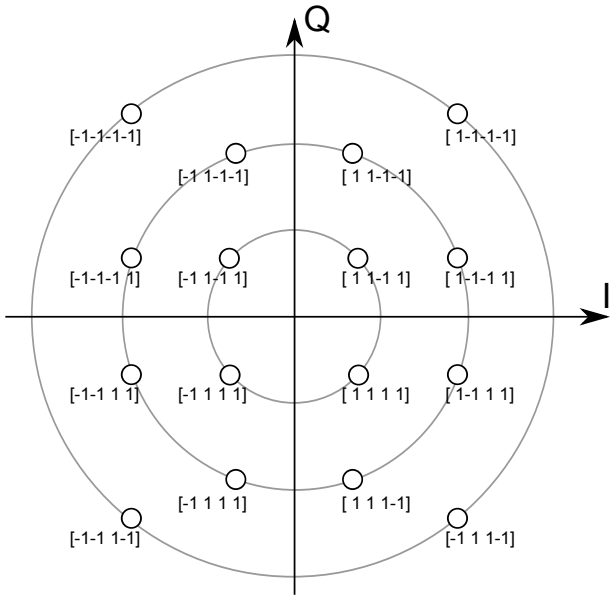


Figura 3.20: Ejemplo de modulación QAM de 16 fases y dos amplitudes, 16-QAM.

En la figura 3.21 se muestra el código [1 -1 -1 -1,-1 1 1 1 , 1 1 -1 1,-1 -1 -1 1,-1 1 1 -1,-1 1 1 1, 1 1 -1 1,-1 -1 1 -1, 1 1 -1 1,-1 1 1 1] modulado con una señal senoidal (*I=sin(x)*, *Q=cosine(x)*) y siguiendo la constelación mostrada en la figura 3.20.

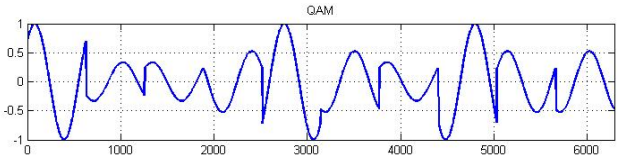


Figura 3.21: Ejemplo de modulación con la constelación de la figura 3.20.

3.3. Sistemas de posicionamiento

Un sistema de posicionamiento se centra en posicionar un elemento en un entorno conocido. Al hablar de posicionamiento existen dos técnicas:

- **Posicionamiento relativo:** consiste en el incremento de la posición de un móvil a partir de la posición inicial del mismo. Este tipo de posicionamiento tiene como inconvenientes la necesidad de conocer dicha posición inicial y que el error de posicionamiento es acumulativo, sin embargo tiene la ventaja del fácil procesamiento de datos que presenta.
- **Posicionamiento absoluto:** consiste en el posicionamiento de un elemento respecto a un sistema de referencia externo. En caso de utilizar este tipo de técnicas no se tienen errores acumulativos a costa de un procesamiento de datos más complejo y presentan otras fuentes de errores más complejos. Destacan en este tipo los sistemas de visión, balizamiento y GPS.

Los sistemas de posicionamiento se pueden dividir en dos grupos, los sistemas outdoor e indoor, y se utilizará uno, otro o una fusión de ambos dependiendo del entorno en el que se encuentre el elemento

a posicionar.

3.3.1. Sistemas outdoor

Los sistemas outdoor son sistemas de posicionamiento local o global que permiten la estimación de la posición de un móvil en un mapa conocido situado en el exterior. Hoy en día el sistema de posicionamiento de exteriores más extendido es el GPS, desarrollado, instalado y empleado por el Departamento de Defensa de los Estados Unidos, aunque existen otros sistemas similares llevados a cabo por otras instituciones, como el proyecto *GALILEO* en proceso de desarrollo por la Unión Europea, el GLONASS desarrollado por la Unión Soviética ...

GPS: del acrónimo inglés *Global Positioning System*. El GPS funciona mediante una red de 24 satélites en órbita sobre la tierra, a 20.200km, con trayectorias sincronizadas para cubrir toda la superficie de la Tierra. Cuando se desea determinar la posición, el receptor localiza automáticamente como mínimo tres satélites de la red, de los que recibe unas señales indicando la identificación y la hora del reloj de cada uno de ellos. Con base en estas señales, el aparato sincroniza el reloj del GPS y calcula el tiempo que tardan en llegar las señales al equipo, y de tal modo mide la distancia al satélite mediante medida de tiempo de vuelo. Conocidas las distancias, se determina fácilmente la propia posición relativa respecto a los tres satélites mediante triangulación. Conociendo además las coordenadas o posición de cada uno de ellos, se obtienen las coordenadas reales del punto de medición. La figura 3.22 muestra la captura de una animación donde se muestran desplegados todos los satélites GPS con sus órbitas y la visibilidad que tienen los satélites desde el elemento a posicionar en cada momento.

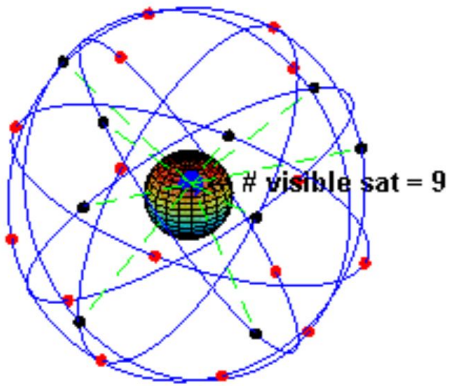


Figura 3.22: Despliege de satélites GPS y visión por el elemento a posicionar[12].

3.3.2. Sistemas indoor

Los sistemas indoor son sistemas de posicionamiento local, o LPS (del acrónimo anglosajón Local Positioning system), que permiten la estimación de la posición de un móvil en un mapa conocido. Tratan de cubrir las limitaciones del sistema GPS que demanda visión directa con los satelites, y por tanto es incapaz de posicionar en interiores, y tiene menor precisión. Cabe destacar que la posición se realiza mediante trilateración a partir de las distancias a cada una de las balizas. Dicha distancia se puede medir mediante la temporalización de tiempo de vuelo, medida de fase ... A continuación se muestran algunas de las tecnologías utilizadas para el posicionamiento en interiores:

[12] http://upload.wikimedia.org/wikipedia/commons/9/9c/ConstellationGPS.gif

Ultrasonidos: se basan en la emisión/recepción de ultrasonidos y medida de tiempos de vuelo o ToF (*Time of Fligth*) desde que se produjo la emisión hasta la recepción o a través de medidas diferenciales. En la figura 3.23 se muestra un sistema con 8 balizas emisoras de ultrasonidos y un micrófono puesto en el móvil a localizar (también podría darse el caso de un emisor y un sistema de balizas con 8 micrófonos).

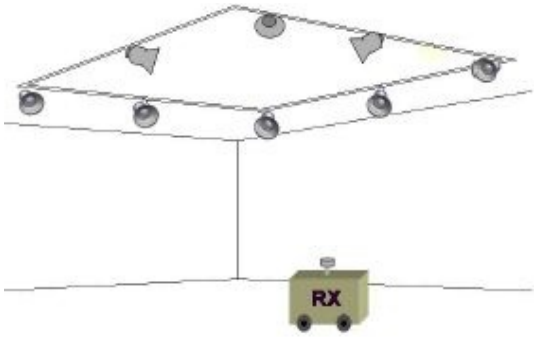


Figura 3.23: Despliege de balizas y móviles a posicionar[13].

Los ultrasonidos tienen como ventajas y desventajas frente a otros sistemas de posicionamiento local:

- Ventajas:
  - Sistema económico y robusto.
  - Tiene una precisión de centímetros dada la baja velocidad de propagación de los ultrasonidos en aire.
  - No producen interferencias radio-eléctricas con otros componentes.
- Desventajas
  - Puede haber falsas medidas debidas a las reflexiones en muros, objetos ... El algoritmo de posicionamiento tendrá que considerarlo.
  - Lentitud relativa en los tiempos de medida.
  - Efectos indeseados en la medida debido a la velocidad del móvil.
  - Mal comportamiento en ambientes exteriores debido a factores como turbulencias, la heterogeneidad de la temperatura del medio ...

Radio frecuencia (o RF): éstos sistemas de posicionamiento están basados en una onda electromagnética. La velocidad de propagación será dependiente de las características del medio de propagación siendo la velocidad máxima la de la luz (299.792.458 m/s) y dándose en caso de que el medio de propagación sea el vacío. Cabe destacar que los efectos de la propagación afectan al rendimiento de los LPS que usen tecnología RF.

- **Ultrawide-band radio:** se considera ultrawide-band, o UWB, a aquellas emisiones cuyo ancho de banda fraccional, calculado como se muestra en la ecuación 3.1, es menor a 0.2 y el ancho de banda de la señal es de al menos 500MHz. En la figura 3.24 se muestra el ancho de banda de una señal UWB y la máscara espectral impuesta para esta tecnología.

$$B_f = 2 \cdot \frac{f_H - f_L}{f_H + f_L} \tag{3.1}$$

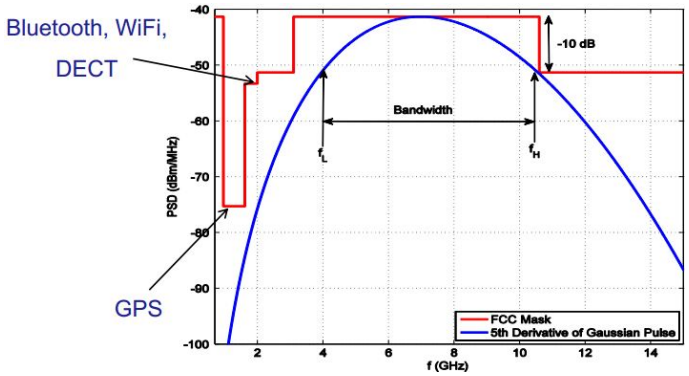


Figura 3.24: Limitaciones en ancho de banda de la tecnología UWB[14].

Los sistemas de posicionamiento basados en esta tecnología pueden realizar la estimación de la medida a partir de:

- La medida de potencia de señal: para poder utilizar esta técnica se necesita realizar una medida de puntos en la escena para realizar un mapeo y poder realizar una asociación entre medidas y puntos en el mapa.
- La medida de ángulos de llegada: la medida se realiza a partir de la medición de los ángulos del móvil, éstos ángulos se obtienen por medio de redes de antenas. La principal limitación de este método está relacionada con la necesidad de la utilización de redes de antenas ya que son dispositivos caros y relativamente grandes.
- La medida de tiempos de vuelo: se basa en la medida de tiempo de llegada al receptor de la señal transmitida desde el emisor.

Para cualquiera de las tipologías de medida se presentan una serie de ventajas y desventajas de esta tecnología:

- Ventajas:
  - La señal UWB es más inmune al efecto del multicamino debido al pequeño ancho de la señal emitida.
  - Alta velocidad de transmisión.
  - Alta resolución de medida.
  - Las señales UWB pueden coexistir con comunicaciones de banda estrecha, mejorando la eficiencia espectral debido a la baja densidad espectral, tienen el mismo nivel que el ruido de fondo producido por emisiones no intencionadas.
- Desventajas
  - Tecnología cara. Los receptores basados en detección por correlación han de tener una alta frecuencia de muestreo y los componentes de radiofrecuencia han de tener un alto ancho de banda.
  - Se emiten señales con potencia de emisión menor al ruido de fondo.

[13]Imagen extraída de `http://www.depeca.uah.es/depeca//repositorio/asignaturas/200358/Tema4_1_ENG.pdf`

[14]Imagen extraída de `http://www.depeca.uah.es/depeca//repositorio/asignaturas/200358/Tema4_1_ENG.pdf`

- **Posicionamiento/seguimiento de teléfonos móviles: GSM, LTE.** El posicionamiento se lleva a cabo usando multirateración basado en la potencia de señal o en el tiempo de llegada de la señal a la estación base más cercana. Existen tres aproximaciones para el posicionamiento:

  - **Basados en red:** la red del proveedor de servicios identifica la ubicación del dispositivo móvil. No está orientado a privacidad.
  - **Basados en el dispositivo móvil:** requiere un software cliente instalado en el dispositivo móvil para determinar su posición. Está orientado a la privacidad.
  - **Híbrido:** usan los dos, tanto la red como el dispositivo móvil.

- **Redes inalámbricas:** en este grupo se incluyen Wi-Fi, Bluetooth, Zigbee ... Normalmente las medidas están basadas en el indicador de potencia de señal recibida, o RSSI, y dependiendo de la densidad de emisores y de la complejidad del ambiente se pueden conseguir precisiones de 1 a 5 metros. Algunas de las ventajas de ésta tecnología son:

  - Estandarización, ubicuidad y bajo coste.
  - En algunos casos, infraestructura ya desplegada puede ser rehusada, por ejemplo en el caso de redes Wi-Fi.
  - Se puede superponer a la labor de posicionamiento otra de transmisión de datos.

  Dado que de cara al posicionamiento estas tecnologías se usan de manera similar, se explicará únicamente la red ZigBee. Esta tecnología tiene tres tipos de dispositivos como se puede observar en la figura 3.25.

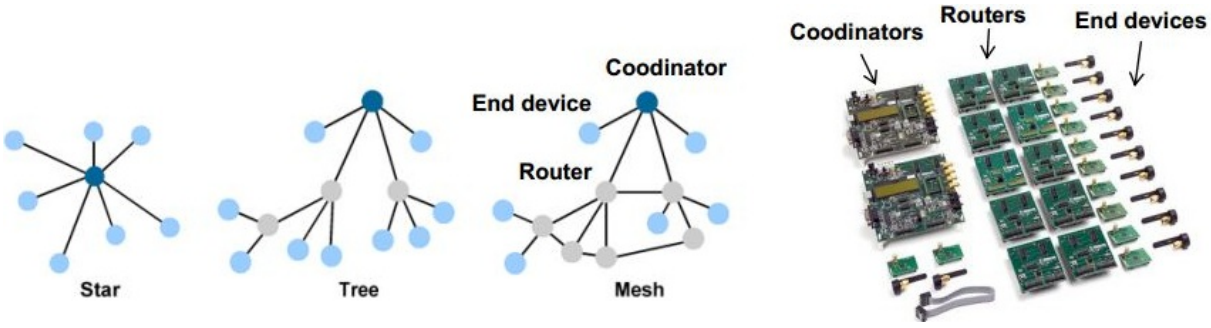


Figura 3.25: Distribuciones de redes ZigBee[15].

  - Coordinador de red, que puede generar las tres tipologías mostradas en la figura 3.26; de estrella, de árbol y tipo malla.
  - Router, cuya función es la de interconexión de los distintos dispositivos de la red. La topología de mallado presenta la ventaja de que si un router se pierde puede realizarse todavía la conexión a todos los dispositivos a través de otros routers.
  - Dispositivo final, cuya función es mandar la información de la posición al coordinador y a los routers.

  En la figura 3.26 se muestra un mapa de una planta sobre la cual se ha desplegado un sistema de posicionamiento basado en ZigBee, utilizando los elementos mostrados en la figura 3.25. Con una situación como la presentada, el sistema sería capaz de diferenciar en qué habitación o pasillo se encuentra el móvil.

[15]Imagen extraída de `http://www.depeca.uah.es/depeca//repositorio/asignaturas/200358/Tema4_1_ENG.pdf`

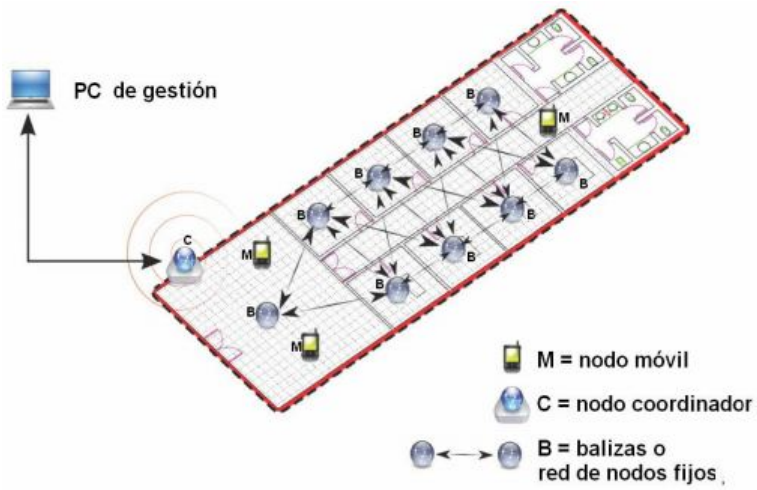


Figura 3.26: Ejemplo de sistema de posicionamiento basado en ZigBee[16].

- **RFID:** del término anglosajón *Radio Frecuency IDentification*, se trata de una tecnología que usa ondas de radio-frecuencia para transferir datos entre un lector y un ítem móvil etiquetado para identificar, categorizar, seguir . . . La tecnología RFID es rápida, fiable y no necesita contacto visual directo entre el lector y el ítem etiquetado. Se muestran en la figura 3.27 los dos tipos de etiquetas existentes en la tecnología RFID.

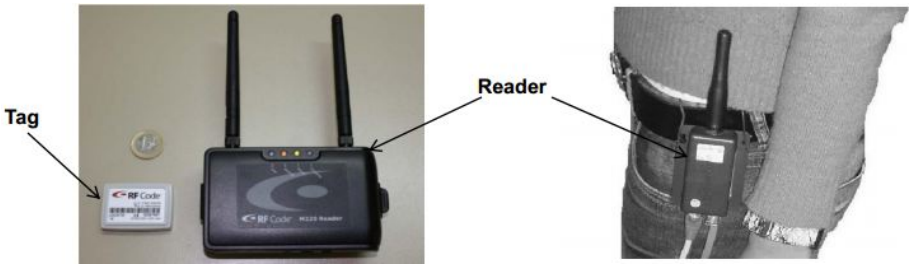


Figura 3.27: Comparación de etiquetas activa y pasiva[17].

  - Etiquetas activas: emiten la señal de radio, para ello necesitan baterías y tienen un alto rango de lectura.
  - Etiquetas pasivas: las etiquetas reflejan la señal de radio del lector modulada en función de la etiqueta, por tanto el rango de lectura dependerá de la potencia del lector, en cualquier caso es muy inferior a la de las etiquetas activas.

Visión: el posicionamiento basado en visión utiliza el mismo principio que el posicionamiento basado en marcas o mapas pero utilizando sensores ópticos. Además, dado que las cámaras proporcionan información en 2 dimensiones, suponiendo que el móvil se mueve en un plazo Z conocido, será suficiente una cámara para realizar el posicionamiento. Por contrapartida, se necesita información detallada previa del ambiente y visión directa del elemento a posicionar. En la figura 3.28 se muestra un ejemplo de

[16]Imagen extraída de `http://www.depeca.uah.es/depeca//repositorio/asignaturas/200358/Tema4_1_ENG.pdf`

[17]Imagen extraída de `http://www.depeca.uah.es/depeca//repositorio/asignaturas/200358/Tema4_1_ENG.pdf`

posicionamiento y seguimiento utilizando una cámara de tráfico convencional.

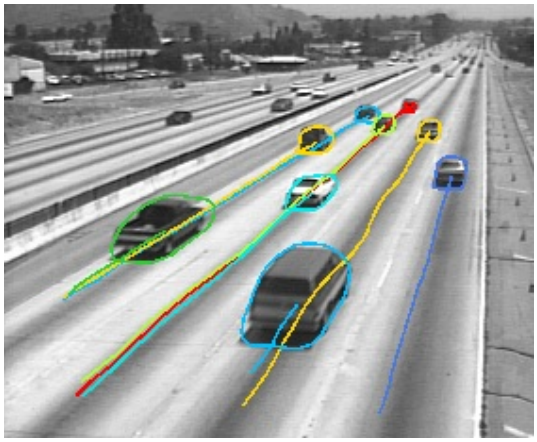


Figura 3.28: Ejemplo de posicionamiento y seguimiento[18].

Infrarrojos: las longitudes de onda de señales infrarroja son más largas que la de la luz visible, pero más corta que la de la radiación de terahercios. Por lo tanto, la luz infrarroja es invisible para el ojo humano bajo la mayoría de condiciones, por lo que esta tecnología es menos invasiva en comparación con el posicionamiento en interiores basado en la luz visible. Los tres métodos generales de la explotación de las señales infrarrojas son:

- **Balizas activas:** se basa en receptores de infrarrojos fijos colocados en lugares conocidos a lo largo de un espacio interior y balizas móviles cuyas posiciones son desconocidas. La arquitectura del sistema puede incluir un solo receptor en cada habitación para precisión sencilla en una habitación o un receptor con capacidades adicionales para precisión mayor. Para lograr precisiones a nivel de metro o mejores, se necesita la configuración de un sistema de seguimiento de IR basado en balizas activas, la cual debe incluir varios receptores desplegados en cada habitación para eliminar la ambigüedad de los sectores de una habitación. Hay que tener en cuenta que las señales de IR son incapaces de penetrar en materiales opacos, como paredes y techos.
- **Imágenes por infrarrojos utilizando la radiación natural (térmica).** En la literatura, los sistemas de posicionamiento que utilizan radiación infrarroja natural se conocen como sistemas de localización de infrarrojos pasivos. Los sensores operan en el espectro de longitud de onda larga de infrarrojos (8 micras a 15 micras, también conocida como la región de termografía) son capaces de obtener una imagen completamente pasiva de emisiones térmicas naturales del mundo circundante. Por lo tanto, no es necesario emplear iluminadores infrarrojos activos o cualquier otra fuente térmica dedicada. La radiación infrarroja térmica se puede utilizar para determinar de forma remota la temperatura de personas u objetos sin ninguna necesidad de utilizar etiquetas o emisores. Los detectores térmicos existentes son: cámaras térmicas, detectores de banda ancha (Las células Golay), sensores de infrarrojos piro-eléctricos utilizados para la detección de movimiento o termopares utilizados para convertir los gradientes de calor en electricidad o para medir la temperatura sin contacto. Como desventaja, los enfoques de infrarrojos pasivos se ven comprometidos por una fuerte radiación solar.
- **Fuentes de luz artificial.** Los sistemas de posicionamiento en interiores basados en fuentes de luz IR activos y cámaras CCD sensibles IR son una alternativa común a los sistemas ópticos que

[18] `http://www.societyofrobots.com/images/programming_vision_tracking.gif`

operan en el espectro de luz visible. La figura 3.29 se muestra el proceso de medida de una cámara de infrarrojos basada en medidas de tiempo de vuelo.

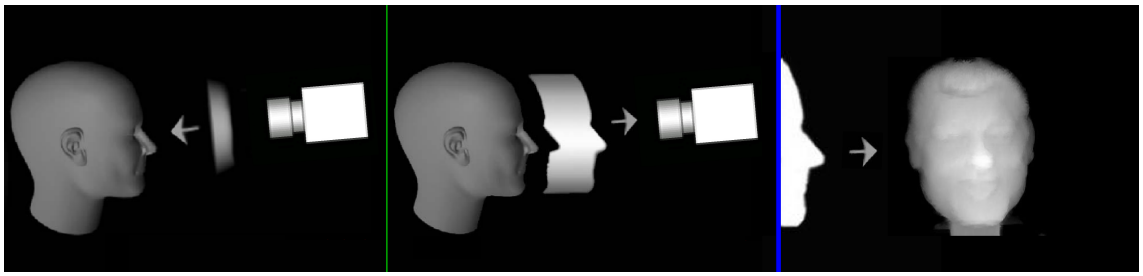


Figura 3.29: Cámara de tiempo de vuelo[19].

La tabla mostrada en la figura 3.2 presenta algunos de los sistemas tanto en estado de desarrollo, en fase de demostración o como producto final de mercado. El producto de mercado más conocido mostrado en la tabla es la *Kinect*, desarrollada por *Microsoft* para la plataforma de entretenimiento *Xbox 360*.

Name	Year	Measuring Principle	Reported Accuracy	Coverage	Target Ilumination	Update Rate	Market Maturity
Active Badges	1999	cell of origin	6 m	scalable	signal transmission	0.1 Hz	product
Atsuumi	2010	polarized light	2 %	3 m	photo detector	high	demonstrator
Hauschildt	2010	angle of arrival	dm	30 m^2	natural IR radiation	-	demonstrator
Ambiplex	2011	angle of arrival	20-30 dm	10 m	natural IR radiation	50 Hz	product
Boochs et al.	2010	IR camera	0.05 mm	4 m^3	active, LED	-	development
Lee and Song	2007	IR camera	dm	36 m^3	retro reflective	30 Hz	development
ALCON ProCam	2011	IR camera	0.01 mm	vehicle	retro reflective	20 Hz	product
Hagisonic StarGazer	2007	IR camera	$cm - dm$	scalable	retro reflective	10 Hz	product
Kinect	2011	Structured light	1 cm	3.5 m	passive	30 Hz	product

Tabla 3.2: Tabla comparativa[20].

[19] http://upload.wikimedia.org/wikipedia/commons/1/19/TOF_Kamera_3D_Gesicht.jpg
[20] Tabla extraída de http://e-collection.library.ethz.ch/eserv/eth:5659/eth-5659-01.pdf

Capítulo 4

Plataforma propuesta

El presente Trabajo Fin de Máster se centra en el diseño e implementación de un sistema de control y configuración de un LPS, con una serie de parámetros configurables controlados desde una entidad externa mediante una conexión Wi-Fi. Además el sistema tendrá una serie de salidas analógicas y una entrada digital que permitirán la interactuacción con el LPS. El sistema completo consta de una entidad externa encargada de la configuración de los parámetros de emisión del LPS, un receptor Wi-Fi que permite el acceso a la plataforma de emisión de forma inalámbrica y desde diversas unidades externas, una plataforma Genesys que se encargará de la generación de los diferentes códigos modulados y los módulos de interacción con los transductores que forman el LPS.

4.1. Entidad externa con emisor Wi-Fi

Se ha elegido como entidad externa un PC con conexión Wi-Fi, pero podría ser cualquier elemento con conexión Wi-Fi. En este trabajo se ha elegido como software tanto para emisión como para la interfaz gráfica Matlab. Mediante la interfaz gráfica desarrollada, Matlab, se recogen todos los parámetros configurables. Éstos serán tratados en diferentes scripts programados en Matlab y posteriormente mandados vía Wi-Fi. Los parámetros que se pueden configurar a través de la interfaz gráfica desarrollada son:

- Tipo de modulación del código a emitir: BPSK, APK, QPSK o QAM.
- Separación temporal entre emisiones, separados en múltiplos de 5 milisegundos.
- Códigos a emitir, que se encuentran en un archivo que contendrá los códigos a emitir por el LPS.
- Moduladora a emitir. Portadoras de perfil sinusoidal o cuadradas.
- Número de símbolos por código, permite introducir el número de moduladoras por cada bit del código.
- Número de muestras por unidad moduladora, es decir, el número de muestras que se le da a la moduladora para ser calculada.
- Frecuencia de emisión. 400khz, 416.66khz y 500khz que son las tres frecuencias de trabajo usual de la actual unidad ultrasónica.
- Modo de trabajo, existen dos posibles modos de funcionamiento de los LPS, como maestro o como esclavo.
- IP y puerto de conexión ya que ésta no será única.

