UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada

MASTER EN TECNOLOGÍAS DE TELECOMUNICACIÓN
TRABAJO FIN DE MÁSTER

DESARROLLO DE LA FUNCIONALIDAD BÁSICA DE LA CAPA MAC EN UN SIMULADOR WSN BASADO EN EL ESTÁNDAR IEEE 802.15.4

AUTOR: JAVIER NAVARRO ESPINO

TUTORES: DR. ROBERTO ESPER-CHAÍN FALCÓN
DR. FÉLIX B. TOBAJAS GUERRERO

FECHA: JULIO 2015

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Desarrollo de la funcionalidad básica de la capa MAC en un Simulador WSN basado en el estándar IEEE 802.15.4

Autor: D. Javier Navarro Espino

Tutores: Dr. Roberto Esper-Chaín Falcón
Dr. Félix B. Tobajas Guerrero

Fecha: **Julio 2015**

IUMA
t +34 928 451 086 | iuma@iuma.ulpgc.es
f +34 928 451 083 | www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Desarrollo de la funcionalidad básica de la capa MAC en un Simulador WSN basado en el estándar IEEE 802.15.4

HOJA DE FIRMAS

Alumno: D. Javier Navarro Espino Fdo.:

Tutor: Dr. Roberto Esper-Chaín Falcón Fdo.:
Tutor: Dr. Félix B. Tobajas Guerrero Fdo.:

Fecha: **Julio 2015**

IUMA
t +34 928 451 086 | iuma@iuma.ulpgc.es
f +34 928 451 083 | www.iuma.ulpgc.es
Campus Universitario de Tafira
35017 Las Palmas de Gran Canaria

UNIVERSIDAD DE LAS PALMAS DE GRAN CANARIA
Instituto Universitario de Microelectrónica Aplicada
Sistemas de información y Comunicaciones

Máster en
Tecnologías de Telecomunicación

Trabajo Fin de Máster

Desarrollo de la funcionalidad básica de la capa MAC en un Simulador WSN basado en el estándar IEEE 802.15.4

HOJA DE EVALUACIÓN

Calificación: .....

Presidente: Fdo.:

Secretario: Fdo.:

Vocal: Fdo.:

Fecha: Julio 2015

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

ÍNDICE DE CONTENIDOS

Índice de contenidos I

Índice de figuras III

Acrónimos VII

Capítulo 1. Introducción 1

1.1. Antecedentes 2

1.1.1. Simulación 5

1.1.2. Entornos de simulación actuales 7

1.1.3. Contexto de desarrollo 11

1.2. Objetivos 11

1.3. Peticionario 12

1.4. Estructura del documento 12

Capítulo 2. Contexto estructural 15

2.1. Propósito 16

2.2. Estándar IEEE 802.15.4 17

2.2.1. Arquitectura 18

2.2.2. Primitivas 19

2.3. Capa PHY 20

2.3.1. Diseño 20

2.3.2. Implementación 22

2.3.3. Verificación funcional 27

Capítulo 3. Desarrollo MAC 35

3.1. Capa MAC 36

3.2. Diseño e implementación 37

3.2.1. Módulo MAC 37

3.2.2. Módulo MAC_CPU 40

3.2.3. Módulo GET_PHY_REQ 44

3.2.4. Módulo ARBITER 48

3.2.5. Módulo BD 49
3.2.6. Archivo auxiliar de definiciones 51

Capítulo 4. Verificación funcional 53

4.1. Verificación funcional MAC 54
4.1.1. ARBITER Test 54
4.1.2. BD Test 55
4.1.3. GET_PHY_REQ Test 57
4.1.4. CPU Test 60
4.1.5. MAC Test 63

4.2. Verificación funcional MAC+PHY 64

Capítulo 5. Conclusiones y líneas futuras 69

5.1. Conclusiones 70
5.2. Líneas futuras 70

Bibliografía 73

ANEXOS

Anexo I 77
Anexo II 79
Anexo III 85
Anexo IV 93

ÍNDICE DE FIGURAS

Figura 1 – Topologías de red 18

Figura 2 – Arquitectura de dispositivo LR-WPAN 19

Figura 3 – Primitivas de servicio 19

Figura 4 – Modelo de propagación 21

Figura 5 – Modelo de propagación (Estructura) 21

Figura 6 – Módulo PHY (Input/Output) 23

Figura 7 – Módulo PHY (Arquitectura) 24

Figura 8 – Módulo CCA (Input/Output) 24

Figura 9 – Módulo ED (Input/Output) 25

Figura 10 – Módulo TX (Input/Output) 25

Figura 11 – Módulo RX (Input/Output) 26

Figura 12 – Módulo CPU (Input/Output) 26

Figura 13 – Modelo simulado 27

Figura 14 – Enlace punto a punto 28

Figura 15 – Módulo rx_Processor (Input/Output) 29

Figura 16 – Módulo AIR (Input/Output) 29

Figura 17 – Archivo de definiciones, código de test y resultado 30

Figura 18 – Resumen de funcionamiento 31

Figura 19 – Detalle del retardo 31

Figura 20 – Detalle de funcionamiento (I) 32

Figura 21 – Detalle de funcionamiento (II) 32

Figura 22 – Modelo de referencia capa MAC 36

Figura 23 – Módulo MAC (Input/Output) 38

Figura 24 – Módulo MAC (Estructura interna) 39

Figura 25 – Módulo MAC_CPU (Input/Output) 40

Figura 26 – Módulo GET_PHY_REQ (Input/Output) 44

Figura 27 – FSM1 (Diagrama de estados) 45

Figura 28 – FSM2 (Diagrama de estados) (I) 46

Figura 29 – FSM2 (Diagrama de estados) (II) 47

Figura 30 – Módulo ARBITER (Ejemplo de funcionamiento) 49

Figura 31 – Módulo ARBITER (Input/Output) 49

Figura 32 – Módulo BD (Input/Output) 50

Figura 33 – FSM_BD (Diagrama de estados) 50

Figura 34 – ARBITER Test (I) 54

Figura 35 – ARBITER Test (II) 55

Figura 36 – ARBITER Test (III) 55

Figura 37 – BD Test (I) 56

Figura 38 – BD Test (II) 56

Figura 39 – BD Test (III) 57

Figura 40 – GET_PHY_REQ Test (I) 58

Figura 41 – GET_PHY_REQ Test (II) 59

Figura 42 – GET_PHY_REQ Test (III) 59

Figura 43 – CPU Test (I) 60

Figura 44 – CPU Test (II) 61

Figura 45 – CPU Test (III) 61

Figura 46 – CPU Test (IV) 62

Figura 47 – MAC Test 63

Figura 48 – Integración MAC + PHY (Estructura) 64

Figura 49 – Test MAC + PHY (phyCurrentChannel) 65

Figura 50 – Test MAC + PHY (phyChannelsSupported) 65

Figura 51 – Test MAC + PHY (phyTransmitPower) 66

Figura 52 – Test MAC + PHY (phyCCAMode) 66

Figura 53 – Test MAC + PHY (phyCurrentPage) 67

Figura 54 – Test MAC + PHY (phyMaxFrameDuration) 67

Figura 55 – Test MAC + PHY (phySHRDuration) 68

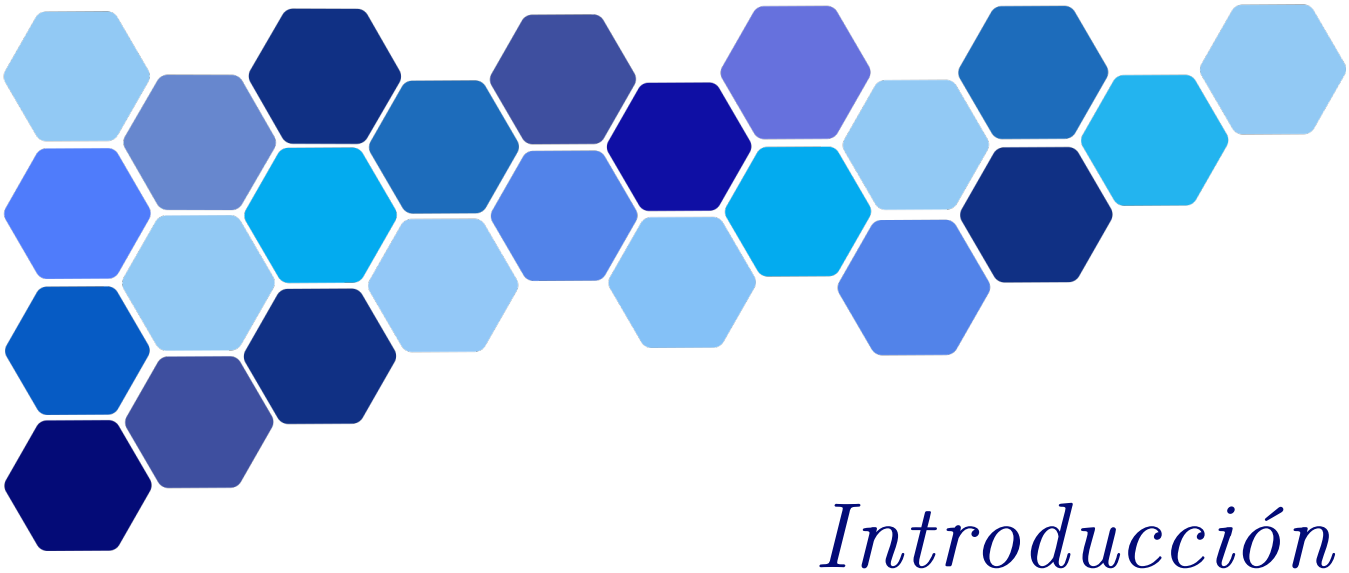
Figura 56 – Test MAC + PHY (phySymbolsPerOctet) 68

ACRÓNIMOS

BER	Bit Error Rate
COM	División de Comunicaciones
CSMA-CA	Carrier Sense Multiple Access with Collision Avoidance
CPU	Central Processing Unit
ED	Energy Detection
FFD	Full-Function Device
GPS	Global Positioning System
GTS	Guaranteed Time Slot
GUI	Graphical User Interface
HDL	Hardware Description Language
IEEE	Institute of Electrical and Electronic Engineers
ISM	Industrial, Scientific and Medical
IP	Internet Protocol
IPv6	Internet Protocol version 6
IUMA	Instituto Universitario de Microelectrónica Aplicada
LQI	Link Quality Indication
LR-WPAN	Low Rate - Wireless Personal Area Network
MAC	Medium Access Control
MCPS	MAC Common Part Sublayer
MLME	MAC Layer Management Entity
MPLS	Multiprotocol Label Switching
NED	NEtwork Description Language
OSI	Open Systems Interconnection
OSPF	Open Shorter Path First
PAN	Personal Area Network
PD-SAP	PHY Data – SAP
PHR	PHY Header
PHY	Physical Layer
PIB	PAN Information Base
PLI	Programming Language Interface

PLL	Phase Locked Loop
PLME	Physical Layer Management Entity
PSDU	Physical Layer Convergence Protocol – Service Data Unit
PPDU	PHY Protocol Data Unit
PPP	Point-to-Point Protocol
P2P	Peer-to-Peer
RF	Radiofrequency
RFD	Reduced-Funcion Device
SAP	Service Access Point
SCTP	Stream Control Transmission Protocol
SFD	Start-of-Frame Delimiter
SHR	Synchronization Header
SSCS	Service Specific Convergence Sublayer
TCP	Transmission Control Protocol
TFM	Trabajo Fin de Máster
UDP	User Datagram Protocol
ULPGC	Universidad de Las Palmas de Gran Canaria
VHDL	Very High Speed Integrated Circuit – HDL
WSN	Wireless Sensor Network

Capítulo 1



Introducción

1.1 Antecedentes

Las redes inalámbricas de sensores, WSN (*Wireless Sensor Networks*), representan una clase de sistemas distribuidos que son parte integral del espacio físico donde son desplegadas. Estas redes están constituidas por nodos, cuya cantidad puede variar desde unos pocos hasta miles de ellos [1]. Típicamente, cada nodo está constituido por una transceptor radio que incorpora o está conectado a una antena, un microcontrolador, una interfaz electrónica para comunicarse con los sensores integrados, y una batería para asegurar su funcionamiento. Individualmente, cada nodo es autónomo y tiene un rango de actuación limitado, si bien, mediante la cooperación con otros nodos, puede cubrir de manera efectiva un área muy superior. El número de nodos variará principalmente según la aplicación para la que vaya destinada la red, pudiendo aumentar o disminuir el rango de actuación de cada sensor según las necesidades de quien realice el despliegue.

Las redes de sensores han captado la atención de muchos investigadores, ya que engloban un amplio espectro de ideas con mucho potencial práctico. A pesar de que se pueden encontrar redes con características muy diversas (tantas como entornos en los que puedan desplegarse), existen una cantidad de elementos comunes a todas ellas. En primer lugar, como se ha comentado, constituyen parte integral del espacio físico que ocupan, estando integradas en el mundo real. En segundo lugar, los sensores detectan la naturaleza física del entorno (intensidad de luz, temperatura, sonido, proximidad, etc.), lo que genera un contraste evidente con la computación tradicional, que por lo general existe exclusivamente en el mundo virtual. Por último, todas las redes de sensores se componen de una colección de nodos, independientemente de su rango, cantidad o prestaciones [1].

Es importante destacar el hecho de que, a diferencia de los típicos dispositivos inalámbricos destinados a los consumidores, donde el objetivo es entregar datos a los usuarios, para las WSN, el objetivo principal de la comunicación es la interacción de los nodos entre sí. De esta manera, los usuarios no son conscientes de los diferentes datos que se transmiten o de las computaciones que se llevan a cabo en cada nodo, sino que son informados en el nivel más alto de los resultados. Así, el objetivo final no tiene por qué ser necesariamente proporcionar un registro completo de la información captada por cada uno de los sensores (*raw data*), sino realizar una síntesis de los mismos que proporcione información útil a alto nivel. Más allá de la conveniencia que tiene establecer esto último como el objetivo principal de una red de sensores inalámbricos, con el paso del tiempo ha pasado a constituir como un principio de diseño de primer orden debido al hecho de que los desarrollos realizados de esta manera han demostrado ser más eficientes energéticamente, siendo la energía el recursos más valioso del que disponen este tipo de redes.

A nivel científico, existen ciertos desafíos que afrontar y mejorar, tales como la eficiencia energética de las redes, entre la cual se incluye la eficiencia en la comunicación (procesado local, colaboración jerárquica o conocimiento del dominio). Un sistema perfecto es aquel que reduce los datos a transmitir a los mínimos imprescindibles y los transmite tan pronto como sea posible, en lugar de enviar toda la información captada por la red (sea útil o no), lo que implicaría un elevado gasto energético. Otro desafío fundamental es el dinamismo de las redes. Este aspecto se basa en la idea de que, tarde o temprano, algún nodo fallará (falta de energía, sobrecalentamiento, sabotaje, accidentes...). Incluso los nodos que se encuentran en posiciones fijas (presumiblemente más estables y seguros) experimentan variaciones intrínsecas en la calidad de los enlaces de RF (*Radiofrequency*). Estos cambios dependen en gran medida del entorno y son complicados de predecir. Ante esta situación existen dos alternativas, disponer de un gran número de personas que se encarguen del mantenimiento y uso de la red, o implementar la red de manera que ésta incorpore cierto grado de autonomía y autoconfiguración. Además, la red debe ser capaz de adaptarse a las alteraciones que se produzcan en su entorno, cambiando constantemente en respuesta a los diferentes eventos que se registren.

Así, en la actualidad, con el objetivo de aumentar las prestaciones y características de las redes inalámbricas de sensores, existen líneas de investigación que exploran diferentes aspectos susceptibles de mejora y que se describen a continuación [1]:

- **Arquitecturas escalonadas (*Tiered architectures*)**: Consisten en un diseño formado por la composición de diferentes elementos seleccionados a partir de decisiones estratégicas sobre factores como las capacidades, los requisitos energéticos, el tamaño, o el precio.

- **Enrutamiento y procesado intrarred (*Routing and in-network processing*)**: El enrutamiento es un concepto que aparece inmediatamente desde el momento en que la red es lo suficientemente extensa como para que un par de nodos no se encuentren directamente conectados. Este concepto puede ser asociado de manera instantánea al concepto de enrutamiento propio de las redes de ordenadores y conocido desde Internet, si bien, las redes de sensores, en su búsqueda de eficiencia, demandan que se realice la mayor cantidad de procesado posible en la propia red. En lugar de encaminar los paquetes de forma "ciega", muchas aplicaciones realizan procesado en cada uno de los saltos dentro de la red, añadiendo datos similares, filtrando información redundante, etc.

- **Localización automática y sincronización temporal**: Los nodos en una red necesitan ser capaces de detectar su posición una vez han sido desplegados. Esta necesidad radica en dos aspectos fundamentales; si se realizan grandes despliegues, localizar manualmente todos los nodos es impracticable, y en el caso de tratarse de una red dinámica, la posición de los nodos puede ser diferente después del despliegue. Una

posible solución vía GPS (*Global Positioning System*) se revela poco práctica en la mayoría de las situaciones debido a que los receptores son costosos en términos de consumo de energía, tamaño y coste. Además de la localización, la sincronización temporal resulta crucial para aquellas situaciones en las que se produzcan variaciones temporales o que estudien fenómenos móviles, de cara a combinar las observaciones de múltiples sensores con el fin de obtener los resultados adecuados.

- **Procesado de señal distribuido**: En el campo del procesamiento de señales, la integración transparente de señales provenientes de múltiples fuentes ha sido una línea de investigación durante muchos años. Estas líneas han conducido al concepto de procesamiento centralizado, un concepto que es absolutamente incompatible con las redes de sensores (por ejemplo, en un procesador centralizado los datos pueden ser compartidos entre los sensores si se dispusiera de un ancho de banda efectivo infinito). En las redes de sensores, la comunicación resulta costosa en términos de energía, limitada en ancho de banda, no trivial en cuanto a su enrutamiento y poco fiable temporalmente, tanto a corto como a largo plazo. Así, en las WSN, este procesamiento se realiza de manera distribuida en nodos independientes que disponen de relojes independientes y cuya sincronización temporal ha de ser explícita.

- **Almacenamiento, búsqueda y recuperación**: Las redes de sensores pueden generar un gran volumen de datos. Las suposiciones habituales para bases de datos acerca de restricción de recursos, características de las fuentes de datos o fiabilidad y disponibilidad, no se pueden llevar a cabo en el contexto de las WSN. Los datos no pueden ser transmitidos y almacenados en un repositorio central sin incurrir en un impacto altamente negativo en las prestaciones de la red. Una alternativa consiste en almacenar estos datos en cada nodo, pero tampoco resulta viable debido a que es posible que estén operando durante largos periodos de tiempo sin que sean atendidos, y las limitaciones de almacenamiento disponible en los nodos resultan evidentes en el caso de tener restricciones de tamaño. Además de estas restricciones en cuanto a almacenamiento, hay que tener en cuenta también las relativas a procesamiento y memoria, que afectan en este caso a las posibilidades de búsqueda y recuperación de datos, o a su procesamiento en tiempo real.

- **Actuación**: Normalmente, las WSN son sistemas totalmente pasivos, capaces de detectar el estado de un entorno pero incapaces de realizar cambios sobre él o sobre la relación que establece con él la red. La actuación extiende las capacidades de una red de dos formas diferentes: En primer lugar, puede mejorar las tareas de detección (por ejemplo, orientando una cámara o apuntando una antena) y en segundo lugar, puede afectar al entorno (por ejemplo, abriendo válvulas, emitiendo sonidos, etc.).

- **Simulación, monitorización y depuración**: Los entornos de simulación y depuración son esenciales en cualquier proyecto de despliegue de WSN a gran escala.

Para un ingeniero de sistemas, existe una paradoja intrínseca al propio despliegue de la red: Los datos que deben ser descartados para no comprometer las restricciones de energía o de capacidad del canal son necesarios para realizar la evaluación y depuración del propio proceso de reducción de datos.

- **Seguridad y privacidad**: El hecho de que los sensores se encuentren repartidos por el entorno implica un problema inmediato de seguridad física. Los posibles atacantes o saboteadores pueden manipular el hardware de los nodos, reemplazarlos con componentes maliciosos o conseguir que los sensores capten información errónea de aquello que observan. Además, los recursos limitados en el caso de los nodos más pequeños implican problemas en cuanto a privacidad, en la medida en que, la mayor parte de los algoritmos criptográficos consumen una gran cantidad de recursos de procesamiento, por lo que el consumo de energía, memoria y tiempo de procesamiento puede provocar que los datos tengan que ser enviados sin protección alguna a través de la red.

El presente Trabajo Fin de Máster (TFM) se engloba en la línea de investigación relativa a la mejora de los entornos de simulación de WSN, basados en el estándar IEEE 802.15.4.

1.1.1 Simulación

En general existen tres tipos de técnicas para realizar la evaluación de las prestaciones de un sistema o de una red: análisis matemático, medidas, o simulación por ordenador. Todas estas técnicas tienen sus ventajas e inconvenientes, pudiendo encontrarse en la literatura múltiples discusiones acerca de cuándo usar cada alternativa [2].

Una de las mayores cuestiones al evaluar las prestaciones de un sistema es cuándo utilizar el sistema real o un modelo del mismo. Resulta evidente que para realizar medidas es necesaria una implementación del sistema pero en ocasiones, resulta conveniente tanto el análisis como la simulación debido a razones de coste y esfuerzo. Estas dos técnicas se basan en un modelo que representa el sistema objeto de estudio de la manera más precisa posible. En el caso del análisis matemático, muchas veces sólo proporciona pequeñas revelaciones acerca del sistema diseñado (los modelos matemáticos en ocasiones resultan inabordables). No obstante, se suele aplicar la simulación para poder comparar diferentes alternativas de diseño o para optimizar ciertos modelos. En investigación y desarrollo de sistemas de comunicación, las dos últimas técnicas son las más importantes durante la fase conceptual, debido a que implementar estos sistemas es prácticamente imposible como consecuencia de las restricciones financieras y técnicas. Particularmente, la simulación se utiliza para sistemas altamente dinámicos y cuyas propiedades son difíciles de modelar de forma matemática. En ocasiones, los métodos

analíticos muestran los extremos del comportamiento del sistema para cubrir objetivos específicos de investigación. Sin embargo, un refinamiento del análisis conlleva habitualmente un incremento inaceptable de la complejidad de dichos modelos.

Por el contrario, la simulación permite a científicos e investigadores disponer de un entorno controlado en el cual el sistema puede ser analizado con mayor detalle. Se pueden controlar diferentes parámetros y escenarios con un esfuerzo considerablemente menor. De esta manera, la simulación representa una metodología potente y versátil para analizar y visualizar el comportamiento y las prestaciones de los sistemas y redes de comunicación [2].

Considerando la simulación como la metodología para analizar las características y prestaciones de un sistema de comunicación (el cual puede que ni siquiera exista), debe considerarse también un hecho muy importante: todas las simulaciones se realizan sobre modelos del sistema bajo investigación y no sobre el propio sistema. Todos los modelos deben ser creados a priori. Analizar un sistema sin un modelo previo es a todas luces imposible y por tanto se debe asumir que toda la información extraída de la simulación proviene del modelo desarrollado, por lo que el proceso de modelado resultará crucial para toda evaluación basada en simulaciones [2].

En una simulación, en definitiva, un proceso o sistema del mundo real es "imitado" a lo largo de un tiempo, utilizándose en diferentes ámbitos y disponiendo de diferentes variantes, tales como simulaciones de eventos discretos, simulaciones continuas, simulaciones de Monte Carlo, etc. De manera genérica y recapitulando, un simulador debe cubrir los siguientes aspectos:

1. Ser paradigma para analizar modelos complejos.
2. Permitir experimentar con sistemas cuya implementación no resulta práctica ni viable.
3. Imitar la operación del sistema real a lo largo del tiempo
4. Inferir conclusiones acerca del sistema real.

Centrando estos aspectos en lo que corresponde a las WSN, se puede obtener una serie de características idóneas para un simulador para este tipo de redes:

- Estudiar una problemática concreta (a nivel de red, protocolos, despliegue...).
- Detectar inconvenientes no contemplados con anterioridad.
- Mejorar la eficiencia en el despliegue de la red desarrollando soluciones más inteligentes.

- Evitar la estrategia prueba/error y reducir los costes del despliegue.

1.1.2 Entornos de simulación actuales

Los simuladores de redes son elementos necesarios para caracterizar y evaluar, entre otros aspectos de diseño, diferentes protocolos de comunicación. De forma general, en la actualidad este tipo de simuladores presentan una serie de inconvenientes que no se encuentran resueltos, entre los que destacan los siguientes:

- Orientados a simulación de eventos discretos.
- Funcionamiento a muy alto nivel.
- Principalmente orientados a modelos de colas.
- La inclusión de nuevos protocolos resulta dramática puesto que están diseñados para evaluar aquellos que ya se encuentran incorporados.

En los siguientes apartados se analizan las principales características de los dos simuladores de redes mayoritariamente utilizados hoy en día, OMNeT++ y Network Simulator, así como las principales deficiencias detectadas en cada uno de ellos a partir del análisis realizado.

OMNeT++

OMNeT++ es un simulador de eventos modular y extensible que utiliza un modelo de protocolos por capas e incluye un entorno gráfico de desarrollo y simulación orientado principalmente al desarrollo de protocolos propios. Esto hace que represente más un entorno de simulación que un simulador propiamente dicho, buscando ser lo más generalista posible. Los modelos se pueden implementar a partir de diferentes componentes y librerías y es por esto que se desarrollan de manera completamente independiente al marco de simulación, siguiendo sus propios ciclos de lanzamiento al mercado [2][3].

OMNeT++ proporciona una arquitectura genérica de componentes y se deja bajo responsabilidad del desarrollador del modelo la tarea de mapear los diferentes elementos, tales como dispositivos de red, protocolos o canales inalámbricos. La estructura modular que resulta de este planteamiento funciona de alguna manera como bloques Lego. Estos módulos se comunican a través de mensajes, por conexiones directas o predefinidas, pudiendo representar estos mensajes eventos, paquetes, comandos, tareas o entidades, en función del modelo.

Una parte importante de OMNeT++ es su entorno de simulación basado en Eclipse. El

objetivo de utilizar un entorno de este tipo es convertirse en plataforma de integración de utilidades y herramientas de simulación de terceros.

Así, el flujo de desarrollo de un modelo realizado con OMNeT++ es el siguiente:

1. Construir el modelo a partir de módulos que se comunican entre sí intercambiando mensajes.
2. Definir la estructura en lenguaje NED (*NEtwork Description Language*).
3. Programar los módulos en C++.
4. Describir un archivo de inicialización para mantener la configuración y los parámetros del modelo.
5. Desarrollar el programa de simulación y ejecutarlo. Se puede desarrollar con línea de comando o con la interfaz gráfica (*Graphical User Interface*, GUI).
6. Obtener y analizar los resultados.

Los entornos de simulación de redes ya desarrollados para OMNeT++ son:

- **INET**: Contiene protocolos como UDP (*User Datagram Protocol*), TCP (*Transmission Control Protocol*), SCTP (*Stream Control Transmission Protocol*), IP (*Internet Protocol*), IPv6 (*Internet Protocol version 6*), Ethernet, PPP (*Point-to-Point Protocol*), IEEE (*Institute of Electrical and Electronic Engineers*) 802.11, MPLS (*Multiprotocol Label Switching*) y OSPF (*Open Shorter Path First*).
- **INETMANET**: Extiende INET con soporte para redes móviles *ad-hoc*.
- **OverSim**: Es un marco de simulación P2P (*Peer-to-Peer*) de código abierto. Contiene varios modelos para sistemas P2P estructurados y desestructurados. Está basado también en INET.
- **MiXiM**: Soporta simulaciones inalámbricas y móviles. Proporciona modelos detallados del canal inalámbrico, conectividad inalámbrica, modelos de movilidad, modelos para obstáculos y varios protocolos de comunicación especialmente en el nivel MAC (*Medium Access Control*).
- **Castalia**: Es un simulador para WSN, redes de área personal y en general para redes de dispositivos empotrados de bajo consumo.

Sin embargo, las principales deficiencias que aparecen al utilizar OMNeT++ como simulador de WSN son las siguientes:

1. Imposibilidad de describir modelos concurrentes.
2. El código generado resulta desproporcionado.
3. No está orientado a comportamiento.
4. Está totalmente desarrollado en C++.
5. El compilador resulta extremadamente complejo.
6. La incorporación de nuevos protocolos conlleva una gran carga de trabajo.

