Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

MÁSTER UNIVERSITARIO EN
SISTEMAS ELECTRÓNICOS AVANZADOS

TRABAJO FIN DE MÁSTER

PROCESADO DE COMUNICACIONES TIME-SENSITIVE NETWORKING (TSN) EN FPGA Y ANÁLISIS DE LAS MISMAS EN PC

Estudiante *Rodríguez Elorriaga, Igor*
Director/Directora *Bidarte Peraita, Unai*
Departamento *Tecnología Electrónica*
Curso académico *2019/20*

Bilbao, 14 de septiembre de 2020

Resumen

Hoy en día, la industria 4.0 es una realidad y junto con ella la tendencia de introducir el internet de las cosas (IoT, *Internet of Things*) y los CPS (*Cyber-Physical-Systems*) al mundo de las Tecnologías de la Operación (OT, *Operational Technology*), de la industria. Para esto, no basta con la tecnología hasta ahora utilizada en el ámbito de las tecnologías de la información (IT, *Information Technology*); ya que ésta no cumple con los requisitos específicos de la automatización industrial como son: la fiabilidad, la eficiencia y el determinismo. Como solución a este problema surge *Time-Sensitive Networking* (TSN). Un conjunto de estándares en curso dentro del grupo de estandarización de IEEE, para garantizar comunicaciones en tiempo real y con baja latencia. Por otro lado, también ofrece mayor ancho de banda, rango físico, espacio de direccionamiento, etc., evitando así limitaciones, que protocolos de campo como CAN o PROFIBUS presentan. TSN permite adaptarse de esta manera a la industria 4.0 y comunicarse o controlar datos en tiempo real, siendo ésta una de las mayores demandas por parte de la industria hoy en día.

Por otro lado, junto a estos estándares de IEEE es necesario un hardware que sea capaz de obtener el beneficio completo que TSN ofrece. Los sistemas basados únicamente en microprocesador presentan problemas de eficiencia y fiabilidad en comunicaciones con alta tasa de transferencia y tramas de gran tamaño. Por otro lado, los circuitos integrados para aplicaciones específicas (ASIC, *Application-Specific Integrated Circuit*) presentan problemas de obsolescencia frente a la constante evolución de los estándares de TSN. Finalmente, la utilización de un SoC (*System-on-Chip*) con FPGA (*Field-Programmable Gate Array*), que integra un procesador y la parte de lógica programable, resulta ser la solución más completa. Esta, añade flexibilidad ya que permite la reconfiguración y la FPGA permite llevar a cabo una comunicación en tiempo real fiable.

Por otra parte, los estándares TSN no son soportados en numerosos dispositivos ya conectados a la red, como tarjetas de red estándares y eso puede provocar una pérdida de información constante. De hecho, equipos de análisis de redes o PC-s no especializados no procesan adecuadamente las tramas, llegando a eliminar ciertas cabeceras de los paquetes y perdiendo de esa manera información importante de los mismos.

Teniendo esto en cuenta, en este proyecto se pretende realizar un IP Core que se pueda integrar en un switch de TSN implementado en un SoC con FPGA. Este IP Core, permitirá encapsular tráfico TSN en tramas de ethernet estándar, para desencapsularlas posteriormente en un PC. Con ello, se podrá analizar el tráfico TSN en equipos no especializados. Además, como el equipo a utilizar es reconfigurable esto permitirá ajustar o modificar el encapsulado según TSN evolucione.

Palabras clave: TSN, FPGA, SoC, reconfigurable.

Laburpena

Gaur egun, 4.0 Industria errealitate bat da eta berarekin industriako Operazio teknologien mundura, OT (*Operational Technology*), bai gauzen interneta (IoT, *Internet of Things*), zein CPS-ak (*Cyber-Phisycal-Systems*) sartzeko joera. Horretarako, ez da nahikoa informazio teknologiaren eremuan (IT, *Information Technology*) orain arte erabilitakoekin; zeren eta hauek ez dituzte betetzen ondoren zerrendaturiko automatizazio industrialaren berariazko baldintzak: fidagarritasuna, eraginkortasuna eta determinismoa. Arazo honi aurre egiteko TSN-a (*Time-Sensitive Networking*) sortzen da. IEEE estandarizazio taldearen barnean garatzen hari diren estandar multzo honetan, komunikazioak denbora errealean eta latentzia baxuan bermatzen dira. Horretaz gain, banda-zabalera, tarte fisiko, helbideratze espazio, etab. handiagoa eskaintzen du, CAN edo PROFIBUS bezalako eremu protokoloek aurkezten dituzten mugak saihestuz. Modu honetan TSN-ak 4.0 industriara moldatzea eta denbora errealean komunikatzea edo datuak kontrolatzea ahalbidetzen du, hau izanik egungo industriaren eskaera handienetariko bat.

Bestalde, IEEE-ko estandarrekin batera TSN-aren onura guztiak ateratzeko gai den hardwarea beharrezkoa da. Mikroprozesadoreetan bakarrik oinarrituriko sistemek eraginkortasun eta fidagarritasun arazoak izaten dituzte bilbe handiekin eta transferentzia indize altuko komunikazioetan. Beste alde batetik, aplikazio espezifikoetarako zirkuitu integratuetan oinarriturikoek (ASIC, *Application-Specific Integrated Circuit*) zaharkitze arazoak aurkezten dituzte TSN-aren estandarren etengabeko bilakaera dela eta. Azkenik, FPGA duten SoC-en (System-on-Chip) erabilera, prozesadorea eta zati logiko programagarria integratzen dutenak, irtenbide egokienean bilakatzen da. Honek, malgutasuna gehitzen du birkonfigurazioa ahalbidetzen baitu eta FPGA-k denbora errealean buruturiko komunikazio fidagarria ahalbidetzen du.

Honetaz aparte, TSN estandarrak ez dira jasaten dagoeneko sarera konektaturik dauden gailu askotan, sare txartel estandarrak bezala eta honek etengabeko informazio galera sor dezake. Izan ere, sareen analisirako ekipamenduek edo espezializatu gabeko PC-ek ez dituzte bilbeak behar bezala prozesatzen, batzuetan paketeen goiburu batzuk ezabatuz eta horrela informazio garrantzitsua galduz.

Hau kontuan izanik, proiektu honetan FPGA dun SoC batean inplementaturiko TSN switch batean integratu ahal den IP Core bat sortzea da helburua. IP Core honek, TSN trafikoa Ethernet estandarreko bilbetan enkapsulatzea ahalbidetuko du, geroago PC batean desenkapsulatu ahal izateko. Horrekin, TSN trafikoa, espezializaturik ez dagoen ekipoarekin analizatu ahal izango da. Gainera, erabiliko den ekipoa birkonfiguragarria denez, TSN estandarrak garatu ahala enkapsulazioa moldatzeko aukera emango du.

Hitz-gakoak: TSN, FPGA, SoC, birkonfiguragarria.

Abstract

Nowadays, 4.0 Industry is a reality and with it the tendency to introduce the Internet of Things IoT as well as the Cyber-Physical-Systems CPS to the world of Operational Technologies. To accomplish this on a wider scale, it is not enough to utilise the technology currently used in the scope of the Information Technologies; because this does not fulfil the specific requirements of industrial automation, which are, in summary: reliability, efficiency, and determinism. Time-Sensitive Networking (TSN) arises as a solution to this problem. TSN offers a set of standards within the IEEE standardization group, which aim to ensure real-time communications with low latency. At the same time TSN also offers a higher bandwidth capacity, physical range, addressing space, etc. thus avoiding the limitations, inherent to fieldbus protocols in current use, such as CAN or PROFIBUS. TSN allows us to adapt to the 4.0 Industry and enable communications or control of data in real time, which have proven to be one of the greatest challenges and demands in the industry today.

To fully implement TSN, together with the aforementioned IEEE standards, requires hardware that is capable of obtaining the full benefit that TSN offers. On the one hand, systems based solely on microprocessors unfortunately present efficiency and reliability problems in communications with high transfer rates and large frames, while systems based on the Application-Specific Integrated Circuit (ASIC) present obsolescence problems in the face of the constant evolution of TSN standards on the other. Thus, the use of a SoC (System-on-Chip) with FPGA (Field-Programmable Gate Array), which integrates a processor as well as a programmable logic part, turns out to be the most complete solution. This adds flexibility as it allows reconfiguration and the FPGA enables reliable real-time communication.

Additionally, the TSN standards are not supported by many current devices with networking capability, such as standard network cards, and this can cause a constant loss of information. In fact, non-specialized network analysis equipment or PCs do not adequately process the frames while utilising TSN standards, which lead them to eliminate certain packet headers and thus losing important information.

Taking this into account, this project aims to produce an IP Core that can be integrated into a TSN switch that at the same time is implemented in a SoC with FPGA. Such a set up will allow the encapsulation of TSN traffic in standard Ethernet frames, to be later decapsulated on a PC. Through such a solution, it will be possible to analyse the TSN traffic with non-specialized equipment. Furthermore, as the equipment that is used is reconfigurable, this will allow adjustment or the modification of the encapsulation process as TSN evolves.

Keywords: TSN, FPGA, SoC, reconfigurable.

Índice

Lista de tablas

Lista de figuras

Lista de Acrónimos

APERT	Applied Electronics Research Team
APU	Application Processor Unit
ASIC	Application-Specific Integrated Circuit
C-API	C/C++ Application Programming Interface
CFI	Canonical Format Indicator
CLB	Configurable Logic Blocks
COE	Coefficient
CPLD	Complex Programmable Logic Device
CPS	Cyber-Physical-Systems
CRC	Cyclic Redundancy Check
DDR	Double-Data-Rate
DMA	Direct Memory Acces
DRC	Design Rule Checking
DSP	Digital Signal Processing
EMIO	Extendable multiplexed I/O
FCS	Frame Check Sequence
FIFO	First In, First Out
FPGA	Field-Programmable Gate Array
FSBL	first stage boot loader
GMII	Gigabit Media Independent Interface
ICMP	Internet Control Message Protocol
ILA	Integrated Logic Analyzer
IOP	I/O Peripherals
IoT	Internet of Things
IT	Information Technology
MAC	Media Acces Control
NOP	NoOperation
OMC	on-chip-memory
OT	Operational Technology
PAL	Programmable Array Logic
PCAP	Processor configuration Acces port
PCIe	Peripheral Component Interconnect Express
PCP	Priority code point
PHY	Physical layer
PL	Programmable Logic
PLD	Programmable Logic Devices
PS	Processing System
QoS	Quality of Service
RGMII	Reduced Gigabit Media Independent Interface
RTL	Register-Transfer Level
SFD	Start Frame Delimiter

SoC	System-onChip
SRAM	Static Random Access Memory
SSBL	second stage boot loader
TCP	Transmission Control Protocol
TPID	Tag Protocol Identifier
TSN	Time-Sensitive Networking
UDP	User Datagram Protocol
VHDL	Very High-Speed Integrated Circuit Hardware Description Language
VID	VLAN Identifier
VLAN	Virtual LAN

1. Introducción

La cuarta revolución industrial, en un inicio fue apodada como *Industrie 4.0* por el gobierno alemán [1]. Más tarde, en 2013 el grupo de trabajo de Industrie 4.0 (*Industrie 4.0 Working Group*) desarrollo las primeras recomendaciones para su implementación [2], que se siguen compartiendo hoy en día. En esta publicación, se enumeraban los siguientes componentes principales para la industrie 4.0: IoT (*Internet of Things*), CPS (*Cyber-Physical Systems*) y las fábricas inteligentes. Partiendo de que, IoT hace referencia a la interconexión de objetos inteligentes de uso cotidiano mediante la red [3] y CPS a la nueva generación de sistemas con capacidades físicas y computacionales integradas para interactuar con humanos [4]; combinando ambas emergen las fábricas inteligentes. Las cuales, producirán un incremento en la productividad y eficiencia gracias a la gestión remota e inteligente [5].

Por otro lado, para que esto sea posible, la automatización industrial requiere de fiabilidad, eficiencia y determinismo en las comunicaciones. Siendo estos, unos de los requisitos que TSN (*Time-Sensitive Networking*) cumple. TSN es un conjunto de estándares en desarrollo que nace del grupo de trabajo de IEEE 802.1 [6], y mediante el cual se pretende garantizar comunicaciones en tiempo real y con baja latencia [7]. Además, estos estándares reducen las limitaciones que los protocolos de campo como CAN o PROFIBUS utilizados actualmente presentan; ofreciendo mayor ancho de banda, rango físico, espacio de direccionamiento, etc. [8]. Con todo esto, TSN se presenta como una de las herramientas para la adaptación a la industria 4.0, permitiendo comunicarse y/o realizar un control en tiempo real.

En cuanto a la implementación de estos estándares, es importante que esta se realice en un hardware capaz de implementar las herramientas que TSN integra y con el rendimiento adecuado. Un sistema completamente basado en microprocesador puede llegar a presentar problemas de eficiencia y fiabilidad en comunicaciones de alta tasa de trasferencia con tramas de gran tamaño. Por otro lado, los circuitos integrados para aplicaciones específicas (ASIC, *Application-Specific Integrated Circuit*), a presar de no presentar problemas de rendimiento no permiten la reprogramación, por lo que no pueden adaptarse a los cambios que un estándar en evolución pueda introducir, volviéndose obsoletos. Por último, el uso de un SoC (*System-on-Chip*) con FPGA (*Field-programmable Gate Array*), parece ser la solución más adecuada. Este hardware que integra un procesador y una parte lógica programable añade una flexibilidad ante la posibilidad de ser reconfigurado, además de ofrecer las características necesarias para gestionar comunicaciones en tiempo real que sean fiables [9].

Por otra parte, actualmente los estándares de TSN no están soportados por la mayoría de dispositivos conectados a la red, como las tarjetas de red estándares de PCs o equipos de análisis de redes no especializados. Esto provoca que el tráfico TSN que llega a dichos dispositivos no se procese de manera adecuada, en algunos casos llegando a eliminar

cabeceras de paquetes o no aceptándolos, perdiendo con ello información importante de los mismos. Por ello, sería interesante disponer de alguna herramienta capaz de encapsular tráfico TSN en tráfico de Ethernet estándar. Ya que, el tráfico Ethernet estándar es aceptado por todos los dispositivos y una vez en el dispositivo destino desencapsularlo para obtener el tráfico TSN.

Teniendo esto en cuenta, en este proyecto se analizará, por un lado, el estado del arte de los estándares de TSN, junto con el del SoC Zynq-7000. Con ello, se llevará a cabo el estudio del tráfico a encapsular y la plataforma en la que se pretende implementar el diseño a elaborar en el proyecto. Seguidamente se diseñará un IP Core de encapsulado de tráfico TSN en Ethernet estándar, el cual será reconfigurable mediante PS, para que se pueda modificar en según los estándares evolucionan. A continuación, se llevará a cabo la implementación del diseño en el módulo SMARTZynq [10]. Por otro lado, se analizará el estado del arte de las herramientas Python y Scapy mediante las cuales se elaborará un script en Python para completar el desencapsulado en entorno PC.

Por último, como objetivo final del proyecto, se pretende implementar el IP Core diseñado en un producto real como es RELY-RB [11]. Este switch de la empresa RELYUM integra tecnología para evitar perdida de paquetes con Ethernet redundante, además de sincronización bajo microsegundos y ciberseguridad. Al añadir este IP Core al módulo, se pretende aumentar la funcionalidad que este ofrece.

1.1 Contexto

El Proyecto se ha realizado en el Grupo de Investigación de Electrónica Aplicada de la UPV/EHU, APERT (*Applied Electronics Research Team*). Este grupo, está conformado por profesores, tanto investigadores predoctorales y doctores, como estudiantes de grado y máster de la Escuela de Ingeniería de Bilbao. Las principales actividades que se llevan a cabo en el grupo son las siguientes:

- Colaboración en Proyectos de I+D financiados mediante convocatorias públicas de diferentes administraciones.
- Proyectos de I+D bajo contrato para empresas o instituciones dentro del ámbito de las líneas de investigación en las que trabaja el grupo.
- Formación para empresas en temas relacionados con las líneas de investigación del grupo.
- Asesoría tecnológica, estudios técnicos o informes sobre diferentes aspectos relacionados con las líneas de investigación del grupo.
- Realización de Tesis Doctorales, publicaciones en revistas internacionales, congresos, patentes, etc.

Teniendo esto en cuenta el grupo se enfoca en dos líneas de investigación. Por un lado, los circuitos reconfigurables y SoC, y por el otro, circuitos de potencia y control para

convertidores de energía. Concretamente, este trabajo se centra en la rama de circuitos reconfigurables y SoC, y en los circuitos digitales de comunicaciones TSN.

En este momento, la rama de investigación de electrónica digital de APERT está enfocada en la especialización de las siguientes ramas:

- Circuitos digitales de comunicaciones: En esta línea de investigación se estudian los sistemas de comunicaciones desde un punto de vista electrónico. Para ello, analizando entre otros: sistemas digitales reconfigurables que den solución a ciertas aplicaciones o Cores para el procesamiento de tramas a alta velocidad para buses normalizados.
- Circuitos digitales para industria 4.0: En esta línea se estudian los CPS. Mediante estos se pretende unir el mundo OT de la industria con el IT, además de ofrecer interconexión y sincronización entre dispositivos de distintos fabricantes.

Concretamente, este trabajo de fin de master se enmarca en el análisis y tratamiento de TSN. Un conjunto de estándares de comunicación que se comienza a utilizar en la industria y permite la comunicación en tiempo real y la priorización de ciertas tramas frente a otras. Por otro lado, también se analizarán varias herramientas de análisis y tratamiento de tramas de PC.

