REDES DEPENDIENTES DE LA TEMPORIZACIÓN BASADAS EN MÉTODOS DE SINCRONIZACIÓN ULTRA-PRECISOS

TIME-SENSITIVE NETWORKS BASED ON ULTRA-ACCURATE SYNCHRONIZATION MECHANISMS

JORGE SÁNCHEZ GARRIDO

UNIVERSIDAD
DE GRANADA

REDES DEPENDIENTES DE LA TEMPORIZACIÓN BASADAS EN MÉTODOS DE SINCRONIZACIÓN ULTRA-PRECISOS

TIME-SENSITIVE NETWORKS BASED ON ULTRA-ACCURATE SYNCHRONIZATION MECHANISMS

JORGE SÁNCHEZ GARRIDO

Directores
ANTONIO JAVIER DÍAZ ALONSO
EDUARDO ROS VIDAL

Programa de Doctorado
TECNOLOGÍAS DE LA INFORMACIÓN Y LA COMUNICACIÓN

Departamento de Arquitectura y Tecnología de los computadores
Escuela Técnica Superior de Ingenierías en Informática y Telecomunicaciones
ATC TIC-117

Noviembre 2020

Editor: Universidad de Granada. Tesis Doctorales
Autor: Jorge Sánchez Garrido
ISBN: 978-84-1306-766-7
URI: http://hdl.handle.net/10481/66660

A mi madre, por todo ...

Y a mi familia: a mi padre y a mi hermano, que se encuentra en estos mismos quehaceres, a mi querido abuelo, y a mis tíos que siempre estuvieron al lado.

RESUMEN

Las comunicaciones deterministas son un requisito esencial en múltiples aplicaciones, tales como las plantas industriales, las redes de automoción, o los sistemas de aeroespacial. Los buses de campo han venido empleándose de forma tradicional en estos ámbitos para suplir sus necesidades de comunicación e intercambio de datos. A modo de ejemplo, algunos protocolos muy conocidos para estos ámbitos son los buses *CAN* o *FlexRay* en automoción, o *Spacewire* para los vehículos espaciales. Algunas de estas soluciones pueden ser propietarias o necesitar del uso de equipamiento especializado. Por lo tanto, el panorama existente en las tecnologías de buses de campo es bastante diverso, con una gama de soluciones que puede incluir interfaces que van desde lo puramente analógico (HART) hasta aquéllas que son completamente digitales. Estas últimas suelen emplear interfaces de tipo serie y, en particular, tecnologías Ethernet. De hecho, existe una fuerte tendencia a implementar buses de campo usando bloques funcionales de Ethernet estándar (como en el caso de Profinet). Estas soluciones compensan la falta de capacidades de comunicación determinista de las interfaces Ethernet convencionales que, sin embargo, son altamente eficientes para transmitir datos con un ancho de banda considerable, aunque únicamente pueden realizar su entrega con un servicio de tipo *best-effort* (de mejor esfuerzo). No obstante, el giro hacia las interfaces basadas en Ethernet para implementar buses de campo ha demostrado que esta filosofía tiene ventajas inherentes, como mayor rendimiento y una compatibilidad entre dispositivos más amplia. Esto, a su vez, fue el germen de un cambio global que buscaba definir interfaces Ethernet genéricas basadas en estándares que fueran deterministas. En consecuencia, las principales repercusiones de estos cambios se han visto, en primer lugar, con la aparición de la conmutación de audio y vídeo para Ethernet (*audio/video bridging* - AVB), a la que posteriormente siguieron las redes sensibles a la temporización (TSN).

En esta tesis se exploran aspectos avanzados en cuanto a la construcción, implementación y validación de sistemas TSN. De este modo, en la **Parte I**, se le proporciona al lector una descripción general de las tecnologías y especificaciones principales que gobiernan las redes TSN. Asimismo, ponemos en contexto el uso y la definición de perfiles para TSN, al tiempo que enfatizamos su papel fundamental como garantes de la adopción generalizada de TSN en múltiples industrias y aplicaciones, al proporcionar las "plantillas" que definen de forma maestra el diseño y parámetros que debe cumplir un sistema TSN para adaptarse a una aplicación determinada. A continuación, presentamos las metodologías experimentales, herramientas, material de laboratorio, y el diseño de los bancos de pruebas que utilizamos en las etapas de desarrollo y caracterización experimental. De este modo, conseguimos construir sistemas TSN convergentes al combinar todos estos elementos. La convergencia de flujos de datos en TSN es la propiedad fundamental que llevará al eventual reemplazo de los buses de campo, dado que los sistemas TSN pueden manejar tanto datos críticos, como flujos *best-effort*, y protocolos de sincronización simultáneamente. En este contexto, también desglosamos un caso de uso con la sincronización de White Rabbit (WR) en el Array de Telescopios de Cherenkov

(CTA) como una motivación en la que se ilustra cómo al integrar esta tecnología con TSN se posibilita el despliegue de los mismos para aplicaciones científicas.

La **Parte II** de la memoria se dedica al desarrollo e implementación del sistema TSN. De este modo, presentamos las plataformas hardware empotradas y las placas de desarrollo de nuestro sistema, que están basadas en los dispositivos Zynq-7000 de Xilinx: la placa WR-ZEN y la placa Main. Además, también desglosamos su arquitectura software y hardware durante esta sección. Comenzamos con los diferentes subsistemas FPGA que se combinan en nuestro diseño para construir el nodo TSN (Ethernet, sincronización, sistema de conmutación, y los propios módulos de TSN), para seguidamente pasar a examinar los elementos software que les proporcionan soporte (sistema operativo, interfaces de programación, *drivers* de red, ...). Al diseño de los componentes del sistema TSN se les dedica especial atención, con un capítulo separado en el que se explica de forma detallada el diseño del clasificador de tráfico (TAS), el módulo de VLAN, la MAC de Ethernet y el TAS mejorados con la funcionalidad de interrupción de tramas (*frame preemption*), así como los módulos para la redundancia. Esto, además, nos llevó a diseñar una arquitectura genérica que es compatible con las principales funcionalidades y componentes de TSN que se sustenta sobre un diseño que hace un uso moderado de los recursos de la FPGA: 802.1AS (gPTP), 802.1Qbv (TAS), 802.1Qbu (TAS con *preemption*), 802.3br (MAC de Ethernet con *preemption*), 802.1CB (transmisiones redundantes), 802.1Q (etiquetado con VLAN e identificación de flujos de tráfico).

Nuestros casos de uso experimentales se detallan en la **Parte III**, donde aplicamos TSN en escenarios industriales y de aeroespacial. Comenzamos explorando la aplicación de TSN a una subestación eléctrica, que representa nuestro caso de uso en industrial. En este escenario, procedimos a reemplazar las interfaces de señalización analógica de la subestación por un sistema TSN, el cual también era capaz de agregar todos los datos y protocolos presentes en la subestación. Esto permitió verificar que era posible emplear una red TSN para este escenario de forma efectiva, al ser capaz de entregar los datos críticos de manera más rápida y con mayor fiabilidad que las interfaces convencionales de la subestación. De modo adicional, también establecimos que el determinismo del sistema estaba condicionado por la configuración del usuario. También hemos realizado la implementación de un caso de uso prometedor en aeroespacial al diseñar e implementar los nodos de aviónica del microlanzador Miura 1. En este marco, analizamos las principales consideraciones para diseñar dicho sistema de aviónica, presentamos su arquitectura, y caracterizamos su rendimiento de manera exhaustiva. Como resultado, demostramos que nuestra implementación de TSN puede reemplazar de forma eficaz a los buses de campo habituales en los vehículos para espacio, que podrían quedar desplazados en favor de una solución basada en componentes comerciales (COTS). Además, aprovechamos el buen funcionamiento del sistema en nuestros resultados para hacer una primera propuesta de un perfil de TSN para espacio.

La integración experimental de la sincronización WR con el sistema de TSN se detalla en la **Parte IV** de la memoria, en donde se dedica un capítulo entero a presentar la investigación conjunta que se llevó a cabo con los investigadores de la Universidad Técnica de Dinamarca (DTU) durante la estancia del doctorado. Una parte importante de la estancia en DTU tuvo como objetivo estudiar la producción automática de parámetros de configuración para sistemas de TSN con herramientas especializadas. De este modo, en el capítulo se explica el desarrollo de la arquitectura "híbrida" con WR y la posterior caracterización de sus retardos internos de procesamiento, cuyo conocimiento era necesario para usar estas herramientas. Seguidamente, presentamos resultados preliminares en los que llegamos a la conclusión de que, a pesar de que se ha conseguido integrar la sincronización WR de forma exitosa con el sistema TSN, todavía hacen falta mejoras adicionales en nuestra arquitectura para mejorar su determinismo. Ésta será una de las principales áreas a cubrir en el trabajo futuro.

La **Parte V** muestra las conclusiones del trabajo de la tesis. De este modo, se enfatiza el haber realizado la implementación exitosa de un sistema TSN determinista basado en una arquitectura para FPGA que es altamente personalizable a la vez que adaptable, por lo que se puede ajustar y parametrizar a múltiples dispositivos y escenarios. En este contexto, cabe destacar la realización con éxito de casos de uso para las Smart Grids y para la aviónica del microlanzador del Miura 1. También se han mostrado resultados preliminares de una integración experimental con WR para explorar la posible aplicación de sistemas TSN en infraestructuras científicas, así como otros aspectos avanzados en relación con la generación automática de parámetros de configuración. Se concluye con una evaluación del nivel general de cumplimiento de los objetivos iniciales del trabajo y proponiendo una hoja de ruta para la mejora y actualización del sistema en el trabajo futuro. Por último, hemos incluido algunas consideraciones en los Apéndices sobre temas seleccionados relacionados con el desarrollo de drivers de red en entornos Linux empotrados.

ABSTRACT

Deterministic communications are an essential requirement for a variety of different application domains, such as the industrial plants, the automotive networks, or the aerospace systems. Traditionally, they have relied on the use of specialized fieldbuses to fulfill their communication needs. Some well-known examples are the *CAN* or the *FlexRay* buses for the automotive, or *Spacewire* for space vehicles. Some of these solutions are either proprietary or may require that they be used in combination with specialized equipment. Hence, the landscape of fieldbus communications is fairly diverse, presenting a range of differing solutions that range from the purely analog interfaces (e.g., HART) all the way to the fully digital ones. These latter cases typically make use of serial digital interfaces, and particularly there is a growing trend that seeks to build functioning implementations of different subtypes of fieldbuses out of standard Ethernet components (e.g., Profinet). These solutions make up for the lack of deterministic communication capabilities of ordinary Ethernet interfaces, which can do a great job at transmitting substantial amounts of data with large throughputs, although they can only provide a best-effort type of service for the delivery of data. Nonetheless, the shift towards Ethernet-based interfaces for the construction of fieldbuses has made the community realize that their use has apparent advantages, such as enhanced performance and streamlined compatibility and integration. This in turn was the germ of a larger shift towards the definition of generic, standard-based deterministic Ethernet interfaces. Consequently, this has materialized into the emergence of *audio/video bridging* (AVB) first, and then it was followed by the upgrade of *time-sensitive networking* (TSN).

This thesis explores advanced topics with respect to the construction, implementation, and validation of TSN systems. Hence, in *Part I*, we provide the reader with an overview of the main technologies and specifications that lie at the foundation of TSN networks. We also introduce the use and definition of profiles for TSN and emphasize their pivotal role in ensuring that the adoption of TSN will be widespread in multiple industries and applications, as they provide well-defined "templates" tailored to the requirements of a specific application domain. After that, we present the experimental methodologies, tools, laboratory material, and the layout of the test benches that we used during the development and experimental characterization stages. We built convergent TSN network systems with the use of these elements. This convergence is the fundamental property that will allow the eventual replacement of fieldbuses. Hence, TSN networks can handle critical data, best-effort traffic, and timing synchronization protocols simultaneously. In this context, we also introduce a motivational case with White Rabbit (WR) synchronization in the Cherenkov Telescope Array to illustrate how an integration with TSN could pave the way for the use of TSN networks for scientific infrastructures.

Part II of the manuscript is devoted to the implementation and development of our TSN system. We introduce our embedded hardware platforms and development boards based on the Zynq-7000 devices from Xilinx: the WR-ZEN and the Main Board. Furthermore, we present the main software and hardware components of our architecture in this section.

We start by looking at the various FPGA subsystems that have be combined to build a TSN node (Ethernet, timing, switching, and the TSN cores themselves), and then we examine the corresponding software elements for supporting them (the operating system, application programming interfaces, network drivers, . . .). We make special emphasis in the description of the implementation, design, and construction of the elements of the TSN system by devoting a specialized section for their documentation. This is where we provide an in-depth explanation of the design of the time-aware traffic shaper (TAS), the VLAN module, the preemptable Ethernet MAC, the preemptable TAS, and the modules for the seamless redundancy feature. Hence, we present the design of a generic architecture with the capability for supporting the main subcomponents of TSN while making moderate use of FPGA resources: 802.1AS (gPTP), 802.1Qbv (TAS), 802.1Qbu (preemptable TAS), 802.3br (preemptable MAC), 802.1CB (seamless redundancy), 802.1Q (VLAN-tagging and traffic identification).

We present our experimental use cases with TSN in the industrial and aerospace domains in **Part III**. We start by exploring the application of TSN to an electrical substation. This is our industrial case where we replace the analog signaling interfaces of the substation with a TSN system that can also aggregate all the substation protocols and data that are present in this scenario. We verified that a TSN system could be put to this use effectively as it could deliver critical data faster and more reliably than the traditional substation interfaces. In addition, we found that the attainable determinism of the system was dependent on the user settings. We also present a promising use case for aerospace in this part of the manuscript: the design and implementation of the avionics nodes of the Miura 1 microlauncher. In this framework, we show the main design considerations of the system, present its architecture, and carry out a thorough characterization of its performance. All of this demonstrates that our TSN implementation can effectively replace the usual fieldbuses for aerospace and, thus, implement a functional avionics system with off-the-shelf components (COTS) instead. Furthermore, we take advantage of these results to make an early proposal of the elements that should be included in an aerospace profile for TSN.

We introduce an experimental integration of WR timing and our TSN system in **Part IV** of the manuscript, where we devote a chapter to presenting the research that we conducted jointly with collaborators from the Technical University of Denmark (DTU) during a research visit. A substantial part of the research at DTU aimed to study the automatic production of configuration settings for our TSN system with specialized tools. Hence, the chapter presents the "hybrid" architecture with WR timing alongside a characterization of internal processing delays required by the tool. We present some preliminary results where we conclude that, even though we have successfully integrated our system with WR timing, we still need to supply additional improvements to our architecture to improve its determinism. This will be explored in the future work.

Part V contains the conclusions of the thesis project. Thus, we show that we have implemented a deterministic TSN system using a highly customizable FPGA architecture, that is also adaptable, and that can be fitted and targeted to multiple devices and scenarios. In this context, we emphasize our successful implementation of the system for major use cases in the Smart Grid and for the avionics of the Miura 1 microlauncher. Also, we have shown the preliminary results of our integration with WR timing for exploring the application of TSN to scientific infrastructure and other advanced topics related to

the automatic generation of configuration parameters. We conclude by assessing the overall level of compliance of our initial objectives and with the proposal of an upgrade path for the system as future work. Lastly, we have included some considerations in the Appendixes on select topics relating to network driver development for embedded Linux environments.

The world needs inventors - great ones. You can be one. If you love what you do and are willing to do what it really takes, it's within your reach.

— Steve Wozniak

AGRADECIMIENTOS

Le quiero agradecer a mis directores, Javier Díaz y Eduardo Ros, por su apoyo constante, orientación, y las oportunidades que me han dado. Gracias a ellos salgo con una visión mucho más madura, meditada y reforzada del papel que las tecnologías de comunicación determinista desempeñan a la hora de resolver problemas de ingeniería acuciantes en el mundo real.

Abordar todo este trabajo habría sido imposible sin el apoyo del grupo de investigación en sincronización de la Universidad de Granada, o si no hubiera contado con la ayuda inestimable de colaboradores en el Instituto Andaluz de Astrofísica (IAA) o en la empresa Seven Solutions, gracias a los cuales esta tesis ha podido explorar aspectos técnicos, experimentales, y casos de uso que son difíciles de encontrar en la universidad y que han enriquecido aún más su contenido.

Sobre el trabajo del comienzo de la tesis, me quería acordar de Antonio Miguel López, por toda la ayuda que me dio con los drivers de Linux, de Antonio Jurado por el apoyo en los experimentos de CTA, y de Pablo Marín por su trabajo en los TDCs de la ZEN-CTA. De José Luis Gutiérrez, por explicarme sobre sistemas empotrados y White Rabbit. Sobre el trabajo de TSN, le agradezco a Beatriz Aparicio del IAA su contribución a módulos importantes de la arquitectura, y por su orientación y consejos para el diseño de sistemas en espacio. También le estoy agradecido enormemente a Seven Solutions y a su equipo de TSN por su trabajo en la sincronización, en RTEMS, y en módulos principales del sistema. En concreto, a Rafael Rodríguez, Luis Medina, Jorge Machado, y Marco Fuentes. He aprendido muchísimo con vosotros y he podido participar en proyectos muy interesantes y apasionantes. En general, a todos los que han estado involucrados en los proyectos de TSN, desde el diseño de las placas a la validación. También me quiero acordar de Francisco Barranco, por darnos otra área de aplicación y estudio en videovigilancia.

¡Muchísimas gracias a todos!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

A

ADEV	46, 51, 164
Allan deviation.	
AMBA	80
Advanced microcontroller bus architecture.	
API	81–86, 88, 130, 152, 154, 175, 177, 181, 197, 217, 220, 223, 227, 231, 235
Application Programming Interface.	
ARM	40, 41, 71, 77, 79, 82, 89, 90, 92, 105, 140, 152, 178, 181, 189, 195, 196
Advanced RISC Machines.	
ARP	26
Address Resolution Protocol.	
ASIC	34, 56, 79
Application-specific integrated circuit.	

AVB 21, 28–30
Audio/Video Bridging.

AXI 80, 83, 85, 89, 90, 99, 105, 118, 123, 124, 133, 135, 140, 153, 154
Advanced eXtensible Interface.

B

BMCA 29, 31, 35, 81, 84, 175, 183, 186
Best Master Clock Algorithm.

BRAM 122, 180, 198, 217, 228
Block RAM.

C

CBS 29, 30, 32
Credit-based shaper.

CERN 40, 87
European Organization for Nuclear Research.

COTS 173, 174, 176, 177, 188, 189, 218, 229
Commercially available off-the-shelf.

CPF 22, 23
Communication Profiles Families.

CPU 73, 79
Central processing unit.

CRC 90, 119, 137, 138, 140
Cyclic redundancy check.

D

DMA 59, 62, 71, 73, 78, 83, 89–91, 93, 152, 179, 196, 198, 201, 202
Direct memory access controller.

DSP 104, 143, 180, 199, 217, 228
Digital Signal Processor.

E

ESA 181
European Space Agency.

F

FIFO 126, 131
First-in, first-out.

FMC 181
Front-mezzanine card.

FPGA . 6, 8, 15, 17, 40, 41, 44, 48, 61, 73, 77, 140, 154, 184, 196, 197, 216, 222, 226, 232, 234
Field-Programmable Gate Array.

FRER 101, 114–118
Frame replication and elimination for reliability.

FSM 85, 97, 98, 120, 123, 124, 131, 133, 211, 219, 222, 230, 235
Finite State Machine.

G

GCL85, 121–124, 126–131, 134, 143, 147, 151, 154, 162, 175, 184, 186, 187, 193, 194, 199, 202, 204–208, 210, 218–220, 223, 228–232, 235
Gate Control List.

GPS 56, 58, 72, 151
Global positioning system.

GPTP . . 8, 17, 29, 31, 34, 35, 45, 77, 81, 84, 87, 88, 97, 147, 151, 152, 154, 162, 164, 165, 170, 181–183, 185, 188, 189, 194, 200, 201, 210–212, 220, 221, 223, 232, 236
Generalized precision time protocol (PTP).

H

HDL 7, 42, 44, 54
Hardware description language.

HSR 32
High-Availability Seamless Redundancy.

I

IEEE 28, 34, 36–38, 134, 150, 194, 216, 226
Institute of Electrical and Electronics Engineers.

ILA 42, 44, 49, 201
Integrated Logic Analyzer.

IOT 37
Internet of Things.

IP 7, 17, 37, 40, 42, 48, 49, 78–80, 85, 86, 95, 141, 180, 194, 196–198, 200, 203
Intellectual Property.

J

JTAG 44
Joint Test Action Group.

L

LM32 196
LatticeMico32 soft-core processor.

LUT 104, 141, 180, 198, 199, 217, 228
Look-up Table.

LVDS 4, 14, 59
Low-voltage differential signaling.

M

MAC 5, 15, 32, 79, 84, 89–91, 93, 95, 96, 114, 118, 120, 132–136, 139, 152, 179, 196, 198, 202, 222, 234
Medium Access Control.

MDIO 93
Management Data Input/Output.

O

OPC-UA 37, 171
Object linking and embedding for process control - unified architecture.

OS 19, 39, 41, 44, 50, 77, 81, 82, 91, 152, 154, 175, 177, 179, 181, 189, 195, 197
Operating system.

P

PC 42, 161
Personal computer.

PCB 10, 19, 56, 79, 181, 182, 196
Printed circuit board.

PCI 92
Peripheral component interconnect.

PCP 162, 186
Priority code point.

PDV 8, 17, 28, 30, 45, 51, 147, 176, 184, 186, 187, 189, 193, 194, 204, 218, 228, 232
Packet Delay Variation.

PHC 123, 127, 154, 180, 183
PTP hardware clock.

PHY 91, 153, 179, 180
Physical-layer transceiver.

PID 196
Proportional-integral-differential controller.

PLC 37
Programmable logic controller.

PLL 87
Phase-locked loop.

PPS 45–47, 164
Pulse-per-second.

PRP 32
Parallel Redundancy Protocol.

PTP 29, 56, 73, 87, 123, 153, 194, 202, 212, 221, 232
Precision Time Protocol.

R

RF 155, 178
Radio frequency.

RX 155, 211
Reception.

S

SAE 216, 226
Society of Automotive Engineers.

SFP 43, 91
Small-form pluggable.

SOC 7, 16, 41, 77, 98, 141, 176, 197
System-on-chip.

SRC 120
Source.

SSH 86
Secure shell protocol.

T

TAI 127
International Atomic Time.

TAS . 10, 32, 79, 80, 85, 94–97, 120–124, 127–131, 134, 135, 142, 143, 151, 175, 180, 184, 186, 196, 198, 202, 206, 210, 211, 219, 220, 222, 230, 231, 234, 235
Time-Aware Traffic Shaper.

TCP 48
Transport control protocol.

TDC 46, 47, 51, 53–57, 59, 61, 71–73, 161, 184, 188
Time-to-digital converter.

TDEV 46
Time deviation.

TDMA 24, 124
Time-division multiple access.

TSN 5–8, 15, 17, 21, 25, 31, 32, 51, 53, 74, 77, 82, 116, 140, 143, 155, 174, 189, 193–195, 198–202, 210, 215–223, 225–230, 232, 234, 235
Time-Sensitive Networking.

TX 155, 211
Transmission.

U

UDP 48, 70, 71, 87
User datagram protocol.

UTC 129
Coordinated Universal Time.

V

VCO 88, 196
Voltage-controlled oscillator.

VHDL 48
Very high-speed integrated circuit hardware description language.

VIO 42, 44, 49
Virtual Input/Output.

VLAN . . 25, 79, 80, 85, 95, 96, 105, 115, 117, 118, 140, 143, 180, 186, 196, 198, 199, 201, 202, 211, 222, 234
Virtual Local Area Network Module.

W

WR7, 8, 16, 17, 39, 41, 53, 71, 73, 77, 84, 193–196, 198, 203, 210, 212, 217, 219–223, 226–230, 232, 234–236
White Rabbit.

Part I

Introduction and State of the Art

1 INTRODUCCIÓN

Este capítulo contiene la introducción al proyecto de tesis que se presenta en esta memoria. Se comienza con una motivación para el desarrollo de sistemas red basados en TSN. A continuación, se hace una exposición de cuáles son los objetivos principales que se persiguen con la realización del trabajo. Esto se complementa con la presentación del marco de trabajo de la tesis; incluyendo los proyectos y colaboraciones de transferencia industrial y científica que se han llevado a cabo como resultado de su realización. Además, para presentar este trabajo se han reutilizado los resultados de la publicación [1] en el Capítulo 5 para ilustrar el potencial del uso de la sincronización de White Rabbit con TSN. También proporcionamos las consideraciones adicionales de [1] en los Apéndices A y B. Por último, se describe de forma esquemática cuál es la organización del contenido en la memoria de la tesis.

Índice del capítulo

1.1 MOTIVACIÓN TRAS EL DESARROLLO DE COMUNICACIONES ABIERTAS Y DETERMINISTAS.

El uso de comunicaciones deterministas es un requisito fundamental en multitud de aplicaciones. Los ejemplos más comunes van desde las plantas industriales, las plataformas de sensores y mecanismos de control en los vehículos, hasta prácticamente cualquier tipo de entorno en el que exista la necesidad de establecer un bucle de control crítico o en el que sea necesario entregar mensajes de alta criticidad en un plazo temporal estricto. Se vienen empleando buses de campos especializados de forma tradicional con este fin, e incluso se dan casos en los que se usan interfaces analógicas sencillas. Estos buses de campo permitían establecer una comunicación determinista adaptada los parámetros de una aplicación o escenario determinados. Por lo tanto, a estos sistemas de comunicación normalmente se les ha denominado como "de mundo cerrado", y se caracterizan por su alto grado de eficiencia, fiabilidad, y robustez. Además, pueden proporcionar distintos valores de ancho de banda o de latencia extremo a extremo de forma que se puedan adaptar a los requisitos y limitaciones concretos de aplicaciones determinadas. Entre los ejemplos más conocidos se encuentran buses de campo tales como EtherCAT [2], CAN [3], Profinet [4], ... Sin embargo, presentan algunos inconvenientes considerables que hacen que su integración en sistemas a gran escala y heterogéneos presente numerosos desafíos, dado que una mayoría de las alternativas existentes entre los buses de campo son o bien soluciones propietarias, o dependen de equipamiento especializado vinculado con un proveedor específico. A su vez, esto se traduce en mayores costes de despliegue o de propiedad; lo cual se agrava aún más cuando se considera la falta de interoperabilidad entre distintos proveedores. De este modo, resulta común encontrar escenarios, como en el caso de la automoción, en los que los diferentes sensores y componentes de la red los suministran proveedores distintos; y en los que el equipamiento de cada proveedor puede incluso encontrarse vinculado a interfaces y protocolos de transmisión de datos distintos.

En este contexto, puede verse que desplegar un sistema funcional en estas condiciones puede ser una tarea inmanejable, dado que el sistema que acaba diseñándose tiene que asegurar que proporciona una representación coherente de los datos a través de toda la red, y posiblemente incluso entre múltiples interfaces o protocolos de transporte diversos, desde la fuente de datos y a lo largo de todo el sistema hasta el nodo donde se procesen. A menudo esto tiene como consecuencia que la arquitectura del sistema está fragmentada de facto in varios subsistemas gestionados con torres de protocolos o tecnologías de interfaz distintas. De este modo, es necesario usar elementos de "puente" (*bridge*) que actúen como conversor entre protocolos y pasarela de enlace entre los distintos dominios de bus. El caso de las redes de automoción se suele referir por lo paradigmático que resulta, dado que normalmente suelen presentar un arnés de cableado que integra múltiples protocolos e interfaces distintos, tales como FlexRay [5], CAN, e incluso señalización analógica con LVDS. Integrar todos estos componentes resulta complejo y conlleva costes sustanciales. Además, la multitud de interfaces que normalmente se encuentran en una red típica de autmóvil, junto con los elementos asociados de conversión y adaptación entre buses, hacen del arnés de cableado del vehículo moderno uno de los componentes más pesados y caros del mismo hoy en día. De este modo, esto sirve como un ejemplo típico para ilustrar cómo el uso de una capa de comunicaciones abierta y universal, que sea capaz de actuar de interfaz entre sistemas de distintos vendedores y de proporcionar flujos de datos deterministas, supondría una enorme simplificación en el proceso de diseño de sistemas complejos de "mundo cerrado". En el caso de las aplicaciones para automoción, esto tendría además el efecto colateral de aumentar la eficiencia del vehículo como resultado de la reducción del peso y tamaño del arnés de cableado, que se reduciría considerablemente si usara una única interfaz unificada.

Por el contrario, a diferencia de los sistemas de los sistemas de "mundo cerrado" que saldrían beneficiados de usar una interfaz de comunicaciones universal, los estándares que definen interfaces de este tipo normalmente se reservan para las comunicaciones de tipo masivo basadas en Internet. En este último ámbito es donde, por ejemplo, se definen estándares interoperables como TCP/IP o Ethernet. Estos protocolos, a diferencia de aquellos escenarios en los que son más prevalentes los buses de campo, se basan estándares abiertos e interoperables, por lo que se pueden usar como la base sobre la que desplegar redes de área amplia con anchos de banda considerables. Estos protocolos forman el grueso de las autodenominadas comunicaciones de "mundo abierto" y, en consecuencia, se han hecho ubicuas al garantizarse su interoperabilidad mediante un esfuerzo importante de estandarización. Además, el uso masivo de estas redes asegura el desarrollo continuo de innovaciones y mejoras con relativa frecuencia que pueden incorporar nuevas características de alto rendimiento.

Éste es el caso de las redes de tipo Ethernet, que representan una de la principales interfaces de comunicación en Internet. Además, se caracterizan por usar una capa de enlace fuertemente estandarizada de amplio conocimiento entre ingenieros y desarrolladores, dadas la extensa documentación disponible y su especificación abierta. Sin embargo, históricamente se ha empleado como la base de las comunicaciones conocidas como de tipo *best-effort* (entrega de mejor esfuerzo), que son las que normalmente se dan en entornos tradicionales asociados con Internet. Esto ha supuesto un impedimento sustancial para su adopción en sistemas críticos, dado que su entrega de mejor esfuerzo no puede garantizar ni un comportamiento determinista ni una variación del retardo de entrega reducidos. Son también propensas a experimentar pérdidas por congestión en aplicaciones con alta demanda de datos, a no ser que la capacidad del sistema se pueda sobredimensionar, lo cual aumentaría en mayor medida el coste general del sistema sin ser capaz de proporcionar ningún mecanismo de control explícito sobre el nivel de determinismo del tráfico más crítico. Estos factores contribuyen a explicar la prevalencia de buses de campo especializados en los sistemas de control, que sí pueden garantizar una latencia extremo a extremo acotada a la vez que presentan una solución más competitiva frente a alternativas equivalentes basadas en Ethernet.