Network Simulator (*ns-3*)

La herramienta *ns-3* es un simulador libre de eventos discretos que continua la familia iniciada por *ns* y *ns-2*, con la peculiaridad de que no es compatible con sus antecesores, lo que quiere decir que aquellos desarrollos realizados en las versiones anteriores no pueden ser utilizados en *ns-3*. Se usa principalmente en investigación y enseñanza e incorpora modelos de colas, generadores y consumidores. El objetivo principal del desarrollo de *ns-3* fue mejorar el realismo de los modelos, acercándolos a las implementaciones software reales que representan. Los lenguajes de modelado de alto nivel hallan precisamente en este aspecto uno de sus puntos débiles, y la experiencia de autores y desarrolladores indica que a mayor grado de abstracción, mayor es la divergencia encontrada entre los resultados de la simulación y los resultados experimentales. Esta última versión de *ns* está escrita desde cero utilizando C++ y Python [2][4].

Este simulador proporciona modelos para los siguientes elementos de red:

- **Nodos**: Pueden ser terminales, enrutadores, concentradores y conmutadores.
- **Dispositivos**: Representan los dispositivos físicos que conectan un nodo con el canal de comunicación.
- **Canales de comunicación**: Representan el medio utilizado para enviar la información entre dispositivos de red.
- **Protocolos de comunicación**: Modelan la implementación de las descripciones de los protocolos.
- **Cabeceras de protocolos**: Son subconjuntos de los datos que integran los paquetes de red, teniendo formatos específicos para cada protocolo al que están asociados.
- **Paquetes de red**: Representan la unidad fundamental de intercambio de información en redes.

El flujo de desarrollo es el que se muestra a continuación:

1. **Definición de la topología de red**: Consiste en la instanciación de objetos C++ para los nodos, dispositivos, canales y protocolos de red que estén siendo modelados en la simulación.
2. **Desarrollo de los modelos y configuración de nodos y enlaces**: Creación de los modelos de simulación de varias aplicaciones de red que envíen y reciban información desde una red y provoquen que diferentes paquetes sean tanto creados como aceptados y procesados.
3. **Ejecución**: Típicamente, la ejecución de la simulación le indica al simulador que debe entrar en el bucle de eventos principal, que leerá y eliminará eventos en un orden temporal predeterminado y de una estructura de datos establecida en el paso anterior del flujo de desarrollo. Este proceso finalizará cuando la lista de eventos se vacíe o se alcance un tiempo de finalización preestablecido.
4. **Análisis de prestaciones**: Se realizará sobre la traza de información generada a partir de la ejecución del programa. Los archivos de trazas normalmente dispondrán de suficiente información para calcular la utilización media de los enlaces sobre el canal de comunicación, el tamaño medio de las colas y el índice de paquetes descartados en las colas.
5. **Visualización gráfica**: *ns-3* incluye la posibilidad de generar animaciones que muestren de forma gráfica el comportamiento de la simulación de red que se haya llevado a cabo. Este último paso permite observar la vocación académica y de investigación que caracteriza a este simulador.

En lo que respecta a las deficiencias que pueden encontrarse en *ns-3* para la simulación de WSN, en primer lugar, cabe destacar que implica una mayor complejidad de aprendizaje que otros simuladores y carece de un gran número de protocolos que sí estaban incluidos en las versiones anteriores del software (*ns-2* y *ns*). Además, se encuentran otra serie de inconvenientes que se enumeran a continuación:

1. Orientado a simulación de operaciones sencillas.
2. Incorporar nuevos modelos implica recompilar el núcleo.
3. No existe retrocompatibilidad con las versiones anteriores.
4. Es necesario describir en C++ el flujo de vida de las variables.
5. Imposibilidad de descripción de modelos concurrentes.
6. La simulación no está orientada a comportamiento.

7. Menor número de protocolos que sus predecesores.

1.1.3 Contexto de desarrollo

Este TFM se engloba dentro de un proyecto de mayor alcance cuyo objetivo final consiste en implementar un simulador de redes inalámbricas de sensores basado en el estándar IEEE 802.15.4. Este estándar define el nivel físico y el Control de Acceso al Medio de redes inalámbricas de área personal con baja tasa de transmisión de datos. El documento en el que se recoge el estándar IEEE 802.15.4 detalla los diferentes parámetros, primitivas de servicio, atributos, constantes, estados y singularidades que deben quedar contempladas en el desarrollo de la funcionalidad básica de la capa MAC que se pretende llevar a cabo en este TFM.

Se continua el trabajo comenzado con el diseño e implementación de la capa PHY (*Physical Layer*) del estándar y se amplía verificando la misma y desarrollando la funcionalidad básica de la capa MAC.

Entre las características del proyecto que engloba este TFM se encuentra la utilización del lenguaje HDL (*Hardware Description Language*) Verilog, atendiendo a las ventajas que proporciona y que serán analizadas en este documento.

1.2 Objetivos

El objetivo principal de este TFM es desarrollar la funcionalidad básica de la capa MAC para un simulador de WSN basado en el estándar IEEE 802.15.4-2006, utilizando para ello el lenguaje de descripción hardware Verilog HDL. Para la consecución de este objetivo principal, el trabajo ha sido dividido en las siguientes tareas:

- Estudio en profundidad del estándar IEEE 802.15.4-2006.
- Estudio de la implementación de la capa física (PHY).
- Simulación y verificación de la capa física.
- Implementación de la funcionalidad básica de la capa de control de acceso al medio.
- Simulación y verificación de los desarrollos correspondientes a la capa MAC.
- Integración entre las capas MAC y PHY.
- Simulación y verificación funcional.

- Extracción y análisis de conclusiones.

1.3 Peticionario

Actúa como peticionario de este TFM la División de Comunicaciones (COM) del Instituto Universitario de Microelectrónica Aplicada (IUMA) de la Universidad de Las Palmas de Gran Canaria (ULPGC), en el contexto de las líneas de investigación promovidas por la citada división.

Por otro lado, la realización de un Trabajo Fin de Máster es requisito indispensable para la obtención del título de Máster en Tecnologías de Telecomunicación por el Instituto Universitario de Microelectrónica Aplicada.

1.4 Estructura del documento

Este documento se divide en cinco capítulos en los que se describe el trabajo realizado, con la estructura indicada a continuación:

- **Capítulo 1. Introducción**:

  Se detallan los antecedentes, objetivos, peticionario y estructura del documento.

- **Capítulo 2. Contexto estructural**:

  Se describe el propósito de este TFM y se analiza el estándar IEEE 802.15.4, profundizando en su arquitectura y primitivas. Se realiza un análisis de la implementación existente de la capa PHY y se detalla la verificación funcional que se ha llevado a cabo.

- **Capítulo 3. Desarrollo MAC**:

  En este capítulo se realiza una introducción a la capa MAC y se describe su diseño e implementación.

- **Capítulo 4. Verificación funcional**:

  En este capítulo se describe el proceso de verificación realizado sobre la capa MAC desarrollada, así como el realizado sobre la integración entre la capa MAC y la capa PHY.

- **Capítulo 5. Conclusiones y líneas futuras**:

  Se detallan las conclusiones extraídas a partir de la consecución de este TFM y sus posibles líneas futuras de desarrollo.

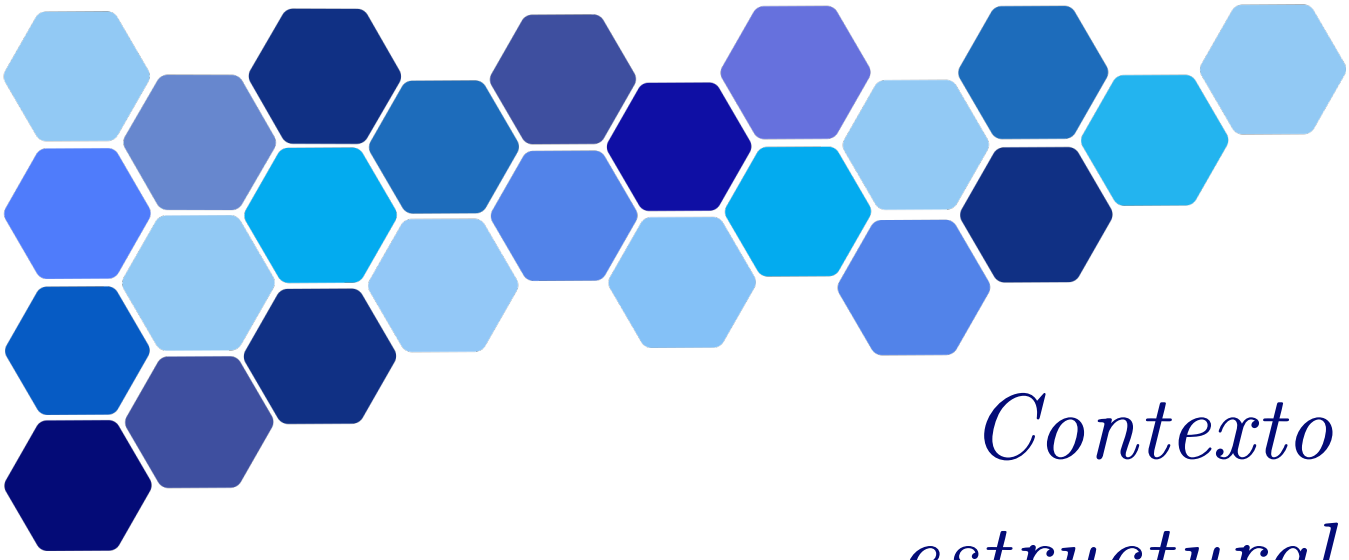
- **Bibliografía**:

  Se incluyen las referencias consultadas para la realización de este Trabajo Fin de Máster y para la elaboración de esta memoria.

- **Anexos**:

  Se han añadido cuatro anexos. Los anexos I y II contienen información de consulta relevante sobre las capas PHY y MAC, respectivamente. Los anexos III y IV contienen la descripción detallada de las interfaces de entrada/salida de los módulos de la capa PHY y de la capa MAC, respectivamente.

Capítulo 2



Contexto estructural

2.1 Propósito

A partir del análisis de las características principales de los simuladores de WSN actuales y de la experiencia previa de los tutores del presente TFM, se puede concluir que las soluciones disponibles resultan demasiado incompletas, ineficaces, simples e incoherentes con el estándar, alejando las simulaciones realizadas de las implementaciones que posteriormente se llevarán a cabo en el mundo real, con el perjuicio que ello puede conllevar.

Bajo la premisa de resolver estos aspectos surge la idea de utilizar lenguajes de descripción hardware como base a partir de la cual desarrollar un nuevo simulador de WSN. Un simulador que proporcione una aproximación más fiable, realista y coherente con el estándar IEEE 802.15.4 de las redes de sensores inalámbricas. Esta elección no es fruto del azar y añade una serie de ventajas para llevar a cabo este propósito:

- Facilitan la descripción de comportamientos.
- Son altamente concurrentes.
- Existen herramientas de desarrollo robustas y multiplataforma.
- Facilitan la integración de los desarrollos.
- Posibilitan realizar operaciones detalladas a nivel de bit.

Como requisitos necesarios para llevar a cabo un simulador de estas características, el lenguaje escogido debe soportar números reales y modelado de expresiones y funciones matemáticas, además de que ya haya sido probado en tareas de modelado. Estas características se pueden encontrar tanto en el lenguaje VHDL (*Very High Speed Integrated Circuit – HDL*) como en el lenguaje Verilog HDL. La decisión de escoger este último responde a la existencia de herramientas de desarrollo de software libre y a la experiencia de los tutores del TFM.

Este TFM parte de la experiencia e implementación previa de la capa física correspondiente al estándar IEEE 802.15.4-2006, con el propósito de verificar dicha implementación y realizar el desarrollo de las funcionalidades básicas de la capa de Control de Acceso al Medio (MAC). El trabajo previo que se ha realizado incluye [6]:

- Implementación completa según se especifica en el estándar para 2450 MHz.
- Implementación sencilla de propagación y atenuación.
- Modelado de interferencias y coexistencias en la red.
- Operación detallada a nivel de bit.

2.2 Estándar IEEE 802.15.4

El estándar IEEE 802.15.4 proporciona las especificaciones para las capas de Control de Acceso al Medio (MAC) y Física (PHY) para las redes inalámbricas de área personal y baja tasa de transmisión, LR-WPAN (*Low Rate – Wireless Personal Area Network*) [5]. Define el protocolo y la interconexión de dispositivos vía radio en una red de área personal. El estándar utiliza CSMA-CA (*Carrier Sense Multiple Access With Collision Avoidance*) como mecanismo de acceso al medio y soporta topologías tipo P2P y estrella.

Las redes LR-WPAN son simples y de bajo coste, permitiendo conseguir conectividad inalámbrica en aplicaciones con restricciones de alimentación y con unos requisitos bajos en cuanto a *throughput*. Los principales objetivos a conseguir son: facilidad de instalación, fiabilidad en la transferencia de datos, rango de operación corto, muy bajo coste, y una vida de batería razonablemente alta, todo ello unido a un protocolo simple y flexible [5].

Algunas de las características de este tipo de redes son:

- Tasa de transmisión de datos "*over-the-air*" de 250 kbps, 100 kbps, 40 kbps y 20 kbps.
- Operatividad P2P y estrella.
- Direcciones cortas de 16 bits o extendidas de 64 bits.
- GTSs (*Guaranteed Time Slots*).
- Acceso al canal de tipo CSMA-CA.
- Protocolo totalmente reconocido para conseguir fiabilidad en las transferencias.
- Bajo consumo de energía.
- ED (*Energy Detection*).
- LQI (*Link Quality Indication*).
- 16 canales en la banda de 2450 MHz, 30 canales en la banda de 915 MHz y 3 canales en la banda de 868 MHz.

Por otra parte, en estas redes pueden participar dos tipos de dispositivos, los dispositivos FFD (*Full-Function Device*) y los dispositivos RFD (*Reduced-Function Device*). Los FFDs pueden operar en tres modos, funcionando como coordinadores de PAN (*Personal Area Network*), como coordinadores, o como dispositivos. Pueden comunicarse tanto con RFDs como con otros FFDs, mientras que los RFDs solamente pueden comunicarse con un FFD. El propósito de los RFDs es participar en aplicaciones

extremadamente simples (por ejemplo, un sensor infrarrojo pasivo), donde no tengan necesidad de enviar grandes cantidades de datos y solo se tengan que asociar con un único FFD cada vez. Consecuentemente, los RFDs pueden implementarse utilizando un número mínimo de recursos. La red WPAN debe incluir como mínimo un FFD que actúe como coordinador PAN.

En la Figura 1 se pueden observar las dos topologías de red contempladas en el estándar.

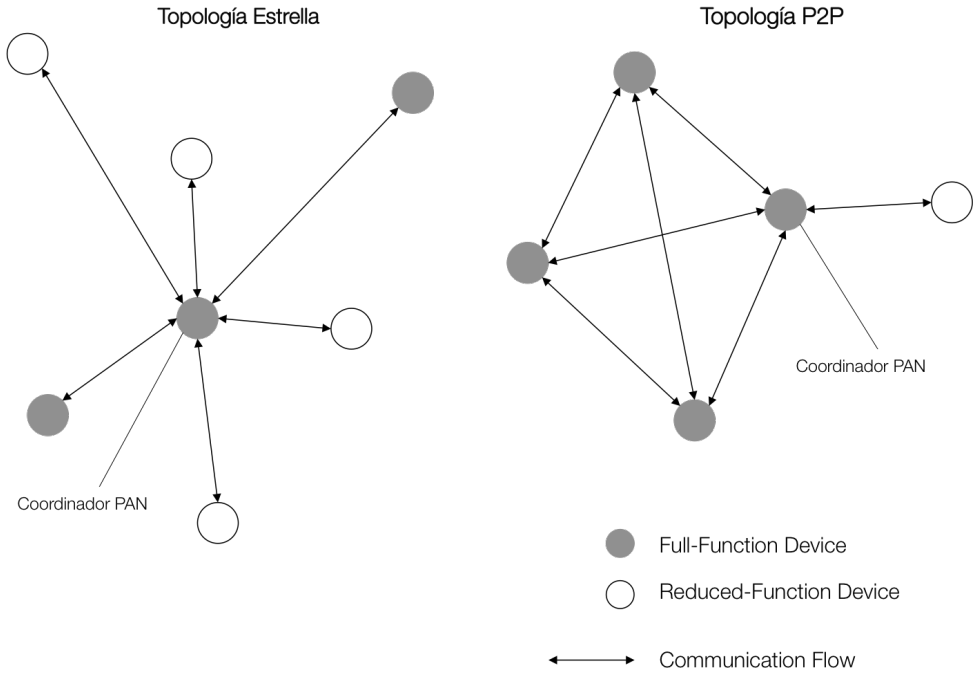


Figura 1 - Topologías de red

2.2.1 Arquitectura

La arquitectura del estándar IEEE 802.15.4 está definida por bloques que se denominan "capas". Cada capa es responsable de una parte del estándar y presta servicio a las capas superiores. La arquitectura se basa en el modelo de capas OSI (*Open Systems Interconnection*).

En la Figura 2 se muestra en mayor detalle la arquitectura de los dispositivos LR-WPAN, observando las interfaces existentes entre capas que sirven para definir los enlaces lógicos descritos en el estándar. Las capas superiores que se observan en la Figura 2 corresponden a la capa de red y la capa de aplicación, que proporcionan configuración de red, enrutamiento y gestión de mensajes, y la función deseada al dispositivo, respectivamente.

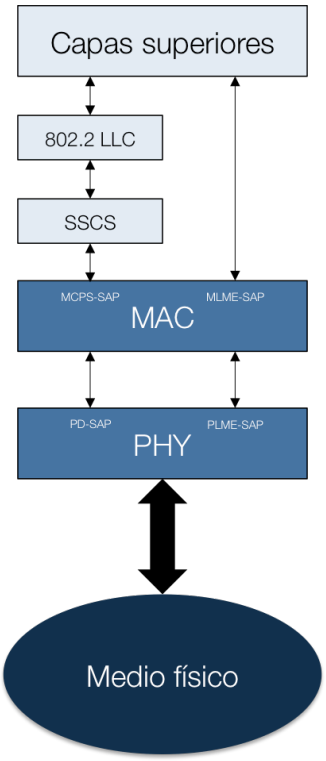


Figura 2 - Arquitectura de dispositivo LR-WPAN

2.2.2 Primitivas

Los servicios de una capa son las capacidades que ésta ofrece al usuario en la capa inmediatamente superior a partir de la inclusión de sus funciones en los servicios de la capa inmediatamente inferior. Este concepto puede observarse con mayor claridad en la Figura 3.

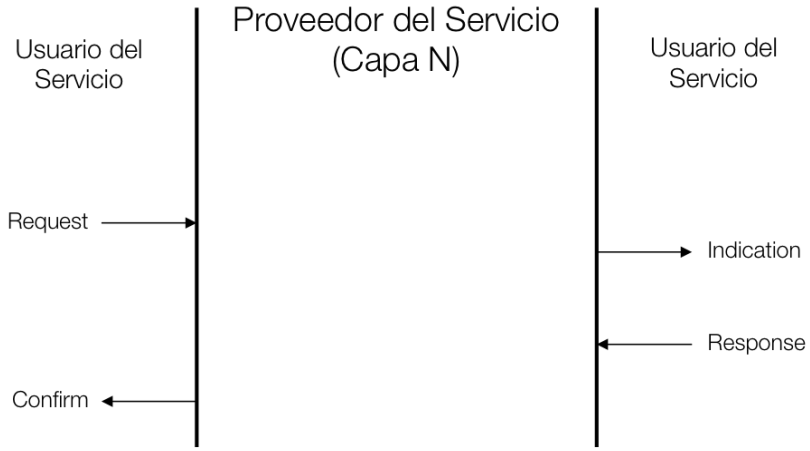


Figura 3 - Primitivas de servicio

Los servicios se especifican describiendo el flujo de información entre el Usuario del Servicio y la Capa N. Este flujo de información es modelado mediante eventos discretos e instantáneos, que caracterizan la provisión de un servicio. Cada evento consiste

en pasar una primitiva de servicio de una capa a otra a través de la capa SAP (*Service Access Point*) asociada al Usuario. Las primitivas de servicio comunican la información requerida proporcionando un servicio particular. Estas primitivas representan una abstracción debido a que únicamente especifican el servicio proporcionado en lugar de la forma mediante la cual se proporciona. Un servicio, por tanto, se describe especificando la primitiva de servicio y los parámetros que lo caracterizan, pudiendo tener una o más primitivas asociadas, lo que constituye la actividad que está asociada a cada servicio particular. Cada primitiva puede tener algún parámetro que comunique la información requerida por el servicio.

Una primitiva puede ser de cuatro tipos genéricos:

- **Request**: Esta primitiva se pasa del usuario a la capa correspondiente para solicitar el inicio de un servicio.
- **Indication**: Esta primitiva se pasa de una capa a un usuario para indicar que un evento interno de la capa debe ser atendido por el usuario.
- **Response**: Esta primitiva se pasa del usuario a la capa correspondiente para completar el procedimiento previo invocado por la primitiva *Indication*.
- **Confirm**: Esta primitiva se pasa de una capa a un usuario para transmitir los resultados de una o más solicitudes de servicio previas asociadas (*requests*).

2.3 Capa PHY

La capa física proporciona fundamentalmente dos servicios: el servicio de datos y el servicio de gestión, que se encarga de servir de interfaz al PLME-SAP (*Physical Layer Management Entity – Service Access Point*). El servicio de datos posibilita la transmisión y recepción de PPDUs (*PHY Protocol Data Units*) a través del canal físico vía radio.

2.3.1. Diseño

El diseño de la capa física ya implementada consiste, básicamente, en dos modelos: un modelo de propagación y un modelo de protocolos. En los siguientes subapartados se van a analizar dichos modelos con el objetivo de comprender el funcionamiento de su implementación.

Modelo de propagación

El modelo de propagación/radio propuesto consiste en una matriz NxN donde N es el número de transceptores TRX. De esta manera, cada puerto RX accede a un multiplexor de N entradas conectado a cada TX. Cada puerto tiene una localización física (x, y, z), lo que permite modelar desplazamientos de los nodos en la red. En las Figuras 4 y 5 se puede observar el modelo descrito.

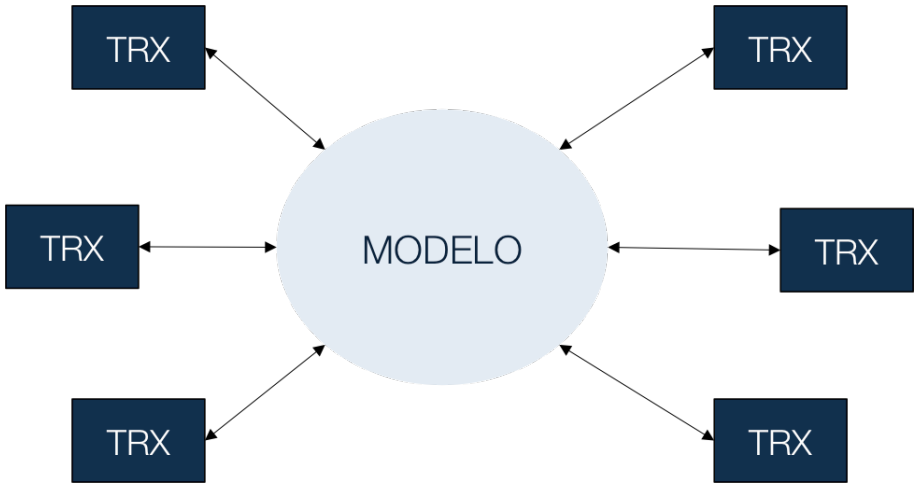


Figura 4 - Modelo de propagación

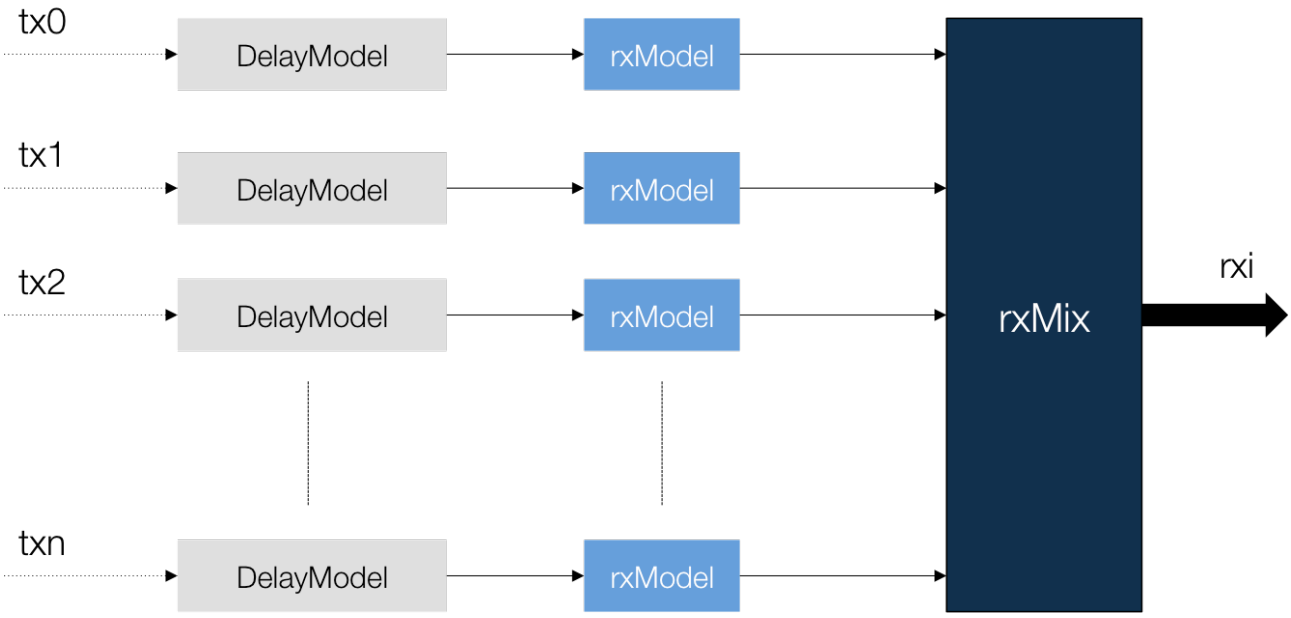


Figura 5 - Modelo de propagación (Estructura)

Los principales elementos que conforman la estructura del modelo de propagación representado en la Figura 5, son:

- **DelayModel**: Con este bloque se incorpora el retardo y la fórmula de Friis al modelo. De la misma manera, existe la posibilidad de modelar obstáculos de diversa índole [6].
- **rxModel**: Se encarga de controlar la modulación, la frecuencia y el ancho de banda para que sea compatible con el modo de escucha del receptor RX. Además, decide el dato que se recibe asignando potencia de señal y de ruido, descartando señales por debajo de un umbral, con el objetivo de reducir el número de eventos.
- **rxMix**: Está diseñado a partir de elementos mezcladores 2 a 1 y se encarga de mezclar datos, potencia de señal y potencia de ruido. Incorpora un modelo simple de interferencias.
- **Modelo de radio**: El modelo de radio dispone de sensibilidad parametrizable y utiliza las frecuencias estándar para los canales 11-26. Incorpora un modelo interpolado de estimación de BER (*Bit Error Rate*) y un generador de errores aleatorios. Se encarga de generar las tramas estándar y modelar PLLs (*Phase Locked Loops*).

Modelo de protocolos

Representa una implementación completa del capítulo 6 del estándar IEEE 802.15.4-2006 en lo que respecta a la frecuencia de 2450 MHz, disponiendo de las siguientes características:

1. Proporciona soporte para parámetros PIB (*PAN Information Base*).
2. Incorpora medida de ED.
3. CCA (*Clear-Channel Assessment*).
4. Modelo de transmisión y recepción TX/RX.

El modelo de comunicación utiliza el puerto paralelo. La comunicación se gestiona mediante colas de mensajes, siguiendo un formato estándar. La medida de ED y el CCA son autómatas individuales, que son instanciados, manteniendo una arquitectura lo más modular posible.

2.3.2. Implementación

La implementación de la capa PHY es, como se ha comentado en el apartado anterior, modular. Existe un módulo principal que está compuesto por una serie de submódulos, encargados de distribuir las responsabilidades correspondientes a los distintos

aspectos especificados en el estándar IEEE 802.15.4.