1.2 Objetivos y alcance del proyecto

En este proyecto se llevan a cabo dos objetivos principales. El primero, **realizar un procesado de comunicaciones *Time-Sensitive Networking* (TSN),** en el que se encapsularan tramas de estándares TSN, y el segundo, **Realizar un análisis de estas tramas en un PC,** para el cual se realizará un script de Python, que desencapsule las tramas TSN para su posterior visualización. Teniendo esto en cuenta, los objetivos secundarios a completar son los siguientes:

1. Realizar un estudio del estado del arte de TSN y las FPGAs mediante las bases de datos disponibles en la universidad.

2. Elaborar el circuito descrito en VHDL del Core, para el encapsulado de tráfico TSN recibido. En él, se generarán tramas que incluyan una cabecera estándar (MAC destino multicast/configurable, MAC origen fija/configurable, Ethertype propietario + la trama encapsulada + FCS estándar).

3. Integración del IP Core creado en un sistema con parte PS e interconexión mediante buses AXI.

4. Realizar la implementación del diseño en SmartZynq. Además, en la parte PS del mismo se hará correr un Sistema operativo de kernel Linux. Y con todo esto se comprobará la configuración remota del sistema.

5. Realizar un estudio del estado del arte de Python y la librería *Scapy*, que se usará para el tratamiento de las tramas en PC. Para ello, haciendo uso de las bases de datos disponibles en la universidad.

6. Elaboración de script en Python mediante la librería *Scapy*, que capture tramas, extraiga la trama encapsulada y la guarde en un fichero PCAP compatible con Wireshark.

7. Comprobación de como mediante el PS se controla la parte PL y como las tramas encapsuladas en la FPGA se reciben en el PC para su posterior desencapsulado y análisis.

Por lo tanto, el alcance de este proyecto es llevar a cabo un IP Core que permita convertir tráfico TSN en tráfico ethernet básico mediante el encapsulado y proporcionar una herramienta para la visualización de tramas TSN encapsuladas en PC. Además, con ello se crea una herramienta que puede llegar a implementar en un switch o equipo de análisis de tráfico TSN.

2. Estado del Arte

2.1 Estado del Arte de TSN

2.1.1 Introducción

Las comunicaciones en la industria habitualmente se conforman de mensajes en tiempo real y transmisiones fiables, ya que procesos físicos dependen de ellos. Y a pesar de que ya existan este tipo de comunicaciones, muchas de ellas son incompatibles entre sí. Esto genera que se hayan de usar diferentes soluciones en paralelo para cada tipo de comunicación. Ante esto surge *Time-Sensitive Networking*, el conjunto de estándares y proyectos publicados y en desarrollo del grupo de estandarización de TSN, parte del grupo de trabajo 802.1 de IEEE [6]. Estos se encargan de definir mecanismos que garanticen servicios deterministas para las redes de IEEE 802.

2.1.2 Estándares de TSN IEEE 802.1

El conjunto de estándares TSN es una herramienta flexible para los diseñadores de redes, de la cual se pueden utilizar los estándares necesarios, en el momento preciso, para la aplicación objetivo. Estos estándares se pueden clasificar por las características en las que se centra cada uno: tiempo y sincronización, baja latencia acotada, fiabilidad y gestión de recursos. A continuación, se mencionan algunos de los más importantes [12-14]:

A. Tiempo y Sincronización.

   El estándar IEEE 802.1AS-2011 permite una sincronización precisa entre nodos de la red. Incluso se ha demostrado una exactitud mejor a 1 μs. Sin embargo, la pérdida del maestro activo se soluciona con la elección de uno nuevo y esto puede provocar saltos de tiempo en algún nodo. Para solucionar esto, el estándar IEEE 802.1AS-Rev tiene como objetivo el mantenimiento de tiempo sincronizado para aplicaciones sensibles al tiempo. En este estándar, la redundancia está mejorada de manera que se consigue una transferencia continua de latencia baja y una recuperación sincronización rápida ante fallos.

B. Baja latencia acotada

   TSN mejora el funcionamiento en tiempo real mediante dos estándares. El primero, IEEE 802.1Qbv define una comunicación programada, para aprovechar tiempo sincronizado en decisiones de transmisión y reenvío de mensajes en la red. Y el segundo, IEEE 802.1Qbu, que permite que las transmisiones de prioridad puedan interrumpir transmisiones que no sean críticas.

C. Fiabilidad

   Diversos estándares de TSN contribuyen a la fiabilidad en la comunicación y lo hacen de distinta manera: Con el duplicado y la eliminación de paquetes para reducir la probabilidad de pérdida de paquetes, el control del camino (path) y la reserva del

ancho de banda, la detección de flujos que no se comportan bien y la aplicación de acciones de mitigación, y la configuración.

D. Gestión de recursos

La reserva y gestión de los recursos es la clave para conseguir redes deterministas. Múltiples estándares de TSN contribuyen de manera diferente, a la reserva y gestión del ancho de banda, aliviando así la carga de tráfico y haciendo que la transmisión sea más eficiente.

2.1.3 Dispositivos no especializados frente a TSN

Los estándares de TSN, partiendo de Ethernet estándar, añaden ciertos campos a los paquetes transmitidos. Por ejemplo, en la figura 1 se puede observar qué para el estándar IEEE 802.1Q, en el que se gestiona la prioridad de paquetes, se añade una cabecera de 4 bytes. En esta se introducen el TPID (*Tag Protocol Identifier*), que por defecto tiene el valor de 0x8100; el PCP (*Priority code point*), que define el nivel de prioridad de 0 a 7; el CFI (*Canonical Format Indicator*), que dependiendo de su valor 0 o 1 indica si la dirección MAC esta en forma canónica o no; y por último el VID (*VLAN Identifier*), que identifica con un valor entre 0-4095 el VLAN a la que la trama pertenece [15].

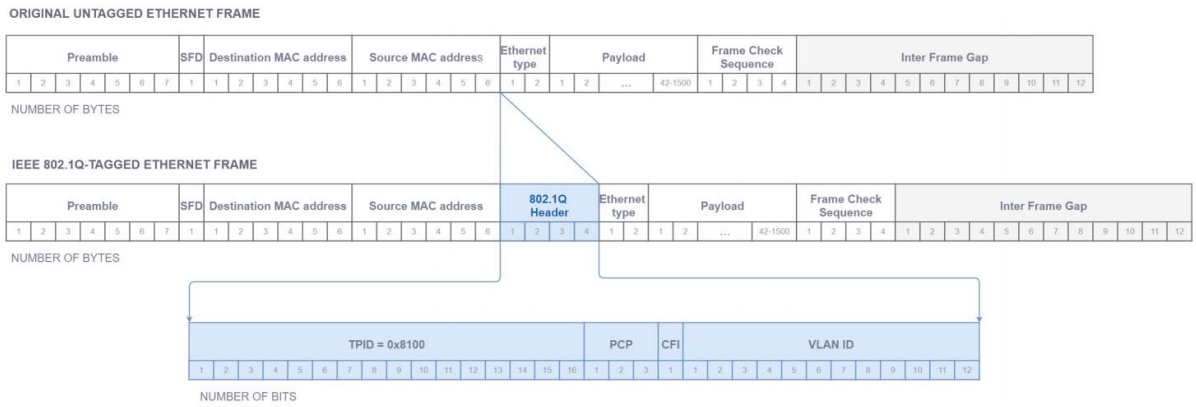

Figura 1. Trama Ethernet estándar y trama con tag de IEEE 802.1Q [15].

Con esta cabecera añadida al paquete, este deja de cumplir el formato de Ethernet estándar. Por ello, los dispositivos de tratamiento de tráfico no especializados pueden llegar a eliminar las cabeceras del paquete o a no aceptarlo, provocando de esta manera que se pierda su información y la funcionalidad para la que se ha introducido. Lo mismo puede suceder para los distintos estándares de TSN en desarrollo, cuando son tratados con dispositivos no especializados.

2.1.4 Implementación

Para llevar a cabo la implementación de control de tráfico TSN se necesita un dispositivo que permita reconfigurarse. De esta manera, se podrá adaptar a los cambios que los estándares recibirán, ya que la mayoría se encuentra en desarrollo. Debido a esto, los dispositivos como los ASIC no son adecuados para este tipo de aplicación, ya que habría que cambiar de dispositivo por cada actualización.

Por otro lado, los diseños basados exclusivamente en microcontroladores tampoco son adecuados ya que presentan problemas de eficiencia y fiabilidad, en comunicaciones con alta tasa de transferencia y tramas de gran tamaño [16].

Sin embargo, los dispositivos que comparten FPGA con un microprocesador, permiten llevar a cabo aplicaciones de tiempo real con baja latencia y además ofrecen la posibilidad de ser reconfiguradas.

Por todo ello, en este proyecto se ha utilizado un SoC con FPGA para la implementación del IP Core de encapsulado de tráfico TSN. Concretamente se ha utilizado un dispositivo que contiene un Zynq-7000 de la empresa Xilinx.

2.2 Estado del arte de las FPGA

2.2.1 Introducción

La evolución de los dispositivos lógicos programables (*Programmable Logic Devices*, PLD) más importantes, pasa por el uso de las matrices lógicas programables, PAL (*Programmable Array Logic*), que permitían integración a media escala mediante arrays de puertas lógicas programables por el usuario. A continuación, se introdujeron las CPLD (*Complex Programmable Logic Device*), que presentaban una combinación de matriz completamente programable de AND/OR y bancos de macroceldas [17]. Y finalmente, llegaron las FPGAs que, tras su evolución durante estos últimos años, se presentan como dispositivos semiconductores basados en matrices de bloques lógicos configurables CLB (*Configurable logic Blocks*), que se conectan mediante interconexiones programables. Entre ellas, también se encuentran disponibles las FPGAs programables una única vez, sin embargo, la clase más común es la de las reprogramables basadas en SRAM [18, 19].

En cuanto a la producción de las FPGA, dos empresas destacan ante las demás adquiriendo la mayor parte del mercado. Por un lado, el mayor fabricante que es Xilinx y por el otro Altera, que desde hace unos años forma parte de la compañía Intel. Y de entre sus dispositivos, este proyecto se va a realizar en la Zynq-7000, FPGA del fabricante Xilinx.

2.2.2 Zynq-7000

La familia de Zynq-7000 se basa en la arquitectura SoC de Xilinx. Este integra procesadores dual-core de ARM Cortex-A9 en la parte de PS (*Processing System*) y 28 nm de lógica programable PL (*Programmable logic*) en el mismo dispositivo. Las CPUs de ARM también incluyen una memoria en el chip, aparte de interfaces de memoria externa y una amplia variedad de interfaces de conexión de periféricos (figura 2).

La arquitectura de Zynq-7000 permite la implementación de lógica personalizada en la parte de PL y de software personalizado en la parte de PS. Además, gracias a la integración de PS con PL, consigue ofrecer un nivel de rendimiento que soluciones de dos chips no pueden igualar.

La parte de PL y la de PS tienen alimentaciones separadas, de manera que se puede cortar la corriente de la parte de PL para control de energía si es necesario. Por otra parte, los procesadores de PS siempre se encienden primero, con ello se consigue un control de la configuración de PL centrado en software. Para ello, existe una comunicación entre ambas partes, que permite el intercambio de información [20].

Componentes

Los componentes principales que la parte PS integra en la Zynq-7000 (figura 2) son los siguientes: La unidad de procesador de aplicación o APU (*Application Processor Unit*), la unidad de interfaz de memoria, y los periféricos de I/O (IOP, *I/O Peripherals*).

En cuanto a la parte PL, esta integra principalmente CLBs, block RAM de 36kb, Slices de DSP (*Digital Signal Processing*), bloques I/O programables, transceptores de baja potencia en equipos seleccionados, bloque integrado para PCIe en equipos seleccionados, 2 convertidores analógico digital (XADC) y el módulo de configuración de PL.

El interfaz que comunica PS con PL y viceversa incluye: Interfaces de AMBA AXI para comunicaciones de datos principales; DMA, interrupciones y señales de eventos; I/O multiplexadas extensibles (EMIO, *Extendable multiplexed I/O*), que permiten a los periféricos no mapeados de PS acceder a I/O de PL; Relojes y resets; y configuraciones y más, entre los que se encuentran, los interfaces XADC y JTAG, y el puerto de acceso de configuración de procesador (PCAP, *Processor configuration Access port*), que permite la configuración parcial y completa de PL, y la desencriptación y autentificación del inicio seguro de PS [20].

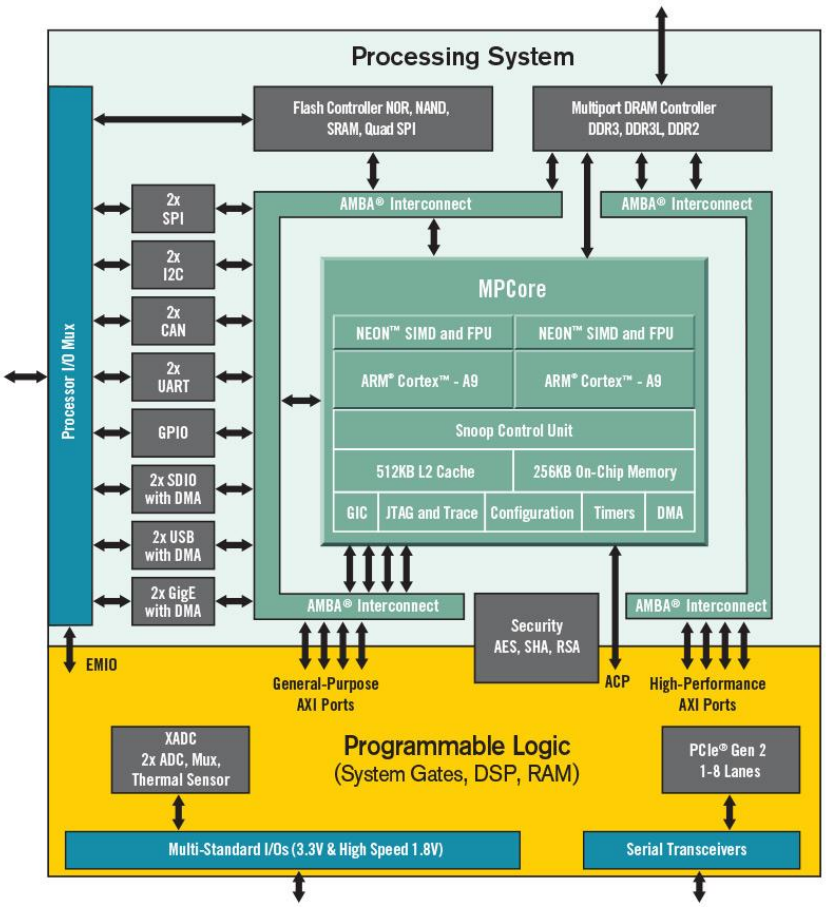

Figura 2. Arquitectura ZYNQ-7000 [20].

Interconexión

La unidad de procesador de aplicación, la unidad de interfaz de memoria, y los periféricos de I/O, están conectados entre sí y a la parte PL mediante AMBA AXI interconnect multicapa de ARM. La interconexión sin bloqueo soporta múltiples transacciones de maestro-esclavo.

La interconexión está diseñada con maestros sensibles a latencia, como la CPU de ARM. También, con los caminos más cortos a la memoria y maestros de con ancho de banda crítico, como un posible maestro de PL. Obteniendo de esta manera, conexiones de gran *throughput* con los esclavos a los que comunican.

El tráfico de la interconexión se puede regular mediante el boque de servicio de calidad, QoS (Quality of Service). Este, controla el tráfico generado por la CPU, el controlador de DMA y una entidad que representa los maestros en los periféricos de I/O [20].

Configuración

En el arranque, uno de los CPU Cortex-A9 de ARM ejecuta el código de la memoria ROM y copia a la memoria del chip (OMC, *on-chip memory*) el FSBL (*first stage boot loader*).

Después de ello, el procesador ejecuta el FSBL e inicia el arranque de PS. En algunos casos, se carga la configuración de PL también, o se puede dejar este paso para más adelante. Normalmente, el FSBL carga una aplicación de usuario o de manera opcional un SSBL (*second stage boot loader*), como U-Boot. A continuación, el SSBL continua con el arranque cargando código de cualquier objeto de carga primario o mediante interfaces como USB, Ethernet, etc. Finalmente, si la configuración de PL no se había realizado con el FSBL se puede realizar en esta etapa o hacerlo a continuación [20].

2.3 Estado del Arte de las herramientas Python y Scapy

2.3.1 Introducción

Para llevar a cabo el desencapsulado de las tramas Ethernet en entorno PC, se ha decidido analizar las herramientas que el lenguaje Python ofrece, ya que es un lenguaje de programación de alto nivel muy enfocado a aplicaciones científicas de investigadores e ingenieros [21]. Además, la herramienta Scapy está basada en lenguaje Python y permite realizar aplicaciones de gestión de tráfico de red en ese entorno.