La aparición de las redes sensibles a la temporización (TSN) ha supuesto una innovación revolucionaria en este aspecto; ya que cierra la brecha entre la comodidad de las interfaces abiertas e interoperables de las comunicaciones de Internet, y el determinismo y tiempo real de los buses de campo. Esto se consigue mediante el desarrollo de varias mejoras para el conjunto de estándares que gobiernan el funcionamiento de Ethernet, de modo que puedan pasar a emplearse como la infraestructura principal de comunicación de sistemas de "mundo cerrado" al ser capaces de hacer cumplir fuertes condicionantes de tiempo real y criticalidad para aplicaciones industriales, de automoción, o en aeroespacial. Estas mejoras, entre otros aspectos, actualizan el mecanismo de control de acceso al medio (MAC) de los dispositivos Ethernet con metodologías novedosas de clasificación de tráfico, procedimientos de enrutado estático y de reserva de recursos, o transmisiones redundantes. Además, se apoyan en un servicio de sincronización subyacente (*gPTP* [6]) que resulta esencial para conseguir asegurar que el sistema actúa de modo determinista para las transmisiones de datos en cada uno de los elementos de la red, llegando a establecer una latencia acotada bien definida para cada flujo de tráfico.

1.2 OBJETIVOS PRINCIPALES

El objetivo principal de este trabajo es construir y caracterizar un sistema de red TSN completamente funcional con características esencialmente novedosas. De este modo, por una parte, usar una arquitectura eficiente basada en FPGA con un consumo optimizado de recursos permitiría el uso del sistema en sectores tales como las Smart Grids o en el ámbito de aeroespacial. Por otra parte, también ha de explorarse el impacto que tiene usar métodos de sincronización de alta precisión, de modo que se puedan derivar conclusiones que ayuden al despliegue de sistemas TSN más avanzados, escalables, y deterministas que usen dispositivos FPGA reconfigurables. De manera gráfica, estas ideas se resumen en el diagrama de la Fig. 1.1.

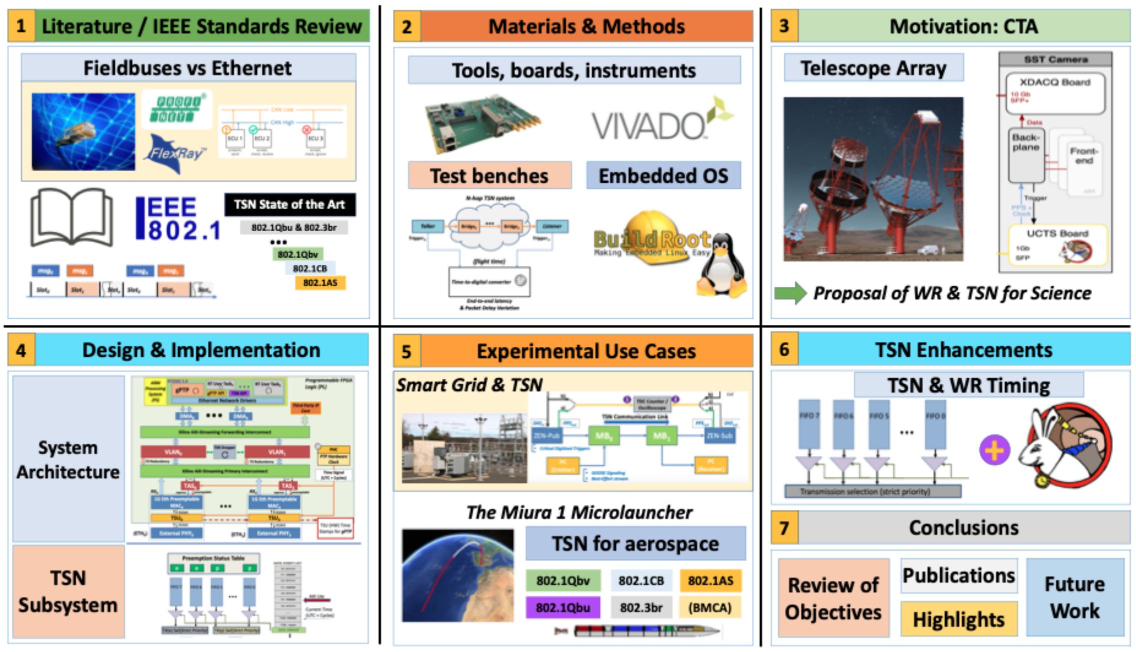


Figura 1.1
Resumen gráfico de la estructura de la tesis.

Tal como se aprecia en la imagen, una vez establecidad la motivación para desarrollar sistemas TSN y los objetivos principales del proyecto de tesis en este capítulo, pasamos a presentar y explicar en detalle los principales ejes temáticos de la tesis. En consecuencia, comenzamos proporcionándole al lector un repaso a los sistemas de comunicación deterministas y buses de campo, e introducimos el uso de Ethernet determinista como reemplazo de todas estas aproximaciones anteriores (*legadas*). Estos aspectos se examinan en el capítulo del Estado del Arte (1). Seguidamente, introducimos la metodología experimental, herramientas, y recursos que se han usado para construir los nodos TSN de nuestros experimentos. Esto se explica en detalle en la sección de material y métodos (2). A continuación, se explica un caso de uso de motivación para TSN (3) en el Array de Telescopios de Cherenkov (CTA), donde usamos un sistema de temporización con White Rabbit (WR) para adquirir sellos temporales de alta precisión al recibir ráfagas de luz Cherenkov. En este contexto, mostramos cómo nuestros datos científicos y la sincronización de WR pueden coexistir los mismos enlaces Ethernet, y razonamos cómo añadir TSN en este caso podría ser beneficioso para las infraestructuras científicas. Continuamos con la exposición detallada de la implementación de nuestros nodos TSN (4) a lo largo de dos líneas principales: la arquitectura general de los nodos, y la implementación del subsistema de TSN. Una vez explicado este último, presentamos nuestros principales casos de uso experimentales (5) para las Smart Grids y para la aviónica de un microlanzador en aeroespacial. Tras esto, exploramos los efectos de una integración de nuestros sistemas de TSN con la temporización de WR junto con sus efectos derivados sobre el determinismo (6). Por último, repasamos los objetivos del proyecto, revisamos su grado de cumplimiento, y esbozamos el trabajo futuro en las conclusiones (7).

Nuestros diseños se basan en los dispositivos Zynq-7000 de Xilinx, que integran un procesador ARM endurecido para ejecutar un sistema operativo empotrado, como Linux, así como lógica programable de FPGA para integrar *cores* (coprocesadores) logicos que pueden interactuar con el procesador. A estos cores se les suele denominar como *cores* de propiedad intelectual (IP) ya que están descritos únicamente en código HDL para que se sintetice sobre la FPGA. La elección de esta plataforma, que ha sido calificada como un sistema-en-chip (SoC) programable no es casual, dado que uno de los principales objetivos de este proyecto de tesis es evaluar la influencia de distintas metodologías de sincronización sobre el nivel de determinismo alcanzable en el sistema. Como el dispositivo Z-7015 [7] es una de las variantes de bajo coste de la plataforma Zynq-7000 que, además, ya se ha usado con anterioridad en el desarrollo del nodo "WR-ZEN" de alta precisión con White Rabbit [8], procedimos a seleccionarlo como el punto de partida y entorno de desarrollo de este trabajo, que se emprendió con los siguientes objetivos generales de alto nivel.

1. Explorar el diseño de sistemas basados en WR para infraestructura científica y su soporte para el envío simultáneo de datos e información de sincronización. En este contexto, tratar de mostrar un caso de uso que sirva de motivación y que resalte los beneficios potenciales de integrar las características de las redes TSN junto a la temporización de WR.
2. Desarrollar un conjunto de *cores IP* para FPGA configurables que proporcionen las características básicas de un sistema TSN, que hagan un uso de recursos moderado, y que permita simplificar su integración con *cores IP* y componentes de terceros en casos de uso más complejos.

   2.1) El sistema debe implementar un conjunto de metodologías y mecanismos para diferenciar los diferentes tipos de tráfico entre sí, de modo que puedan recibir un procesamiento diferenciado a lo largo de la red en función de su clase de tráfico y prioridad.

2.2) Además, la red debe permitir un método conveniente para que el usuario final del sistema pueda proporcionar parámetros de configuración y reservar rutas y recursos de enrutamiento para el envío de diferentes flujos.

3. Diseñar una arquitectura de FPGA parametrizable para TSN que presente capacidad de conmutación entre múltiples puertos.
4. Caracterizar el rendimiento del sistema TSN y mostrar su aplicabilidad a nuevos casos de uso experimentales en áreas fundamentales, como escenarios industriales en Smart Grid, o en casos de uso novedosos (por ejemplo, aeroespacial).

   4.1) El sistema debe ser capaz de agregar diferentes tipos de flujo de tráfico que van desde aquellos con servicio de mejor esfuerzo (BE), pasando por los de prioridad media, hasta aquéllos que son flujos críticos de alta prioridad, al tiempo que se respetan sus prioridades relativas.

   4.2) El mecanismo de agregación y reenvío debe ser capaz de evitar la interferencia entre los diferentes tipos de tráfico y mantener la integridad de los flujos de mayor prioridad.

   4.3) Dicho sistema también debería poder garantizar y hacer cumplir una metodología de transmisiones determinista para el tráfico crítico, asegurando una latencia extremo a extremo acotada con una variación en el retardo de entrega de los paquetes reducida (PDV) por cada clase de tráfico.

   4.4) El sistema también debe integrar capacidades de transmisión y recepción redundantes, de modo que le brinden de una mayor robustez al garantizar la entrega de los datos más críticos mediante el uso de rutas físicas de transmisión alternativas y de respaldo.

5. Integrar un mecanismo de sincronización de alta precisión, como White Rabbit (WR), junto con el mecanismo de sincronización por defecto de TSN (gPTP) para estimar los efectos sobre el determinismo del uso de un sistema de sincronización más preciso.

   5.1) Se debe evaluar la interoperabilidad entre la sincronización de White Rabbit y la sincronización gPTP por defecto incluida en TSN y, cuando sea necesario, se deben preparar planes para permitir la interoperabilidad entre ambos.

6. Proponer una ruta de actualización para mejorar el determinismo general de un sistema TSN aprovechando las nuevas características de nuestra arquitectura, así como el uso de una sincronización mejorada.

   6.1) Se debe permitir la interoperabilidad entre la sincronización de White Rabbit y la sincronización gPTP por defecto incluida en TSN mediante la definición de un plan de actualización que lleve a la compatibilidad total entre ambos sistemas. Estas actualizaciones serán parte de las tareas de mejora futura que seguirán a los resultados de este proyecto de tesis.

   6.2) El diseño debe garantizar la interoperabilidad y la compatibilidad descendente con otros sistemas Ethernet convencionales o entre otros dispositivos TSN basados en diversas implementaciones de la capa física; como las que usan par trenzado de cobre o fibra óptica.

1.3 EL MARCO DE TRABAJO DE LA TESIS

Este trabajo de investigación se ha desarrollado en el marco de diferentes acciones de transferencia de tecnología, incluyendo proyectos de investigación nacionales y europeos que han actuado como los principales motores científicos de nuestro trabajo, y agentes financieros para apoyar nuestros esfuerzos de desarrollo. Se enumeran los más relevantes a continuación:

A) **Contratos de transferencia tecnológica con nuestro socio industrial Seven Solutions**:

   – **ACTECA. *"Aceleradores tecnologías Asociadas para Grandes instalaciones Científicas* "**: Proyecto financiado por el CDTI español donde se subcontrató a la Universidad de Granada (UGR) para el desarrollo de nuevas tecnologías para el control de aceleradores de partículas y para la difusión de RF a través de redes Ethernet .

   – **Redes deterministas basadas en TSN para sistemas RF de aceleradores de partículas**: Gobierno de España, convocatoria Innoglobal con el objetivo de evaluar las tecnologías TSN como sustitución de la interfaz de bus para la transmisión de datos y datos de control para aceleradores de partículas.

   – **Comunicaciones deterministas para la Industria 4.0: TSN para Smart Grid**: Granada Ontech technology cluster, convocatoria para *"Agrupaciones Empresariales Innovadoras"* del MINECO (España). Su principal objetivo fue la evaluación del uso de tecnologías TSN para aplicaciones en Smart Grid.

B) **Proyectos de investigación nacionales: *AMIGA6 & AMIGA7***. Un proyecto nacional *MICIN* liderado por el Instituto Andaluz de Astrofísica (IAA) en el que participó la Universidad de Granada con el objetivo de evaluar tecnologías de transferencia de temporización en el marco del array de telescopios Square Telescope Array (SKA).

C) **Proyectos europeos: *ASTERICS***. Clúster de Infraestructura de Investigación y Astronomía ESFRI, ID de proyecto: 653477, financiado en el marco de la convocatoria H2020-INFRADEV-1-2014-1. Nuestro objetivo en este proyecto era aplicar mejoras basadas en las tecnologías White Rabbit a infraestructuras para astrofísica, como es el caso de los telescopios KM3Net, CTA o SKA.

Como consecuencia de nuestra participación en todos los proyectos y contratos antes mencionados, tuvimos la oportunidad de colaborar y trabajar conjuntamente en la especificación del proyecto, implementación del código fuente y definición de los casos de uso con muchos de los actores con los que trabajamos. Por tanto, los resultados y parte del desarrollo del sistema que presentamos en esta memoria son consecuencia de un importante esfuerzo de colaboración. Esta colaboración fue esencial, ya que una parte importante de nuestros resultados y los conocimientos que derivamos de ellos no se habrían podido realizar si no hubiera sido por las contribuciones de nuestros colaboradores en los proyectos de este marco de trabajo. En particular, los módulos de temporización de White Rabbit de alta precisión para algunos de nuestros experimentos los proporcionó el grupo de investigación en sincronización de la Universidad de Granada, así como la amplia comunidad del repositorio del Open Hardware (OHWR). El diseño del clasificador de tráfico sensible a la temporización (TAS) [9] se realizó junto a colaboradores del Instituto Andaluz de Astrofísica (IAA), quienes también proporcionaron consejos y orientación claves sobre aspectos del diseño de plataformas para aplicaciones aeroespaciales. Por último, nuestro colaborador industrial Seven Solutions sumunistró algunos de los *cores IP* para FPGA que usamos para la configuración general de las placas PCB de nuestros nodos, el firmware integrado para soportar la ejecución de gPTP, y los componentes para implementar las mejoras de interrupción de tramas para TSN (*frame preemption*). También se encargaron del desarrollo del sistema operativo RTEMS de nuestras plataformas empotradas. La combinación de todos estos elementos ha hecho que sea asequible asumir la enorme cantidad de trabajo prevista en este proyecto de doctorado, liberándonos así del desarrollo de algunos de sus aspectos más puramente técnicos, y haciendo que sea factible explorar los principales desafíos científicos de nuestra investigación y producir los resultados de la misma dentro de un marco temporal razonable para la finalización del trabajo de la tesis.

1.4 VISIÓN GENERAL DE LA ESTRUCTURA DE LA TESIS

Una vez presentados la motivación y los requisitos de este trabajo, el desarrollo y la caracterización de los sistemas TSN en los que se ha trabajado durante la tesis se irá describiendo en las secciones correspondientes de esta memoria, de acuerdo con la estructura que se reseña a continuación.

- Comenzamos proporcionando al lector un análisis en profundidad de los antecedentes y las capacidades de las tecnologías que conforman el grueso de las mejoras que llevaron a la aparición de las redes sensibles a la temporización. Esto se examina en el Estado del Arte en el Capítulo 3.
- A continuación, usamos el Capítulo 4 para presentar los materiales y métodos utilizados a lo largo del estudio.
- Seguidamente, a modo de motivación, explicamos el caso del Array de Telescopios Cherenkov en el Capítulo 5, como ejemplo de un caso en el que el uso de una sincronización muy precisa puede verse complementada con un sistema de TSN. De este modo, se sientan las bases de muchos de los elementos de la arquitectura del sistema que se implementarán a partir de este punto.
- El diseño de los nodos TSN en sí se divide a su vez en varios capítulos separados, cada uno especializado en diversos aspectos del sistema.
  - El Capítulo 6 se usa para presentar la arquitectura global del sistema.
  - El Capítulo 7 da una descripción detallada de cada subcomponente y *core IP* del subsistema de TSN de los nodos de red que hemos diseñado.
- Posteriormente, procederemos a caracterizar el rendimiento del sistema TSN con el despliegue e implementación de redes TSN en dos casos de uso representativos.

  - En una aplicación que integra el tráfico de todos los datos de una subestación eléctrica en el contexto del despliegue de las Smart Grids, que se presenta en el Capítulo 8.
  - En el desarrollo del sistema de aviónica para un microlanzador para un caso en aeroespacial, que se presenta en el Capítulo 9.

- La integración con la sincronización de White Rabbit y la evaluación de la influencia del uso de diferentes mecanismos de sincronización para TSN se analiza en el Capítulo 10.
- Las conclusiones y mejoras futuras se presentan en el Capítulo 11 (Capítulo 12 con la versión en castellano).
- Por último, los Apéndices A y B contienen, respectivamente, varias consideraciones sobre el entorno de software de nuestras placas, así como una descripción de las tareas de optimización generales que hemos llevado a cabo al desarrollar drivers de red de Linux para interfaces Ethernet.

2 INTRODUCTION

This chapter contains the introduction to the thesis project that we have presented in this manuscript. We start off with a motivation for encouraging the development of TSN-based networking systems. Next, we review the main objectives that we aim to achieve with the studies that we have included in this thesis. This is complemented with an introduction into the framework of our research during the thesis project, where we acknowledge all the industrial and scientific transfer activities that we have performed during its elaboration. Furthermore, we have reused the contents of the publication in [1] in Chapter 5 in order to illustrate the potential of the use of the White Rabbit synchronization in combination with the features of TSN systems. We also include the additional considerations of [1] in the Appendixes A and B. Lastly, we conclude by providing a simplified description of the structure of the contents of the manuscript.

Chapter contents

2.1 MOTIVATION. THE DRIVE TO DEVELOP OPEN, DETERMINISTIC COMMUNICATIONS.

Deterministic communications are a major requirement found in multiple applications. The usual examples include industrial plants, automotive control and sensing platforms, electrical substation facilities, or any other environment where there is a necessity to establish a time-critical control loop or deliver highly critical messages with a strict reception time bound. To this end, we have traditionally relied upon specialized fieldbuses or even simple analog interfaces tailored to fulfill the role of a deterministic communication system in their respective domains or applications. These are the so-called "closed world" communication systems, which are characterized by their high degree of efficiency, reliability, and robustness. Moreover, they can manage to attain different ranges of bandwidth or end-to-end latency figures to conform to the particular set of constraints of a given application. Some well-known examples could be found in fieldbuses such as EtherCAT [2], CAN [3], Profinet [4], ... Nonetheless, they have some considerable shortcomings that make their integration for larger scale systems challenging as many fieldbus alternatives are either proprietary solutions or dependent on specialized, vendor-locked equipment. This in turn results in higher deployment and ownership costs, which are compounded by a usual lack of interoperability amongst different vendors. Hence, it is often common to find some scenarios, like the case of automotive applications, where the multiple components and sensors of the network are sourced from several suppliers, with each one supporting different interfacing and data transmission protocols.

In this framework, it can be seen that deploying functional systems under these conditions can easily become an unwieldy endeavor: the system design has to ensure a consistent representation of data throughout the network, and possibly across multiple interfaces or transport protocols, from their source all the way to their processing node. Oftentimes, this results in an architecture that is effectively split into multiple subsystems handled by a different protocol stack or interface technology, with several bridging nodes that operate as gateways or protocol converters amid the multiple domains. The case of automotive networks is paradigmatic, as these usually consist of a wiring harness that integrates multiple protocols and interfaces; such as FlexRay [5], CAN, or even analog signaling like LVDS. The integration of all of these components is complex and carries a substantial cost. Furthermore, the myriad interfaces found in a typical automotive network, and all their corresponding bridging and adaptor elements, make the wiring harness one of the heaviest, most expensive components of modern automobiles. Hence, this is a typical example of how the use of an open, universal communications layer, capable of interfacing with systems from different vendors and of handling deterministic traffic, would enormously simplify the design process of complex, "closed world" systems and, in the case of automotive applications, even help increase fuel economy as a result of the reduced size and weight of a "decluttered" wiring harness.

Nonetheless, contrary to the "closed world" control systems that stand to benefit from such a universal communication layer, it is often the more massive, Internet-like systems that are the realm of the open, interoperable interfaces and protocols such as TCP/IP or Ethernet. These protocols, in contrast to the domains where fieldbuses are more prevalent, are based on open, interoperable standards, and hence can be used as the foundation for deploying wide area networks with relatively large bandwidths. These protocols make up the so-called "open world" communications and, as a result, have become pervasive and heavily standardized to guarantee interoperability. Moreover, the massive utilization of these networks provides continuous innovations that bring high-performance features very frequently.

This is the case of Ethernet networks, which represent one of the main communication interfaces for the Internet. Furthermore, they are a heavily standardized link layer protocol that is well-known in the engineering community. However, it has historically been targeted at supporting the best-effort type of service commonly found in Internet-related applications. This has been a substantial hurdle for its deployment to support time-critical applications, as the best-effort data delivery of Ethernet networks cannot guarantee a deterministic behavior or a reduced packet delay variation. They are also prone to experiencing congestion losses under data-intensive applications unless the system capacity can be overprovisioned, which further increases its overall cost but still does not provide any explicit control on the level of determinism of the time-critical traffic. These facts account for the dominance of specialized fieldbuses for control systems, which can guarantee end-to-end bounded latency and also remain more competitive than their equivalent Ethernet-based solutions.

The emergence of time-sensitive networking (TSN) has thus been a disruptive innovation that has aimed to close this gap between the convenience of the open, interoperable interfaces of Internet-like communications and the real-time determinism of fieldbuses. This has been achieved with the development of several enhancements for the Ethernet set of standards that have eventually allowed their deployment as the core communication infrastructure of "closed world" industrial, automotive, or aerospace systems with tight determinism and criticality constraints. These enhancements, amongst other things, improve the legacy medium access control (MAC) layer of Ethernet devices with novel traffic shaping schemes, packet inspection methodologies, static route and resource reservation procedures, or redundant transmissions. Furthermore, they rely on an underlying synchronization service (*gPTP* [6]) that helps enforce the deterministic transmission and behavior of the network elements (i.e. well-defined bounded latency).

2.2 THE MAIN OBJECTIVES

The main objective of this study is to build and explore a fully functional TSN networking system with some novel and key features. On the one hand, an efficient architecture with optimized utilization of TSN systems based on reconfigurable FPGA devices would allow their utilization in markets such as the Smart Grid or Space. On the other hand, the impact of timing accuracy will also be explored to derive new insights about the future deployments of more advanced, scalable, and deterministic TSN systems using reconfigurable FPGA devices. All of these ideas can be graphically represented and summarized in the diagram from Fig. 2.1.

As seen in the figure, once we have established the motivation for developing TSN systems and the main objectives of our project in this chapter, we move on to presenting and thoroughly explaining the main subjects of the thesis. As a result, we start off by providing the reader with a review of deterministic communication systems, fieldbuses,

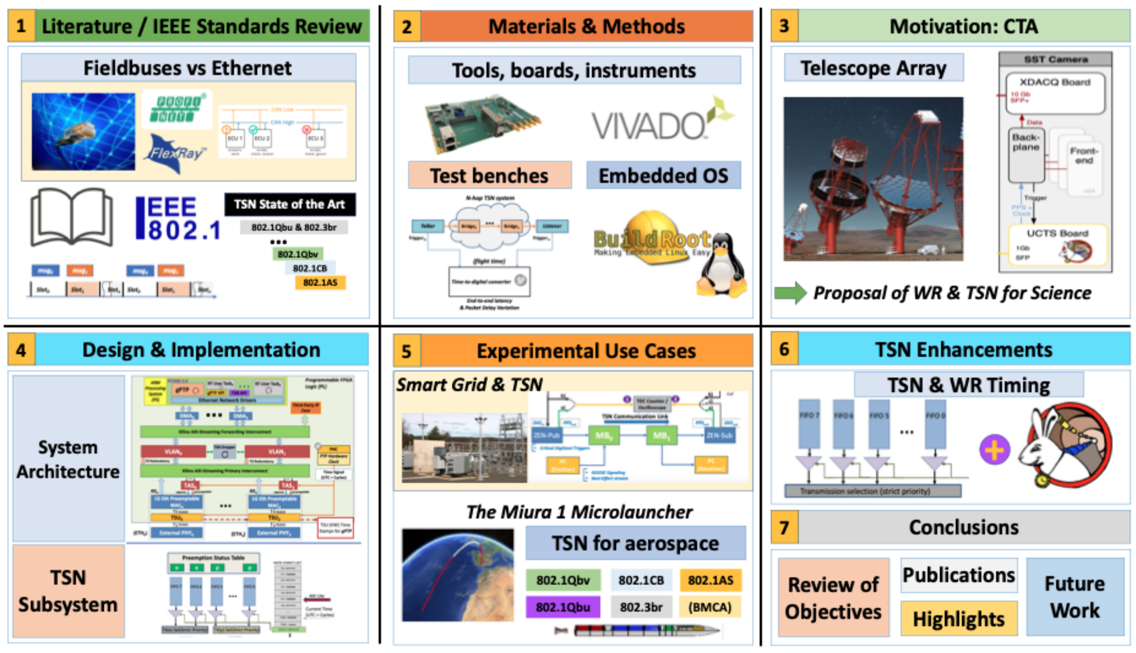


Figure 2.1
Graphical overview of the structure of the manuscript.

and the application of deterministic Ethernet in the form of a TSN system to replace all of these legacy approaches. These aspects are examined in the State of the Art (1). Afterwards, we introduce the experimental methodology, tools, and resources that we have used for building the TSN network nodes for our experiments. This is explained in detail in our section for materials and methods (2). Next, we present a motivational case (3) in the Cherenkov Telescope Array (CTA), where we used a WR timing system to acquire high-accuracy time stamps upon receiving Cherenkov flashes. We show how our scientific data and the WR synchronization can coexist over the same Ethernet links; and discuss how TSN could be beneficial for scientific infrastructures. Next up, we break down the implementation of our TSN nodes (4) along two main lines: the general architecture design of the nodes, and the implementation of their TSN subsystem. Once we have explained the TSN system, we present our main experimental use cases (5) in the Smart Grid and for the avionics of an aerospace microlauncher. After that, we explore the effects of an integration of our TSN systems with WR timing and its effects on determinism (6). Lastly, we recount the objectives, review the degree of compliance, and set out the future work in the conclusions (7).

Our designs are based on the Zynq-7000 devices from Xilinx, which bundle together a hardened ARM processor that allows the execution of an embedded operating system, such as Linux, and programmable FPGA logic for building intellectual property (IP) cores that can interface with the processor. The choice of this platform, which has been dubbed a programmable system-on-chip (SoC), is not coincidental, as one of the main goals of this thesis project is to carry out an evaluation of the influence of different methods of network synchronization on the attainable determinism of the system. Since the Z-7015 [7] is a low-cost Zynq-7000 platform that had previously been used for developing the highly accurate "WR-ZEN" White Rabbit timing node [8], we selected it as the starting point and development environment for this study, which presented the following high-level objectives:

1. Explore the design of WR-based systems for scientific infrastructure and their support for joint data and timing information transmission. In this context, show a motivational use case that could benefit from including the features of TSN networks alongside WR timing.
2. Develop a set of customizable IP cores that provide the basic features of a TSN system with a moderately sized footprint that simplifies their integration with third-party IPs and components in more complex use cases.
   2.1) The system should implement a set of methodologies and mechanisms to tell different types of traffic apart from one another so that they can receive differentiated processing over the network on the basis of their traffic class and priority.
   2.2) Moreover, the network should allow for a convenient method for the end user to supply configuration parameters and reserve routing paths and resources for the forwarding of different flows.
3. Design a parameterizable FPGA architecture for TSN with multi-port switching capabilities.
4. Characterize the performance of the TSN system and show its applicability to new experimental use cases in both well-known areas, such as industrial scenarios in the Smart Grid, or novel use cases (e.g., aerospace).
   4.1) The system should be able to aggregate different types of traffic flows ranging from best-effort (BE), medium priority, through to critical high-priority flows whilst respecting their relative priorities.
   4.2) The aggregation and forwarding mechanism should be able to avoid interference amid the different types of traffic and maintain the integrity of the higher priority flows.
   4.3) Such a system should also be able to guarantee and enforce a deterministic forwarding behavior for critical traffic by ensuring bounded end-to-end latency and reduced packet delay variation (PDV) on a per-traffic-class basis.
   4.4) The system should also integrate built-in redundant transmission and reception capabilities that provide enhanced robustness by ensuring the delivery of the most critical data by providing an alternative, failsafe physical transmission path.
5. Integrate a highly accurate synchronization mechanism, such as White Rabbit (WR), alongside the default synchronization mechanism of TSN (gPTP) in order to estimate the effects on determinism of the use of a more accurate timing stack.
   5.1) The interoperability between White Rabbit timing and the default gPTP synchronization of TSN should be assessed and, when needed, plans should be made to allow the interoperability between the two.
6. Propose an upgrade path for enhancing the overall determinism of a TSN system by leveraging new architectural features and enhanced timing.

6.1) The interoperability between White Rabbit timing and the default gPTP synchronization of TSN should be allowed with the definition of an upgrade path for allowing the full compatibility between the two timing systems. These enhancements will be part of the tasks for future improvement following up the results of this thesis project.

6.2) The design should guarantee interoperability and backwards compatibility with other legacy Ethernet systems or TSN-capable devices over multiple physical layer implementations; e.g., copper twisted pair, optical fiber.