Módulo PHY

El módulo PHY es el de mayor nivel de la jerarquía, estando compuesto por los submódulos CPU, ED, CCA, RX y TX. Se encarga de definir la interfaz de entrada/salida de la capa física, así como de interconectar los diferentes elementos. En las Figuras 6 y 7 se puede observar la interfaz de entrada/salida del módulo PHY, y una representación global de su arquitectura, respectivamente. En el Anexo III de esta memoria se ha incluido la descripción completa de esta interfaz de entrada/salida.

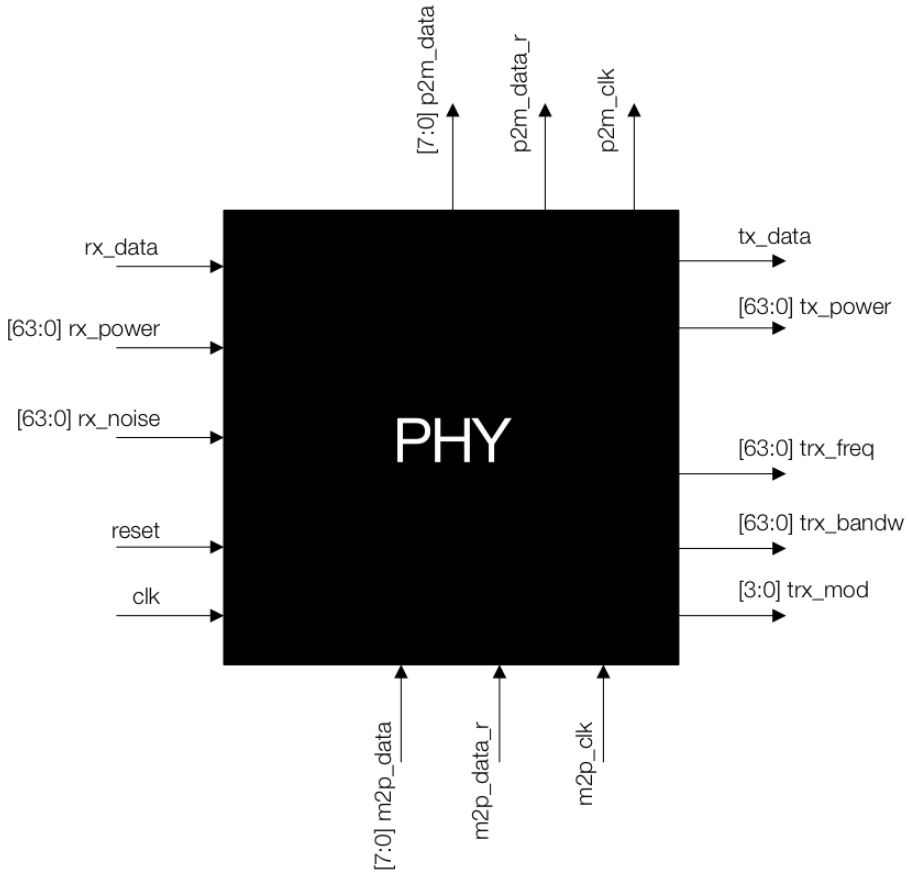


Figura 6 - Módulo PHY (Input/Output)

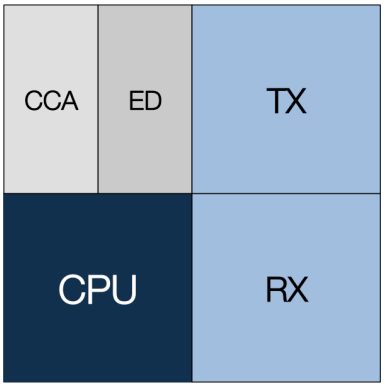


Figura 7 - Módulo PHY (Arquitectura)

Módulo CCA – *Clear Channel Assessment*

Este módulo dispone de tres modos de operación:

- **Mode 1**: Energía por encima del umbral. Debe informar de que el medio está ocupado desde que detecte un nivel de energía superior al umbral.
- **Mode 2**: Sólo detección de portadora. Debe informar de que el medio está ocupado si detecta una señal que cumple con el estándar y tiene la misma modulación y características de propagación de la PHY que esté siendo utilizada en ese momento por el dispositivo.
- **Mode 3**: Detección de portadora con energía superior al umbral.

La interfaz de entrada/salida del módulo CCA se puede observar en la Figura 8 y ha sido incluida de forma detallada en el Anexo III de esta memoria.

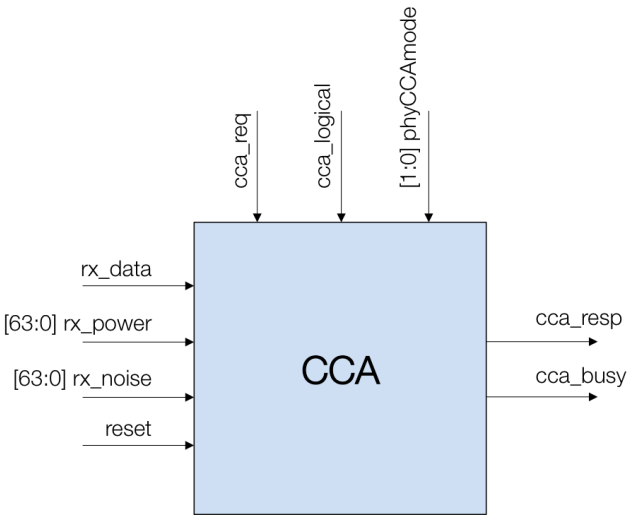


Figura 8 - Módulo CCA (Input/Output)

Módulo ED

El parámetro ED representa una estimación de la potencia de señal recibida. Su resultado debe pasarse a la capa MAC utilizando la primitiva *PLME-ED.confirm*. Se implementa mediante un entero de 8 bits cuyo valor va desde 0x00 a 0xFF. Su valor mínimo indica que la potencia recibida es 10 dB inferior a la sensibilidad especificada para el receptor. En la Figura 9 se representa la interfaz de entrada/salida del módulo ED, incluyendo su descripción en el Anexo III de esta memoria.

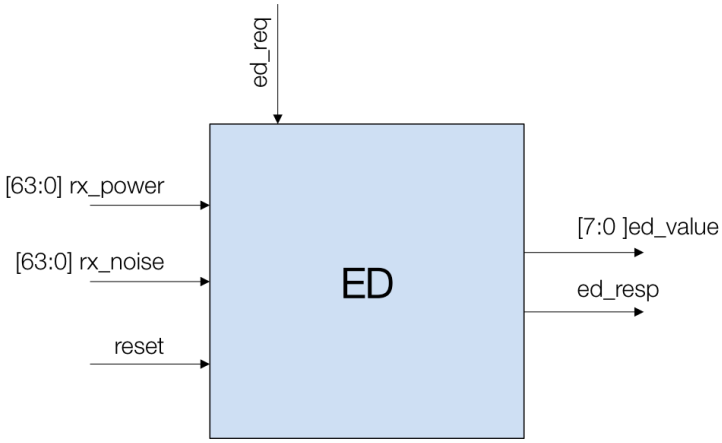


Figura 9 - Módulo ED (Input/Output)

Módulos TX & RX

La responsabilidad de estos dos submódulos es la de realizar todas las tareas asociadas a la gestión de los datos que se desean enviar, así como de los que se reciben, como son controlar las cabeceras, PSDUs (*Physical Layer Convergence Protocol – Service Data Units*), PHR (*PHY Header*), SHR (*Synchronization Header*), SFD (*Start-of-Frame Delimiter*) o los preámbulos. En las Figuras 10 y 11 se muestran las interfaces de entrada/salida de los módulos TX y RX, respectivamente. La descripción detallada de estas interfaces ha sido incluida en el Anexo III de la memoria.

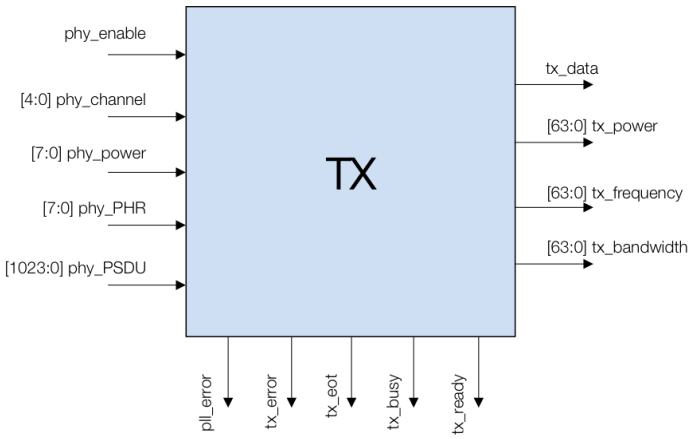


Figura 10 - Módulo TX (Input/Output)

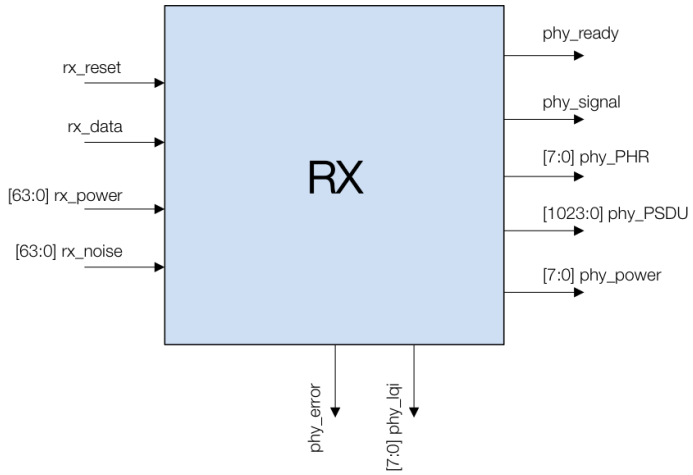


Figura 11 - Módulo RX (Input/Output)

Módulo CPU

Este módulo se ha denominado CPU (*Central Processing Unit*) de forma genérica debido a que es el responsable de organizar y proporcionar inteligencia a todo el conjunto. Se encarga de gestionar las colas de comunicación, gestionar los parámetros PIB y controlar el correcto funcionamiento de la capa física. En la Figura 12 se representa la interfaz de entrada/salida del módulo CPU y su descripción ha sido añadida en el Anexo III de esta memoria.

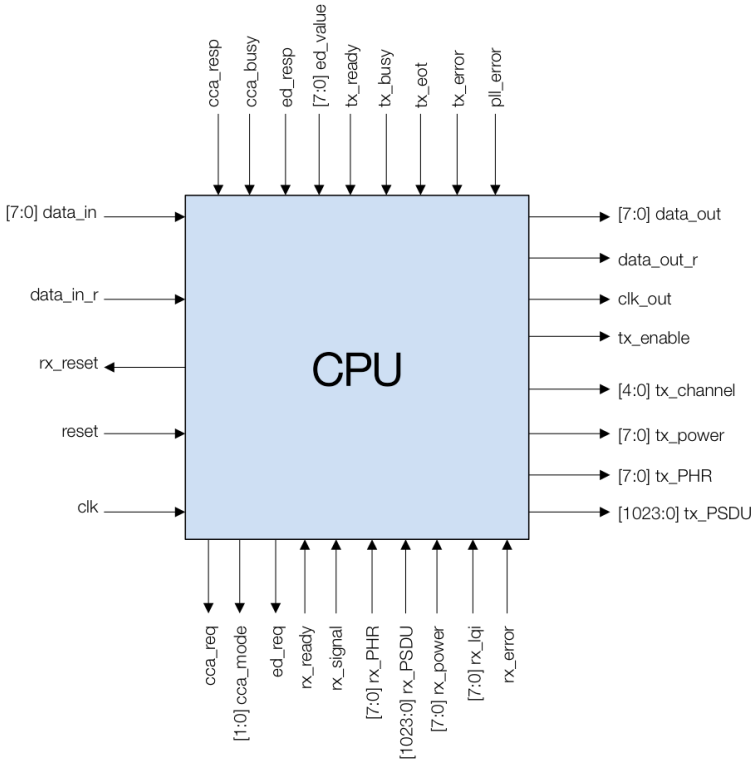


Figura 12 - Módulo CPU (Input/Output)

Archivo auxiliar de definiciones

Además de los módulos especificados anteriormente, para que la capa física pueda funcionar correctamente se ha definido un archivo en el que se han codificado los elementos necesarios especificados en el estándar. Así, las constantes, los parámetros PIB y las directivas de estado (*MAC enumerations*) se han codificado tal y como aparecen en el estándar, si bien las primitivas de servicio se han codificado bajo criterio propio, con el objetivo de simplificar su utilización posterior en cada uno de los módulos que las requieran.

2.3.3. Verificación funcional

De manera general, para realizar las simulaciones necesarias correspondientes a la verificación del funcionamiento de los diferentes módulos desarrollados en este Trabajo Fin de Máster, se han utilizado dos herramientas: Icarus Verilog y GTKwave [7][8].

Para la verificación básica de la funcionalidad de la capa física se ha utilizado este software, unido al desarrollo de un modelo de simulación, compuesto por diferentes módulos. En este apartado se explica dicho modelo, así como el código generado y los resultados globales obtenidos.

Modelo simulado

El objetivo de la verificación funcional básica realizada es establecer un enlace punto a punto, suponiendo que el medio de transmisión es el aire. En la Figura 13 se puede observar el contexto que se desea simular.

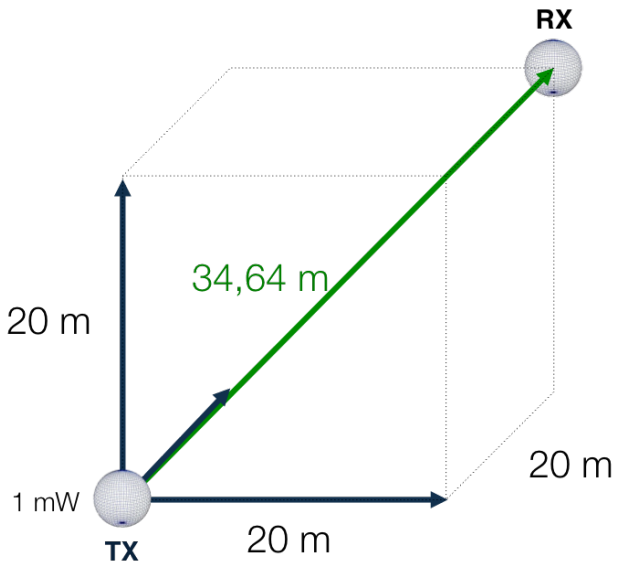


Figura 13 - Modelo simulado

Para llevar a cabo esta simulación se ha establecido una distancia de 34,64 m entre el transmisor y el receptor, suponiendo que el medio de transmisión es el aire y que se emite con una potencia de 1 mW. Se han definido dos módulos, *air* y *rx_processor*. En el caso del módulo *air*, su responsabilidad es establecer el modelo de propagación que se desea para esta simulación. Permite configurar todos los parámetros relativos a la atenuación o a las posiciones del transmisor y del receptor, siendo fácilmente modificables en el caso de simular cualquier otro medio de propagación. En el caso del módulo *rx_processor*, su función es asegurar que la frecuencia, el modo de operación y el ancho de banda de la información que se desea transmitir coincide con lo que el receptor espera recibir. En caso afirmativo, debe permitir el envío de los datos al módulo PHY de recepción.

En la Figura 14 se muestra un esquema modular del contexto que se desea simular.

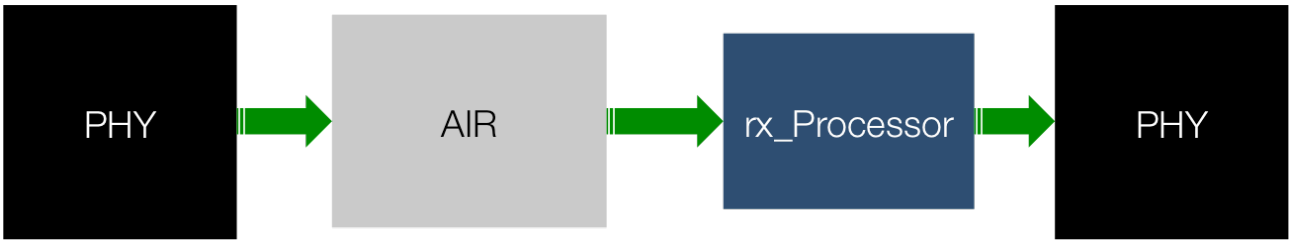


Figura 14 - Enlace punto a punto

Además de este esquema, donde se puede apreciar cómo el módulo PHY situado a la izquierda realiza la función de TX y el que se encuentra situado a la derecha realiza la función de RX, en las Figuras 15 y 16 se incluye la descripción detallada de la entrada/salida de los módulos *air* y *rx_processor*, respectivamente, para lograr una mayor comprensión de los mismos, así como de la arquitectura simulada. Estas interfaces de entrada/salida han sido descritas de forma detallada en el Anexo III de la memoria.

Para continuar con la descripción del proceso de simulación, en la Figura 17 se muestran tres capturas de pantalla correspondientes al archivo de definiciones ".h" (donde se establece la codificación para los diferentes parámetros y primitivas), el código Verilog HDL del testbench realizado, así como la salida del mismo por el terminal de comandos.

Se pueden observar también debidamente señalizados los elementos del archivo de definiciones que entran en juego en esta simulación y su aparición posterior en el resultado de la simulación extraído por el terminal de comandos, comprobándose el correcto funcionamiento de los elementos bajo verificación.

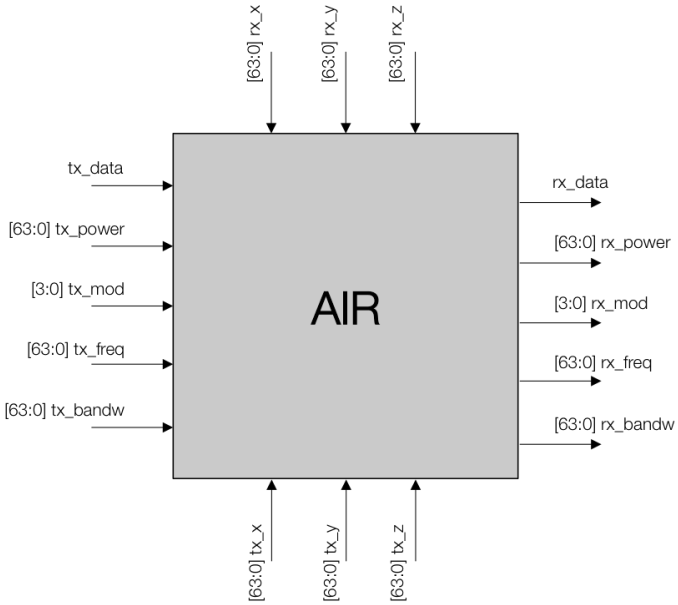


Figura 15 - Módulo air (Input/Output)

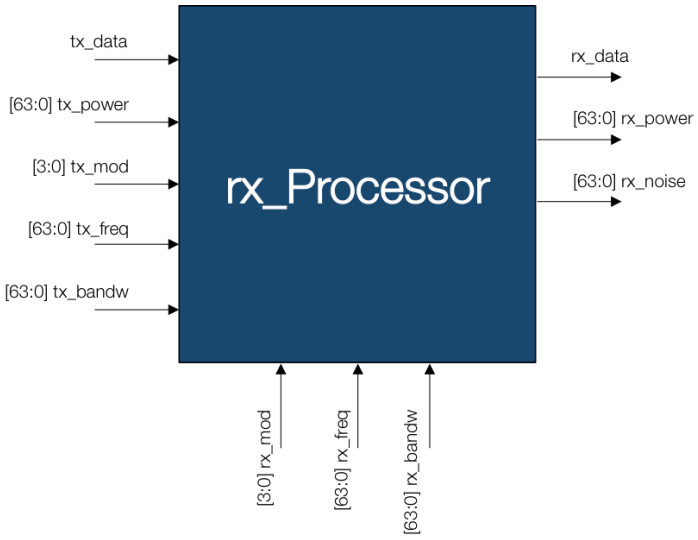


Figura 16 - Módulo rx_Processor (Input/Output)

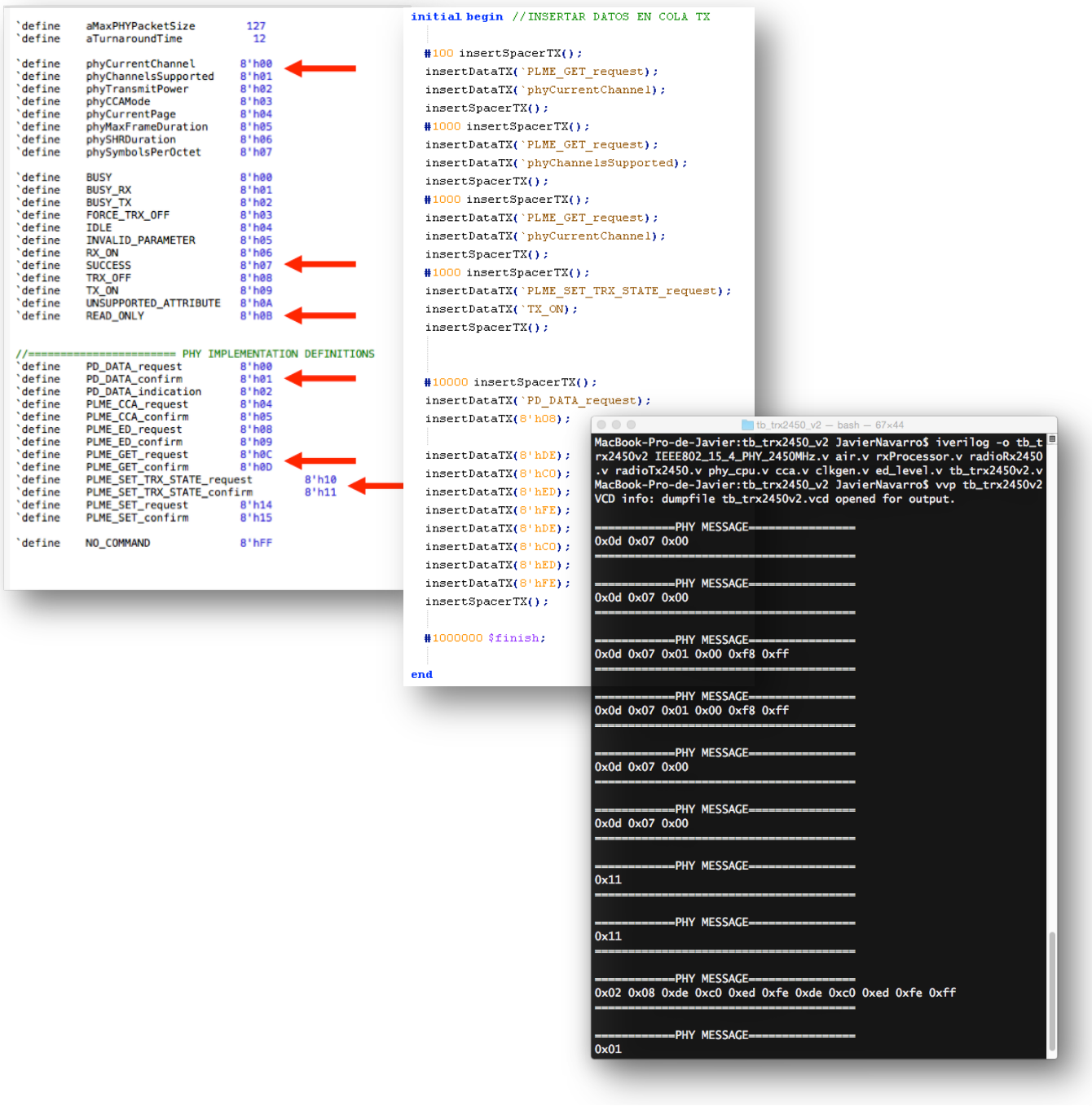


Figura 17 - Archivo de definiciones, código de test y resultado

De manera adicional, también se ha realizado un análisis exhaustivo del comportamiento de las señales implicadas en el desarrollo de la simulación, extrayendo datos de potencia y retardo y variando los parámetros correspondientes en el módulo *air*, con el objetivo de observar sus efectos en los datos analizados. En las Figuras 18, 19, 20 y 21 se muestran las capturas de pantalla correspondientes al resumen de funcionamiento general del sistema, al resultado concreto correspondiente al retardo y al funcionamiento más detallado, respectivamente.