2.3.2 Python

Este lenguaje de programación, originalmente diseñado por Guido van Rossum, ofrece las siguientes características generales [21]:

- Es un lenguaje de libre acceso y permite vender, utilizar o distribuir las aplicaciones creadas por uno mismo, sin necesidad de permisos adicionales.
- El hecho de que Python se pueda ejecutar en prácticamente todas las plataformas hace que las aplicaciones creadas con el mismo no tengan problemas de portabilidad.
- La clara sintaxis y las construcciones sofisticadas permiten escribir de manera procedural u orientada a módulos, según sea necesario.
- Un intérprete interactivo potente permite el desarrollo de código en tiempo real, eliminando de esta manera el paso de compilación del proceso de desarrollo.
- Python se puede embeber en una aplicación existente, lo que significa que puede añadir una capa de manera instantánea sobre una aplicación confiable.
- La capacidad de interactuar con una amplia variedad de programas diferentes, permite aprovechar los conocimientos en otros softwares.
- Su amplia gama de librerías tanto instaladas en la librería estándar o descargadas adicionalmente, permiten construir programas sofisticados de manera rápida.

- La relación de Python con las herramientas estándares de GUI (*Graphical User Interface*) permite realizar desarrollos de interfaces de manera rápida.

- En la página web, www.python.org/pypi se puede encontrar un repositorio de módulos de Python organizados, que pueden simplificar la gestión del software.

De entre las características mencionadas, destacan principalmente la clara sintaxis, la utilidad de los objetos integrados, las funciones y clases, las librerías estandarizadas, la facilidad de extensión, *SciPy* y *NumPy* [21,22]:

Sintaxis clara

Un factor significativo de Python como lenguaje de computación para científicos e ingenieros es la claridad de la sintaxis, haciendo que un código sea fácil de entender y mantener. Algunos de los detalles que esta sintaxis incluye son la sangría definida para bloques de código, el uso extendido de los espacios de nombres (módulos), construcciones de bucles fáciles de leer, el manejo de las excepciones y las *strings* de documentación.

Objetos integrados útiles

En Python cada cosa es un objeto de un determinado tipo, y la manera estándar de construir un tipo es mediante la llamada de una función. Sin embargo, ciertos tipos se construyen mediante sintaxis simplificada ya integrada. Entre los tipos integrados se pueden encontrar, por ejemplo: *integer, floating point y complex*.

Por otro lado, una manera de clasificar cualquier objeto de Python son las listas, que pueden contener incluso objetos de su mismo tipo. Además, con ellas se pueden llegar a construir arrays multidimensionales y se pueden editar según se quiera.

Por último, y entre otros objetos que quedan por mencionar están los *strings*, que son uno de los objetos más ampliamente utilizados en los códigos de Python.

Funciones y Clases

Además de ofrecer una sintaxis clara, Python también contribuye a la construcción de un código que se pueda mantener, mediante el uso de módulos, clases y funciones, que permiten la separación del código en grupos.

Por su parte, los módulos son código de Python agrupado en un archivo de extensión .py, y habitualmente contienen gran cantidad de clases o funciones relacionadas. Después, una vez utilizado el comando *import* en un módulo, se pueden utilizar las funciones, clases o variables definidas en el mismo utilizando la notación del punto.

En cuanto a las funciones, estas normalmente se definen de la siguiente manera: "def nombre_de_función(...): ‹bloque indentado›". Y para funciones simples se puede usar la expresión lambda que permite escribir funciones de una línea.

Python soporta estilos de programación tanto orientada a objetos como procedimental. En cada módulo, se pueden definir funciones para implementar comportamientos o crear nuevos objetos definiendo clases. Estos nuevos objetos pueden tener métodos, atributos e incluso métodos especiales que enseñan a Python cómo interpretar la sintaxis especifica de un objeto.

Las clases contienen atributos a los que se accede mediante notación de punto, *"object.attribute"* y los métodos son atributos que pueden ser llamados como las funciones. Estos se definen dentro de un bloque de clase para obtener el objeto como primer argumento. Por otro lado, métodos especiales se definen entre guiones bajos "_" para indicar que el intérprete de Python las llama en situaciones especiales.

Librerías estándares

Python trae un grupo de librerías integrado las cuales añaden una gran cantidad de funcionalidades a lenguaje. Entre ellas se pueden encontrar: librerías de tiempo, generación de números aleatorios, compresión y descompresión de archivos, gestión de archivos .csv, escaneos de puertos nmap, módulos para trabajo con URL, módulos que proveen funcionalidades dependientes de sistemas operativos, etc. Todas ellas y más se pueden encontrar en.

Facilidad de extensión

El hecho de que Python no sea una herramienta lo suficientemente rápida para la realización de ciertos cálculos, no descarta a este lenguaje como herramienta valida. Python es un leguaje fácilmente extensible mediante C-API (*C/C++ Application Programming Interface*). Esta herramienta, permite que los programadores de C y C++ puedan escribir módulos de extensión o embeber Python.

Un módulo de extensión es una librería compartida que replica un módulo de Python con variables, funciones y atributos de clases. Este se crea con una función de una única entrada, a la cual Python llama cuando es importado. Este punto de entrada configura el módulo añadiendo constantes, tipos nuevos definidos, y la tabla de funciones del módulo, la cual interconecta los nombres de las funciones del módulo y las funciones de C a las que ha de llamar.

SciPy

SciPy como tal, hace referencia a más de un concepto en concreto. Como comunidad se refiere a las personas que desarrollan el conjunto de herramientas de SciPy; pero también se puede hacer referencia tanto a las librerías o al ecosistema SciPy.

En cuanto al ecosistema de SciPy, la comunidad científica de Python, partiendo del mismo lenguaje hay varios paquetes que la conforman:

- *NumPy*, un paquete fundamental para la computación numérica [23].
- La librería *SciPy*, un conjunto de algoritmos numéricos y herramientas de dominio especifico, que pueden incluir: procesado de señales, optimización, estadística y más.
- *Matplotlib*, es un paquete para la elaboración de graficas tanto en 2D como 3D.

Partiendo de esto, el ecosistema *SciPy* incluye herramientas tanto generales como especializadas para el tratamiento de datos y la computación, la experimentación productiva y la computación de alto rendimiento [24].

2.3.3 Scapy

Scapy es un programa basado en Python desarrollado por Philippe Biondi [25]. Este, es capaz de enviar, interceptar y analizar, y crear paquetes de red. Con estas opciones, permite crear herramientas para analizar, escanear o atacar redes. Por otro lado, como set de módulos de Python, se pueden crear scripts de Python en los que importen estos módulos o también se puede ejecutar Scapy directamente e interactuar con él.

Lo que convierte a Scapy en una herramienta diferente es que permite una simplicidad de código alta comparado, por ejemplo, con un código en c de una aplicación de gestión de tramas. Además, ofrece una gran flexibilidad en la elaboración de paquetes. De hecho, Scapy permite enviar tramas de paquetes erróneos o que no tengan una estructura lógica. Ya que, no interpreta las respuestas recibidas o los paquetes enviados, simplemente los descodifica y obtiene resultados [26].

Crear Paquetes

Los paquetes se crean mediante instanciación de clases. Cuando un objeto se instancia se crea un paquete y cuando se modifican sus atributos o se llama a los métodos del objeto instanciado se está manipulando el paquete. Si se implementan paquetes como objetos, los paquetes se definen mediante una línea de código, como se ve en los siguientes ejemplos.

```
>>> Ether()/IP()/TCP()
<Ether type=0x800 |<IP frag=0 proto=TCP |<TCP |>>>
>>> IP()/TCP()/"GET / HTTP/1.0\r\n\r\n"
<IP frag=0 proto=TCP |<TCP |<Raw load='GET / HTTP/1.0\r\n\r\n' |>>>
```

En Scapy se pueden describir paquetes o un conjunto de paquetes que tengan capas apiladas. A pesar de que estas capas tengan valores predeterminados, Scapy permite al usuario introducir los valores que este quiera. De esta manera, no hace falta reescribir un

código completo cuando las características del entorno de la aplicación desarrollada cambian [26].

Valores predeterminados

Esta herramienta trata de utilizar valores predeterminados para todos los paquetes, siempre que no se infiera ninguno. Entre ellos están [26]:

- La dirección IP de origen, la cual se elige en base al destino y la tabla de rutado.
- El checksum es calculado.
- La dirección MAC de origen se determina por la interfaz de salida.
- El Ethernet Type y el protocolo IP se determinan mediante la capa superior que tengan.

Por otra parte, otros valores de ciertos campos son seleccionados de manera que sean más útiles:

- TCP puerto de origen 20 y puerto de destino 80.
- UDP Puerto de origen y destino 53.
- ICMP type Como solicitud eco

Funciones principales

Las funciones principales de Scapy se han organizado en 3 grupos diferentes: Funciones de básicas, funciones de análisis de un paquete y funciones de gestión de un conjunto de paquetes.

Entre las funciones básicas se pueden encontrar funciones de ayuda o información, diferentes métodos para enviar y recibir paquetes, o realizar otras operaciones entre otras (tabla 2.1).

Tabla 2.1. Tabla de operaciones básicas

Función	Descripción
help()	Menú de ayuda
ls()	Lista de capas disponibles
lsc()	Lista de funciones disponibles
send()	Envía paquetes en la capa 3
sendp()	Envía paquetes en la capa 2
sr()	Envía y recibe paquetes en la capa 3
srp()	Envía y recibe paquetes en la capa 2
sr1()	Como sr() pero solo recibe la primera respuesta
srp1()	Como srp() pero solo recibe la primera respuesta
sniff()	Realiza un "Sniffing" de paquetes
AsyncSniffer()	Realiza un "Sniffing" de paquetes de forma asíncrona
traceroute()	Función de trace route
arping()	Envío de solicitud ARP para determinar equipos conectados

Por otro lado, entre las funciones de análisis de un paquete, se pueden encontrar diferentes maneras de manipular o visualizar un paquete (tabla 2.2).

Tabla 2.2. Funciones de operación con 1 paquete

Función	Descripción
raw(pkt)	Ensambla el paquete
hexdump(pkt)	Hacer un hexdump del paquete
ls(pkt)	Obtiene valores de los campos del paquete
pkt.summary()	Vista reducida del paquete
pkt.show()	Vista avanzada del paquete
pkt.show2()	Igual que show pero en un paquete ensamblado
pkt.sprintf()	Imprime un string con formato con los valores del paquete
pkt.decode_payload_as()	Cambia la forma de decodificar el paquete
pkt.psdump()	Dibuja un diagrama de PostScrip de manera especificada
pkt.pdfdump()	Dibuja un PDF de manera especificada
pkt.command()	Devuelve comando de Scapy que puede generar el paquete

Finalmente, las funciones de gestión de conjuntos de paquetes incluyen funciones de visualización, de organización o filtrado (tabla 2.3).

Tabla 2.3. Funciones de operaciones con conjuntos de paquetes

Función	Descripción
summary()	Muestra una lista de resúmenes de cada paquete
nsummary()	Igual que summay, pero de un numero de paquetes determinado
conversations()	Muestra un gráfico de conversaciones
show()	Muestra la representación preferida
filter()	Devuelve una lista de paquetes filtrada por una función lambda
hexdump()	Devuelve el hexdump de todos los paquetes
hexraw()	Devuelve el hexdump de la capa de Raw de todos los paquetes
padding()	Devuelve un hexdump de paquetes con relleno
nzpadding()	Devuelve un hexdump de paquetes que no tengan 0 relleno
plot()	Traza una función lambda aplicada a una lista de paquetes
make table()	Muestra una tabla en función de la expresión lambda

Con todas las funciones mencionadas no se llega a analizar todas las funciones disponibles en esta herramienta, pero si las suficientes como para empezar a diseñar una aplicación.

3 Desarrollo

3.1 Plataforma de diseño de IP Core

3.1.1 Introducción a Vivado

El diseño del IP Core realizado se ha completado utilizando la herramienta Vivado Design Suite. Esta plataforma de diseño de Xilinx ofrece diferentes maneras de realizar tareas relacionadas con el diseño, la implementación y la verificación de dispositivos del fabricante. Para ello, se pueden llevar a cabo diferentes flujos de diseño (figura 3).

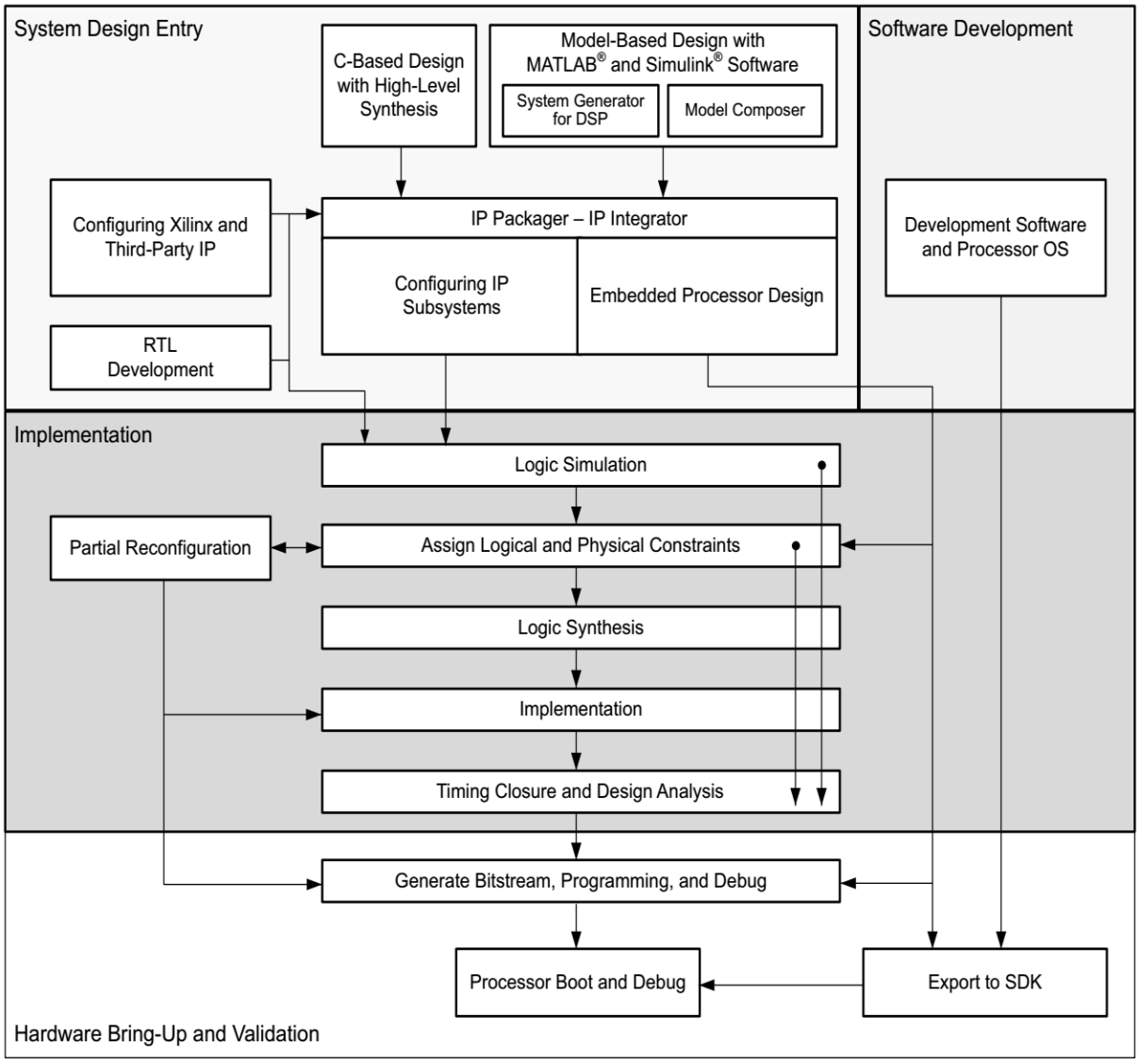

Figura 3. Flujos de diseño en Vivado [27].

3.1.2 Flujo de diseño e implementación de IP Core encapsulado.

Vivado ofrece la posibilidad de utilizar IPs de Xilinx, terceros o diseñados por los usuarios. A la hora de diseñar un IP propio ofrece la herramienta IP packager, que permite crear y empaquetar archivos de VHDL, constraints y/o datos en un IP de formato standard. Además, estos pueden integrar de base el estándar de AXI4 interconnect que permite una más rápida integración a nivel de sistema.

Sin embargo, antes de empaquetarlo hay que tener en cuenta si el IP creado funciona adecuadamente. Por ello, se recomienda realizar una simulación que permita visualizar el funcionamiento del IP y también comprobar que la síntesis e implementación del diseño se completa correctamente.

Simulación

La simulación se puede realizar antes que la síntesis y lo que principalmente requiere es un Testbench. Mediante este, se emulan las señales que el sistema obtendrá una vez implementado y se comprueba que las respuestas obtenidas son como se espera.

Síntesis

La síntesis en Vivado realiza un proceso de transformación en el que el diseño que está a nivel de RTL *(Register-Transfer Level)* se transforma en una representación a nivel de puertas. Como resultado se obtiene una Netlist de los componentes que conforman el circuito descrito. Por otro lado, en este punto en el diseño, Vivado permite introducir un analizador lógico interno (ILA), para analizar las señales seleccionadas en el dispositivo.

Implementación

Con la netlist obtenida del proceso de síntesis, la implementación de Vivado ofrece todas las herramientas necesarias para optimizar, localizar y rutar la netlist en recursos reales del dispositivo en el que se pretende realizar la implementación. La implementación funciona haciendo que se cumplan las restricciones lógicas, físicas y de tiempo del diseño.

Por otro lado, en la implementación se ofrecen algunas herramientas de análisis, que permiten testear distintos aspectos que influyen en el funcionamiento del diseño como: Análisis de tiempos, potencia, ruido, utilización de recursos o normas de diseño (DRC, Design Rule Checking) [27].