El sistema está pensado teniendo en cuenta que puede existir más de un LPS conectado a una misma red Wi-Fi, por lo que ha de ser posible detectar qué unidades están encendidas así como su configuración inicial en caso de que la tenga.

4.2. Receptor Wi-Fi

Se ha utilizado un router Wi-Fi de Tp-Link, más concretamente el modelo TL-WR702N mostrado en la figura 4.1. Para poder conectarse es necesario asignar las IP de cada componente de forma manual. Esto es, en cada elemento se escribe la IP, ésta debe estar dentro del rango de IPs del Router que van entre *192.168.0.X* (donde X puede ser cualquier valor entre 0 y 253), la máscara de subred, que es *255.255.255.0* y la puerta de enlace que es *192.168.0.254*, que es la IP del router.

Figura 4.1: Router Wi-Fi[1].

4.3. Plataforma Genesys de Xilinx.

El diseño se ha realizado sobre la plataforma Genesys de Xilinx mostrada en la figura 4.2. Dicha plataforma esta compuesta por varios elementos, entre los que cabe destacar: una FPGA virtex-5, 4 conectores PMOD, un conector Ethernet, un controlador de la MAC de Ethernet y unidades de memoria RAM y FLASH como se muestra en la figura 4.5. Se ha elegido una FPGA de la familia Virtex dado que se necesita el despliegue de un MicroBlaze y esta ofrece recursos suficientes como para no plantear problemas a nivel de espacio físico y de memoria. Además se ha seleccionado la placa de desarrollo Genesys dado que tiene un controlador de la capa MAC de Ethernet, memoria FLASH y los conectores PMOD suficientes.

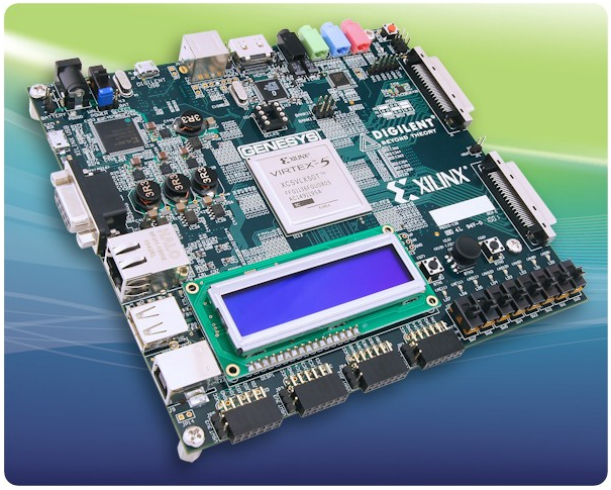


Figura 4.2: Tarjeta Genesys[2].

El SoC se despliega en la Virtex-5. El diseño se ha de repartir de forma óptima entre el software, llevado a cabo en el MibroBlaze y el diseño hardware que se llevará en el resto de la FPGA-fabric. El microBlaze se encarga de gestionar la comunicación Ethernet de entrada y de parametrizar el periférico de propósito específico de generación propia (a partir de este momento se referirá a este como controlador)

[1] http://www5.pcmag.com/media/images/289641-tp-link-tl-wr702n-nano-router-ports.jpg

[2] http://www.digilentinc.com/Data/Products/GENESYS/GENESYS-obl-600.jpg

y del control de la FLASH donde se guardaran datos necesarios una vez se inicie el sistema. El controlador es configurable a partir de una serie de registros que comparte con el microBlaze. Este implementa los diagramas de estado y los módulos de gestión de los bloques necesarios para el control de los DACs, así como del bit de sincronía. Cuando la plataforma haya sido configurada como esclava, este pin indica el momento de emisión de las balizas. La comunicación entre el microBlaze y el periférico propio se realizará a través del bus PLB, el cual escribe en una serie de registros que sirven para configuración.

4.4. Interacción con LPS

El último paso es la conexión de la placa Genesys, con los elementos externos. Por un lado se necesitan seis DACs, cinco para la emisión de los códigos modulados y uno para la emisión de la señal de sincronía para los LPS esclavos. Este DAC sólo estará activo si el LPS es configurado como maestro. Xilinx proporciona unos circuitos integrados de fácil conexión con sus plataformas mediante los puertos PMOD, éste es el PMOD DA2 mostrado en la figura 4.3.Cada PMOD DA2 tiene dos DACs, por lo que serán necesarios tres módulos PMOD DA2.

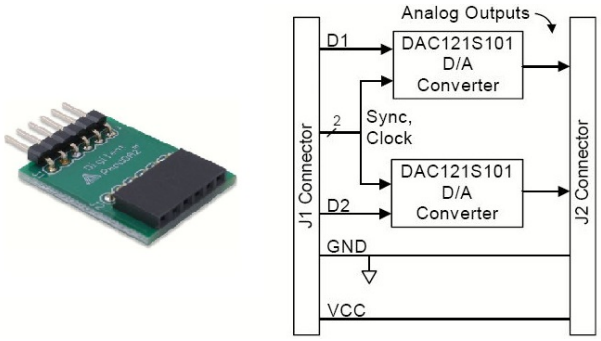


Figura 4.3: PMOD DA2 de Genesys[3].

4.5. LPS

La plataforma externa a controlar es un LPS (*Local Positioning System*) o sistema de posicionamiento local, en la figura 4.4 se muestran dos LPS desplegados en la sección oeste de la planta 2 de la Escuela Politécnica Superior de Alcalá de Henares.

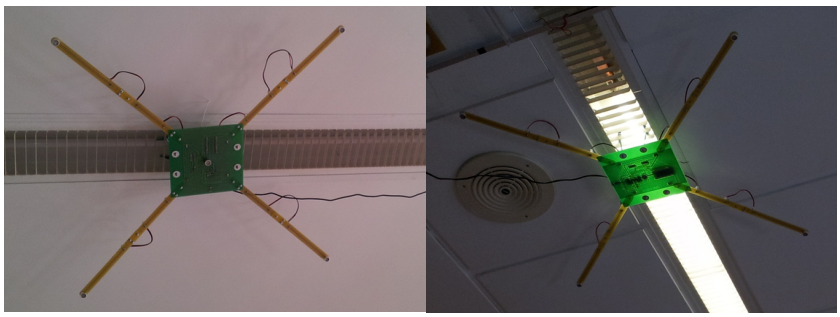


Figura 4.4: Despliegue de dos LPS.

Éste se trata de un sistema de cinco balizas ultrasónicas cuya señal de entrada son la salidas, amplificadas en potencia, de los DACs (4.5). Además, como se ha mencionado anteriormente, el LPS se puede

[3] http://www.digilentinc.com/Data/Products/PMOD-DA2/PMod%20DA2_rm.pdf

configurar como esclavo o como maestro de tal forma que si es maestro emita periódicamente, tal y como se haya configurado, y si es esclavo emite cuando recibe el pulso de inicio de conversión a través de un módulo radio instalado. Este módulo se conecta a la placa de desarrollo Genesys a través de la entrada digital de uno de los PMOD (señal de sincronismo en la figura 4.5).

4.6. Diagrama general

Como punto final se presenta en forma de diagrama la plataforma propuesta en la figura 4.5.

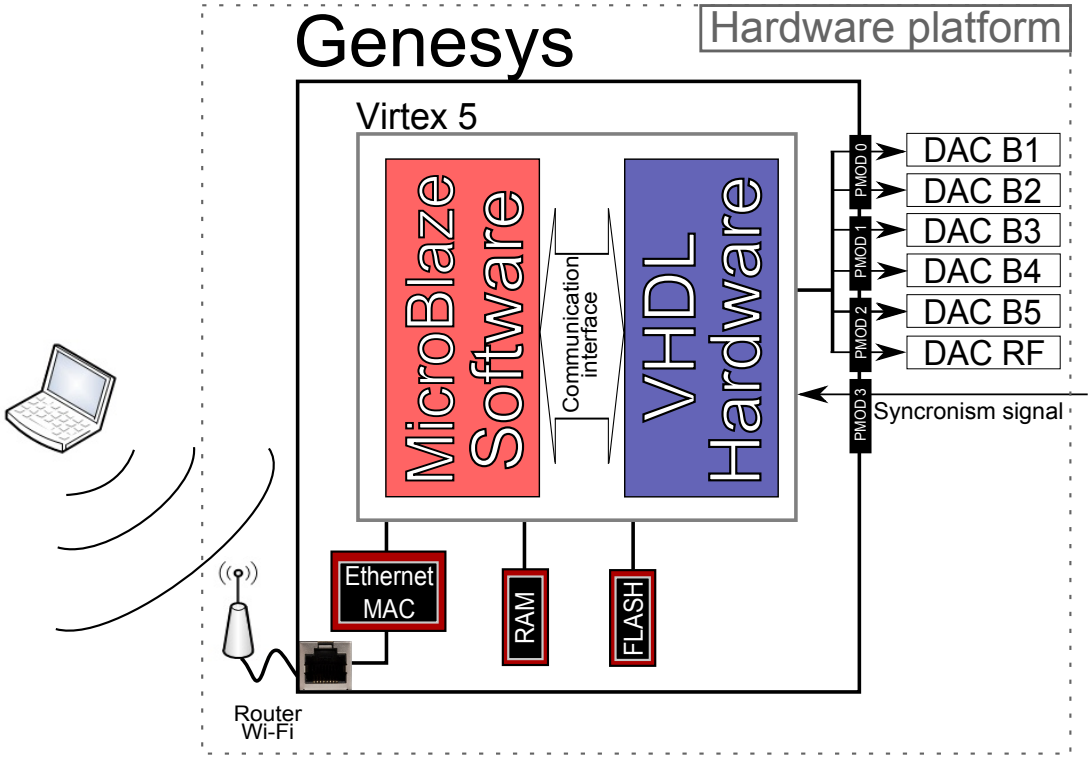


Figura 4.5: Plataforma propuesta.

De izquierda a derecha se presenta el flujo lógico del sistema con todos los módulos que se han ido comentando en los apartados anteriores y la interconexión entre ellos. En primer lugar se muestra el PC (subapartado 4.1) que mediante conexión Wi-Fi manda los datos al receptor (subapartado 4.2), éste mediante la entrada Ethernet de la tarjeta Genesys (figura 4.2) se conecta con el controlador MAC de Ethernet que a su vez está conectado a la Virtex-5. El control de la lectura y escritura es realizado por el MibroBlaze que posee un periférico brindado por Xilinx para dirigir el controlador de la MAC de Ethernet, los datos recibidos se pasan tanto al periférico realizado en FPGA-fabric como a la memoria FLASH. Por último, el periférico realizado sobre la FPGA-fabric es el encargado de controlar los DACs basándose en los parámetros establecidos por el MicroBlaze. Como punto importante cabe destacar que la plataforma hardware puede ser configurada como maestro o como esclavo, de tal manera que si es maestro, la emisión se realizará de manera periódica dando además el inicio de la emisión, si es esclavo de modo que el DAC etiquetado como DAC RF en la figura 4.5 no se utilizaría y el inicio de emisión se llevaría a cabo con un flanco de subida producido en la señal de sincronía. Además, el sistema debe ser capaz de guardar los parámetros de configuración, de tal forma que si se apaga y se vuelve a encender, la plataforma siga funcionando con la misma configuración.

Capítulo 5

Implementación

Para optimizar el diseño e implementación del sistema propuesto en el capítulo 4, es apropiada una exploración de las especificaciones con el fin de elegir una arquitectura y un reparto de tareas adecuado. Con tal fin, en primer lugar se listan las distintas funcionalidades que el sistema debe cumplir:

1. Interfaz con el usuario con el fin de recoger parámetros de configuración.
2. Formación de paquetes a enviar a partir de los parámetros recogidos.
3. Enviar los parámetros vía Wi-Fi.
4. Recepción de los parámetros vía Wi-Fi.
5. Control del chip externo de la MAC de Ethernet.
6. Control de memoria memoria FLASH.
7. Modulación de una serie de códigos mediante las moduladoras y los parámetros, recibidos vía Wi-Fi.
8. Control de cinco DACs y lectura de una entrada digital.

Los tres primeros puntos se han llevado a cabo en un PC con conexión Wi-Fi mediante el programa Matlab, la recepción de los parámetros vía Wi-Fi lo realizará el módulo presentado en la figura 4.1 y el resto de puntos se llevan a cabo en la tarjeta Genesys (4.2).

5.1. Plataforma hardware

Las tareas realizadas por las plataforma hardware, puntos 4 a 8 del apartado anterior, se han subdividido en dos apartados. Las tareas implementadas mediante VHDL en FPGA fabric (5.1.1.1), y las tareas llevadas a cabo mediante software, realizadas en lenguaje C/C++ sobre el MicroBlaze (5.1.1.2). Además se ha aprovechado que Xilinx proporciona una serie de controladores de periféricos de la tarjeta Genesys para realizar los puntos 5 y 6, para los puntos 7 y 8 se realizará un periférico propio sobre la FPGA fabric.

5.1.1. System on Chip

El primer paso es la generación del SoC sobre la Virtex-5 y para ello se usa la herramienta XPS (*Xilinx Platform Studio*) de Xilinx. Inicialmente se necesita especificar los parámetros del MicroBlaze y los distintos periféricos que va a tener el diseño. En la figura 5.1 se muestran las diferentes opciones configuradas:

- Plataforma Digilent de Genesys, puede ser una plataforma de Xilinx o incluso una plataforma de creación propia.
- Sistema de un sólo procesador, la herramienta permite la configuración de un sistema con un coprocesador.
- Configuración del procesador, elegido MicroBlaze con una frecuencia de trabajo de 100MHz y 64kb de memoria local. En otras plataformas se puede instanciar un tipo de microprocesador distinto como un PicoBlaze, o en algunas FPGAs se pueden instanciar versiones de hardcores como el ARM de la FPGA Zynq.
- Periféricos, en este punto se especifican los diferentes periféricos que facilita Xilinx ya implementados. Se mantienen activados los controladores de memoria SDRAM, memoria FLASH, capa MAC de Ethernet, RS232 UART, buses de memoria local (dlmb,ilmb), la entrada de pulsadores e interruptores y los LEDs como salida visual.
- MicroBlaze da la posibilidad de activar o no memoria caché tanto de instrucciones como de datos, se trata de una arquitectura Hardvard y por lo tanto se pueden activar independientemente. En este caso se activan ambas con una memoria de 4kb.

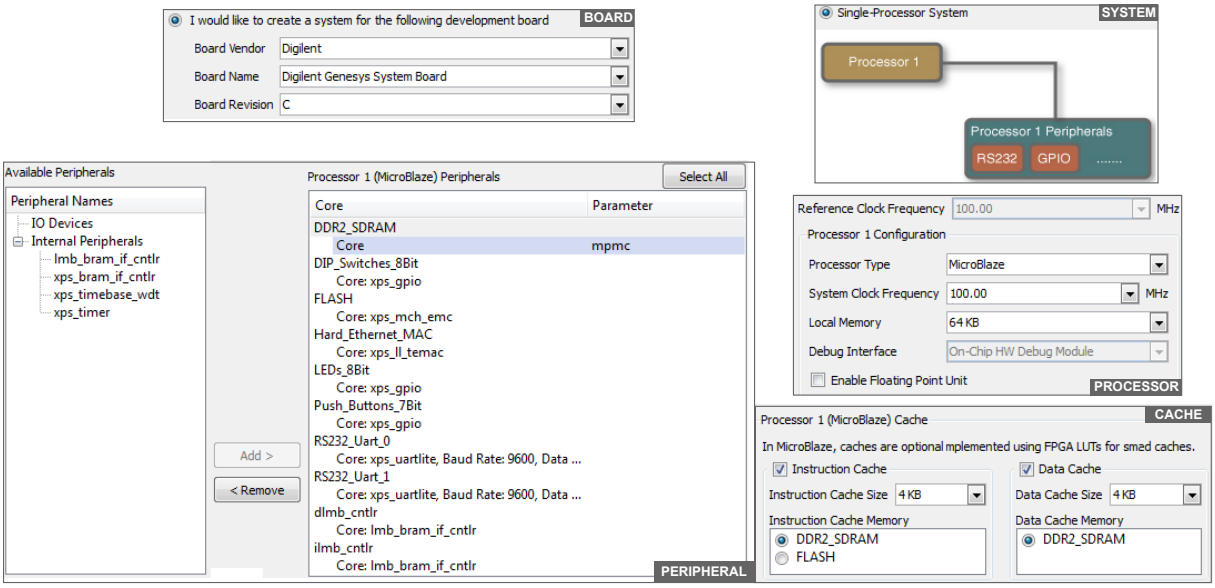


Figura 5.1: Creación del SoC - Especificación del sistema.

Una vez realizado la especificación del sistema se presenta una interfaz de buses como la mostrada en la figura 5.2-[A], donde se muestra el bus de comunicación PLB, al que están conectados los controladores de la capa MAC Ethernet, de la memoria FLASH, del periférico propio y del MicroBlaze, además de los controladores necesarios para la gestión de la memoria RAM, la memoria de datos y la memoria de programa que corre en el MicroBlaze, también los controladores del bus PLB y periféricos de control de LEDs, interruptores y pulsadores que se han usado para la depuración del código. La especificación de la conexión de las líneas de los distintos periféricos se realiza en la ventana mostrada en la figura 5.2-[B]. Hay que detallar las conexiones internas entre ellos y si se quiere acceder a algún pin de la FPGA hay que marcarlo como externo. La asociación del nombre del pin marcado como externo y el pin físico se realiza en el archivo *system.ucf*, donde también se marcan todas las restricciones de sintetización del circuito.

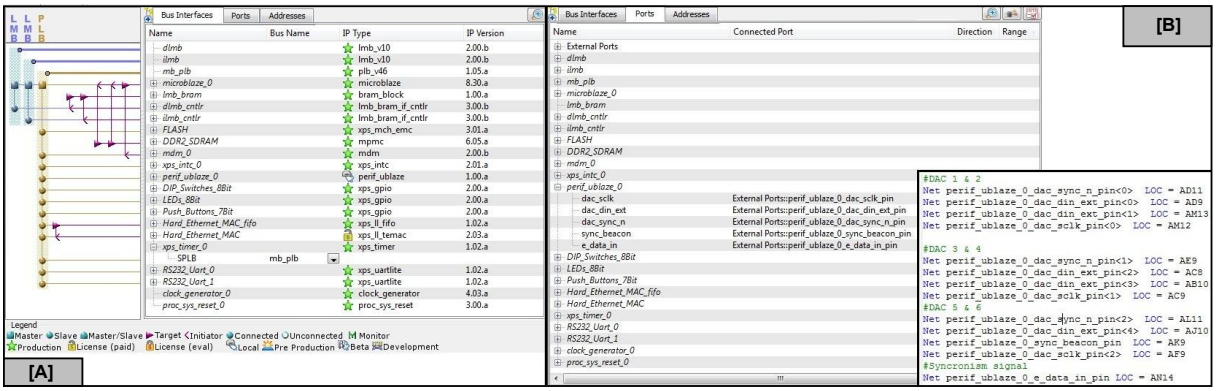
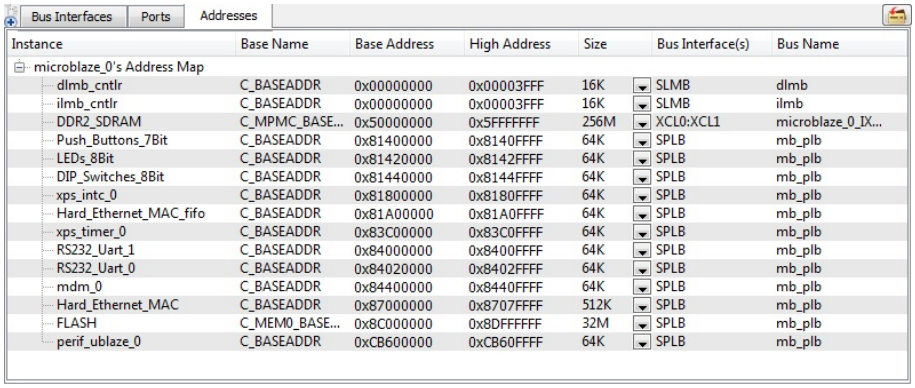


Figura 5.2: Creación del SoC - [A] Conexionado a bus PLB [B] Conexionado de puertos y captura parcial del archivo *system.ucf*.

Por último hay que realizar el mapeo en memoria de los periféricos, esto se realiza dentro del campo *Addresses* mostrado en la ventana que aparece en la figura 5.3. En este punto se asocia una dirección base y una ocupación de memoria para cada uno de los periféricos que se han añadido al diseño.



Instance	Base Name	Base Address	High Address	Size	Bus Interface(s)	Bus Name
microblaze_0's Address Map						
dlmb_cntlr	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB	dlmb
ilmb_cntlr	C_BASEADDR	0x00000000	0x00003FFF	16K	SLMB	ilmb
DDR2_SDRAM	C_MPMC_BASE...	0x50000000	0x5FFFFFFF	256M	XCL0:XCL1	microblaze_0_IX...
Push_Buttons_7Bit	C_BASEADDR	0x81400000	0x8140FFFF	64K	SPLB	mb_plb
LEDs_8Bit	C_BASEADDR	0x81420000	0x8142FFFF	64K	SPLB	mb_plb
DIP_Switches_8Bit	C_BASEADDR	0x81440000	0x8144FFFF	64K	SPLB	mb_plb
xps_intc_0	C_BASEADDR	0x81800000	0x8180FFFF	64K	SPLB	mb_plb
Hard_Ethernet_MAC_fifo	C_BASEADDR	0x81A00000	0x81A0FFFF	64K	SPLB	mb_plb
xps_timer_0	C_BASEADDR	0x83C00000	0x83C0FFFF	64K	SPLB	mb_plb
RS232_Uart_1	C_BASEADDR	0x84000000	0x8400FFFF	64K	SPLB	mb_plb
RS232_Uart_0	C_BASEADDR	0x84020000	0x8402FFFF	64K	SPLB	mb_plb
mdm_0	C_BASEADDR	0x84400000	0x8440FFFF	64K	SPLB	mb_plb
Hard_Ethernet_MAC	C_BASEADDR	0x87000000	0x8707FFFF	512K	SPLB	mb_plb
FLASH	C_MEM0_BASE...	0x8C000000	0x8DFFFFFF	32M	SPLB	mb_plb
perif_ublaze_0	C_BASEADDR	0xCB600000	0xCB60FFFF	64K	SPLB	mb_plb

Figura 5.3: Creación del SoC - Direcciones.

Una vez se han realizado todos los pasos anteriores se exportará la plataforma creada al SDK (*Software Development Kit*) de Xilinx, donde se desarrollará la aplicación software sobre el MicroBlaze (subsección 5.1.1.2), con los periféricos facilitados por la herramienta de creación y el periférico propio añadido (subsección 5.1.1.1).

5.1.1.1. Periférico propio en FPGA fabric

El periférico es el encargado de dar a las distintas balizas que componen el sistema de posicionamiento local (*LPS*) los códigos modulados y parametrizados a través de cinco conversores digitales-analógicos (*DAC*). Por último también se encargará de la señal digital proporcionada por el módulo RF, cuando el LPS sea configurado como esclavo. La figura 5.4 muestra la idea de la funcionalidad que debe tener el periférico para realizar la tarea de creación de los códigos modulados y parametrizados. A través del bus PLB (subsección 5.1.1) se realiza la escritura en los registros de configuración (figuras5.5 a 5.9). Mediante estos registros también se realiza la escritura en la memoria del periférico, dicha memoria tiene

tres bancos donde se guarda la señal moduladora, los códigos y las modulaciones en amplitud [I, Q] respectivamente. Según los valores que tengan los distintos parámetros se modifica el esquema funcional mostrado en la figura 5.4 para poder realizar las modulaciones de fase y/o amplitud BPSK, QPSK, APK y QAM (explicadas en la sección 3.2).

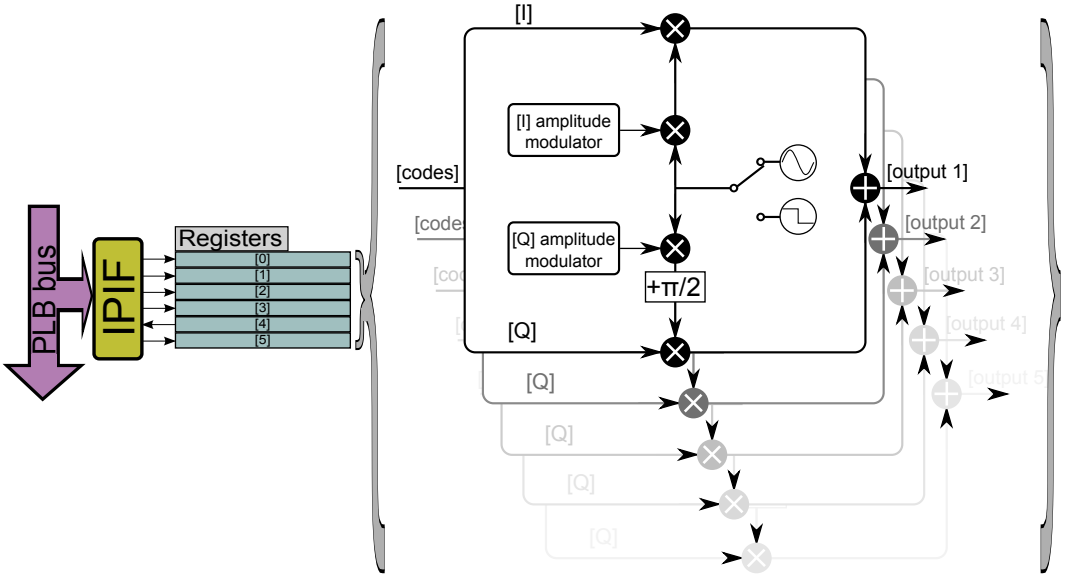


Figura 5.4: Funcionalidad del diseño.