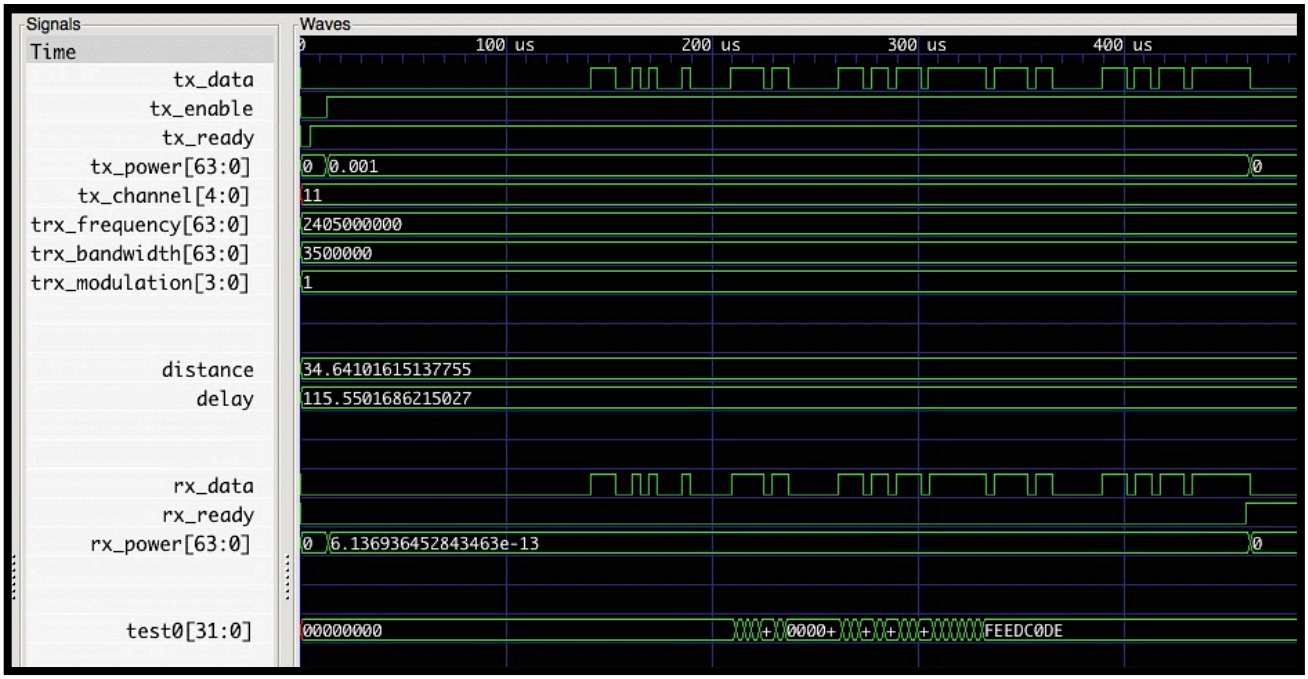


Figura 18 - Resumen de funcionamiento

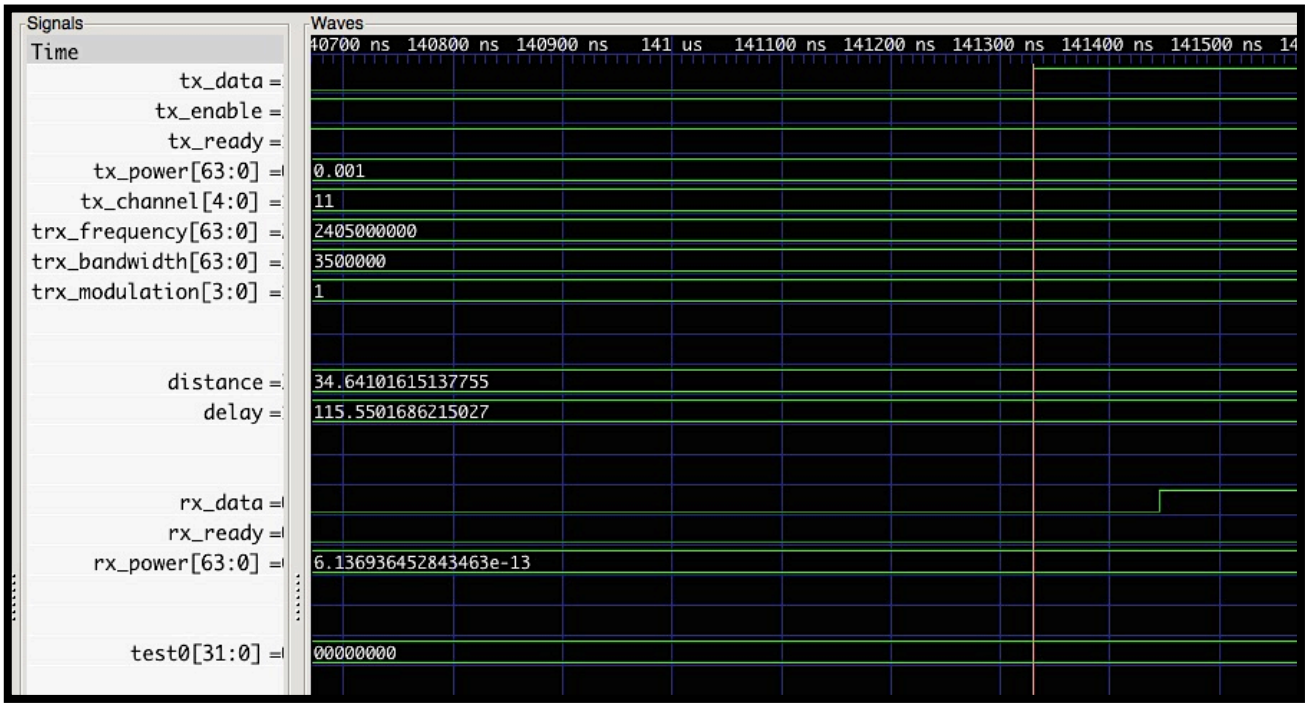


Figura 19 - Detalle del retardo

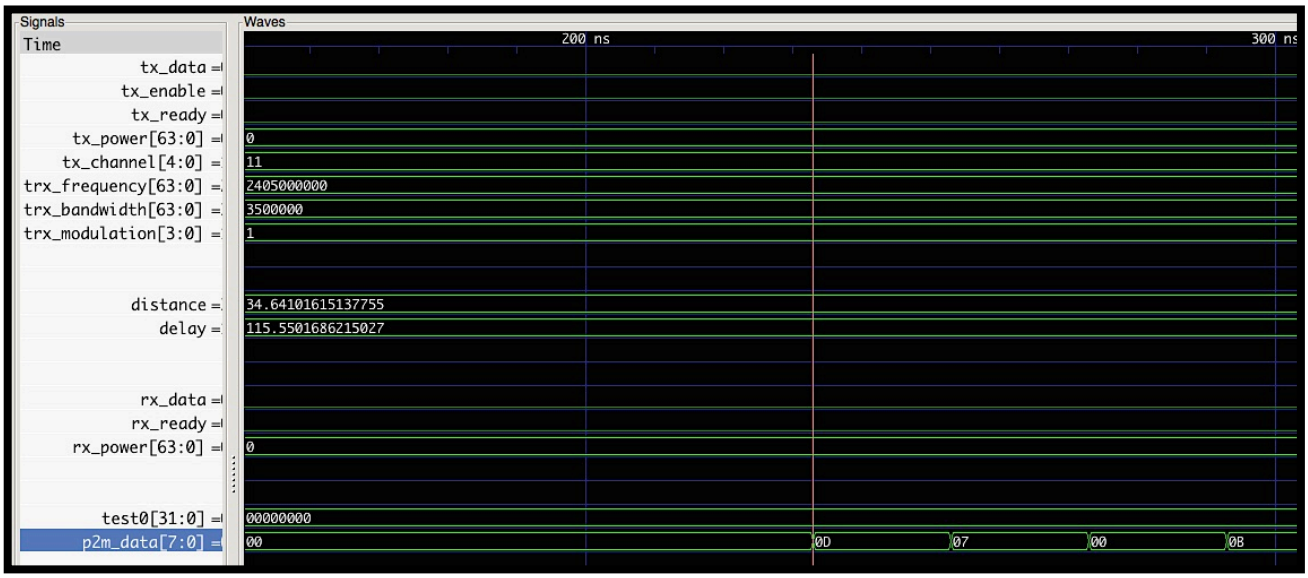


Figura 20 - Detalle de funcionamiento (I)

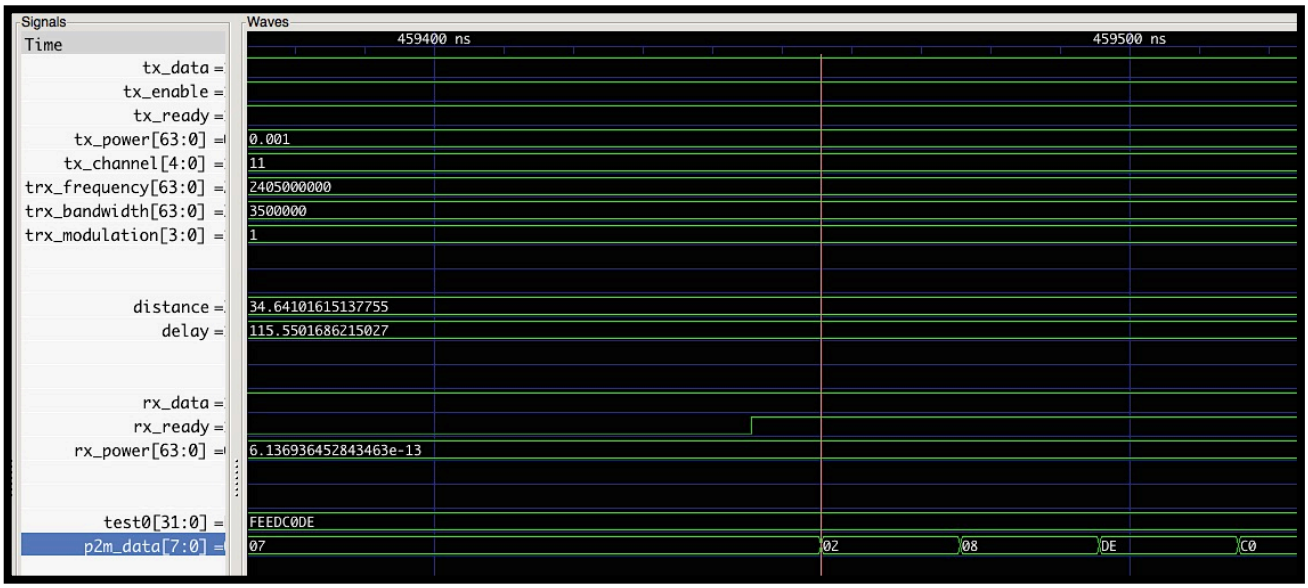


Figura 21 - Detalle de funcionamiento (II)

Así, en la Figura 18 se puede observar el funcionamiento general del sistema. Las señales *tx_data* y *rx_data* muestran los datos enviados y recibidos, comprobándose que la información se transmite y se recibe correctamente. Otras señales que interesa analizar son *tx_power* y *rx_power*, que indican la potencia emitida y la potencia recibida, respectivamente, observándose el decaimiento previsto de la misma. En cuanto a la

distancia y al retardo, el valor de la distancia es el esperado que se había calculado previamente en la Figura 13, mientras que el retardo entre la transmisión y la recepción, a priori no se puede apreciar en esta captura, por lo que se ha añadido la Figura 19, en la que se ha realizado una ampliación de la zona donde comienza la transmisión y la recepción de los datos para comprobar la existencia de dicho retardo entre las señales *tx_data* y *rx_data*. Variando los parámetros de distancia se han obtenido diferentes valores de potencia recibida y de retardo, hasta llegar a la situación en la que no se recibía una señal con potencia suficiente por encima del nivel de ruido, y en consecuencia no se producía la transmisión de los datos.

Además, en las Figuras 20 y 21 se han recogido dos capturas que muestran el detalle de este funcionamiento. Para la primera captura, se ha detallado la respuesta de la capa física a una solicitud de la capa MAC (simulada) de tipo 0x0C | 0x00, tal y como aparece especificado en la Figura 17, una solicitud de tipo *PLME_GET_request* | *phyCurrentChannel*. La respuesta por parte de la capa física (en la señal *p2m_data*) será de tipo 0x0D | 0x07 | 0x00 | 0x0B, que es la respuesta esperada: *PLME_GET_confirm* | *SUCCESS* | *phyCurrentChannel* | *READ_ONLY*. De esta manera queda comprobada que comunicación básica entre la capa física y la capa MAC es la adecuada.

Para la segunda captura se ha escogido el instante en el que se comienzan a enviar los datos, correspondiendo el dato 0x02 con la respuesta al comando *PD_DATA_request* ordenado en el test, concretamente, *PLME_DATA_indication* y el resto de la secuencia, | 0x08 | 0xDE | 0xC0 | ... , reflejando los datos introducidos en el test y que estos se transmiten correctamente.

Capítulo 3

Desarrollo MAC

3.1 Capa MAC

La capa de Control de Acceso al Medio (MAC) gestiona todo el acceso al canal físico de radio y es responsable de las siguientes tareas [5]:

- Generar los *beacons* de la red si se trata del nodo coordinador.
- Sincronizar los *beacons*.
- Proporcionar soporte para la asociación y disociación de la PAN.
- Proporcionar soporte para seguridad.
- Emplear el mecanismo CSMA-CA para el acceso al canal.
- Manejar y mantener el GTS.
- Proporcionar un enlace fiable entre dos puntos.

La capa MAC proporciona una interfaz entre el SSCS (*Service Specific Convergence Sublayer*) y la capa PHY. De manera conceptual, incluye una entidad de organización denominada MLME (*Mac Layer Management Entity*), que proporciona las interfaces de servicio a través de las cuales la capa invoca a las funciones de gestión. La MLME también es responsable de la gestión de la base de datos de aquellos objetos concernientes a la capa MAC. La capa MAC proporciona dos servicios, a los que se accede a través de dos SAPs:

1. **MCPS-SAP (*MAC Common Part Sublayer*)**: Servicio de datos.
2. **MLME-SAP**: Servicio de gestión.

Estos dos servicios posibilitan la interfaz entre el SSCS y la capa PHY, a través del *PD-SAP* (*PHY Data – SAP*) y del *PLME-SAP* vistos con anterioridad. En la Figura 22 se representan los componentes e interfaces de la capa de Control de Acceso al Medio.

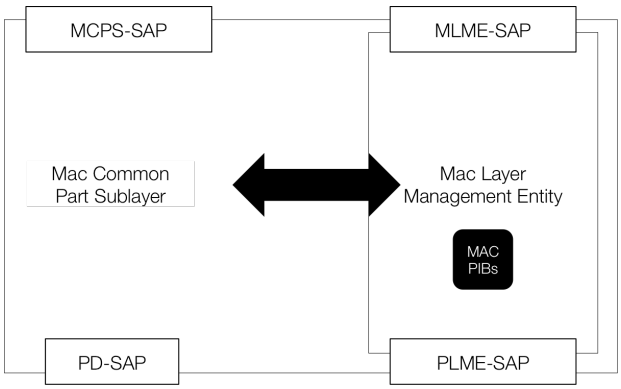


Figura 22 - Modelo de referencia capa MAC

3.2 Diseño e implementación

El diseño de la Capa de Acceso al Medio se ha llevado a cabo teniendo en cuenta el trabajo previo realizado para la capa física. Así, se han respetado las decisiones de implementación tomadas para la capa inferior y se ha mantenido el sistema de colas desarrollado, puesto que se ha considerado como el más adecuado para el funcionamiento que se desea.

Al igual que en la capa PHY, se ha buscado lograr un diseño lo más modular posible, creando módulos independientes con tareas muy bien especificadas, gestionados por un módulo "central" CPU que se encarga de organizar y ordenar las responsabilidades del resto.

En los siguientes subapartados se describen los módulos desarrollados en el presente TFM, con sus funcionalidades específicas, además de mostrarse la estructura general del módulo MAC que los integra a todos. Cabe comentar que, también para la capa MAC, se ha definido un archivo que contiene los parámetros, primitivas y elementos necesarios para simplificar los códigos de comunicación. Como se ha comentado desde el principio de esta memoria, el lenguaje de descripción hardware utilizado para llevar a cabo estas implementaciones ha sido Verilog HDL [9][10].

En este TFM se han implementado de forma completa las primitivas *MLME_GET* y *MLME_RESET*, mientras que la primitiva *MLME_SET* se ha implementado exclusivamente a nivel de la capa MAC.

3.2.1. Módulo MAC

El módulo MAC, incluido en el fichero *IEEE802_15_4_MAC_2450MHz.v* representa el módulo superior de la jerarquía que integra todos los submódulos desarrollados. Este módulo es el encargado de gestionar todo el acceso al canal físico y proporcionar una interfaz entre las capas SSCS y PHY. En la Figura 23 se puede observar la interfaz de entrada/salida del módulo mientras que su descripción ha sido incluida en el Anexo IV de esta memoria. A la izquierda del módulo se representan la entrada de datos y la de solicitud de entrada. Los datos siempre llegarán octeto a octeto, en base al modelo de colas que se ha establecido como consideración de diseño. Por la parte derecha se muestra la salida de datos y las solicitudes de salida, tanto de datos como de comandos. Por último, en la parte inferior del módulo se encuentra la interfaz de comunicación con la capa PHY. Las señales se han denominado "*p2m*" o "*m2p*" para identificar su función "PHY to MAC" o "MAC to PHY", respectivamente, y conservar la coherencia con la interfaz de entrada/salida existente en el módulo PHY.

Cabe destacar que estas últimas señales provienen y se dirigen a un bus de comunicaciones que va situado entre las dos capas y cuyo acceso estará controlado por un módulo árbitro (que será abordado posteriormente). De esta manera se disponen señales de datos, de solicitud de datos y de reloj (para sincronización) en ambos sentidos, en correspondencia con lo establecido en la capa inferior. Las señales de salida hacia el bus son de tipo triestado, con el objetivo de que cuando éstas se encuentren en alta impedancia, dejen de tener relevancia desde el punto de vista del bus compartido.

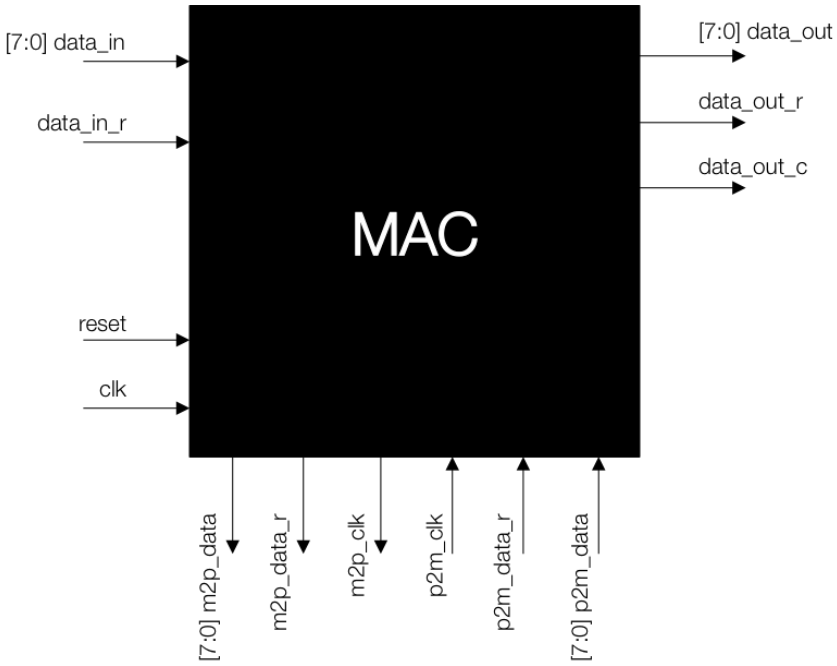


Figura 23 - Módulo MAC (Input/Output)

Para tener una visión más completa del módulo *mac* desarrollado, en la Figura 24 se representa su estructura interna, incluyendo el conexionado entre los submódulos que lo conforman, a saber: *mac_cpu*, *get_phy_req*, *arbiter* y *bd*, que serán analizados en profundidad en los siguientes subapartados.

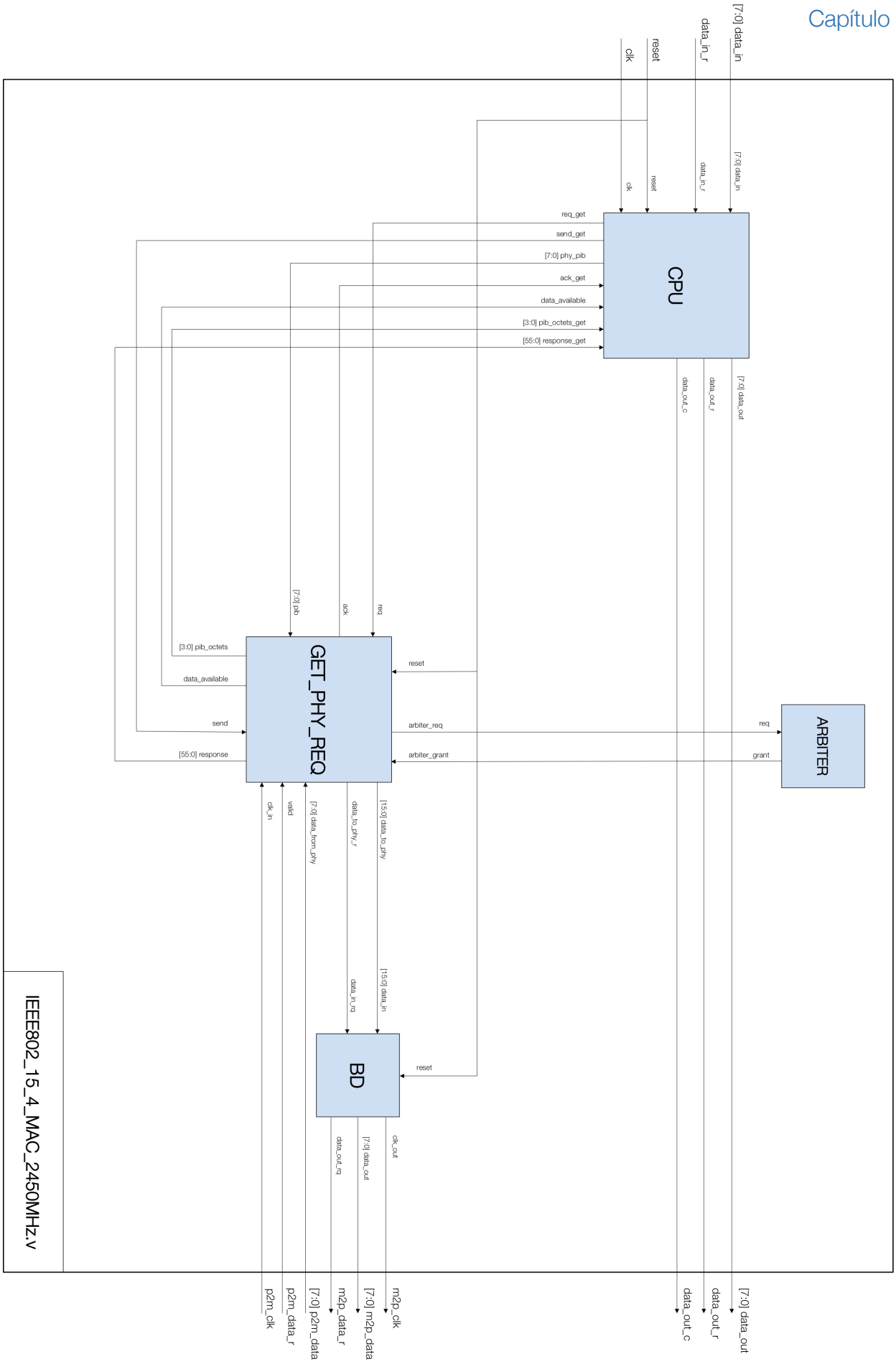


Figura 24 - Módulo MAC (Estructura interna)

3.2.2. Módulo MAC_CPU

Al igual que en la capa PHY, el módulo responsable de organizar y gestionar toda la actividad de la capa MAC se ha denominado *mac_ cpu*. Resulta el módulo de mayor complejidad debido a que debe gestionar las colas de entrada y salida, disponer de un intérprete de comandos que permita resolver las primitivas de servicio que le vayan llegando, y distinguir cuándo las primitivas van dirigidas a la capa física para transferir la información al submódulo correspondiente y esperar por su respuesta, además de conocer los diferentes PIBs (tanto de PHY como de MAC).

En la Figura 25 se puede observar con mayor detalle la interfaz de entrada/salida del módulo *mac_ cpu*. En el Anexo IV de la memoria ha sido incluida una descripción detallada de esta interfaz.

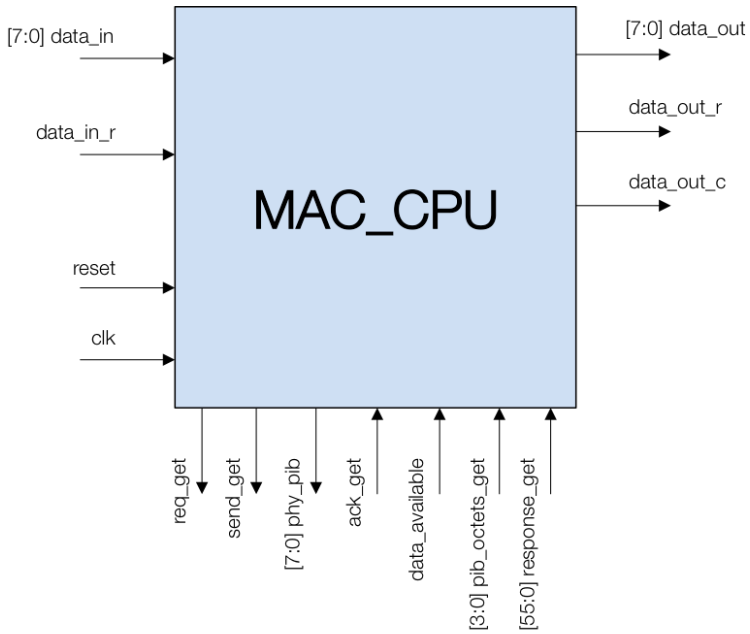


Figura 25 - Módulo MAC_CPU (Input/Output)

Como se ha comentado anteriormente, el funcionamiento del sistema se basa en un modelo de colas mediante el cual la información (datos, comandos, parámetros...) se envía octeto a octeto. A través de una serie de registros se conoce en todo momento cuál es el estado de las colas (de datos y de comandos), la posición del primer elemento y del último ("*head*" y "*tail*", respectivamente), y si se encuentran vacías y/o están disponibles. Todos estos registros contribuyen a la definición de una señal de solicitud interna ("*REQUEST*") que activa o desactiva el funcionamiento de ciertas partes del módulo. Hay que destacar también el sistema de "desencolado" que se realiza con el objetivo de conformar las señales de salida de datos.

En lo que respecta al intérprete de comandos, existen tres elementos clave que aseguran su comportamiento. El primero de ellos es la propia señal interna *REQUEST*, que indica la llegada de datos válidos por el puerto de entrada ("*data_in*"). En segundo y tercer lugar se encuentran dos registros ("*ccount*" y "*cmd_request*") que se encargan respectivamente de gestionar el número de octetos que van llegando por el puerto de entrada y de mantener el primer octeto que llega, correspondiente al comando que debe ejecutar el módulo. Una vez que la señal *REQUEST* ha pasado a nivel bajo, se utiliza una estructura condicional de tipo *case* para comprobar (gracias al registro "*cmd_request*") la primitiva de servicio que se ha solicitado, para así poder llevarla a cabo. Sobre el contador "*ccount*", éste se incrementará una unidad en cada ciclo de reloj mientras la señal "*data_in_r*" de solicitud de datos de entrada permanezca a nivel alto.

En este punto hay que mencionar que de las diferentes primitivas de servicio que existen en la capa MAC y en línea con los objetivos de este TFM, se ha optado por describir las primitivas *MLME_GET*, *MLME_SET* y *MLME_RESET*, estando la primera de ellas completamente descrita, incluyendo su interacción con la capa PHY, mientras que las dos últimas están descritas exclusivamente a nivel de la capa MAC. En los siguientes epígrafes se va a realizar una aproximación a las primitivas desarrolladas con el fin de facilitar su comprensión.

MLME_GET

Esta primitiva solicita información acerca de un PIB dado. Su efecto en recepción es tal que en primer lugar se comprueba si el atributo PIB que se ha recibido es de MAC o de PHY. En caso de recibir un PIB de la capa MAC, la MLME intenta recuperarlo de su base de datos. Si no lo encuentra, se enviará un comando *MLME_GET_confirm* con un estado de *UNSUPPORTED_ATTRIBUTE*. En caso contrario, se enviará un *MLME_GET_confirm* con un estado de *SUCCESS*.

En el caso de recibir un atributo PIB de la capa PHY, se ordenará un comando *PLME_GET_request* y se trasladará la responsabilidad y los datos a la capa física. Una vez que la MLME reciba el comando *PLME_GET_confirm*, traducirá el valor de estado recibido (puesto que no es igual para MAC que para PHY) y posteriormente a esta traducción, se generará un comando *MLME_GET_confirm* con el estado correspondiente.

Como aclaración, la primitiva *MLME_GET_confirm* devolverá el estado y el PIB solicitado, así como su valor, completando de esta forma la solicitud realizada por el comando *MLME_GET_request*.

MLME_SET

Esta primitiva intentará escribir un valor dado por el atributo PIB especificado. Es generada en la capa superior y planificada en la MLME, que en primera instancia comprobará si el atributo es de MAC o de PHY. Si se da el primer caso, se intenta escribir el valor en la base de datos de la MLME. Si el parámetro "*PIBAttribute*" de la primitiva especifica que el atributo es de sólo lectura, el estado será *READ_ONLY*. En el caso de que se especifique un parámetro PIB que no esté en la base de datos, entonces el estado será *UNSUPPORTED_ATTRIBUTE*. Si el parámetro "*PIBAttributeValue*" especifica un valor erróneo, el estado será *INVALID_PARAMETER*. Si todo se desarrolla correctamente, el estado será *SUCCESS*. En todos los casos, el estado irá precedido de un comando *MLME_SET_confirm*, que se encargará de indicar el resultado del intento de escribir el valor de un atributo PIB en la base de datos.

MLME_RESET

Esta primitiva permite a la capa superior solicitar a la MLME que realice un *reset*. Al recibirla, la MLME ordena un "*set*" a los valores por defecto, realizando un "*clean*" de todas las variables internas a sus valores por defecto. Si, además, el parámetro "*SetDefaultPIB*" de la primitiva es *TRUE*, los PIBs de la capa MAC se deben inicializar también a sus valores por defecto. Una vez completado el proceso, se enviará un comando *MLME_RESET_confirm* con un estado *SUCCESS*, informando de esta manera del resultado del *reset*.

Habiendo sido descritas las diferentes primitivas de servicio que se han llevado a cabo, y continuando con el análisis del módulo *mac_cpu*, al entrar en la sentencia *case* que comprueba lo que hay en el registro "*cmd_request*", existiendo tres opciones, que constituirán las tres primitivas analizadas.

En el primer caso, *MLME_GET_request*, como la primitiva tiene dos parámetros, es necesario esperar a que el registro contador ("*ccount*") se incremente hasta el valor 2 para poder comenzar a operar. Cuando el contador toma el valor 1, lo que se hace es mantener el valor que hay en el puerto de entrada (recuérdese que los parámetros se reciben octeto a octeto) en una pequeña memoria denominada "*attQueue*", que almacenará el primer parámetro (y subsiguientes en caso necesario) para su uso posterior.

Cuando el contador toma el valor 2, se comprueba el parámetro almacenado en "*attQueue*", que se corresponderá con el PIB deseado, implementado mediante otra sentencia de tipo *case*. En primer lugar se determina si dicho PIB es perteneciente a la capa MAC o a la capa PHY. Si el PIB es de la capa PHY, se almacenará el identificador del mismo en el registro "*phy_pib*", se activará la señal "*req_get*" de solicitud al módulo encargado de gestionar la comunicación entre MAC y PHY (será analizado en el siguiente

subapartado) y se activará también la señal "*send_get*", indicando que el módulo *mac_cpu* se encuentra preparado y esperando a recibir la información pertinente.