2.3 THE FRAMEWORK OF THE THESIS

This research has been developed in the framework of different technology transfer actions, including national and European research projects that have acted as the main scientific drivers of our work, and financial agents to support our development efforts. We list the most relevant of them in the following lines:

A) **Technology transfer contracts with the industrial partner Seven Solutions**:

- **ACTECA. "*Aceleradores tecnologías Asociadas para Grandes instalaciones Científicas*"**: Project funded by the Spanish CDTI where the University of Granada (UGR) was subcontracted for the development of novel technologies for the control of particle accelerators and for RF dissemination over Ethernet networks.
- **Redes deterministas basadas en TSN para sistemas RF de aceleradores de partículas**: Spanish government, Innoglobal call with the goal of evaluating TSN technologies as to replace the bus interface for transmitting data and control data for particle accelerators.
- **Comunicaciones deterministas para la Industria 4.0: TSN para Smart Grid**: Granada Ontech technology cluster, call for "*Agrupaciones Empresariales Innovadoras*" from Spanish MINECO. Its main objective was to evaluate TSN technologies for Smart Grid applications.

B) **National research projects: *AMIGA6 & AMIGA7***. A national *MICIN* project led by Andalusian Institute of Astrophysics (IAA) that the University of Granada participated in with the goal of evaluating time transfer technologies in the framework of Square Telescope Array (SKA).

C) **EU projects: *ASTERICS***. Astronomy ESFRI and Research Infrastructure Cluster, Project ID: 653477, funded in the framework of H2020-INFRADEV-1-2014-1 call. Our goal in this project was to apply enhancements based on the White Rabbit technologies to astrophysics infrastructures, such as KM3Net, CTA or SKA.

As a consequence of our participation in all of the aforementioned projects and contracts, we had the opportunity to collaborate and work jointly on the project specification, source code implementation and use case definition with many of these different actors. Hence, the results and part of the system development that we present in the manuscript are the consequence of a significant collaboration effort. This collaboration was paramount, as a significant portion of our results and the insights that we derive from them would not have been attainable had it not been for the contributions of our collaborators in the projects of our framework. In particular, the high-accuracy White Rabbit timing modules for some of our experiments were provided by the timing research group at the University of Granada and the greater open hardware repository (OHWR) community. The design of the time-aware traffic shaper [9] was conducted jointly with collaborators from the Andalusian Institute of Astrophysics (IAA), who also provided key advice for designing platforms for aerospace applications. Finally, some of the FPGA cores that we use for the overall configuration of the PCB boards of our nodes, the embedded firmware to support gPTP, the frame preemption components for TSN, or the development of the RTEMS OS for our embedded platforms were supplied by our industrial collaborator Seven Solutions. The combination of all these elements has allowed us to take on the huge amount of work envisioned in this project, thus relieving us from the development of some of its most purely technical aspects, so that it would be feasible to explore the key scientific challenges and results of our research within a reasonable timeframe for the completion of the thesis project.

2.4 OVERVIEW OF THE STRUCTURE OF THE MANUSCRIPT

Once the motivation and requirements of this study have been presented, the development and characterization of the TSN-capable systems developed throughout this thesis project will be described over the different sections of this manuscript, with the structure that we outline below.

- We start off by providing the reader with a more in-depth analysis of the background and capabilities of the technologies that make up the time-sensitive networking enhancements. This is examined in the State of Art contained in Chapter 3.
- Next, we use Chapter 4 to present the materials and methods used throughout the study.
- After that, we move on to examining a motivational case in Chapter 5 for the Cherenkov Telescope Array that shows how scientific infrastructure could complement very precise timing with a TSN system. This lays the foundation for many of the architectural elements that we implemented for our system thereafter.
- The design of the TSN nodes themselves and their functionalities are in turn split over several dedicated chapters.
  - Chapter 6 is used for presenting the global system architecture.
  - Chapter 7 gives a detailed description of each subcomponent and IP core in the TSN subsystem of our nodes.
- Afterwards, we characterize the performance of the TSN system with the deployment and implementation of TSN networks for two representative use cases.
  - An application for integrating the data traffic in an electrical substation for the rollout of the Smart Grid in Chapter 8.

  - The development of the avionics system for an aerospace microlauncher in Chapter 9.

- The integration with White Rabbit timing, and the assessment of the influence of the use of different synchronization mechanisms for TSN is discussed in Chapter 10.
- The conclusions and future improvements are presented in Chapter 11.
- Lastly, Appendixes A and B contain several considerations on the software environment of our boards and a description of our optimization efforts for developing Linux network drivers for Ethernet interfaces, respectively.

3 STATE OF THE ART OF TSN SYSTEMS AND THEIR APPLICATIONS

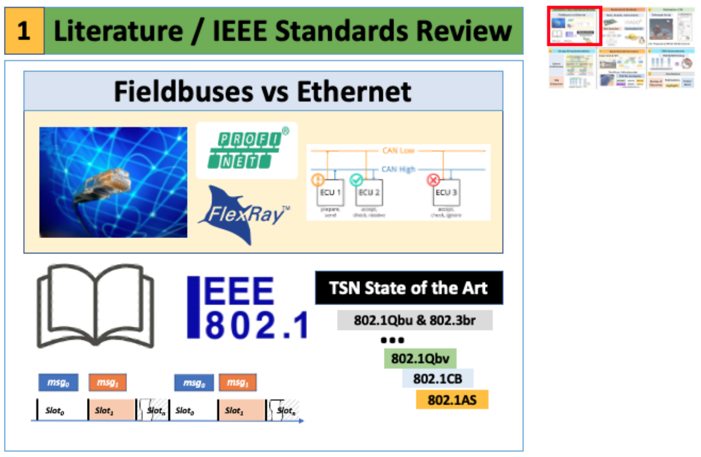


Figure 3.1
Overview of the contents of Chapter 3 with the State of the Art.

A review of the state of the art of TSN systems and their applications. This chapter introduces the background that led to the emergence and development of TSN networking systems. We start off by reviewing the prevalent fieldbus landscape that preceded this technology. Hence, we examine some of the main fieldbuses that were commonly used in the industry and their most usual areas of application, such as industrial automation, automotive networks, and aerospace. We also mention the main standardizations and specifications that encompass them. During the analysis, we emphasize how the design paradigm with fieldbuses started shifting towards Ethernet-based interfaces, which drove the industry to start the development of a series of enhancements to implement a fully deterministic Ethernet link layer. This translated into the initial specification of Audio/Video Bridging (AVB) first, which was later on replaced with the more advanced TSN systems that we present in this manuscript. We also provide an introduction to the main components and standards of TSN, a review of its main application profiles, and comments on the products and solutions from leading TSN equipment suppliers.

Chapter contents

3.1 REAL-TIME COMMUNICATION WITH FIELDBUSES

Real-time communication in industrial, automotive, or aerospace environments has usually been achieved through the use of dedicated fieldbuses. These generally implement deterministic communication services for transmitting the critical messages from multiple sensors or towards different actuators in an industrial plant. Hence, their use allows us to enforce the timeliness required for implementing processes such as control loops, which usually demand that data be received within a certain deadline for their control algorithm to be applied successfully.

Traditional and legacy systems relied on a number of methodologies for achieving this. For instance, in traditional industrial environments this could even be achieved with analog cabling, i.e., the implementers would deploy control systems that used dedicated wiring for controlling each one of the processes in the factory floor: from the sensing of the physical variables of the controlled processes to the activation of a certain actuator. Hence, these architectures were not scalable and required a considerable wiring harness that was also difficult to maintain and repair. Fieldbuses represented a significant improvement over this scenario.

Since the initial publication of one of the main specifications for fieldbuses (IEC 61158) [10] in the early 2000's, the standard has grown to accommodate a large number of the fieldbuses that are commonly found in the industry. Their goal, as stated in the presentation from [11], is to provide "an industrial network system for real-time distributed control." Thus, this would allow the seamless exchange of process control and manufacturing data in real time through technologies such as PROFIBUS, SwiftNet, HART, EtherCAT, SERCOS, MODBUS, ... Many of these protocols have already been included in the IEC 61158 specification. They provide an efficient mechanism for carrying out remote supervision tasks or implementing distributed control loops. They may even assist with advanced functionalities such as the remote calibration of the instrumentation in the plant.

As mentioned previously, there are multiple implementations of different fieldbuses. We can usually trace each "*flavor*" to a different manufacturer of industrial automation equipment. This is the case of EtherCAT [2], which was initially developed by Beckhoff Automation [12]. The subsequent IEC standardization encompassed most of these buses by supplying a broad definition of different profiles (IEC 61784) [13], whereby each fieldbus would fall within the category of one of the working fieldbus profiles contemplated in the specification. Thus, the different fieldbus profiles define several families (*CPFs*), with each family having an assigned collection of fieldbuses. Hence, *CPF1* includes the original FOUNDATION fieldbus, *CPF3* features the well-known PROFINET, ... The fieldbuses contemplated in the IEC 61784 specification represent a comprehensive list of available fieldbus solutions. Nonetheless, there are other alternatives, such as FlexRay [5], CAN [3], or AFDX [14], which are not included in the specification, but that have nonetheless been successfully applied to the real-time communications of different systems; such as the automotive or aerospace systems. Moreover, we can classify the wide family of fieldbuses according to additional criteria, such as their performance or their intended applications. We introduce some of their most significant classifications in the following points and give examples of applicable fieldbuses in each case.

3.1.1 *Level of performance*

From the standpoint of performance, many of these communication interfaces implement different methodologies for fulfilling the requirements of a real-time, deterministic exchange of data. Hence, on one end of the range, Profinet can achieve relatively high transmission speeds as it is built on top of the Ethernet link layer with fast forwarding mechanisms and custom protocols. Other fieldbuses, given their initial applications for sensing and control, may only be able to handle a few hundreds of kb/s and support update cycles (i.e., transmission windows) on the order of the millisecond. That was the case of the FOUNDATION fieldbus [15] or Profibus [16].

3.1.2 *Representative applications and use cases*

From the point of view of their intended applications, fieldbuses find their main uses in the areas of industrial automation, automotive communications, and aerospace systems. Moreover, they may also be applied as the foundation for building low-latency, reliable communications for the Internet of Things (IoT) and in other niche applications, such as home automation (e.g., KNX [17]).

a) **Industrial Automation:** Since the scenarios of industrial automation represented the original application domain of fieldbuses, we can expect to find a large number of protocols that could be applied to this end. Some typical examples can be found in the main CPFs contemplated in the IEC 61784 specification, such as Profibus or MODBUS. The former can achieve deterministic communication for data rates of up to 31.25 kb/s, whereas the latter dates back to an older specification that lacks determinism but that could nonetheless handle faster rates of up to 12 Mb/s. Many of the fieldbuses that find industrial applications may have either built-in or optional redundancy and feature the capacity to interconnect hundreds of devices. This helps ensure data integrity for the more critical messages while simplifying the process of deploying distributed sensing and control applications. In addition, they are supported with multiple types of physical layer interfaces. As an example, in the case of MODBUS, it is twisted pair wiring that gets used, whereas Profibus may be supported with a range of options from twisted-pair copper wiring to optical fiber. This allows the deployment of Profibus within environments or processes that might otherwise be prone to combustion or explosion (e.g., oil refineries, chemical processes, . . .).

b) **Automotive systems:** In-vehicle networks are another representative case of application for fieldbuses. Hence, the increasing number of sensor and actuator elements in modern vehicles (the steering system, the tire pressure sensors, the engine controller, ...) has made it necessary to find an appropriate communication system that could be deployed inside the car to interconnect all of its essential elements. Traditionally, this was achieved by combining different fieldbuses, so that each bus domain would be specialized in the handling of a different type of process such as the engine control or the braking system. The diversity of elements of different nature in vehicles often made the wiring harness of cars a bulky and complex topology with different kinds of buses. Nonetheless, we could claim that they all had in common that their interfaces needed to be able to withstand large temperature swings or high levels of electromagnetic interference. Some typical examples of the buses used in this domain are the CAN [3] or FlexRay [5]. Both are supported with twisted-pair physical interfaces with differential signaling, with FlexRay portrayed as an enhancement over the CAN bus with faster transmission rates: $\sim$ 1 Mb/s for CAN in contrast to the 10-Mb/s performance of FlexRay. They have traditionally been extensively deployed for in-vehicle communications in cars and other transportation systems ranging from trains, aircraft, and even through to elevators.

c) **Aerospace systems:** Aerospace is a completely different domain with its specific set of requirements. These systems usually require high resiliency to thermal variations, radiation-induced effects such as single-upset events, and robustness in data delivery. We have conducted a thorough review of these systems in the context of the study that we present in Chapter 9 on the development of the avionics communication system for an aerospace microlauncher. Some typical examples of these buses include the well-known MIL-STD-1553B [18], AFDX [14], Spacewire [19], or the time-triggered Ethernet protocol ("*TTEthernet*") [20]. The MIL-STD-1553B is nearly a legacy technology at this point. It uses TDMA for access control and could achieve a bandwidth of up to $\sim$ 1 Mb/s. It is on its way to being superseded by faster, more deterministic alternatives such as AFDX or Spacewire. Both can achieve hundreds of Mb/s in terms of throughput and make use of different traffic shapers to ensure determinism. Lastly, *TTEthernet* implements an Ethernet-based interface that uses traffic shapers for forwarding critical data deterministically.

3.1.3 *The shift towards deterministic Ethernet*

Throughout the evolution of the different standardization efforts for the multiple fieldbus technologies, it can be seen that there is a clear trend towards shifting the design paradigm of the physical and link layers of these interfaces. Hence, traditionally, a majority of field buses relied on some sort of serial communication interface that was usually based on differential signaling over twisted-pair wiring. That was the case of CAN, FlexRay, or MODBUS. A similar trend was observed in aerospace with the legacy MIL-STD-1553B. In contrast to these, a substantial number of the new fieldbuses gathered in the IEC 61784 specification underwent a change in their design paradigm and started evolving towards Ethernet-based interfaces. That was the case of EtherCAT, Profinet, or SERCOS. In particular, this trend was standardized in a specific real-time profile over Ethernet of the IEC specification (IEC 61784-2) [21]. As a result, the shift towards Ethernet had some obvious advantages such as the faster communication throughput. Moreover, the new fieldbus protocols would now be able to take advantage of the diverse variety of Ethernet interfaces and their link layer implementations (e.g., optical fiber, RJ45 connectors, ...). Thus, as was the case of EtherCAT, this usually meant that the fieldbus was built on top of standard Ethernet components and was complemented with specific message formats ("*telegrams*") or custom elements for switching and forwarding messages promptly with low latency. Furthermore, these messages are usually transmitted using standard Ethernet headers. This shift towards the link layer of Ethernet can be understood as part of a larger trend that seeks to make Ethernet networks into reliable interfaces for critical data forwarding. Consequently, it is this shift towards deterministic Ethernet that eventually led to the emergence of the deterministic technologies of time-sensitive networking (TSN) that are bound to supersede a large swath of the existing fieldbuses. This is a natural evolution as a result of this process. In fact, TSN can be thought of as a "one-size-fits-all" type of solution: since TSN can encapsulate different types of messages on top of standard VLAN-tagged Ethernet frames and provide a differentiated service for each one of them, it could also integrate the transmission of fieldbus messages alongside the rest of the flows in the network. This is further evidenced by the fact that many Ethernet-based fieldbus protocols have already defined different operational profiles to ensure their integration in a TSN system. That has been the case of EtherCAT or even Profinet, which have released the corresponding documentation [22, 23] with the methodology for their integration with TSN.

3.2 THE DRIVE FOR FULLY DETERMINISTIC ETHERNET

As mentioned in the introduction, TSN was developed with the intent of replacing the legacy fieldbuses that are commonly found in multiple applications, such as the industrial or the automotive engineering domains, with one single open, standardized, interoperable communication interface. Fieldbuses, such as *Profinet* [4] or *CAN* [3], can already do a great job for handling the data traffic needed for supporting critical processes in diverse use cases such as factory automation. Nonetheless, the myriad interfaces and protocols between devices, the varying requirements associated with the different types of equipment, and the usual lack of interoperability between proprietary interfaces may often result in a complex system landscape: a combination of fieldbuses may sometimes have to be deployed simultaneously within the same infrastructure to serve different devices or to operate different industrial controllers. In some cases, we may even find situations of analog cabling for operating legacy equipment in coexistence with fieldbuses for implementing other real-time control loops. In practice, all of these considerations often translate into a substantially large wiring harness, which could be difficult to maintain or service, and a control plane that may be hard to manage. This scenario was commonplace in industrial plants or automotive platforms, which normally required a complex integration of different interfaces, protocols, and subsystems to build an operational design.

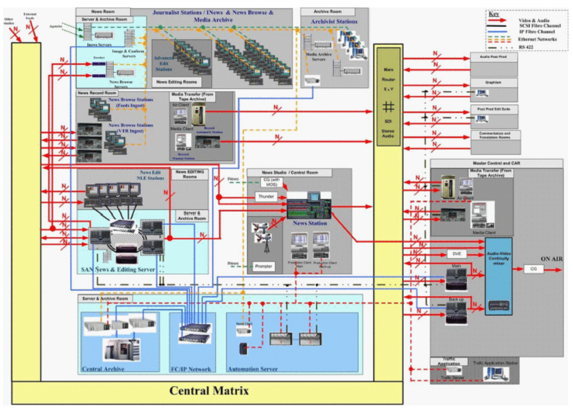
From the standpoint of system designers and integrators, it has always made sense to seek an alternative interface that could take on the role of the "one, single, and true" gold standard of interoperable interfaces. Nonetheless, the main hurdle for designating one specific protocol or interface for this task was the subject of interoperability. As a matter of fact, the designs and implementations of many fieldbuses are proprietary specifications and, while there may be other interfaces with an open specification (EtherCAT [2]), these latter solutions are usually only present in the devices, controllers, and product lineups from particular manufacturers rather than being a largely accepted, industrywide interface for control and automation. There may even be the case when different manufacturers feature competing fieldbus specifications in their devices, and hence the use of a bridging device may be in order to exchange data between the different bus domains of the network.

Ethernet networks could have been an answer to this conundrum, had it not been for their longstanding best-effort nature. In this context, there are numerous reasons to argue in favor of Ethernet to fulfill this role. Firstly, it is a well-known communication standard with nearly pervasive support across the major communication equipment. Its building blocks and specifications are openly available to the community, and hence most suppliers and manufacturers have the ability of building it into their product portfolio. Secondly, and most importantly, as it is one of the building blocks for the so-called "open world" communication technologies, like the Internet, it is also highly interoperable. Thus, even though there are several variants of Ethernet with different link speeds (1 Gb/s, 100 Mb/s, . . .), communication modes (e.g., *full-duplex*, *half-duplex*), or physical interfaces (twisted-pair copper cable, optical fiber), its specification is so comprehensive that it contemplates the use of "*autonegotiation*" schemes to ensure that Ethernet implementations on any given pair of nodes with different sets of features can agree on a common mode of operation for exchanging data with the highest possible attainable performance; i.e., any given pair of Ethernet interfaces respectively supporting 10 Mb/s and 10/100 Mb/s with full-duplex may agree to exchange data frames at the minimally common supported speed of 10 Mb/s. These are some of the features that have made Ethernet so successful for solving the broader problem of communications amongst computer systems: it is backward compatible with legacy versions of the standard, and it can accommodate the use of different communication modes or physical interfaces between devices. Furthermore, it does so in a manner that is consistent across different manufacturers and seamless for the end user.

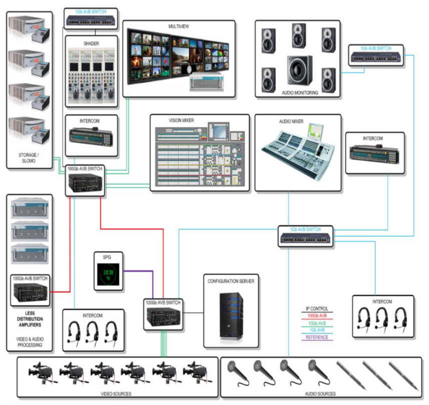
3.2.1 *A presentation of motivational use cases*

Consequently, it can easily be seen how these features allowed the emergence of Ethernet networks as one of the prevailing technologies in the networking landscape. Nonetheless, their use for solving the needs of industrial control scenarios was hampered by the fact that Ethernet networks rely on a best-effort paradigm. This implies that data should be forwarded to the destination as "quickly and efficiently" as the communication link would allow at a given time. Yet, there are no "implicit guarantees" for data delivery or that the user should come to expect a certain quality of service or a significantly prioritized treatment of the data flows [24]. This characteristic is inherent to the behavior and operation of Ethernet networks: Ethernet is a packet-switching system that routes data frames on a hop-by-hop basis using a variety of routing tools and protocols (e.g., ARP) to decide which ports of a given Ethernet bridge an Ethernet packet should be forwarded to. Hence, the bridging devices for Ethernet networks, which are also known generally as "switches", are internally built with interconnect devices using different methodologies, as can be seen in [25] or as outlined in Section 7.5 of this thesis project. These packet switching interconnects are thus meant to process data transactions on their input ports, arbitrate between them, and deliver them to the appropriate output port. Packet arbitration is a major source of packet delivery variation, and in some switching architectures it can lead to diverse effects, such as packet contention, head-of-line blocking, and, in the most severe of cases, a combination of the aforementioned effects may lead to congestion losses on the buffering elements of the input ports of the switch. These effects are all well-known, and the main strategy for circumventing them is to overprovision the capacity of the Ethernet system. This implies using higher speed links and larger packet-switching interconnects (e.g., larger buffering elements, more egress ports, . . .). However, even an overprovisioned system could still be ineffective for handling critical services, as we shall see next.

In the literature [24], this point is often backed by examining the use cases that drove the initial developments of TSN. Hence, the case of multimedia networks for professional audio/video (AV) production is described as paradigmatic. This situation is most commonly found in television studios or during the production of live content, such as the broadcast of a sport event or a concert, where the AV production team usually has to combine different audio and video sources (*feeds*) to produce a finished a program. This scenario involves multiple video sources, namely from different cameras for the audience, the event itself, the studio commentators, . . . Likewise, the corresponding audio feeds also have to be routed to a mixer and embedded alongside the video of the event to produce the finished content that will be broadcast to the viewers' homes. As seen in Fig. 3.2a from [26], the legacy AV production studio for television content usually consisted of different interfaces for each type of signal, including audio, video, or synchronization. Replacing this infrastructure with Ethernet posed several challenges (Fig. 3.2b). The first one was that the respective audio and video feeds in the studio are often generated in a raw, uncompressed format. As indicated in [26], the average studio generally uses up to 30 different cameras for sending high-definition (HD) content to the video switcher for video production. Uncompressed video can use up close to 2.97 Gb/s of the Ethernet link capacity [27], and hence the use of multiple video sources may imply the use of faster Ethernet specifications (e.g., 40 Gb/s, 100 Gb/s, . . .) to support this traffic; i.e., 1 Gb/s-Ethernet, which is one of the most popular implementations of the standard would likely be insufficient on its own. Moreover, synchronizing the reception of video flows can be difficult in a packet-switching network. Secondly, Ethernet systems, just like any other packet-switching technologies, are highly statistical in nature: there is a congestion loss probability as a function of the size of the ingress buffering elements and the traffic rate; and a latency variation with a large distribution. These effects can be observed in the qualitative illustration from [28] in Fig. 3.3.



a



b

Figure 3.2
One the first use cases of TSN/AVB for the transformation of the legacy AV production environments into a full-fledged IP-based system supported through Ethernet networking and bridging with determinism and robustness guarantees. Images reproduced and adapted from [27] and [26]: Fig. 3.2a is the legacy TV studio architecture, Fig. 3.2b is the upgraded environment with TSN/AVB bridging.

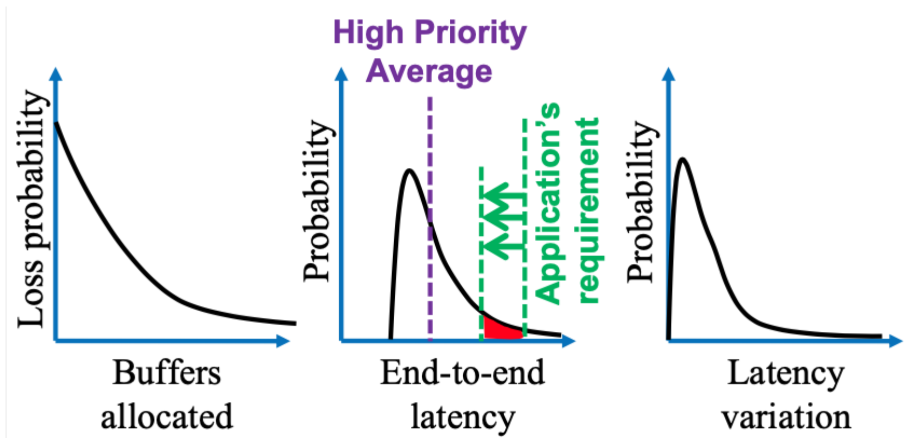


Figure 3.3
A quantitative illustration from the presentation in [28] to showcase the statistical, and hence best-effort nature, of ordinary Ethernet bridging.

In this context, even a relatively low packet loss probability can have a tremendous repercussion in the production of AV content: By means of a quick estimate, we can see how 30 video feeds at 2.97 Gb/s with a loss probability of 0.05% could produce a whopping ∼13.37 million lost frames (1500 B) within the expanse of one hour. This translates into 18.67 GB of lost data. These are missing or corrupt values for the pixels of the finished video that will be sent to the viewers. Hence, using standard Ethernet could produce an end product with degraded quality when it is compressed and prepared for broadcast.

A similar analogy could be drawn for an industrial automation scenario with potentially hundreds of nodes forwarding multiple classes of control, monitoring, and ordinary data flows. Admittedly, the demands for bandwidth consumption may not be as high for an industrial plant, which usually demands greater determinism and a lower PDV for its critical flows. Nonetheless, we could easily envision a situation with an analogously low probability for discarding packets, albeit with flows with lower speed, that could result in the eventual loss of a single frame on average during a working shift of the plant. If the lost packet was intended for a critical process, the industrial controllers of the plant may shut down automatically out of precaution and halt the assembly line. This downtime is unexpected, could take several hours to revert to normal operation, and leads to an impact in decreased productivity and larger production costs. These are some of the reasons why fieldbuses have traditionally been chosen as the preferred solution for industrial networks.

3.2.2 *A brief description of Audio/Video Bridging*

To solve these issues and, thus, provide a robust communication system, the IEEE residential Ethernet study group [29] advanced the specification of several addenda to improve Ethernet in this direction. Some key elements of this improvement were the realization that distributed synchronization and traffic shaping were essential requirements to establish a network that could handle multiple flows, guarantee latency variation within ∼1 μs, and avoid congestion losses altogether. This led to the initial description of Audio/Video Bridging (AVB), whose designation harks back to its initially intended application to professional multimedia environments. This was officially considered the predecessor to TSN before the residential Ethernet study group shifted its focus to industrial applications with the inception of the TSN working group (TSN WG) [30]. Thus, the main components of AVB are highlighted next.

- **The network synchronization component**, which is described in ***the 802.1AS specification*** [6]. As we will explain in the corresponding implementation sections (6.3.6) and in the TSN overview of our publications in Chapter 8 and 9, this component is tasked with instantiating a distributed time synchronization service using a particularized profile of the PTP synchronization protocol [31]: This profile designates nodes as either end systems or bridges with gPTP timing capability, bars the forwarding of synchronization messages over non-gPTP systems, and restricts the packet format to raw Ethernet frames (*"L2"*). Since one of the goals of distributed systems is to deploy applications (e.g., control loops) that can operate synchronously and in coordination with respect to a common time base, the use of gPTP serves this purpose by distributing a system-wide time reference which should be better than $\sim 1\ \mu s$ at least. This would fulfill the synchronization requirements of most industrial [32] and multimedia systems [27], although the implementation that we have introduced in this thesis project, which we characterize in Section 8.6.3.1, can narrow this margin down to the tens of nanoseconds. Furthermore, to increase the robustness of the timing distribution system, the standard also contemplates the use of the best master clock algorithm (BMCA) to automatically select the ports of the system nodes that will act as synchronization sources (*master*) or synchronization recipients (*slave*).
- **The traffic shaping mechanism** that was initially designated for AVB was ***the credit-based shaper (CBS)***, and its description can be found in the 802.1Qav specification [33]. The main goal of this shaping algorithm is to prevent the occurrence of congestion losses by evenly spacing out the packet bursts from the different traffic flows in the network. This is accomplished by assigning a certain amount of *credit* units to the different queues of the shaper, so that each queue is associated with a certain priority level and the amount of *credit* should be proportional to the flow bandwidth. As a result, when packets are injected into the shaper, the corresponding credit for each queue is *debited* (i.e., consumed) at a rate of *"activeSlope"* credits per amount of time. When the available credit is exhausted, transmission from its corresponding queue is halted until the credit has had a chance to regenerate at a rate of *"idleSlope"* credits per amount of time. As observed in the Fig. 3.4, which we have reproduced from [34], the effect of this mechanism is to space out data from different flows and thus prevent the buffering elements of concatenated bridges from becoming saturated. This method is a fundamental achievement for allowing the seamless coexistence of flows with different levels of criticality over the same physical Ethernet link.
- **Resource management** is another important part of AVB. This functionality is defined in the ***802.1Qat*** [35] specification, which details the operation of the stream reservation protocol (SRP). This protocol has the AVB emitter (talker) and receiver (listener) nodes work with a publisher/subscriber model. Hence, the talker devices announce their respective streams by issuing beacon frames that are propagated throughout the network to the listener devices. The AVB listeners would then

proceed to select the streams they would like to subscribe to; and then propagate this connection request backwards over the different hops of the network all the way to the talker device. This process is supported with the stream reservation protocol (SRP), which reserves all the necessary resources along the intermediate data forwarding bridges of the network. Hence, SRP has the AVB bridges advertise their internal message forwarding latency and their available resources, such as the available VLAN identifiers or the types of message priorities that the bridge is capable of handling. If a reservation of the node resources can be performed, the protocol will then move to next hop on the path to the talker to perform its corresponding resource reservation. Once the entire path has been covered, then the talker may begin exchanging its AVB flow with the listener. Moreover, the protocol description indicates that the path reservation for AVB flows can have two different levels of overall end-to-end latency: class "A" can guarantee 2 ms, whereas class "B" is more lenient and guarantees a latency of 10 ms.