Los parámetros configurables así como las señales de control de memoria, y los datos de códigos y moduladoras, están distribuidos en los registros de comunicación con el MicroBlaze de la siguiente forma:

- Registro 0: contiene parámetros necesarios por el periférico para la escritura y posterior lectura de los códigos de memoria RAM, y los datos necesarios para la modulación. La distribución de datos en el registro 0 se muestra en la figura 5.5:
  - Tamaño del código a descargar en la memoria RAM, señal *size_cod* cuyo rango de valores está en 0-4095. Necesario para saber cuando se termina la escritura de códigos en la memoria RAM y no permitir escribir más una vez se haya realizado la total escritura, así como a la hora de leer el código saber cuando se ha llegado al final.
  - Tamaño de la señal moduladora a descargar en memoria RAM, señal *size_mod* cuyo rango de valores está entre 0-127. Este dato es necesario para saber cuando se termina la escritura de la señal moduladora en la memoria RAM y no permitir escribir más una vez se haya terminado el número de escrituras máximas, así como a la hora de leer dicha señal saber cuándo se ha llegado al final de la señal moduladora.
  - Marcado de inicio de nueva escritura, señal *init_system* que puede tomar los valores 1 ó 0 y el inicio lo marca el flanco de subida producido en la transición del nivel 0 a 1.
  - Frecuencia de emisión, señal *frec_conf*. El inicio de conversión de un dato por parte de los DACs puede realizarse a tres frecuencias distintas; 400kHz, 416.66kHz y 500kHz que corresponden con los valores de configuración 0, 1 y 2 respectivamente.
  - Tiempo de espera entre emisiones de código de la misma baliza, señal *time_gap* cuyo rango de valores está entre 0-31. Cada incremento de un valor en dicha señal significará un incremento de 5ms en el tiempo de espera, por lo tanto el tiempo entre emisiones puede ir desde 0 a 155ms.

- Habilitar modulación en amplitud, señal *amp_act* que puede tomar los valores 1 ó 0 para habilitado o no respectivamente. Si está habilitada la modulación en amplitud, se utilizan los multiplicadores necesarios para realizar las multiplicaciones en amplitud.
- Valor máximo de la amplitud de los códigos (en caso de que sea una modulación en amplitud), señal *quantifier* cuyo rango de valores está entre 0-15. Se usa para el cálculo de una tensión de *offset* necesaria en las modulaciones de amplitud que se explicará en el bloque decodificador.

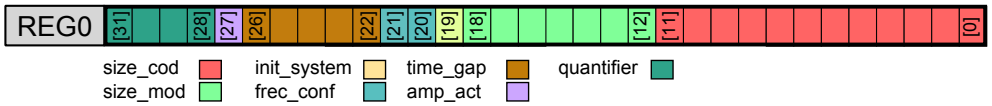


Figura 5.5: Distribución de datos en los registros - Registro 0.

- Registro 1: control de escritura en memoria RAM y datos a escribir en las memorias RAM de códigos y amplitud. La distribución de datos en el registro 1 se muestra en la figura 5.6:
  - Habilitación de escritura en las memorias RAM, señal *we_cod* que puede tomar los valores 1 ó 0 para habilitar escritura o no respectivamente.
  - Señalización de nuevo dato a grabar en la memoria RAM presente en el periférico, señal *new_data* que puede tomar los valores 1 ó 0 y el flanco de subida en la transición del nivel 0 a 1, marca el inicio de una nueva escritura en memoria.
  - Código, señales *RAM1*, *RAM2*, *RAM3*, *RAM4* y *RAM5* que pueden tomar un valor en el rango 0-3 cada una y son las encargadas de las selecciones de las señales a emitir por cada código y por cada una de las balizas.
  - Amplitud del código, señales *AMP_RAM1*, *AMP_RAM2*, *AMP_RAM3* y *AMP_RAM4* que pueden tomar un valor en el rango 0-15 cada una. En caso de que sea una modulación en amplitud, éste valor multiplicará el valor de la moduladora.

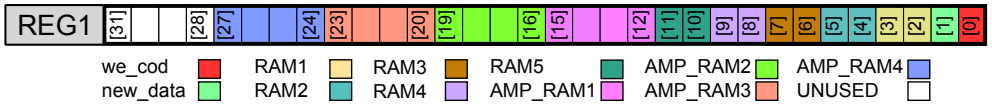


Figura 5.6: Distribución de datos en los registros - Registro 1.

- Registro 2 y 3, contiene los valores de escritura en memoria RAM de las señales moduladoras. La distribución de datos de los registros 2 y 3 se muestra en la figura 5.7:
  - Moduladora, señales *MOD_RAM1*, *MOD_RAM2*, *MOD_RAM3* y *MOD_RAM4* que pueden tomar un valor en el rango 0-4095 cada una y se trata de las señales que realizan la modulación de los códigos.

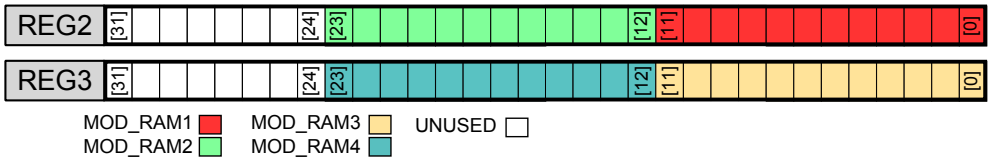


Figura 5.7: Distribución de datos en los registros - Registros 2 y 3.

- Registro 4, se trata del único registro donde el periférico puede escribir para comunicarse con el MicroBlaze y se usa para comprobar la correcta escritura de los datos en el periférico. La distribución de datos del registro 4 se muestra en la figura 5.8:
  - Mostrar que la escritura demandada ha sido realizada correctamente, esto se realiza por la señal *ack_ublaze* cuyos valores son: 1 cuando la escritura se ha realizado o 0 en estado de reposo. Se vuelve al estado de reposo cuando la señal *new_data* del registro 1 (5.6) tiene el valor 0, ésta a su vez se espera que vuelva al valor 0 después de que se active la señal *ack_ublaze*.
  - Mostrar la finalización de la escritura en el periférico, señal *long_cod_r* muestra con un 1 cuando el proceso de escritura ha escrito el total de valores de código esperados, 0 en estado de reposo. Vuelve al estado de reposo cuando la señal *we_cod* toma el valor 0. Si la señal *long_cod_r* se encuentra todavía activa no se permite una nueva escritura, por lo que para iniciar una nueva escritura, al menos se debe generar un nuevo flanco de subida en la señal *we_cod*.

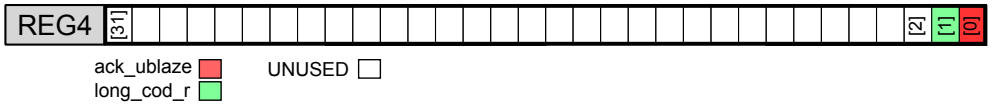


Figura 5.8: Distribución de datos en los registros - Registro 4.

- Registro 5, contiene los últimos parámetros para la configuración de la emisión y de la modulación. La distribución de dichos datos en el registro 5 se muestra en la figura 5.9:
  - Habilitar modulación QAM, señal *qam_act* que puede tomar los valores 1 ó 0 para habilitado o no respectivamente. Si este bit se encuentra habilitado significa que la modulación a realizar es QAM.
  - Configuración de la plataforma como maestro o como esclavo, señal *master_slave* que puede tomar los valores 1 ó 0 para maestro y esclavo respectivamente. La diferencia radica en que la configuración como maestro realiza una emisión periódica y envía al emisor RF un pulso cuadrado de duración 1ms cada vez que se realice una emisión. Por el contrario, la configuración como esclavo realiza una emisión cada vez que se recibe un flanco de subida en la señal de entrada digital y deja el sexto DAC, encargado de enviar la señal de sincronismo, sin uso.

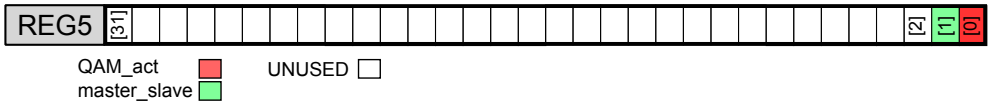


Figura 5.9: Distribución de datos en los registros - Registro 5.

Una vez comentados los parámetros de configuración del periférico así como su función y forma de uso, se explican los distintos bloques que conforman el sistema que va a realizar la funcionalidad mostrada en la figura 5.4:

Gestor de registros: el primer bloque de conexionado es el gestor de registros, mostrado en la figura 5.10. Éste bloque es el encargado de separar los parámetros guardados en los registros mostrados en las figuras 5.5, 5.6, 5.7 y 5.8, en las señales correspondientes comentadas en dichas figuras para poder repartirlos por los distintos elementos que necesiten dichas señales. Pero además, a partir de la comunicación con

el MicroBlaze, el gestor de registros gestiona el control de la escritura en el el bloque de memorias RAM mediante las señales *we_data*, *addr_data*, *din_data*, *clk_data*, *din_ampl*, *we_cod*, *addr_cod*, *din_mod* y *clk_mod*. Cabe destacar que las señales *AMP_RAM1*, *AMP_RAM2*, *AMP_RAM3* y *AMP_RAM4* están colocadas en ese orden en la señal *din_ampl*, *MOD_RAM1*, *MOD_RAM2*, *MOD_RAM3* y *MOD_RAM4* en la señal *din_mod* y *RAM1*, *RAM2*, *RAM3*, *RAM4* y *RAM5* en la señal *din_data*.

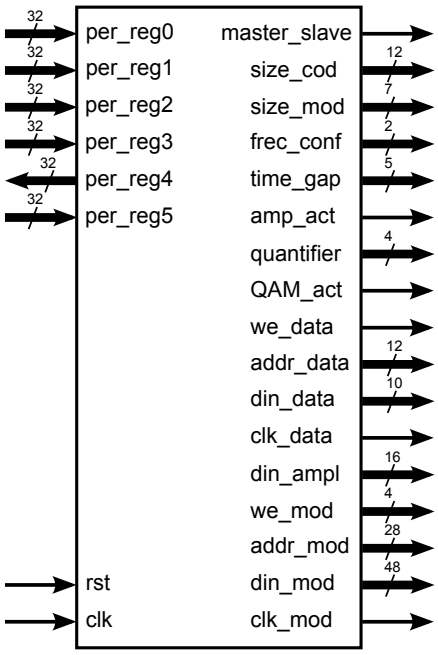


Figura 5.10: reg_manager.

Por claridad se separan en las figuras 5.11 a 5.12 las simulaciones del gestor de registros. En la figura 5.11 se muestra la primera funcionalidad comentada, se trata de la asignación de los valores de los registros a su correspondiente señal de configuración.

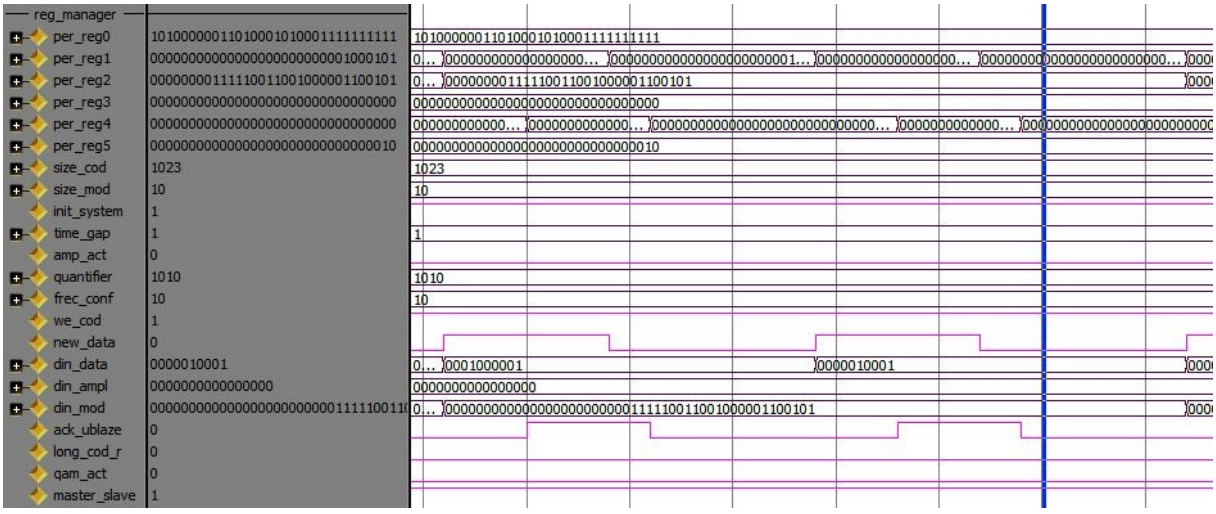


Figura 5.11: Simulación del gestor de registros - Simulación 1.

La figura 5.12 muestra la simulación del control de las direcciones de escritura, la habilitación de la escritura en memoria RAM y la orden de escritura en las memorias RAM asociadas a códigos, amplitud

y señal moduladora. Los datos de salida hacia dicho bloque son asociaciones directas mostradas en la figura 5.11.

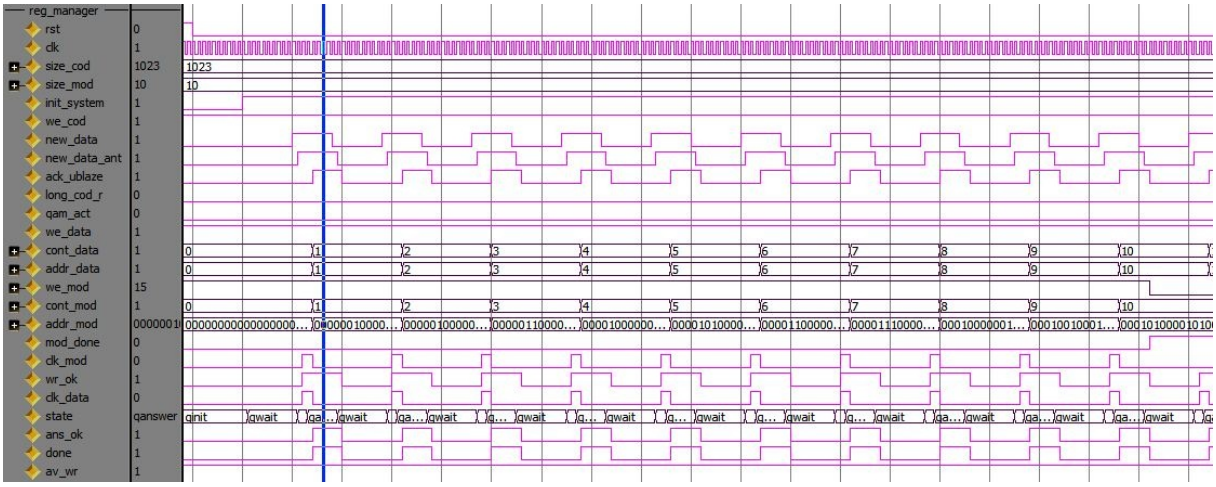


Figura 5.12: Simulación del gestor de registros - Simulación 2.

Para el control de la escritura que representan las señales de la figura 5.12 se implementa el diagrama de estados mostrado en la figura 5.13.

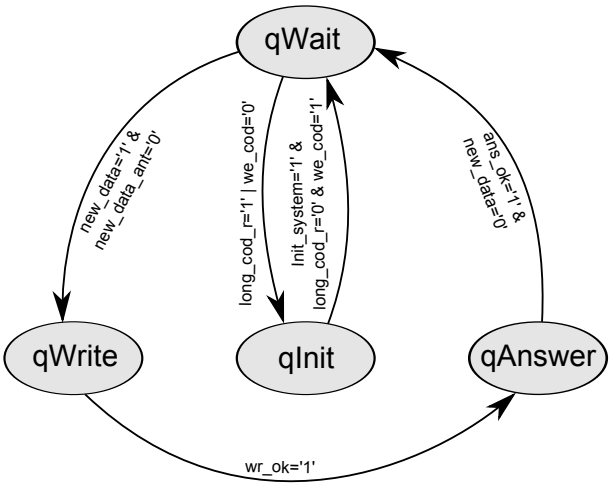


Figura 5.13: Diagrama de estados de control de escritura en memoria RAM.

- El estado de reposo es *qInit* en el cual se inicializan todos los valores de control. Con el fin de salir de dicho estado hay que marcar la intención de inicio de un nuevo proceso de escritura, para ello se activan las señales *init_system*, *we_cod* y hay que comprobar que la petición de escritura no haya sido atendida con anterioridad, esto lo marca la señal *long_cod_r*.
- *qWait* es el estado de espera de un nuevo dato. Éste deja las señales con los valores de reposo y vigila la llegada de los contadores que controlan las direcciones de escritura a su máximo valor, tanto de memoria de códigos como de la moduladora. Los valores máximos vienen dados por las señales *size_cod* y *size_mod*, que marcan el tamaño del código y de la moduladora respectivamente. Cuando los contadores de control de direcciones alcanzan dichos valores máximos, se activan las señales *long_cod_r* y *mod_done* respectivamente. El hecho de que se active la señal

mod_done implica que no se escribe más en la memoria RAM de la moduladora y la activación de la señal *long_cod_r* conlleva que se ha finalizado la escritura en las memorias RAM. Que la señal *long_cod_r* esté activa desencadena el cambio al estado de reposo *qInit*, en caso contrario una vez se haya recibido un nuevo dato, marcado por un flanco de subida en la señal *new_data*, el siguiente estado es *qWrite*.

- El estado *qWrite* se encarga de realizar la escritura de los datos en memoria RAM creando un flanco en las señales *clk_data* y *clk_mod*. En el caso de *clk_mod* sólo se produce el flanco de escritura si la señal *mod_done* no ha sido activada, en caso contrario es que se ha llegado al final de la escritura de los datos de la moduladora y no se han de escribir más datos. Una vez hecho esto se pasa automáticamente al estado *qAnswer*.
- El estado *qAnswer* incrementa la dirección de escritura, dejándola preparada para el nuevo dato y genera un pulso en la señal *ack_ublaze* como respuesta al sistema externo (al cual le llega a través del registro 4, ver figura 5.8) indicando una correcta escritura en la memoria RAM. Una vez realizada la respuesta, cuando el sistema externo recibe la señal activada debe responder poniendo a 0 la señal *new_data* y pasar al estado *qWait*.

Control maestro/esclavo del sistema: si el sistema se configura como esclavo el inicio de una nueva emisión viene marcado por una señal digital externa, este modo de funcionamiento se configura poniendo a 0 el bit *master_slave* del registro 5 (5.9).

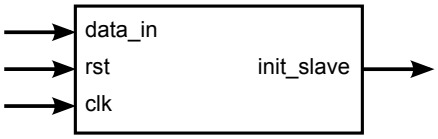


Figura 5.14: masterSlave_Control.

La finalidad de este bloque es la detección de un flanco de subida en la señal asíncrona digital *e_data_in*. Para que no se produzcan errores de detección por falta de tiempo es importante la sincronización de la señal. La señal sincronizada es *e_data_in_s*, y a partir de esta se crea un pulso de un ciclo de reloj en la señal *init_slave* que marca el inicio de una nueva emisión.

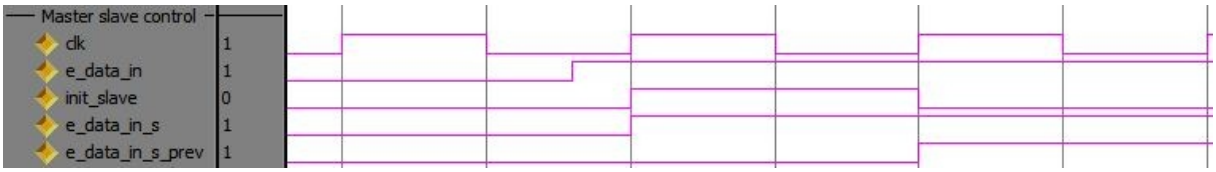


Figura 5.15: Simulación de inicio de emisión en configuración como esclavo.

Inicio de nueva conversión: el sistema se puede configurar con tres frecuencias de emisión distintas: 400kHz, 416.66kHz y 500kHz. La elección de una de esas frecuencias es controlada por la señal *frec_conf* configurada en el registro 0 (5.5).

El módulo consta de un contador cuyo fin de cuenta se cambia con un multiplexor controlado por la señal *frec_conf*. Con un valor "10" se obtiene la señal de 500kHz, con "01" se obtiene 416.66kHz y con un "00" se obtiene 400kHz. El número de cuentas a realizar para conseguir un pulso cada 2.5 μs, 2.4 μs o 2 μs con un reloj de plataforma cuyo periodo es 10 ns, son 250, 240 y 200 respectivamente. Cuando

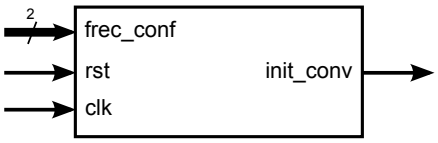


Figura 5.16: gen_init_conv.

el contador se ha incrementado hasta el número de cuentas establecido por cada frecuencia, se crea en la señal *init_conv* un pulso de duración un ciclo de reloj. La figura 5.17 muestra los resultados para lo antes comentado y con las medidas de tiempo mostradas por los cursores.

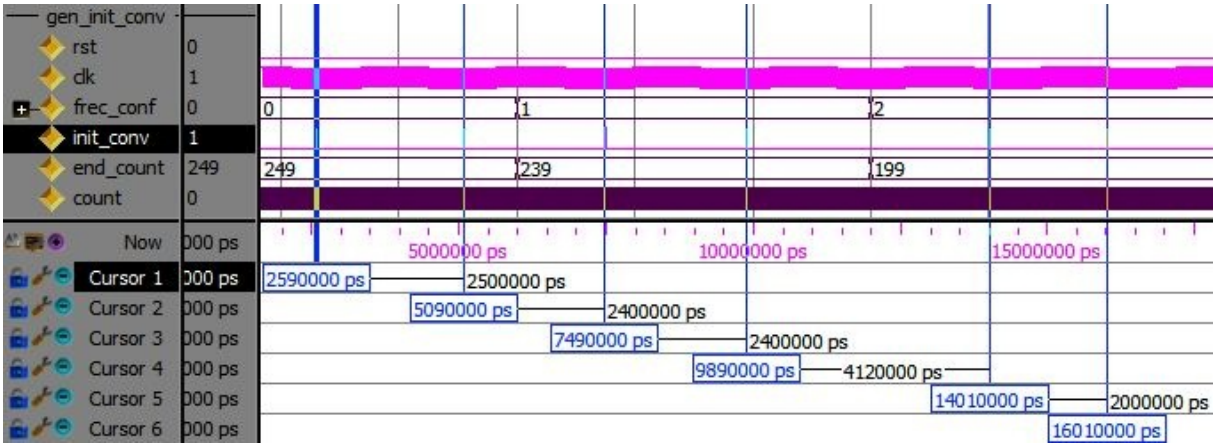


Figura 5.17: Simulación de pulso de inicio de nueva conversión.

Control de bloque de memorias RAM: La funcionalidad principal de este bloque, mostrado en la figura 5.18, es el control de la lectura de las distintas memorias RAM en función de los parámetros del sistema introducidos en los registros 0 y 5 de configuración (5.5 y 5.9).

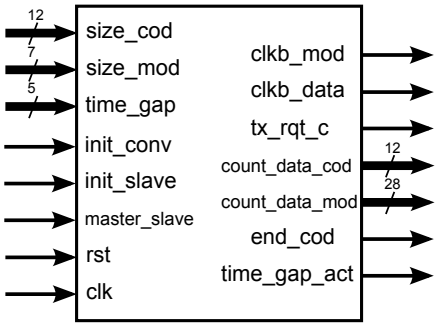


Figura 5.18: RAM_control.

Existen tres bloques de memoria a controlar que son: la memoria de códigos y de amplitud comparten las señales de control *count_data_cod* que selecciona la dirección de lectura y *clkb_data* que ordena una lectura en la dirección correspondiente. Las señales de control de la memoria de la señal moduladora son *count_data_mod* y *clkb_mod* con la misma funcionalidad que las anteriores señales. Además se pueden observar las señales de salida *end_cod*, *time_gap_act* y *tx_rqt_c*, que marcan el final del código

leído, si existe separación de emisión entre códigos y realización de conversión por nuevo dato listo respectivamente.

La figura 5.19 muestra la máquina de estados implementada para la realización del proceso de lectura:

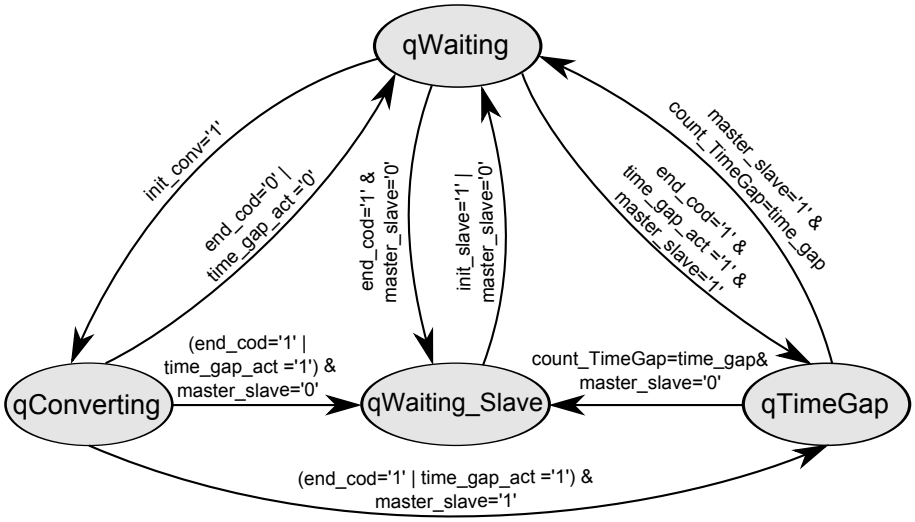


Figura 5.19: Diagrama de estados de control de lectura de memoria RAM.

- El estado *qWaiting* es el estado de espera y no se modifican los valores de las señales. Con la recepción de un pulso en la señal de inicio de conversión *init_conv*, si no se ha acabado el código a leer, se pasa al estado *qConverting*. En caso de que se haya acabado el código, si la configuración es de esclavo, se pasa al estado *qWaiting_Slave*, y si el estado es de maestro y además está activa la pausa de emisión entre códigos se pasa al estado *qTimeGap*. En cualquier otro caso se mantiene el estado *qWaiting*.