De esta manera, el módulo literalmente esperará a través de un "*wait*" a que se active la señal "*data_available*", a partir de lo cual se establecerá la señal "*send_get*" a nivel bajo y, en función del número de octetos válidos de la señal "*response*", la información será transferida a la cola de salida, quedando disponible en el puerto de salida de datos.

Si el PIB que se ha recibido es de la capa MAC, se actuará en consecuencia al comportamiento contemplado en el estándar, introducido en el epígrafe anterior. Octeto a octeto se irá pasando a la cola de datos y de comandos en primer lugar el comando *MLME_GET_confirm*, en segundo lugar el estado, seguido del identificador del PIB concreto y su valor (si el valor es un entero de 32 bits, habrá que pasar 4 octetos correspondientes sólo al valor, 7 en total). Con esto quedaría descrito el comando *MLME_GET_request* de forma completa, pudiendo gestionar tanto atributos PIB de la capa PHY como de la capa MAC.

En el caso del comando *MLME_SET_request*, hay que remarcar que tiene tres parámetros, por lo cual el valor del contador tendrá que ser mayor que 2 para empezar a operar y en "*attQueue*" se almacenarán dos de los tres parámetros. Bajo esta premisa, se comprueba el parámetro PIB (de la capa MAC) al que se le desea aplicar el comando *SET*.

Una vez comprobado el PIB, se actúa una vez más en correspondencia con el comportamiento definido en el estándar. Octeto a octeto se envía a la cola de datos y de comandos el comando *MLME_SET_confirm*, se almacena en el registro correspondiente al PIB solicitado los datos disponibles en el puerto de entrada (mediante concatenación de los mismos), se envía el estado, el identificador, el índice, y por último, el separador.

Dentro de cada caso, será necesario realizar comprobaciones en los datos que llegan al puerto de entrada para poder saber si dichos datos son válidos, inválidos o se encuentran fuera de rango, con el objetivo de establecer correctamente el estado de la respuesta *MLME_SET_confirm*. Para llevar a cabo esta parte de la descripción se han tenido que analizar uno por uno todos los PIBs posibles de la capa MAC, con sus valores y tipos de datos correspondientes.

Por último, en el caso del comando *MLME_RESET_request*, sólo es necesario realizar una comprobación, puesto que la primitiva consta únicamente de un parámetro, que puede ser *TRUE* o *FALSE*. En caso de ser *TRUE*, además de enviar a la cola el comando *MLME_RESET_confirm* y el estado (*SUCCESS* en todo caso), se inicializa toda la base de datos de PIBs de la capa MAC a sus valores por defecto, tal y como está estipulado en el estándar.

3.2.3. Módulo GET_PHY_REQ

El módulo *get_phy_req* es el encargado de actuar como intermediario para la gestión de la comunicación entre la capa MAC y la capa PHY, y viceversa. Existirá un módulo de este tipo por cada una de las primitivas de servicio de la capa MAC que deban comunicarse o gestionar parámetros PIB relativos a la capa física. Este módulo consta principalmente de dos autómatas que se harán cargo de controlar la asignación de señales y datos, y que serán descritos en los siguientes epígrafes. Además, hay que comentar que, al ser este módulo uno de varios (tantos como primitivas de servicio dependientes de la capa PHY haya), podrá darse el caso de que de forma simultánea varios de ellos deseen acceder a la capa PHY, incurriendo en un potencial problema de acceso. Este problema se ha solucionado estableciendo un sistema de control basado en un árbitro y un bus de comunicaciones que serán analizados en los próximos subapartados.

En la Figura 26 se puede ver el esquema de la interfaz de entrada/salida del módulo *get_phy_req* y en el Anexo IV se ha incluido su descripción.

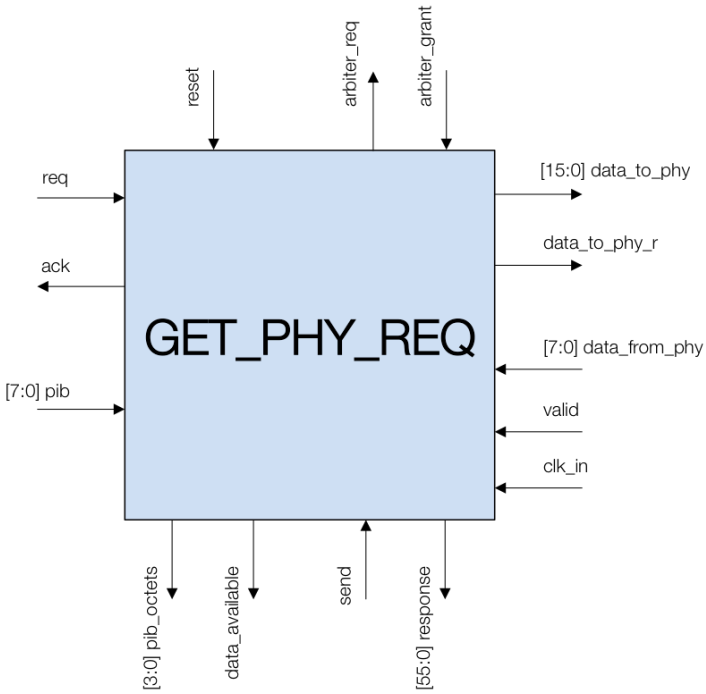


Figura 26 - Módulo GET_PHY_REQ (Input/Output)

Este módulo ha sido diseñado de forma que en el caso de no tener tareas que realizar, entrará en un estado "inactivo". Esto se consigue mediante la activación de los relojes internos de forma externa, a través de una señal se solicitud o directamente introduciendo una señal de reloj. Como se ha comentado en el principio del apartado, el funcionamiento de este módulo está basado en dos autómatas interdependientes entre sí, denominados FSM1 y FSM2.

FSM1

Este primer autómata consta de cuatro estados (*s10*, *s11*, *s12*, *s13*). En la Figura 27 se puede observar su diagrama de estados.

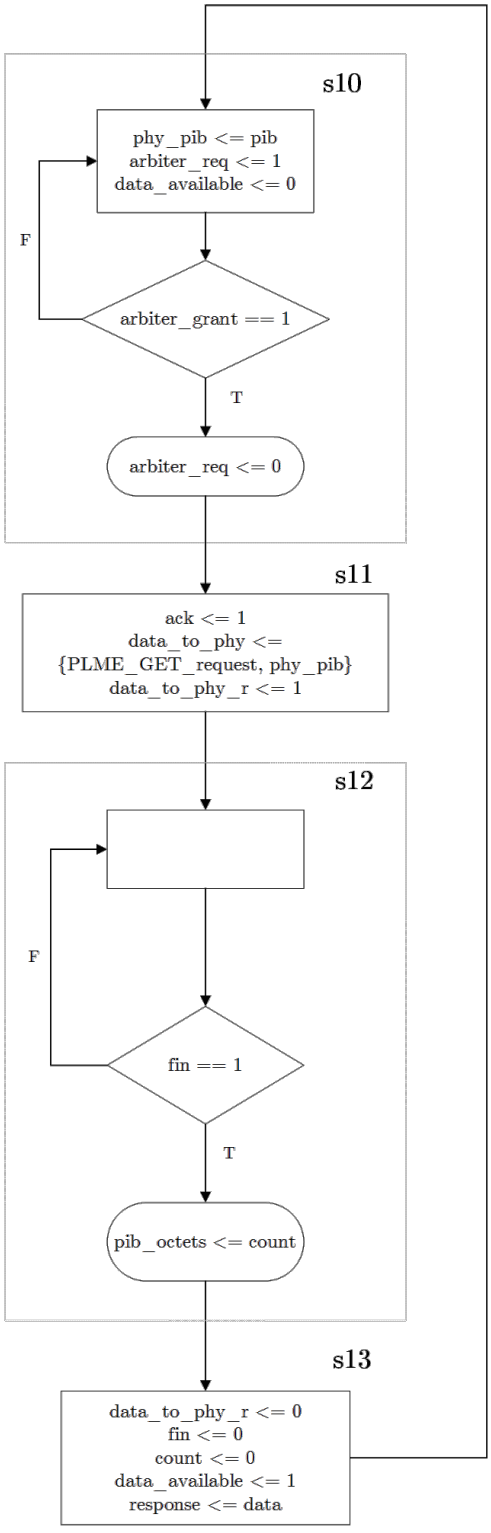


Figura 27 - FSM1 (Diagrama de estados)

En este diagrama pueden observarse de forma clara las diferentes asignaciones y condiciones que se realizan y evalúan a lo largo de la ejecución de la máquina de estados implementada. Más allá del funcionamiento que se aprecia, merece la pena indicar en primer lugar cómo el control del árbitro está contemplado en el estado *s10*. Mientras no se reciba la señal "*arbiter_grant*", la ejecución estará bloqueada. En segundo lugar, en el estado *s12* se puede ver también cómo mientras la señal "*fin*" se encuentra a nivel bajo, la ejecución vuelve a bloquearse. La señal "*fin*" es generada por la segunda máquina de estados, por lo que la finalización de la ejecución de la FSM1 estará supeditada a que la FSM2 genere la señal "*fin*". Por último, es importante mencionar que el número de octetos del PIB es variable. Para resolver esta cuestión se realiza la asignación de la señal "*count*" en "*pib_octets*". En el siguiente epígrafe se abundará más en la solución aportada a este problema.

FSM2

El segundo autómata implementado consta de cinco estados (*s20*, *s21*, *s22*, *s23* y *s24*), siendo el encargado de procesar los datos que llegan de la capa PHY y conformar el paquete con toda la información, así como de calcular el número de octetos del PIB que se ha solicitado. En las Figuras 28 y 29 se muestra su diagrama de estados.

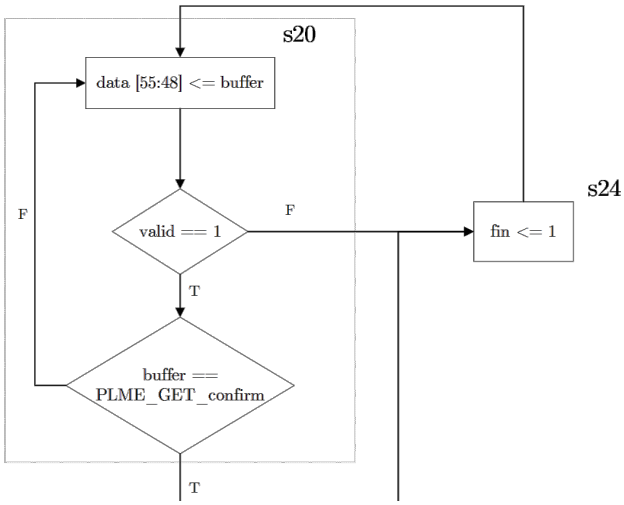


Figura 28 - FSM2 (Diagrama de estados) (I)

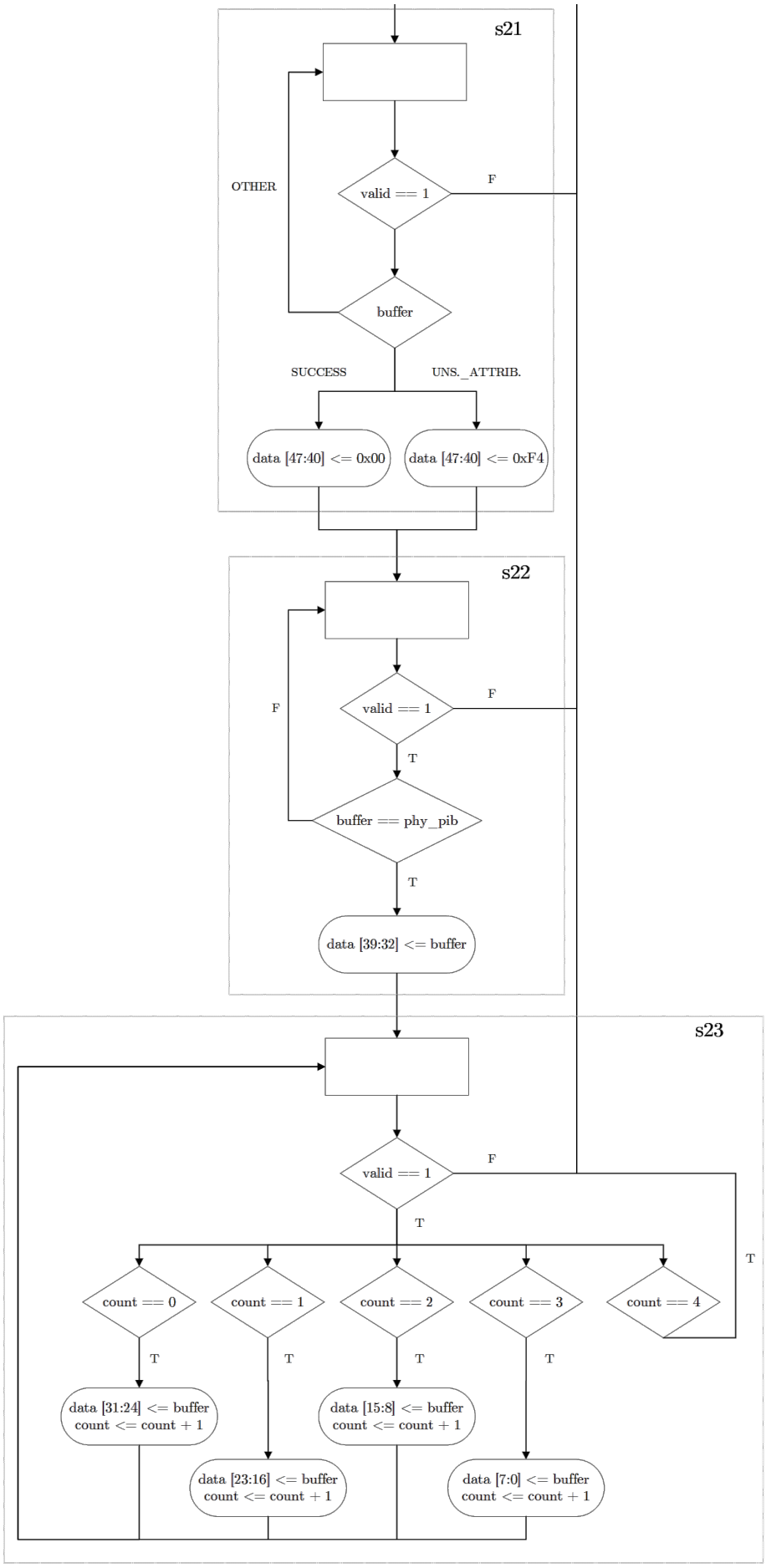


Figura 29 - FSM2 (Diagrama de estados) (II)

Como puede observarse en el diagrama de estados, la señal "*valid*" indicará en todo momento si se deben aceptar los datos que llegan por el puerto de datos, comprobándose al principio de todos y cada uno de los estados del autómata. Además de esto, en cada estado se realizan comprobaciones para asegurar que los datos que llegan son correctos. En el estado *s20* se verifica que llega un comando *PLME_GET_confirm*, en el estado *s21* se verifica que llega un estado *SUCCESS* o *UNSUPPORTED_ATTRIBUTE*, realizando la traducción de los valores de la capa PHY a la capa MAC, y en el estado *s22* se comprueba que el identificador del parámetro PIB es el esperado (se encuentra almacenado en "*phy_pib*").

Finalmente, resta completar el registro de datos con los datos que van llegando hasta que la señal "*valid*" se ponga a nivel bajo, lo que indicará el final de la transmisión por parte de la capa PHY. De esta manera se establece una estructura multicondicional en la que, con ayuda de una variable contadora "*count*", los datos se van pasando octeto a octeto, incrementando el contador hasta un valor máximo de 4. Esto se debe a que el PIB de mayor tamaño que podrá transmitir la capa física tiene 4 octetos, por lo que nunca se sobrepasará dicha cantidad. El contador también servirá para establecer el número de octetos que el módulo *get_phy_req* debe detectar para conformar su salida de datos.

En el estado *s24*, la señal "*fin*" se establece a nivel alto con el objetivo de devolver el control del módulo al autómata FSM1.

3.2.4. Módulo ARBITER

El módulo *arbiter* ha sido desarrollado teniendo en cuenta la estructura del diseño general de la capa MAC. Así, el diseño realizado contempla que, para llevar a cabo las diferentes primitivas de servicio de la capa que requieren interacción con la capa PHY (solicitud y envío de información), se desarrollen módulos independientes encargados de dicha interacción. Ante el potencial problema que surge de la necesidad de acceder a un recurso limitado (único) por parte de varios módulos, resulta vital establecer un sistema de arbitraje que recoja las diferentes peticiones de acceso y conceda los derechos de lectura o escritura en el bus de comunicaciones de forma ordenada.

El funcionamiento del árbitro estará basado en un bucle que recorre todos los bits de la señal "*req*" (conformada por todas las señales de solicitud que provienen de los módulos que desean acceder a la capa PHY) para comprobar cuáles se encuentran a nivel alto e ir estableciendo la prioridad adecuada. Se ha añadido la variable entera "*last_granted*" y el registro "*yet_granted*" para que si el árbitro ya ha concedido el acceso al bus a algún módulo, y éste aún tiene su señal "*req*" a nivel alto, no se le retire hasta que haya terminado. En la Figura 30 se ha añadido un ejemplo de esta situación para que se pueda apreciar con mayor claridad.

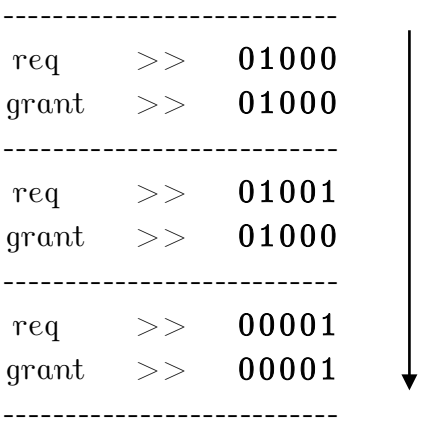


Figura 30 - Módulo ARBITER (Ejemplo de funcionamiento)

Como puede apreciarse, en la primera situación existe una solicitud en el cuarto bit y la señal "*grant*" concede el acceso, puesto que no hay ninguna solicitud más. No obstante, en el segundo caso se observa cómo se recibe una nueva solicitud en el módulo *arbiter* (ahora situada en el primer bit). La señal "*grant*" continúa concediendo el acceso al módulo representado por el cuarto bit, puesto que éste sigue a nivel alto. Por último, en el tercer caso, se ve cómo la solicitud del cuarto bit ha pasado a nivel bajo mientras que la del primer bit se ha mantenido, por lo que ya puede ser concedida por parte del árbitro.

Para acabar con el análisis del módulo *arbiter*, en el Anexo IV de la memoria se ha incluido la descripción detallada de su interfaz de entrada/salida, representado en la Figura 31.

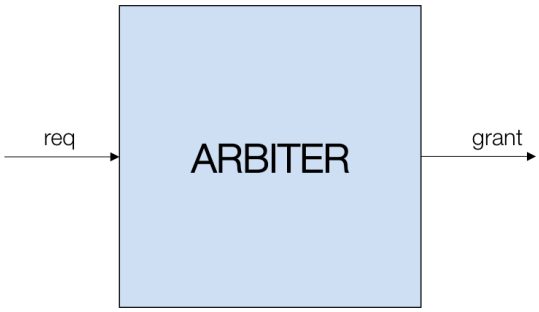


Figura 31 - Módulo ARBITER (Input/Output)

3.2.5. Módulo BD

El módulo *bd* representa el "*bus driver*" que es necesario añadir entre cada uno de los módulos responsables de alguna primitiva que requiera acceso a la capa PHY y el bus de comunicación. Debido a que a la salida del módulo *get_phy_req* la señal "*data_to_phy*" es de 16 bits, ha sido necesario definir una máquina de estados en el módulo *bd* para comunicar los datos al bus octeto a octeto.

En las Figuras 32 y 33 se puede apreciar la interfaz de entrada/salida del módulo *bd*, así como el diagrama de estados del autómata, denominado FSM_BD, para posteriormente realizar el análisis de su funcionamiento. La descripción completa del interfaz ha sido incluida en el Anexo IV de esta memoria.

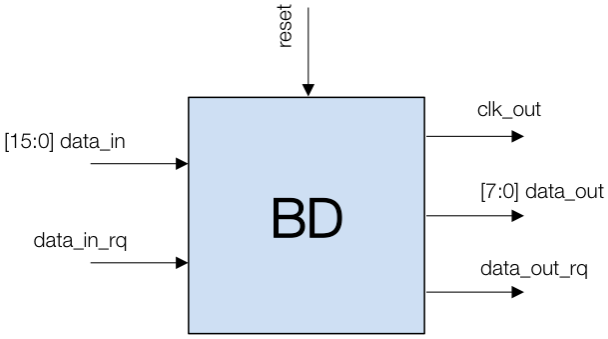


Figura 32 - Módulo BD (Input/Output)

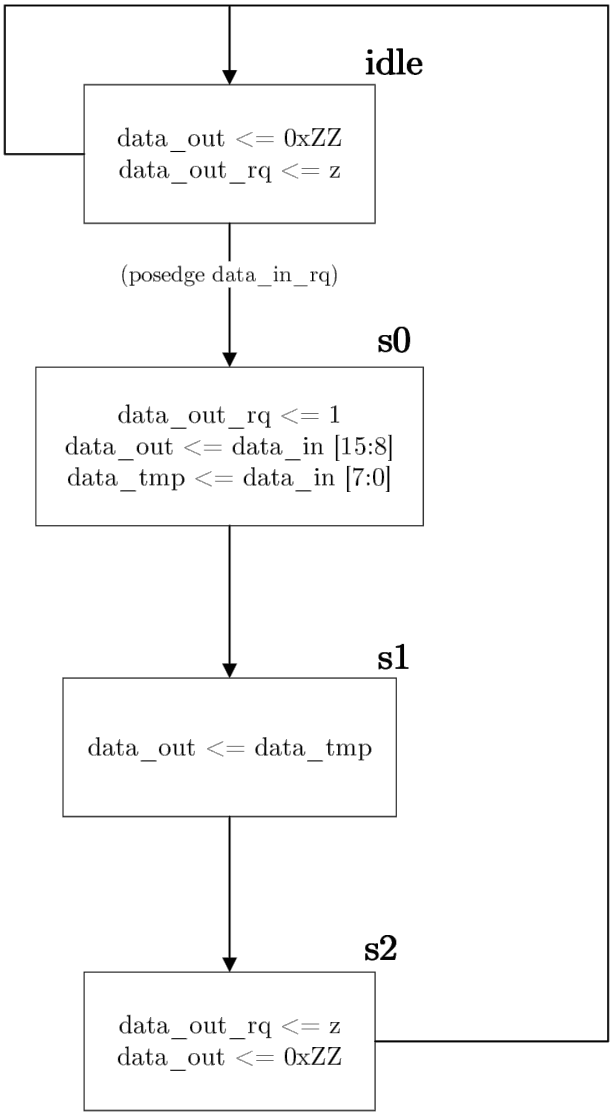


Figura 33 - FSM_BD (Diagrama de estados)

En cuanto a la estructura del módulo, se puede apreciar cómo el funcionamiento está basado en una entrada de datos de 16 bits y señal *request* y una salida de datos de 8 bits, *request* y señal de reloj. El reloj tendrá funciones de sincronismo y será generado de manera interna mediante la instancia del módulo *clkgen*. Es importante destacar que las señales de salida del módulo son triestado, lo que permite establecer los puertos a alta impedancia cuando se desea que ya no tengan relevancia para el bus de comunicaciones.

Como puede observarse, el autómata está compuesto por 4 estados (*idle*, *s0*, *s1* y *s2*), estando inactivo constantemente hasta que detecte un flanco de subida en la señal "*data_in_rq*". A partir de ese momento, los dos estados siguientes serán utilizados para transmitir al bus los dos octetos y el separador que conforman el mensaje que la capa física espera. Una vez realizado esto, el último estado vuelve a situar los puertos en alta impedancia y regresa al estado inactivo inicial.

3.2.6. Archivo auxiliar de definiciones

Al igual que en la implementación de la capa PHY, se ha optado por crear un archivo auxiliar de definiciones con el objetivo de simplificar la utilización de primitivas de servicio, codificaciones de estado, parámetros PIB y constantes propias de la capa. De todos estos elementos, los únicos cuyo valor no viene especificado en el estándar son las primitivas de servicio, a las que se les ha asignado una codificación de 8 bits con un valor comprendido entre 0x00 y 0x23, reservando el valor 0xFF para el caso de que no exista primitiva.

Aunque está especificado en el archivo de definiciones, se debe indicar que para los parámetros PIB "*macMinLIFSPeriod*" y "*macMinSIFSPeriod*" el estándar no aporta información acerca de su codificación, por lo que se ha tomado la decisión de asignarles un identificador observado en la hoja de especificaciones de un fabricante concreto.

Este archivo se ha revelado de gran utilidad, tanto durante el desarrollo de los diferentes módulos que componen la capa MAC, como durante el proceso de simulación y verificación de los mismos realizado posteriormente.

Capítulo 4

Verificación funcional

4.1 Verificación funcional MAC

El proceso de verificación funcional de los módulos desarrollados en este TFM para conformar la funcionalidad básica de la capa MAC ha consistido en realizar simulaciones módulo a módulo, comprobando su funcionamiento y finalmente, una vez conformado el módulo MAC con todos sus submódulos, realizar una simulación que mostrara su correcto comportamiento. De esta manera, en los siguientes subapartados se irá describiendo cada simulación y mostrando los resultados pertinentes. Para lograr una mejor comprensión de los resultados, el orden de los apartados se ha establecido de forma que se comienza por los módulos de menor complejidad y se termina por el módulo *mac*.

4.1.1. ARBITER Test

El módulo que describe el test realizado sobre el módulo *arbiter* se ha denominado *tb_ arbiter*. La simulación consiste principalmente en establecer un ancho de 32 posibles señales de solicitud de acceso a la capa PHY e ir modificando a nivel alto o nivel bajo determinados bits para comprobar que el árbitro cumple correctamente con la responsabilidad que se le ha encomendado. En las Figuras 34, 35 y 36 se puede ver su funcionamiento.

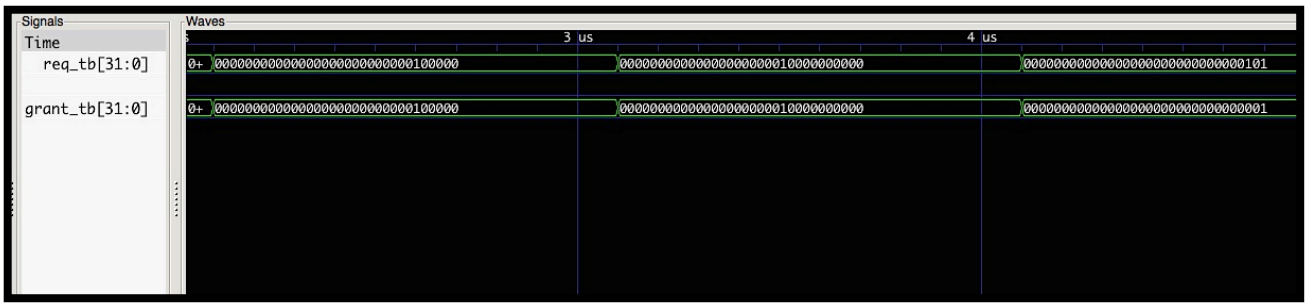


Figura 34 - ARBITER Test (I)

En la Figura 34 se puede observar cómo la señal "*grant*" va reflejando las solicitudes que llegan a través de la señal "*req*". En el último caso que se muestra se aprecia claramente que en caso de llegar dos solicitudes al mismo tiempo se asigna al bit de menor peso, mientras que las otras posibles solicitudes tendrán que esperar a que finalice la que ha sido concedida.