3.1.3 Flujo de diseño del sistema de encapsulado completo

Una vez el diseño del IP se ha añadido al catálogo de IP-s de Vivado, el IP se puede introducir en el sistema en el que se pretendía integrar. El IP se añade al diseño de bloques del sistema y conectan los pines de entrada y salida del mismo.

Cuando el diseño de bloques queda correctamente definido, se lleva a cabo la síntesis e implementación del sistema completo. Tras realizar el análisis de tiempos, consumo, recursos utilizados... necesario del diseño, solo queda realizar el Bitstream para su posterior implementación.

3.2 Diseño del Sistema de encapsulado de tramas TSN en FPGA

3.2.1 Introducción

En este apartado se lleva a cabo la descripción del diseño desarrollado para el encapsulado de tráfico TSN en tramas de Ethernet. Para ello, se analiza brevemente la plataforma utilizada y seguidamente se explica detalladamente el diseño del IP core.

3.2.2 Diseño del IP core

El IP Core diseñado (Anexo A) se sitúa entre dos PHYs (*Physical layer*) que lo conectan a la red, de manera que el tráfico que se introduce en el mismo es encapsulado y redirigido a la dirección MAC deseada, en paquetes de Ethernet estándar. El tráfico en ambos sentidos se dará haciendo uso del interfaz RGMII (*Reduced Gigabit Media Independent Interface*) [28]. Este interfaz es una alternativa al GMII (*Gigabit Media Independent Interface*), que reduce el número de pines necesarios y también funciona a 1 Gbit/s. Por otro lado, el Core también se comunicará con el procesador del Zynq permitiendo así la configuración del mismo (figura 4).

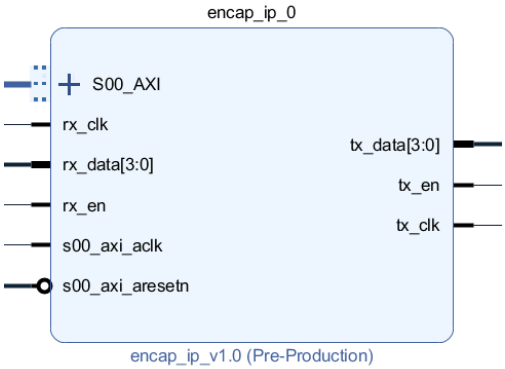

Figura 4. Bloque de IP de encapsulado.

Los datos de las tramas (rx_data/tx_data) introducidos mediante RGMII van sincronizados con un reloj (rx_clk / tx_clk), que tiene una frecuencia de 125 MHz, tanto en la recepción como en la transmisión de tramas. Además, la señal de control (rx_en / tx_en) define la transmisión de datos valida, cuando esta con valor lógico alto. Teniendo esto en cuenta y como se puede ver en la figura 5, los datos de 4 bits se reciben en el flanco

ascendente y descendente de la señal del reloj. Sin embargo, en este proyecto se hace uso de flip-flops de DDR (*Double-Data-Rate*) en la entrada, para conseguir sincronizar los datos por bytes en el flanco de subida del reloj.

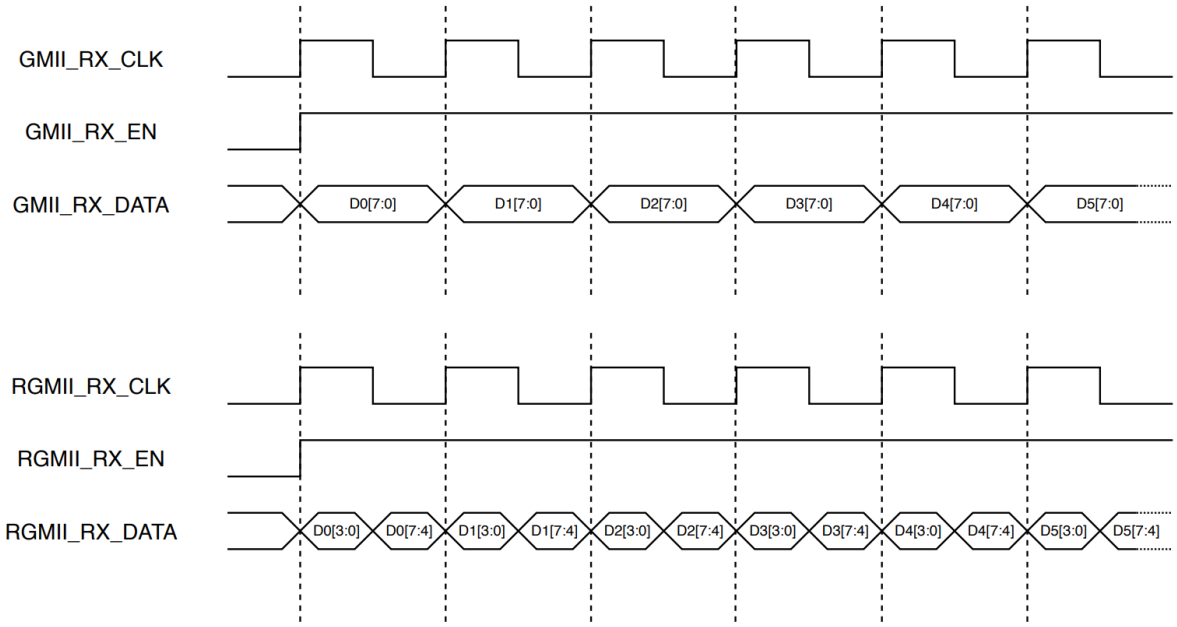

Figura 5. Señales GMII y RGMII.

A continuación, el funcionamiento del modelo de encapsulado queda definido por una máquina de estados y un diagrama de bloques (figura 6 y 7). El funcionamiento comienza por el estado de espera, en el que se mantendrá hasta que llegue la señal de habilitador de recepción (rx_en_i).

En cuanto la señal que advierte de la llegada de datos validos se habilita, se pasa al estado trama1, en el que los datos recibidos, es decir, el tráfico TSN, se comienza a almacenar en una pila FIFO (*First In, First Out*). De igual manera, se comienza a transmitir el paquete de ethernet estándar, que contendrá la trama de TSN recibida, comenzando por el preámbulo y el byte de SFD. Sin embargo, para ello es necesario realizar la conversión de 8 a 4 bits para que sea un formato compatible con RGMII. Esto, se consigue utilizando de nuevo flip-flops de DDR, aunque en este caso, los datos de 8 bits que vienen sincronizados en el flanco de subida del reloj se dividen enviando en un flanco los 4 bits más significativos y en el otro los 4 menos significativos. Además de los datos (tx_data_o[3:0]), la señal de habilitador de comunicación (tx_en_o) y el reloj de sincronización (tx_clk_o) también son transmitidos. En el diagrama de bloques no se representa el cambio de formato de las señales de datos a formato RGMII.

En el momento en el que se han transmitido el preámbulo y SFD (preamble_SFD_done = '1') se pasa al estado trama2. En este punto, se habilita la señal (crc_en_i) que inicia el cálculo de CRC. Para ello, se ha utilizado un modelo de cálculo de CRC-32 en paralelo, que

permite realizar el cálculo en un ciclo de reloj [29]. Se introducen datos de 8 bits en 8 bits, comenzando por las direcciones MAC y el Ethertype, que están almacenados en registros. Al mismo tiempo, se comienzan a transmitir las direcciones MAC de destino y origen, y el Ethertype).

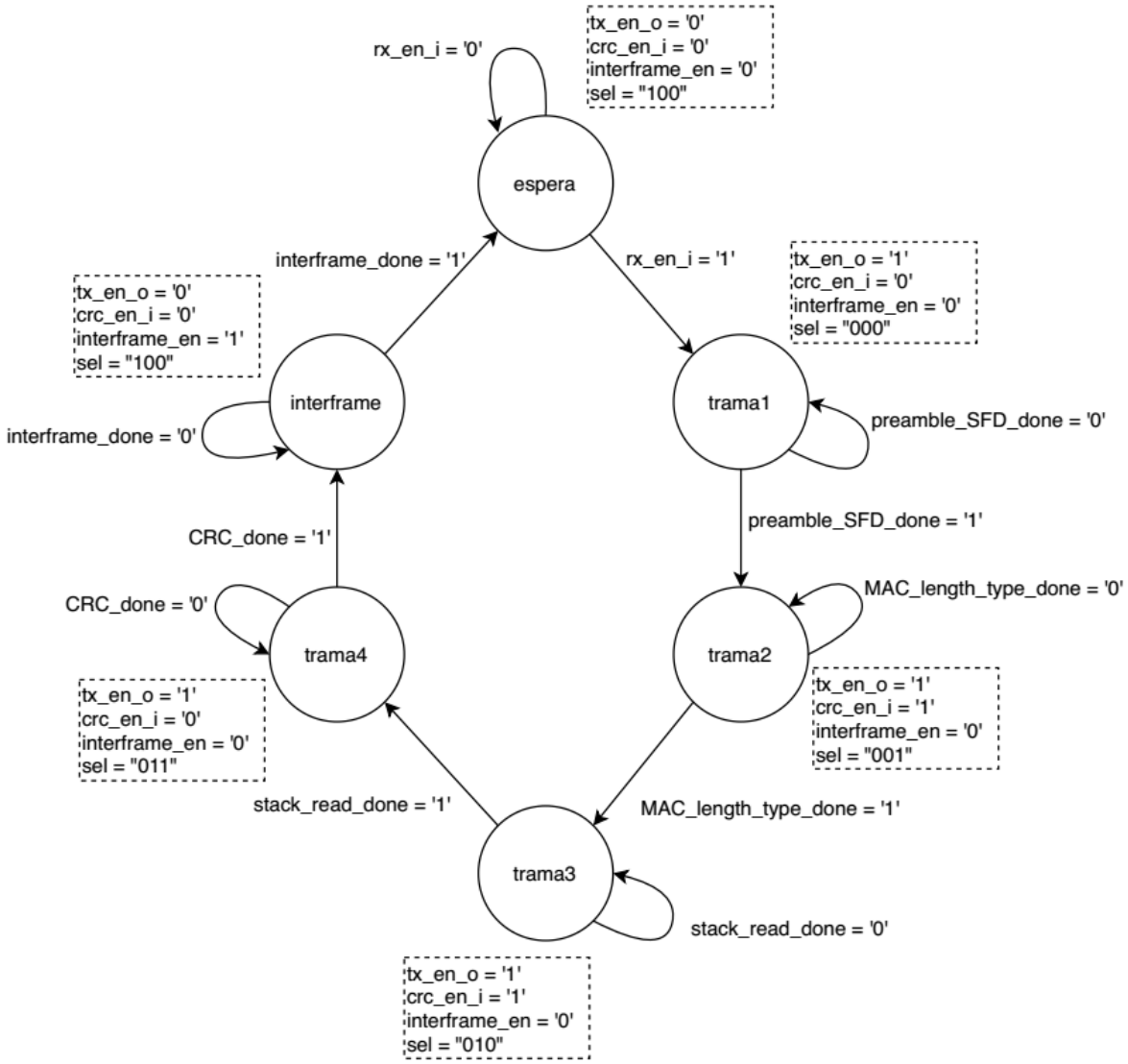

Figura 6. Máquina de estados del Core de encapsulado.

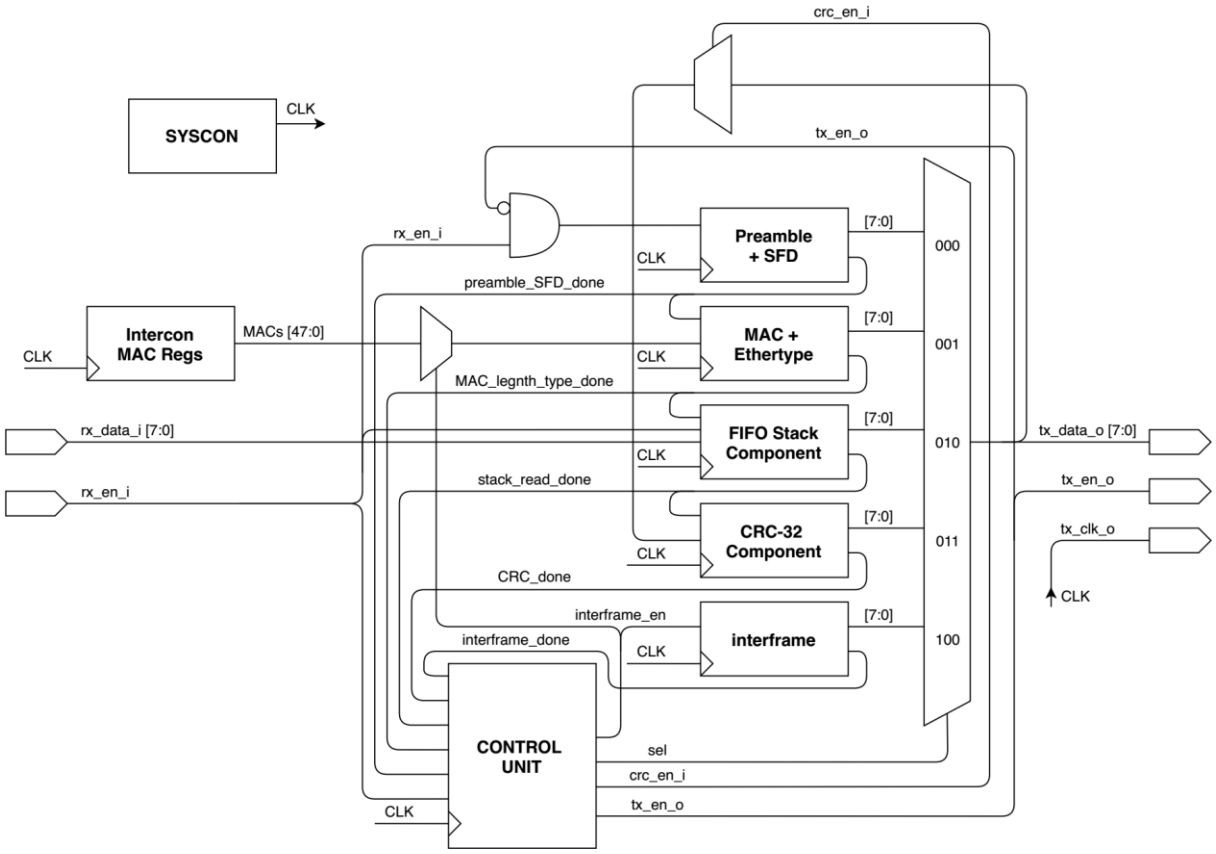

Figura 7. Diagrama de bloques de IP Core.

A continuación, una vez terminada la transmisión de las MAC y el Ethertype (MAC_length_type_done = '1') se llega al estado trama3. En este momento, se comienza a leer y a transmitir los datos de la trama de TSN que la pila contiene, mientras se continúa calculando el CRC del paquete encapsulado.

Cuando la lectura de la pila FIFO termina (stack_read_done = '1') se pasa al estado trama4. En este estado se deshabilita el cálculo de CRC y se transmiten los 32 bits del resultado obtenido.

Finalmente, tras haber completado el envío del CRC al completo (CRC_done = '1') comienza el estado de interframe en el que se mantendrá hasta que se complete el tiempo de espera necesario entre paquetes. Este tiempo de espera se llevará a cabo mediante un contador y una vez termine (interframe_done = '1') se volverá al estado de espera de inicio, en el que se podrá recibir el siguiente paquete.

A parte de esto, el IP Core se ha diseñado con un interfaz de buses AXI Lite esclavo, para permitir la interconexión con PS y posibilitar la configuración remota de las características del IP. Por ello, consta de las conexiones habituales de este interfaz. En concreto las características que se podrán configurar son las direcciones MAC destino y origen. Sin embargo, este interfaz de AXI Lite solo permite transferir datos de hasta 32 bits, por lo que la configuración de las direcciones MAC se realizara por partes. Por un

lado, los 24 bits más significativos y por el otro los menos significativos. Para poder saber de qué parte de dirección MAC destino u origen se trata habrá que introducir un numero en los bits 31-28 de la transmisión de datos de PS al IP de encapsulado.

Dependiendo de los números introducidos estos son los resultados:

- "0001" → Guarda los datos recibidos mediante AXI en los 24 bits de datos menos significativos de la dirección MAC destino.
- "0010" → Guarda los datos recibidos mediante AXI en los 24 bits de datos más significativos de la dirección MAC destino.
- "0011" → Guarda los datos recibidos mediante AXI en los 24 bits de datos menos significativos de la dirección MAC origen.
- "0100" → Guarda los datos recibidos mediante AXI en los 24 bits de datos más significativos de la dirección MAC origen.
- Los demás → No guarda los datos recibidos mediante AXI.

Por último, estos datos de configuración obtenidos solo se cargarán en los registros MAC de las tramas estándar de Ethernet una vez el IP se encuentre en estado de interframe (interframe_en = '1').

3.3 Diseño de desencapsulado de tramas Ethernet en PC

3.3.1 Introducción

La funcionalidad, que pretende ofrecer el script en Python para el desencapsulado de tramas Ethernet estándares en PC (Anexo I), es la siguiente: el programa capturará tramas de Ethernet en función de la MAC destino u origen de las mismas. Eso, lo decidirá el usuario antes de comenzar con la captura. Por otro lado, se deberá introducir el nombre de la interfaz en la que se quiera llevar a cabo la captura. Después de esto, el usuario decidirá cuando comenzar y detener la captura de tramas. Finalmente, una vez completada la captura se generará un archivo .pcap con el tráfico Ethernet desencapsulado. De esta manera, abriendo ese archivo mediante wireshark se podrán visualizar las tramas de tráfico TSN.