- El estado *qConverting* es el encargado de realizar el incremento de las direcciones de lectura mediante las señales *count_c* para código y *count_m* para la señal moduladora (ambas asignadas posteriormente a las señales de salida *count_data_cod* y *count_data_mod* respectivamente). Este estado también marca el inicio de lectura mediante las señales *clkb_mod* y *clkb_data* y además produce la señal *tx_rqt_c* para señalar que hay un nuevo dato. Si no se ha terminado el código o la pausa entre emisión de códigos está desactivada se pasa al estado *qWaiting*, en caso contrario si la configuración es de maestro se pasa al estado *qTimeGap* y si es de esclavo al estado *qWaiting_Slave*.

- El estado *qTimeGap* es el encargado de gestionar las pausas entre emisiones en caso de que estén activadas. Se trata de dos contadores, un primer contado cuya señal de cuenta es *count_5ms* crea un pulso cada cinco milisegundos y comienza la cuenta con la entrada en el estado *qTimeGap*, y un segundo contador cuya señal de cuenta es *count_time_gap* que incrementa su valor con cada pulso generado en la señal *count_5ms*, de tal forma que cuando el valor de la señal *count_time_gap* y la señal *time_gap* tengan el mismo valor, indica que se ha llegado al fin de la pausa y se pasa al estado *qWaiting*. En caso de que durante la pausa se cambie la configuración de maestro a esclavo, automáticamente se pasa al estado *qWaiting_Slave*.

- El estado *qWaiting_Slave* es un segundo estado de espera al que sólo se llega si el sistema está configurado como esclavo, en este estado se detienen las emisiones. Para salir de este estado se necesita comunicar a este bloque que se solicita una emisión, esto se produce cuando se recibe

un pulso en la señal *init_slave*. También se puede cambiar la configuración del sistema a modo maestro para cambiar de estado, en cualquiera de las dos situaciones se pasa al estado *qWaiting*.

En la figura 5.20 se muestra el direccionamiento y control de lectura de las últimas direcciones en modo maestro para comprobar el funcionamiento de los estados *qWaiting*, *qConverting* y *qTimeGap*. Completando la explicación de los estados, se puede observar que para cada código se leen todos los datos de la moduladora de la señal *size_mod* y que cuando se ha completado la lectura de todos los códigos se pasa al estado *qTimeGap* y se activa la señal que indica que se ha alcanzado el final del código *end_cod*.

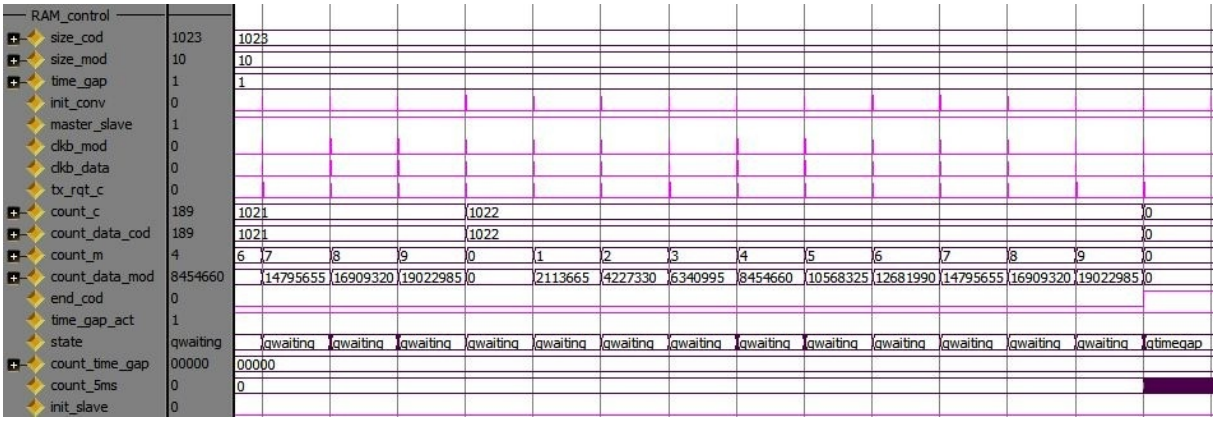


Figura 5.20: Simulación del control de lectura de memoria RAM - Simulación 1.

En la figura 5.21 se muestra el fin de el espaciado entre emisiones y por tanto el comienzo de un nuevo ciclo de lectura. Además se puede observar que mientras el sistema está configurado como maestro, no se producen nuevas emisiones con el flanco de subida de la señal de inicio de emisión de esclavo *init_slave*. Una vez que el sistema cambia su configuración a esclavo (señal *master_slave* = '0') se pasa el estado de espera *qwaiting_slave* una vez que se ha terminado la presente lectura y espera un pulso en la señal *init_slave*, cuando ésta se produce se comienza un nuevo ciclo de lectura de las memorias RAM.

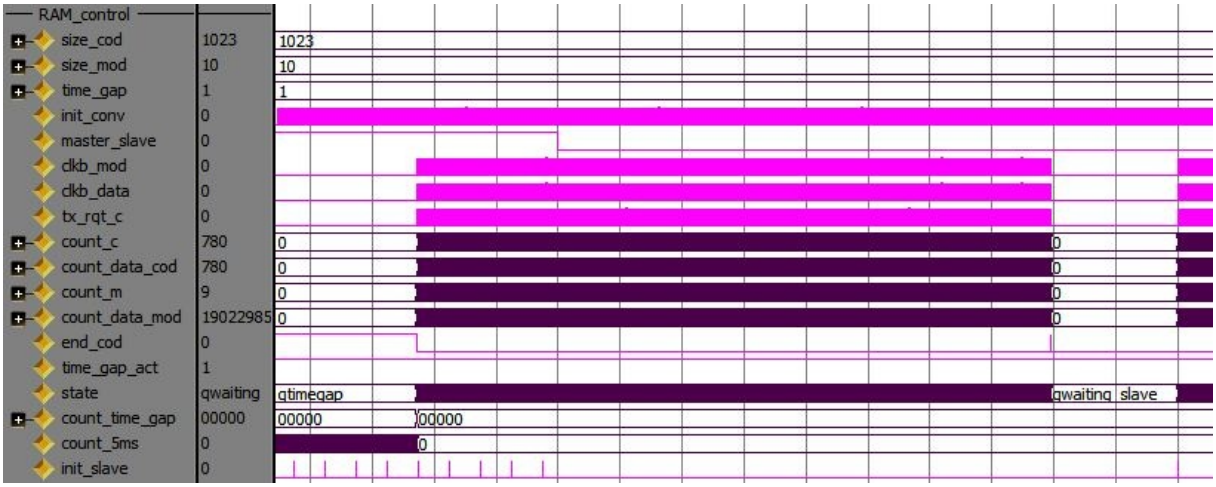


Figura 5.21: Simulación del control de lectura de memoria RAM - Simulación 2.

Bloque de memorias RAM: el bloque mostrado en la figura 5.22 muestra todas las memorias RAM usadas en el diseño. Este bloque consta de las siguientes memorias configuradas *simple dual-port* :

- Cuatro memorias RAM que almacenan las moduladoras. Éstas son las referenciadas como *RAM_mod0*, *RAM_mod1*, *RAM_mod2* y *RAM_mod3*, todas ellas tienen un tamaño de 128 x 12 bits.
- Una memoria RAM con los códigos, referenciada como *RAM_cod* y dimensionada a 4k x 10 bits.
- Una memoria RAM con las amplitudes de las moduladoras, referenciada como *RAM_ampl* y con la misma dimensión que la memoria RAM que contiene los códigos.

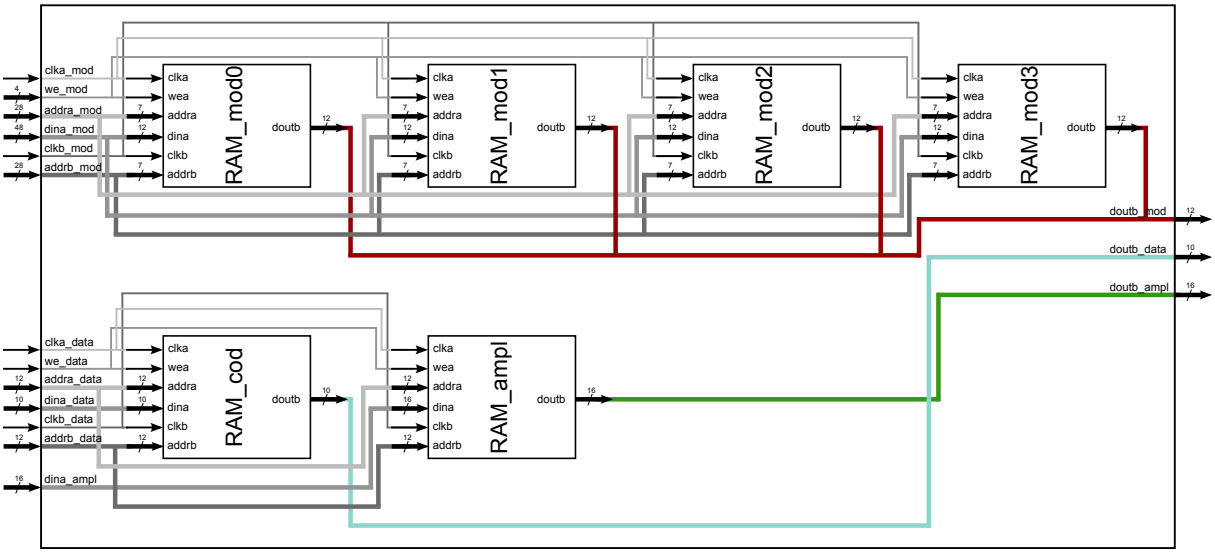


Figura 5.22: RAM_memory.

Las memorias RAM constan de dos puertos independientes; un puerto A para la escritura controlado por las señales *clka_X*, *we_X*, *addra_X* y *dina_Y* (donde, X es *mod* y *data*, e Y es *mod*, *data* y *ampl*) y un puerto B controlado por las señales *clkb_X*, *addrb_X* y *doutb_Y* (donde, X es *mod* y *data*, e Y es *mod*, *data* y *ampl*). En la figura 5.23 se puede observar la independencia entre los procesos de escritura y lectura. En una primera parte se escriben los datos de todas las RAM hasta que se finaliza la escritura de la moduladora y ésta queda desactivada. En la segunda parte se termina la escritura de las demás memorias.

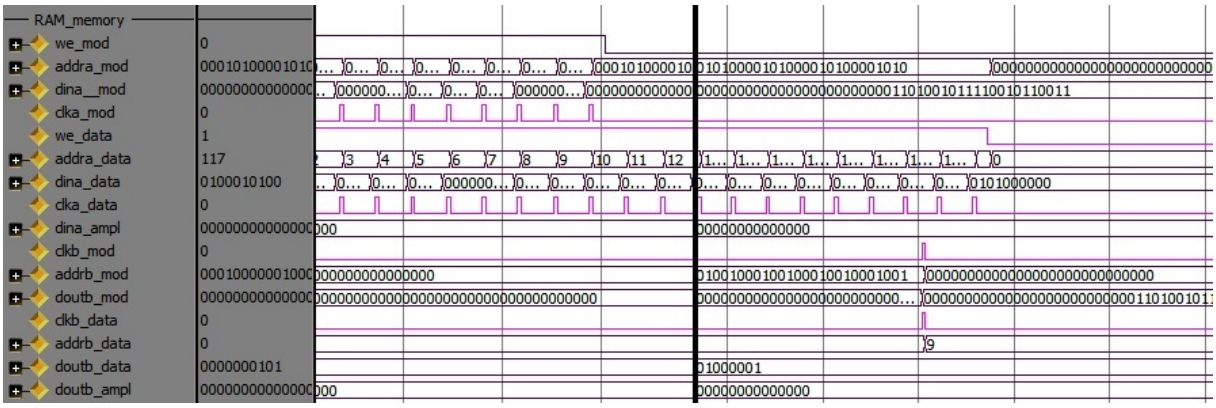


Figura 5.23: Simulación del bloque memorias RAM.

Bloque de multiplicación: se muestra en la figura 5.24 el bloque encargado de multiplicar las señales de amplitud de los códigos y las moduladoras en caso que se haya configurado una emisión con modulación en amplitud, APK o QAM, en cualquier otro caso la salida es la misma que la entrada.

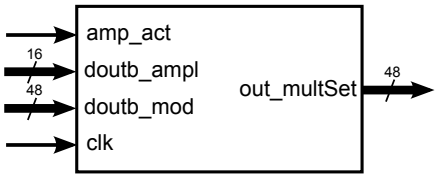


Figura 5.24: multiplier_set.

Como ejemplo en la figura 5.25 se muestra la salida de este bloque para una señal APK cuyo valor máximo de amplitud es cuatro, donde el valor mostrado entre corchetes es el valor de la amplitud asociada a cada moduladora y ésta depende del código. En este caso, de todas las posibles moduladoras, la que se almacena en memoria es la que se debe multiplicar por la unidad (en el ejemplo de la figura 5.25, la referenciada como [1]), ésta situación se tiene que prever a la hora de guardar las distintas moduladoras en la memoria RAM asociada.

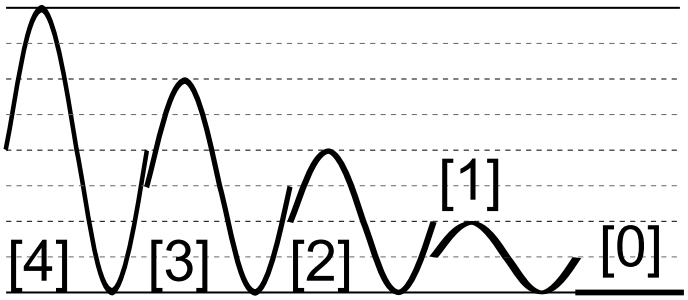


Figura 5.25: Salida del bloque de multiplicación.

Decodificador: este bloque es el encargado de generar la señal que ha de ser convertida por los DACs a partir de todos los datos leídos de las memorias RAM. La figura 5.26 muestra la distribución y anchos de puertos del bloque.

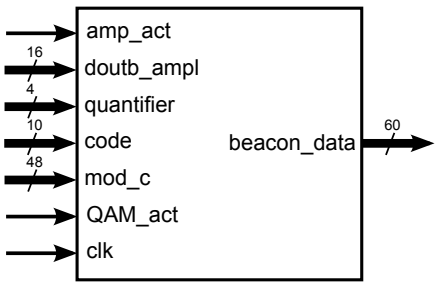


Figura 5.26: decoder.

Para mayor comprensión del bloque se presenta la figura 5.27, donde se puede observar los diferentes subelementos que lo componen: un bloque creador de *offset* (entidad vhdl *offset_creator*), unos sumadores (realizado mediante suma y asignación de señales directamente) y unos decodificadores (generados con procesos en código vhdl dentro del bloque principal).

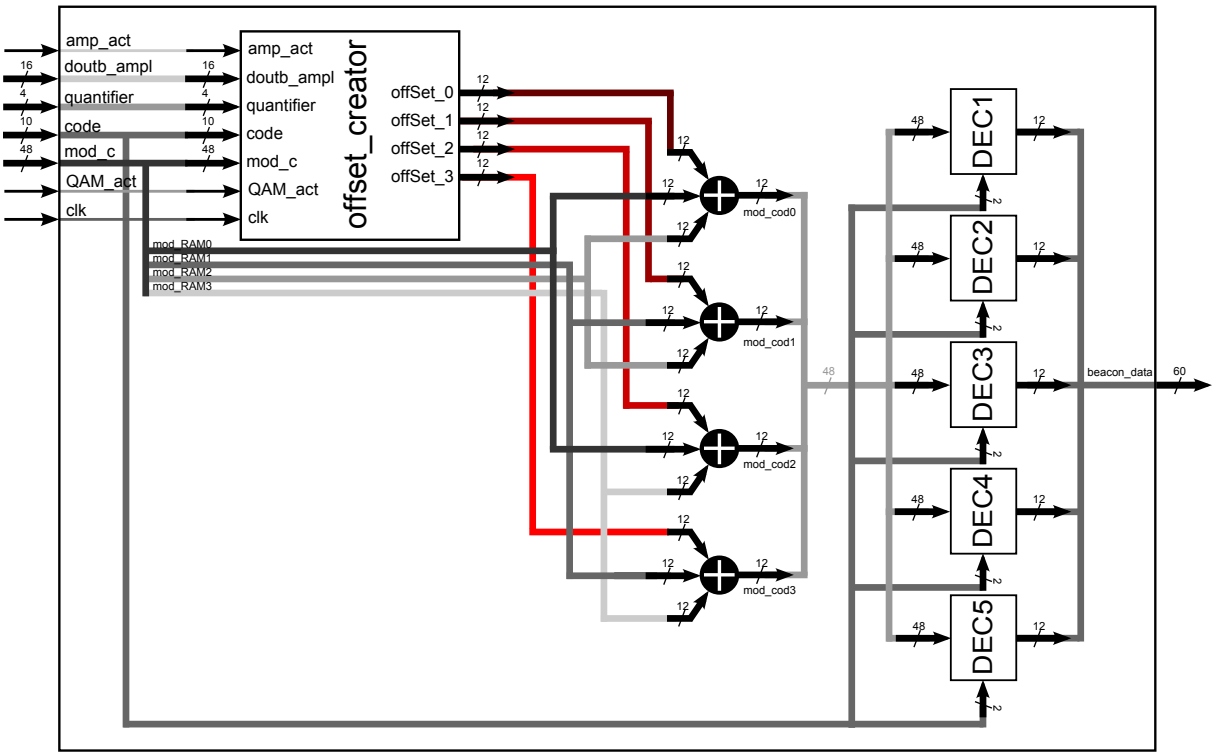


Figura 5.27: Esquema funcional del bloque *decoder*.

Debido a la forma en la que se almacenan las señales moduladoras en las memorias RAM, cuando se configura el sistema con una modulación en amplitud, es necesario introducir un *offset* en la señal para centrar la salida en la mitad del *SPAN*. Para el caso concreto de una señal senoidal, modulación APK y con un cuantificador de amplitud de 4, lo que se encuentra en *mod_RAM0*, *mod_RAM1*, *mod_RAM2*, y *mod_RAM3* (que es la salida del bloque *multiplier_set*) son unas señales sinusoidales como las mostradas en la figura 5.25, dependiendo de la amplitud tomará un valor u otro. La salida correcta ha de estar centrada en al mitad del SPAN, es decir ha de ser una señal como la mostrada en la figura 5.28. Para las distintas amplitudes asociadas a cada código se necesita un offset distinto y por tanto ha de ser calculado para cada caso.

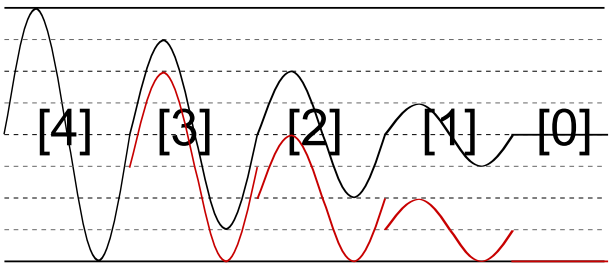


Figura 5.28: Explicación de la necesidad del bloque de generación *offset*.

Con ese propósito se introduce el módulo *offset_creator*, que implementa las ecuaciones 5.1 y 5.2 en función de si es una modulación APK o una modulación QAM respectivamente, en cualquier otro caso las salidas de offset son nulas. Para evitar realizar divisiones, ya que estas son difícilmente implementables, se ha limitado el valor de la variable quantifier entre valores comprendidos entre 2 y 10, de esta forma es posible precalcular y almacenar en memoria los valores resultantes de la división. Por tanto, por

imposición de diseño, el valor *quantifier* queda comprendido en el rango [2..10] y además SPAN = 4095, por tanto los valores $\frac{SPAN/2}{quantifier}$ y $\frac{SPAN/4}{quantifier}$ pueden ser precalculados y pregrabados, de tal manera que cambiando el valor con un multiplexor controlado por la señal *quantifier* se tenga el valor deseado. Cabe destacar que la multiplicación se lleva a cabo con un multiplicador instanciado de la FPGA y que las señales *amplitude_X* (con X={1,2,3,4}) corresponden con los valores de la memoria de amplitud *RAM_mod* comentados en la figura 5.22 y que son subseñales de la señal *doutb_ampl*.

$$
\begin{aligned}
offSet_0 &= \frac{SPAN/2}{quantifier} \cdot (quantifier - amplitude_0) \\
offSet_1 &= \frac{SPAN/2}{quantifier} \cdot (quantifier - amplitude_1) \\
offSet_2 &= \frac{SPAN/2}{quantifier} \cdot (quantifier - amplitude_0) \\
offSet_3 &= \frac{SPAN/2}{quantifier} \cdot (quantifier - amplitude_1)
\end{aligned} \tag{5.1}
$$

$$
\begin{aligned}
offSet_0 &= \frac{SPAN/4}{quantifier} \cdot (2 \cdot quantifier - (amplitude_0 + amplitude_3)) \\
offSet_1 &= \frac{SPAN/4}{quantifier} \cdot (2 \cdot quantifier - (amplitude_1 + amplitude_3)) \\
offSet_2 &= \frac{SPAN/4}{quantifier} \cdot (2 \cdot quantifier - (amplitude_0 + amplitude_4)) \\
offSet_3 &= \frac{SPAN/4}{quantifier} \cdot (2 \cdot quantifier - (amplitude_1 + amplitude_4))
\end{aligned} \tag{5.2}
$$

La función de los sumadores de la figura 5.27 depende del tipo de modulación. Así para una modulación QPSK es la de sumar dos moduladoras, para una modulación APK es la de sumar una moduladora con el *offset*, para una modulación QAM es la de sumar dos moduladoras y *offset* y para una modulación BPSK el sumador deja pasar directamente la sumadora. Esto se implementa directamente con la suma de los tres parámetros y anulando cada uno de los valores por separado en el resto del diseño. Por último queda, en la misma figura 5.27, un bloque de decodificadores realizados por un multiplexor 4 a 2 donde la señal de control son los códigos guardados en la señal *code*, donde hay un código de dos bits por cada salida, y las señales de entrada son señales como las mostradas con color negro en la figura 5.28 guardadas en las señales mod_codX (con X={1,2,3,4}).

En la figura 5.29 se muestra la simulación del decodificador con una modulación APK que tiene un valor máximo en amplitud de 3 y moduladora senoidal, ya que hay señales redundantes debido a que tienen el mismo funcionamiento son excluidas de la figura por claridad.

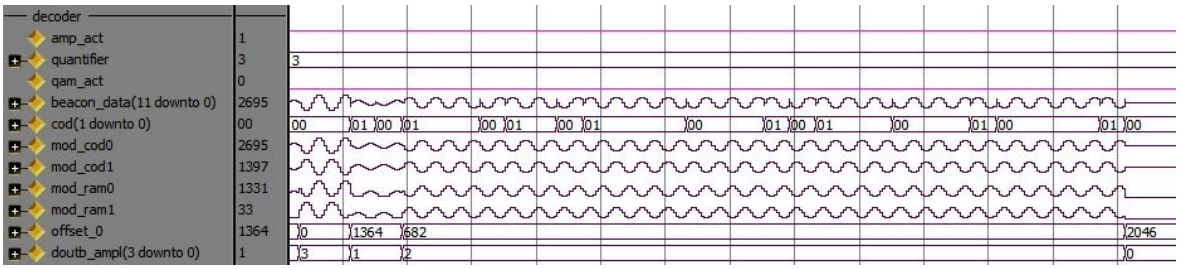


Figura 5.29: Simulación del bloque decodificador con señal APK.

Puesto que se trata de una modulación APK las señales *mod_ram2* y *mod_ram3* tienen valor nulo y las señales mod_cod2 y mod_cod3 no son usadas. Se pueden observar tres amplitudes, y el valor nulo, de las señales moduladores *mod_ram0* y *mod_ram1* como indica el valor de la señal *quantifier*. Dichas señales tienen la forma mostrada en la figura 5.25, las señales con el *offset* añadido son las señales *mod_cod0* y *mod_cod1* con la forma mostrada en la figura 5.28. Por último se muestra la subseñal *beacon_data(11 downto 0)* de salida que muestra la modulación controlada con la subseñal *cod(1 downto 0)*, la cual multiplexa entre las señales *mod_cod0* y *mod_cod1* en función del código de entrada. En la figura 5.30 se muestra una simulación similar pero para un caso de modulación QAM.

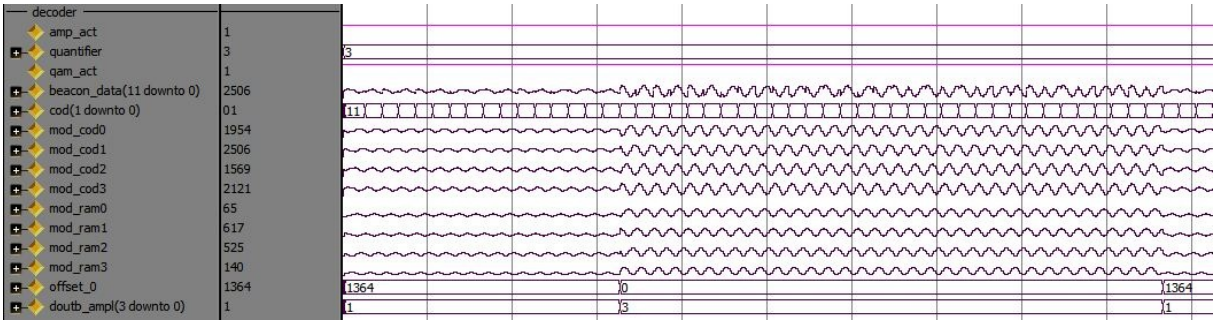


Figura 5.30: Simulación del bloque decodificador con señal QAM.

Multiplexor: su función es multiplexar entre el valor proporcionado por el bloque *decoder* y un valor de $SPAN/2$ para cuando el sistema tiene que estar en reposo, es decir, sin emisión de códigos.

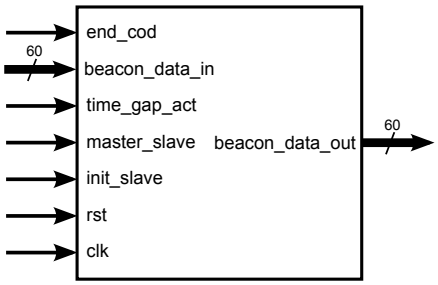


Figura 5.31: multiplexer.