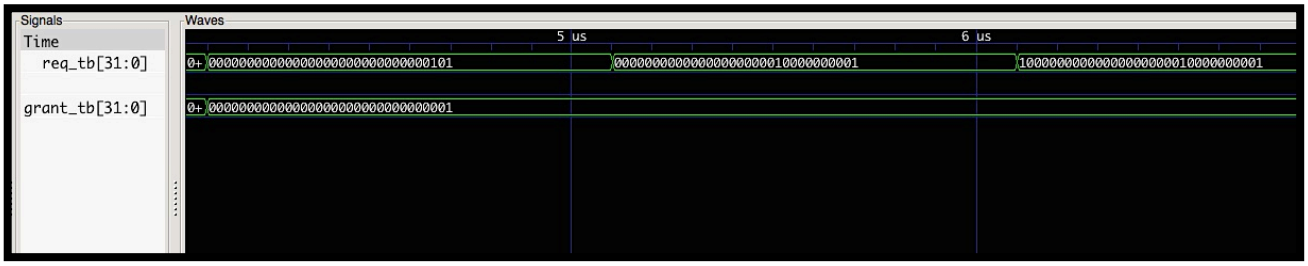


Figura 35 - ARBITER Test (II)

La Figura 35 continúa las formas de onda reflejadas en la Figura 34, profundizando en la idea de que, independientemente de las solicitudes que se reciban en el árbitro, mientras la que ha sido concedida no pase a nivel bajo, continuará de la misma forma.

Figura 36 - ARBITER Test (III)

Por último, la Figura 36 muestra el momento en el que una solicitud que ha sido concedida pasa a nivel bajo y se vuelve a evaluar toda la señal "*req*" para conceder acceso al bit de menor peso.

4.1.2. BD Test

El módulo que recoge la descripción del test realizado sobre el módulo *bd* se ha denominado *tb_ bd*. Para su realización se han instanciado 3 módulos del *bus driver* con el fin de comprobar que la escritura octeto a octeto, la generación de la señal de *request* de salida y la generación del reloj de salida eran correctas. Se han añadido tres figuras que muestran el correcto funcionamiento del módulo *bd* en las tres solicitudes que se realizan.

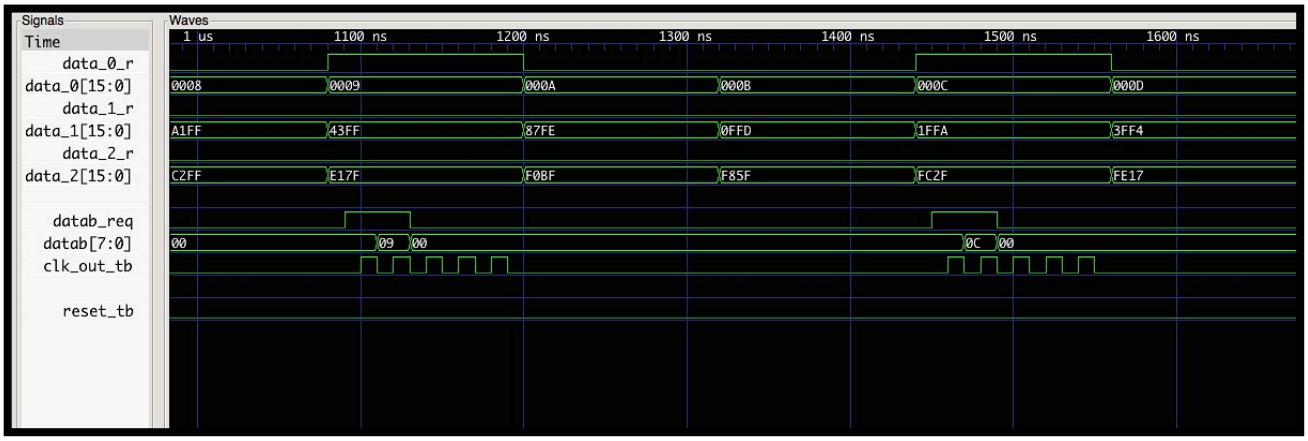


Figura 37 - BD Test (I)

En la Figura 37 se puede observar cómo la señal "*data_0_r*", que representa la solicitud correspondiente a la señal de datos "*data_0*", se pone a nivel alto y seguidamente, en la salida del *bus driver* se observa que la señal de solicitud de salida de datos se pone a nivel alto, transmitiéndose octeto a octeto los datos 0x0009 y 0x000C junto con el separador 0x00. Además, se observa la generación de la señal de reloj de salida "*clk_out_tb*", necesaria para sincronizar las dos capas.

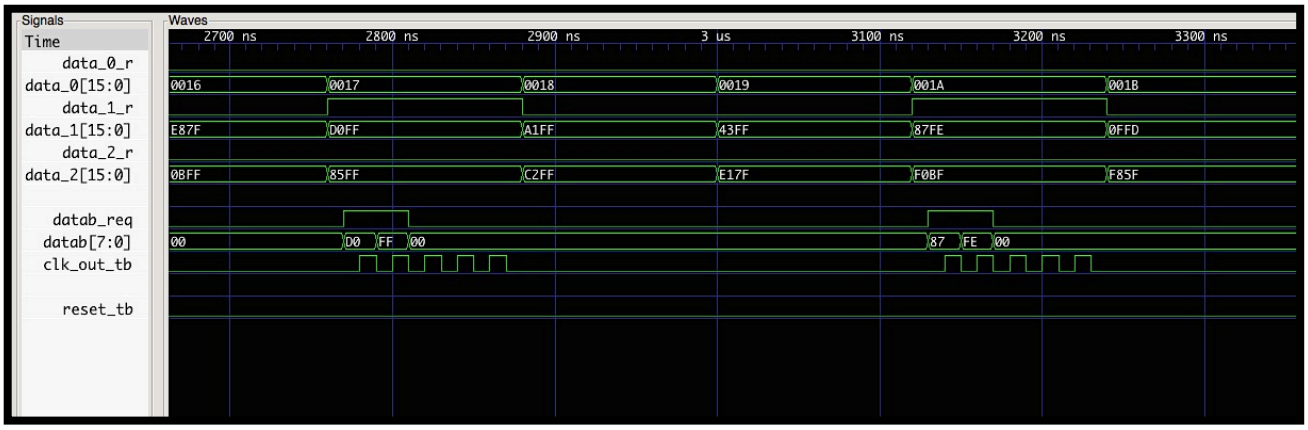


Figura 38 - BD Test (II)

En la Figura 38, la señal "*data_1*" es la que desea transmitir sus datos, por lo que su señal de solicitud "*data_1_r*" se establece a nivel alto y se transmiten los datos 0xD0FF y 0x87FE octeto a octeto, junto con el separador. Nuevamente la señal de solicitud de salida de datos y el reloj de salida se comportan de la forma esperada.

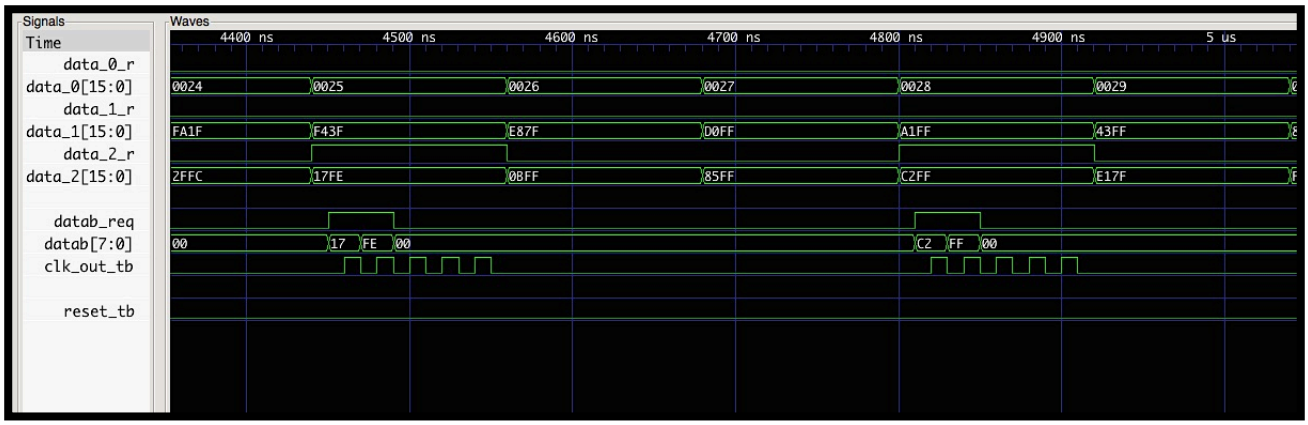


Figura 39 - BD Test (III)

En las formas de onda mostradas en la Figura 39, al igual que en las Figuras 37 y 38, se realiza una solicitud de datos (en este caso correspondiente a "*data_2*"), se transmiten los datos 0x17FE y 0xC2FF junto con el separador, y se activa la señal de solicitud de salida de datos y el reloj de salida. Con esto queda comprobado el funcionamiento básico del módulo *bd* descrito.

4.1.3. GET_PHY_REQ Test

El módulo *tb_get_phy_req* es el encargado de testear el funcionamiento del módulo *get_phy_req*, cuya responsabilidad es comunicar el módulo *mac_cpu* del sistema con la capa física y viceversa cuando se solicita una primitiva de servicio de tipo *MLME_GET_request*.

Para llevar a cabo este test se ha introducido un parámetro PIB por el puerto del módulo dedicado a ello, unido a una solicitud de *request*. De esta manera se comprueba la activación de la solicitud de acceso al árbitro y la conformación del mensaje que irá destinado a la capa PHY (previo paso por el bus, motivo por el que no se le ha añadido el separador, que se incluye en el módulo *bd*).

Posteriormente se introducirán los datos provenientes de la capa PHY octeto a octeto, junto con el reloj de sincronización de entrada y la señal "*valid*" que funciona como *request*. Finalmente, una vez estén listos los datos para ser transmitidos a módulo *mac_cpu*, y en función de la señal "*send*", se transmitirá el número de octetos que tiene el PIB en cuestión, se señalizará que los datos están disponibles, y se enviarán los datos todos juntos mediante la señal "*response*". Esto último es importante, de forma que los datos se transmiten en un único paquete de 56 bits al módulo *mac_cpu* y este se encarga de determinar (gracias a la señal que indica el número de octetos del PIB) cuántos debe transmitir.

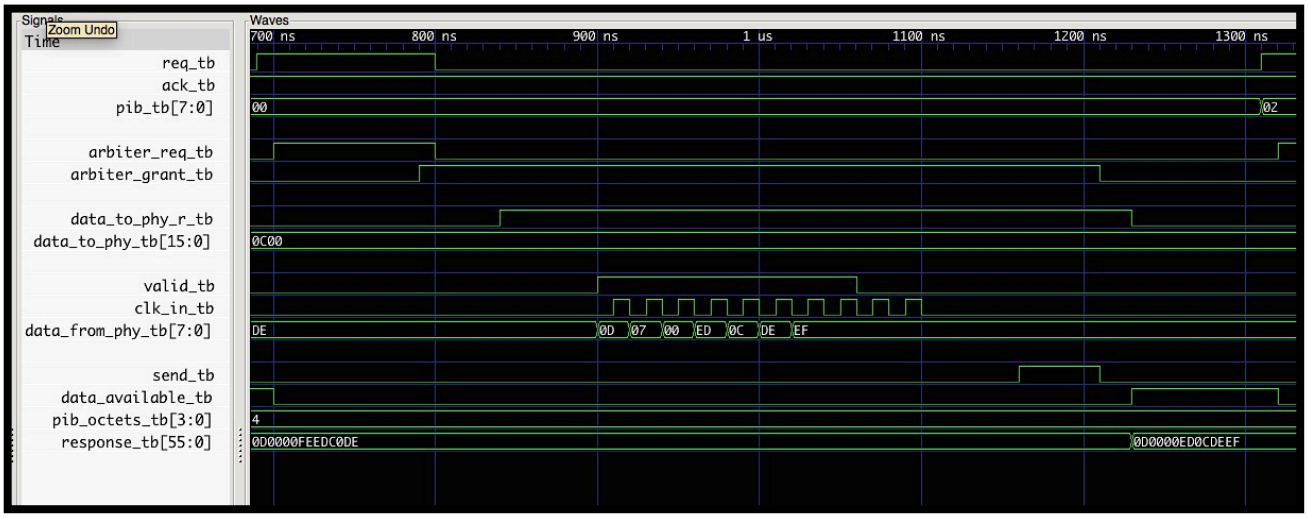


Figura 40 - GET_PHY_REQ Test (I)

En la Figura 40 se puede apreciar el funcionamiento del módulo *get_phy_req*. Cuando se activa la señal de solicitud de entrada se solicita permiso de acceso al árbitro y se conforma el mensaje que irá a la capa física, en este caso 0x0C00, esto es, *PLME_GET_request + phyCurrentChannel*. Una vez y el árbitro activa la señal "*grant*", se realiza la solicitud de escritura estableciendo a nivel alto la señal "*data_to_phy_r*" y el módulo quedará inactivo hasta la llegada de datos desde la PHY.

Cuando se activa la señal "*valid*" y el reloj de entrada "*clk_in*", se van registrando los datos que vienen de la capa inferior, en este caso 0x0D | 0x07 | 0x00 | 0xED | 0x0C | 0xDE | 0xEF, que se corresponden con la respuesta esperada, *PLME_GET_confirm + SUCCESS + phyCurrentChannel + 0xED0CDEEF*. Respecto a los últimos octetos, se ha planificado la transmisión de este mensaje para realizar el test con el peor caso (cuatro octetos).

Para finalizar, una vez y los datos están preparados, se activa la señal "*data_available*", el número de octetos está especificado a 4 y la señal "*response*" es conformada con el mensaje. Se puede observar cómo ahora el octeto 0x07, que se corresponde a la codificación del estado *SUCCESS* en la capa física, ha pasado a 0x00, que corresponde con la codificación del estado *SUCCESS* en la capa MAC, por lo que la traducción del estado se ha realizado también correctamente.

En la Figura 41 se muestra la realización de una prueba con un identificador de PIB erróneo (0x08), por lo que la capa física debería responder con *UNSUPPORTED_ATTRIBUTE*, codificado como 0x0A, y ser traducido como 0xF4 en la señal "*response*" hacia la capa MAC.

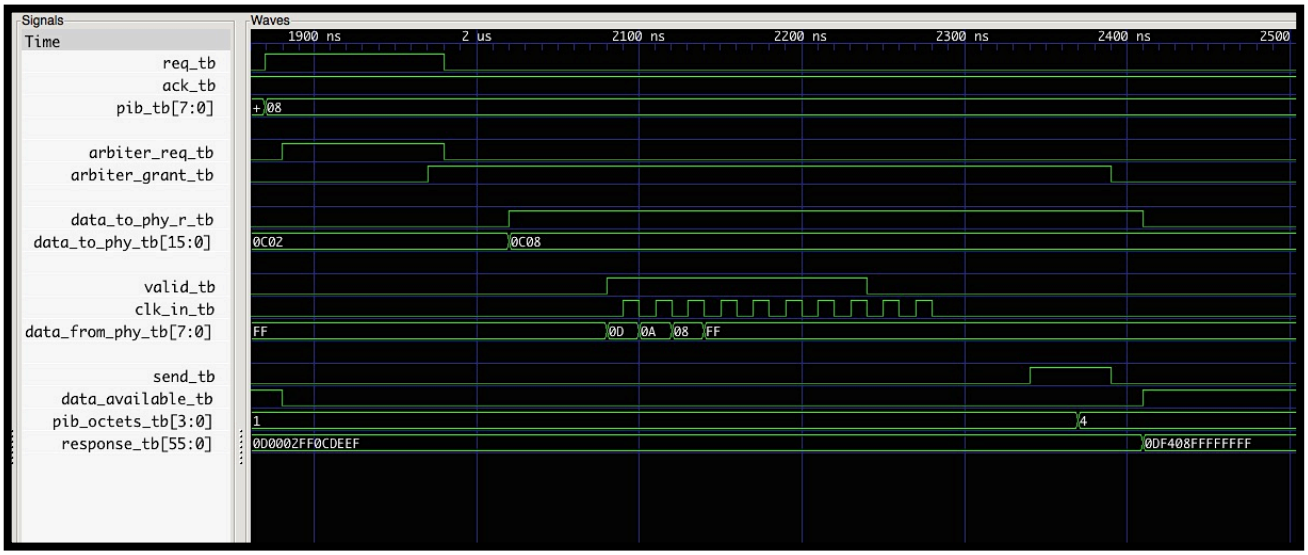


Figura 41 - GET_PHY_REQ Test (II)

En la Figura 42 se muestra el caso en el que la capa PHY responda con un solo octeto de datos.

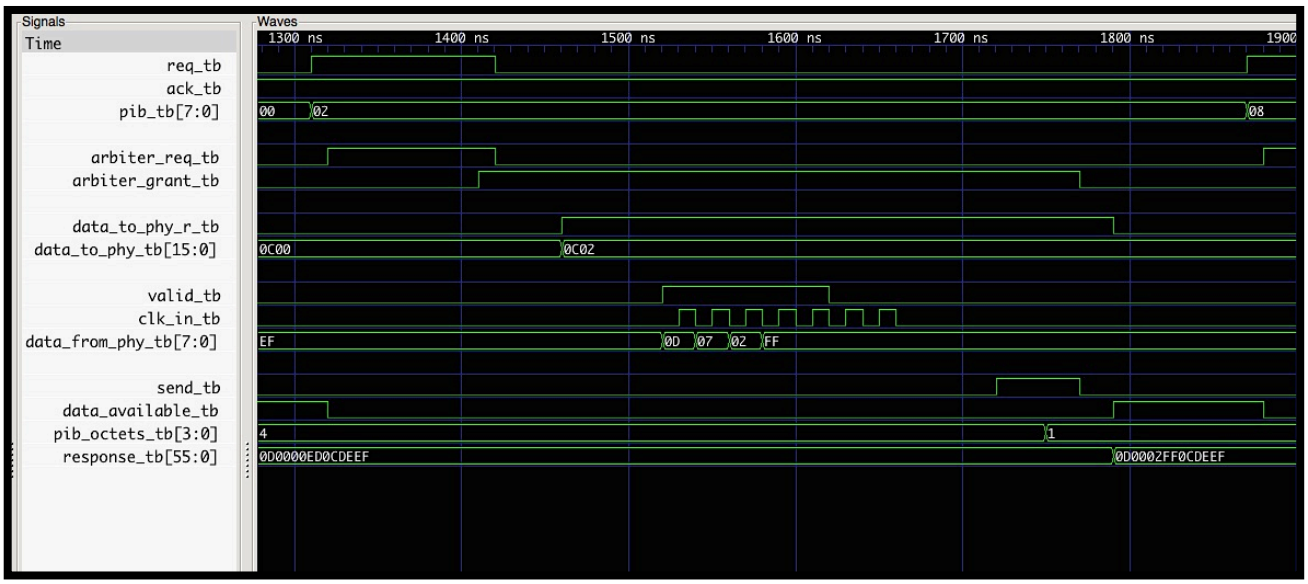


Figura 42 - GET_PHY_REQ Test (III)

En este caso el identificador de PIB que se transmite es 0x02, a lo que la capa PHY responde con el valor 0x07; correspondiente al estado *SUCCESS*, si bien sólo se transmite un octeto de datos, 0xFF. En la conformación de la señal "*response*" se escribe dicho valor 0xFF en la parte alta de esos últimos 32 bits, pero el resto de la señal mantiene los valores anteriores. Esto no es problema, ya que el módulo *mac_cpu* posteriormente discriminará los 24 bits restantes.

4.1.4. MAC_CPU Test

Para realizar las simulaciones sobre el módulo *mac_cpu* se han desarrollado dos tests. Inicialmente se realizó un test que cumplió su función durante las primeras etapas de desarrollo del módulo y que simplemente comprobaba el correcto funcionamiento en lo que respecta a las tareas exclusivas de la capa MAC (*tb_mac_cpu*). Con la inclusión del resto de módulos para completar la comunicación con la capa física, fue necesario describir un nuevo módulo de test para comprobar que las nuevas características añadidas eran plenamente funcionales. De esta manera, el módulo *tb_mac_layer* incluye instancias a los módulos *mac_cpu*, *get_phy_req* y *arbiter*.

En las siguientes figuras se muestra la ejecución de una primitiva *MLME_GET_request* con parámetro PIB de la capa MAC, una primitiva *MLME_GET_request* con parámetro PIB de la capa PHY, una primitiva *MLME_SET_request* con parámetro PIB de la capa MAC, y por último una primitiva *MLME_RESET_request*.

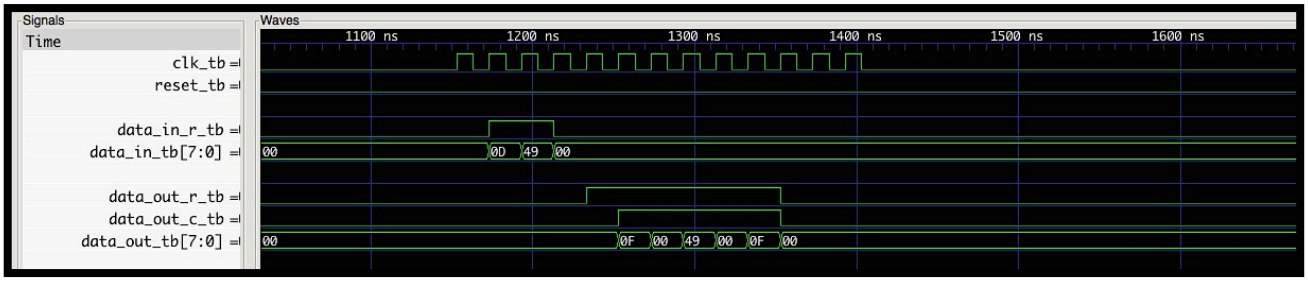


Figura 43 - CPU Test (I)

En la Figura 43 se observa que en el puerto de entrada de datos se dispone de los octetos 0x0D | 0x49 | 0x00, que representan la codificación de *MLME_GET_request* + *macBSN* + *Separador*. La respuesta por parte de la CPU es 0x0F | 0x00 | 0x49 | 0x00 | 0x0F | 0x00. Esta respuesta es correcta, puesto que se corresponde con un comando *MLME_GET_confirm* + *SUCCESS* + *macBSN* + *Valor del parámetro PIB solicitado* + *Separador*.

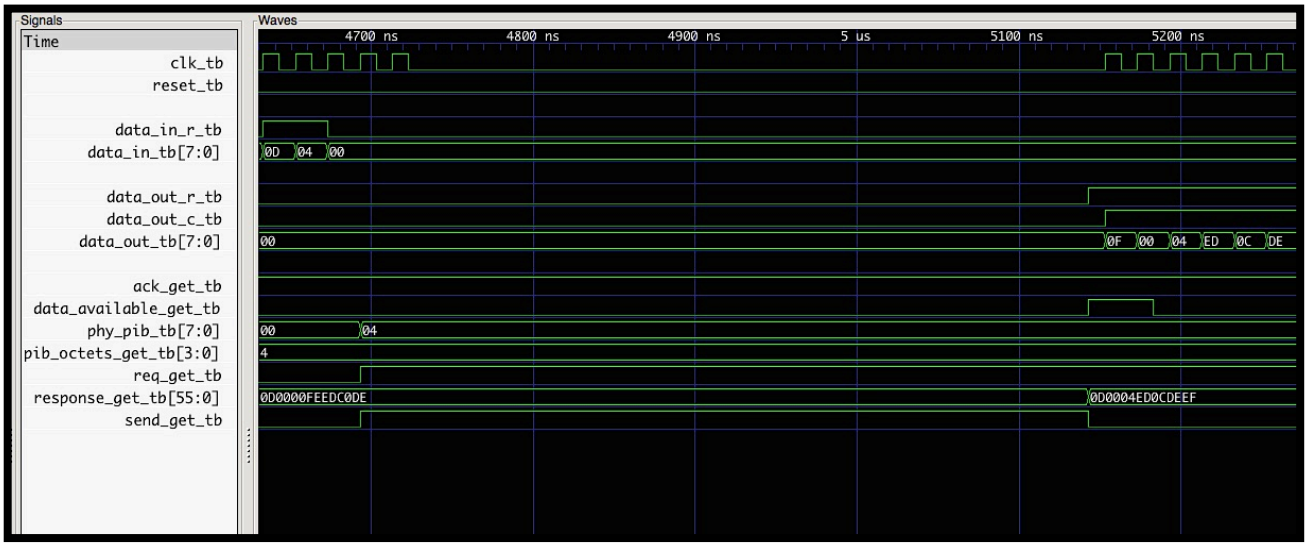


Figura 44 - CPU Test (II)

En la Figura 44 se puede apreciar que de nuevo se tiene una solicitud de tipo *MLME_GET_request*, pero en esta ocasión el parámetro PIB que acompaña a la solicitud es concerniente a la capa PHY, en concreto 0x04, que corresponde con *phyCurrentPage*.

De la misma forma que se ha observado el funcionamiento del resto de módulos en los apartados anteriores, aquí puede verse cómo se realiza la traducción del estado y se devuelve la respuesta al módulo *mac_cpu*, que conforma el mensaje en su puerto de salida tal y como se espera, transmitiendo 0x0F | 0x00 | 0x04 | 0xED0CDEEF, correspondiendo con lo esperado, *MLME_GET_confirm + SUCCESS + phyCurrentPage + Valor del PIB proveniente de PHY (simulado en este caso como 4 octetos para probar el peor caso) + Separador*.

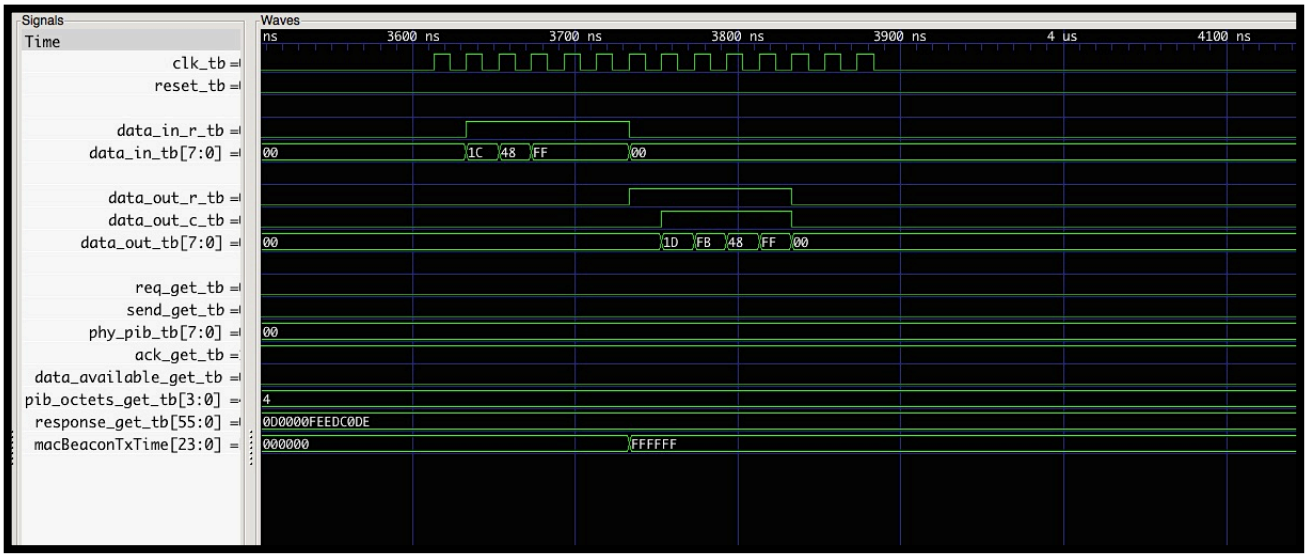


Figura 45 - CPU Test (III)

En la Figura 45 se observa la llegada por el puerto de entrada de una primitiva de tipo *MLME_SET_request*, para el parámetro *macBeaconTxTime*, con un valor de 0xFFFFFF y el separador. La respuesta *MLME_SET_confirm* + *READ_ONLY* + *macBeaconTxTime* + *Index* + *Separador* se corresponde con la codificación mostrada en la captura, 0x1D | 0xFB | 0x48 | 0xFF | 0x00. Es importante observar cómo en el atributo PIB se establece el valor que se le pasa a través de la primitiva de servicio.

Se ha realizado este test, aunque el estado que devuelve el parámetro PIB sea *READ_ONLY* para comprobar que la devolución de estados diferentes a *SUCCESS* funcionaba correctamente, así como la escritura en la base de datos de PIBs.