- Lastly, the foregoing mechanisms are complemented with the specification of "**systems, configuration profiles, defaults**, protocols, and procedures for bridges and end stations" that are contemplated in the specification of 802.1BA [36]. Thus, the goal of this specification of the most usual configuration options and processes for an AVB system is to allow for the faster and more agile deployment of this type of networking systems. Hence, the approach taken here is to allow a non-networking expert to quickly build an AVB system using a set of commonly used blocks and configuration options with a "plug-and-play" type of approach.

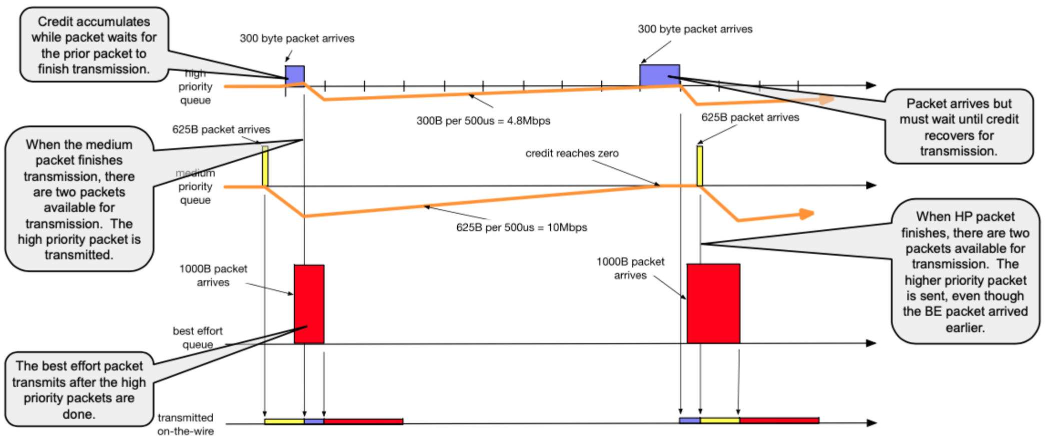

Credit accumulates while packet waits for the prior packet to finish transmission.
300 byte packet arrives
300 byte packet arrives
high priority queue
300B per 500us = 4.8Mbps
Packet arrives but must wait until credit recovers for transmission.
625B packet arrives
625B packet arrives
When the medium packet finishes transmission, there are two packets available for transmission. The high priority packet is transmitted.
credit reaches zero
medium priority queue
625B per 500us = 10Mbps
When HP packet finishes, there are two packets available for transmission. The higher priority packet is sent, even though the BE packet arrived earlier.
1000B packet arrives
1000B packet arrives
best effort queue
The best effort packet transmits after the high priority packets are done.
transmitted on-the-wire

Figure 3.4
An example of the operation of the credit-based shaper to space out traffic bursts and enforce a certain level of guaranteed bandwidth. Image reproduced from the presentation in [34] from the TSNA 2017 Conference.

3.3 OVERVIEW OF TSN AND ITS FUNCTIONALITIES

The model of AVB was effective for handling multiple flows in an Ethernet network and provide an entry-level type of determinism, namely with 1 ms and 10 ms PDV, and safeguards for guaranteeing their bandwidth with the CBS shaper. This approach was sufficient for the initial domain of AVB networks in professional AV production environments. Nonetheless, the concepts at its foundation of bounded latency and determinism could also be applied to upgrading the systems deployed for factory automation; although these use cases called for several changes in the original paradigm for handling traffic in AVB networks. This led to the emergence of time-sensitive networking (TSN), which we introduce in the following lines.

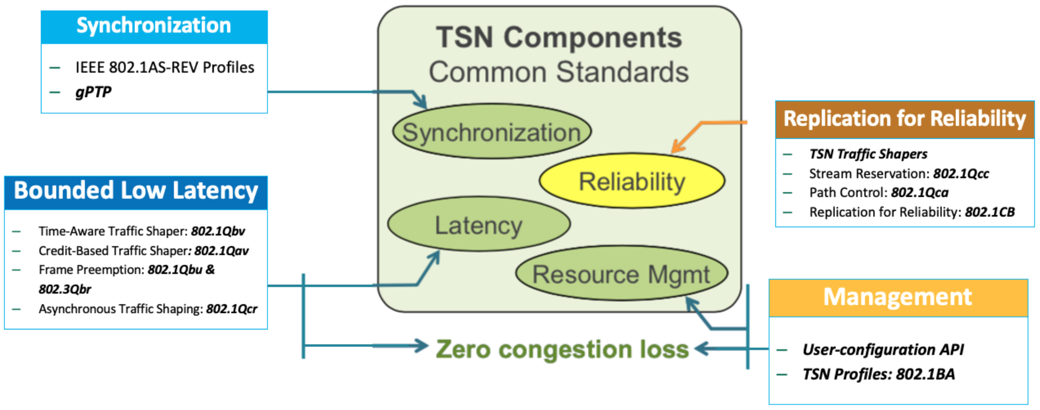

Synchronization
– IEEE 802.1AS-REV Profiles
– gPTP
Bounded Low Latency
– Time-Aware Traffic Shaper: 802.1Qbv
– Credit-Based Traffic Shaper: 802.1Qav
– Frame Preemption: 802.1Qbu & 802.3Qbr
– Asynchronous Traffic Shaping: 802.1Qcr
TSN Components
Common Standards
Synchronization
Reliability
Latency
Resource Mgmt
Zero congestion loss
Replication for Reliability
– TSN Traffic Shapers
– Stream Reservation: 802.1Qcc
– Path Control: 802.1Qca
– Replication for Reliability: 802.1CB
Management
– User-configuration API
– TSN Profiles: 802.1BA

Figure 3.5
TSN is built around four main operational pillars with interchangeable subcomponents. Different profiles may include a subset of this specification. Adapted from [37].

The development and definition of TSN subsequently moved to the IEEE TSN working group, which had a focus on aggregating multiple flows over the same Ethernet links with a greater degree of determinism than can be afforded with an AVB system. This represents a shift in the paradigm of AVB, since the ultimate goal is to aggregate as many flows as possible over the system with bounded latency and negligible congestion loss. This is the ideal scenario for factory and assembly line automation, which were the use cases that drove a substantial part of the development of TSN. Furthermore, TSN aims to be a far more comprehensive solution than AVB, with greater versatility, configuration, and implementation options. Hence, it is built around four main foundations, as seen in Fig. 3.5, that offer different components that implementers may select for building TSN applications tailored to different scenarios.

– **The timing system** is still based on the gPTP protocol, albeit the advent of TSN also saw the definition of an updated version thereof: 802.1AS-REV [38]. This version of gPTP aims to offer greater synchronization accuracy down to the nanosecond level with a high-accuracy mode. This is achieved by mandating that all gPTP instances in the network be syntonized, i.e., use a common frequency, for measuring time intervals and estimating offset values. As before, this protocol uses raw Ethernet encapsulation for transmitting its messages and restricts their forwarding to strictly within network subsystems made up of gPTP-enabled bridges. Furthermore, the use of the BMCA increases its robustness by adding support for redundant ring topologies that could theoretically be able to synchronize tasks across different nodes with the type of accuracy expected in the most demanding industrial applications. Lastly, the protocol also includes support for multiple timing domains and defines generic interfaces for building time-aware applications.

– **Forwarding determinism**. Another fundamental avenue of TSN, as seen in the picture, is to provide mechanisms that **guarantee a certain level of bounded end-to-end latency**. This is essential for many real-time processes, which rely on the timely reception of data within a well-defined deadline. The network environment of most industrial plants requires that multiple moderate-bandwidth flows should be aggregated over shared infrastructure with different types of topologies (e.g., linear bus chain, ring or tree layouts, . . .). As indicated in [39], it is common for industrial networks to process flows on the order of several megabits per second (∼1 Mb/s for legacy fieldbuses) and they usually enforce processing cycles on the order of ∼1 ms or less. Moreover, these systems usually involved a large quantity of nodes. This was an important limitation for AVB systems, which had been conceived for assembling networks with a smaller scope of up to seven hops. Hence, TSN set out to improve this by defining new traffic-shaping methodologies and novel strategies for reducing the packet-processing latency on a TSN bridge. The former implied the definition of a new traffic shaper based on a time-division multiplexing methodology that is driven by the synchronized gPTP time: the time-aware traffic shaper (TAS - 802.1Qbv) [9], which we explain in great detail in Chapter 7. This approach is more effective than the CBS for enforcing a strict cycle time and reduce the delivery jitter. The latter aspect of the new TSN systems, i.e., reducing the internal processing time, could be achieved with the enhancements for frame preemption for the Ethernet MAC service and for the TAS shaper (802.3br [40] and 802.1Qbu [41]), whose implementation we have also undertaken (Chapter 7). Frame preemption can effectively provide a low-latency service for the transmission of highly critical traffic, which can be beneficial in large networks. In addition, the specification of TSN also defines alternative traffic shapers [34], such as the asynchronous traffic shaper (802.1Qcr) [42], which can be used when determinism constraints are more relaxed – it does not require a working gPTP service – and operates following a scheme that resembles that of the CBS shaper.

– **System Robustness**. Given the use of TSN systems for transmitting and processing highly critical data, the role of the shapers for guaranteeing a timely delivery is complemented with several mechanisms for ensuring that critical traffic will be delivered at its expected recipients. Consequently, the joint action of the shapers and the mechanisms for system robustness ensure that the possibility of congestion losses can be minimized: not only can we guarantee that data delivery is bounded with this two-tier approach, but we can also ensure that if frames are dropped at the intermediate nodes for buffering congestion or link failure, then a copy of the message could still be received at the destination. We can achieve this with the redundancy methodologies defined for TSN systems. Thus, since TSN is based on an Ethernet foundation, some well-known methodologies for redundancy such as HSR or PRP [43] could be built alongside the implementation for TSN. Nonetheless, the TSN specification has come up with its own native seamless redundancy service for critical data: the 802.1CB [44] specification. This component allows the definition of multiple redundant streams that can be forwarded over the network by establishing a redundant, ring-style topology. It defines several methods for identifying the traffic classes that will be transmitted over redundant paths, and it implements a filtering function for discarding duplicates based on the use of a sliding window. We show a detailed implementation of this protocol for this thesis project in Chapter 7. In addition, the standards for TSN define complementary mechanisms for enhancing data transmission reliability, such as the use of *ingress filtering* (802.1Qci [45]). This allows the constant monitoring on the ingress paths of the TSN nodes to ensure that the TSN messages are received within their time slots. Otherwise, they can be discarded to prevent them from "clogging" system resources that might be reserved for a different traffic class during the rest of the cycle time.

– **Resource management**. The multiple components that make up a TSN system require the application of a specific, user-driven configuration dependent on the network scenario to achieve different effects, such as the establishment of a well-defined deadline for a high-priority flow. In practice, given the modular nature of TSN, this requires the combination of the settings applied to the traffic shapers, the traffic identification modules (802.1Q [46]), the timing system, ... TSN systems address this problem with the configuration model specified in the 802.1Qcc standard [47]. In our experience, the initial approach for configuration in TSN was to upload settings on a node-by-node basis using some type of interface (e.g., a web server) for supplying parameters. Some early commercial implementations from *Innovasic* [48] and others featured this approach. The specification has since evolved towards a centralized configuration system with a user-friendly interface (e.g., a GUI, a web page, ...) that network implementers can use for defining high-level system parameters, as seen in Fig. 3.6. This configuration specified in the centralized user configuration (CUC) component is then sent to an analytics engine – the central network configurator (CNC) – which derives the settings that should be applied to each individual node of the system. To do this, the CNC maintains a system model for each node in accordance with the YANG models for TSN (802.1Qcw, 802.1CBcv, ...) and connects to each node in the system to upload their corresponding parameters. This trend of centralized, automated configuration will allow the fast deployment of TSN networks with an almost "plug-and-play" feel. Parts of the specification of the YANG models for TSN are still a work in progress, but there are some successful commercial implementations to date, such as the *Slate* configuration utility from TTTech [49].

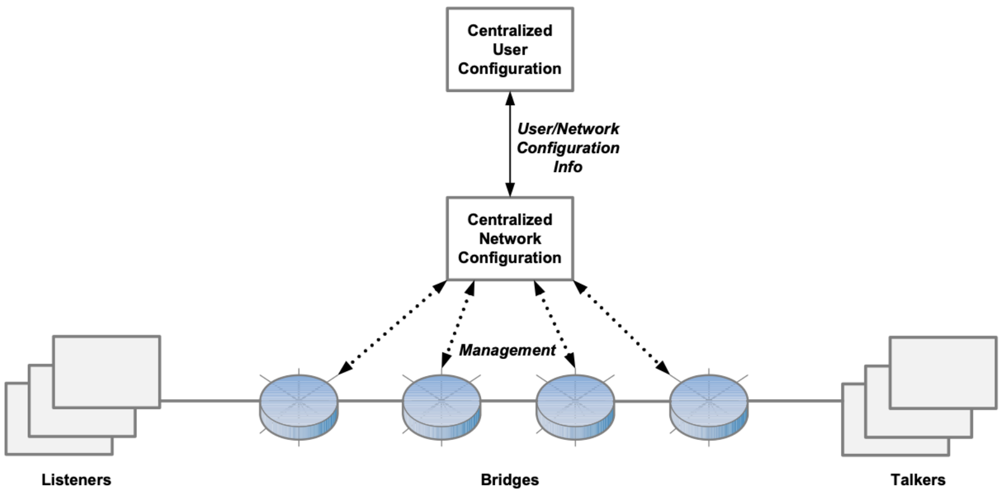

Centralized User Configuration
User/Network Configuration Info
Centralized Network Configuration
Management
Listeners
Bridges
Talkers

Figure 3.6
The TSN centralized configuration model and network architecture hierarchy outlined in the IEEE 802.1Qcc specification. Image reproduced from [47].

It can be seen that TSN is highly modular as it can combine different elements from each one of its foundational *pillars* to produce distinct flavors of TSN systems. As a matter of fact, many of these elements have already been included in the newer IEEE specification for bridged networks: 802.1Q-2018 [46]. This diversity of options is beneficial to cover a wide range of applications but may make some questions about interoperability arise. Hence, special emphasis is usually placed on the issue of the interoperability between the implementations of TSN from different manufacturers. This encompasses a broad range of considerations: from the underlying Ethernet communications interfaces and physical layer transceivers, to the interoperability between the modules and design solutions supplied from different implementers. An example of this would be to ensure that the frame preemption modules from different manufacturers can exchange data (including fragmented Ethernet frames) seamlessly. Another area of concern would be to ensure compatibility between the main timing stacks applicable to TSN, such as the "ptp4l" [50], which has been adapted to feature a gPTP profile in some commercial products (e.g., the KONTRON TSN kit [51]), or the AVNU gPTP implementation [52]. To this end, one of the leading industry associations driving the development of TSN – the AVNU Alliance [53] – has contributed to setting up interoperability laboratories and industry-wide test benches, such as that at the University of New Hampshire [24], where manufacturers and implementers can test and verify the interoperability of their solutions against those from other makers with a comprehensive testing scheme.

3.4 AN INTRODUCTION TO TSN PROFILES

As we mentioned in the foregoing points, TSN is a highly modular technology. Yet, as of today, there has not been a commercial implementation so far to feature all of its components integrated in one single design. This is true both for ASIC-based designs, which cannot be altered after their implementation, and for FPGA designs, which are normally built to fulfill one specific case.

This is a major point of focus for the industry, as evidenced by the discussions in this regard in some of the main industrial forums driving the development of TSN [54]. Thus, TSN can support use cases in a diverse range of applications: from the next-generation automotive networks through to the well-known use cases of industrial automation. Each case has different peculiarities, constraints, and requirements that can be satisfied through specific combinations of subsets of the components of TSN. In practice, given the original applications of TSN, the definition of several profiles for handling the typical use cases of the automotive, factory automation, or multimedia is currently underway. Many of these profiles are already partially defined as part of a number of standardization projects from the IEEE, whereas there is ongoing work for other new cutting-edge applications such as those supporting aerospace systems or 5G networks. We summarize some of the main profiles TSN, and those still under definition, in the points below.

- **Professional AV and multimedia**. The profiles for Audio/Video bridging, which originate from the initial specification of AVB. These are documented in the 802.1BA specification [36], which we mentioned previously.
- **Industrial automation**. The use of TSN for supporting industrial automation and the transition to the so-called "Industry 4.0" is considered in the IEC/IEEE 60802 standard [55]. It is currently an ongoing development that contemplates

the use of the main features of TSN in some of its early proposals [56]: gPTP timing with the 802.1AS specification, traffic shaping with preemption (802.1Qbv, 802.1Qbu, 802.3br), ultra-low latency cut-through modes for select traffic flows, and a centralized network configuration system as outlined in 802.1Qcc. Furthermore, to increase the robustness of these systems, the use of the mechanisms for redundant clock synchronization and the BMCA are also considered [57].

- **Automotive networks**. There is also an ongoing project for standardizing an automotive profile for TSN networks. As explained in [58], automotive networks have several key different requirements from those of industrial networks. One of them is related to their physical layer interface (PHY), which has to be able to withstand larger amounts of electromagnetic interference and is usually built to operate at 100 Mb/s or less over single twisted pair [59]. From the standpoint of the functionalities of TSN systems, automotive networks require that data be sent deterministically and reliably. The former can be achieved with the use of gPTP timing with the BMCA and redundant clock synchronization to drive the TAS shaper (802.1Qbv) with frame preemption (802.1Qbu & 802.3br). In addition, the profile would also allow for the use of the asynchronous traffic shaper (ATS - 802.1Qcr) for sending high-priority flows without having to rely on the execution of a complementary timing service. The latter point of reliability can be achieved with the use of the TSN enhancements for seamless redundancy (802.1CB) and the ingress policing mechanism (802.1Qci), which guards against TSN flows exceeding their allocated transmission windows and, thus, prevents them from clashing with higher priority traffic and cannibalizing the resources assigned to other flows. Lastly, given the relatively reduced size of the automotive networks, the use of a centralized configuration scheme as outlined in 802.1Qcc could be considered optional, as it is expected that the topology of these networks will be fixed by design and that no additional nodes will be added dynamically into the system. Hence, a static system-wide configuration whereby each node is preprogrammed with its own settings could be preferrable in these cases.

- **Mobile fronthaul**. TSN could also be used for supporting the deployment of the next generation 5G networks by implementing the data transport interface of its radio access network (RAN). As a result, there is already a standardized specification of a TSN profile for the fronthaul of the RAN for 5G: the IEEE Std 802.1CM profile [60]. Specifically, the standard is concerned with specifying interfaces for providing connectivity between the functional blocks of the base station (BS) in a cellular network, such as the radio equipment (RE), the radio control equipment (REC), or their corresponding eCPRI [61] versions: eRE and eREC, respectively. These interfaces allow the BS to be physically partitioned to provide flexible implementations of its architecture and even split its functionalities over different blocks. This degree of flexibility is often demanded for the deployment of mobile networks and TSN can be put to use to this extent with its new fronthaul profile. Hence, the specification indicates that the 5G fronthaul could be supported with the use of 802.1AS for timing synchronization, traffic shaping (802.1Qbv) with preemption (802.1Qbu) for ultra-low latency over Ethernet links of up to 10 Gb/s, and traffic identification blocks (802.1Q) for handling a minimum of up to three traffic classes.

- **Aerospace systems**. The networks for supporting the communications of the critical systems found in avionics and aerospace applications could also be built by applying a corresponding TSN profile. This would imply the use of a deterministic Ethernet-based network to replace the typical fieldbuses that are normally used for this role in aerospace systems, such as Spacewire [62] or the MIL-STD-1553B [18]. There is a growing interest in the development of this profile, as indicated in leading industry forums [63]. In fact, there is a joint standardization effort currently underway between the IEEE and SAE [64] to define a working profile for aerospace, as mentioned in [65]. In this context, we have made our own proposal during one of the studies that we have presented in this thesis project (Chapter 9). Hence, in this study we developed the avionics system of a sounding rocket with an architecture based on the 802.1AS timing with the BMCA, TAS traffic shaping (802.1Qbv) with preemption (802.1Qbu & 802.3br), and seamless redundancy (802.1CB).

These profiles correspond to the main areas of application and potential deployment of TSN systems, and hence the drive from the IEEE TSN WG [65] and other standardization committees to provide accurate descriptions of their use cases and their accompanying TSN features. By way of a summary, we also provide a comparison of the features expected to support each profile in Table 3.1.

Table 3.1
A short comparison of the TSN features expected for supporting its main profiles.

Comparison of TSN profile features

	802.1AS	Traffic Shaper	Preemption	Cut-through	Redundancy	Traffic Identification	Central configuration
802.1BA	Yes	CBS	-	-	-	Yes	-
Industrial Automation	Yes (BMCA / Redundant)	TAS (if < 1 Gb/s)	Yes	Yes	Yes (802.1CB)	Yes	Yes (802.1Qcc)
Automotive networks	Yes (BMCA / Redundant)	TAS, ATS	Yes	-	Yes (802.1CB)	Yes	Optional (802.1Qcc)
Mobile fronthaul	Yes	TAS	Yes	-	-	Yes	-
Aerospace networks*	Yes (BMCA)	TAS	Yes	-	Yes (802.1CB)	Yes	-

**Our aerospace proposal from Chapter 9.*

3.5 COMMENTS ON CURRENT COMMERCIAL SOLUTIONS

With the specification of TSN systems well advanced into a stable standard, we have seen multiple manufacturers implement and deploy their TSN-enabled products for their application in different domains. Many of these implementers come from a variety of sectors, although they usually have a background in the development of solutions for industrial automation or automotive networks, such as the legacy fieldbuses.

With the advent of TSN, many of these suppliers have switched to the production of TSN-based control solutions for multiple application domains. At a glance, some of the leading providers in this area are companies such as TTTech, NXP, Bosch, Ericsson [66], Analog Devices, KONTRON, ... It can be seen that they are all relative influential institutions, with a solid background in the areas of communications, industrial control, and automation that also lead in the development of innovative solutions. Indeed, many of them are driving the process of the widespread adoption, standardization, and definition of the main features that TSN should include in order to replace the legacy fieldbuses. This has been evidenced by their growing participation and involvement in major industrial forums and consortia, such as the Industrial Internet Consortium [67] or the AVNU Alliance [53]. Hence, besides the IEEE TSN WG itself, these are the main venues that the most active participants in the development of the standard have for defining joint test beds, assess interoperability, or evaluate the overall evolution and cutting-edge developments in the field. In this regard, the yearly TSN/A [54] conference series can be used as a main reference of the recent developments and pressing topics in the evolution of the TSN standards.

Furthermore, in order to get a glimpse of the current state of affairs on the industrial front, we have prepared a short review of some of the most important products and solutions from leading TSN suppliers. This review of their general TSN offering can be helpful to better understand the level of maturity of the technology and some of its main areas of application.

– **TTTech** [68] is an Austrian supplier of communication and control equipment. They started out with an implementation of SAE AS6802 specification of the time-triggered Ethernet protocol, which could be considered an Ethernet-based fieldbus. Later on, they transitioned to TSN products, which they provide in the form of custom IP cores for FPGA or as standalone boards or nodes. Examples of the latter could be their TSN starting kit [69] or their network edge processing node for building OPC-UA systems [70]. Moreover, they have successfully applied the technology to multiple cases in the automotive, industrial, or the aerospace domains. In the case of aerospace, they have well-documented applications of Ethernet-based interfaces to support the communications of space vehicles (Ariane [71]) with TTEthernet or AFDX. Thus, given their track record in the development of Ethernet-based fieldbuses, they have already transitioned to TSN and with products such as TSN switches, TSN-capable edge nodes, or automated configuration tools for uploading operation parameters to all the elements in a TSN network automatically. That was the case of their Slate configuration utility [49].

– **NXP semiconductors** [72] is another main manufacturer of industrial control and networking equipment. Among their product offerings, they provide a TSN reference board (the *Layerscape reference board*). The board is intended to be used as a development platform for the IoT and comes with a five-port TSN switch with support for some of the main elements of the standard: 802.1AS timing, time-aware traffic-shaping (802.1Qbv), credit-based shaping (802.1Qav), and ingress filtering (802.1Qci). They also support an early integration with the YANG device models to assist in the automated configuration of the system.

– Other providers with TSN products are **National Instruments** or **Analog Devices**. The former offers TSN-enabled PLCs with its "*CompactRIO*" [73] solution. These are programmable controllers that can be configured through the LabView environment for industrial plant automation; and they feature TSN communication. **Analog Devices** also offers its own set of TSN products, such as an interesting option of a TSN starting kit [48].

- **Kontron** is another interesting option for procuring TSN equipment. One of the main lines of business of the company lies in the production and supply of industrial computers; namely the type of industrial PCs that often feature an expansion rack for adding expansion control modules. Hence, **Kontron** offers an industry-leading TSN starting kit consisting of one such industrial PC that is fitted with a PCI expansion card [51]. The expansion card comes with a four-port TSN switch featuring 802.1AS timing, traffic-shaping with preemption (802.1Qbv, 802.1Qbu, 802.3br), 802.1Qcc resource reservation, and 802.1CB seamless redundancy (pending a future upgrade).

All of this is to indicate that TSN is a mature solution, currently on its way to becoming widespread in the industry. This is evidenced by its support from a majority of influential manufacturers, industry-wide associations, and standardization bodies such as the IEEE, which are the main drivers behind its adoption.

4

THE MATERIALS AND METHODS

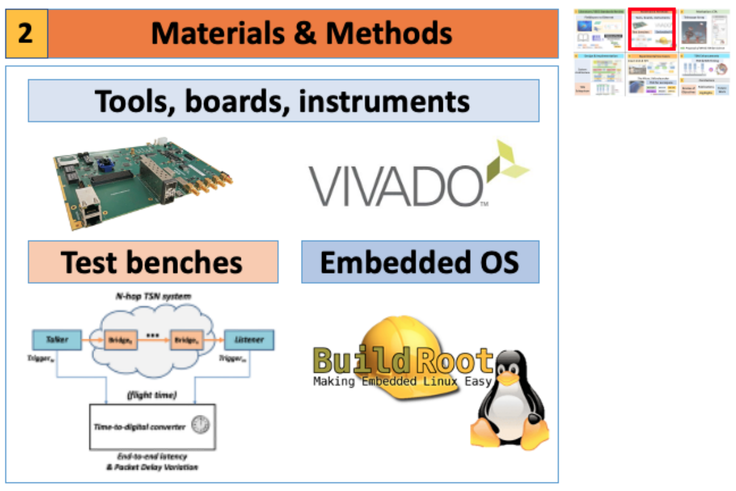


Figure 4.1
Overview of the contents of Chapter 4 for the materials and methods.

This chapter introduces the materials and experimental methodologies that we used throughout the thesis project to derive and support our results. Since our goal was to build functioning TSN systems that could also be combined with the highly precise White Rabbit synchronization, we start off introducing the WR-capable boards that we used for a substantial part of our prototyping: the WR-ZEN boards. Next, we present the tools that we used for developing, validating, and debugging these systems: embedded OSes and the Vivado development environment for building our FPGA modules. We also used multiple types of benchmarking utilities, simulation tools, and network traffic inspectors such as *Wireshark*. Afterwards, we introduce the laboratory environment that we have used for building the different test benches required for carrying out our experiments and extracting their performance data. Lastly, we discuss the key experimental parameters and performance indicators that we use for characterizing our designs. In this context, we also provide the reader with additional background on the significance of the indicators and the corresponding experimental process for their derivation.

Chapter contents

We describe the main materials and methodologies used for this project in the following sections. We include clarifications and references to the key and underlying methodologies that we have used for substantiating the results of our studies.

4.1 THE WR-ZEN AND OTHER WR-CAPABLE BOARDS

The WR "Zynq-embedded node" (WR-ZEN) has been one of the main components used throughout this study. Hence, the WR-ZEN [8] has been used as a development, testing, and integration platform during the prototyping and design stages of the different elements that make up the TSN systems presented in this dissertation.

As implied by its designation, the WR-ZEN board is a time-keeping device intended to distribute highly accurate synchronization using the White Rabbit protocol [74, 75], which combines *syncE* frequency distribution [76], several enhancements for PTP [77], and precise phase measurement techniques with digital dual mixer time-difference (DDMTD) detectors [78] to synchronize network nodes with sub-nanosecond accuracy and disseminate the reference clock of the WR master device across the network. This protocol was initially devised for the time and frequency transfer systems in use for some of the experiments at CERN and has since been implemented in a variety of nodes for different projects and scientific facilities, such as the SPEC board [79] or the detectors of the KM3NET [80] neutrino telescope. The WR-ZEN is another variant that has been designed by Seven Solutions S.L. that features an architecture with enhanced clocking circuitry and the hallmark use of one of the low-cost versions of the Zynq-7000 programmable SoC: the Z-7015 device [7]. This device bundles together a double-core ARM processor and programmable FPGA logic. In the Xilinx-coined terminology, the former is referred to as the "processing system" (PS), whereas the latter is called the "programmable logic" (PL). This type of architecture allows for the implementation of complex designs that make use of the PS for running an embedded operating system (OS) with utilities that can be accelerated with specific IP cores on the PL, as can be observed in the diagram from Fig. 4.2 provided by Xilinx.

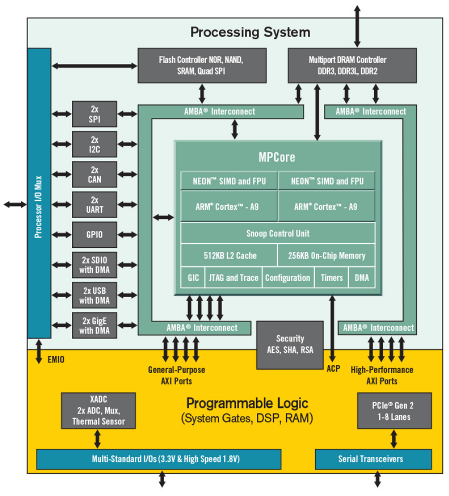


Figure 4.2
The block diagram of the Zynq-7000 programmable SoC devices from Xilinx. Image reproduced from [81].

For the case of the White Rabbit timing, this technology was used for building a versatile stand-alone device: The entire WR timing system alongside the Ethernet networking interfaces and their components (e.g., PHYs, time-stamping units, . . .) were instantiated on the PL FPGA, whereas the processor was used for running an embedded Linux OS that managed the WR implementation with the use of custom Linux drivers, system services, and other configuration utilities.