Las ocasiones en las que se debe mantener la salida a un valor de $SPAN/2$ es en la espera entre emisiones de código, este instante está controlado por la señal *end_cod*, y en caso de que el sistema esté configurado como esclavo y no se haya iniciado una emisión, controlados por las señales *end_cod*, *master_slave* e *init_slave*, en cualquier otro caso la señal de salida *beacon_data_out* es igual a la señal de entrada *beacon_data_in*. La figura 5.32 muestra los dos estados necesarios cuando el sistema es esclavo.

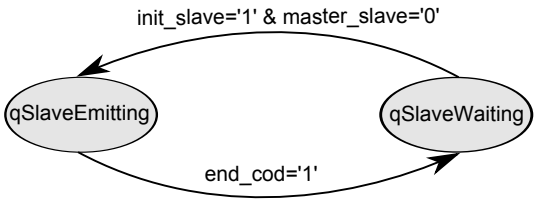


Figura 5.32: Diagrama de estados del multiplexor.

En caso de que el sistema esté configurado como esclavo, el sistema permanece en estado de espera (*qSlaveWaiting*) y la señal de salida se establece en $SPAN/2$. Cuando llega un pulso de inicio de emisión de esclavo, *init_slave = '1'*, se pasa a un segundo estado de emisión (*qSlaveEmitting*) en el que se permanece hasta que el código llega al final, *end_cod = '1'*, en este estado, la señal de salida es igual a la de entrada. La figura 5.33 muestra la simulación del multiplexor en modo esclavo, en modo maestro es similar y no aporta más información. En la señal *beacon_data_in* cuando se acaba el código se queda el último valor que tuvo la señal, pero en la señal de salida interesa que el punto de reposo sea $SPAN/2$ (2048 en el presente caso) y ésta es la salida que se puede observar hasta el inicio de una nueva emisión.

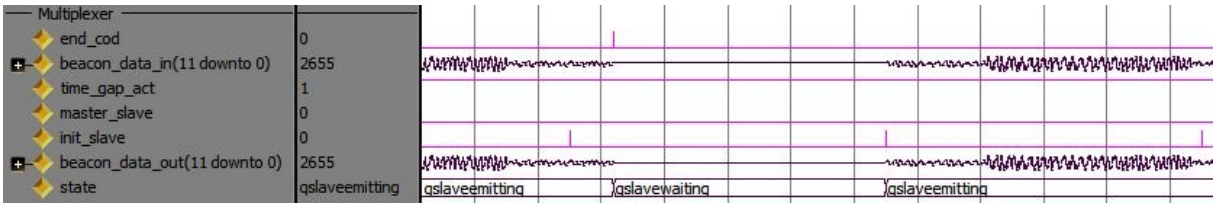


Figura 5.33: Simulación del multiplexor.

Control de los DACs: el control de los DACs lo lleva a cabo el bloque *extended_control_DAC* mostrado en la figura 5.34 y cuyas funcionalidades son: las de controlar cinco DACs para la emisión de los códigos modulados y controlar el DAC encargado de la emisión del pulso de inicio de nueva emisión que además es generado por éste bloque.

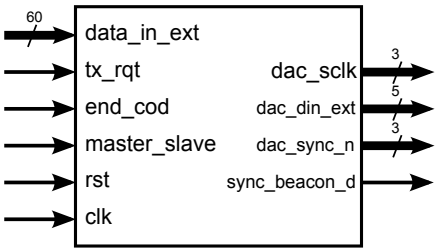


Figura 5.34: extended_control_DAC.

La generación de la señal de nueva emisión (o pulso de sincronía), se controla con las señales externas *master_slave* y *end_cod* y su control se realiza mediante el diagrama mostrado en la figura 5.35. Cuando se detecta un inicio de código, ésto es cuando se produce un flanco de bajada en la señal *enc_cod*, se activa la señal de sincronía y se pasa al estado *qCount* en el que se realiza una temporalización de 1 milisegundo, tras el cual se vuelve al estado *qWait* en el que se desactiva la señal de sincronía y se espera un nuevo flanco en la señal *enc_cod*. Para que el pulso de sincronía se produzca el sistema a de estar configurado como maestro, en caso contrario la salida siempre es nula.

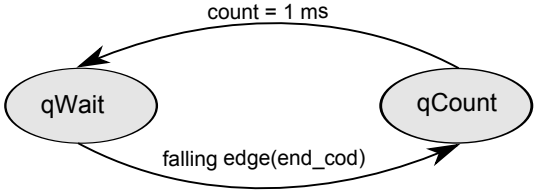


Figura 5.35: Diagrama de estados generación de pulso de sincronía.

Los DAC empleados en el diseño son el modelo DAC121S101 y están integrados en el Pmod-DA2 de Digilent, cada Pmod contiene 2 DACs, como se puede ver en la figura 4.3. El diagrama de tiempos mostrado en la figura 5.36 es el que ha de respetar el controlador para el correcto funcionamiento del módulo. Tras un flanco de bajada de la señal *tsync* se recogen los datos con el flanco de bajada de la señal *sclk*, siendo los primeros cuatro datos un 0 y los doce siguientes los datos a convertir. Por último, cuando se acaba la conversión se deja la señal *sync* a 0 para esperar otro dato. Durante todo este proceso se deben respetar los tiempos marcados por el esquema de funcionamiento.

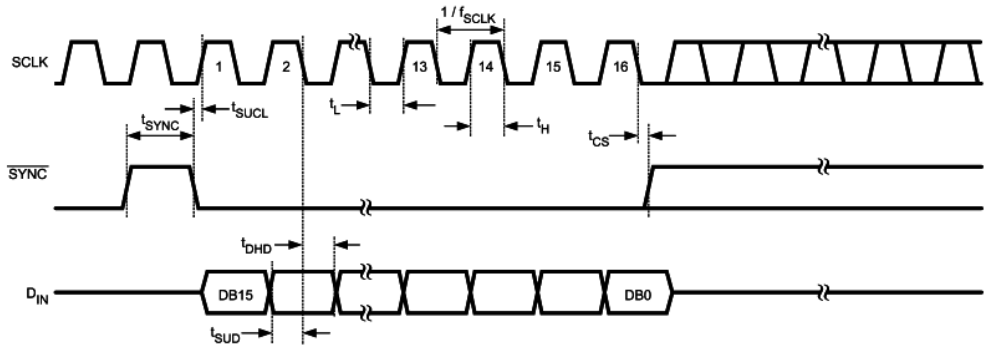


Figura 5.36: Cronograma del DAC.

Para comprobar el correcto funcionamiento de los bloques de control se ha realizado un modelo del DAC con las restricciones temporales de la figura 5.36 y con la distribución de puertos mostrado en la figura 5.37.

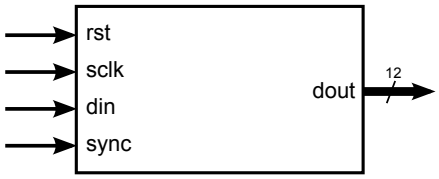


Figura 5.37: DAC.

El simulador tiene dos estados, un estado de reposo, *qInit* en el que se inician los valores de las señales y del cual se sale con un flanco de bajada de la señal *sync* y un estado de conversión, *qConverting* en que se capturan los datos de la entrada serie con cada flanco de bajada de la señal *sclk* hasta llegar a quince, esa cuenta la lleva a cabo un contador y cuando termina la cuenta se activa el bit de fin de conversión y entonces se muestran en la salida los datos paralelizados y se pasa al estado *qInit*.

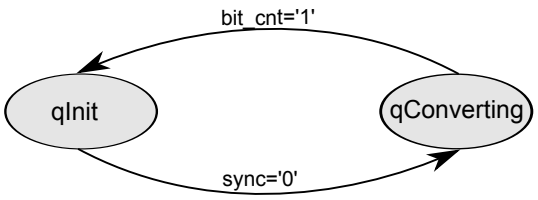


Figura 5.38: Diagrama de estados de conversión del simulador DAC.

Se muestra en la figura 5.39 lo que se explicaba en el párrafo anterior. Cuando la señal *sync* pasa a nivel bajo se desencadena con cada flanco de bajada de la señal *sclk* , la captura de los datos que le llegan

al bloque mediante al línea serie *din* en la señal *data_prev*. Una vez se han capturado 16 bits, el valor capturado se vuelca en la salida *dout*.

Figura 5.39: Simulación del simulador DAC.

Para el control de los DACs externos se introducen seis bloques *DAC_serial_port* como el mostrado en la figura 5.40, cinco para controlar la emisión de los códigos modulados y uno más para el pulso de sincronía.

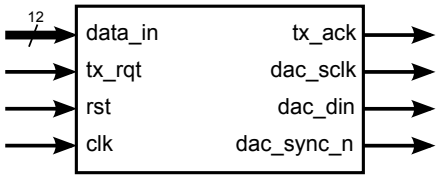


Figura 5.40: DAC_serial_port.

La función principal del bloque *DAC_serial_port* es mandar cuatro ceros iniciales de configuración, para funcionamiento normal del DAC, y serializar un dato paralelo de doce bits de entrada. Cada dato serie enviado se ha de acompañar con un flanco de bajada en la señal *sclk* para que el DAC los capture y además debe marcar el inicio de conversión con la señal *sync*, una vez el contador de datos enviados llegue a cero se ha terminado la conversión. Éste proceso se muestra en la simulación 5.41, donde el dato paralelo que en un inicio está en la entrada, se serializa, se envía y una vez se ha terminado el proceso de conversión aparece el mismo dato en la señal marcada como *DAC*, que es la salida del simulador del DAC y por tanto es el dato que se vería en el pin físico analógico de salida en caso de que se midiera externamente.

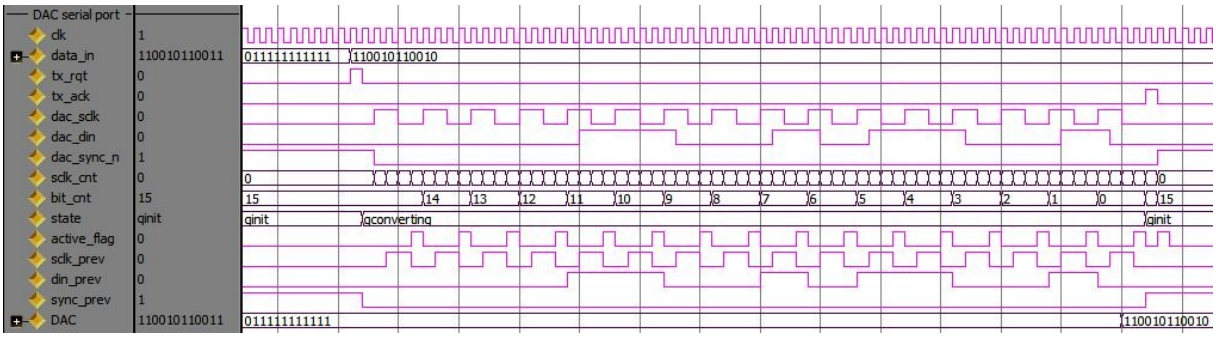


Figura 5.41: Simulación del controlador *DAC_serial_port*.

El controlador del DAC implementa el diagrama de estados mostrado en la figura 5.42. El estado de espera *qInit* inicializa los valores de todas la señales y se pasa de este estado al estado *qConverting* con la petición de una nueva conversión, *tx_rqt = '1'*. En estado *qConverting* se gobiernan las señales de

control de escritura y se serializa el dato de entrada. De este estado se sale cuando el dato se ha enviado correctamente. Esto se produce cuando el contador *bit_cnt* cuenta quince bits y *active_flag='1'*.

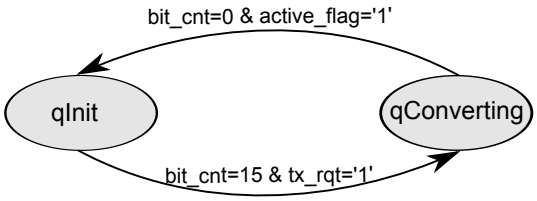


Figura 5.42: Diagrama de estados de controlador del DAC.

Por último se muestra la simulación del bloque completo *extended_control_DAC* con las salida de los distintos controladores conectados a los simuladores de DAC externo en la figura 5.43, de ésta simulación cabe destacar la generación de pulso de sincronía. Tras detectar un flanco de bajada en la señal de fin de espera entre códigos *end_cod* se genera un pulso en la señal *falling_edge*, esto muestra el inicio de una nueva emisión y cambia el estado *qWait* a *qCount* (figura 5.35) y tras un milisegundo deja el sistema en reposo hasta una nueva detección.

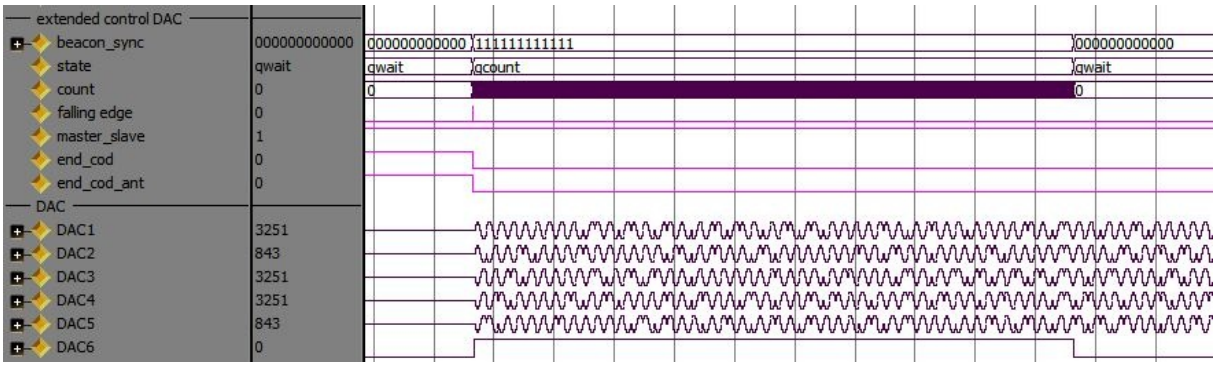


Figura 5.43: Simulación del bloque *extended_control_DAC*.

Sistema global de interconexión: una vez presentados todos los bloques que constituyen el sistema global, se muestran en la figura 5.44 los puertos que forman el periférico que irá instanciado en el SoC.

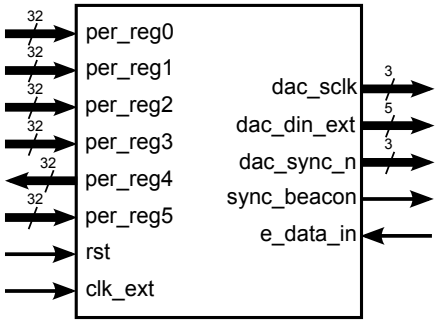


Figura 5.44: beacon_control_peripheral.

Para no perder la visión global del interconexionado de todos los bloques y seguir las simulaciones del periférico expuestas a continuación se muestra la figura 5.51, los bloques aparecen con los puertos

cambiados de lugar con respecto a la explicación individual para simplificar el conexionado y hacer más legible el esquema. Sobre dicho sistema global se realizan una serie de simulaciones para comprobar el correcto funcionamiento de todos los parámetros que se podrán cambiar mediante la entidad externa:

- Modulaciones BPSK, QPSK, APK y QAM. En la figura 5.45 se muestran las cuatro modulaciones soportadas por el diseño.

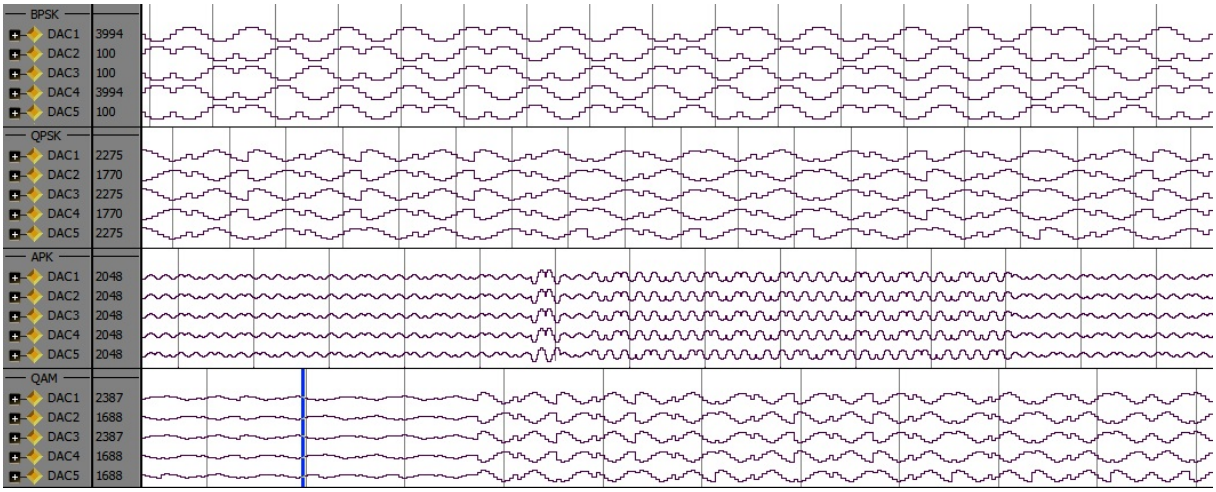


Figura 5.45: Modulaciones.

- Frecuencia de muestro para la generación de señal, las configuraciones disponibles son 400khz, 416.66khz y 500khz (en la figura 5.46 se muestran portadoras con 10 muestras).

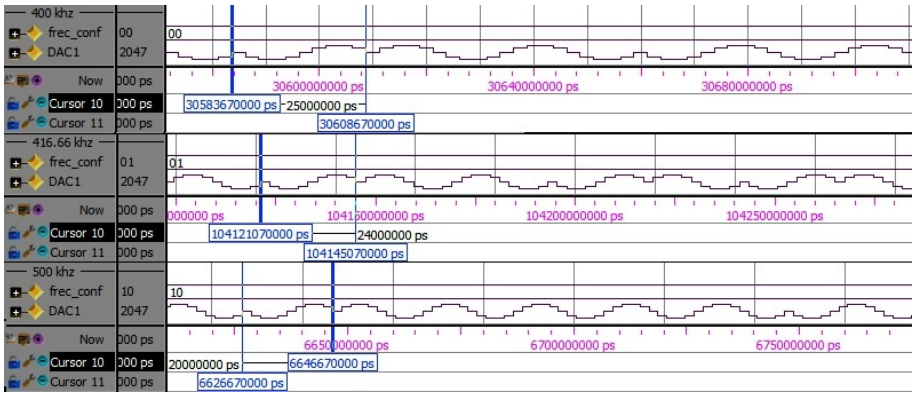


Figura 5.46: Frecuencia de muestreo.

- Diferentes separaciones temporales entre emisiones, se muestra en la figura 5.47 tres separaciones temporales distintas entre emisiones, 5ms, 10ms y 15ms.

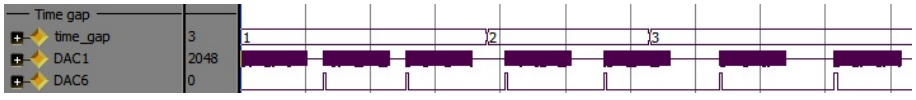


Figura 5.47: Diferentes separaciones temporales entre emisiones.

- Diferente número de muestras por código y diferente número de símbolos por código. En la figura 5.48 se muestran tres distribuciones de los dos parámetros, una primera con 10 muestras y un símbolo por código, una segunda con 25 muestras y tres símbolos por código y una última con 50 muestras y dos símbolos por código.

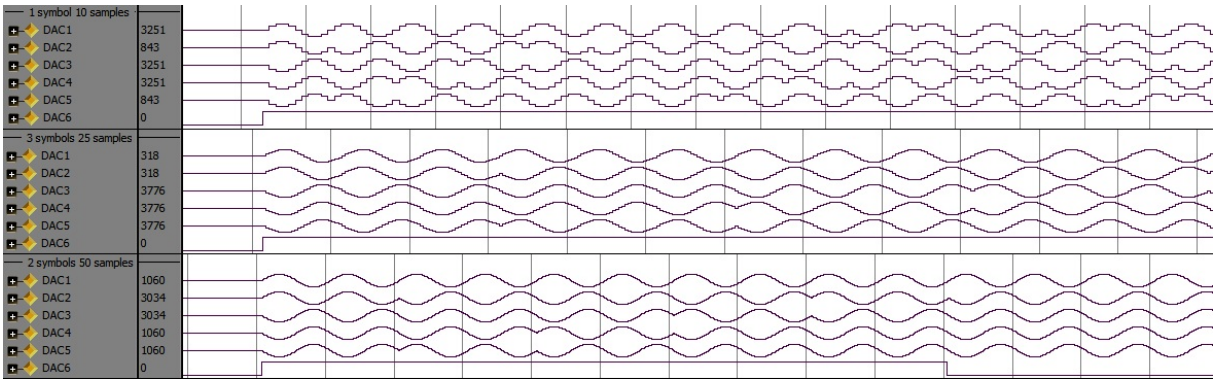


Figura 5.48: Diferentes muestras y símbolos por código.

- Señal moduladora seno o señal moduladora cuadrada, en la figura 5.49 se pueden observar las dos posibilidades.

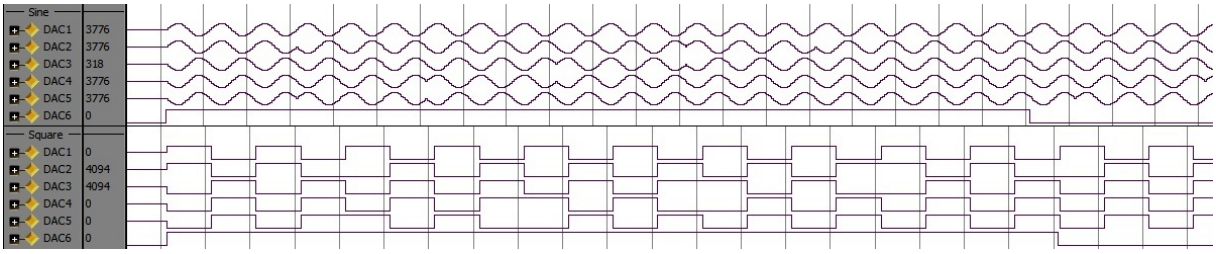


Figura 5.49: Señal moduladora seno y cuadrada.

- Configuración maestro o esclavo. En la figura 5.50 se pueden observar las dos configuraciones, y como cuando el sistema está configurado como esclavo, la emisión comienza con el flanco de subida de la señal *e_data_in* y en el modo maestro la emisión es independiente de dicha señal.

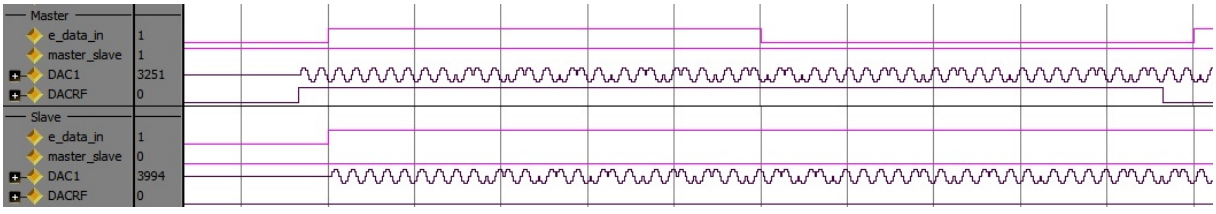


Figura 5.50: Configuraciones maestro y esclavo.

Módulo IPIF: como se muestra en la figura 5.4 existe un bloque de interconexión entre el bus PLB y el periférico creado. IPIF proviene del término anglosajón *IP Interface* y se trata de una plantilla en vhdl dada por Xilinx para facilitar la conexión de cualquier IP al bus PLB. Un bloque de propiedad intelectual, o IP (*Intellectual Property*), es un bloque diseñado y testeado para un proceso específico, como procesadores, interfaces Ethernet . . . y en el presente proyecto es el mostrado en la figura 5.44.

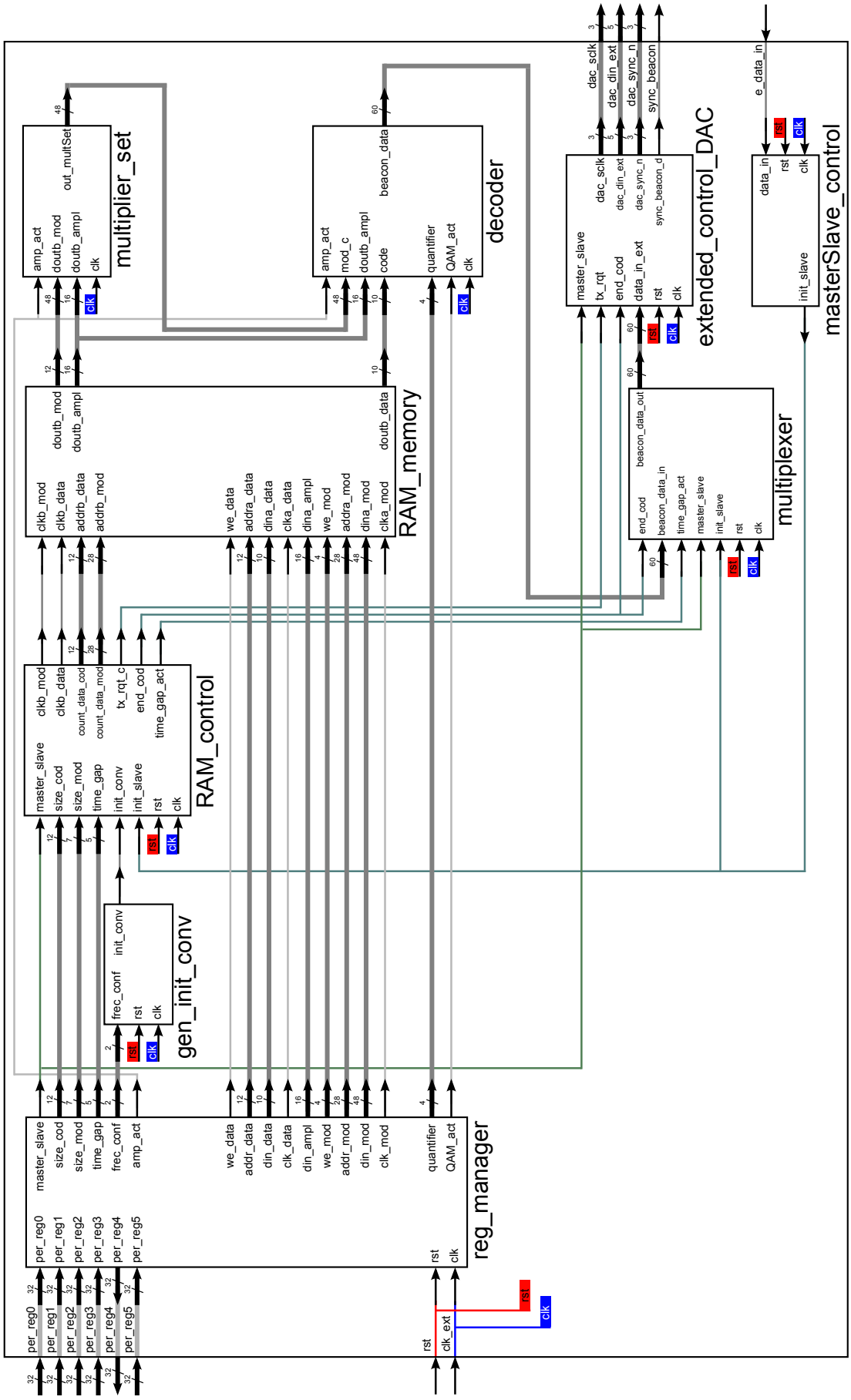


Figura 5.51: Interconexionado del sistema global.