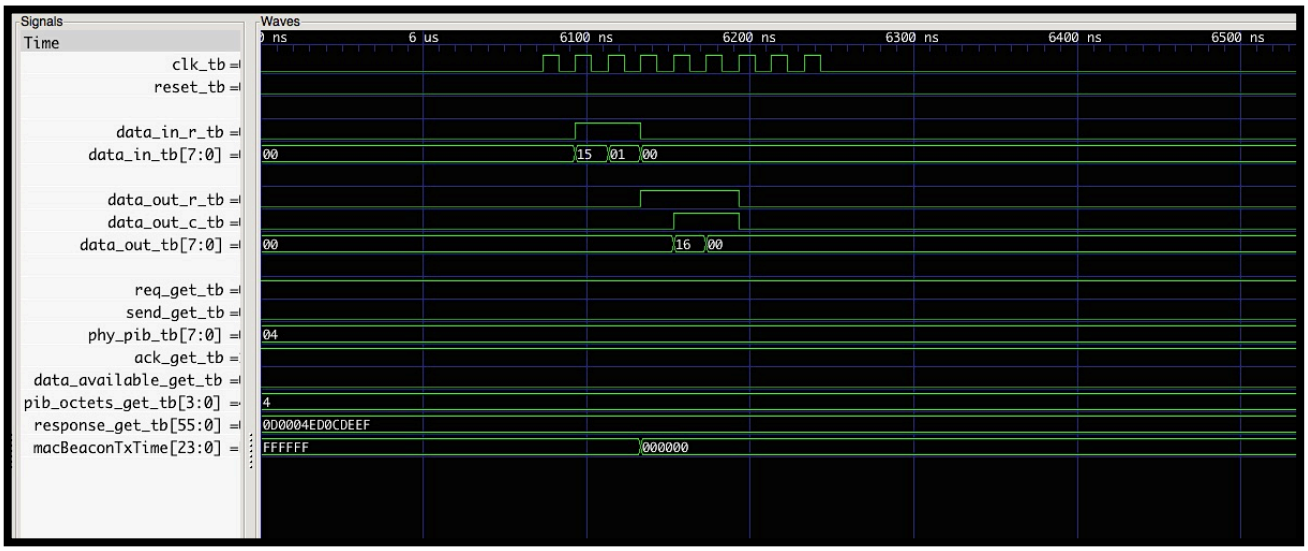


Figura 46 - CPU Test (IV)

Por último, en la Figura 46 se muestra la comprobación del funcionamiento del comando *MLME_RESET_request*. La codificación transmitida es 0x15 | 0x01 | 0x00, que se corresponde con la propia primitiva y el valor booleano que conlleva la inicialización de todos los PIB de la capa MAC a sus valores por defecto. La realización de esto último se puede observar en el parámetro situado en la última forma de onda (que venía establecido a 0xFFFFFF del comando *SET_request* anterior), que retorna a su valor por defecto, 0x000000. Además, en el puerto de salida se obtiene la codificación 0x16 | 0x00, que corresponde con el *MLME_RESET_confirm* + *Separador*.

4.1.5. MAC Test

En el último test realizado exclusivamente sobre el módulo *mac*, a los módulos ya mencionados en el test anterior se les ha añadido el módulo *bd*, para completar la descripción total de la capa MAC. En la Figura 47 se muestra por lo tanto una transmisión hacia la capa PHY y la respuesta para que la capa MAC transfiera los datos por su puerto de salida.

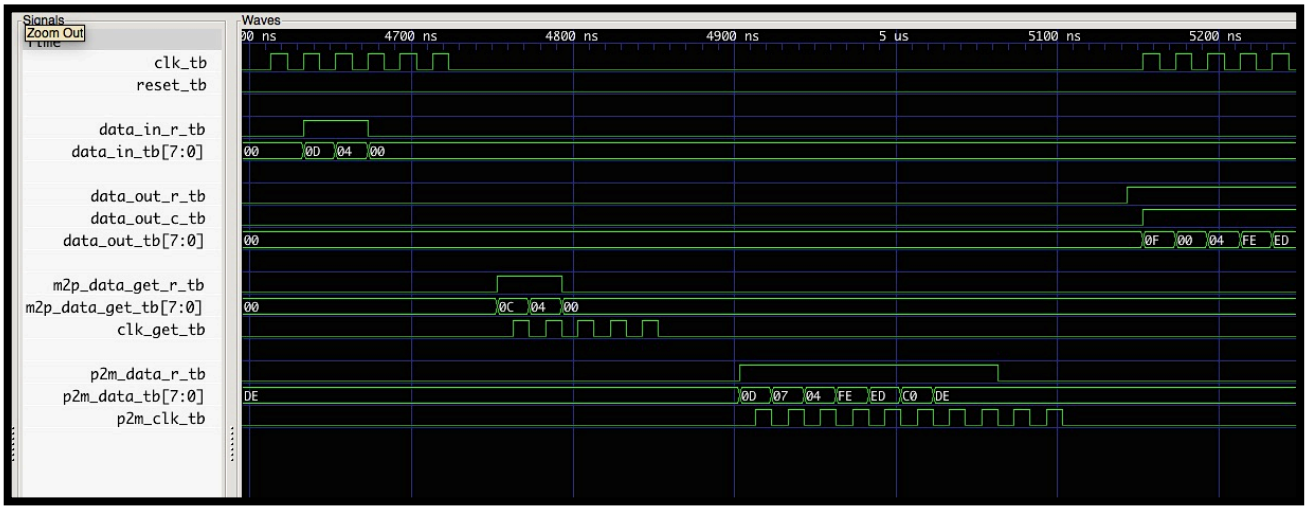


Figura 47 - MAC Test

Al igual que en el apartado anterior, el mensaje que llega al puerto de entrada de la capa MAC es codificado como 0x0D | 0x04 | 0x00, que se corresponde con un comando *MLME_GET_request + phyCurrentPage + Separador*. Ante esto, se envía a la capa PHY la solicitud, el mensaje *PLME_GET_request + phyCurrentPage + Separador*, codificado como 0x0C | 0x04 | 0x00, y el reloj de sincronización.

La capa física responde con un mensaje conformado como *PLME_GET_confirm + SUCCESS (PHY) + phyCurrentPage + 4 octetos (para probar peor caso)*, codificado como 0x0D | 0x07 | 0x04 | 0xFEEDC0DE. Este mensaje que devuelve la capa PHY es debidamente traducido y se envía por el puerto de salida de la capa MAC como *MLME_GET_confirm + SUCCESS (MAC) + phyCurrentPage + 4 octetos*, codificado como 0x0F | 0x00 | 0x04 | 0xFEEDC0DE.

Con este último test queda completado el proceso de verificación funcional de la capa MAC.

4.2 Verificación funcional MAC+PHY

Una vez verificado el funcionamiento básico, tanto de la capa PHY como de la capa MAC de forma independiente, se procedió a realizar una verificación funcional de la integración de ambas. En la Figura 48 se muestra la estructura que se desea simular.

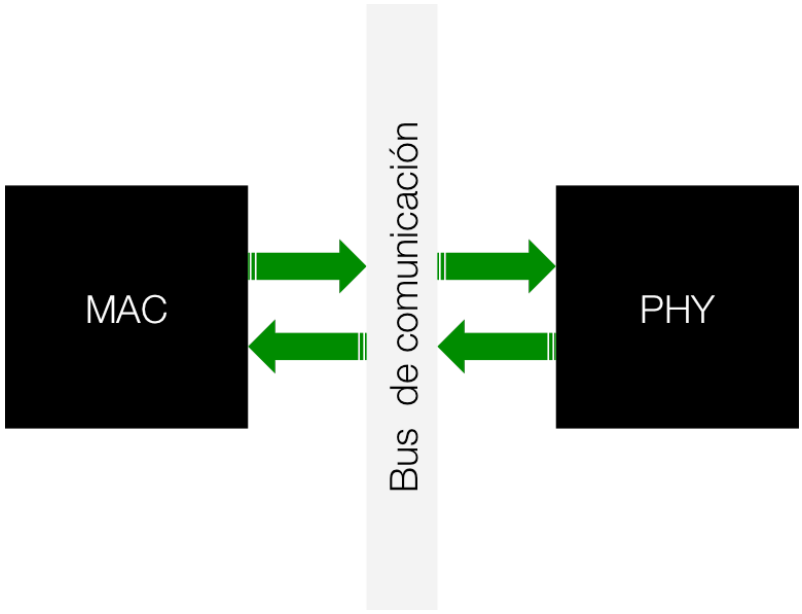


Figura 48 - Integración MAC + PHY (Estructura)

Para llevar a cabo esta verificación fue modificado el módulo de test *tb_mac*, añadiéndole las instancias a los módulos MAC y PHY e interconectándolos. El objetivo principal consistió en comprobar el correcto funcionamiento de la primitiva de servicio *MLME_GET_request* cuando se le pasaba cualquier atributo PIB correspondiente a la capa PHY puesto que esta primitiva es la que está implementada de forma completa.

En las siguientes figuras se recogen los resultados obtenidos a partir de esta verificación funcional, siendo el procedimiento común para todos los casos. Por el puerto de entrada de la capa MAC entra una primitiva de servicio *MLME_GET_request* (codificación 0x0D) y un parámetro PIB de la capa PHY (codificación 0x00 - 0x07). A continuación la capa MAC envía por su puerto de salida hacia la capa PHY la primitiva de servicio *PLME_GET_request* (codificación 0x0C) y el parámetro PIB introducido, unidos a la señal de solicitud de envío de datos y al reloj de sincronización. Seguidamente, la capa PHY responde a través de su puerto de salida hacia la capa MAC con la primitiva *PLME_GET_confirm* (codificación 0x0D), el estado *SUCCESS* (codificación 0x07), el parámetro PIB solicitado, el valor de dicho PIB (en un máximo de 4 octetos) y el separador. Por último, la capa MAC envía la información por su puerto de salida hacia las capas superiores. Este mensaje final consiste en el comando *MLME_GET_confirm*

(codificación 0x0F), el estado *SUCCESS* (codificación 0x00), el PIB solicitado, el valor de dicho PIB y el separador. La Figura 49 se corresponde con la solicitud del parámetro PIB *phyCurrentChannel*, mientras que en la Figura 50 se muestra el caso correspondiente a la solicitud del parámetro PIB *phyChannelsSupported*. Del mismo modo, en la Figuras 51, 52, 53, 54, 55 y 56 se representan las formas de onda correspondientes a la solicitud de los parámetros PIB *phyTransmitPower*, *phyCCAMode*, *phyCurrentPage*, *phyMaxFrameDuration*, *phySHRDuration* y *phySymbolsPerOctet*., respectivamente.

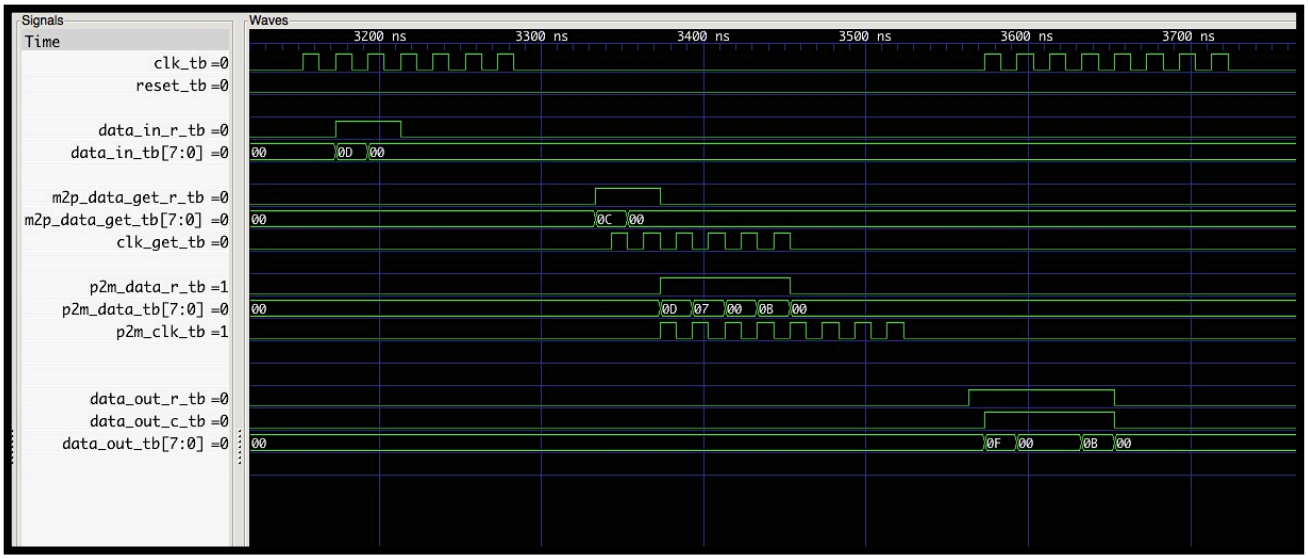


Figura 49 - Test MAC + PHY (phyCurrentChannel)

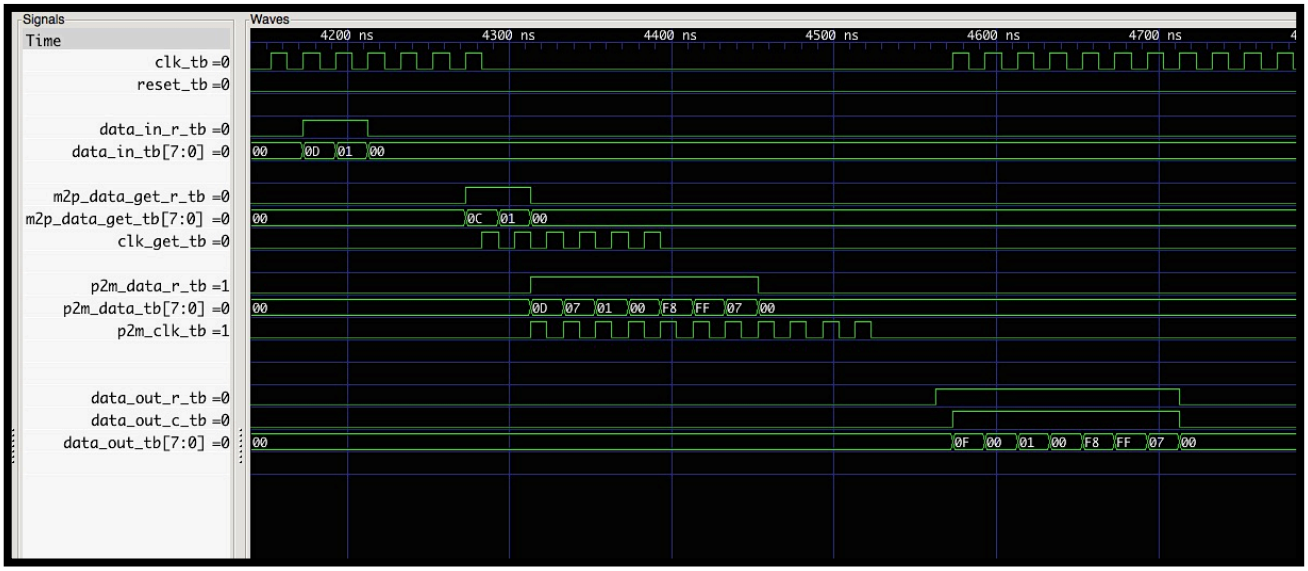


Figura 50 - Test MAC + PHY (phyChannelsSupported)

Figura 51 - Test MAC + PHY (phyTransmitPower)

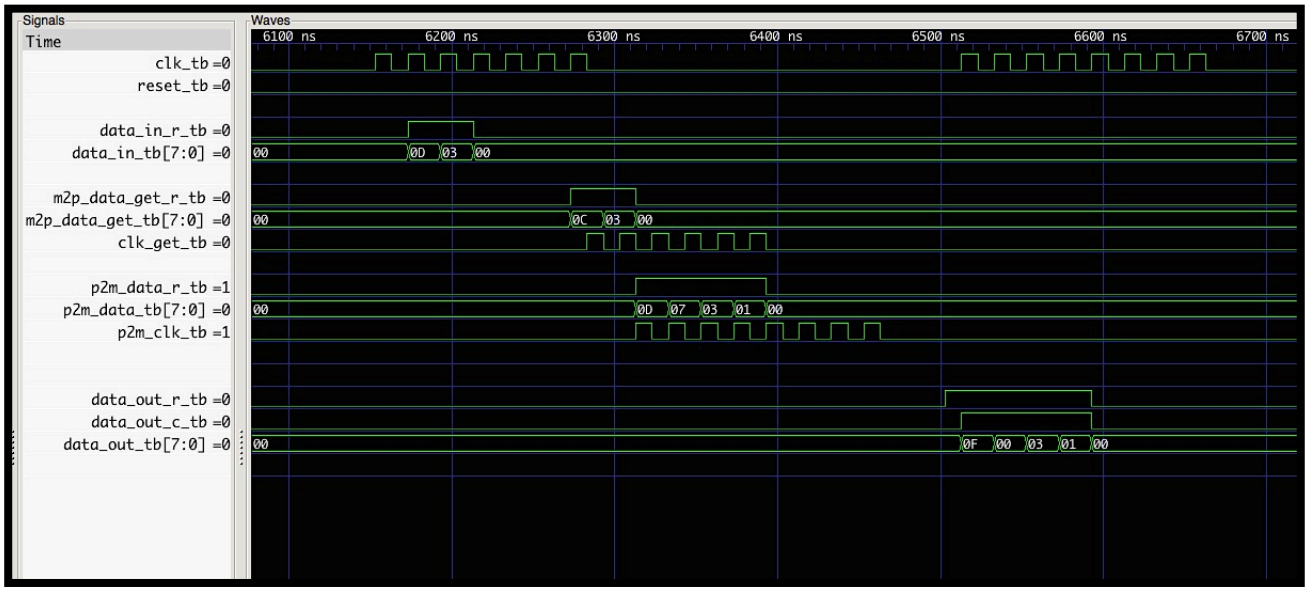


Figura 52 - Test MAC + PHY (phyCCAMode)

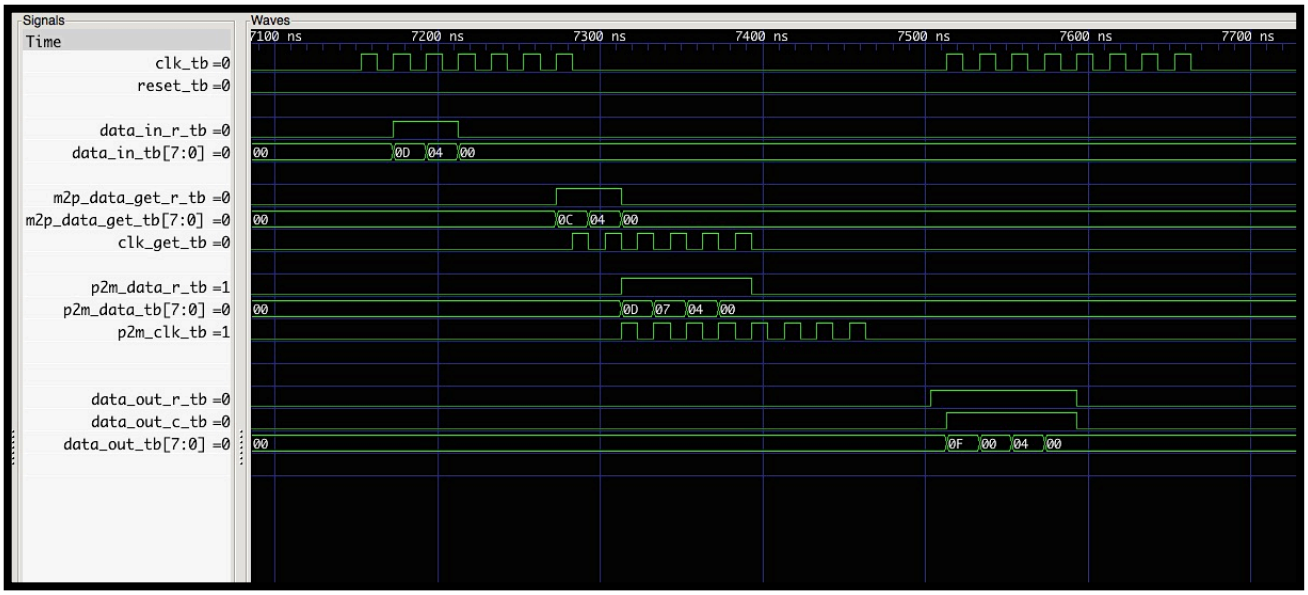


Figura 53 - Test MAC + PHY (phyCurrentPage)

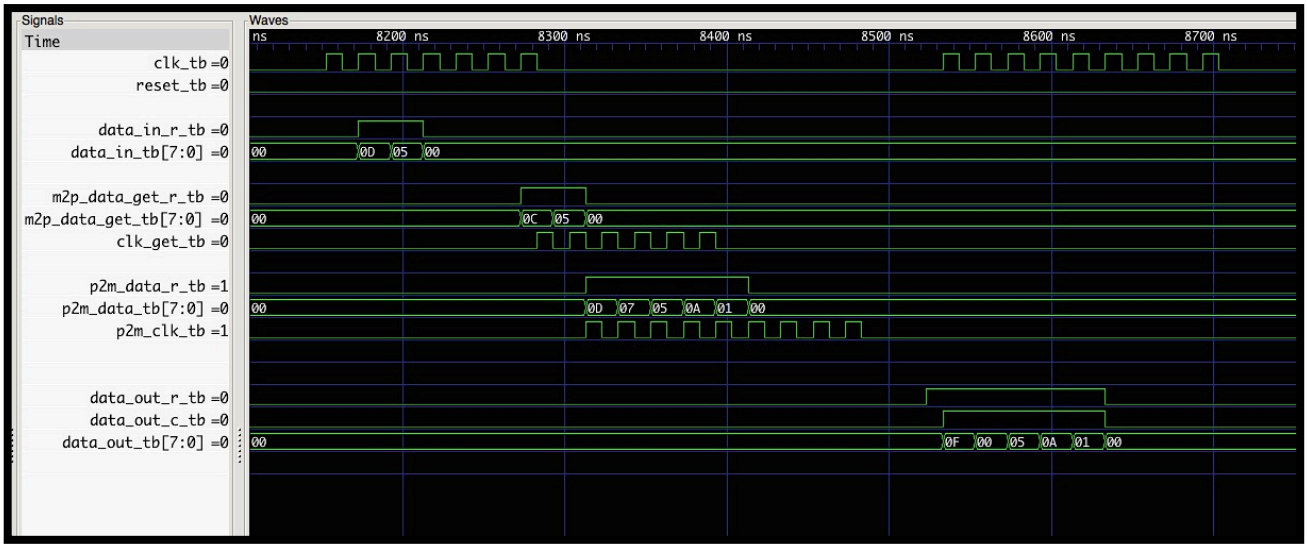


Figura 54 - Test MAC + PHY (phyMaxFrameDuration)

Figura 55 - Test MAC + PHY (phySHRDuration)

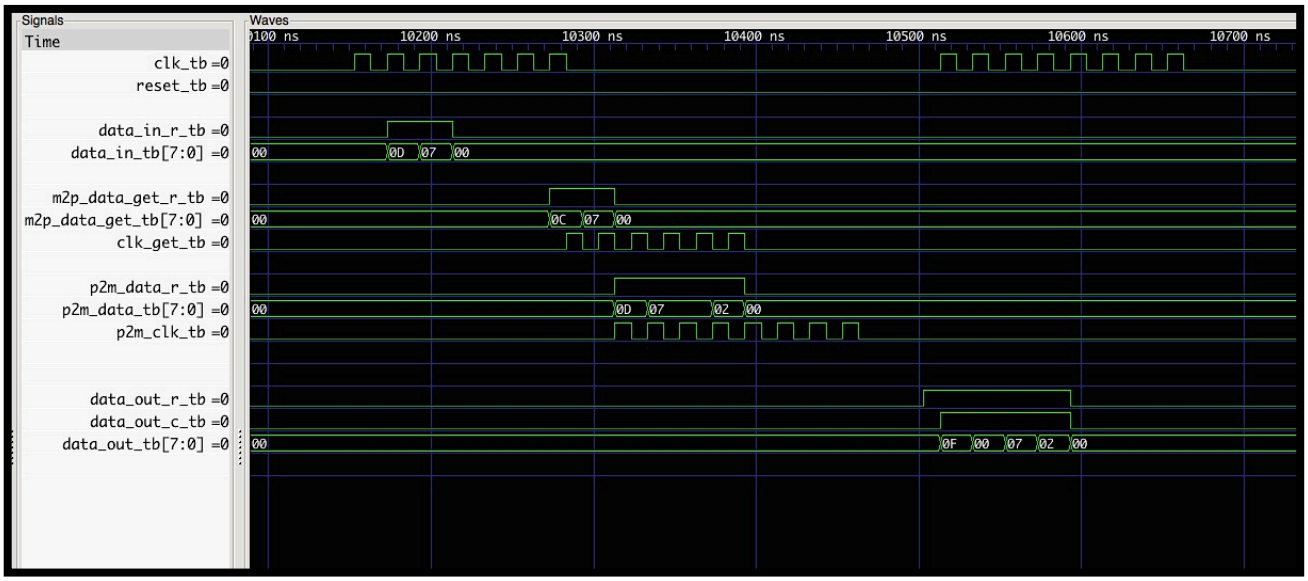


Figura 56 - Test MAC + PHY (phySymbolsPerOctet)

Capítulo 5

Conclusiones y líneas futuras

5.1 Conclusiones

En este Trabajo Fin de Máster se ha partido de una implementación completa de la capa PHY definida en el estándar IEEE 802.15.4 de WSN que incluye soporte para parámetros PIB, medida de ED, CCA, transmisión y recepción de mensajes, comunicación por puerto paralelo, y sistema de colas de mensajes. Esta implementación se ha estudiado en profundidad y su funcionalidad ha sido verificada.

Para realizar el desarrollo básico de la funcionalidad de la capa MAC se ha estudiado con detalle la especificación del estándar IEEE 802.15.4. Este desarrollo ha sido llevado a cabo planificando un diseño modular centralizado en una unidad que condensa la gestión y organización del resto de módulos. El diseño de la capa MAC ha sido coherente con la implementación de la capa PHY previamente desarrollada, describiendo su comportamiento en lenguaje Verilog HDL, utilizando el sistema de colas establecido, y respetando las codificaciones.

Además del módulo central se ha descrito un módulo intermedio que incluye dos autómatas que permiten gestionar la comunicación entre el módulo *mac_cpu* desarrollado y la capa PHY, un módulo con funciones de arbitraje para gestionar el acceso al recurso único que representa la capa PHY, y un módulo que realiza las funciones de *bus driver*, que es particular para cada módulo intermedio y que se comunica directamente con el bus de comunicaciones una vez el árbitro ha concedido el acceso.

Se ha verificado el funcionamiento de todos los módulos implementados, así como su integración en un módulo representativo de la totalidad de la capa MAC, cubriendo los objetivos establecidos para este Trabajo Fin de Máster.

5.2 Líneas futuras

Este TFM representa un elemento más en el desarrollo de un proyecto de mayor entidad con el objetivo de lograr la implementación en Verilog HDL de un simulador de WSN basado en el estándar IEEE 802.15.4. Como proyecto representa una gran oportunidad para conseguir un avance significativo en el campo de los simuladores de redes. A corto plazo se plantean las siguientes líneas futuras:

- Completar la capa MAC para que implemente todo el estándar IEEE 802.15.4.
- Desarrollar un entorno de simulación a alto nivel utilizando *Python* o *Java*. Este entorno facilitaría la generación de escenarios y configuración de simulaciones, haciendo la operación del simulador más amigable.

- Realizar una comparación con los resultados obtenidos en OMNeT++ y *ns2*. Si bien es cierto que estos simuladores incorporan modelos bastante incompletos del estándar IEEE 802.15.4, el hecho de comparar este simulador con la parte funcional en OMNeT++ y *ns2* podría ser considerada una validación de la implementación.

A medio y largo plazo se plantean las siguientes líneas futuras:

- Incorporación de modelos de propagación complejos que permitan modelar redes en interiores de edificios y construcciones.
- Incorporación de modelos de radio de *WiFi* y *Bluetooth*. Esto permitiría estudiar escenarios de coexistencia de redes, cosa que actualmente no permite ninguno de los simuladores mencionados, y que es uno de los principales desafíos actuales en los protocolos en la banda ISM (*Industrial, Scientific and Medical*).
- Modelado de dispositivos comerciales. En el proceso de simulación de redes inalámbricas, actualmente no existe posibilidad de simular completamente un protocolo, incluyendo el *firmware*/sistema operativo que es ejecutado en una CPU, junto con la radio. Sin embargo, sobre el papel es posible integrar una simulación en Verilog HDL, junto con un simulador de software utilizando el interfaz PLI (*Programming Language Interface*), lo cual permitiría simular versiones de software, sin necesidad de realizar implementaciones físicas ni despliegue de sensores.