Hence, unlike the case of other WR-compatible boards like the SPEC node, which does not include a built-in embedded processor, the use of a Zynq-7000 device allows for faster prototyping, tighter integration with the FPGA firmware, the more agile inclusion of new software features, and greater software execution performance. Furthermore, its design managed to fit an entire, dual-port implementation of a WR timing system into the low-cost, reduced-size Z-7015 SoC, even leaving some remaining room for integrating other third-party IP cores should new features be needed. As a result, given the flexibility afforded by this platform, we selected the WR-ZEN (Fig. 4.3) as the development environment where many of the new IP cores and features of the TSN systems presented in this study will be verified and tested.

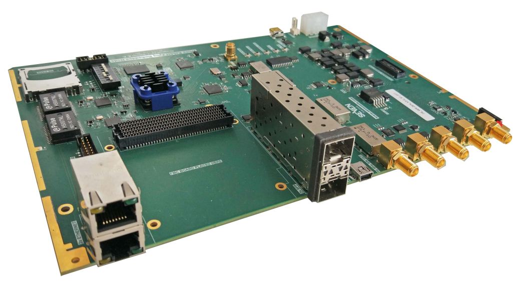


Figure 4.3
A picture of the WR-ZEN board used as the main prototyping and development environment for this thesis project.

4.2 THE DEVELOPMENT ENVIRONMENT: XILINX ZYNQ-7000 SOCS AND FPGAS

As indicated previously, the development platform of this thesis has made a consistent use of the Zynq-7000-based WR-ZEN board, which features a programmable SoC with a hardened ARM microprocessor coupled with FPGA logic. Correspondingly, the development environment that had to be used for this platform was a combination of the Vivado Design suite [82] for IP core and general FPGA development, embedded Linux build tools (e.g., cross compilers, general C programming, buildroot, . . .), and the development of Linux drivers and hardware-monitoring utilities.

a) The Vivado Design suite is the comprehensive design environment for the different FPGA and SoC devices from Xilinx, with some editions allowing for high-level synthesis. It implements a top-down design methodology whereby the user starts off specifying a high-level block diagram of the system architecture and then gradually works its way down to the definition of specific IP cores, which are implemented using hardware description languages (HDL) such as VHDL or Verilog. Furthermore, the Vivado Design suite, which can be seen as a collection of tools for FPGA development, comes with additional utilities that are useful for designing and debugging IP cores and FPGA designs in general. These tools include a built-in simulator for implementing FPGA test benches or interfaces with debug cores (ILA, VIO) for real-time on-board debugging. Although this development suite is built on top of a powerful scripting language (*tcl*) for finer control of the design process which also allows the orchestration over the command line of more complex design tasks.

b) The Buildroot [83] utility is a very well-known tool for building embedded systems, as it conveniently generates and packages all the necessary elements for deploying a working embedded system with the use of a cross-compilation toolchain. Hence, this tool is run on a host, desktop computing system – a PC – to generate an embedded Linux image with its appropriate device tree driver binaries, a root file system image, and the corresponding bootloader for the targeted embedded platform. Buildroot can thus assist in the generation of highly customized embedded designs: the kernel settings, including its built-in modules, the type of processor support, and other parameterizations can all be adjusted to conform to the type of hardware and software environment envisioned by the designer. It also includes predefined configuration templates for well-known platforms such as the Zynq-7000 devices, which help produce working designs faster. Furthermore, it comes with a build system that allows the user to easily customize the contents and utilities that are built into the file system image. These utilities can be user-selectable packages from a content repository, such as *Busybox* [84] for the usual command-line tools of Unix-like systems, miscellaneous applications like an open secure shell (SSH) server, or even third-party or custom libraries and utilities that the user can easily integrate into the build system.

c) Lastly, on several occasions, the implementation of some of the TSN designs presented in this study will require the development of specific kernel-level drivers or the rollout of certain kernel patches for enabling the operation of new functionalities. Even though this is not admittedly the main focus of our research, knowledge and management of kernel-level functions is a necessity for building custom embedded environments. Hence, we have found and followed a useful reference on the operation of the main kernel programming interfaces for the different types of devices in [85].

4.3 THE LABORATORY INSTRUMENTATION

The IP cores and functionalities implemented for this thesis project have been tested and verified at the Time and Frequency Transfer Laboratory of the University of Granada (EQC2018-005214-P). This laboratory is fitted with all the necessary equipment for testing the validity of the TSN systems supplied in this project, as well as for assessing their overall performance and attainable determinism. Thus, the equipment available for testing and debugging our system ranges from standard networking equipment, multiple measurement instruments such as oscilloscopes and frequency counters, through to protocol analysis tools. The foregoing elements are introduced in the points below.

4.3.1 *The networking equipment*

The networking equipment encompasses all the elements required for assembling a fully functional Ethernet network. As will be introduced in Chapter 6, the enhancements brought forward by TSN networking are largely independent of the underlying physical layer of the Ethernet network system where they are used. Moreover, some of the solutions and use cases that are presented in Chapters 5, 8, 9, or 10 make indistinctive use of both twisted-pair copper RJ-45 wiring or optical fiber connections. Hence, our networking laboratory is equipped with all the material needed for establishing 1-Gb/s Ethernet networks using the 1000Base-T and 1000Base-X specifications. To this end, we have used the following elements.

- Network wiring will be either the standard twisted-pair copper wiring with RJ-45-type connectors or single-mode fibers (SMF) for making 1-Gb/s Ethernet network deployments. However, the experiments of Chapter 9 make use of a specific ruggedized connector for avionics applications that is based on the twisted-pair copper interface of 1000Base-T Ethernet.
- The Ethernet interfaces of the WR-ZEN board feature two so-called "cages" for small form-factor pluggable (SFP) connectors. These SFPs operate as adaptors between the Ethernet transceivers (PHY) of the WR-ZEN board and the appropriate physical interface of the network. Since our implementation of White Rabbit is supported with SMF fiber networks, the laboratory will be fitted with both optical fiber [86] and twisted-pair copper-based SFPs [87]. The former will allow us to interface between the PHY of the WR-ZEN and the optical fiber network required for supporting WR timing, whereas the latter will be used for connecting to 1000Base-T equipment without WR support for other testing and debugging purposes.

4.3.2 *The measurement equipment*

The measurement equipment available in the laboratory features general-purpose oscilloscopes, frequency counters, and FPGA debugging equipment. Hence, the oscilloscopes and the frequency counters [88] will be used for assessing the overall timing and determinism performance of the system by examining the behavior of some key analog signals, such as its clock sources or the pulse-per-second (PPS) monitoring output of the WR-ZEN board or that of the prototype TSN nodes. The PPS output produces a short analog trigger once per second and is a convenient way of estimating timing stability when combined with frequency counters, which are often dubbed "*time-to-digital converters*" and are capable of gathering large data sets for further study, as will be explained later on in Section 4.4.

4.3.3 *The FPGA debugging equipment*

The FPGA debugging equipment has consisted of the Xilinx programming and debugging cables capable of interfacing with the FPGA design over the JTAG port [89] for carrying out real-time on-board testing and debugging. This is done in combination with a Vivado instance running on a host PC or a Vivado debug server that directs the JTAG programmer to interface with the appropriate debug IP cores (i.e., ILA, VIO) instantiated on the FPGA for carrying out user-specified tests.

4.3.4 *Other laboratory equipment*

Other equipment available in the laboratory has also included highly precise oscillators, such the oven-controlled Morion oscillator for providing a stable clock source [90], multiple power supplies, or signal generators [91] for producing multiple waveforms or for injecting analog triggers into prototype TSN nodes for driving the generation of critical messages.

4.3.5 *Network inspection tools*

The structure of the TSN data units and the TSN messages themselves have also been examined using generic network sniffing/inspection tools, such as Wireshark [92] or *tcpdump* [93]. These utilities allow straightforward examination of the contents of the TSN streams forwarded with the prototypes devised in this study, as well the estimation of the available bandwidth, or the calculation packet loss ratio. The traffic pattern expected from the application of a certain scheduling policy at the TSN traffic shapers can also be examined in detail using the enhanced analysis features of Wireshark.

4.3.6 *Ethernet benchmarking utilities*

The highest attainable forwarding performance of the TSN-capable Ethernet interfaces was also verified with generic networking benchmark tests, such as those provided by the *iperf* [94] tool.

4.4 THE EXPERIMENTAL AND DESIGN METHODOLOGY

In this study, we start with a proper theoretical analysis and literature review where the different needs for deterministic networks are evaluated. Then, a technical implementation is addressed based on reconfigurable hardware devices using HDL languages, embedded firmware OSes, and generic programming languages (e.g., C, shell scripting). Proper experimental setups using advanced instrumentation and networking tools are used to validate the performance of the developed solutions. The following sections will present the required explanation and all the clarifications for understanding the specific methodologies that we used to support our general approach.

4.4.1 *The general approach for TSN devices*

The TSN nodes prototyped in this study will be verified and tested against four main performance parameters: the timing transfer stability and its accuracy, the end-to-end latency and the associated packet delay variation (PDV), the attainable throughput for forwarding TSN streams, and the protocol and message delivery integrity. These parameters will thus be used for carrying out the experimental characterization of the system by means of the different indicators and measurement methodologies outlined next. Interoperability can be verified by connecting to other TSN devices from other manufacturers; such as National Instruments or Innovasic (now Analog Devices). A timing-evaluation platform such as Paragon-X [95] could be used to verify the interoperability of our gPTP implementation, although this latter aspect is still ongoing work and will probably be presented as a future result after this thesis project.

4.4.2 *Timing transfer accuracy*

The timing transfer accuracy is studied by monitoring the respective behavior of the clocks of the timing master and slave devices in the network. The goal here is to assess the overall quality of the synchronization and measure the influence of effects such as phase noise, which is one of the main concerns when developing a timing transfer mechanism. This is achieved by deploying a measurement setup like that shown in Fig. 4.4.

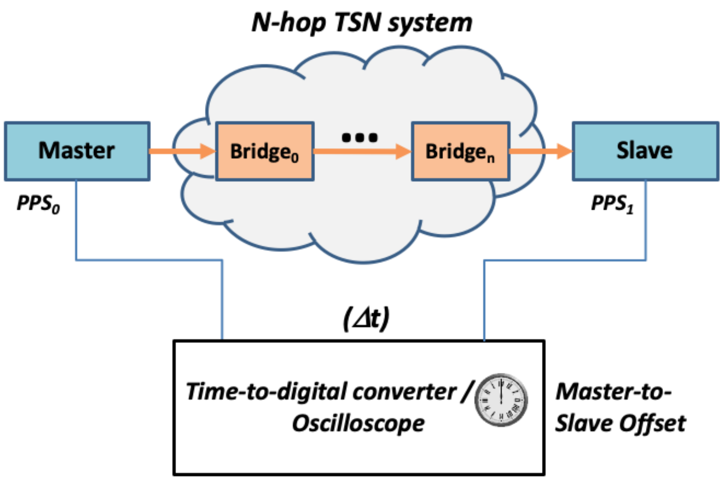


Figure 4.4
Schematic diagram of the procedure for assessing timing transfer stability and performance.

As shown in the figure, the process for assessing the timing transfer accuracy will consist of the monitoring of the corresponding PPS outputs of the timing master and slave devices in the networks. The PPS output is a convenient interface for keeping track of the operation of the timing transfer system of a TSN node as it produces an analog signal at the start of every second which is disciplined to the internal oscillators and clocking mechanisms of the node. Hence, an indicator of accurate synchronization is a sufficiently low deviation between the PPS pulses of the master and slave devices with ideally negligible jitter. An oscilloscope can be used to graphically inspect the deviation between the PPS signals, whereas the TDC frequency counter is used for gathering large data sets for further statistical analysis.

Thus, the TDC can record large amounts of back-to-back measurements of time differences between the instants that the PPS pulses are triggered at the master and slave devices. These data sets are then used for deriving parameters such as the average master-to-slave offset, its standard deviation, or the peak-to-peak range of the PPS difference. This latter parameter gives an idea of the amount of jitter associated with a certain synchronization scheme. The data sets recorded through the use of the TDC can be further analyzed to study the performance of the timing and frequency distribution of a given synchronization scheme by deriving indicators such as the Allan Deviation (ADEV) or the Time Deviation (TDEV). Their derivation is explained in detail in [96] and we have used them for assessing the overall performance of some of our TSN implementations. The former assesses the overall frequency transfer accuracy and can even account for the presence of different types of phase noise and quantify their respective influence on the attainable synchronization. The latter measures the average time error between the master and slave devices in the timing system. In practice, we have used some openly available toolsets, such as the "*AllanTools*" python library [97], to calculate ADEV indicators and to plot them graphically to show our results.

4.4.3 *The end-to-end latency and determinism at data forwarding*

One of the main features of a TSN system is its capacity to forward different types of messages and provide a service characterized by a certain amount of reserved bandwidth, bounded end-to-end latency, and reduced packet delay variation on the delivery of the higher priority messages. Specifically, provided that the allocated bandwidth is sufficient for transmitting a specific TSN flow, the determinism of the system is characterized by its ability to deliver messages within a known deadline with low delivery jitter. Thus, we have selected the end-to-end latency as the main variable for assessing the delivery determinism of our TSN system, which is measured using a setup like that of Fig. 4.5.

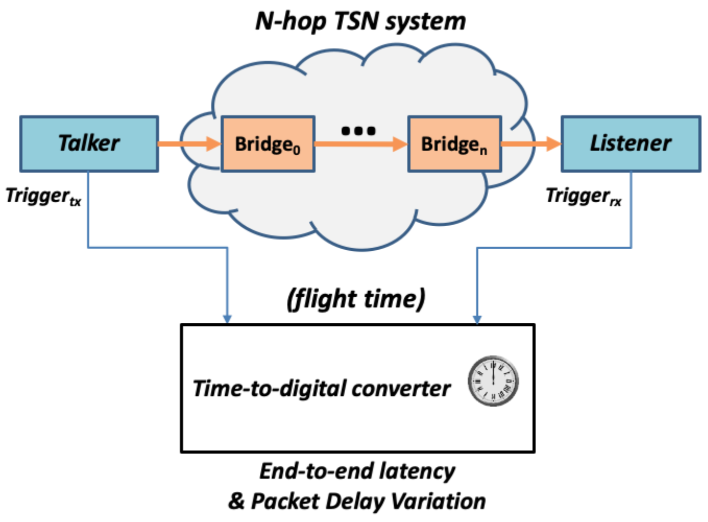


Figure 4.5
Diagram of the procedure for estimating the attainable determinism of a TSN network system.

As shown in the figure, the setup uses an analogous layout to that used for the assessment of timing performance (Fig. 4.4) with one key difference: instead of measuring the PPS difference, these experiments are meant to record the flight time (*propagation time*) of a given message over the network. This provides an indication of the end-to-end latency associated with a specific type of message. To increase the accuracy of our estimations, all the latency measurements were conducted directly by inspecting the internal signals of the TSN nodes. As indicated in Fig. 4.5, this was done by using a TDC counter that records the time differences between the monitoring pulses produced at the TSN nodes originating (*talker*) and receiving (*listener*) the messages. These nodes are in turn fitted with specific debug IP cores that produce short analog triggers upon generation and reception of a specific type of message.

As before, the data sets so captured with the TDC counter can be further analyzed for deriving additional statistics such as the average end-to-end latency, its standard deviation, or the peak-to-peak range, which gives an idea of the packet delay variation associated with the experiment.

4.4.4 *Network congestion and message transmission integrity*

The type of service expected of a TSN networking device is meant to provide a deterministic delivery of the critical data while avoiding the discarding of messages on account of the network congestion effects. Network congestion can cause the internal buffering elements of the different bridging nodes of an Ethernet network to become saturated and, thus, to start dropping frames. The enhancements of TSN attempt to remedy this undesirable effect by applying traffic shaping policies that can potentially spread out traffic bursts over time by defining a set of time slots reserved for the transmission of a specific traffic flow. Hence, the effect of this mechanism is largely dependent of the configuration parameters supplied by the user: if the highly demanding traffic is not allocated appropriately sized transmission slots or is not served frequently enough, congestion losses are still bound to occur anyways. Hence, the efficiency and robustness associated with a specific user configuration is evaluated by totalizing the number of packets injected into the network at the TSN *talker* device and those received at the *listener*, as shown in Fig. 4.6.

Hence, as shown in the diagram, the attainable robustness of the TSN system will be measured by calculating the packet loss ratio associated with the forwarding of a set of messages. This is estimated by inspecting the network traffic injected at the TSN *talker* with a network inspection tool such as Wireshark or *tcpdump* to quantify the number of injected messages. The same process is then repeated at the TSN receiver to determine the number of received messages. The packet loss ratio (*PLR*) can then be estimated directly as the complement of the ratio between the number of received (N_{out}) and injected (N_{in}) data frames, as indicated in (4.1).

$$PLR_{\%} = (1 - \frac{N_{out}}{N_{in}}) \cdot 100\% \tag{4.1}$$

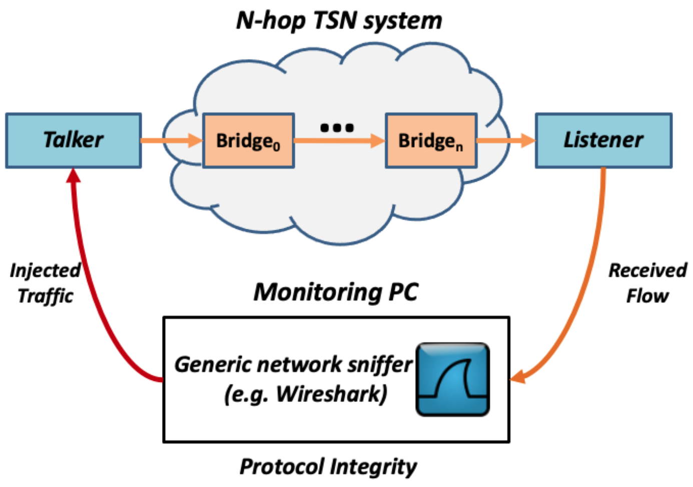


Figure 4.6
Diagram of the methodology for estimating the packet loss ratio associated with specific TSN system settings.

4.4.5 *The bandwidth characterization and performance estimations*

The attainable bandwidth for the TSN communication links was also measured using a setup similar to that of Fig. 4.6. Thus, as the different traffic shaping policies of the TSN system may have a direct influence on the reserved bandwidth for data forwarding of specific flows, their effects on the available bandwidth are empirically quantified by running a series of link speed benchmarking tests. These tests consist of the injection of different TCP or UDP flows from the TSN data source that get forwarded to the TSN *listener* using the *iperf* tool. The data sink PC connected to the TSN *listener* runs an *iperf* server instance for reporting the results of test, which estimate the attainable bandwidth associated with either the full link or the transmission of specific TSN stream.

On other occasions, the bandwidth measurement will consist of the injection of a fixed-rate flow produced at the TSN *talker* and the corresponding bandwidth estimation will then be carried out at the TSN receiver by capturing the traffic with Wireshark and inspecting the packet dump capture statistics. This method is also convenient for verifying that the expected traffic pattern for the TSN traffic is present on the link as a result of the configuration of the traffic shapers.

4.4.6 *The FPGA design and synthesis workflow*

As for the design workflow of the different IP cores that are present in our TSN nodes, the approach that we have taken is to follow a design-driven cycle with a twofold verification stage using VHDL simulation test benches and on-board debugging.

The IP core implementation workflow for the FPGA-based devices of the different product families from Xilinx usually requires the use of Vivado for assembling a high-level description of the system by interconnecting functional elements in a block diagram, and then defining or instantiating custom IP cores as needed. The design of these cores involves the definition of the corresponding logic or state machines through a VHDL description. This description is then verified for behavioral correctness using the built-in simulator included with the Vivado Design Suite. Hence, the simulation process involves the definition of a test bench containing a set of predefined stimuli that are fed to the user IP core logic. The logic will return an expected output pattern provided that it can handle the input stimuli correctly, otherwise the designer will be able to iteratively debug the core logic until the expected output can be attained. This design and verification process, which can save enormous design and validation time by helping designers spot the most substantial errors in their IP core logic before undergoing the time-consuming process of generating a bitstream, can be examined in Fig. 4.7.

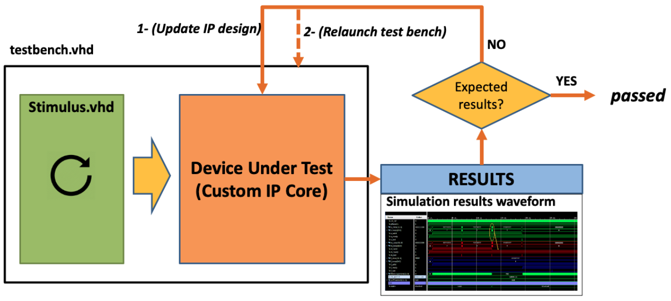
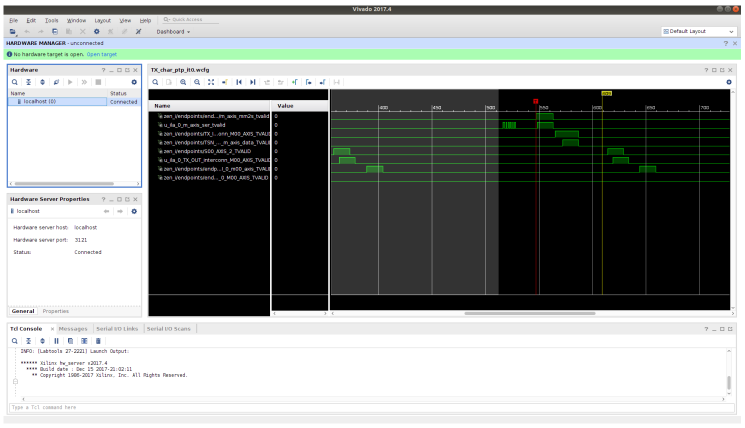


Figure 4.7
Generic diagram of the iterative simulation process for the IP core verification with user-defined test benches.

Once the behavioral simulation model is complete, the designer can proceed to generate the bitstream for programming the FPGA device. It the event that the newly synthesized IP core exhibits unexpected behavior, the user can still perform real-time on-board verification by integrating ILA or VIO debug cores [89]. ILA cores allow the user to define capture conditions on the set of signals being examined, whereas VIO cores allow the user to inject a series of predefined patterns for studying the response of a logical circuit to a certain set of trigger signals. We have performed this process with an instance of the Vivado logic analyzer, whose user interface can be examined in Fig. 4.8.



Name	Status
localhost (0)	Connected

Hardware Server Properties

localhost

Hardware server host: localhost
Hardware server port: 3121
Status: Connected

Name	Value
zen_i/endpoints/end.../m_axis_mm2s_tvalid	0
u_ila_0_m_axis_ser_tvalid	0
zen_i/endpoints/TX_I...onn_M00_AXIS_TVALID	0
zen_i/endpoints/TSN_..._m_axis_data_TVALID	0
zen_i/endpoints/S00_AXIS_2_TVALID	0
u_ila_0_TX_OUT_interconn_M00_AXIS_TVALID	0
zen_i/endpoints/endp...l_0_m00_axis_TVALID	0
zen_i/endpoints/end..._0_M00_AXIS_TVALID	0

```
INFO: [Labtools 27-2221] Launch Output:

****** Xilinx hw_server v2017.4
  **** Build date : Dec 15 2017-21:02:11
    ** Copyright 1986-2017 Xilinx, Inc. All Rights Reserved.
```

Type a Tcl command here

Figure 4.8
The graphical user interface (GUI) for interacting with the ILA and VIO debug cores from the Vivado hardware server.

This process has allowed us to gather stimuli directly from a running FPGA implementation that might have been overlooked and hence left out of the simulation behavioral model. Thus, the new stimuli can be included in the simulation and debugged iteratively in a two-stage process: an initial simulation with a primary stimulus model of the device which is further complemented with the data gathered from the subsequent on-board verifications with the debug cores. This workflow is depicted in the diagram from Fig. 4.9.

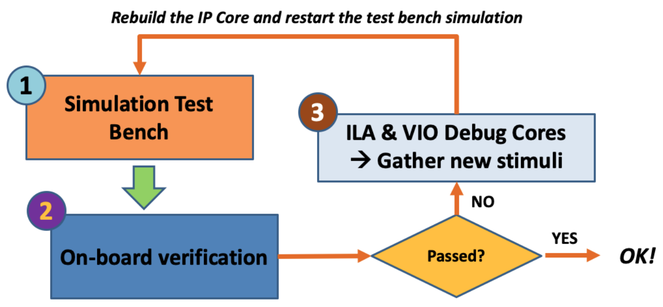


Figure 4.9
The iterative FPGA characterization and debug process based on formal behavioral verification with VHDL test benches and live on-board testing with the Xilinx ILA and VIO debug cores.

4.4.7 *Overview of the experimental procedure*

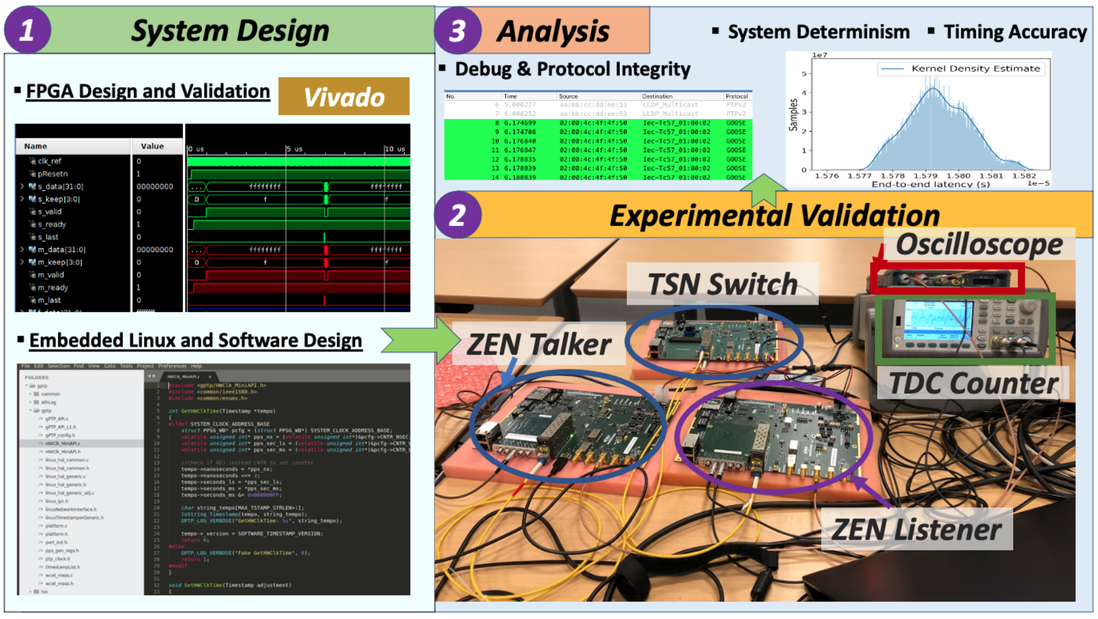


Figure 4.10
The overview of the methodology for building and validating our TSN nodes. We start with an initial design stage, and then we move on to the laboratory characterization, and complete the process with the corresponding analyses.

An overview of the experimental methodology that we have used throughout the thesis can be examined in Fig. 4.10. Hence, our workflow generally consists of a combination of design, validation, and analysis tasks. We normally start with the design stage, where we usually supply the implementation of a specific FPGA firmware module or software component. We can accomplish the design of FPGA cores with the use of tools such as ***Vivado***, with its built-in simulator and hardware monitoring server. The design of software elements usually involves following the usual procedures for building system images and software applications for an embedded OS. In the case of Linux environments, we usually rely on cross-compilation and the ***Buildroot*** [83] tool suite for building any new functionalities that may be needed. After the design stage, we validate our designs in the appropriate ***laboratory test benches***. As shown in the figure, these will usually be made up of our ***TSN prototype nodes*** (e.g., WR-ZEN boards) with the appropriate connectors and interfaces (e.g., optical fiber, SFP connectors, . . .). Our measurements will usually attempt to characterize the end-to-end latency or the synchronization accuracy of the timing component of the TSN nodes. We will use ***TDC counters or oscilloscopes*** to this end. Lastly, once the experiments are complete, **we finish with an analysis stage**. This is where we calculate the appropriate indicators resulting from our measurements, such ***end-to-end latency histograms*** to assess the overall system determinism or the value of the PDV. Likewise, we may also derive statistical indicators such as the ***ADEV deviation*** to study the accuracy of the timing system. Furthermore, we also verify the effectiveness of our traffic-shaping policies with TSN and the integrity of protocol messages with network sniffing tools, such as ***Wireshark, tcpdump, or iperf3***.

5 A WHITE RABBIT-SYNCHRONIZED ACCURATE TIME-STAMPING SOLUTION FOR THE SMALL-SIZED CAMERAS OF THE CHERENKOV TELESCOPE ARRAY

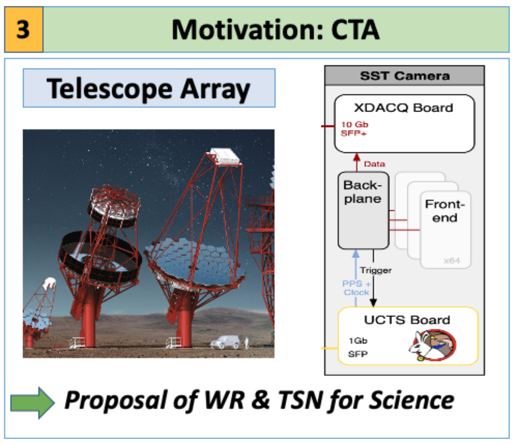


Figure 5.1
Overview of the contents of Chapter 5 with the motivational use case for the Cherenkov Telescope Array.

This chapter presents a motivational use case to illustrate the potential of combining TSN networking systems with the highly accurate WR timing system. We use a reprint of our journal contribution in [1] to show this. Hence, in the publication, we present the development of one of the components of the data acquisition system of the small-sized telescopes (SSTs) at the Cherenkov Telescope Array (CTA). Thus, CTA is a scientific facility for high-energy physics devoted to the study of Cherenkov light flashes. An array of telescopes is used for accurately detecting the source of the Cherenkov light cone in the sky, and their acquisition system would subsequently time stamp each Cherenkov event for further processing or for discarding false positives. In the publication, we present the time-stamping module that we developed to this end: the ZEN-CTA node. The node relies on WR synchronization and an advanced TDC for producing the time stamps of each observation. The time stamps are in turn forwarded over the Ethernet interfaces of the nodes alongside the WR protocol data. In this context, we show how our system can produce highly accurate time stamps and support their transmission at substantially large rates over Ethernet without any significant effect on the operation of WR timing. We claim that this is a motivational use case for the integration of TSN features and WR timing, which is the subject of the discussion that we present in Section 5.8 as a follow-up to the contents of the article. Furthermore, some of the lessons that we learnt with regards to kernel driver optimization and development have been put to use throughout the research that we present in this manuscript. We acknowledge this fact in the Appendixes A and B, which reuse the supplementary materials of the publication in [1].