3.3.2 Desarrollo del script de Python

Para completar las funcionalidades mencionadas se han llevado a cabo los siguientes pasos:

En primer lugar, se realiza la importación de las librerías de Scapy, que se utilizarán para escuchar los paquetes ethernet determinados y desencapsularlos posteriormente.

```
import scapy.all as scapy
```

Después, se han inicializado las variables y listas que se van a utilizar. A continuación, se muestra cómo se pide por pantalla, que se introduzca el nombre de la interfaz por la que se realizara la captura; que se seleccione si se realizara el filtrado de la captura por MAC origen o destino y que se introduzca la dirección de la MAC:

```
in_iface = str(input("Specify interface on which to sniff packets: "))

while filter_ok == 0:
  in_dst_src = str(input("Filter src or dst MAC? "))
  if in_dst_src == 'src' or in_dst_src == 'dst':
    while mac_ok != 'y':
      mac_address = str(input("Enter the Mac address: "))
      print('The mac is -> ' + mac_address)
      mac_ok = str(input("Is it correct? y or n :"))
  if mac_ok == 'y':
    filter_ok = 1
```

Tras esto, se agrupan los datos obtenidos en un mismo string para que la función AsyncSniffer pueda utilizarlo como filtro:

```
filter_in = "ether " + in_dst_src + " " + mac_address

t = scapy.AsyncSniffer(filter=filter_in, iface=in_iface, store=True)
```

Una vez configurado el filtro para realizar la captura se han descrito las siguientes condiciones para que comience a capturar tráfico al introducir una 'g' y se detenga al introducir una 's'.

```
while start_in != 'g':
  start_in = str(input('Enter g to go: '))
t.start()

while stop_in != 's':
  stop_in = str(input('Enter s to stop: '))
a = t.stop()
```

Una vez terminado el proceso de captura se lleva a cabo el paso de desencapsulado de las tramas y su posterior escritura en un archivo .pcap:

```
for x in a:
  load = x[scapy.Raw].load
  c.append(load)

scapy.wrpcap("desencap_result.pcap",c)
```

3.4 Implementación de IP Encapsulado en SoC

3.4.1 Conexión a internet

Para este diseño, en el que se pretende poder configurar la FPGA remotamente y en el que se gestiona tráfico de la red, es evidente que la conexión a internet ha de estar configurada. Por ello, a modo de introducción se ha tratado de conectar una tarjeta de evaluación Zedboard a la red de internet de internet de la universidad. Para completar esta tarea inicialmente se han planteado diferentes opciones: Por un lado, clonar la dirección MAC de un equipo con acceso a la red al que se conectara la Zedboard o utilizar dispositivos auxiliares.

En primer lugar, se ha optado por llevar a cabo la conexión mediante dispositivos auxiliares. Para ello, siguiendo el esquema de la figura 9 se han dado los siguientes pasos:

- Se ha instalado el sistema operativo Ubuntu mate en la Raspberry. Este, permite que la Raspberry se conecte a la red WiFi eduroam y que comparta internet mediante cable Ethernet. Llegado a este punto, si se conecta la Zedboard por cable Ethernet a la Raspberry, se obtiene la conexión a internet. Sin embargo, no se puede acceder a la Zedboard con un PC conectado a la red eduroam.

- A continuación, se ha introducido un router en el sistema, para que funcione como Access point. De esta manera, se completa el esquema de la figura 8. Por otro lado, en la Zedboard se ha cargado un sistema operativo, en este caso Xillinux, que ajusta automáticamente la configuración IP.

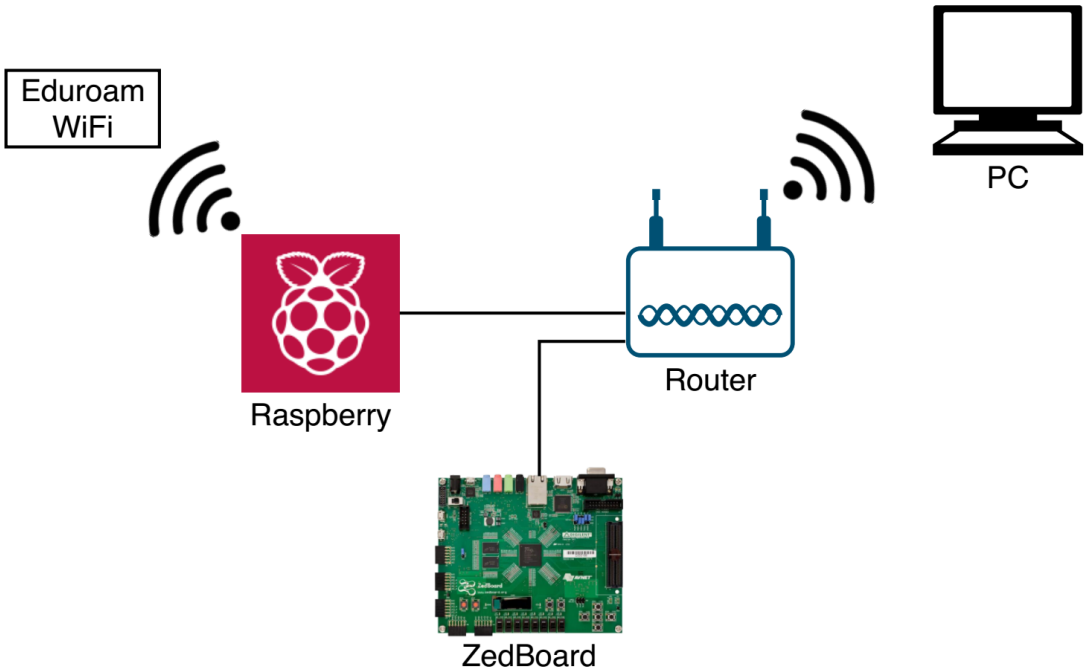

Figura 8. Diagrama de conexión de Zedboard a internet.

Después de esto, para conseguir una solución más robusta, se ha sustituido la Raspberry y el router por un switch. En este caso, el switch se conecta directamente la red de la UPV mediante Ethernet. Para ello, se le ha cambiado la MAC, por una válida para la red y tras conectar por ethernet la Zedboard al switch, esta, ya tiene acceso a internet. En caso de que se quiere acceder a la Zedboard con un PC, se puede conectar vía Ethernet mediante el switch. Finalmente, el router se ha conectado al switch también, así ofrece un punto de acceso mediante WiFi a la Zedboard.

Implementación Xillinux

Xilinux es una distribución de Linux basada en Lubuntu 16.04 para equipos con Zynq-7000 [30]. Es una plataforma de rápido desarrollo para proyectos que integran software y lógica. Además, este conjunto de programas que lo conforman, le dan casi las mismas funcionalidades que un ordenador que tenga Linux.

Para su implementación en Zedboard, mediante tarjeta SD, se han llevado a cabo los siguientes puntos:

- Descarga de la imagen de Xillinux y el kit de la partición de arranque de la página web xillybus.com/xillinux. La imagen comprimida en formato gzip tiene una tabla de particiones en la cual se encentran un sistema de archivos FAT, para los archivos del primer arranque, y otro de tipo ext4 para el sistema de archivos de raíz de Linux. En el kit de arranque están incluidos el boot.bin, el devicetree.dtb y el proyecto de vivado, con el que hay que generar el archivo bitstream (xillydemo.bit) del diseño que se implementa en la FPGA.

- Entre los archivos del proyecto de vivado hay que modificar el archivo xillydemo.vhd, de manera que se comentan las siguientes entradas:

```
PS_CLK : IN std_logic;
PS_PORB : IN std_logic;
PS_SRSTB : IN std_logic;
```

  Y se descomentan las señales que se pueden ver a continuación:

```
-- signal PS_CLK : std_logic;
-- signal PS_PORB : std_logic;
-- signal PS_SRSTB : std_logic;
```

- Seguidamente, se ejecuta desde vivado el script de Tcl llamado xillydemo-vivado.tcl que crea el proyecto de vivado. Y una vez finalice, solo queda generar el Bitstream.

- Llegado este punto, simplemente queda cargar los archivos correspondientes en la tarjeta SD. Para ello, hay que descomprimir la imagen descargada y copiarla en la SD con sus particiones. Hay que utilizar otro programa para ello, como por ejemplo *"usb image tool"*.

- Finalmente, solo queda copiar los 3 archivos restantes: boot.bin, devicetree.dtb y xillydemo.bit a la partición de arranque. Con ello, en esta partición deberían de quedar la librería kernel de Linux, uImage y los 3 archivos mencionados.

Figura 9. Escritorio Xillinux [30].

De esta manera, se ha conseguido integrar un proyecto con sistema operativo Linux en la Zedboard, al que se podía acceder mediante internet, siguiendo el esquema previamente explicado.

3.4.2 Implementación en modelo SMARTZynq

Una vez completado el diseño del IP de encapsulado y haber conectado la Zedboard a internet se ha llevado a cabo la implementación del IP encapsulado en un sistema compatible con SMARTZynq. Este, módulo creado por la empresa SoC-e (System-on-Chip engineering) está diseñado para permitir una fácil integración de las redes de Ethernet industrial. El módulo integra un ZC7020 Zynq SoC y 5 PHYs de Gigabit entre otros. Permite la implementación de switches e incluso el procesado de tramas de red mediante hardware utilizando IP cores, como es el caso.

El diseño completado (figura 10) está formado por un reloj que funciona a 125 MHz, la parte de PS que se comunica con el IP de encapsulado mediante AXI interconnect y el IP de encapsulado, que aparte de estar conectado a AXI interconnect tiene las señales de entrada y salida de RGMII.

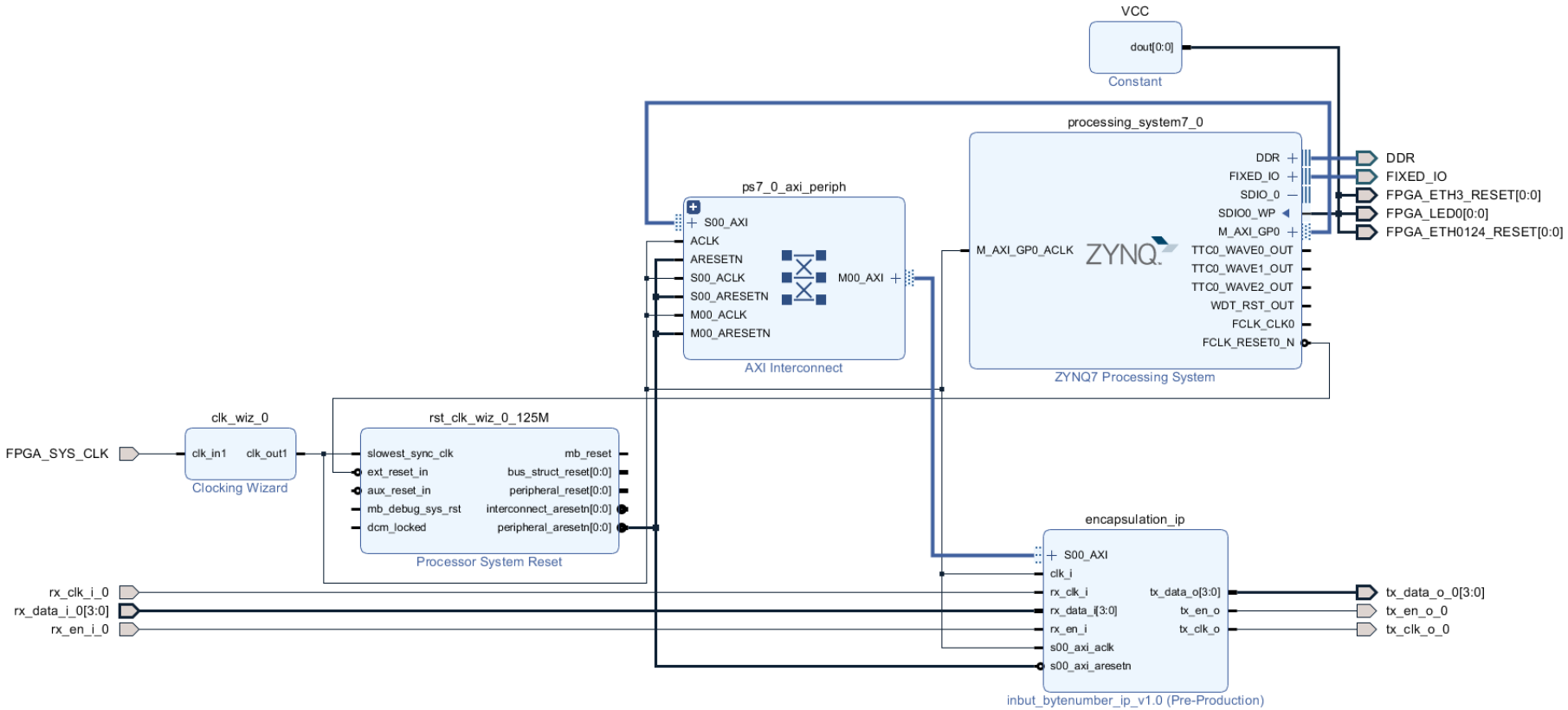

Figura 10. Diseño de Core de encapsulado integrado para SMARTZynq.

Una vez se ha realizado el proceso de implementación en Vivado sin errores, ni resultados en el análisis de tiempos que sean negativos. Por ello, se deduce que la lógica programada en el modelo se ejecuta completamente sin problemas de falta de tiempo. También se han obtenido los resultados de utilización de recursos (figura 11). En ellos se puede verificar, que el uso de recursos es muy reducido, con lo que el IP desarrollado se podría llegar a integrar en un sistema de otra aplicación de control de tráfico de red más completa.

Utilization Post-Synthesis | **Post-Implementation**

Graph | **Table**

Resource	Utilization	Available	Utilization %
LUT	773	53200	1.45
LUTRAM	60	17400	0.34
FF	920	106400	0.86
BRAM	0.50	140	0.36
IO	16	125	12.80
BUFG	2	32	6.25
MMCM	1	4	25.00

Figura 11. Resultados de utilización de recursos de la FPGA.

3.4.3 Implementación en SoC

La implementación en el SoC es el último paso a completar en la implementación de este diseño. Para ello, se utilizará Petalinux, el cual es un kit de herramientas para desarrollo de software Linux en diseños SoC basados en FPGA.

Flujo de trabajo en Petalinux

El flujo de trabajo a seguir con esta herramienta será el siguiente, partiendo por el diseño hardware del que se dispone [31]:

- **Exportar hardware a Petalinux.** Después de haber completado el diseño hardware y haber obtenido el Bitstream, el proyecto de Petalinux requiere de un archivo de descripción hardware (archivo .xsa) con la información del procesador. Este archivo se obtiene ejecutando exportar hardware en Vivado Design Suite. Durante los próximos pasos a ejecutar, Petalinux creara los archivos de arranque, drivers etc. en base a este archivo de descripción hardware.

- **Crear un proyecto nuevo.** Una vez abierta la consola de comando en la ruta deseada, para crear un proyecto nuevo de Petalinux hay que ejecutar el siguiente comando:

```
$ petalinux-create --type Project --template ‹PLATFORM› --name ‹PROJECT_NAME›
```

Donde ‹PLATFORM› se sustituye por zynq para dispositivos de Zynq-7000, y en vez de ‹PROJECT_NAME› se introduce el nombre del proyecto que se desee.

- **Importar la configuración hardware.** Mediante el siguiente comando la descripción hardware del diseño se importa al proyecto de Petalinux:

```
$ petalinux-config --get-hw-description ‹PATH-TO-XSA Directory›
```

Donde ‹PATH-TO-XSA Directory› se sustituye por la ruta del archivo .xsa obtenido al exportar hardware desde Vivado.

Esto abrirá un menú de configuración del sistema en que se pueden configurar los ajustes del hardware de manera automática. Con ello, se actualizarán los archivos de device.tree, de configuración de U-Boot y de configuración de kernel.

- **Construir la imagen del sistema.** Para ello, se ejecuta un comando que genera el archivo DTB de devicetree, un first stage boot loader, el U-Boot, el kernel de Linux y una imagen del sistema de la raíz de archivos. Finalmente, genera las imágenes de inicio necesarias.

```
$ petalinux-build
```

Las imágenes generadas se almacenarán en el directorio /images/Linux o /tftpboot dentro del proyecto.

- **Generar la imagen de Boot para Zynq-7000.** La imagen de Boot permitirá que el dispositivo se arranque al encenderlo. Para generar la imagen se ejecuta el siguiente comando:

```
$ petalinux-package --boot --fsbl ‹FSBL image› --fpga ‹FPGA bitstream› --u-boot
```

Donde ‹FSBL image› sera sustituido por el archivo fsbl del proyecto y ‹FPGA bitstream› por el bitstream del diseño.

- **Iniciar una imagen de Petalinux en el hardware con una tarjeta SD.** En este punto, se monta una tarjeta SD en el PC y se copian los siguientes archivos en la primera partición, que tiene formato FAT32: BOOT.BIN, image.ub y boot.scr.

A continuación, se conecta mediante el puerto de serie el dispositivo al PC y se configura para que inicie desde la tarjeta SD. Después, se introduce la tarjeta SD y se enciende el dispositivo.

En proceso de inicio y ejecución se podrá visualizar mediante cualquier programa de comunicación serie.