BIBLIOGRAFÍA

[1] C.S. Raghavendra, K.M. Sivalingam, T. Znati, Wireless Sensor Networks, Estados Unidos de América: Kluwer Academic Publisher, 2004, ISBN 1-4020-7883-8.

[2] K. Wehrle, M. Günes, J. Gross, Modeling and Tools for Network Simulation, Alemania, Springer, 2010, ISBN 978-3-642-12330-6.

[3] OpenSim Ltd., (Junio, 2015), OMNeT++ Discrete Event Simulator [Online]. Available: http://www.omnetpp.org.

[4] ns-2, (Junio, 2015), User Information Nsnam [Online]. Available: http://nsnam.isi.edu/nsam/index.php/User_Information.

[5] Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (WPANs), IEEE 802.15.4, 2006.

[6] R. Esper-Chaín, (Junio, 2015), vWNetSim. Simulador de protocolos wireless sobre Verilog [Slides].

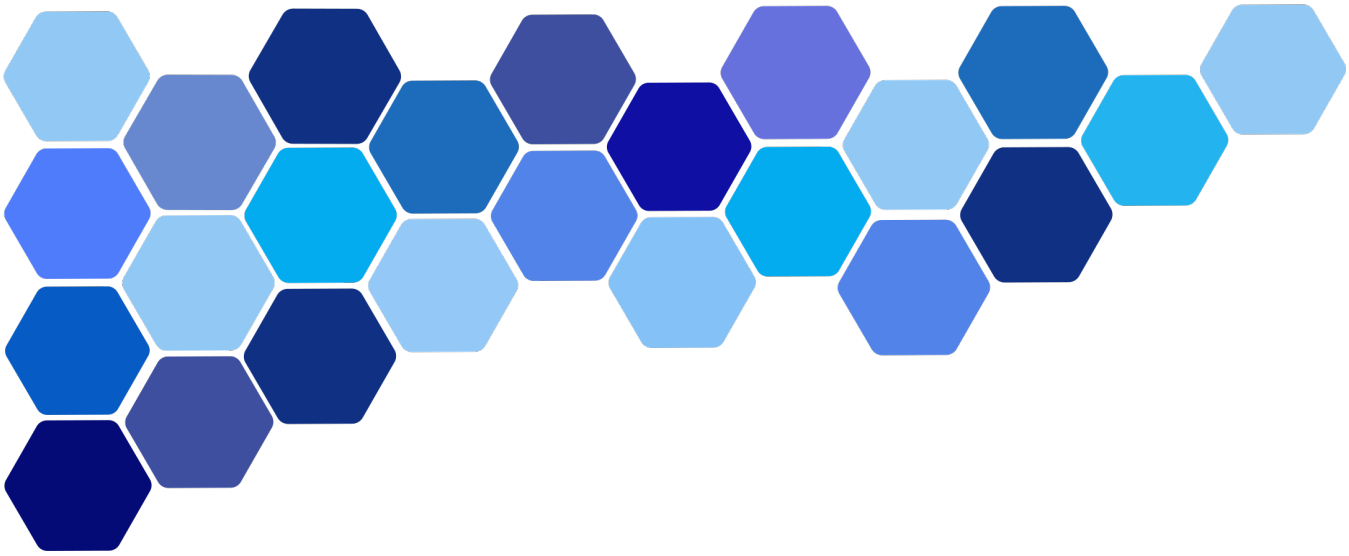
[7] Stephen Williams, (Junio, 2015), Icarus Verilog [Online]. Available: http://iverilog.icarus.com.

[8] GNU GPL., (Junio, 2015), GTKwave [Online]. Available: http://gtkwave.sourceforge.net.

[9] M.D. Ciletti, Modeling, Synthesis and Rapid Prototyping with the Verilog HDL, Estados Unidos de América: Prentice Hall, 1999, ISBN 0-13-977398-3.

[10] P.P. Chu, FPGA Prototyping by Verilog Examples, Estados Unidos de América: Wiley, 2008, ISBN 978-0-470-18532-2.

ANEXOS



Anexo I

En este anexo se recogen las primitivas, directivas de estado (*enumerations*), constantes y atributos PIB de la capa PHY.

PLME-SAP Primitives		
PLME-CCA	Request	Confirm
PLME-ED	Request	Confirm
PLME-GET	Request	Confirm
PLME-SET-TRX-STATE	Request	Confirm
PLME-SET	Request	Confirm

Tabla 1 - Primitivas de la capa PHY

PHY Enumerations	
Enumeration	**Value**
BUSY	0x00
BUSY_RX	0x01
BUSY_TX	0x02
FORCE_TRX_OFF	0x03
IDLE	0x04
INVALID_PARAMETER	0x05
RX_ON	0x06
SUCCESS	0x07
TRX_OFF	0x08

TX_ON	0x09
UNSUPPORTED_ATTRIBUTE	0x0A
READ_ONLY	0x0B

Tabla 2 - Estados de la capa PHY

PHY Constants	
Constant	**Value**
aMaxPHYPacketSize	127
aTurnaroundTime	12

Tabla 3 - Constantes de la capa PHY

PHY PIB Attributes			
Attribute	**Identifier**	**Type**	**Range**
phyCurrentChannel	0x00	Integer	0 - 26
phyChannelsSupported	0x01	Array	R x 32 bit array
phyTransmitPower	0x02	Bitmap	0x00 - 0xBF
phyCCAMode	0x03	Integer	1 - 3
phyCurrentPage	0x04	Integer	0 - 31
phyMaxFrameDuration	0x05	Integer	55, 212, 266, 1064
phySHRDuration	0x06	Integer	3, 7, 10, 40
phySymbolsPerOctet	0x07	Float	0.4, 1.6, 2, 8

Tabla 4 - Atributos PIB de la capa PHY

Anexo II

En este anexo se recogen las primitivas, directivas de estado (*enumerations*), constantes y atributos PIB de la capa MAC.

MCPS-SAP Primitives			
MCPS-DATA	Request	Confirm	Indication
MCPS-PURGE	Request	Confirm	-

Tabla 1 - Primitivas de la capa MAC (MCPS)

MLME-SAP Primitives				
MLME-ASSOCIATE	Request	Indication	Response	Confirm
MLME-DISASSOCIATE	Request	Indication	-	Confirm
MLME-BEACON-NOTIFY	-	Indication	-	-
MLME-GET	Request	-	-	Confirm
MLME-GTS	Request	Indication	-	Confirm
MLME-ORPHAN	-	Indication	Response	-
MLME-RESET	Request	-	-	Confirm
MLME-RX-ENABLE	Request	-	-	Confirm
MLME-SCAN	Request	-	-	Confirm
MLME-COMM-STATUS	-	Indication	-	-

MLME-SET	Request	-	-	Confirm
MLME-START	Request	-	-	Confirm
MLME-SYNC	Request	-	-	-
MLME-SYNC-LOSS	-	Indication	-	-
MLME-POLL	Request	-	-	Confirm

Tabla 2 - Primitivas de la capa MAC (MLME)

MAC Enumerations	
Enumeration	**Value**
SUCCESS	0x00
-	0x01 - 0xDA
-	0x80 - 0xDA 0xFE - 0xFF
BEACON_LOSS	0xE0
CHANNEL_ACCESS_FAILURE	0xE1
COUNTER_ERROR	0xDB
DENIED	0xE2
DISABLE_TRX_FAILURE	0xE3
FRAME_TOO_LONG	0xE5
IMPROPER_KEY_TYPE	0xDC
IMPROPER_SECURITY_LEVEL	0xDD
INVALID_ADDRESS	0xF5

INVALID_GTS	0xE6
INVALID_HANDLE	0xE7
INVALID_INDEX	0xF9
INVALID_PARAMETER	0xE8
LIMIT_REACHED	0xFA
NO_ACK	0xE9
NO_BEACON	0xEA
NO_DATA	0xEB
NO_SHORT_ADDRESS	0xEC
ON_TIME_TOO_LONG	0xF6
OUT_OF_CAP	0xED
PAN_ID_CONFLICT	0xEE
PAST_TIME	0xF7
READ_ONLY	0xFB
REALIGNMENT	0xEF
SCAN_IN_PROGRESS	0xFC
SECURITY_ERROR	0xE4
SUPERFRAME_OVERLAP	0xFD
TRACKING_OFF	0xF8
TRANSACTION_EXPIRED	0xF0
TRANSACTION_OVERFLOW	0xF1
TX_ACTIVE	0xF2

UNAVAILABLE_KEY	0xF3
UNSUPPORTED_ATTRIBUTE	0xF4
UNSUPPORTED_LEGACY	0xDE
UNSUPPORTED_SECURITY	0xDF

Tabla 3 - Estados de la capa MAC

MAC Constants	
Constant	**Value**
aBaseSlotDuration	60
aBaseSuperframeDuration	aBaseSlotDuration*aNumSuperframSlots
aExtendedAddress	Device specific
aGTSDescPersistenceTime	4
aMaxBeaconOverhead	75
aMaxBeaconPayloadLength	aMaxPHYPacketSize – aMaxBeaconOverhead
aMaxLostBeacons	4
aMaxMACSafePayloadSize	aMaxPHYPacketSize – aMaxMPDUUnsecuredOverhead
aMaxMACPayloadSize	aMaxPHYPacketSize – aMinMPDUOverhead
aMaxMPDUUnsecuredOverhead	25
aMaxSIFSFrameSize	18
aMinCAPLength	440
aMinMPDUOverhead	9
aNumSuperframeSlots	16

aUnitBackoffPeriod	20

Tabla 4 - Constantes de la capa MAC

MAC PIB Attributes			
Attribute	**Identifier**	**Type**	**Range**
macAckWaitDuration	0x40	Integer	Based on equation
macAssociatedPANCoord	0x56	Boolean	TRUE or FALSE
macAssociationPermit	0x41	Boolean	TRUE or FALSE
macAutoRequest	0x42	Boolean	TRUE or FALSE
macBattLifeExt	0x43	Boolean	TRUE or FALSE
macBattLifeExtPeriods	0x44	Integer	6 - 41
macBeaconPayload	0x45	Set of octets	-
macBeaconPayloadLength	0x46	Integer	0 - aMaxBeaconPayloadLength
macBeaconOrder	0x47	Integer	0 - 15
macBeaconTxTime	0x48	Integer	0x000000 - 0xFFFFFF
macBSN	0x49	Integer	0x00 - 0xFF
macCoordExtendedAddress	0x4A	IEEE address	64-bit IEEE address
macCoordShortAddress	0x4B	Integer	0x0000 - 0xFFFF
macDSN	0x4C	Integer	0x00 - 0xFF
macGTSPermit	0x4D	Boolean	TRUE or FALSE
macMaxBE	0x57	Integer	3 - 8
macMaxCSMABackoffs	0x4E	Integer	0 - 5

macMaxFrameTotalWaitTime	0x58	Integer	Based on equation
macMaxFrameRetries	0x59	Integer	0 - 7
macMinBE	0x4F	Integer	0 - macMaxBE
macMinLIFSPeriod	0x5E	Integer	Based on table
macMinSIFSPeriod	0x5F	Integer	Based on table
macPANId	0x50	Integer	0x0000 - 0xFFFF
macPromiscuousMode	0x51	Boolean	TRUE or FALSE
macResponseWaitTime	0x5A	Integer	2 - 64
macRxOnWhenIdle	0x52	Boolean	TRUE or FALSE
macSecurityEnabled	0x5D	Boolean	TRUE or FALSE
macShortAddress	0x53	Integer	0x0000 - 0xFFFF
macSuperframeOrder	0x54	Integer	0 - 15
macSyncSymbolOffset	0x5B	Integer	0x000 - 0x100
macTimestampSupported	0x5C	Boolean	TRUE or FALSE
macTransactionPersistenceTime	0x55	Integer	0x0000 - 0xFFFF

Tabla 5 - Atributos PIB de la capa MAC

Anexo III

En este anexo se recogen las descripciones de los interfaces de entrada/salida de los módulos correspondientes a la capa PHY.

Módulo PHY

Puerto	Tamaño	E/S	Descripción
clk	1	E	Señal de reloj del módulo.
reset	1	E	Señal de *reset* del módulo, activa a nivel alto.
rx_data	1	E	Señal de entrada de datos.
rx_power	64	E	Señal de entrada de potencia.
rx_noise	64	E	Señal de entrada de potencia de ruido.
tx_data	1	S	Señal de salida de datos.
tx_power	64	S	Señal de salida de potencia.
trx_freq	64	S	Señal de salida del parámetro frecuencia.
trx_bandw	64	S	Señal de salida del parámetro ancho de banda.
trx_mod	4	S	Señal de salida del parámetro modo de transmisión.
m2p_data	8	E	Señal de entrada de datos proveniente de la capa MAC.
m2p_data_r	1	E	Señal de entrada de solicitud proveniente de la capa MAC.
m2p_clk	1	E	Señal de entrada de reloj de sincronización proveniente de la capa MAC.
p2m_data	8	S	Señal de salida de datos hacia la capa MAC.
p2m_data_r	1	S	Señal de salida de solicitud hacia la capa MAC.
p2m_clk	1	S	Señal de salida de reloj de sincronización hacia la capa MAC.

Tabla 10 - Módulo PHY (Input/Output)

Módulo CCA

Puerto	Tamaño	E/S	Descripción
reset	1	E	Señal de *reset* del módulo, activa a nivel alto.
rx_data	1	E	Señal de entrada de datos.
rx_power	64	E	Señal de entrada de potencia.
rx_noise	64	E	Señal de entrada de potencia de ruido.
cca_req	1	E	Señal de solicitud de activación del módulo.
cca_logical	1	E	Señal de operación lógica en función del modo ($1 = AND$, $0 = OR$).
phyCCAmode	2	E	Señal de entrada del modo.
cca_resp	1	S	Señal de salida con la respuesta.
cca_busy	1	S	Señal de salida indicando que el módulo está ocupado.

Tabla 11 - Módulo CCA (Input/Output)

Módulo ED

Puerto	Tamaño	E/S	Descripción
reset	1	E	Señal de *reset* del módulo, activa a nivel alto.
rx_power	64	E	Señal de entrada de potencia.
rx_noise	64	E	Señal de entrada de potencia de ruido.
ed_req	1	E	Señal de solicitud de activación del módulo.
ed_value	8	S	Señal de salida con el valor.
ed_resp	1	S	Señal de salida de respuesta.

Tabla 12 - Módulo ED (Input/Output)

Módulo TX

Puerto	Tamaño	E/S	Descripción
phy_enable	1	E	Señal de activación del módulo.
phy_channel	5	E	Señal de entrada del canal de transmisión.
phy_power	8	E	Señal de entrada de potencia de transmisión.
phy_PHR	8	E	Señal de entrada de la cabecera PHR.
phy_PSDU	1024	E	Señal de entrada del *payload*.
tx_data	1	S	Señal de salida de datos.
tx_power	64	S	Señal de salida del parámetro potencia.
tx_frequency	64	S	Señal de salida del parámetro frecuencia.
tx_bandwidth	64	S	Señal de salida del parámetro ancho de banda.
pll_error	1	S	Señal de error en la configuración del PLL.
tx_error	1	S	Señal de error en la transmisión.
tx_eot	1	S	Señal de éxito en la transmisión.
tx_busy	1	S	Señal de transmisión en progreso.
tx_ready	1	S	Señal indicativa de que el transmisor está listo.

Tabla 13 - Módulo TX (Input/Output)

Módulo RX

Puerto	Tamaño	E/S	Descripción
rx_reset	1	E	Señal de activación del módulo.
rx_data	1	E	Señal de entrada de datos.

rx_power	64	E	Señal de entrada de potencia de transmisión.
rx_noise	64	E	Señal de entrada de potencia de ruido.
phy_ready	1	S	Señal de paquete recibido.
phy_signal	1	S	Señal de detección de señal.
phy_PHR	8	S	Señal de salida de la cabecera PHR.
phy_PSDU	1024	S	Señal de salida del *payload*.
phy_power	8	S	Señal de salida de potencia de transmisión.
phy_error	1	S	Señal de error en la transmisión.
phy_lqi	8	S	Señal de salida del parámetro LQI.

Tabla 14 - Módulo RX (Input/Output)

Módulo CPU

Puerto	Tamaño	E/S	Descripción
data_in	8	E	Señal de entrada de datos.
data_in_r	1	E	Señal de solicitud de entrada de datos.
rx_reset	1	S	Señal de salida para activación de módulos.
reset	1	E	Señal de *reset*, activa a nivel alto.
clk	1	E	Señal de reloj del módulo.
cca_resp	1	E	Señal de entrada de CCA.
cca_busy	1	E	Señal de indicación de CCA ocupado.
ed_resp	1	E	Señal de entrada de ED.
ed_value	8	E	Señal de entrada del valor de ED.

tx_ready	1	E	Señal de indicación de que el transmisor está preparado.
tx_busy	1	E	Señal de indicación de que el transmisor está ocupado.
tx_eot	1	E	Señal de indicación de éxito en la transmisión.
tx_error	1	E	Señal de indicación de error en la transmisión.
pll_error	1	E	Señal de indicación de error en el PLL.
data_out	8	S	Señal de salida de datos.
data_out_r	1	S	Señal de solicitud de salida de datos.
clk_out	1	S	Señal de salida de reloj de sincronización.
tx_enable	1	S	Señal salida para activación del transmisor.
tx_channel	5	S	Señal de salida del parámetro canal.
tx_power	8	S	Señal de salida de potencia de transmisión.
tx_PHR	8	S	Señal de salida de la cabecera PHR.
tx_PSDU	1024	S	Señal de salida del *payload*.
cca_req	1	S	Señal de solicitud de CCA.
cca_mode	2	S	Señal de indicación de modo CCA.
ed_req	1	S	Señal de solicitud de ED.
rx_ready	1	E	Señal de indicación de que el receptor está preparado.
rx_signal	1	E	Señal de detección del módulo RX.
rx_PHR	8	E	Señal de entrada de la cabecera PHR.
rx_PSDU	1024	E	Señal de entrada del *payload*.
rx_power	8	E	Señal de entrada de la potencia transmitida.

rx_lqi	8	E	Señal de entrada del parámetro LQI.
rx_error	1	E	Señal de error en la recepción.

Tabla 15 - Módulo CPU (Input/Output)

Módulo rx_Processor

Puerto	Tamaño	E/S	Descripción
rx_data	1	S	Señal de salida de datos.
rx_power	64	S	Señal de salida de potencia.
rx_noise	64	S	Señal de salida de potencia de ruido.
tx_data	1	E	Señal de entrada de datos.
tx_power	64	E	Señal de entrada de potencia.
trx_freq	64	E	Señal de entrada del parámetro frecuencia.
trx_bandw	64	E	Señal de entrada del parámetro ancho de banda.
trx_mod	4	E	Señal de entrada del parámetro modo de transmisión.
rx_mod	4	E	Señal de entrada del parámetro modo de transmisión que espera el RX.
rx_freq	64	E	Señal de entrada del parámetro frecuencia de transmisión que espera el RX.
rx_bandw	64	E	Señal de entrada del parámetro ancho de banda de transmisión que espera el RX.

Tabla 16 - Módulo rx_Processor (Input/Output)

Módulo air

Puerto	Tamaño	E/S	Descripción
rx_data	1	S	Señal de salida de datos.
rx_power	64	S	Señal de salida de potencia.

rx_mod	4	S	Señal de salida del parámetro modo de funcionamiento.
rx_freq	64	S	Señal de salida del parámetro frecuencia.
rx_bandw	64	S	Señal de salida del parámetro ancho de banda.
tx_data	1	E	Señal de entrada de datos.
tx_power	64	E	Señal de entrada de potencia de transmisión.
tx_mod	4	E	Señal de entrada del parámetro modo de funcionamiento.
tx_freq	64	E	Señal de entrada del parámetro frecuencia.
tx_bandw	64	E	Señal de entrada del parámetro ancho de banda.
rx_x	64	E	Señal de entrada de localización espacial del receptor (eje x).
rx_y	64	E	Señal de entrada de localización espacial del receptor (eje y).
rx_z	64	E	Señal de entrada de localización espacial del receptor (eje z).
tx_x	64	E	Señal de entrada de localización espacial del transmisor (eje x).
tx_y	64	E	Señal de entrada de localización espacial del transmisor (eje y).
tx_z	64	E	Señal de entrada de localización espacial del transmisor (eje z).

Tabla 17 - Módulo air (Input/Output)

Anexo IV

En este anexo se recogen las descripciones de los interfaces de entrada/salida de los módulos correspondientes a la capa MAC.

Módulo MAC

Puerto	Tamaño	E/S	Descripción
clk	1	E	Señal de reloj del módulo.
reset	1	E	Señal de *reset* del módulo, activa a nivel alto.
data_in	8	E	Señal de entrada de datos.
data_in_r	1	E	Señal de solicitud de entrada.
data_out	8	S	Señal de salida de datos
data_out_r	1	S	Señal de solicitud de salida de datos.
data_out_c	1	S	Señal de solicitud de salida de comandos.
m2p_data	8	S	Señal de salida de datos hacia la capa PHY.
m2p_data_r	1	S	Señal de solicitud de salida hacia la capa PHY.
m2p_clk	1	S	Señal de reloj de sincronización hacia la capa PHY.
p2m_clk	1	E	Señal de reloj de sincronización desde la capa PHY.
p2m_data_r	1	E	Señal de solicitud de entrada desde la capa PHY.
p2m_data	8	E	Señal de entrada de datos desde la capa PHY.

Tabla 18 - Módulo MAC (Input/Output)

Módulo MAC_CPU

Puerto	Tamaño	E/S	Descripción
clk	1	E	Señal de reloj del módulo.
reset	1	E	Señal de *reset* del módulo, activa a nivel alto.
data_in	8	E	Señal de entrada de datos.
data_in_r	1	E	Señal de solicitud de entrada.
data_out	8	S	Señal de salida de datos
data_out_r	1	S	Señal de solicitud de salida de datos.
data_out_c	1	S	Señal de solicitud de salida de comandos.
req_get	1	S	Señal de solicitud hacia el módulo *get_phy_req*.
send_get	1	S	Señal de indicación de que el módulo está preparado para recibir los datos de la señal *response*.
phy_pib	8	S	Señal indicativa del parámetro PIB.
ack_get	1	E	Señal de entrada indicativa del progreso en el módulo *get_phy_req*.
data_available	1	E	Señal de disponibilidad de datos en el módulo *get_phy_req*.
pib_octets_get	4	E	Señal indicativa del número de octetos de la señal *response* que se deben capturar.
response_get	56	E	Señal de entrada de datos desde el módulo *get_phy_req*.

Tabla 19 - Módulo MAC_CPU (Input/Output)

Módulo GET_PHY_REQ

Puerto	Tamaño	E/S	Descripción
reset	1	E	Señal de *reset* del módulo, activa a nivel alto.

req	1	E	Señal de activación del módulo.
ack	1	S	Señal de indicación de funcionamiento del módulo.
pib	8	E	Señal de entrada del parámetro PIB.
arbiter_req	1	S	Señal de solicitud de acceso al bus de comunicación.
arbiter_grant	1	E	Señal de permiso de acceso al bus de comunicación.
data_to_phy	16	S	Señal de datos de salida.
data_to_phy_r	1	S	Señal de solicitud de salida de datos.
data_from_phy	8	E	Señal de entrada de datos.
valid	1	E	Señal de solicitud de entrada de datos.
clk_in	1	E	Señal de reloj de sincronización.
pib_octets	4	S	Señal indicativa del número de octetos válidos en la señal *response*.
data_available	1	S	Señal indicativa de datos disponibles.
send	1	E	Señal indicativa de que el módulo *mac_cpu* está preparado para recibir los datos.
response	56	S	Señal de salida de datos hacia el módulo *mac_cpu*.

Tabla 20 - Módulo GET_PHY_REQ (Input/Output)

Módulo ARBITER

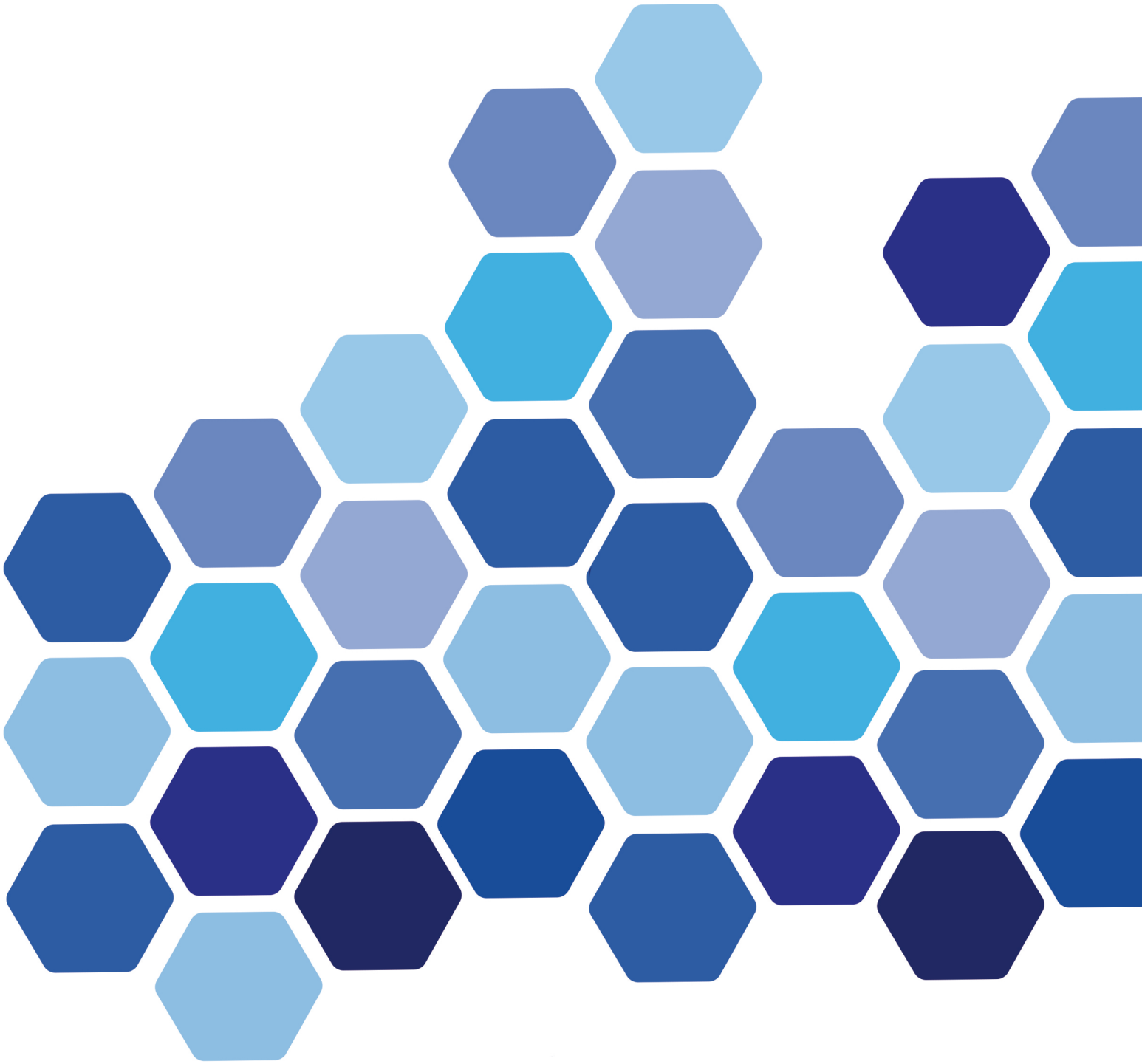
Puerto	Tamaño	E/S	Descripción
req	Variable	E	Señal de solicitud de acceso al bus de comunicación.
grant	Variable	S	Señal de permiso de acceso al bus de comunicación.

Tabla 21 - Módulo ARBITER (Input/Output)

Módulo BD

Puerto	Tamaño	E/S	Descripción
reset	1	E	Señal de *reset* del módulo, activa a nivel alto.
data_in	16	E	Señal de entrada de datos.
data_in_rq	1	E	Señal de solicitud de entrada de datos.
data_out	8	S	Señal de salida de datos.
data_out_rq	1	S	Señal de solicitud de salida de datos.
clk_out	1	S	Señal de reloj de sincronización de salida.

Tabla 22 - Módulo BD (Input/Output)



IUMA