5.1.1.2. MicroBlaze

El MicroBlaze es el encargado del control del chip externo de la MAC de Ethernet, del control de la memoria FLASH externa y además manejará los datos recibidos mediante Ethernet tanto para escribir/-leer de la FLASH, como para escribir en los registros de comunicación con el periférico propio 5.1.1.1.

En la figura 5.52 se refleja el diagrama de flujo del programa principal. Tras el arranque del sistema se realizan una serie de inicializaciones de los distintos periféricos, tras este periodo el programa se queda en un bucle leyendo si ha habido recepción de datos desde la conexión Ethernet.

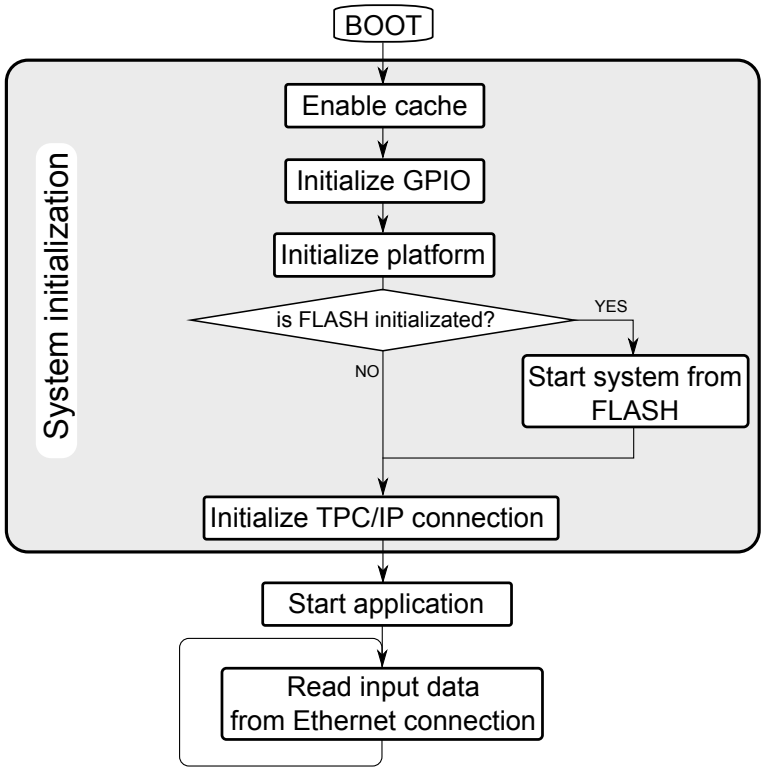


Figura 5.52: Flujograma del hilo principal.

Bloques generados automáticamente por la herramienta SDK de Xilinx: algunos de los bloques presentados en la figura 5.52 son generados automáticamente por la herramienta SDK y se comentan en el presente documento muy brevemente.

- **Habilitar memoria caché de datos y de instrucciones** para una mayor velocidad de ejecución de las instrucciones. Las funciones encargadas de realizar esto son *Xil_ICacheEnable()* y *Xil_DCacheEnable()* respectivamente.
- **Configuración de líneas de entrada/salida genéricas (*GPIO : General Purpose Input/Output)***. Conectadas a interruptores, pulsadores y LEDs, estas son usadas para la depuración de distintos apartados del código y para mostrar al usuario cuando se finaliza la inicialización del sistema. Las funciones encargadas de realizar esto son *XGpio_Initialize()*, *XGpio_SetDataDirection()*, *XGpio_DiscreteWrite()* y *XGpio_DiscreteRead()*, que inicializan, establecen la dirección, escriben y leen respectivamente.

- **Iniciar plataforma**, realiza la inicialización de otros parámetros de MicroBlaze. Realizado por la función *init_platform()*.
- **Iniciar conexión TCP/IP**: se configuran parámetros típicos de la conexión TCP/IP como son la dirección IP, la puerta de entrada (*gateway*) y la máscara de red, estos son establecidos por el programador y en función de las IPs de la red ya ocupadas y del router al que esté conectado. Además se habilitan interrupciones necesarias para la conexión como son: una interrupción periódica por un timer y otra para cuando se detectan nuevos datos de entrada en la conexión. La función *IP4_ADDR()* se encarga de establecer la IP, la puerta de entrada y la máscara de red, las funciones *print_app_header()* y *print_ip_settings()* imprimen por línea serie los tres datos de configuración.
- **Inicio de la aplicación**, con los parámetros establecidos en el punto anterior se crea la conexión TCP/IP que va a tener MicroBlaze en el resto de la aplicación. La función *lwip_init()* se encarga de inicializar la conexión, la función *xemac_add()* añade la interfaz de la red a la lista de conexiones y la función *netif_set_default()* establece la interfaz de red como interfaz de red por defecto.
- **Control de llegada de nuevos datos recibidos desde la conexión:** en un bucle se comprueba constantemente si existen nuevos datos en la memoria FIFO asociada a la conexión Ethernet, además verifica si ha habido alguna corrupción de los parámetros de configuración de la conexión. Todo ello lo realiza la función *xemacif_input()* que está encerrada en un bucle infinito del cual sólo se puede salir cuando se produce un error en el programa.

Ordenación de los datos en memoria FLASH: las funciones facilitadas por la API del SDK hacen accesos a memoria con ancho de un byte, por lo tanto es necesaria una redimensión y reorganización de parámetros para poder guardarlos en la memoria FLASH. En las figuras 5.53 y 5.54 se muestra la configuración elegida.

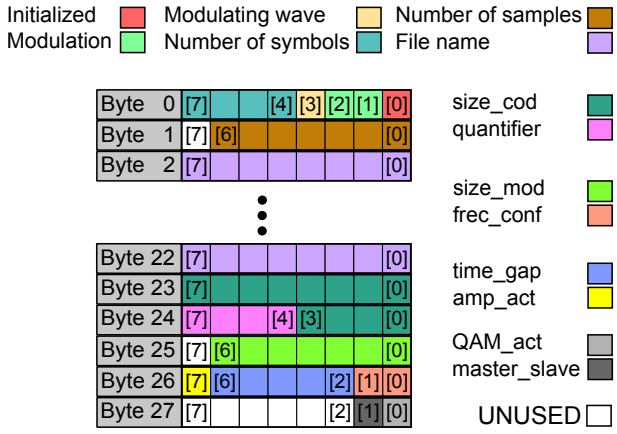


Figura 5.53: Ordenación de datos en memoria FLASH - Bloque1.

En la figura 5.53 se muestra la distribución de los primeros veintiocho bytes. A excepción del bit cero del byte cero que muestra si la memoria FLASH ha sido escrita con anterioridad, los veintitrés primeros bytes contienen información de la anterior escritura y son sólo necesarios para enviar al PC externos vía Wi-Fi (son los parámetros con los que está configurado el dispositivo). Dichos parámetros son: el tipo de modulación (*BPSK*, *QPSK*, *APK* o *QAM*), la señal moduladora (seno o señal cuadrada), número de símbolos por código, número de muestras por unidad moduladora y el nombre del archivo de los códigos que se cargaron. Los cinco siguientes bytes son los parámetros de configuración inicial, es decir los parámetros que van en los registros cero (figura 5.5) y cinco (figura 5.9) del periférico. Del byte veintiocho en adelante se guardan los datos de los códigos, la amplitud de los mismos y la moduladora

siguiendo el esquema de la figura 5.54. Estos datos son los que se escriben en los registros uno (figura 5.6), dos y tres (figura 5.7) del periférico.

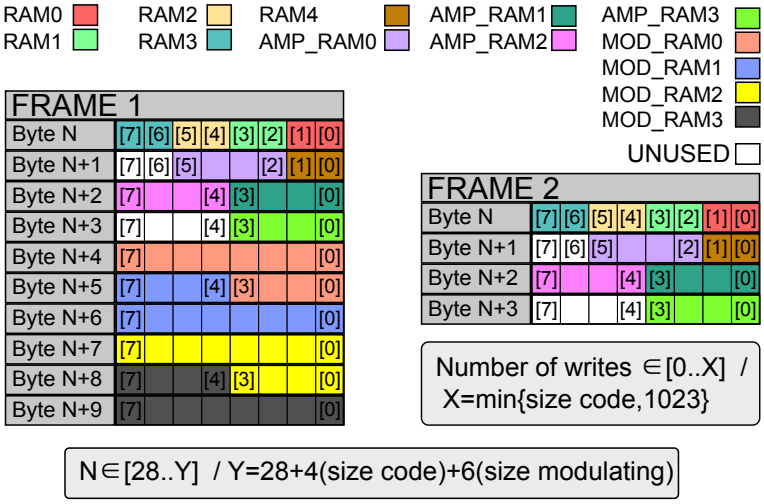


Figura 5.54: Ordenación de datos en memoria FLASH - Bloque2.

Dado que el tamaño del código siempre es superior al de la moduladora, la estructura de los datos en memoria FLASH ha de ser distinta cuando se hayan escrito todos los datos de la moduladora. Se muestran en la figura 5.54 las dos estructuras, y se escribe usando cada una de ellas en función del número de escrituras realizado hasta el momento como se muestra en el flujograma de la figura 5.55.

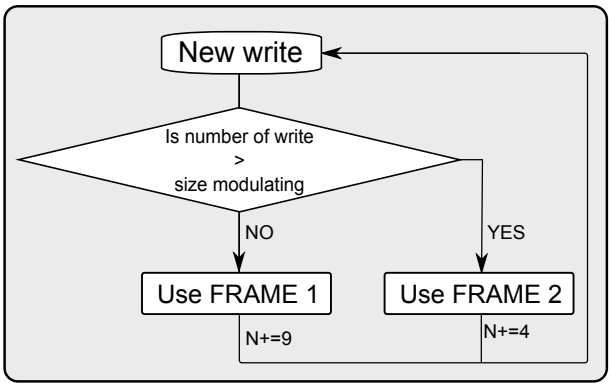


Figura 5.55: Bloque2 - Flujo escritura.

Funciones propias usadas en la ejecución del programa. A lo largo del programa es necesario realizar una serie de tareas para el correcto funcionamiento del sistema, éstas han sido distribuidas en funciones cuyo acometido y forma de uso se explican a continuación. Cabe destacar que existen dos bloques diferenciados de funciones: un bloque de control de memoria RAM (ésta memoria RAM es la que está dentro del periférico de creación propia y se escribe a través de los registros de comunicación con el mismo) y otro bloque para el control de memoria FLASH.

- **Iniciación de escritura en memoria RAM**; se encarga de establecer en el periférico los parámetros de configuración del sistema (configuración maestro o esclavo, activado o no modo QAM, valor del cuantificador de amplitud, cuantificación de amplitud activada o no, configuración de la

frecuencia, tamaño de la moduladora y del código) y la señal de inicialización de nueva escritura del registro cero (figura 5.5). Se necesita llamar a esta función con todos los parámetros cuando se va a introducir una nueva configuración, los parámetros se sitúan en el periférico en los registros cero (figura 5.5) y cinco (figura 5.9) del periférico y activa la señal de escritura, *we_cod*, de la RAM en el registro uno (figura 5.6).

void initWriteRam (Xuint32 master_slave,Xuint32 QAM_act,Xuint32 quantifier,Xuint32 amp_act,Xuint32 time_gap,Xuint32 frec_conf,Xuint32 size_ mod,Xuint32 size_cod)

- **Escritura de parámetros en RAM**; carga en los registros uno (5.6), dos y tres (5.7) del periférico, los valores del código, las amplitudes del mismo y los valores de la moduladora (el código siempre ocupa más que la moduladora, una vez se han acabado los valores de moduladora es indiferente el valor que se le ponga, el periférico los ignorará y por tanto no se toman mayores precauciones), además se encarga de comunicar al periférico que existe un nuevo dato activando el bit *new_data* del registro uno (5.6), el programa espera a la respuesta del periférico de que la escritura ha sido realizada correctamente leyendo el bit *ack_ublaze* del registro cuatro (5.8) y desactiva el bit *new_data* del registro uno (5.6), este proceso de escritura de un dato nuevo en el periférico se muestra en el flujograma de la figura 5.56.

void writeDataToRAM(Xuint32 RAM1,Xuint32 RAM2,Xuint32 RAM3,Xuint32 RAM4,Xuint32 RAM5, Xuint32 AMP_RAM1,Xuint32 AMP_RAM2,Xuint32 AMP_RAM3,Xuint32 AMP_RAM4, Xuint32 MOD_RAM1,Xuint32 MOD_RAM2,Xuint32 MOD_RAM3,Xuint32 MOD_RAM4)

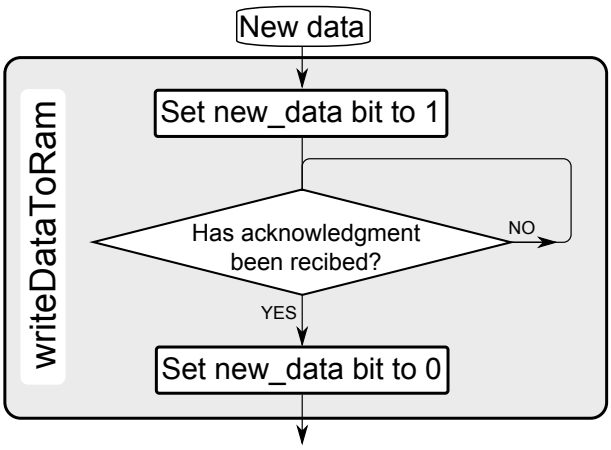


Figura 5.56: Flujo de la función WriteDataToRam.

- **Finalizar proceso de escritura en RAM**, realiza la misma funcionalidad que *initWriteRam* y con los mismos parámetros, pero desactivando la señal de escritura y la de inicialización de nueva escritura. Esta función deja el periférico listo para una nueva escritura repitiendo un proceso de inicialización escritura finalización.

void endWriteRam (Xuint32 master_slave,Xuint32 QAM_act,Xuint32 quantifier,Xuint32 amp_act,Xuint32 time_gap,Xuint32 frec_conf,Xuint32 size_mod,Xuint32 size_cod)

- **Inicialización de la memoria FLASH**. Se diferencia la inicialización para lectura y para escritura: la inicialización para lectura, realiza la llamada a tres funciones de la API de SDK,

XFlash_Initialize(), *XFlash_Reset()* y *XFlash_Unlock()*, que se encargan de iniciar un puntero a la dirección inicio de la memoria FLASH y establece el ancho de la misma, un reseteo de la memoria FLASH donde apunta dicho puntero y un desbloqueo de la memoria respectivamente. En el caso de escritura además se necesita un borrado, realizado por la función *XFlash_Erase()*, de lo que existía antes para poder escribir, en caso contrario cualquier intento de escritura arrojará un error. Esta función necesita ser invocada antes de una escritura o una lectura a memoria FLASH, pasándole un cero si es de lectura o un uno si es de escritura, siguiendo el flujograma presentado en la figura 5.57.

int initFlash(int writeAv)

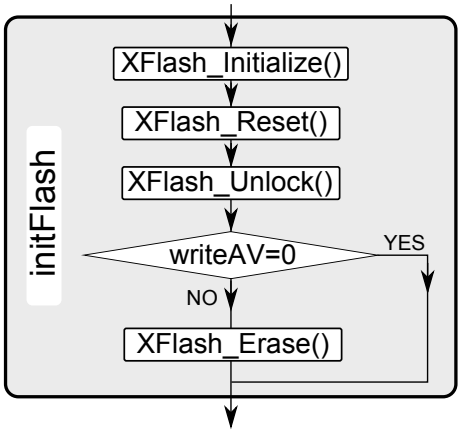


Figura 5.57: Flujo de la función initFlash.

- **Escritura en memoria FLASH**. Esta función carga todos los códigos (código, amplitud y moduladora) y todos los parámetros de configuración del sistema en FLASH y se encarga de poner a cero el bit cero del byte cero como marca de que la memoria FLASH ha sido escrita con anterioridad. Esta función se llama únicamente cuando se tienen todos los datos a escribir listos y estos se escriben con las distribuciones que se muestran en la figuras 5.53 y 5.54. En el caso de la figura 5.54 siguiendo el flujograma mostrado en la figura 5.55 (en el puntero *fileName* están los veintitrés primeros bytes y en el puntero data_flash se encuentran todos los datos ya estructurados que van a ir a partir del byte veintiocho) .

*void writeToFlash(char * fileName,u8 * data_flash,Xuint32 master_slave,Xuint32 QAM_act,Xuint32 quantifier,Xuint32 amp_act,Xuint32 time_gap,Xuint32 frec_conf,Xuint32 size_mod,Xuint32 size_cod)*

- **Lectura de parámetros de configuración de emisión de la memoria FLASH**. Esta función se encarga de leer sólo los parámetros de configuración (configuración maestro o esclavo, activado o no modo QAM, valor del cuantificador de amplitud, cuantificación de amplitud activada o no, configuración de la frecuencia, tamaño de la moduladora y del código) de memoria FLASH que como se muestra en la figura 5.53 se encuentran ubicados en los bytes de veintitrés a veintisiete desalineados, por lo tanto esta función deberá alinearlos a enteros de treinta y dos bits, y los devolverá cada uno en una posición de memoria distinta.

*Xuint32 * readParamFromFlash(void)*

- **Comprobación de que la memoria FLASH ha sido escrita con anterioridad**. Comprueba que en el bit cero del byte cero mostrado en la figura 5.53 hay un cero escrito (cuando la memoria FLASH tiene los valores de fábrica, estos están todos a uno), si es así la memoria FLASH ha sido escrita con anterioridad y en ese caso devuelve un 1, en caso contrario devuelve un 0. Esta función es llamada en el hilo principal, como se muestra en la figura 5.52, para realizar una carga de los parámetros de FLASH en el periférico o no. Cabe destacar que si no ha sido escrita con anterioridad la memoria FLASH y se intenta leer, el MicroBlaze se queda bloqueado.

u8 isFlashInitiated(void)

- **Lectura de datos de la memoria FLASH y escritura de los mismos en los registros del periférico**. Esta función se llama directamente en el hilo principal siempre que la comprobación de que la memoria FLASH ha sido escrita con anterioridad sea positiva, como se muestra en al figura 5.52.

void startSystemFromFlash(void)

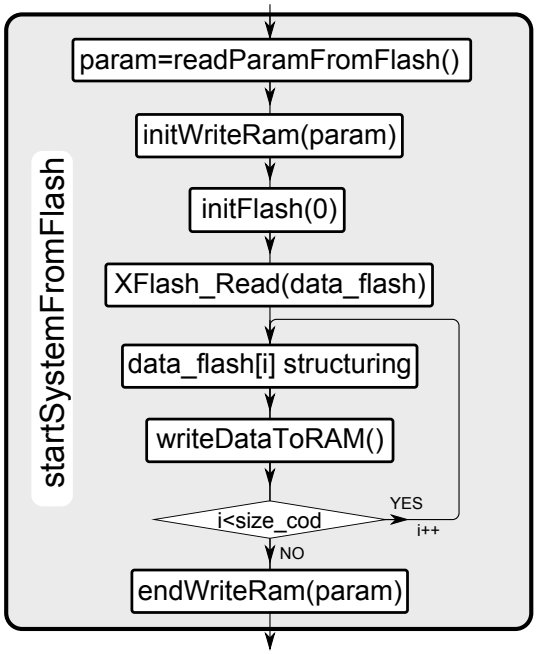


Figura 5.58: Flujo de la función startSystemFromFlash.

- **Lectura y emisión vía Ethernet de los parámetros con los que fue configurada la FPGA en la anterior escritura**. Estos parámetros están guardados en los primeros veintitrés bytes mostrados en la figura 5.53 y son: el tipo de modulación (BPSK,QPSK,APK o QAM), la señal moduladora (seno o señal cuadrada), número de símbolos por código, número de muestras por unidad moduladora y el nombre del archivo de los códigos que se cargaron. Solamente es necesario pasarle la estructura que guarda la configuración de la conexión y la propia función se encarga de mandar los datos.

*void readAndSendConf(struct tcp_pcb *tpcb)*

Recepción de datos mediante conexión Ethernet: cuando existen nuevos datos recibidos desde la conexión Ethernet se produce una interrupción hardware y se salta a una subrutina de tratamiento de la misma. En dicha subrutina está implementado el diagrama de estados que se muestra en la figura 5.59, dicho diagrama de estados tiene como fin la adquisición y ordenación de los parámetros enviados desde el PC así como la escritura de los mismos tanto en memoria FLASH como en los registros de comunicación.

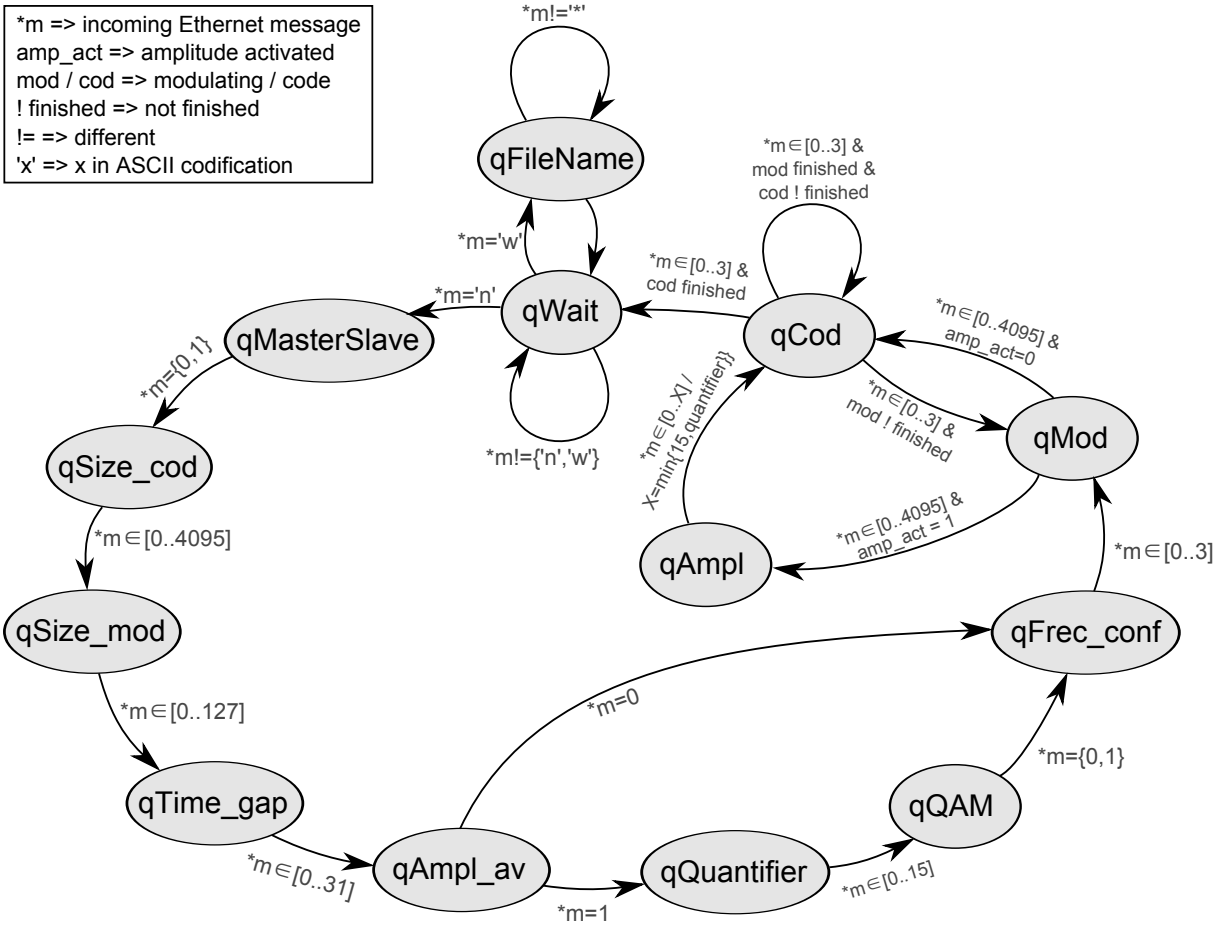


Figura 5.59: Diagrama de estados - recepción Ethernet.

1. **Estado *qWait***: es el estado de reposo y se espera una recepción de uno de los siguientes en códigos ASCII:

   - Código 'c', llama a la función *readAndSendConf()*.
   - Código 'w', modifica el estado a *qFileName*.
   - Código 'n', modifica el estado a *qMasterSlave*

2. **Estado *qFileName***: se esperan los bytes de configuración que se alojarán en en los 2 primeros bytes de la memoria FLASH, y el nombre del archivo de los códigos cargado que podrá tener una extensión máxima de 20 caracteres ASCII, ya que así se dimensionó la repartición de memoria FLASH como se muestra en la figura 5.53. Cuando se recibe el carácter '*' se vuelve al estado de espera *qWait*.

3. **Estado *qMasterSlave***: en la variable *master_slave* se introduce un 0 en caso de que se configure como esclavo o un 1 en caso de que se configure como maestro. Se pasa al estado *qSize_cod* tras

la llegada del dato de configuración.

4. **Estado *qSize_cod***: el tamaño del código se configura en la variable *size_cod*, por diseño, este valor sólo puede estar entre 0 y 1023. Se pasa al estado *qSize_mod* tras la llegada del dato de configuración.