4 Simulación y resultados

4.1 Simulación de IP Core de encapsulado

La simulación del IP Core se ha realizado en un testbench. En este, se genera una señal de reloj de 125 MHz de frecuencia, que utilizará para el IP y para el tráfico de TSN que se le introduzca. Además, también se genera la señal de enable de la transmisión, y como se puede ver en VHDL del testbench (figura 12) y en los resultados de la simulación (figuras 13 y 14), también se genera la señal de datos de entrada al IP (rx_data_i[3:0]), por la que se envían datos cada flanco de reloj, cuando la señal de rx_en está habilitada.

```
process (clk)                --datai
    type    BinFile    is file of character;
    file    fich_ent : BinFile open read_mode is "fichero_entrada.txt";
    variable datapack   : std_logic_vector(3 downto 0);
    variable leyendo :boolean;
    variable caract :character;
begin
    if (clk'event) then
        if rx_en_i = '1' then
            datapack := (others=>'0');
                read (fich_ent, Caract);
                case Caract is
                when '0' => datapack := "0000";
                when '1' => datapack := "0001";
                when '2' => datapack := "0010";
                when '3' => datapack := "0011";
                when '4' => datapack := "0100";
                when '5' => datapack := "0101";
                when '6' => datapack := "0110";
                when '7' => datapack := "0111";
                when '8' => datapack := "1000";
                when '9' => datapack := "1001";
                when 'A' => datapack := "1010";
                when 'B' => datapack := "1011";
                when 'C' => datapack := "1100";
                when 'D' => datapack := "1101";
                when 'E' => datapack := "1110";
                when 'F' => datapack := "1111";
                when others => datapack := "0000";
                end case;
            rx_data_i <= datapack(3 downto 0);
        elsif rx_en_i = '0' then
            datapack := "0000";
            rx_data_i <= datapack(3 downto 0);
        end if;
    end if;
end process;
```

Figura 12. VHDL de Testbench para lectura de .txt.

Estos datos, se leen de un archivo .txt añadido al diseño en el que se han introducido datos en formato hexadecimal. De esta manera, se obtienen las señales de input signals de las figuras 13 y 14, en las que se simula tráfico TSN de entrada. Después, estos datos se envían encapsulados como se explica a continuación.

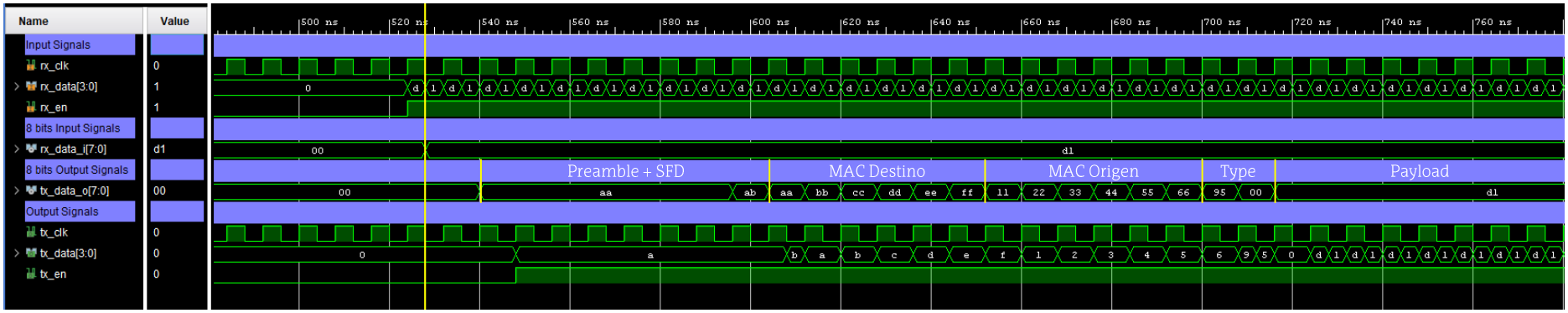

Figura 13. Captura 1 de simulación de encapsulado de tramas.

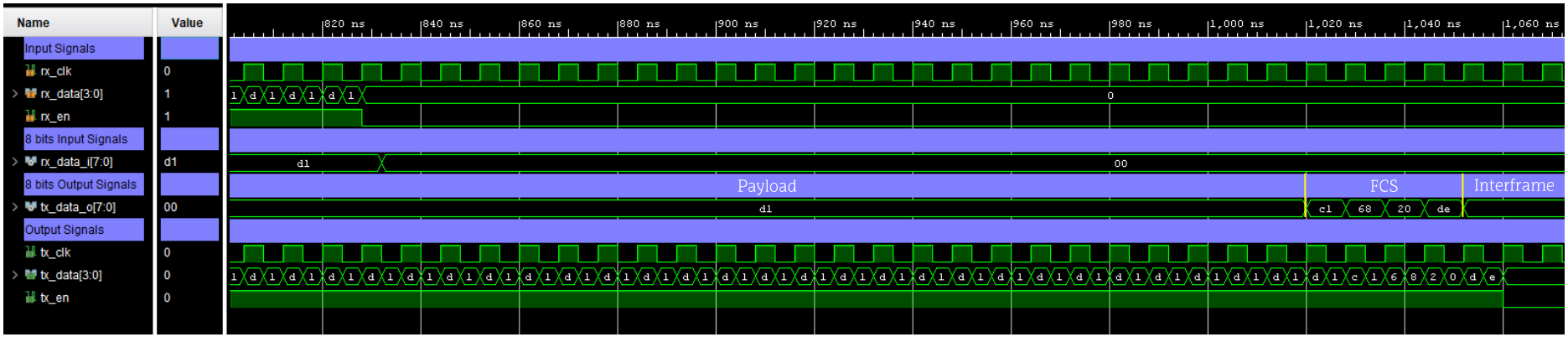

Figura 14. Captura 2 de simulación de encapsulado de tramas.

El paquete está representado por una serie de datos en la que se repite "D1". Se puede observar cómo estos datos que entran de 4 en 4 bits, se agrupan de 8 en 8. Además, si se comparan los datos de entrada de 8 bits (rx_data_i[7:0]) con los de salida (tx_data_o[7:0]), se puede observar cómo en el de salida se van introduciendo los campos de la trama de ethernet estándar. En primer lugar, se introduce el preámbulo y el byte de SFD ("AAAAAAAAAAAAAAAB"), seguidamente, se introducen las MAC de destino ("AABBCCDDEEFF") y la de origen ("112233445566") y tras estas el ethertype ("9500").

Después, se comienza a introducir el paquete según se ha recibido ("D1D1...D1") y en cuanto se termina, se transmite el CRC calculado para el paquete de ethernet estándar ("C16820DE")

Por último, también se puede observar, cómo únicamente con 3 ciclos de retraso respecto al paquete recibido, se va transmitiendo el paquete de ethernet estándar con el tráfico TSN encapsulado y sincronizado con las demás señales de RGMII.

4.2 Simulación configuración remota

Para obtener unos resultados de funcionamiento que se asemejen a la realidad, se ha llevado a cabo una simulación del IP Core de encapsulado utilizando el bloque AXI Traffic Generator. Este bloque en el catálogo de IPs de Vivado permite generar accesos vía AXI. De esta manera, se consigue simular los accesos para la configuración de dirección MAC que se llegarían a realizar desde PS, si la implementación en hardware se hubiera completado.

AXI Traffic generator

Axi Traffic Generator es un IP del catálogo que Xilinx ofrece, el cual es completamente sintetizable y compatible con AXI4. Este IP ofrece las siguientes características [32]:

- Una opción configurable para generar o aceptar data de acuerdo con distintos tipos de tráfico.
- Tamaño de direcciones configurable para el interfaz de maestro de AXI4.
- Soporta transmisiones dependientes/independientes entre lecturas/escrituras de puertos maestros con retrasos configurables.
- Cuenta de repetición programable para transacciones constantes / incrementales / aleatorias.
- Un habilitador/deshabilitador externo para generar tráfico sin intervención del procesador.

- Opción de generar tráfico específico para un IP en interface de AXI para protocolos predefinidos.

Por otro lado, este IP tiene dos tipos principales de perfiles de funcionamiento. Por un lado, el perfil Custom, que permite seleccionar diferentes tipos de generación de tráfico de interface AXI4 y por el otro el High Level Traffic, que permite generar tráfico con protocolos predefinidos.

En este caso el perfil de funcionamiento que se ha utilizado ha sido Custom. Y de entre los modos de funcionamiento que este ofrece se seleccionado el modo System Init/Test Mode.

System Init/Test Mode

System Init mode es un tipo de funcionamiento que cumple con las necesidades de este proyecto, ya que, ofrece la posibilidad de generar señales sin necesidad de un procesador, haciendo que sea ideal a la hora de realizar simulaciones.

El Core funciona de manera que cuando sale del estado de reset lee los archivos COE (coefficient) de datos y direcciones, y genera transmisiones de interfaz AXI4-Lite. Además, permite configurar distintos parámetros visibles en la figura 15.

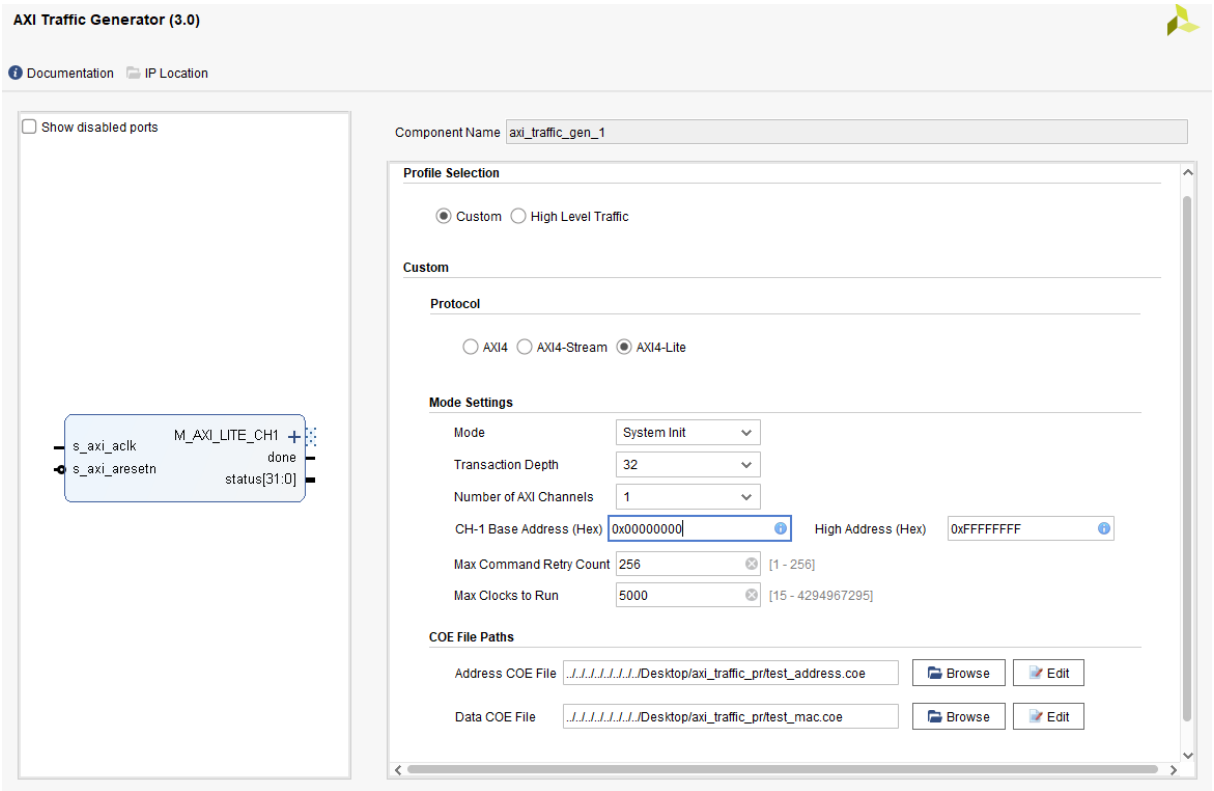

Figura 15. Ventana de configuración de IP AXI Traffic Generator.

Para realizar la simulación del proyecto, los datos que se han introducido en el archivo COE de datos son los necesarios para realizar un cambio de dirección MAC destino y origen. El formato de los mismos se ha explicado anteriormente en el punto 6.2. En cuanto a los datos del archivo COE de direcciones, se ha introducido una dirección por cada dato MAC en el archivo de datos. Para terminar, se ha añadido la dirección 0xFFFFFFFF al final, la cual define un NOP (No Operation) y detiene el Core de AXI Traffic Generator en cuanto se lee esta dirección.

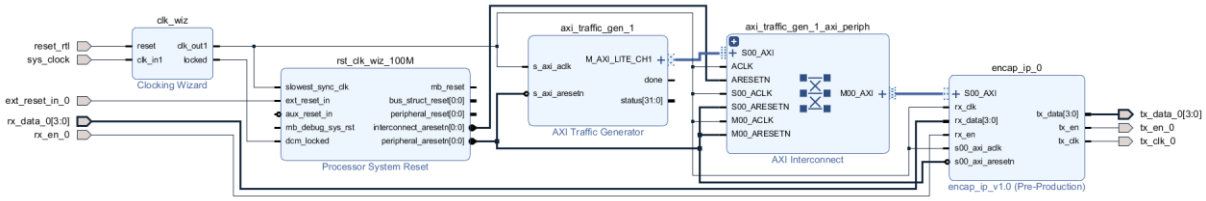

Figura 16. Modelo de encapsulado con AXI Traffic Generator.

En bloque de diagramas obtenido para la simulación es el de la figura 16. Como se puede observar, el IP de AXI Traffic Generator se conecta mediante AXI interconect al IP Core de encapsulado, como lo haría el procesador. Además, para la simulación se ha generado una señal de reset que se deshabilita pasado un tiempo, haciendo que en ese momento comience a funcionar el Core de AXI Traffic Generator. De esta manera, se pretende emular lo que podría ser un acceso desde PS para cambiar las direcciones MAC de los paquetes.

En la figura 17 se puede observar cómo mientras hay un tráfico constante de paquetes que entran y se devuelven encapsulados, se realizan 4 accesos AXI. En la figura 18, se puede observar cómo en estos accesos se transmiten datos con 32 bits, de los cuales 24 bits son las direcciones MAC. Según llegan los datos se cargan en los registros temporales de las direcciones MAC (MAC_dst_reg, MAC_src_reg). Sin embargo, no se cargan en los registros del paquete, con los cuales se crean los paquetes encapsulados, hasta que la máquina de estados del IP Core de encapsulado entra en estado de interframe.

Como resultado de la configuración de las direcciones MAC, se obtiene que la dirección de destino que en un inicio → "AABBCCDDEEFF" se ha cambiado por la dirección → "555555666666". Para lo ello, se han modificado primero los 24 bits más significativos "555555" y a continuación los 24 bits menos significativos en el siguiente acceso "666666". En cuanto a la dirección MAC origen, se han llevado a cabo los mismos pasos cambiando en este caso "112233445566" por "444444333333". Con ello, se comprueba que la configuración de las direcciones MAC se realiza correctamente.

Figura 17. Simulación de accesos desde PS con AXI Traffic Generator.

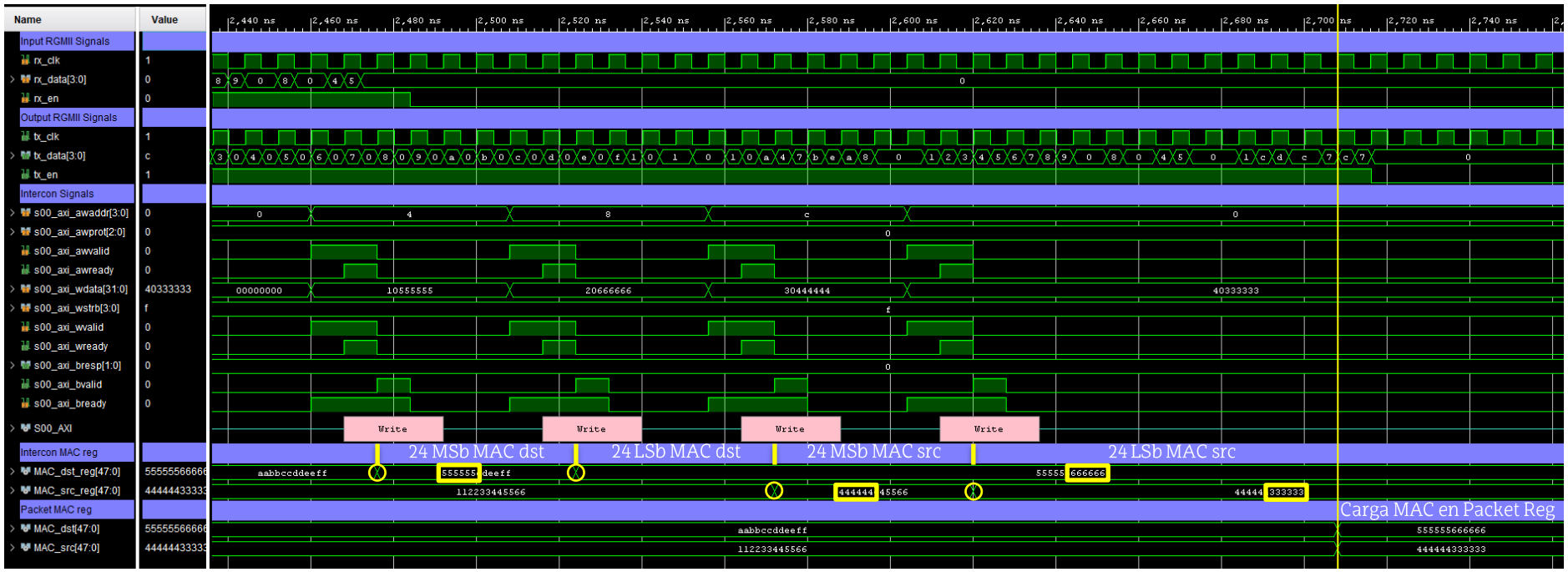

Figura 18. Ampliación de captura de simulación de accesos desde PS con AXI Traffic Generator.