Chapter contents

5.1 ABSTRACT

This paper presents the ZEN-CTA node: a programmable system-on-chip (SoC) with White Rabbit-synchronization capability. It targets a solution for the uniform clock and trigger time-stamping module of the small-sized telescopes in the Cherenkov Telescope Array. This module is tasked as a distributed acquisition device with a focus on obtaining time stamps for candidate Cherenkov events, which could be generated at potentially high rates from very high-energy gamma rays, and their subsequent distribution over Ethernet. In this context, the customized design of the ZEN-CTA node is examined thoroughly, including its generic implementation aspects and its main functional blocks. The design of the White Rabbit-assisted time-to-digital converters (TDCs) for time-stamping analog triggers is presented in detail alongside the implementation of an upgraded high-speed data path (1 Gbps) for the White Rabbit-compatible Ethernet interfaces of the node. The new data path will feature a direct memory access engine for direct software transmissions and a hardware description language (HDL) coprocessor for high-speed forwarding. Next, the time-stamping accuracy of the White Rabbit-enhanced TDCs will be characterized alongside the forwarding efficiency of the new data path. Lastly, conclusions are drawn and the main contributions of this research are enumerated, a potential deployment within the Cherenkov Telescope Array infrastructure to support the acquisition of Cherenkov light is considered, and additional use cases are mentioned.

5.2 INTRODUCTION

Data acquisition systems are at the heart of most scientific infrastructure facilities; such as particle accelerators, telescope arrays, or even seismic data acquisition systems. These have in common that they are usually implemented in the form of distributed systems, with a series of networked nodes scattered about the application domain which are devoted to the acquisition of data, and usually a centralized backend for further processing and storage of all the collected data.

The use of a timing transfer mechanism is normally required in order to synchronize all the nodes and ensure that they can gather data synchronously. This can be achieved with two main approaches: either through a dedicated timing transfer system, which is usually segregated from the main data network, or through a convergent system combining timing transfer protocols and user data. Examples of the former paradigm are the GPS-based solutions for synchronizing distributed nodes, used in the Atacama Large Millimeter Arrays [98], whereas examples of the latter are convergent data and timing networks utilizing White Rabbit (WR), which was used in KM3NET [80]. The first approach is costly but typically used because of its simplicity or its ability to provide a reasonable performance. The second is more cost-sensitive but more complex to design, manage and maintain if more demanding specifications are required.

This study examines the development of an event-sampling node for the distributed acquisition system of the Cherenkov Telescope Array (CTA) [99] that will be integrated on the control board of its small-sized telescopes (SSTs) as its White Rabbit synchronization source and time-stamping component. Thus, this node – the ZEN-CTA – will be able to capture candidate Cherenkov events, generate their associated time stamps with high-accuracy time-to-digital converters (TDCs), and forward them to a software-level decision trigger using a high-speed data path capable of keeping up, or even exceeding, with the event generation rates expected in the requirements of the SSTs in CTA. The rest of this section is then dedicated to providing an overview of different distributed sampling systems and time-stamping techniques that are applicable for scientific infrastructure before examining the node in greater detail. After this general overview, the specific case of the SSTs in CTA and their requirements are presented in Section 5.3. The implementation of the ZEN-CTA node and its time-stamping solution is presented in detail in Section 5.4, and its renovated data path is introduced in Section 5.5. Section 5.6 then proceeds to experimentally validate these improvements in a laboratory setup. Lastly, Section 5.7 contains the conclusions and introduces possible future research. Additionally, select implementation topics are presented in Appendixes A and B.

5.2.1 *Overview of Distributed Event Acquisition Systems*

It is admittedly difficult to establish the generalized, canonical architecture of the distributed acquisition systems of scientific infrastructure given the wide variety of requirements, use cases, and constraints required in different projects and facilities. Yet, there are some interesting publications [100] that aim to give a broad understanding of the different types of distributed acquisition systems and their individual components that, when considered as a whole, make up a specific framework – a toolkit – for distributed data acquisition. Thus, as noted in [100], these frameworks can be classified according to their size, their functionalities and their underlying technologies. They usually feature a transport system, system steering and job controls, data processing interfaces coupled with storage solutions, and event-building systems. These latter processes analyze raw streams of data to extract the set of parameters that will represent the outcome of a specific observation. System-wide synchronization of the entire system with GPS receivers, White Rabbit [101], or the precision time protocol (PTP) [77] timing may assist in the event-building and parameter extraction process.

There are also, data acquisition suites that provide an implementation of all or some of the preceding components as an out-of-the-box solution. Prominent examples of these types of data acquisition systems include specialized middleware solutions such as CORBA [102], featured in the ATLAS experiment, although, as of 2016, it transitioned to a simpler message-passing protocol based on the ZeroMQ library [103] [104]. Other well-known solutions are MIDAS [105], DAQ-Middleware [106], or the LabView Suite from National Instruments [107]. The Cherenkov Telescope Array has opted for a middle ground: Its small-sized telescopes use the ZeroMQ Library for data acquisition and message-passing, Google Protocol Buffers for Serialization [108], and the entire system is steered using a CORBA framework (ALMA) [109].

5.2.2 *Time-Stamping Devices for Scientific Applications*

The event-building process assembles complex events from the parameters associated with the reception of raw data streams from the data-gathering nodes in the system. Time-to-digital converters are an essential part of this task, as they generate the high-accuracy time stamps upon reception of events that are needed for the event-building time correlation processes, which involves the use of precise time synchronization to produce accurate time stamps. Moreover, scientific applications and instrumentation require highly accurate TDC implementations. This can be achieved by either incorporating a specialized application-specific integrated circuit (ASIC) onto the printed circuit board (PCB) of the node or as an expansion card. Some of these applications include commercial-grade solutions, such as the TDC7000 ASIC from Texas Instruments (55 ps accuracy) [110], or highly accurate components such as the ACAM TDCs (15 ps accuracy) [111]. The use of ASICs or physical expansion cards has many advantages, including the fact that they are built with internal temperature-variation compensation processes and relatively high precision. However, as pointed out in [112], their use is only feasible with mass-produced applications. In contrast, field-programmable gate array (FPGA) designs, which are used for many measuring nodes, are more flexible and represent the ideal candidates for implementing TDCs.

These FPGA implementations are generally composed of two different time estimation stages that are combined for producing an enhanced time stamp with sub-nanosecond precision. The general idea is based on providing a digital implementation of a Vernier scale measuring circuit, as explained in [113]. Thus, the initial step of this Vernier-style measurement provides a rough time estimation with the system clock of the TDC implementation. A sub-clock cycle estimation can then be produced by means of a delay line that provides an associated sub-clock cycle delay code. This latter step usually requires complex design and calibration to make up for temperature and routing variations in the FPGA fabric. Some approaches implement this delay line using carry chain primitives [114], whereas others make use of the internal I/O (input/output) elements of the FPGA, which avoid most calibration-related issues, such as the study in [112] or the TDC functionality presented in this publication.

TDCs are one of the main building blocks of the event acquisition systems deployed in different projects and facilities. In these environments, different implementations of distributed nodes integrating time-stamping capabilities are usually present with TDCs tailored to meet the accuracy and event acquisition rates expected in their corresponding scenarios. In the framework of CTA, the time and clock stamping (TiCkS) node [115] is another alternative to the node presented in this study. Other notable projects for scientific facilities with distributed time-stamping nodes are KM3NET (arrays of sensors for a neutrino telescope) [80] and DIAPHANE (Muon Tomography for geological structures) [116].

5.3 OVERVIEW OF THE CHERENKOV TELESCOPE ARRAY

CTA is one of the major future facilities in the field of astroparticle physics and high-energy astronomy, dedicated to exploring the high-energy universe with gamma rays above 20 GeV [117], [118]. When such a high-energy gamma ray hits the top of the atmosphere, it initiates a shower of particles which emit Cherenkov light. These Cherenkov light flashes last only a few nanoseconds and typically illuminate an area of a few hundred meters in radius on the ground. By placing multiple telescopes within the Cherenkov light cone, it is possible to measure simultaneously the weak and short light flash of an air shower from different positions and thus reconstruct accurately the primary gamma-ray energy and direction.

To increase the stereoscopic detection of gamma-ray-initiated air showers, the CTA will consist of more than 100 telescopes spread between two sites, one on La Palma (Spain) and one near Paranal (Chile). It is expected that the arrays will be built with an approximately circular layout, although a final distribution is still being analyzed as shown in [119]. The northern hemisphere array will be more limited in size ($\sim$500 m in diameter) and will focus on the CTA's low- and mid-energy ranges (20 GeV to 20 TeV), whereas the southern array with a diameter of about 2.5 km will span the entire energy range of the CTA, covering gamma-ray energies from 20 GeV to 300 TeV with three types of telescopes: large (LSTs), medium (MSTs), and small-sized telescopes. Whilst the telescope types differ in sensitivity, energy range covered and field of view, they will all consist of tessellated mirrors which focus the Cherenkov light onto fast-recording cameras with a few thousand photo-sensors. Whenever a camera identifies an image of an air shower candidate, usually defined as a configurable cluster of photo-sensors hit by a coincident light signal, a trigger is formed, and a time stamp is assigned to the camera event. For LSTs, the expected telescope trigger rates are on average of the order of 15 kHz, for MSTs at around 7.5 kHz and for SSTs about 600 Hz with individual burst rates expected to be twice as high for each of them.

To identify all telescopes which have recorded the same air shower event and furthermore suppress local background events, we will use an array-level trigger in the

CTA. The trigger logic searches for coincidences in time (typically within 10 to 100 ns) between neighboring telescopes. For this mechanism to operate properly, a network transmission latency of up to 100 ms between the telescopes and the CTA data center has to be enforced. To allow for scalability and flexibility, the array-level trigger will be implemented in software and be based on the camera trigger time stamps. Precise timing is therefore mandatory for the CTA and the relative timing precision between different telescope cameras is specified to be better than 2 ns on average with less than 1 ns jitter (RMS). The requirement for the absolute time precision with 1 μs is less stringent.

To achieve such a high time precision, the CTA will use a unified timing system which is based on a hierarchical White Rabbit network, as shown in Fig. 5.2. The network itself consists of a single master switch which is connected to an external GPS clock to provide absolute timing information. The master switch then distributes the time and frequency reference over a network of intermediate White Rabbit switches, located at the on-site CTA data center, via optical fibers until reaching the CTA telescopes. In each of the CTA telescope cameras, a uniform clock and trigger time-stamping (UCTS) board will act as the White Rabbit timing node and will thus provide the interface between the telescope cameras and the White Rabbit timing system. The cameras will also feature the XDACQ acquisition card [120] to forward raw observation data from the telescopes. At the UCTS board, a precision time stamp will be associated with the camera trigger pulse and a trigger message is formed. This message is then sent to the software array trigger in the central data center. In the case of a positive array-level trigger decision, the bulk camera data are transferred to the storage system for scientific analysis. Bulk data from camera triggers which fail the timing coincidence criterion are usually discarded as they are probably not from genuine gamma-ray air showers.

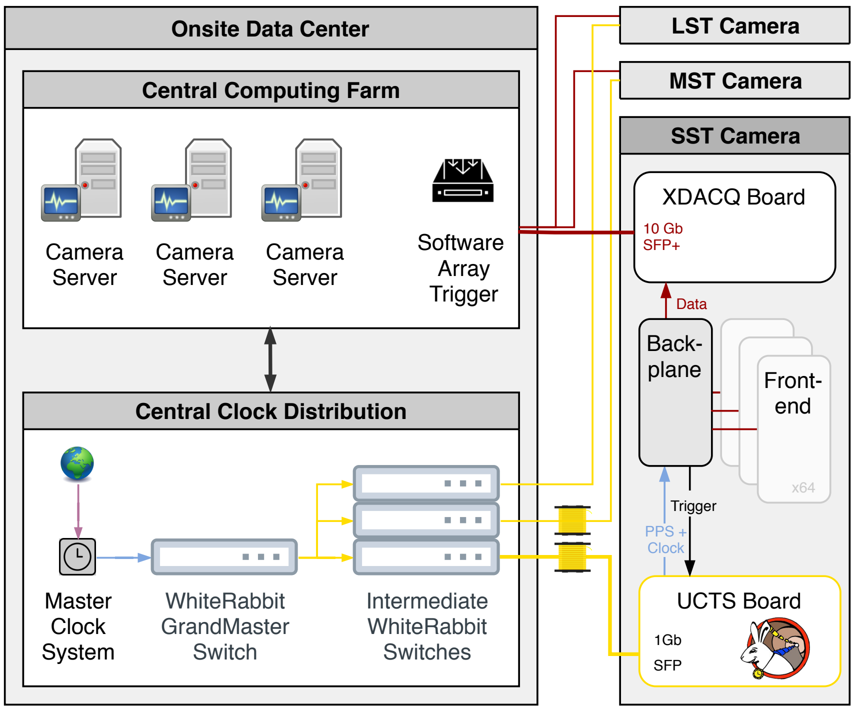


Figure 5.2
Schematic overview of the CTA timing system described in the text. © **2020 IEEE**.

The UCTS nodes are thus key to the CTA operation: they have to ensure stable time and clock synchronization amongst all telescopes on-site at the nanosecond level and in return provide high-precision time stamps at rates of up to several tens of kHz. We will develop different prototypes which will be partly based on existing WR nodes. One prototype is the TiCkS board [115], which is based on the WR SPEC node [79]. The other, the ZEN-CTA node, is the solution presented in this paper for implementing the functionality of the UCTS node featuring enhanced time-stamping interfaces and improved event transfer capabilities. It is based on the WR-ZEN (White Rabbit – Zynq-Embedded Node) and its design will be described in detail in the following sections.

5.4 UCTS DESIGN WITH THE WR-COMPATIBLE ZEN-CTA NODE FOR THE SMALL-SIZED TELESCOPES

The ZEN-CTA board, shown in Fig. 5.3, is a customized version of the WR-ZEN node [121], providing the highly accurate WR timing distribution system in a reduced form factor (60 mm $\times$ 164 mm) that makes it suitable for its use as the UCTS board of the CTA SST Camera [120, 122]. The implementation of the ZEN-CTA node is described in detail in this paper. This node is presented as a plausible alternative for supplying an implementation of the UCTS component in the data acquisition system of CTA, which will be tasked with acquiring sequences of events, which indicate a possible Cherenkov detection, and generate their associated time stamps. These time stamps will then be forwarded to a remote data center for processing. Hence, in order to conform to these requirements, the proposed board is fitted with a full, dual-port implementation of the WR PTP core (WRPC) [123] that will provide the timing distribution functionality, and an advanced TDC for generating enhanced precision WR time stamps for the trigger signals produced at the CTA camera upon detection of possible Cherenkov light flashes. The time-stamping module is described in this section, whereas the Linux environment of the board, including services and applications, and its direct memory access (DMA) networking support are explained in-depth in Appendixes A and B, respectively. The Ethernet data path of the board, which had to be renovated to ensure that it could keep up with events received at substantial rates, is presented in a separate section devoted to its implementation (Section 5.5).

The TDC is tasked with receiving triggers initiated by generic camera events, which could either be Cherenkov event candidates or calibration patterns, and generate an associated WR time stamp with enhanced accuracy. A number of alternatives have been proposed for detecting and recording trigger data for CTA, as is the case of the Compact High Energy Camera (CHEC) [124] or the use of digital camera triggers processed on an FPGA [125]. In contrast to these, this study introduces an approach that detects and records array-level triggers by executing a software algorithm in a backend server that correlates time-stamp data originating from several telescopes. The TDC, which is tasked with producing these time stamps, is fully implemented in FPGA logic, and only requires a simple signal adaptation stage on the ZEN-CTA circuit board that is implemented with a low-skew, 1-to-4 differential to low-voltage differential signaling (LVDS) fan-out buffer that replicates the differential input signal into four separate copies. Thus, the TDC will feature two differential input ports for acquiring analog signal

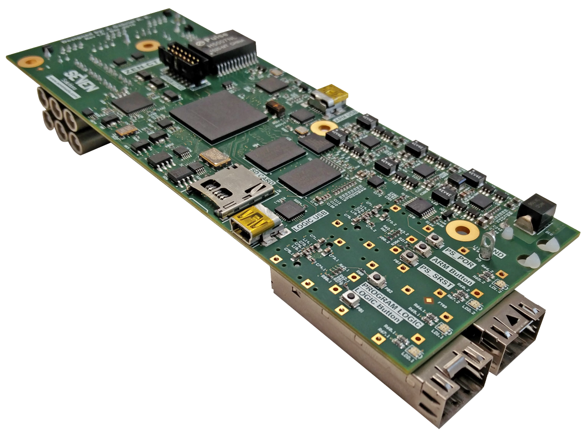


Figure 5.3
Picture of the ZEN-CTA board, which features WR timing on two 1G Ethernet SFP ports, advanced TDC-assisted time-stamping, and a reduced form factor. © **2020 IEEE**.

triggers: The *Readout* trigger input, and the *Busy* trigger input. These analog control signals are generated by the trigger system to indicate the detection of Cherenkov light patterns. Furthermore, their input ports into the TDC are physically implemented on the ZEN-CTA Board as RJ45-type connectors utilizing differential signaling, with each port meant to produce its own time stamp associated with the reception of events from either the *Busy* or *Readout* signals. Consequently, as these input signals are fanned out as separate copies, the FPGA implementation of the TDC in the ZEN-CTA actually consists of two separate time-stamping instances for each trigger input port. The architecture of each TDC instance is thus shown in Fig. 5.4. This TDC can be considered an efficient implementation of an arbitrary trigger acquisition device [126] that, given the use of White Rabbit synchronization, is suitable for its application to distributed systems.

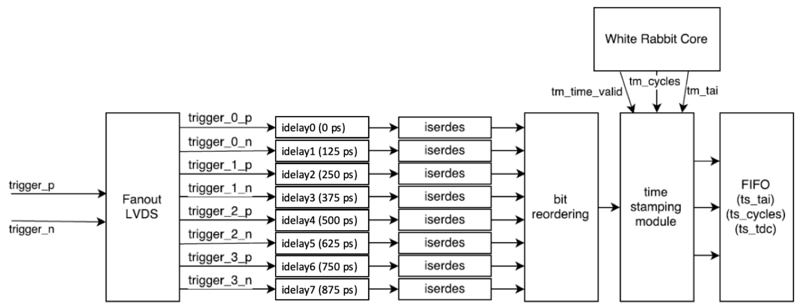


Figure 5.4
Block diagram showing the implementation of the TDC module used for each trigger input port (*Readout* or *Busy*) of the ZEN-CTA Board. © **2020 IEEE**.

As shown in the block diagram, each TDC instance will take a differential signal supplied through one of the trigger inputs of the ZEN-CTA Board to produce an enhanced WR time stamp with sub-clock cycle precision. This signal will then be split, converted into LVDS and fed into the FPGA TDC module. Specifically, for each time-stamping instance, the split, fanned-out copies of the signal will be fed into 8 different *idelay*

[127] primitives configured to produce incremental delay values. These primitives have designated tap steps of 78 ps, which will be combined as needed to produce the approximate effect of gradual 125 ps delays (Table 5.1). These gradually delayed versions of the input signal will subsequently be sampled with double data rate (DDR) input deserializer (*iserdes*) components [127]. Thus, each of the eight deserializers in the setup uses a WR-synchronized, 125 MHz (8 ns cycle) clock domain for producing an 8-bit output, and a helper clock of 500 MHz (2 ns cycle) for sampling the delayed signals supplied from the *idelay* primitives in DDR mode. Hence, this ensures that each delayed version of the differential input signal is sampled at least eight times during each 8 ns clock cycle. As a result, each bit in the 8-bit output words of the deserializers will have an associated resolution of 1 ns. Furthermore, as the outputs of the deserializers have an approximate relative phase shift of 125 ps, their 8-bit outputs can be combined to form a 64-bit pattern indicating transitions within an 8 ns clock cycle in approximate steps of 125 ps. Hence, this design would theoretically be able to handle pulses longer than 8 ns with dead times of up to 16 ns.

Once the enhanced time stamp has been composed, it is transferred to a specially reserved bare system memory location using the general-purpose AMBA AXI (*Advanced Microcontroller Bus Architecture with Advanced eXtensible Interface*) infrastructure (e.g., Xilinx AXI Data Movers and customized Interconnect components). Then, upon transfer to system memory, a specially designed user-level application can be used for forwarding these time stamps over the network to the CTA camera server for further analysis. We have proposed an alternative approach for achieving a high-throughput transfer of the enhanced WR time stamps and it involves an FPGA-based, low-latency solution that allows the direct injection of user datagram protocol (UDP) packets carrying the enhanced WR time stamps in their payload.

The study of the time-stamping accuracy of the TDC modules becomes mandatory since the *idelay* components cannot be configured to produce an exact delay value of 125 ps as the actual *tap* value of each *idelay* is 78 ps instead, as shown in Table 5.1.

Component	Expected Delay (ps)	Actual Delay (ps)	Deviation (ps)
idelay0	0	0	0
idelay1	125	156	-31
idelay2	250	234	16
idelay3	375	390	-15
idelay4	500	468	32
idelay5	625	624	1
idelay6	750	780	-30
idelay7	875	858	17

Table 5.1
Comparison between the expected delay values produced at each *idelay* input element, and the synthesizable delay values that can be produced by the implementation tools and the FPGA device using an 8 ns reference clock cycle. © **2020 IEEE**.

It should be noted that the proposed TDC component is an FPGA-based solution that only requires a simple signal conversion stage on the PCB. Dedicated, hardened implementations based on specialized PCBs or front-mezzanine cards (FMC) do exist and can provide varying degrees of accuracy; such as the *ACAM* integrated TDCs which can be used to measure time intervals from the range of 1 ns down to several picoseconds [111]. This FPGA-based solution for the TDC component is preferred because, contrary to using dedicated hardware, it comes with the advantage of using built-in I/O FPGA primitives from Xilinx in a design that uses the IDELAYCTRL [127] primitive to continuously calibrate the *idelay* components to ensure that they can provide consistent delay values, thereby offsetting the effects of varying temperatures, voltage, or fabrication process. Furthermore, as the synthesizable delay value produced by each *idelay* tap is tied to the specific speed-grade of a particular device, the degree of accuracy of the TDC implementation could be improved by selecting higher speed-grade devices, which would allow for synthesizable tap delays of up to 39 ps when using higher speed components (grade -3) [128].

5.5 IMPLEMENTATION OF WR-CAPABLE NETWORKING. DMA-BASED UPGRADE AND TDC TIME STAMP TRANSMISSION

White Rabbit-compatible nodes; such as the WR-ZEN Board [129] or the ZEN-CTA node implementation presented in this paper, were designed with timing transfer performance in mind and therefore feature a real-time White Rabbit stack implementation paired with highly precise clocking circuitry [130] for delivering near-deterministic timing distribution. As an added convenience, these nodes also feature an ancillary data transfer functionality supported by the WR network interface core (NIC) Module [131]. This allowed the coexistence of regular Ethernet data traffic and timing synchronization information over the same physical network links, which results in simplified infrastructure management and decreased deployment costs. Nonetheless, the WR NIC has major bandwidth limitations and a high-bandwidth DMA-based upgrade for the WR-capable network interfaces of the WR-ZEN Board has had to be introduced [132]. This upgrade makes the use of these boards feasible for applications such as distributed Analog to Digital Converters or distributed oscilloscopes.

The research described in this paper shows the process whereby the original design of the ZEN-CTA Board was adapted to support a dual-port, WR-compatible, Gigabit Ethernet network implementation that also provides a fast, direct-to-network data path for transmitting WR time stamps generated at the TDC module using the design of the WR-ZEN Board [132] as a starting point. Consequently, this migration process will be twofold. The initial stage will revolve around the adaptation of the ZEN-CTA network data path to include the new DMA-based upgrade and the direct-to-network path for the TDC modules. The subsequent stages will focus on integrating this new data path with the embedded Linux environment running on the processing system with the goal of exposing this upgrade as regular Ethernet interfaces that also feature White Rabbit support.

5.5.1 *DMA-based Networking. Implementation of a dual-port, high-speed data path for the WR-compatible Ethernet interfaces of the ZEN-CTA Board.*

A simplified block diagram of the new data path architecture of the ZEN-CTA Board is shown in Fig. 5.5. When the legacy data path was enabled, which is still visible in the diagram as the connections between the processor and the WR End-Points (*EP0* and *EP1*) via the *"AXI-Lite to Wishbone Bridge"*, the WRPC was used for the purposes of distributing timing and synchronization information throughout the network, as well as for sending ordinary Ethernet traffic over the WR synchronization network using the WR NIC modules, which are connected to their corresponding WR End-Point (*EP0* or *EP1*). The entire data path between the WR NIC modules and the ARM-based processing system [133] used memory-mapped interfaces without support for data burst transfers, which explains their relatively poor performance. These interfaces were the Wishbone system bus and the no burst-compatible, AXI-Lite specification of the AMBA AXI bus, which are bridged using the *"AXI-Lite to Wishbone"* adaptor shown in the figure. The use of the TDC modules for acquiring enhanced precision time stamps for external analog triggers is also displayed. These modules use the time reference signal generated by the WRPC to produce their time stamps and forward them to a reserved, bare-memory location of the main system memory of the processing system using the Xilinx Data Mover Modules [134]. Given the promising results obtained in a previous study for enhancing the throughput of the WR-compatible network interfaces of the WR-ZEN Board [132], we considered a similar approach for its application on the WR-compatible interfaces of the ZEN-CTA board. This led to the implementation of the new architecture presented in Fig. 5.5, featuring a dual-port design that provides a high-bandwidth data path for network data transfers between the two WR-compatible interfaces of the ZEN-CTA board and the ARM-based processing system.

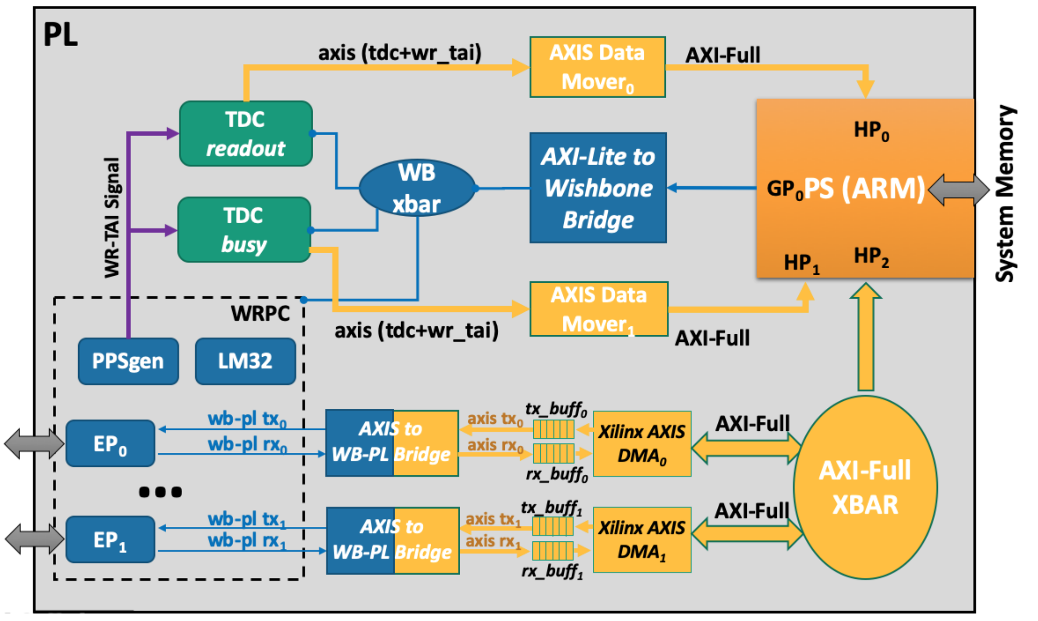


Figure 5.5
Simplified block diagram architecture of the ZEN-CTA board displaying the implementation of the dual-port, DMA-based upgrade on its Programmable Logic (PL) to enable high-throughput data transfers between the WR-compatible interfaces of the WRPC (*EP0* and *EP1*) and the ARM-based processing system (PS). © **2020 IEEE**.

In order to apply the throughput optimization to the WR-compatible interfaces of the ZEN-CTA board, the following elements, which were introduced in [132], had to be combined. These consist mainly of a number of custom IP cores, AMBA AXI and Wishbone bus infrastructure, and auxiliary buffering components which are described below.

- **XILINX AXI DMA**. Xilinx-owned core [135] tasked with handling high-throughput data transfers between the WR-compatible interfaces of the board and the processing system using a central processing unit (CPU)-efficient scatter/gather mode.
- **AXI-STREAMING TO WISHBONE FABRIC CONVERTER (*AXIS to WB-PL* Bridge)**. Module tasked with implementing the transformation between the Wishbone Pipelined bus domain of the WR PTP core, and the AXI-Streaming System Bus used by the DMA engine interfacing with the processing system. Additionally, this module is used for producing Ethernet padding when required and for enabling the generation of time stamps for each transmitted Ethernet frame at the time-stamping units (TSUs).
- **WHITE RABBIT PTP CORE (*WRPC*) [136]**. Highly modular component and one of the central elements of the ZEN-CTA board, featuring a soft-core LM32 processor implementing the WR protocol stack, and custom implementations for an Ethernet medium access control (MAC) and physical coding sublayer (PCS) modules. It provides the deterministic timing distribution service, holds the internal time representation (PPSgen), and forwards ordinary data traffic to an external component (e.g., DMA, WR NIC) for its processing.
- **ADDITIONAL AMBA AXI BUS INFRASTRUCTURE (*AXI buses and crossbar interconnects*)**. The necessary components for implementing the additional bus connections required for interconnecting the elements of the high-speed data path between the Xilinx DMA modules and the processing system, such as the AXI-Full crossbar (“*XBAR*”) interconnect shown in Fig. 5.5.
- **AXI-STREAMING BUFFERING (*tx_buff_i, rx_buff_i*)**. The buffering elements for Ethernet frames while the system is busy with previous transactions or other tasks can hold up to 20 frames of MTU (Maximum Transmission Unit) size (1500 B), or even potentially accommodate *jumbo* frames, using 16k-word AXI-Streaming “first in, first out” (FIFO) buffers. These were implemented for both the transmission (MM2S) and reception (S2MM) interfaces of the Xilinx AXI DMA.