5. **Estado *qSize_mod***: el tamaño de la moduladora se configura en la variable *size_mod*, por diseño, este valor sólo puede estar entre 0 y 123. Se pasa al estado *qTime_gap* tras la llegada del dato de configuración.

6. **Estado *qTime_gap***: la espera entre emisiones queda configurada en la variable *time_gap*, por diseño, este valor sólo puede estar entre 0 y 63. Cada incremento de la variable *time_gap* tiene un efecto de un incremento de 5ms entre emisiones. Se pasa al estado *qAmpl_av* tras la llegada del dato de configuración.

7. **Estado *qAmpl_av***: dado que se han implementado modulaciones con la posibilidad de modular en amplitud, se ha usado la variable *amp_act* para diferenciar con respecto a las que no se modula en amplitud. Un 1 indica que se trata de una modulación en amplitud y además se marca el salto al estado *qQuantifier* donde se configuran parámetros de la configuración en amplitud, por otro lado, un 0 indica que no es una modulación en amplitud y por tanto no se necesita configurar dichos parámetros y se saltaría al estado *qFrec_conf*.

8. **Estado *qQuantifier***: en caso de que se trate de una modulación en amplitud, se necesita saber el valor máximo por el que se multiplica la señal moduladora debido al diseño de la modulación llevado a cabo en el periférico. Dicho valor se introduce en la variable *quantifier*. Se pasa al estado *qQAM* tras la llegada del dato de configuración.

9. **Estado *qQAM***: en este estado se especifica si la modulación en amplitud es una modulación QAM o se trata de una modulación APK, indicado con un valor 1 o 0 en la variable *qam_act* respectivamente. Se pasa al estado *qFrec_conf* tras la llegada del dato de configuración.

10. **Estado *qFrec_conf***: el sistema habilita la emisión a tres frecuencias distintas, la elección de una u otra se realiza mediante un multiplexor en el periférico y dicho multiplexor se controla con la variable *frec_conf*. Se pasa al estado *qMod* tras la llegada del dato de configuración.

11. **Estado *qMod***: en este estado se espera el valor en amplitud de la moduladora que oscila entre un valor 0 y el valor 4095, se corresponden con los valores cero voltios y tres con tres voltios de salida en el DAC respectivamente. Si se está ante una modulación en amplitud se pasa al estado *qAmpl*, en otro caso se pasa al estado *qCod* tras la recepción del dato de configuración.

12. **Estado *qAmpl*:** en este estado se espera el valor por el que se va a multiplicar la amplitud de cada una de las moduladoras asociadas a cada código, este valor puede oscilar entre 0 y el valor de la variable *quantifier*. Se pasa al estado *qCod* tras la llegada del dato de configuración.

13. **Estado *qCod***: en este estado se espera el valor de cada código que sólo puede tomar valores entre 0 y 3. Este código realiza la función de selección de unos multiplexores que elegirán cual es la señal de las cuatro disponibles a la entrada que pasará a la salida. Si se está ante una modulación en amplitud se pasa al estado *qAmpl*, en otro caso se pasa al estado *qCod* tras la recepción del dato de configuración. En caso de que no se haya llegado al fin de la recolección de valores de las moduladoras se pasa al estado *qMod*, si no es así y no se han escrito todos los códigos el siguiente estado es el mismo estado *qCod* y en caso de que se hayan escrito todos los códigos se pasa al estado de espera del diagrama *qWait*.

5.2. Interfaz software

En este apartado se explica la interfaz software desarrollada en Matlab. Éste tiene la apariencia mostrada en la figura 5.60 donde se describen los parámetros a modificar de manera remota:

- **Tipo de modulación**: BPSK, APK, QPSK o QAM, explicados en la sección 3.2. Esto permite configurar la creación de las señales moduladoras y la modificación de los códigos antes de ser enviados.
- **Separación temporal inactivo entre las sucesivas emisiones**: parámetro *Time_gap*, que puede tomar cualquier valor entre 0ms y 155ms. Este parámetro sólo se tiene en cuenta en caso de que el sistema esté configurado como maestro.
- **Secuencias utilizadas para codificar las transmisiones ultrasónicas**. Los códigos se facilitan a través de un archivo, en la interfaz se indica el nombre de dicho archivo que en el caso mostrado sería *kasami_bpsk.mat*. La longitud máxima del nombre del archivo es de 15 caracteres ASCII.
- **Señal portadora**: cuadrada o senoidal.
- **Número de símbolos por código enviado**, es decir, número de senos/señales cuadradas que se emitirán por cada código de señal.
- **Número de muestras por unidad de la señal moduladora por código**: esto es el número de valores dados a la señal moduladora. El número de muestras multiplicado por el número de símbolos no debe exceder 127.
- **Frecuencia de muestreo usado en la generación de la transmisión ultrasónica**: 400kHz, 416.66 kHz y 500kHz.
- **Configuración del sistema**: maestro o esclavo.
- **Dirección IP del *SoC* a configurar y puerto de enlace para la conexión Ethernet**.

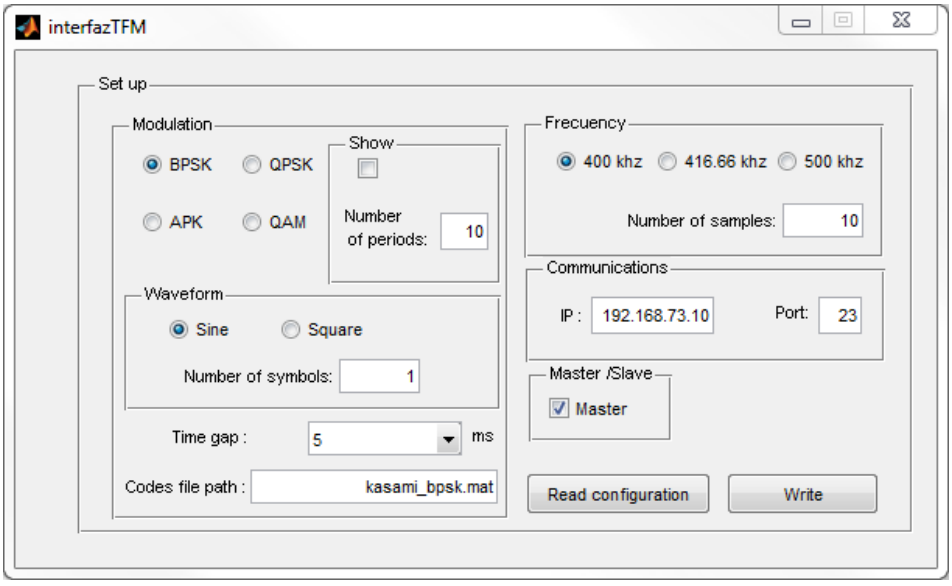


Figura 5.60: Captura de la interfaz gráfica de Matlab.

Además de la modificación de los parámetros con los diversos selectores, existen otros interactuadores de la interfaz cuya finalidad es:

- Opción *Show* [$\surd$] cuando se activa se muestran la moduladora que se envía y un número de periodos modulados esta señal sería la que saldría por el conversor DAC. Esto ayuda a comprobar el correcto funcionamiento del sistema.

- Botón *Read configuration*, permite leer la configuración de los LPS ya inicializados en escrituras anteriores y que están dentro de la red.

- Botón *Write* envía los parámetros recogidos a través de la conexión Ethernet con el puerto y la IP especificados.

5.2.1. Establecimiento de parámetros

Cuando se pulsa el botón *write* el hilo del programa salta a la función *WriteButton_Callback*. El primer paso consiste en la recolección de los parámetros configurados en la interfaz gráfica principal (figura 5.60) con los cuales se llama a la función *uBlaze_tcpip*, que es la encargada de a partir de los parámetros recogidos preparar todos los datos, mostrar las señales si es requerido, manejar la conexión TCP_IP y enviar los datos.

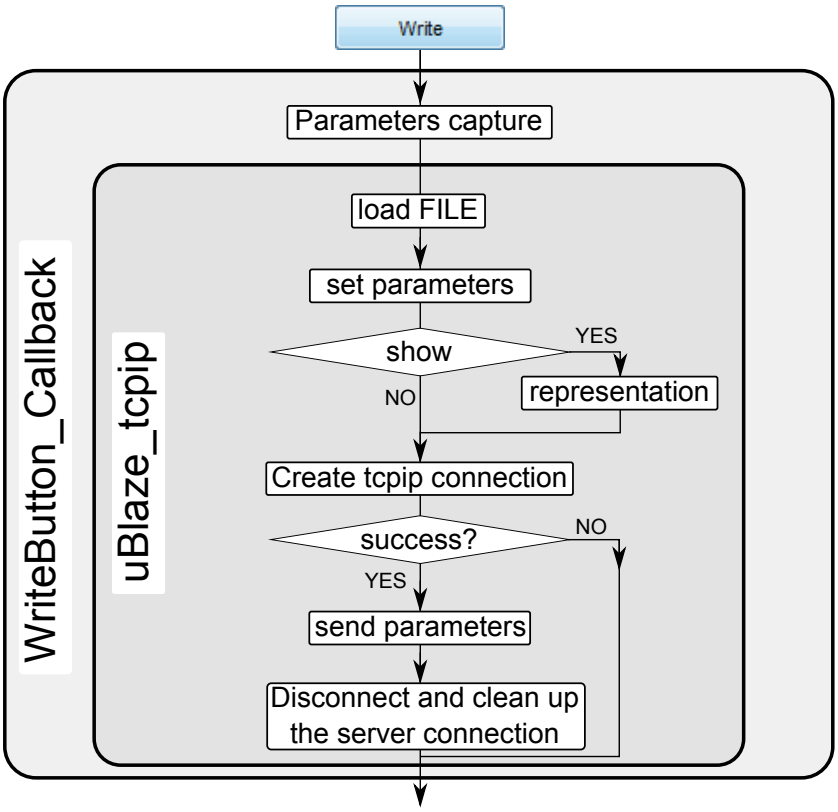


Figura 5.61: Flujo principal.

5.2.1.1. Preparación de datos a enviar

En la figura 5.62 se muestra el flujograma de la preparación de datos a partir de los datos recogidos de la interfaz gráfica.

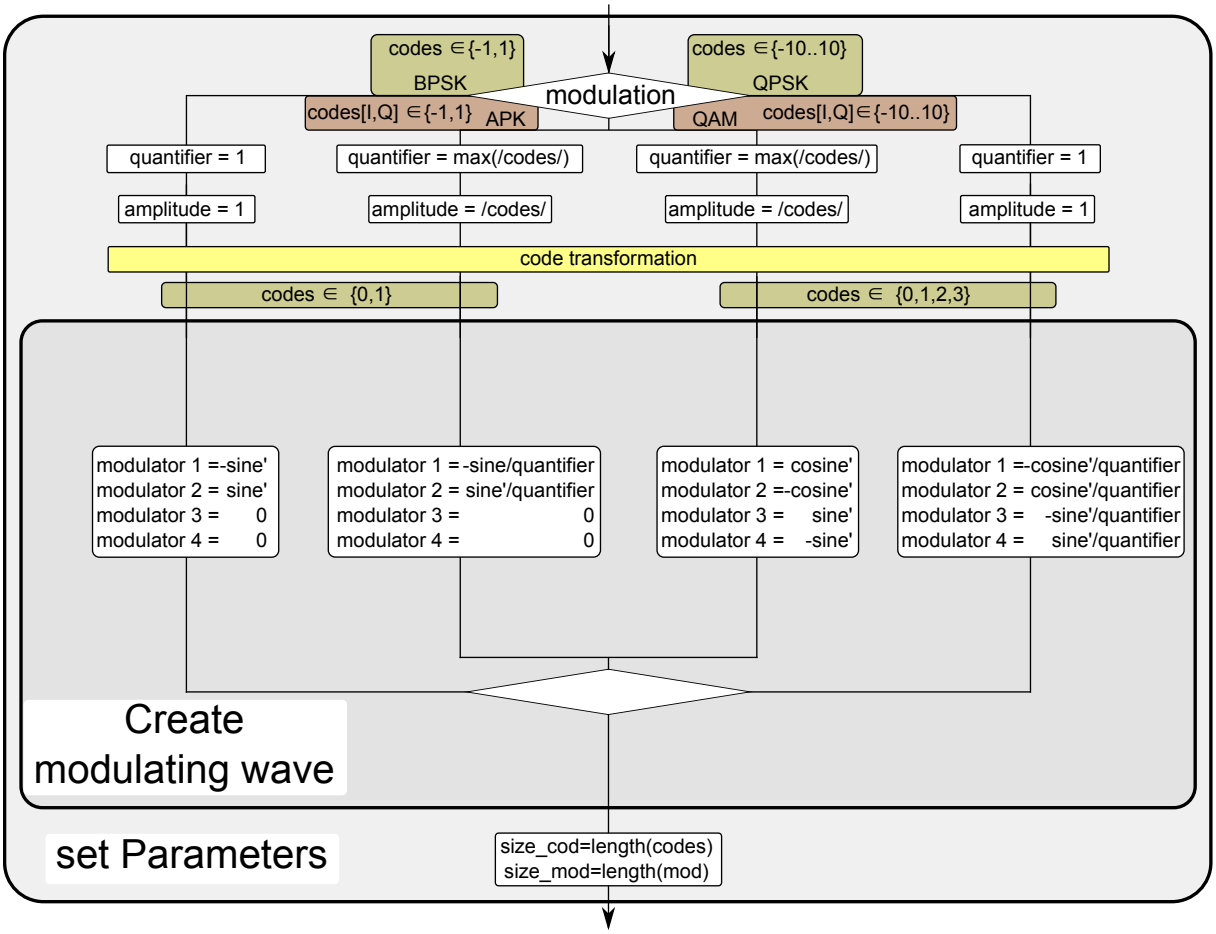


Figura 5.62: Flujograma de preparación de datos.

Los códigos leídos del fichero pueden tomar cualquier número entero en el rango [-10,10] y han de ser tratados antes de ser enviados ya que en la FPGA los códigos esperados sólo pueden tomar los valores 0, 1, 2 y 3. Además del código se tiene que generar el cuantificador, que es el valor máximo de amplitud de los códigos, y la amplitud de los códigos, que es el valor absoluto de los códigos. Una vez realizado esto se realiza la transformación de códigos, para los códigos que usan modulación en cuadratura (*QAM* y *QPSK*) se realiza a partir de los signos de los códigos [I,Q] siguiendo la tabla 5.1.

	[I]<0	[I]>0
[Q]<0	$0 \rightarrow [-sine - cosine]$	$2 \rightarrow [+sine - cosine]$
[Q]>0	$1 \rightarrow [-sine + cosine]$	$3 \rightarrow [+sine + cosine]$

Tabla 5.1: Transformación de código para modulaciones en cuadratura.

Para las modulaciones *BPSK* y *APK* la transformación es más sencilla, basta con cambiar los valores -1 del código por un 0. El valor 0 corresponde con una señal moduladora $-sine$ y un valor 1 con una señal $+sine$.

El siguiente paso es la creación de las señales moduladoras. Dicha creación tiene que realizarse siguiendo los parámetros introducidos en *Number of samples* y *Number of symbols*, éstos varían el número de valores dados en un período de señal (con más datos se obtendrá mayor resolución) y el número de períodos por código respectivamente. Se diferencian para cada tipo de modulación los valores de la señal moduladora:

- **Modulación *BPSK***: las señales moduladoras son $\pm sine \in [0 \dots 4095]$, centrados en 2048.
- **Modulación *APK***: la señal moduladora ha de ser tal que cuando se multiplique por el valor máximo del código no supere nunca el valor máximo del *SPAM* de salida, por tanto se crean las señales moduladoras $\pm seno/quantifier \in [0 \dots (4095/quantifier)]$, centrados en $(2048/quantifier)$. La labor de centrado final para la señal de salida la realiza el bloque generador de *offset* mostrado en la figura 5.27 del diseño hardware.
- **Modulación *QPSK***: en caso de la modulación en cuadratura hay que tener en cuenta que la suma de cualquiera de las señales mostradas en la tabla 5.1 ha de estar dentro del rango del *SPAM* de salida $[0 \dots 4095]$ y además centrada en *SPAM*/2. Todo esto lo cumplen las señales moduladoras $\pm sine \pm cosine \in [0 \dots 1024]$ centradas en 2048.
- **Modulación *QAM***: en caso de esta modulación, hay que combinar las precauciones para modulación *APK* y para modulación *QPSK*. En el peor de los casos, las señales deben quedar dentro del rango *SPAM/2* y para cualquier caso la señal ha de estar centrada en *SPAM*/2, al igual que para *APK* la labor de centrado final para la señal de salida la realiza el bloque generador de *offset*. Todo esto lo cumplen las señales moduladoras $\pm sine/quantifier \pm cosine/quantifier \in [0 \dots (1024/quantifier)]$ centradas en 1024.

El número de códigos a transmitir, *size_cod*, y el número de muestras de la portadora, *size_mod*, se sacan directamente de medir la longitud de las cadenas de código y de moduladora respectivamente. Cabe destacar que también se puede realizar la modulación con cualquier otra señal que cumpla las características resaltadas en los puntos anteriores, por ejemplo una señal cuadrada.

5.2.1.2. Representación

En caso de que la opción *Show* esté activada, [$\surd$], cuando se produce una escritura se muestra por pantalla el número de períodos especificados en el espacio *Number of periods* de la señal moduladora y las señales moduladoras. En las figuras 5.63 a 5.66 se muestras distintos casos para la representación:

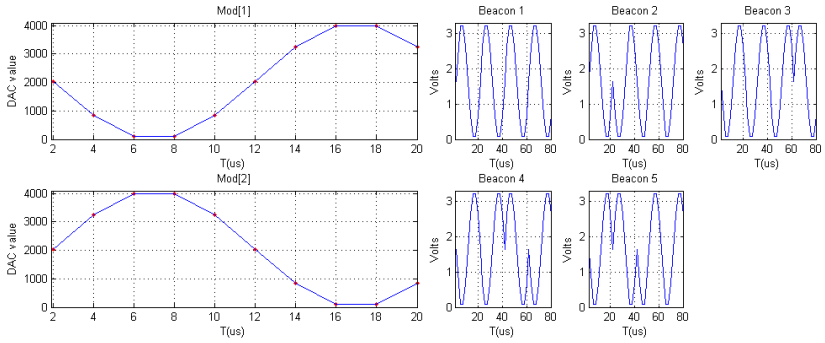


Figura 5.63: Representación BPSK.

En la figura 5.63 se muestran 4 períodos de una modulación BPSK con 10 muestras y un símbolo por código a una frecuencia de 500kHz. En la figura 5.64 se muestran 4 períodos de una modulación QPSK con 25 muestras y dos símbolos por código a una frecuencia de 400kHz.

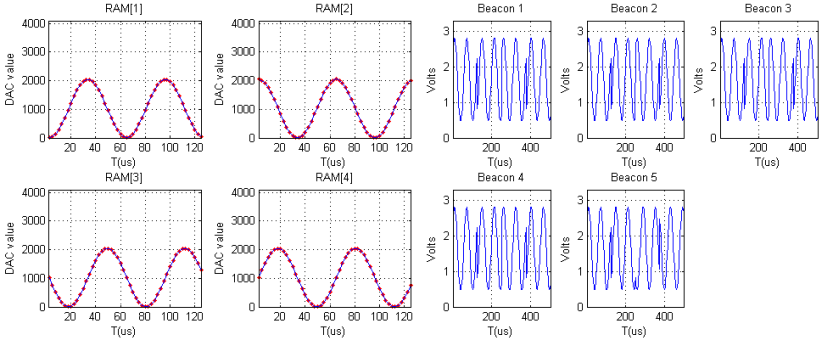


Figura 5.64: Representación QPSK.

En la figura 5.65 se muestra la cadena de códigos completas de una modulación APK con 10 muestras y un símbolos por código a una frecuencia de 400kHz.

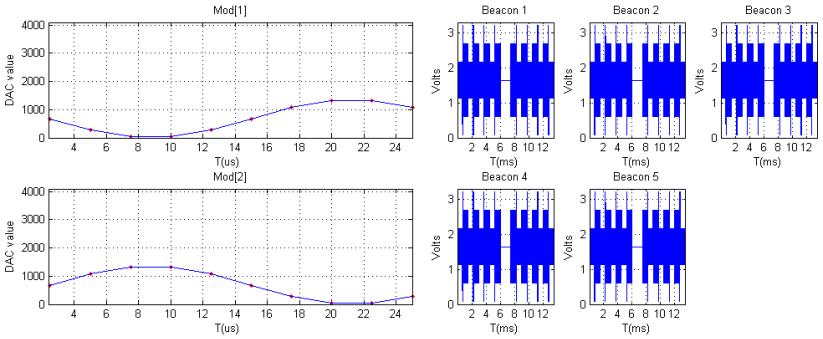


Figura 5.65: Representación APK.

En la figura 5.66 se muestran 18 períodos de una modulación QAM con 100 muestras y un símbolo por código a una frecuencia de 416.66kHz.

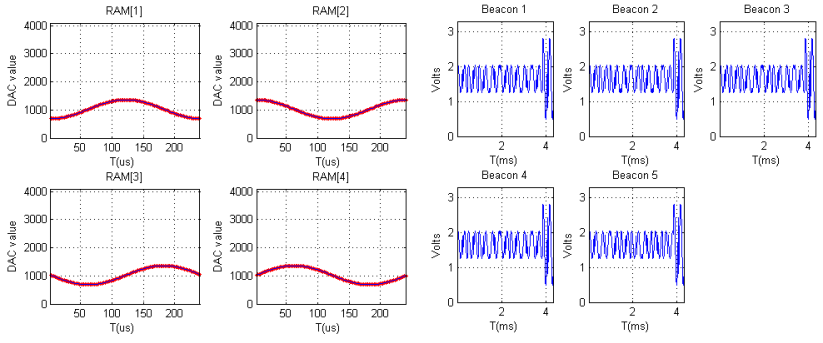


Figura 5.66: Representación QAM.

Estas representaciones se pueden usar a modo de depuración para saber si lo que se va a producir es lo deseado y además si la salida de los DAC del sistema están produciendo las señales deseadas.

5.2.1.3. Conexión TCP-IP y envío de parámetros

Una vez recogidos los parámetros y mostradas las señales en su caso, se ha de crear la conexión TCP_IP, enviar los datos esperados y posteriormente realizar la limpieza y desconexión del servidor tal y como se muestra en la figura 5.61. La secuencia seguida es:

- En primer lugar se ha de crear la conexión en el puerto e IP especificado en la interfaz gráfica. Si la conexión se ha establecido satisfactoriamente se comienza con la escritura de datos.
- A continuación se envían los datos que se alojarán en los primeros 23 bytes de la memoria FLASH de la FPGA (figura 5.53). Se sigue el flujo de escritura mostrado en la figura 5.67, mandando un primer carácter ASCII *'w'* se comienza la escritura del bloque 1. Tras el inicio se mandan los bytes 0 y 1, que están organizados según marcan los primeros dos bytes de la memoria FLASH, seguidamente se mandan los caracteres ASCII que componen el nombre del archivo cargado y por último se manda el carácter ASCII *'*'* para comunicarle a la FPGA que se ha terminado la escritura del primer bloque.

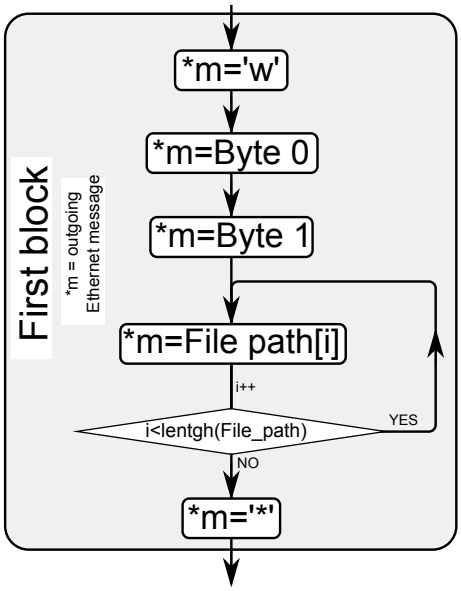


Figura 5.67: Envío del primer bloque.

- El siguiente paso es la escritura del segundo bloque donde se envían los parámetros necesarios para el correcto funcionamiento de la FPGA. Los parámetros se envían siguiendo la misma secuencia que sigue la recepción de datos de la FPGA mostrado en el diagrama de estados de la figura 5.59. La emisión del segundo bloque se acaba una vez se han enviado todos los parámetros esperados.
- Una vez realizado todo el proceso se realiza la desconexión y limpieza de la conexión TCP_IP creada.

En el momento que se termina todo el proceso de escritura desencadenado tras presionar el botón *Write* se muestra en pantalla un mensaje como el mostrado en la figura 5.68, donde se detallan los parámetros elegidos en la interfaz de usuario. Si ha existido algún problema con la conexión, saldrá el mensaje *Unable to open connection*, si se ha realizado el envío correctamente saldrá el mensaje mostrado en la figura 5.68 **** Done !!!* .

```
File path :  * GPC_QAM.mat
Settings  :
             * Modulation : QAM
             * Modulator  : Sine
             * N_symbols  : 1
             * Frecuency  : 416Khz
             * N_samples  : 100
             * IP         : 192.168.73.10
             * PORT       : 23
             * Time gap   : 5 ms
             * Status     : Master
*** Sending...
*** Done !!!
```

Figura 5.68: Mensaje de salida tras escritura.

5.2.2. Lectura de LPSs ya inicializados

Si el sistema ha sido inicializado con anterioridad, es decir, si se ha realizado un proceso de escritura como el mostrado en el punto anterior, se puede realizar una lectura de los datos que fueron establecidos para conocimiento del usuario. Tras presionar el botón *Read configuration* se desencadena el flujograma mostrado en la figura 5.69.