4.3 Simulación de desencapsulado de tramas en PC

Debido a la imposibilidad de llevar a cabo la implementación del desarrollo para la FPGA, se ha realizado una simulación, para verificar que el script de Python funciona correctamente. Para ello, se ha utilizado una herramienta en Linux llamada PackETH [33].

PackETH es una herramienta con interfaz gráfica que permite generar tráfico ethernet. Con ella, se puede generar y enviar cualquier tipo de paquete o secuencia de paquetes. Con las posibilidades que esta herramienta ofrece se han generado paquetes de tipo ethernet estándar con los parámetros visibles en la figura 19. Se puede observar cómo los paquetes generados tendrán la dirección MAC destino "AA:BB:CC:DD:EE:FF" y origen "11:22:33:44:55:66". Además, se a añadido un Ethertype inventado "9500" y en el payload se ha introducido un paquete con IEEE 802.1 Qbv, con la estructura descrita en el punto 2.1.3 del documento.

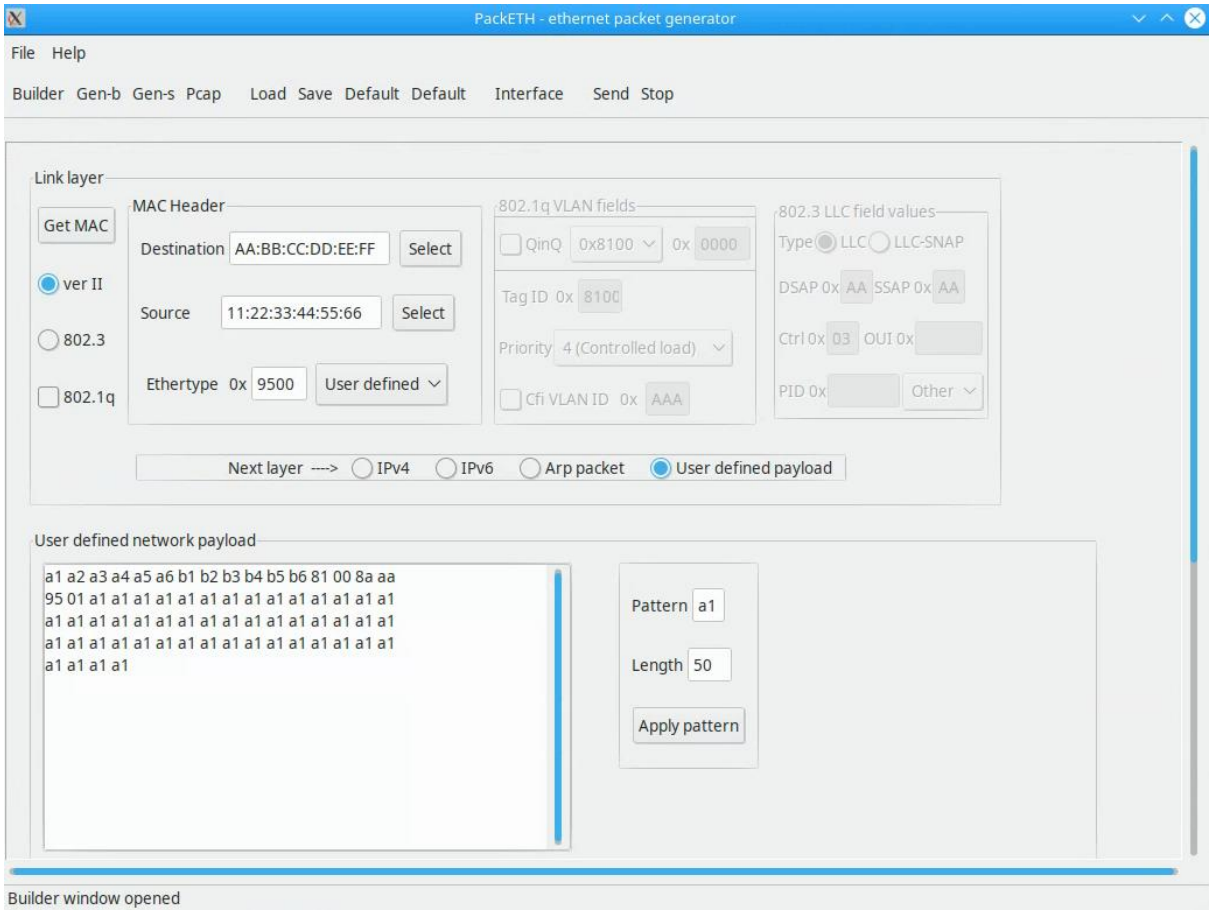

Figura 19. Ventana de configuración de paquetes en PackETH.

A continuación, se ha configurado una secuencia de generación de paquetes infinita que envía un paquete cada 10 ms:

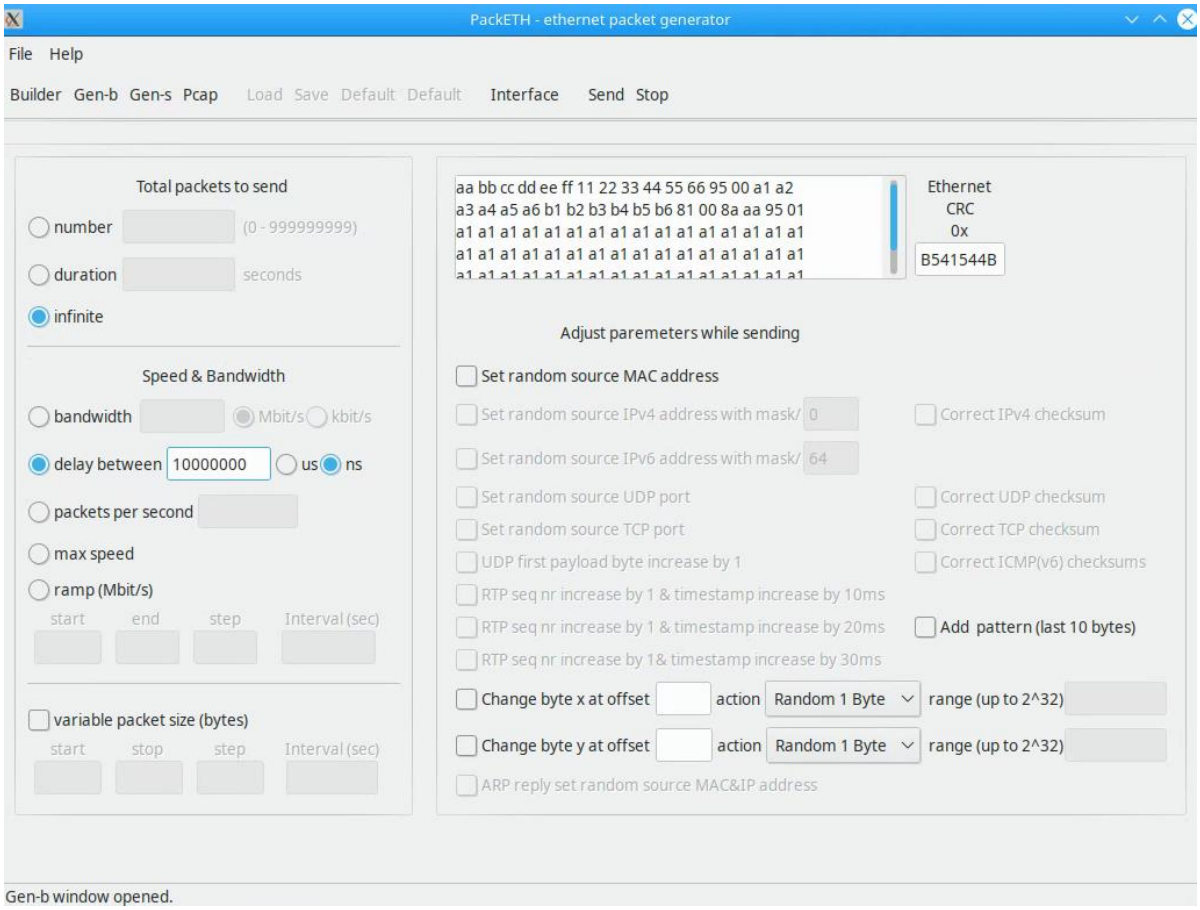

Figura 20. Ventana de configuración de secuencia de envío de paquetes en PackETH.

Una vez el tráfico está siendo generado se ha ejecutado el script de desencapsulado, filtrando los paquetes de entrada por su MAC origen → "11:22:33:44:55:66":

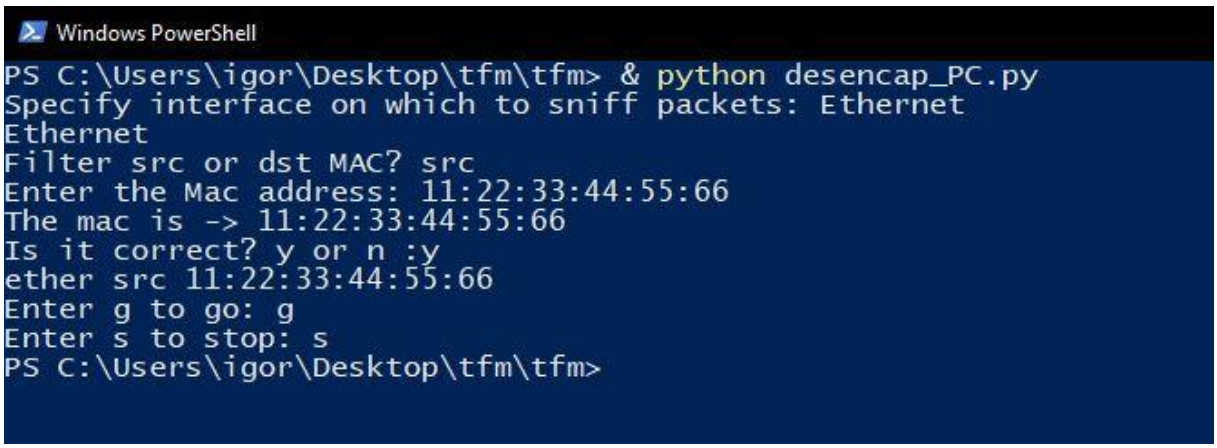

Figura 21. Captura de ejecución de script de desencapsulado.

En este caso, para comprobar que el desencapsulado se realiza correctamente, se han generado dos archivos .pcap, uno del tráfico ethernet estándar según se recibe y otra del tráfico desencapsulado.

Para la visualización de los archivos .pcap, se ha utilizado el programa Wireshark [34]. Este, es uno de los principales y más utilizados programas de análisis de protocolos de red. A parte de permitir el análisis de tráfico de las interfaces de red de un PC, ofrece la posibilidad de visualizar y analizar el tráfico guardado en archivos .pcap.

De esta manera, las tramas obtenidas se han visualizado en wireshark:

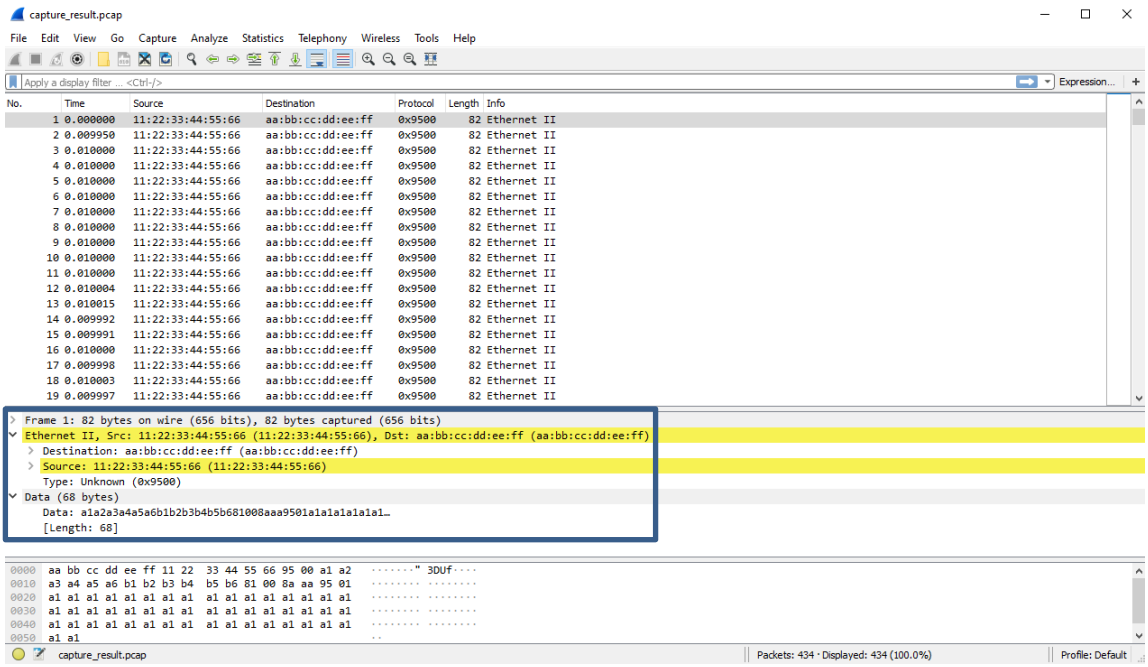

Figura 22. Captura de paquetes ethernet estándar.

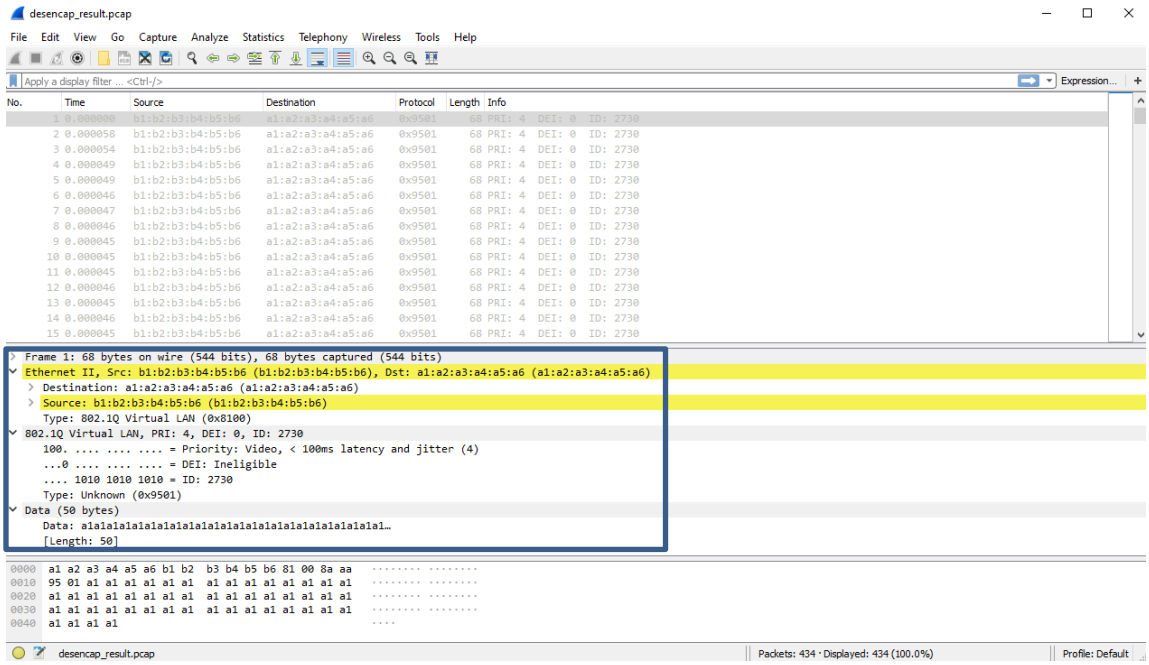

Figura 23. Captura de paquetes desencapsulados.

Los paquetes generados por el programa packETH se pueden visualizar en la figura 22 y en la figura 23 los paquetes ya desencapsulados. Se confirma que el script desarrollado desencapsula el tráfico de Ethernet, de manera que se consigue visualizar el tráfico que se transportaba en la capa de datos de los paquetes de Ethernet estándar. Además, en este caso concreto se puede observar TSN desencapsulado (IEEE 802.1Qbv) y cómo wireshark es capaz de interpretarlo. Por ello, se puede concluir que mediante este script se puede visualizar tráfico TSN, que ha sido encapsulado en tráfico de Ethernet estándar.

5 Metodología seguida en el desarrollo del trabajo

5.1 Descripción de tareas, fases, equipos o procedimientos

En este apartado se describen las tareas que se llevarán a cabo para completar el objetivo principal y los objetivos secundarios fijados para este proyecto de fin de master. Además, se han enumerado los hitos y entregables a obtener. Para el cumplimiento de los 12 créditos del TFM se trabajará una media de 15 horas semanales durante 20 semanas.

Tarea 1. Definición de especificaciones y requisitos del proyecto (semana 1).

Descripción: En esta tarea se definirán las especificaciones del IP Core de encapsulado, como el tipo de encapsulado, interfaz de tráfico en la entrada y salida del mismo. Por otro lado, también se definirán las herramientas con las que se llevará a cabo el Script de desencapsulado y su simulación.