These elements are eventually combined to implement the DMA-based, high-speed data path upgrade for the WR-compatible interfaces of the ZEN-CTA board shown in Fig. 5.5. This design effectively bypasses the legacy data path of the WR NIC and instead forwards the Ethernet data frame transactions from the WR-compatible interfaces of the ZEN-CTA board directly to the main system memory of the Zynq-7000 system-on-chip (SoC) using the Xilinx DMA engine configured to operate with an efficient scatter/gather mechanism using a custom Linux kernel-level driver. This approach achieved an effective throughput of 426 Mbps (TX) and 564 Mbps (RX), and its development is explained in Appendix B.

5.5.2 *Ethernet FPGA Coprocessor for TDC Packet Transmission.*

The DMA-based network interface upgrade described in the previous sections provides a dedicated, high-speed data path that can be used for supporting intensive transmission of data from the ARM processing system. Since it is a requirement for CTA to sample and transmit time-stamped data originating from the reception of Cherenkov events over a high-speed network, this newly developed data path could be used to supply a feasible implementation of such a high-throughput sampling system over the WR synchronization network.

However, as indicated previously, this approach cannot fully utilize the entire capacity of the Gigabit Ethernet interface of the board. Moreover, user-level applications that transmit a large bulk of time stamps by intensively writing data into Linux network sockets are expected to consume a substantial amount of CPU time and incur considerable overheads as well, which would further decrease the effective throughput that a purely software-based solution could use for transmitting time-stamp data from the TDCs. As a result, we have proposed a fully gateware-based solution with a dedicated FPGA coprocessor leveraging the new high-speed data path for the WR-network interfaces that could overcome the aforementioned limitations, as shown in Fig. 5.6.

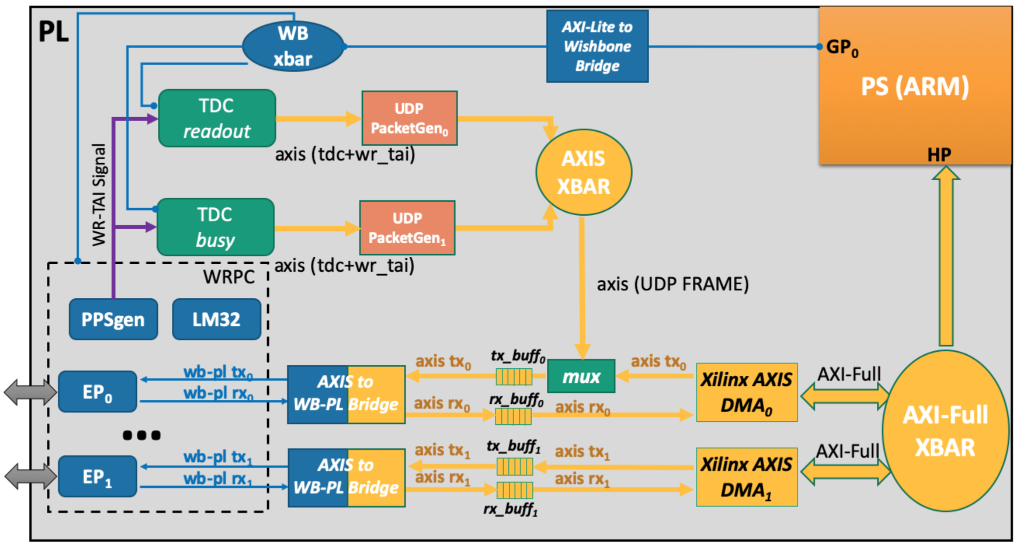


Figure 5.6
Simplified block diagram showcasing the integration of the FPGA coprocessor for forwarding time stamps into the high-speed TX path of the DMA-based upgrade for the WR-compatible Ethernet interfaces of the ZEN-CTA board. © **2020 IEEE**.

The new gateware-based solution is based on the implementation of a custom Packet Generator (*PacketGen*) module, which will compose specially formatted UDP packets for transmitting the time stamps generated at the TDC modules. These UDP packets will in turn be merged onto the main TX data path of the DMA-based upgrade by means of a simple AXI-Streaming Interconnect component, which will multiplex (*mux*) these UDP frames with the transmission transactions originating from the DMA onto the TX path leading to the *AXIS-to-Wishbone* Converter box, and then onto the WR End-Point. The Packet Generator module can be parameterized to encapsulate different numbers of TDC-generated time stamps in the payload of the UDP packets. This parameter can be adjusted to optimize the transmission of TDC-time stamps acquired at growing event-generation rates over the WR-compatible network to accommodate specific bandwidths or latency requirements, as discussed in Section 5.6.2.2.

5.5.3 *FPGA Resource Footprint*

This section summarizes the FPGA footprint that can be attributed to the implementation of the TDC modules, the DMA-enhanced data path for the WR-compatible network interfaces, and the UDP Packet Generator for transmitting TDC-generated time stamps over the network. The UDP Packet Generator can be parameterized to implement a number of different configurations, including lightweight ones that do not excessively tax on FPGA resources and that still achieve the level of performance required for CTA, as discussed in Section 5.6.2.2. Nonetheless, it should be noted that even the most demanding configurations for the architecture developed in this study can still be fitted into the Z-7015 device used by the ZEN-CTA board, as shown in Table 5.2, which considers that the most demanding configuration of the UDP packet generator (90 time stamps per UDP frame) is used.

Module	Slice LUTs		Slice Registers		Slice		BRAM		DSPs[2]		idelay		iserdes	
Xilinx DMA	2231	4.83%	3259	3.53%	907	7.85%	3	3.16%	0	0%	0	0%	0	0%
AXIS-to-WB Converter	231	0.50%	131	0.14%	100	0.87%	0	0%	0	0%	0	0%	0	0%
AXI Bus Infrastructure[1]	2419	5.24%	2649	2.87%	970	8.40%	11.5	12.11%	0	0%	0	0%	0	0%
WR End-Point	1518	3.29%	1442	1.56%	581	5.03%	2	2.11%	0	0%	0	0%	0	0%
WR NIC (legacy)	700	1.52%	975	1.06%	341	2.95%	3	3.16%	0	0%	0	0%	0	0%
WB MUX	99	0.21%	29	0.03%	42	0.36%	0	0%	0	0%	0	0%	0	0%
TDC Module	263	0.57%	256	0.28%	128	1.11%	2	2.11%	0	0%	8	5.33%	8	5.33%
PacketGen 90 TS	4429	9.59%	11724	12.69%	3132	27.12%	2	2.11%	0	0%	0	0%	0	0%
Total	***11890***	***25.74%***	***20465***	***22.15%***	***6201***	***53.69%***	***23.5***	***24.74%***	***0***	***0%***	***8***	***5.33%***	***8***	***5.33%***

[1] Includes necessary bus interfaces, interconnects and buffering elements for supporting the configuration of the data path as well as data transfers.
[2] Digital Signal Processor primitives.

Table 5.2
FPGA Resources (Z-7015 device) for the components implementing the WR-compatible interfaces for each port. **© 2020 IEEE**.

5.6 EXPERIMENTAL SETUP

In this section, we describe a series of experiments that were carried out for evaluating the proposed data acquisition capabilities for the enhanced ZEN-CTA board. The experimental setup is aimed at showing how the system processes and accurately time-stamps a number of analog signal triggers received at a substantially large rate using the TDC modules. The time stamps generated for these analog triggers are in turn transmitted over the network using the DMA-based data path upgrade and forwarded to a remote processing center over the WR-capable network.

As a result, two sets of experiments are conducted. Firstly, the time-stamping accuracy of the TDC module was characterized in order to verify that CTA requirements are met.

Next, the second batch of tests followed, to measure the highest-attainable bandwidth over the WR-capable network interfaces under different configurations when growing rates of analog signal triggers are fed into the TDC modules. Our results will be discussed and we will emphasize how this system can be applied realistically as the UCTS component of the CTA Camera.

5.6.1 *Basic Elements of the Experimental Setup*

The experimental setup used for carrying out the tests described in this section consists of three main elements (Fig. 5.7): a WR-Switch configured as a grandmaster clock, a WR-ZEN used as a precise pulse generator, and a ZEN-CTA board for accurately time-stamping the arrival time of the analog pulses fed into its TDC module. Their main features are outlined below.

- **WR-Switch** (WR Timing Grand Master). 18-Port WR Switch used as time reference for the experiment, featuring a Virtex-6 FPGA and an ARM CPU running GNU/Linux. WR Master or Slave roles for its ports can be configured on a per-port basis. Its external clock source, consisting of pulse-per-second (PPS) and 10 MHz inputs, was connected to the Morion MV89 Double Oven-Controlled Crystal Oscillator (OCXO) for greater time stability [90] [137].
- **WR-ZEN** (Time Provider). WR-capable node intended to act as time reference, featuring a Zynq-7000 programmable SoC: it runs GNU/Linux on its hardened ARM processor, and implements WR on its FPGA fabric. The node is bundled with the digital input/output FMC expansion module to generate clock pulses disciplined to the White Rabbit clock received from the network [138].
- **ZEN-CTA Board** (TDCs+DMA) (slave). Custom version of the WR-ZEN Board [121] implementing the UCTS component of the CTA Camera. It incorporates a TDC module to allow pulse time-stamping with precision in the picosecond range.

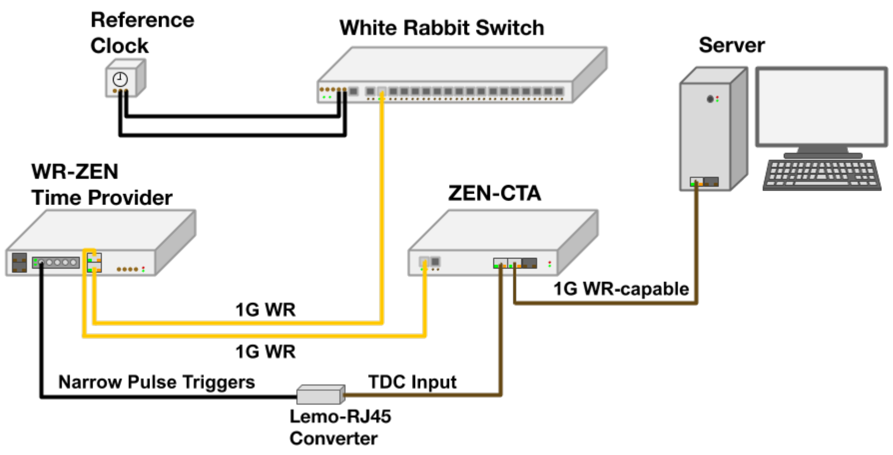


Figure 5.7
Diagram of the experimental setup used for conducting the experiments studying the accuracy of the TDCs (Section 5.6.2.1), as well as those comparing the attainable bandwidth for each different time stamp generation rate (Section 5.6.2.2). © **2020 IEEE**.

5.6.2 *Design of the Experimental Setup*

5.6.2.1 *Characterization of the TDC*

In this experiment we studied the time-stamping accuracy of the TDC module. This was accomplished by using the TDC module of the ZEN-CTA board to measure a periodic pattern of pulses generated with the Time Provider (TP) node. These elements are arranged in an experimental test-bed (Fig. 5.7) where elements have been synchronized using White Rabbit timing. The period of the pulses generated at the TP node is a user-configurable parameter that is adjusted throughout a number of iterations. The TDC module is used for measuring the actual time of arrival of the pulses in each case, and then determining the difference with their theoretically expected time of arrival. This will measure the time-stamping uncertainty of the system, whose impact will be determined by the combination of the effects of inaccuracies in the TDC module implementation and a systematic error attributed to uncalibrated WR devices, as will be explained later in the section. The PC server station has not been used for this experiment. As a result, we found that the average time of arrival uncertainty, considered as the Standard Deviation between the expected time of arrival for the analog pulses and that actually measured by the TDC, was in the order of tens of picosenconds (∼90 ps) and can be considered negligible. The following devices were used for conducting the experiment: the WR-Switch configured as grandmaster, and two boards, WR-ZEN and ZEN-CTA, linked in a daisy-chain layout. These elements are connected by means of small form-factor pluggable (SFP) transceivers and optical fiber cable. Synchronization across all the WR-compatible devices in the test-bed is ensured by monitoring their status information as well as their corresponding PPS outputs. Once the system is stable and a common time base has been distributed to all nodes using WR, then the TP is configured to send pulses to the ZEN-CTA with a user-specified period throughout a number of iterations. The TDC-measured time stamps are stored in a log file which will be used for analyzing the time-stamping accuracy of the system for each iteration. A number of iterations, each with a specific generation rate, were used in this study in order to measure the average time of arrival uncertainty that could be associated with the accuracy of the TDC module. Each iteration will use a specific generation rate and will inject 1000 analog triggers produced at the Time Provider node into the TDC input. As shown in Table 5.3, we found that the average time-stamping error remained stable throughout iterations in the expected range of tens of picoseconds.

These results are in accordance with the expected sub-nanosecond accuracy of the White Rabbit timing distribution system. It should be noted that the theoretical resolution of the TDC-generated time stamps is 125 ps; however, given the actual values used for configuring the delay taps (*idelays*) of the TDCs during implementation, and the synchronization error introduced by an uncalibrated WR system, a systematic time-stamping uncertainty has been introduced. This time-stamping uncertainty should be compensated for by using the appropriate calibration procedures for daisy-chain-type layouts [139]. As a result, the calculated jitter of ∼90 ps can be attributed to the expected WR synchronization jitter ∼[10-20] ps, and to the additional contributions introduced by the actual implementation of the TDC module (∼78 ps synthesizable step for *idelay*

Event No.	Event Rate (Hz)	Time Stamp Uncertainty [Std.Dev.] (ps)
1000	5	95.319
	10	82.703
	20	87.775
	100	94.744
	1000	88.392
	2000	90.531
	5000	95.660
	10000	81.398
	20000	83.244
	100000	85.984

Table 5.3
Time-stamping uncertainty variation for 1000 TDC-measured events. © **2020 IEEE.**

elements). This level of accuracy is in line with the expected performance required for CTA.

The foregoing results have been achieved with an FPGA TDC design that leverages the self-calibrated I/O elements of a Zynq-7000 device. In the literature, there are other studies that show other FPGA-based TDC implementations that also use this same structure to achieve high accuracy with minimized FPGA logic usage. Some notable examples are the studies in [112] (312.5 ps), the 31-channel KM3NET TDC implementation with oversampling and deserializers [140] (1 ns), or the TiCkS board [115] (2 ns). Contrary to these, other approaches have proposed designs that are built entirely over the FPGA fabric. They can still achieve reasonable levels of performance, such as the design with carry chains in [114] (17 ps) or the TDC detector for DIAPHANE based on ring oscillators ($\sim$200 ps) [141]; however, they require greater FPGA resources, careful synthesis, routing constraining, and complex calibration.

5.6.2.2 *Event Rate vs Bandwidth*

The following experiment is aimed at measuring the impact on the WR network of the transmission of TDC-measured time stamps generated at growing event rates by introducing pulse trains with different period values from the TP node, as shown in the setup described in Section 5.6.2.1. A software-based approach for this experiment, i.e., using the ARM CPU and the Xilinx DMA exclusively for transmitting the TDC time stamps, was ruled out as the intensive transmission of time stamp data generated at large rates would exert an enormous impact on CPU processing resources and network latency. Moreover, this approach would provide a maximum attainable bandwidth of 426 Mbps on the TX path, as indicated in Section 5.5.1, whereas the dedicated FPGA coprocessor for transmitting time stamps (Section 5.5.2) can easily overcome these limitations by utilizing close to 90% capacity of the Gigabit Ethernet link under select configurations, as opposed to the theoretical 50% utilization of the software-based solution. Thus, the *PacketGen* module was selected for this experiment.

Consequently, the UDP *PacketGen* module will be used for encapsulating time stamps into UDP packets in a raw representation format, which will then be sent to a remote server for processing (Fig. 5.7). As a result, it is expected that bandwidth demand and network utilization will be correspondingly impacted by the event generation rate configured at the TP node. On the server side, the transmitted time stamps can be processed using a custom application, and their bulk transfer statistics, such as bandwidth usage and absence of packet losses, are calculated using a generic network sniffing tool (*Wireshark*).

The experiment performs a series of iterations by sweeping the encapsulation settings of the UDP Packet Generator, which can bundle together up to 90 time stamps over an MTU-sized (1500 B) frame. The impact on the associated bandwidth usage for each configuration when different trigger generation rates are applied at the TP node is consequently studied (Fig. 5.8). The bandwidth usage figures presented for each generation rate were evaluated during 10 s windows, and hence the number of time stamps injected into the TDC in each case was 10 times the selected generation rate.

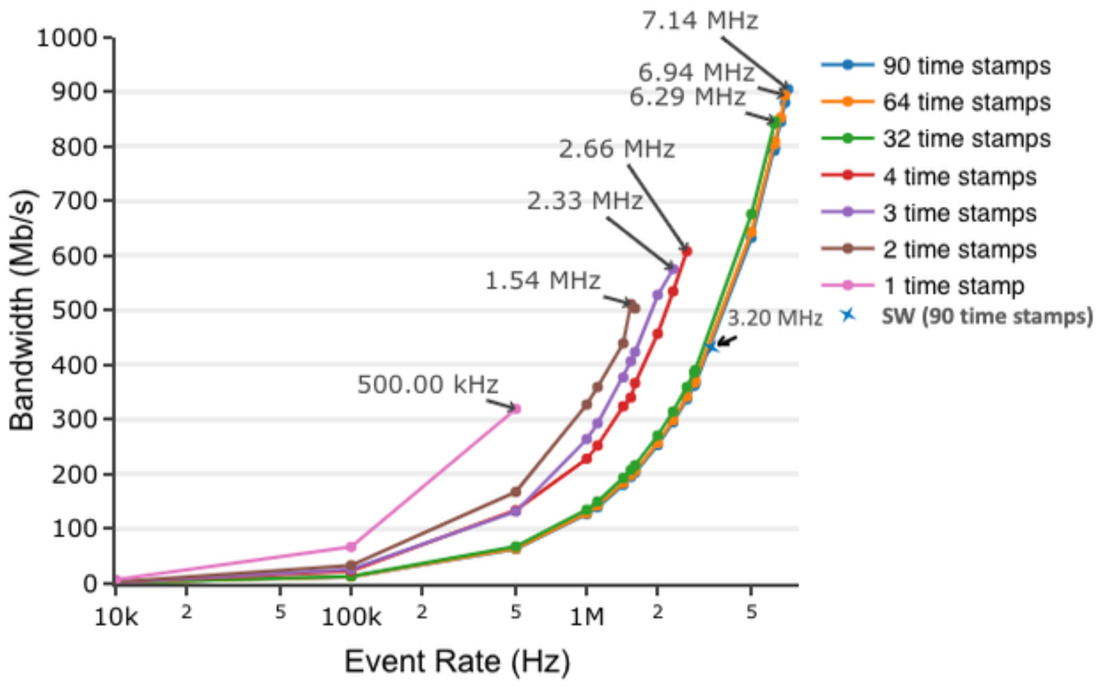


Figure 5.8
Plot of highest attainable bandwidth of the UDP messages before internal buffering congestion occurs when the TDC uses specific time-stamping encapsulation options for different event-generation rates. A comparison with the SW-based solution is also shown. © **2020 IEEE**.

Analysis of the data shows that doubling the size of the UDP time-stamp payload does result in a corresponding bandwidth demand drop. On the other hand, increasing the number of time stamps sent over the UDP payload has proved to be an effective strategy for capturing and successfully transmitting different events produced at growing generation rates, albeit at the expense of incurring greater transmission latency (Table 5.4) through the network. This additional latency should not affect the operation of the software-based, array-level trigger for CTA, which can handle end-to-end latencies lower than 100 ms, but should nevertheless be considered if new processing features were to be built into the system as part of a future development effort. The transmission latency of the UDP Packet Generator can be estimated using the following expression, which was derived from the implementation supplied for the module:

$$L(N, T_{samp}) = (N-1) \cdot T_{samp} + (4N + n_c) \cdot t_{clk} \tag{5.1}$$

In the preceding expression, N indicates the number of time stamps encapsulated in each frame, T_{samp} is the event generation rate period, n_c is a constant FSM (Finite State Machine) processing period of 26 clock cycles, and t_{clk} is the system clock cycle period of 16 ns. These results can also be seen in Fig. 5.8, which shows that the highest attainable event rate that can be transmitted over the network before buffering congestion occurs can be substantially increased by doubling the number of time stamps sent over the UDP payload. Hence, this upgraded data path provides a notable improvement (up to 905.6 Mbps) over the original capacity of the WR NIC-powered Network Interface Card design outlined in [131] (11 Mbps) by applying the upgrade presented in [132], which achieved $\sim$500 Mbps with a DMA-based solution.

Lastly, it should also be noted that UDP payloads of up to 90 time stamps could potentially use close to the maximum allowed link bandwidth of 1 Gbps when sampling events at the highest attainable rate of 7.14 MHz under this configuration, which vastly exceeds the expected trigger rate of the most demanding of the CTA telescopes (*LSTs* at 15 kHz, as indicated in Section 5.3). These results indicate that the system is highly versatile, given that it can be customized to conform to different applications that could potentially extend beyond CTA, where different bandwidth or end-to-end latency constraints may be required. Furthermore, the fact that this is a purely FPGA-based architecture that can be supported with the relatively modest Z-7015 device makes this an interesting reference design for high-performance, distributed event-acquisition systems.

5.6.3 *Outcome of the Experiments*

The preceding results indicate that the proposed solution is highly versatile and, given its large capacity for acquiring and forwarding event time stamps, it can safely handle Cherenkov light flashes generated at substantial rates. From the broader perspective of data acquisition systems, the design of the ZEN-CTA node is a robust and high-performance alternative to many data acquisition nodes from different projects as it incorporates high-accuracy TDCs ($\sim$90 ps resolution) assisted with a precise WR timing implementation. Furthermore, as it is built using a Zynq-7000 device from Xilinx, its FPGA-based design is coupled with a dual-core ARM processor with a Linux operating system, which simplifies the task of implementing additional software utilities or elements from acquisition control frameworks. Moreover, its enhanced Ethernet data path allows forwarding of events acquired at rates of up to 7.14 MHz. These features make our proposed design stand out against those found in other projects. In the context of CTA, the TiCkS board [115] is another alternative that uses WR synchronization with TDCs for time-stamping. However, its TDCs have lower resolution ($\sim$2 ns) and can acquire events generated at rates of up to 320 kHz. Furthermore, its design is based on the SPEC node

Event No.	Event Rate (Hz)	BW (Mbps) 1 TS	BW (Mbps) 2 TS	BW (Mbps) 3 TS	BW (Mbps) 4 TS	BW (Mbps) 32 TS	BW (Mbps) 64 TS	BW (Mbps) 90 TS
$5.00 \cdot 10^6$	$5.00 \cdot 10^5$	**320** (0.48 μs)	**168** (20.54 μs)	**132** (40.61 μs)	**134.4** (60.67 μs)	**68** (622.46 μs)	**64** (1264.51 μs)	**63.2** (1786.18 μs)
$1.60 \cdot 10^7$	$1.60 \cdot 10^6$	-	**504** (1.17 μs)	**424** (1.86 μs)	**367.2** (2.55 μs)	**216.8** (21.84 μs)	**206.4** (43.89 μs)	**203.2** (61.80 μs)
$2.18 \cdot 10^7$	$2.18 \cdot 10^6$	-	-	**576** (1.52 μs)	**535.2** (2.05 μs)	**315.2** (16.68 μs)	**299.2** (33.41 μs)	**295.2** (47.00 μs)
$2.66 \cdot 10^7$	$2.66 \cdot 10^6$	-	-	-	**608.8** (1.80 μs)	**360** (14.12 μs)	**342.4** (28.20 μs)	**337.6** (39.63 μs)
$6.29 \cdot 10^7$	$6.29 \cdot 10^6$	-	-	-	-	**846.4** (7.39 μs)	**809.6** (14.52 μs)	**797.6** (20.33 μs)
$6.94 \cdot 10^7$	$6.94 \cdot 10^6$	-	-	-	-	-	**894.4** (13.60 μs)	**880.8** (19.00 μs)
$7.14 \cdot 10^7$	$7.14 \cdot 10^6$	-	-	-	-	-	-	**905.6** (18.64 μs)

Table 5.4
Comparison between the event generation rate and the required bandwidth for transmitting time stamps when the *PacketGen* FPGA coprocessor is customized to encapsulate different amounts of TDC time stamps (*n TS*) over a UDP packet. In parentheses the transmission latency for each Event Generation rate under a given configuration of *PacketGen*. **© 2020 IEEE**.

[79], which lacks an integrated ARM processor and makes its integration with a control framework more complex.

Other projects, such as KM3NET [80] or DIAPHANE [116], also feature event-acquisition nodes with TDC time-stamping; however, in contrast to the ZEN-CTA node, their solutions are based on stand-alone FPGA devices, further complicating integration in control systems or their PCB design, and they have comparatively lower resolution: ∼1 ns and ∼250 ps, respectively. Moreover, each node from DIAPHANE needs to have its TDCs manually calibrated. Furthermore, KM3NET nodes can easily be integrated in acquisition systems with their support of WR timing, whereas the nodes from DIAPHANE rely on GPS synchronization (OPERA) for their deployment in larger geographical areas.

It is also worth noting that the prototype costs of DIAPHANE (Cyclone V), KM3NET (Kintex), TiCkS (Spartan 6), and the ZEN-CTA (Z-7015 with embedded ARM) should be comparable, given their use of analogous midrange and low-cost FPGA devices, whereas the TDCs in [112] and [114] are not intended for mass production and feature high-end Virtex FPGAs.

5.7 CONCLUSIONS AND FUTURE WORK

We have presented a novel implementation of a White Rabbit synchronization node with an enhanced time-stamping device – a *Time-to-Digital Converter* – targeted to the UCTS camera component for the CTA, but generic enough as to allow its use with other camera prototypes. Together with the compact dimensions of the board (*60 mm* × *164 mm*), the system implements a network of distributed astronomical event samplers using a paradigm that resembles that presented in [142] for distributed digitizers. The TDC module, given its FPGA-based design, is a low-cost solution that is able to achieve similar performance to that of dedicated hardware implementations in terms of accuracy, and it is also immune to the effects of varying temperatures or fabrication processes. Its accuracy has been characterized and pegged to the 90 ps range for "-1" speed-grade devices of the Xilinx 7-Series-based FPGAs, but this figure is subject to further improvement, potentially up to a threefold increase in accuracy, if faster grade devices are used.

The TDC module is then bundled with two different solutions to allow the forwarding of its time stamps to a remote server for processing: a software-based solution that relies on the new DMA-based data path for transmitting high-throughput data over the network, and an exclusively dedicated FPGA coprocessor for transmitting TDC time stamps. These two alternative solutions coexist in the architecture presented in the design and have been studied in a realistic laboratory test bench. It was found that the dedicated FPGA *PacketGen* coprocessor could support the continuous transmission of time stamps generated at rates of up to 7.14 MHz (905.6 Mbps throughput) with no impact on system performance, whereas exclusively using the DMA, software-based approach would only allow for a maximum continuous rate of time stamps of 3.195 MHz theoretically (426 Mbps throughput) and the system CPU resources would be under enormous strain. Thus, the *PacketGen* module is designated as the preferred method for forwarding time stamps from the node, even though pure software-based approaches could still be used when further flexibility is required (e.g., filter time stamps according to user-defined criteria) and the time stamp generation rate is lower.

The *PacketGen* is highly customizable and can be parameterized to accommodate different bandwidths and latency requirements. This leads to an FPGA footprint of the entire architecture that can be fitted into the relatively small Xilinx Z-7015 device in all possible configurations of the core. This device features a dual-core ARM processor with support for Linux and user-designed applications to allow the development of advanced functionalities on the ZEN-CTA node. Ultimately, this gives the proposed solution, which already boasts higher levels of performance than other similar alternatives [80, 115, 116], greater flexibility for integration. Furthermore, as the time stamp processing solutions presented in the design can handle transmission rates greater than those needed for CTA, this architecture could be targeted to a number of additional industrial or scientific applications that demand more stringent bandwidths and end-to-end latency transmission requirements. Some potential applications could be found in the distributed event-acquisition systems of large telescope arrays, particle accelerators, and other similar infrastructures such as KM3NET, EISCAT [143], or ICECUBE [144].

5.8 FOLLOW-UP TO THE ARTICLE. MOTIVATION FOR A MIGRATION TO TSN FUNCTIONALITIES

WR timing uses a standard Ethernet link for transmitting its synchronization protocol messages. Specifically, it relies on a 1000Base-X optical fiber link that adheres to the standard specification of a physical layer for Ethernet and is even interoperable with other non-WR-capable equipment. As a result, our WR-compatible nodes allow both time synchronization messages and ordinary Ethernet data to coexist over the same physical link without exerting any apparent impact on the performance of the synchronization system; i.e., the transmission of the PTP-based messages for WR timing would not be hindered by simultaneous data transmissions so long as there is enough capacity in the link to support all the flows that are being transmitted. Hence, provided that the WR protocol messages can be exchanged between both the WR master and slave devices without losing any critical signaling message for WR, then the synchronization protocol would proceed unimpeded. We have verified this in our experiments with the ZEN-CTA and also in the preliminary study from [132], where we injected ordinary Ethernet data at varying rates over the WR Ethernet links to find that not only could we transmit data using Gigabit speeds, but that we could also maintain sub-nanosecond synchronization at the same time. Likewise, our tests with the ZEN-CTA have pointed in the same direction: we can use the WR synchronization network of the CTA infrastructure for transmitting both synchronization data and the scientific time-stamp payload generated at the UCTS cards of the small-sized telescopes. This would enormously simplify the deployment and integration of the networking system for CTA.