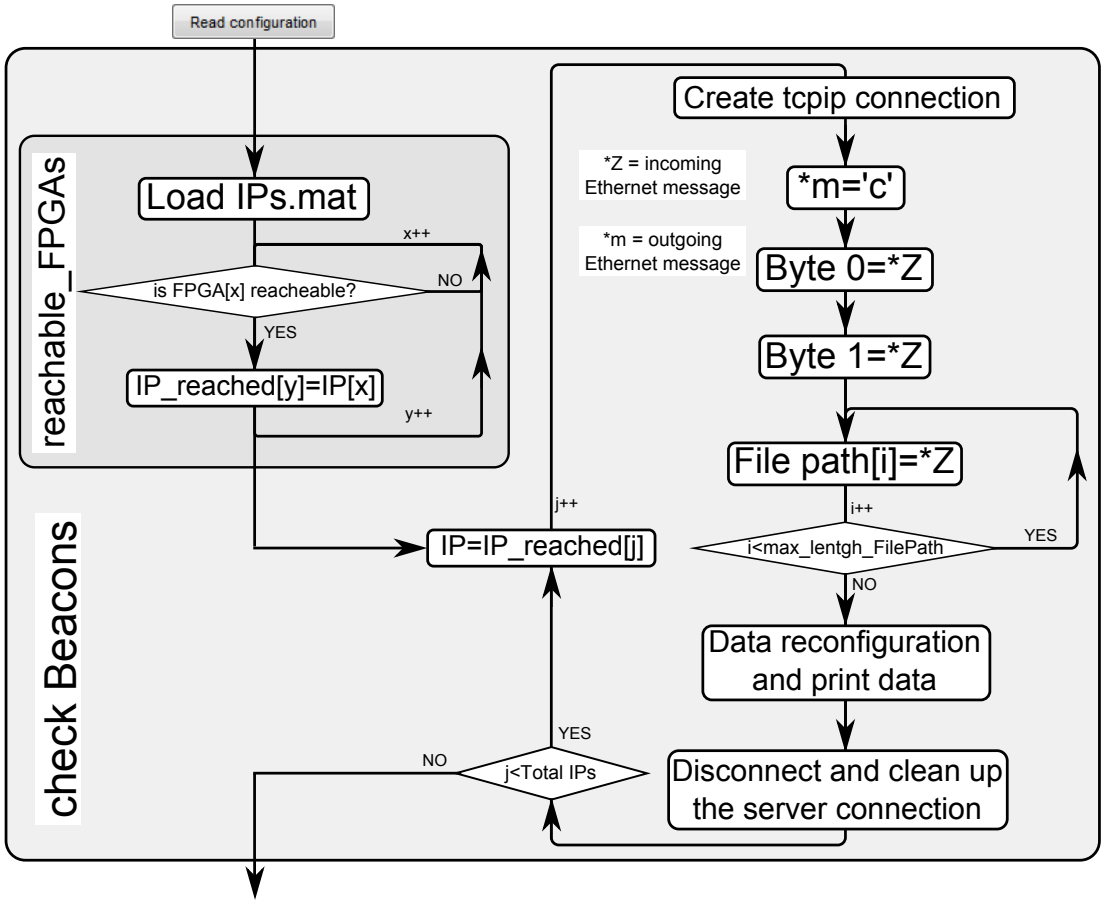


Figura 5.69: Flujograma de lectura de parámetros.

En el fichero *IPs.mat* están guardadas todas las IPs de las FPGAs. El primer paso es cargar todas las IPs y comprobar cuáles de todas las FPGAs son accesibles en la red, esto lo realiza la función *reacha-*

ble_FPGAs comprobando qué FPGAs hacen *ping*. Tras ésto, para cada IP se crea la conexión, se manda el carácter ASCII *'c'* y con esto la FPGA comenzará a enviar los bytes de configuración inicial tal y como se presentan en la memoria FLASH (figura 5.53), por último se separan los datos recibidos para obtener los parámetros, se imprimen por pantalla, se realiza la desconexión y por último la limpieza de la conexión TCP_IP actual. Al finalizar el proceso con una conexión se comienza con la siguiente hasta que se acaben las FPGAs accesibles. La figura 5.70 muestra el mensaje recibido en un sistema con una sola FPGA accesible en la red, en caso de que tuviese más saldría una a continuación de la otra.

```
Reading configuration....
 File path :  * GPC_QAM.mat

 Settings  :
             * Modulation : QAM
             * Modulator  : Sine
             * N_symbols  : 1
             * Frecuency  : 416Khz
             * N_samples  : 100
             * IP         : 192.168.73.10
             * Time gap   : 5 ms
             * Status     : Master
```

Figura 5.70: Mensaje de salida tras escritura.

Capítulo 6

Resultados

6.1. Pruebas reales

Los resultados de la plataforma implementada, con las diversas modificaciones que se pueden realizar, se muestras en las figuras 6.1 a 6.9. Las capturas se han realizado con un osciloscopio digital en la quinta salida del DAC, las medidas que se podrían realizar en las otras cuatro balizas son similares y no aportan mayor información.

- En la figura 6.1 se muestran las modulaciones BPSK, QPSK, APK y QAM. En las modulaciones QPSK y QAM, el rango dinámico de salida es distinto al de las modulaciones BPSK y APK debido a que el sumador (del bloque *decoder* mostrado en la figura 5.27) ha de tener valores de salida en el rango [0 . . . 4095] y además la señal ha de estar centrada en dicho rango.

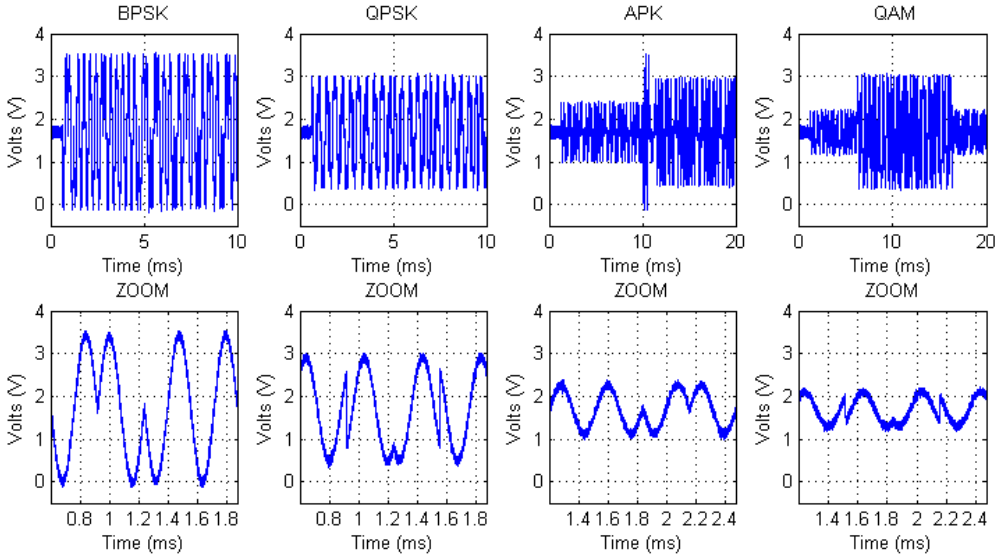


Figura 6.1: Diferentes tipos de modulación soportadas por el sistema propuesto.

- En la figura 6.2 muestra una señal portadora con una resolución de cien muestras por período configurado con las tres posibles frecuencias de muestreo (*fs*) disponibles en el sistema. El período es por tanto $100/fs$, es decir 25ms, 24ms y 20ms para 400kHz, 416.66kHz y 500kHz respectivamente.

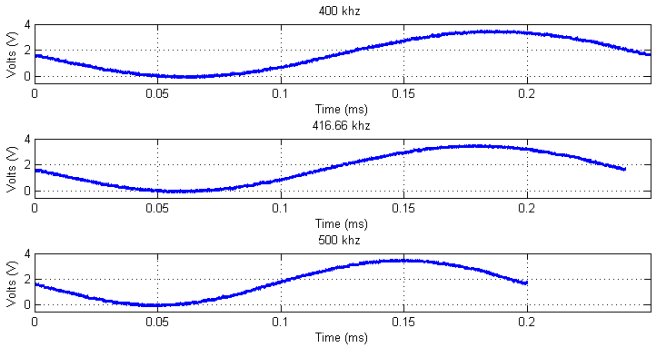


Figura 6.2: Configuración de diferentes frecuencias de muestreo.

- La figura 6.3 muestra tres configuraciones del número de períodos de portadoras por cada bit de código emitido. Se pueden observar los resultados para uno, dos y cuatro períodos de portadoras por bit.

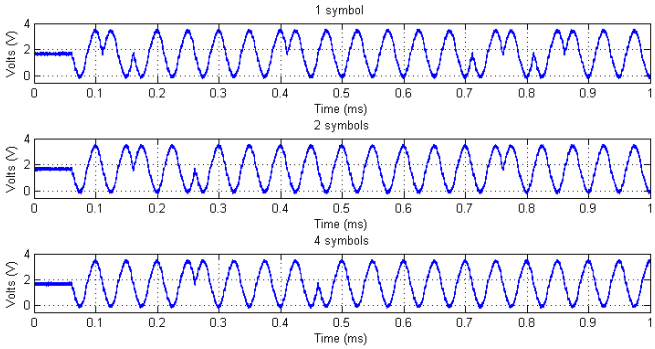


Figura 6.3: Distinto número de símbolos por código.

- La figura 6.4 muestra tres tiempos muertos entre emisiones, 5ms, 10ms y 20ms respectivamente. Cabe destacar que este parámetro sólo tiene validez cuando el sistema está configurado como maestro.

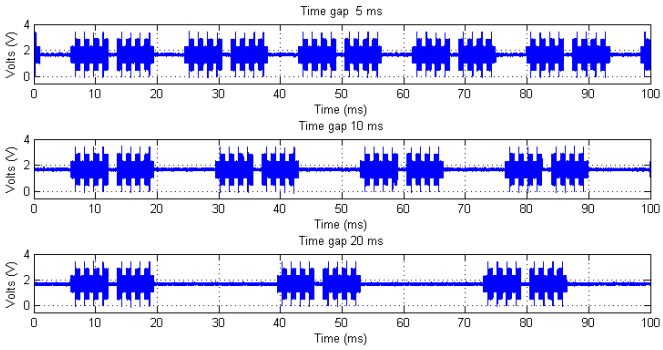


Figura 6.4: Diferentes tiempos muertos entre emisiones.

- La figura 6.4 muestra distinto número de muestras por periodo de portadora. Se muestra el resultado para 10, 50 y 127 muestras. Es importante remarcar que cuanto mayor es el número de muestras por periodo, menor es la distorsión introducida por el DAC debido a la reducción de la diferencia entre dos pasos de conversión consecutivos. Obviamente, cuanto mayor es el número, mayor es el tiempo necesario para emitir un mismo código.

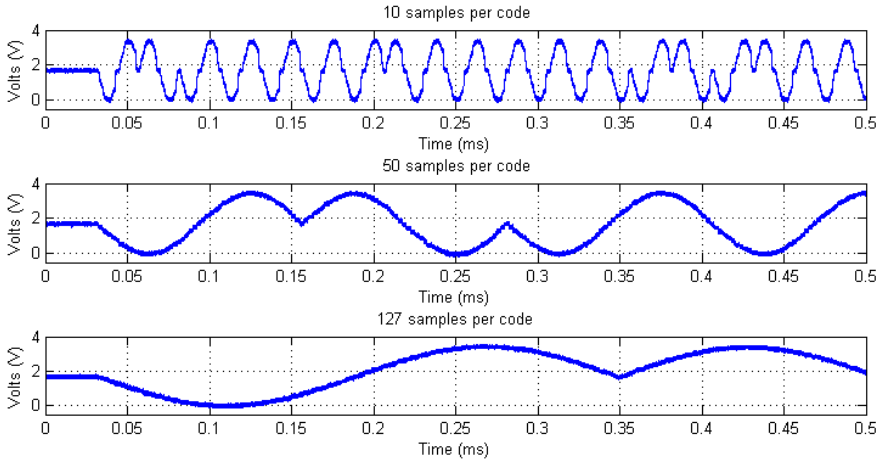


Figura 6.5: Diferente número de muestras.

- La figura 6.6 muestra el sistema en las dos posibles configuraciones, maestro y esclavo. En la configuración de maestro, la emisión se produce cuando se ha terminado el código anterior y ha pasado el tiempo muerto entre emisiones, mientras que en el esclavo la emisión se produce cuando la señal externa produce un flanco de subida.

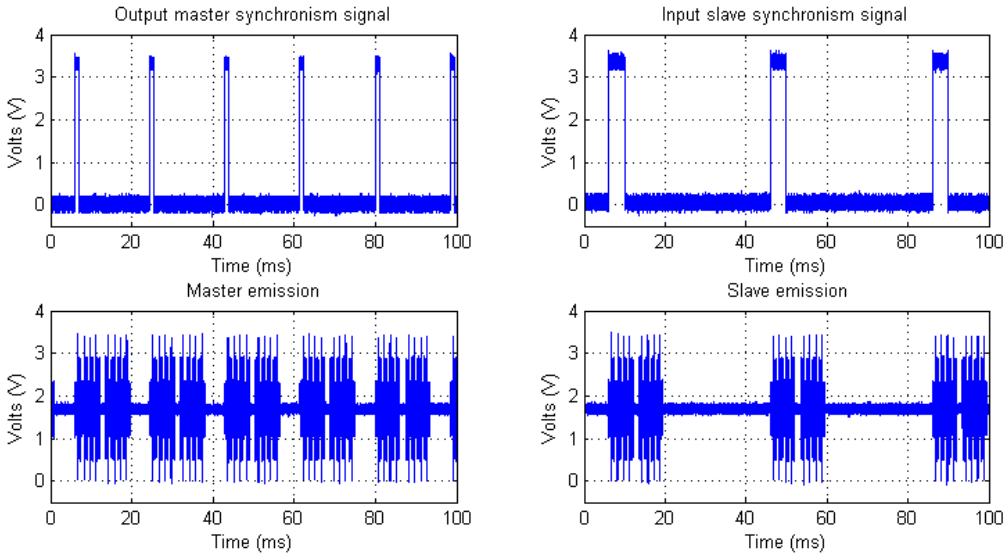


Figura 6.6: Configuración como maestro o esclavo.

Para poder ver un sistema con las dos configuraciones simultáneas, se muestra la figura 6.7 donde se pueden observar las señales de emisión de la baliza número 5 de dos FPGAs, una configurada

como maestra (señal roja) y una como esclava (señal verde), y la señal de sincronía emitida por la FPGA maestra (señal azul).

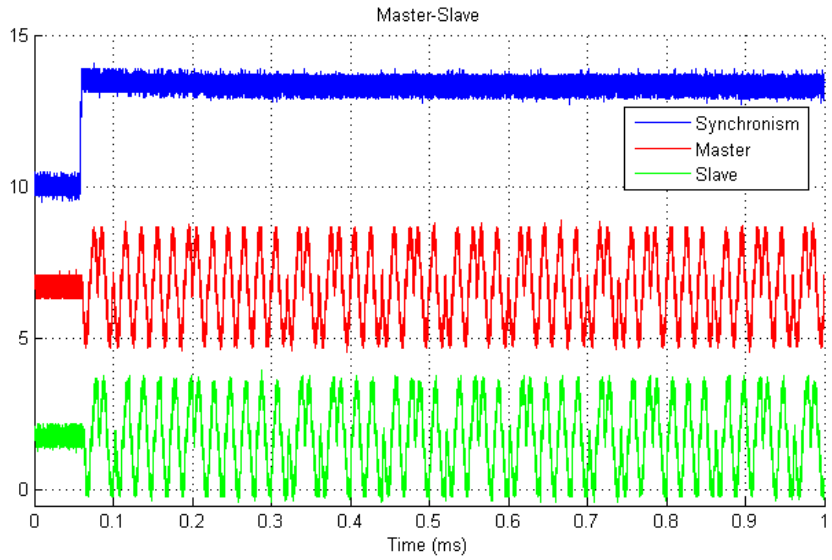


Figura 6.7: Señales de dos FPGAs una como maestro y la otra como esclavo.

Si mostramos una ampliación del inicio de la emisión de las dos señales, como la mostrada en la figura 6.8, se puede observar que existe cierta latencia en la señal de la FPGA esclava, ésto se debe a los retardos de proceso de la FPGA. Siendo éste un retardo determinista de aproximadamente 58μs, se puede corregir posteriormente de forma software por el sistema de recepción.

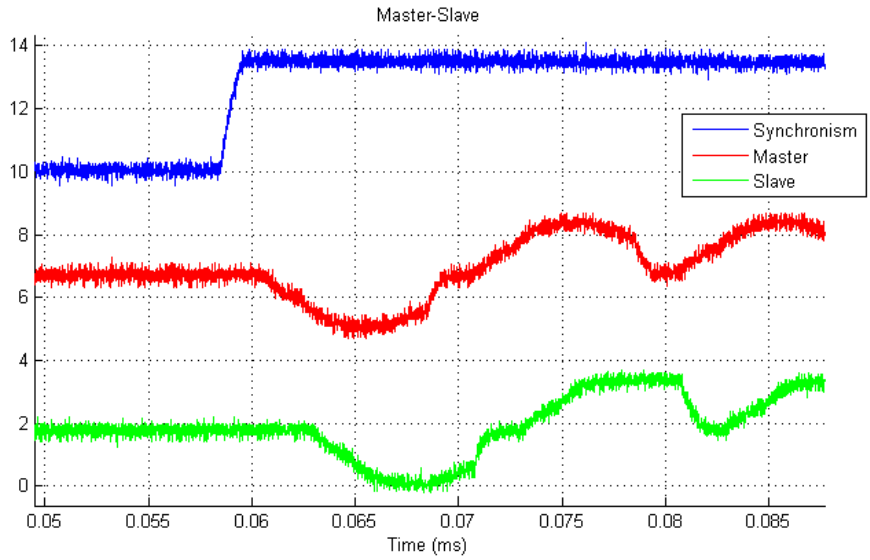


Figura 6.8: Señales de dos FPGAs una como maestro y la otra como esclavo - Ampliación.

- Por último, la figura 6.9 muestra dos distintas formas de onda para la señal portadora, una señal senoidal y una señal cuadrada. No obstante, cabe destacar que el sistema podría soportar otras formas de onda.

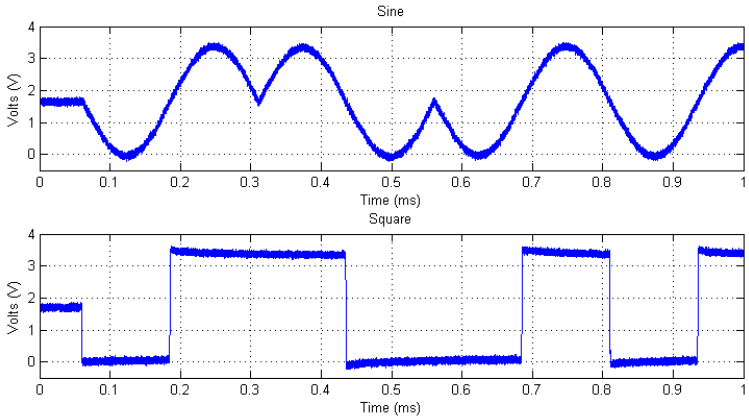


Figura 6.9: Moduladora senoidal y cuadrada.

6.2. Datos de ocupación de la FPGA

En la tabla 6.1 se muestran los datos de ocupación de la FPGA, tanto en número como en porcentaje con respecto al total de recursos que tiene la FPGA Virtex-5. Estos datos son los dados por la herramienta de síntesis XPS de Xilinx y tiene todos los módulos instanciados en el SoC usado en el proyecto.

BLOCK	Slices	BRAMs	DSPs
DDR2_SDRAM	1723 (22.43 %)	13 (21.67 %)	0 (0.00 %)
DIP_Switches_8bit	67 (0.87 %)	0 (0.00 %)	0 (0.00 %)
FLASH	219 (2.85 %)	0 (0.00 %)	0 (0.00 %)
Hard_Ethernet_MAC	910 (11.85 %)	5 (8.33 %)	0 (0.00 %)
Hard_Ethernet_MAC_fifo	507 (6.06 %)	1 (1.67 %)	0 (0.00 %)
LEDs_8bit	64 (0.83 %)	0 (0.00 %)	0 (0.00 %)
Push_Buttons_7bit	66 (0.86 %)	0 (0.00 %)	0 (0.00 %)
RS232_Uart0	103 (1.34 %)	0 (0.00 %)	0 (0.00 %)
RS232_Uart1	97 (1.27 %)	0 (0.00 %)	0 (0.00 %)
dlmb	1 (0.01 %)	0 (0.00 %)	0 (0.00 %)
dlmb_clbr	8 (0.10 %)	0 (0.00 %)	0 (0.00 %)
ilmb	1 (0.01 %)	0 (0.00 %)	0 (0.00 %)
ilmb_clbr	4 (0.05 %)	0 (0.00 %)	0 (0.00 %)
lmb_bram	0 (0.00 %)	4 (6.67 %)	0 (0.00 %)
mb_plb	137 (1.78 %)	0 (0.00 %)	0 (0.00 %)
mdm_0	96 (1.25 %)	0 (0.00 %)	0 (0.00 %)
MicroBlaze	1237 (16.10 %)	4 (6.67 %)	3 (6.25 %)
perif_ublaze_0	465 (6.05 %)	6 (10.00 %)	8 (16.67 %)
proc_sys_reset_0	29 (0.38 %)	0 (0.00 %)	0 (0.00 %)
xps_intc_0	104 (1.35 %)	0 (0.00 %)	0 (0.00 %)
xps_timer	194 (2.53 %)	0 (0.00 %)	0 (0.00 %)
TOTAL	6032 (78.54 %)	33 (55.00 %)	11 (22.92 %)

Tabla 6.1: Porcentaje de ocupación de los recursos.

Capítulo 7

Conclusiones y trabajos futuros

7.1. Conclusiones

Se ha propuesto un SoC basado en una FPGA para controlar y configurar de forma remota un LPS extenso, que consiste en un sistema formado por varios LPS individuales y cada uno de ellos a su vez consta de cinco balizas ultrasónicas. Para ello se ha dividido el proyecto en:

- **Implementación sobre plataforma hardware:** que lleva a cabo el control de una serie de DACs externos a la plataforma de desarrollo en base a una serie de parámetros que llegan mediante la conexión Wi-Fi o Ethernet. Para realizar esto realiza se desarrolla un SoC con los siguientes elementos esenciales:
  - **Controlador de memoria FLASH:** encargado de controlar las escrituras y lecturas sobre la memoria FLASH.
  - **Controlador de memoria multipuerto:** encargado en este caso de controlar el acceso de lecturas y escrituras sobre la memoria *DDR2_SDRAM*.
  - **Periférico de control de la capa MAC Ethernet:** encargado de realizar el control a nivel físico del protocolo TCP-IP.
  - **Periférico de control del LPS:** encargado de realizar la modulación de los códigos recibidos en base a una serie de parámetros recibidos todos ellos por la conexión Ethernet.
  - **MicroBlaze:** se trata del *soft-core* de Xilinx, y realiza las labores de control del sistema: gobierno del controlador de la capa MAC Ethernet, escritura de los parámetros recibidos en el periférico de control del LPS y manejo de la memoria FLASH.
- **Desarrollo de interfaz software:** lleva a cabo la recolección de datos mediante una interfaz gráfica y tras reordenar los datos son emitidos a través de una conexión TCP-IP a la FPGA que se desee.

La arquitectura desarrollada permite la configuración del sistema en tiempo de ejecución con diferentes parámetros, como las secuencias usadas para codificar las transmisiones ultrasónicas, el tipo de modulación, el tipo de portadora, la frecuencia de muestreo y la resolución de las señales. El SoC utiliza un enlace Wi-Fi para comunicarse con un PC externo que controla la recogida y envío de los parámetros, así como de presentar una interfaz de usuario amigable. Tras las pruebas reales mostradas en el capítulo 6 de resultados, se puede observar que se han cumplido los objetivos iniciales de alta parametrización del sistema de manera remota así como el correcto funcionamiento del sistema de manera global.

7.2. Trabajos futuros

Como trabajos futuros se presentan las siguientes líneas:

- Creación de una plataforma hardware para reemplazar a la plataforma de desarrollo, Genesys de Digilent, utilizada en el proyecto por una plataforma final con sólo los módulos externos la FPGA necesarios con el fin de reducir dimensiones, reducir consumo y abaratar costes, ya que la plataforma empleada es cara.
- Cambiar el router utilizado por un módulo Wi-Fi integrado dentro de la plataforma hardware del cual se tenga mayor control y poder acceder a las capas MAC. Con ello se podría realizar, por ejemplo, operaciones de detección de portadora para sincronizar los distintos LPS en lugar de utilizar un módulo de radio adicional.
- Configuración del número de balizas de las que el LPS consta, de esta forma se haría el sistema más flexible de cara a otros LPS.
- Configuración de balizas activas a emitir, puede facilitar estudios de algoritmos de posicionamiento sobre los LPS.

Parte III

Bibliografía

1. P. Harrop, G. Holland and R. Das, “Real Time Locating Systems (RTLS) 2012-2022: Forecasts, Players, Opportunities”, ed. IDTechEx, 2012.

2. S. Boonsriwai, and A. Apavatjrut, “Indoor WIFI localization on mobile devices,” in 10th International Conference on Electrical Engineering, Electronics, Computer, Telecommunications and Information Technology (ECTI-CON), pp. 1-5, 2013.

3. J. Winter, and C. Wengerter, “High resolution estimation of the time of arrival for GSM location”, in 2000 IEEE 51st Vehicular Technology Conference Proceedings, vol. 2, pp. 1343-1347, 2000.

4. J. M. Villadangos, J. Ureña, M. Mazo, A. Hernández, C. De Marziani, A. Jiménez and F. J. Álvarez, “Improvement of Cover Area in Ultrasonic Local Positioning System Using Cylindrical PVDF Transducer”, in IEEE International Symposium on Industrial Electronics (ISIE 2007), pp. 1473-1477, 2007.

5. A. Mike, C. Rupert, H. Steve, N. Joe, S. Pete, W. Andy and H. Andy, “Implementing a Sentient Computing System”, in IEEE Computer Magazine, vol. 34, no. 8, pp. 50-56, 2001.

6. D. Ruiz, E. Garcia, J. Ureña, D. de Diego, D. Gualda and J.C. Garcia, “Extensive Ultrasonic Local Positioning System for navigating with mobile robots”, in 2013 10th Workshop on Positioning Navigation and Communication (WPNC), pp. 1-6, 2013.

7. Xilinx, Inc., “MicroBlaze Processor Reference Guide”, User Guide UG081, 2008.

8. Digilent, Inc., “GenesysTM Board Reference Manual”, Product Specification, 2013.

9. Xilinx, Inc., “Virtex-5 Family Overview”, Product Specification DS100, 2009.

10. TP-LINK Technologies Co., Ltd., “TL-WR702N 150Mbps Wireless N Nano Router”, User Guide, 2013.

11. Dr. Rainer Mautz “Indoor Positioning Technologies”, Institute of Geodesy and Photogrammetry, Department of Civil, Environmental and Geomatic Engineering, ETH Zurich ,2012.