Hitos: Determinar los requisitos funcionales del proyecto.

Entregables: Documento de especificaciones (a generar por parte del director del proyecto)

Tarea 2. Estado del Arte (semana 2, 3 y 14 a 15)

Descripción: Esta tarea se divide en tres subtareas: 2.1 TSN, 2.2 FPGA y 2.3 Python y Scapy. En la primera, se debe estudiar el estado del arte de los estándares TSN, además de comprobar cuál es el dispositivo adecuado para la implementación de los mismos. En la segunda, se analizará el SoC, Zynq-7000, ya que será la plataforma de implementación. Por último, se analizará tanto el lenguaje Python, como la herramienta Scapy, para su posterior uso en la creación del script de desencapsulado. Para ello, se dispondrá de material bibliográfico disponible mediante las suscripciones de la universidad a bases de datos de bibliografía científica (IEEEXplore y Scopus).

Hitos: Comprender el tipo de tráfico a tratar y las posibilidades que la plataforma de implementación y las herramientas a utilizar ofrecen.

Entregables: Apartado del estado del arte incluido en el trabajo de fin de master.

Tarea 3. Formación (semana 4 a 5, 10, 16 a 17 y 18)

Descripción: Esta tarea se divide en cuatro sub tareas: 3.1 Vivado, 3.2 Petalinux, 3.3 Python y Scapy y 3.4 PackETH + wireshark. Para llevar a cabo el proyecto, es imprescindible familiarizarse con las herramientas de desarrollo, simulación e implementación que se utilizarán. Como el proyecto se divide en dos ramas principales, se analizarán por un lado las herramientas que el fabricante Xilinx ofrece para el diseño, simulación e implementación y por el otro, las herramientas que permitan desarrollar y verificar el script de desencapsulado creado en Python.

Hitos: Conseguir manejo avanzado para las herramientas de Xilinx, Python y Scapy.

Entregables: No aplica para esta tarea.

Tarea 4. Diseño (semana 6 a 8, 8 y 17 a 18)

Descripción: Esta tarea se divide en tres subtareas: 4.1 IP Core encapsulado, 4.2 Testbench de simulación y 4.3 Script en Python. En primer lugar, se diseñará el IP Core de encapsulado siguiendo las especificaciones fijadas desde el inicio. A continuación, se creará la plataforma de simulación para la verificación del funcionamiento correcto del Core. Por último, se diseñará el modelo de desencapsulado en lenguaje Python con el uso de las librerías de Scapy.

Hitos: Completar los diseños principales del proyecto.

Entregables: Diseños realizados y los pasos llevados a cabo para su obtención. (explicados detalladamente en el documento).

Tarea 5. Implementación (semana 10 a 11, 12 y 13)

Descripción: Esta tarea está formada por tres subtareas: 5.1 Conexión a internet + Xillinux, 5.2 Modelo SMARTZynq y 5.3 Petalinux. Entre ellas, se llevarán a cabo los pasos necesarios en un proyecto de un IP Core desarrollado, para su implementación en un dispositivo con acceso a internet. Pasando por conseguir conectar una Zedboard a internet, implementar el IP Core en un diseño completo y llevar a cabo los pasos para integrar dicho proyecto en un SoC con sistema operativo.

Hitos: Conseguir los resultados de implementación.

Entregables: Apartado de implementación que se incluye en el documento.

Tarea 6. Simulación (semana 9, 10 y 19)

Descripción: Esta tarea consta de tres subtareas: 6.1 IP Core encapsulado, 6.2 AXI Traffic Generator y 6.3 PackETH + wireshark. En esta tarea se llevarán a cabo las simulaciones necesarias para verificar el correcto funcionamiento del IP Core y del Script de desencapsulado.

Hitos: Comprobar funcionamiento de modelos diseñados.

Entregables: Resultados de simulación contenidos en el apartado de simulación del documento.

Tarea 7. Análisis de resultados y conclusiones (semana 11 a 13 y 19)

Descripción: En esta tarea se contemplan dos subtareas: 7.1 IP Core y 7.2 Script en Python. Siguiendo con el objetivo principal del proyecto se analizarán los resultados obtenidos

para el encapsulado del tráfico TSN mediante el Core y para el desencapsulado realizado en PC.

Hitos: Comprobar el cumplimiento de las especificaciones establecidas.

Entregables: Informe de las conclusiones presentes en el documento.

Tarea 8. Documentación (semana 2 a 20)

Descripción: En esta tarea se llevará a cabo la documentación del trabajo de fin de master. Se completará a medida que se vayan realizando las tareas mencionadas anteriormente.

Hitos: Completar la documentación del trabajo de fin de master.

Entregables: Documento del trabajo de fin de master.

5.2 Diagrama de Gantt

En el diagrama de Gantt (figura 24) se pueden visualizar las tareas anteriormente definidas. A lo largo del proyecto estas tareas se han cumplido debidamente dando como resultado el documento presentado. Por ello, se considera que la planificación del proyecto ha sido satisfactoria.

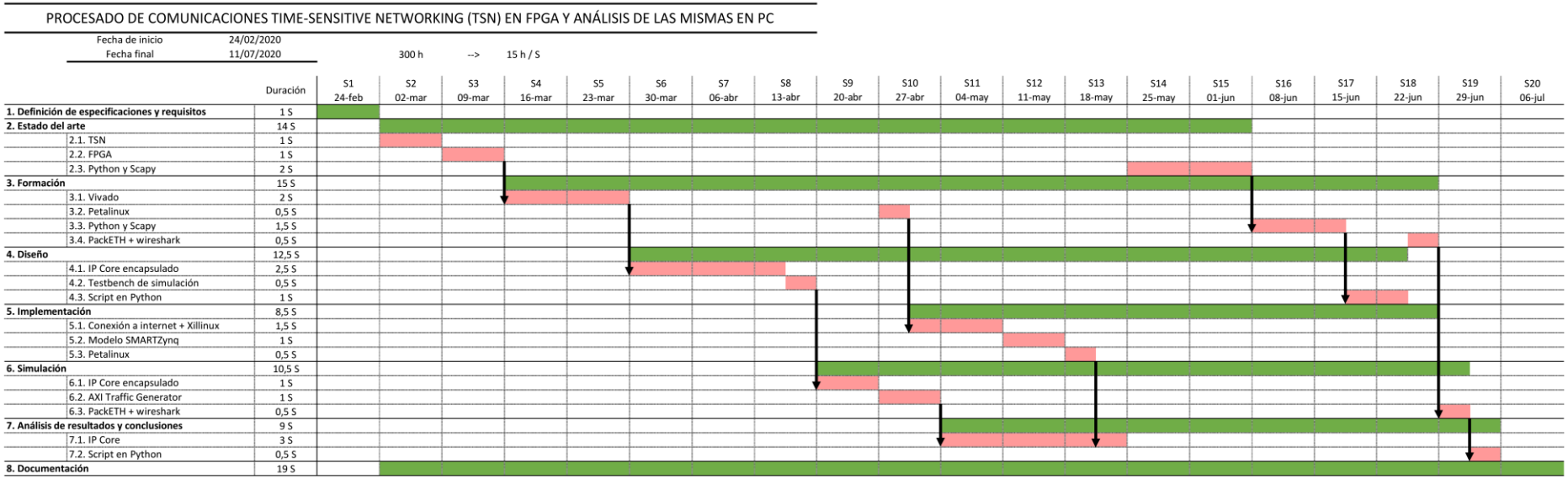

Figura 24. Diagrama de Gantt del proyecto.

6 Conclusiones

Tras analizar la inminente llegada de los estándares de TSN a las comunicaciones de la industria, como paso de adaptación a la industria 4.0, y teniendo en cuenta la perdida de datos que los equipos no especializados sufren ante estos. En este trabajo se propone una solución a implementar en un SoC para poder visualizar tráfico TSN en equipos no especializados como PCs. Además, como objetivo final se plantea llevar a cabo la implementación en un producto real. Para llevar a cabo la solución, se han analizado dos temas principales, un Core de encapsulado a implementar en el SoC, que sea configurable mediante internet y un script de desencapsulado para ejecutar desde un PC. Esto, se ha analizado debidamente el estado del arte disponible en la literatura científica.

En lo que respecta al IP Core de encapsulado, el diseño del mismo se a completado una vez adquiridos los conocimientos de TSN y del hardware de implementación necesarios. Seguidamente, se ha preparado un set up para poder conectar una tarjeta de evaluación Zedboard a internet, de manera que los diseños a implementar en el futuro se puedan conectar de manera sencilla. A continuación, se ha llevado a cabo la implementación del IP Core en un modelo compatible con SMARTZynq. Un módulo de la empresa SoC-e, que permite la fácil integración de redes de Ethernet industrial, mediante un SoC y 5 PHYs de Gigabit entre otros. Esto, se ha completado sin obtener errores en el proceso y con resultados en el análisis de tiempos adecuados. Además, se han analizado los recursos de hardware utilizados por el diseño y se ha comprobado con ellos que se utiliza una cantidad reducida de los mismos. Lo cual hace que este Core sea fácilmente implementable en otros sistemas. Para finalizar con la implementación, se ha analizado el proceso a seguir para poder integrar el bitstream del diseño con un sistema operativo en un SoC. Por último, la implementación en placa se pretendía realizar en el dispositivo RELY-RB de la empresa RELYUM, que es un switch con tecnología anti perdida de paquetes para ethernet redundante, con sincronización inferior al microsegundo y ciberseguridad. Con ello, se pretendía aplicar a un producto real la solución desarrollada. Sin embargo, debido a las condiciones provocadas por el COVID, no se ha podido completar.

Por esa misma razón, se ha decidido llevar a cabo una simulación más completa del diseño, no solo realizando una simulación de tráfico, si no estudiando el IP AXI Traffic Generator y utilizándolo en la simulación para emular el comportamiento de los accesos que se realizarían desde PS al IP. De esta manera, consiguiendo simular los accesos realizados para la configuración remota de ciertas características del Core. Con los resultados de la simulación de encapsulado de tráfico, a pesar de su correcto funcionamiento, se han podido apreciar ciertas limitaciones del diseño. Ya que, el paquete de tráfico TSN no puede ser mayor que el tamaño máximo de payload fijado para el estándar de Ethernet. Como solución a esto, se podría llegar a dividir y enviar paquetes encapsulados por partes, que se unan tras desencapsularlos en PC, para su visualización.

Además de esto, también existen limitaciones de latencia, debido a que los paquetes que se transmiten son más grandes que los que se reciben. Sin embargo, esto se podría mejorar realizando un encapsulado selectivo teniendo en cuenta alguna característica de los paquetes recibidos. Por otro lado, los resultados obtenidos con ello, confirman que la reconfiguración de las MACs del encapsulado se realiza de manera adecuada.

En cuanto al Script diseñado, en este caso, se han analizado las herramientas Scapy y Python, para aplicar las mismas a los objetivos del proyecto. Una vez adquirido el conocimiento necesario se ha completado el script de desencapsulado. Su funcionamiento, se ha podido comprobar al completo con el uso de las herramientas PackETH y Wireshark. Además, se ha probado como se pueden llegar a visualizar tramas de TSN en un PC.

Se puede decir que, aunque con modificaciones, los objetivos principales se han completado de manera satisfactoria. Y lo que respecta al trabajo futuro, quedaría realizar la implementación completa del sistema en la plataforma y comprobar su funcionamiento. Por otro lado, también cabe la posibilidad de continuar con el diseño del IP, haciendo que se pueda encapsular solo el tráfico deseado y/o que el encapsulado sea diferente dependiendo del tráfico. Obteniendo una herramienta completa, se podrían llegar a realizar análisis de sistemas de TSN desde el PC.

7 Referencias

[1] Mariani, M. y Borghi, M. (2019). Industry 4.0: A bibliometric review of its managerial intelectual structure and potential evolution in the service industries. *Technological Forecasting & Social Chane*, vol.149.

[2] Recommendations for implementing the strategic initiative INDUSTRIE 4.0. *Final report of the Industrie 4.0 Working Group. (2013)*

[3] Atzori, L., Iera, A. y Morabito, G. (2010). The Internet of Things: A survey. *Computer Networks*, vol 54, (2787-2805).

[4] Baheti, R., y Gill, H. (2011). Cyber-physical systems. *The impact of control technology*, vol 12, (161-166).

[5] Sisinni, E., Saifullah, A., Han, S., Jennehag, U., y Gidlund, M. (2018). Industrial internet of things: Challenges, opportunities, and directions. *IEEE Transactions on Industrial Informatics*, vol 14, (4724-4734).

[6] IEEE 802.1 Working Group. Último acceso: 06/04/2020. Url: https://1.ieee802.org/

[7] Simon, C., Maliosz, M., & Mate, M. (2018). Design aspects of low-latency services with time-sensitive networking. *IEEE Communications Standards Magazine*, vol 2, (48-54).

[8] Silva, L., Pedreiras, P., Fonseca, P., y Almeida, L. (2019). On the adequacy of SDN and TSN for Industry 4.0. *In 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing* (ISORC) (pp. 43-51).

[9] Brooks, S. y Uludag, E. Time-Sensitive Networking: From Theory to Implementation in Industrial Automation. (White paper)

[10] SMARTzynq module: 5 Port Gigabit Industrial Ethernet Embedded Switch Module. Último acceso: 04/05/2020. Url: https://soc-e.com/products/smart-zynq-module/

[11] RELY-RB, Time-aware Redbox Switch. Último acceso: 04/05/2020. Url: https://www.relyum.com/web/wp-content/uploads/2019/10/RELY-RB-Brochurev3.pdf

[12] Bello, L. L., y Steiner, W. (2019). A perspective on IEEE time-sensitive networking for industrial communication and automation systems. *Proceedings of the IEEE*, vol 107, (1094-1120).

[13] Finn, N. (2018). Introduction to time-sensitive networking. *IEEE Communications Standards Magazine*, vol 2, (22-28).

[14] Messenger, J. L. (2018). Time-sensitive networking: An introduction. *IEEE Communications Standards Magazine*, vol 2, (29-33).

[15] Murselovic, L. (2020). Performance Analysis of the pre-emption mechanism in TSN. (Trabajo de fin de master). School of Innovation Design and Engineering, Västeras, Suecia. Último acceso: 20/08/2020. Url: https://www.diva-portal.org/smash/get/diva2:1437909/FULLTEXT01.pdf

[16] Shreejith, S., Mundhenk, P., Ettner, A., Fahmy, S. A., Steinhorst, S., Lukasiewycz, M., y Chakraborty, S. (2017). VEGa: A high performance vehicular Ethernet gateway on hybrid FPGA. *IEEE Transactions on Computers*, vol 66, (1790-1803).

[17] Complex Programmable Logic Device (CPLD), Xilinx. Último acceso: 11/05/2020. Url: https://www.xilinx.com/products/silicon-devices/cpld/cpld.html

[18] Field Programmable Gate Array (FPGA), Xilinx. Último acceso: 11/05/2020. Url: https://www.xilinx.com/products/silicon-devices/fpga/what-is-an-fpga.html

[19] Alfke, P. (2007). 20 years of FPGA evolution from glue logic to major system component. *In 2007 IEEE Hot Chips 19 Symposium* (HCS), (pp. 1-19).

[20] Zynq-7000 SoC Data Sheet: Overview. DS190. V1.11.1. Xilinx (2018).

[21] Oliphant, T. E. (2007). Python for scientific computing. *Computing in Science & Engineering*, vol 9, (10-20).

[22] Python documentation. Último acceso: 25/05/2020. Url: https://docs.python.org/

[23] NumPy. Último acceso: 25/05/2020. Url: https://numpy.org/

[24] SciPy. Último acceso: 25/05/2020. Url: https://www.scipy.org/

[25] Scapy Project. Último acceso: 25/05/2020. Url: https://scapy.net/

[26] Scapy documentation. Último acceso: 25/05/2020. Url: https://scapy.readthedocs.io/en/latest/

[27] Vivado Design Suite User Guide: Design Flows Overview. UG892. V2018.2. Xilinx (2018).

[28] Low, M. (2006). Using the RGMII to Interface with the Gigabit Ethernet MAC. XAPP692. V1.0.1. Xilinx

[29] Stavinov, E. (2010). A practical parallel CRC generation method. *Circuit Cellar-The Magazine For Computer Applications*, vol 31, (38-45).

[30] Xillinux: A Linux distribution for Z-Turn Lite, Zedboard, Zybo and MicroZed. Último acceso: 11/05/2020. Url:

[31] PetaLinux Tools Documentation: Reference Guide. UG1144. V2020.1. Xilinx (2020).

[32] AXI Traffic Generator v3.0: LogiCORE IP Product Guide. Vivado Design Suite. PG125. Xilinx (2019).

[33] PackETH. Último acceso: 25/05/2020. Url: http://packeth.sourceforge.net/packeth/Home.html

[34] Wireshark. Último acceso: 25/05/2020. Url: https://www.wireshark.org/

A ANEXO I: Diseño VHDL y Código en Python

Los diseños creados y utilizados en este proyecto son los siguientes:

- VHDL de encapsulado con interfaz AXI → encap_ip_S00_AXI.vhd
- VHDL del componente pila FIFO → stack_fifo.vhd
- VHDL de cálculo CRC-32 obtenido de outputlogic.com y modificado para este proyecto → crc_32_8bits.vhd
- Script en Python de la aplicación de desencapsulado → desencap_PC.py

Acceso a los archivos mediante el siguiente enlace → https://drive.google.com/drive/folders/11_hCYyHo6CpQ0s1qWGRCpDKgPMOmCNdM?usp=sharing