In this context, since the SSTs require that we deliver time-stamp data within a specific deadline of 100 ms for the software array-level trigger of CTA to operate correctly, the only way that we could verify that this claim could be upheld was by gradually introducing growing rates of time-stamp data until we determined that the highest rate that our design of the ZEN-CTA could withstand. Our results showed that the highest time-stamp rate was reached when we transmitted packets with bundles of 90 time stamps. This allowed us to capture events generated at rates of up to 7.14 MHz (see Fig. 5.8) with an associated bandwidth 905.6 Mb/s, and a transmission latency of 18.64 μs. These specifications correspond to the worst-case of our system can still fulfill the requirements for the small-sized telescopes, which are much more relaxed as they are expected to receive events at rates of up to 600 Hz. This validation approach shows that the only way that we can guarantee a certain level of performance with a best-effort Ethernet network is by overprovisioning its resources and by estimating the worst-case performance as an implicit assurance of how the system would be expected to behave under normal operating conditions. As a result, it can be seen how an integration with TSN and WR could be beneficial for scientific infrastructure as well. TSN, whose design and implementation we study throughout this thesis, has all the necessary building blocks for providing a deterministic delivery service of critical data with bounded end-to-end latency and reliable delivery (e.g., seamless redundancy). This could complement the robust timing transfer of WR and ensure that the scientific payload produced at the ZEN-CTA could be delivered on time and with bandwidth assurances to the software array-level trigger at the central processing node of CTA. Consequently, an integration of TSN with WR timing is a subject that we explore in Chapter 10 after presenting TSN architecture and our experimental results for industrial and aerospace use cases.

Part II

Design and Implementation of a TSN Networking System

6 THE CONSTRUCTION OF A TSN NODE

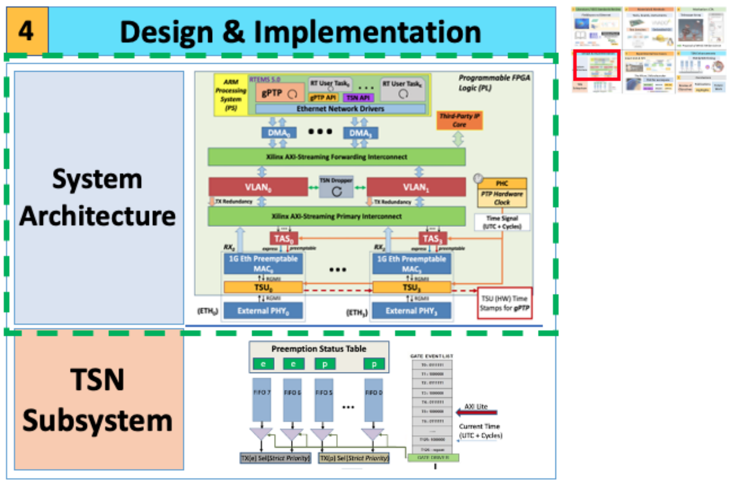

4
Design & Implementation
System Architecture
TSN Subsystem
Programmable FPGA Logic (PL)
Ethernet Network Drivers
Xilinx AXI-Streaming Forwarding Interconnect
Xilinx AXI-Streaming Primary Interconnect
Preemption Status Table
AXI Lite
Current Time (UTC + Cycles)

Figure 6.1
Overview of the contents of Chapter 6 for the structure of our TSN nodes.

This chapter presents a system-level view of all the elements that come into play for the implementation of a TSN node. Thus, we present the general system architecture that we have extensively used for our nodes. In this context, we introduce the main elements of the software environment, including the use of an embedded OS and a synchronization service, such as gPTP or *ppsi* (for WR timing). Next, we introduce the embedded platforms that we have used for developing and prototyping during the thesis project: the Zynq-7000 family of programmable SoCs from Xilinx. These devices feature embedded ARM processors for the running of our software environment, and the FPGA logic that we used for building our TSN solution. Hence, after introducing the embedded platform, we move on to presenting the main components of our FPGA implementation for TSN. These include the cores for the networking subsystem, the timing subsystem, the switching elements, and the TSN subsystem itselft. We provide an overview of their operation and conclude by emphasizing the important relationship between the timing system and the TSN cores for achieving a deterministic forwarding of the critical data.

Chapter contents

6.1 GENERAL STRUCTURE OF A TSN NODE

The process of designing a TSN node consists of the integration of several enhancements for bringing about the capacity to forward traffic deterministically on top of ordinary Ethernet network links. Hence, TSN nodes are built on the foundation of a simple Ethernet service providing the basic support and functionalities for data forwarding, which is enhanced with the integration of two key additional components: a timing service and a TSN subsystem. This latter element is a collection of traffic shapers and packet filters that are ultimately responsible for providing some of the main features of the TSN system. The generic hardware structure of these devices can be examined in the diagram of Fig. 6.2, which also shows some of the required software elements for handling and configuring the operation of this type of systems.

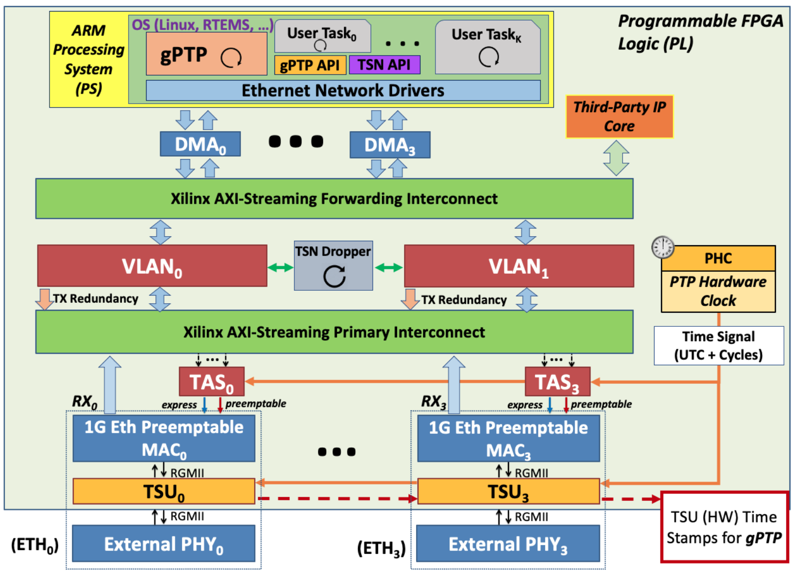


Figure 6.2
The basic hardware and software architecture integrating all the necessary components for building a functioning TSN system.

As stated in Chapter 4, the designs we have worked on are based on the Zynq-7000 programmable SoCs from Xilinx, given the convenience and flexibility afforded by the use of an FPGA-based platform for developing and verifying the inclusion of new functionalities. Hence, the primary elements supporting the operation of the TSN system have been implemented directly on FPGA logic and are enumerated in the following points.

- **The Networking subsystem** is the underlying Ethernet communication service that allows the transmission of data in a TSN system. Hence, it is composed strictly of the off-the-shelf, standardized components that are required for instantiating a 1-Gb/s Ethernet link layer service. In our research, given our choice of a Xilinx-based development environment, we have implemented this feature with the main IPs and design blocks for Ethernet communications of the Vivado environment. Hence, this subsystem has mainly made use of DMA blocks for data transfers [145], PCS/PMA cores [146] for interfacing with the physical layer, the medium access controllers (MAC) from Xilinx for 1-Gb/s Ethernet [147] or custom open

MAC designs [148], and the appropriate Ethernet transceivers (PHY). These latter elements could be the built-in GTPs [149] from the Zynq-7000 devices we have worked with, or even external PHY chips (Chapters 8, and 9).

- **The timing subsystem** is one of the essential components for any working TSN implementation. It is responsible for ensuring the distribution of a common time reference across the network to allow the synchronous operation of the different TSN nodes and their respective traffic shapers. Its operation has been standardized in the IEEE 802.1AS specification [6], which defines a specific PTP [77] profile for TSN networks: the generic PTP (*gPTP*). This synchronization method makes use of three main hardware components that can be instantiated either as FPGA logic or as separate elements, such as application-specific integrated circuits (ASICs) on the printed circuit board (PCB) of the node. These components are the time-stamping unit (TSU), the PTP hardware clock (PHC), and a digital phase-locked loop (DPLL). The TSU, as implied by its designation, is tasked with producing time stamps on the reception and transmission of the PTP protocol messages. These time stamps are in turn used by the synchronization algorithm to calculate the link delay and compensate the master-to-slave offset. This offset compensation is applied to the PHC, which maintains the internal time representation of the local node. In our designs, both the TSUs and the PHC were implemented as FPGA IP cores. In contrast, the DPLL is usually implemented on the PCB as a voltage-controlled oscillator (VCO) for a fine-grain phase compensation of the PHC. It is steered by the gPTP algorithm after the initial coarse phase adjustment stage of the protocol.

- **The TSN subsystem** is tasked with the implementation of the traffic-processing enhancements that allow the deterministic forwarding of the higher priority messages of the system. As outlined in Chapter 3, TSN devices can make use of the features defined in the multiple specifications of the IEEE 802.1 family of standards to achieve this goal. Our design features a reduced subset which we have selected from the aforementioned standards that has allowed us to implement a working system with the minimally required functionality of TSN. Specifically, our designs feature support for traffic identification and reservation modules (802.1Q [46]), time-aware traffic shapers (TAS - 802.1Qbv [9]) with preemption (802.1Qbu [41]), preemptable Ethernet MACs (802.3br [40]), and seamless redundancy (802.1CB [44]). The traffic identification and the redundant transmissions are supported with a VLAN core, the deterministic forwarding is achieved with time-aware traffic shapers operating synchronously to the PHC, and the enhanced robustness of the system is provided by the extensions for redundancy (VLAN and Dropper modules) and preemption (TAS and the Ethernet MAC). Furthermore, we have implemented these modules as FPGA cores with several customization options for streamlining their integration with other components or in smaller FPGA devices.

- **The switching elements** are tasked with forwarding data frames between the different ports of our platform, or between its Ethernet ports and the ARM CPU. Thus, these components enable the use of the bridging and packet-switching functionality that we have built into our nodes. This functionality is essential for the role of our nodes as bridging elements, as they allow any incoming flows into our platforms to be forwarded to either a consumer IP core built into the FPGA logic, or towards the egress path of the node so that it can be forwarded to a neighboring node. In the architecture depicted in Fig. 6.2, the switching elements are implemented using the proprietary Xilinx crossbar switches [150] for the AMBA AXI-Streaming system bus, which allowed us to quickly and conveniently build the packet-switching feature for our nodes. Their operation, and hence the bridging behavior of the nodes, is in turn controlled through the configuration parameters that the system users supply to the VLAN module. Furthermore, to avoid the runaway resource-consumption issues associated with the use of large crossbar switches, we split their implementation into two main functional block: a primary switching block for the ingress data path and for interfacing with the TAS shapers, and a forwarding switch – "*the Redirector*" – for forwarding data between ports or towards third-party IP cores in the FPGA logic.

The foregoing elements represent the basic, generic template for building a TSN system. We have combined and arranged them with different layouts and varied interconnectivity options to assemble several types of FPGA architectures. These variations were developed to conform to some of the requirements that we encountered throughout the implementation of this thesis project that demanded that a moderate-resource FPGA footprint be provided that would also allow for integration with other third-party IP components. These cases, alongside their corresponding architecture, are presented in detail in Chapters 8, 9, and 10.

6.2 THE USUAL ROLES OF OPERATION OF TSN NODES

TSN nodes can take on several roles on the network depending on their relationship to the data source, or their role in the flow routing and its processing. Thus, TSN nodes are usually designated either as ***talker*** or ***listener*** devices. The former refers to the producers of the data streams that are forwarded over the network, whereas the latter indicates the end system that consumes the data. The data transmission model of TSN implies that all data flows have a specific origin at a producer "*talker*" node and are then routed on a point-to-point basis towards their "*listener*" destination. The nodes in the network may possess both roles interchangeably, as they may be both producers and recipients of different types of TSN data streams.

Another possible classification which has profound effects on the overall architectural design of the node is its relationship to the forwarding of data and their processing. Thus, the devices that are located at the edge of the network are usually either gateways from an ordinary Ethernet system or the end systems operating as *talkers* or *listeners*. These nodes may also implement the interfaces with the sensors and actuators of a specific control loop, or they may aggregate and process data from different sources (e.g., sensors) to produce the corresponding control messages. Hence, these types of nodes are often referred to as an "**end-point**" for receiving, originating, or processing the different data flows.

These TSN flows, when forwarded over the network, have to travel through several intermediate nodes. These nodes have the task of identifying the different types of TSN streams and deciding the physical port they should be sent to so that they can reach their destination. As these devices need to include switching and multi-port packet forwarding capabilities, they are often referred to as ***bridges***.

Consequently, these functionalities place different demands on the expected features of each type of node. These differences lie along the following lines.

- **The number of ports** is one of the most obvious factors influenced by this distinction. The *end-points* have a tendency to be relatively simpler devices and hence they are usually either single- or dual-port systems. However, they may feature other analog or digital interfaces for interacting with other control or sensing equipment. In contrast, since the TSN *bridges* are meant to forward data at the intermediate points of the network, they are usually multi-port devices.
- With respect to **timing synchronization and frequency distribution**, all the TSN nodes should be able to support a synchronization service. However, the *end-points* will usually operate as either synchronization master or slave devices, whereas the *bridges* will have their ports take on both roles as required by the best master clock algorithm (BMCA). Thus, TSN bridges will usually have one of their ports operate as a timing slave – the port connected to the timing source of the network – whereas the rest of their ports will behave as timing masters for propagating the time reference to devices in subsequently lower layers of the network. In addition, when the ports of a bridge node are connected to multiple time sources, the BMCA will set configure one single port as a timing slave whereas the rest will be designated as fall-back *passive* ports.
- The **switching capabilities** of each type of node will also differ in accordance with their roles. The *end-points* may often lack any switching capabilities given their use as data sources or sinks, thereby requiring a simpler FPGA architecture. On the contrary, the TSN *bridges* will require internal switching elements (e.g., crossbars) for forwarding data and their architecture and resource usage will be correspondingly more complex.
- There are also differences with respect to **the software environment**. Thus, both the TSN *bridges* and *end-points* will require basic software support consisting of an embedded OS, configuration APIs for the TSN and timing systems, and a timing service (e.g., the gPTP "daemon"). However, the *end-points* may add more complexity with different user-level utilities or even real-time tasks for processing the TSN data flows (e.g., the implementation of the software logic for a control loop).

In our designs, even though we have worked with both multi-port and dual-port devices, all of our nodes can potentially behave as TSN *bridges* with the processing capacity of an *end-point*. In this context, the WR-ZEN board, which is a dual-port device, will often take on the role of an *end-point* in our experiments, although, as it was chosen as the main development platform, it can also redirect data between its ports and behave as a bridge with built-in switching elements. This can be seen in the experiments presented in Chapters 8, 9, and 10. We have also developed a four-port TSN switch – the Main Board – that behaves as a bridge with real-time processing capabilities which is introduced in Chapters 8 and 9.

6.3 THE SOFTWARE ENVIRONMENT OF A TSN NODE

As shown in Fig. 6.2, the software environment of a TSN node consists of an ecosystem composed of an embedded OS for basic system management tasks, the appropriate Ethernet drivers for data communication, a synchronization utility deployed as a system service, configuration APIs for regulating the operation of the timing and TSN modules, and the appropriate user-level tasks for data processing.

6.3.1 *The operating system*

Our nodes make use of embedded operating systems given the convenience and simplicity afforded by their use for managing complex operations; such as user-task scheduling with potentially real-time constraints, or the instantiation of communication interfaces with multiple communication protocols. We have used two different types of OSes during our experiments: a general-purpose embedded Linux OS [151] and a real-time RTEMS [152] implementation. Both OS images were supplied by our industrial partner Seven Solutions so that we could customize them for our experiments. The former has been used for the development and the testing of novel features, as presented in the White Rabbit and TSN integration experiments of Chapter 5 and 10. The latter is a real-time OS commonly used throughout multiple embedded scenarios and for avionics systems in particular. Its use has been showcased in the experiments of Chapters 8 and 9. Moreover, regardless of the scenario under consideration, the OS is the background on top of which the different services and utilities of our nodes have been integrated. These features are presented in the following points.

6.3.2 *The device drivers*

The device drivers are the kernel modules that allow the OS to interface and control the underlying hardware and peripherals that are present on a specific platform. The use of kernel modules is fairly common in general-purpose operating systems such as Linux. Indeed, Linux implements a modular kernel approach whereby the user can choose to customize the kernel image with different modules depending on the capabilities and desired behavior of the targeted platform.

The large diversity of drivers and kernel modules for Linux and Unix-like kernels is explained as a consequence of the multiple embedded platforms available for designers to choose from, with each one having different sets of processing elements or peripherals. Hence, the existence of multiple kernel modules helps detach the main system kernel from handling the complexity of integrating the numerous hardware variants of the different computing platforms supported on a specific OS. Rather, this approach allows the use of a simplified kernel featuring generic system interfaces (e.g., system calls, driver hooks, . . .) that communicate with the appropriate device drivers and modules handling the low-level tasks of the underlying hardware of a given computing platform.

The Zynq-7000 devices we have used throughout this project are no different in this respect. They are embedded platforms based on ARM processors and are thus supported on multiple operating systems, such as Linux or RTEMS. We have used the *Buildroot* [83] tool, which is a well-known tool suite for generating custom OS images for embedded devices, for producing the kernel images we have used in our tests. Hence, *Buildroot* provides a streamlined interface for customizing the kernel with a choice of system modules and other configuration parameters that the designer can choose to activate using an *ncurses* [153], graphical, or plain-text configuration interface to modulate the behavior of the system. The user can then choose the desired modules and drivers for a specific functionality and have them linked into the compiled kernel or generated separately as loadable kernel modules. Loadable modules can be dynamically loaded by the user with commands such as *insmod* or *modprobe*. This is a very powerful mechanism for producing system images tailored to the hardware of custom embedded platforms, like that present in FPGA-based devices. Typical examples of this level of customization include the choice of the specific TCP/IP communication stack (e.g., TCP Tahoe, FAST TCP, TCP Cubic, . . .), the drivers for interfacing with the CPU hardware clock, or the network drivers for managing the Ethernet interfaces.

Furthermore, the *Buildroot* build system also generates a crucial element for handling device drivers: a device tree. Device trees are useful for indicating which drivers should be loaded or the specific parameters that a given module should use during initialization, such as the base address of the hardware being controlled in the memory space of the system or the driver version that a given device is compatible with.

Our custom embedded images have made use of both statically and dynamically loaded modules for their operation. The static modules include the basic framework for supporting the execution of Linux on the ARM processors of the Zynq-7000 devices. Most of them are already selected in the *Buildroot* configuration template for the Zynq-7000 platform. User-designed drivers can also be added to the workflow of the *Buildroot* build system to support the custom IP cores in our FPGA designs with new device drivers. That has been the case with the network drivers in many of our TSN prototypes. Moreover, we had them compiled as separate kernel modules that could be loaded dynamically when needed; i.e., when flashing the FPGA with the appropriate bitstream.

Specifically, in order to support some of the additional functionalities required by our high-speed, TSN-capable interfaces, we had to develop new network drivers, or modify (i.e., *patch*) existing ones, to establish the underlying Ethernet service required for exchanging data. The design of a fully functional network driver from scratch can be a daunting task, and hence we have used several methods for supporting a 1-Gb/s Ethernet service on top of our FPGA designs with incremental modifications to existing drivers.

- For the WR-compatible Ethernet interfaces of Chapter 5, we made use of a modified version of the WR NIC driver [131], which is a part of the White Rabbit resources from the Open Hardware Repository [136]. This driver was then coupled with a repurposed version of the Xilinx AXI DMA driver [151] through the use of the *DMAEngine* kernel API. Although this allowed our design to benefit from the use of a DMA core to boost the forwarding data rates (up to 428 Mbps), it was also an *ad hoc* implementation that required that a kernel update – a patch – be rolled out for the *DMAEngine*. This version has since been superseded and replaced for simpler alternatives in the subsequent experiments of Chapters 8, 9, and 10.

- The experiment of Chapter 10 uses a new flavor of the WR architecture available for the WR-ZEN devices of Seven Solutions S.L., which were the main development platform used for this thesis project. This new architecture supplied a simplified design that replaced the original WR-NIC and WR Ethernet MAC from the Open Hardware Repository with simpler, "off-the-shelf" cores from Xilinx: the Xilinx Ethernet subsystem [154]. Hence, we used a modified version of the corresponding Xilinx drivers (*axienet*) for implementing our Ethernet network drivers in this case. Seven Solutions supplied a version of these drivers for their WR-ZEN platform, and we had to adapt them to allow simultaneous data transmissions alongside the WR protocol messages for our experiments integrating TSN and WR timing.
- The FPGA architectures from Chapters 8 and 9 also use a similar implementation of the Ethernet subsystem to that of Chapter 10. Hence, even though the OS of choice for these experiments was RTEMS, the Ethernet drivers were also implemented as a modified version of the Xilinx Ethernet subsystem drivers (*axienet*); however, they had to be ported to RTEMS as well. The ported Ethernet drivers for the RTEMS environment were supplied by our industrial collaborator Seven Solutions.

6.3.3 *The configuration interfaces*

The operation of a TSN system is eventually determined by its configuration; i.e., the settings the end user may supply for the TSN network in order to attain a specific effect or a desired behavior. Most of the time, these effects are related to ensuring the deterministic delivery of the so-called critical data flows within a tight reception window to the detriment of the lower priority traffic classes such as best-effort video. It can be seen that this behavior can only be achieved by the careful definition of a set of configuration parameters that span the routing topologies in the network, the traffic filtering, or the traffic shaping policy on the egress paths of the TSN bridges. Although the correct calculation and derivation of these parameters is paramount if a certain behavior is to be expected of the system, their derivation is beyond the scope of this study. Nonetheless, the procedures for working out meaningful configurations for TSN systems are a broad field of research where a variety of methodologies can be applied for finding efficient solutions for this problem. The literature shows that there are interesting approximations for this that range from the modeling of the system with network calculus [155] to the application of metaheuristics [156].

Hence, provided that the user settings have been determined by an appropriate method, the configuration interfaces are then responsible for uploading these parameters to the TSN nodes and their IP cores so that the desired behavior of the system can be achieved. This has been implemented in the form of two separate application programming interfaces (APIs): the TSN API and the gPTP API. The former is tasked with supplying the user configuration parameters to all the TSN elements of the node, such as the GCL schedules for the traffic shapers, the traffic identification criteria and routing settings for the VLAN module, or the critical flows that will be subjected to enhanced protection through redundant forwarding. The latter is concerned with adjusting the operation of the gPTP synchronization; e.g., the message exchange rate, or the corresponding port roles of a given node for adjusting the behavior of the best master clock algorithm (BMCA).

We have implemented these APIs as complementary libraries that get linked into our software environment as needed. In the case of Chapters 8 and 9, both the TSN and gPTP APIs were developed as separate kernel libraries, whereas the experiment from Chapter 10 replaced the gPTP library for the corresponding functions and interfaces for managing a WR implementation.

6.3.4 *The TSN API*

The TSN API consists of four basic elements: *a)* a set of configuration data structures describing the applicable settings for the traffic shapers and the VLAN-tagging modules, *b)* the high-level user interface for configuring the TSN node, *c)* a low-level interface with the traffic shapers for uploading GCL schedules, and *d)* another interface with the VLAN module for designating the traffic classes in the system and the operation mode for redundancy. This API was jointly developed with our collaborators from Seven Solutions and the Andalusian Institute of Astrophysics (IAA). The relationship between *b)*, *c)*, and *d)* of the API can be observed in Fig. 6.3.

a) The API data structures are defined in the configuration header file (*config_file.h*), which contains the data models for specifying the behavior of the TSN IP cores. Hence these structures allow the definition of GCL schedules, preemption status, use of redundancy, and VLAN-tagging rules on a per-port basis.

b) The high-level functions allow the user to interact with the TSN IP cores after their configuration has been supplied through the data structures of the configuration file (the *config_file.h* header). These functions allow the user to select the TSN-capable Ethernet port the new settings will be applied to, and then upload the corresponding parameters to the TAS and VLAN modules associated with the port under configuration.

c) The low-level interface with the traffic shapers of the system contains all the necessary functions for accessing the control and configuration registers of the TAS IP core (7.3.2) in its embedded AXI slave bus interface. These functions can write the necessary configuration values for adjusting the behavior of the core according to the schedule specified by the user or the desired preemption status for each queue of the shaper. Moreover, it also contains a main configuration function for programming the core that writes the user settings to the TAS registers following the sequence expected by the TAS internal configuration FSM.

d) Likewise, the low-level interface with the VLAN module contains the functions for accessing the configuration registers of the VLAN core (see 7.1.3). These functions can read and write configuration values from the embedded configuration registers of the AXI slave controller of the core, such as the VLAN tag contents of a given TSN stream or whether redundancy is used. In addition, it contains a main configuration function for uploading these settings to the core in keeping with the expected configuration sequence of its internal configuration FSM.

6.3.5 *The user applications*

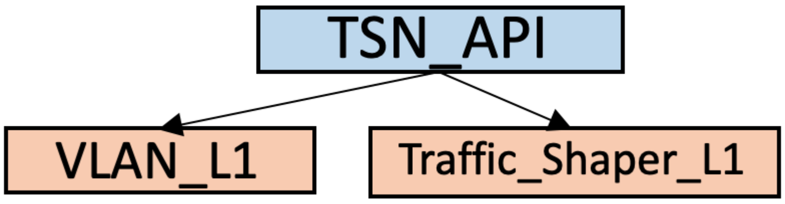


Figure 6.3
The main components of the TSN configuration API.

New user-level applications can easily be integrated in our nodes with the usual *Buildroot* workflow. Thus, its build system also allows the generation of customized file system images that can be fitted with a number of user utilities. These utilities can either be third-party tools that the build environment of *Buildroot* can retrieve from remote content repositories, or they can also be custom applications that the user can include in the cross-compilation build process of the system.

Hence, as indicated in the developer manuals of the *Buildroot* tool suite [83], developers have a wide range of configuration choices for building an embedded environment that suits the platform they are targeting. This includes kernel modifications, as indicated in previous sections, and the contents of the file system image. The latter can be adjusted through the configuration interfaces of *Buildroot* itself (e.g., *ncurses*), which allows the selection of the applications that will be included in the cross-compilation build. Furthermore, developers can include their own packages in the build process by supplying their own code and a specific *Makefile* for *Buildroot* (the ".mk" file). In addition, the code for simple programs can also be cross-compiled and added to the file system image overlay generated by *Buildroot*.

Our TSN nodes have been fitted with the usual applications expected in an embedded Unix-like environment that also doubles as a development and debug platform. Hence, these utilities include an SSH service, network debug tools such as *tcpdump* or *tshark*, Ethernet management resources such as *ethtool*, or network benchmark utilities such as *iperf*. All of the foregoing items can easily be selected from the Buildroot configuration menus for their inclusion in the file image. Additionally, we have also added our own packages to support the main functionalities of a TSN system. These include the following elements:

- The timing service has been implemented with the "ppsi" [157] utility. This application instantiates a WR timing service and has been used for the system images used in our experiments with WR synchronization (Chapter 5) and those combining WR and TSN features (Chapter 10).
- A configuration utility for adjusting the operation of the system has also been implemented as a custom package. This utility was jointly developed with our industrial partner Seven Solutions. Hence, this package leverages the TSN configuration API to produce a simple application that uploads system settings to the appropriate TSN IP cores. Its use has been featured throughout all of our experiments.
- A number of third-party packages that are part of the default design of the WR-ZEN system image from Seven Solutions. These packages help control the operation of

the WR timing service by supplying parameters such as its link calibration values, or the initialization of the WR-ZEN board, which makes use of a simple tool to interface with a configurable PLL chip (AD9516 [158]) on the PCB of the node for generating the required clock sources.

- Other user applications that were added to the WR-ZEN image through the *Buildroot* tool were the message generation utilities that we have used during our experiments to probe the performance of the TSN system with the emission of test data streams. Hence, these utilities have consisted of simple programs for producing UDP messages with different generation rates and header values so that they could be identified and tagged as required as TSN streams for studying the operation of a given TSN network.

Yet, some considerations should be made about the experiments of Chapters 8 and 9. Since these experiments have made use of an RTEMS platform, the build process for integrating third-party or user-designed applications differs from that presented in the foregoing points, which is intended for embedded Linux environments. RTEMS has its own build environment, which is called the RTEMS source builder (RSB) [159]. Hence, we have used the RSB for integrating all the aforementioned user applications, except for the "ppsi" module for WR as our experiments with the RTEMS OS only used the gPTP synchronization for TSN. The gPTP synchronization service for RTEMS was jointly integrated during collaborative research projects (Chapters 8) and 9) with Seven Solutions S.L. on the WR-ZEN development platform and on the Main Board.

6.3.6 *The timing services*

The timing distribution service is essential for guaranteeing the correct operation of any TSN network. In the context of this thesis project, we have worked with two different synchronization mechanisms: gPTP timing and White Rabbit. The former is the synchronization protocol that is included in the standard specification of TSN systems and was developed by our collaborator Seven Solutions in the framework of a technology transfer project. The latter is the high-accuracy timing and frequency distribution protocol from CERN that is built into the WR-ZEN boards from Seven Solutions. WR enhances the precision of PTP synchronization to allow its use for scientific applications such as high energy or particle physics. We have made an effort to document them so that their role in the TSN nodes can be understandable to the reader.

The gPTP synchronization service has been used in the experimental cases that we have showcased in Chapters 8 and 9. Specifically, the implementation of gPTP that we have supplied is an adaptation of the *Open Avnu gPTP project* [52], which was migrated to the RTEMS OS environment that we used for the aforementioned experiments. Hence, this implementation of gPTP for the RTEMS OS consisted of three major components: the gPTP protocol itself, its configuration interface, and a debug utility.

6.3.6.1 *The gPTP protocol implementation*

The gPTP protocol was implemented as a cyclic task for RTEMS with the structure shown in the diagram from Fig. 6.4. Its main components are the networking libraries (*net*, *sys/socket*) for interfacing with the TSN-capable Ethernet ports of the node, the main protocol state machine and the internal variables of the process (*commom_port_port_entity*, *linux_hal_generic*), and the configuration interface (*gPTP_API_L1*). In addition, the gPTP service had to include a set of specific libraries for interfacing with our custom FPGA hardware. These include the interface with the low-level configuration registers of the PHC (*PPS_GEN_REGS*), and that with the VCO controller for the fine-grained frequency adjustments *HWClk_MiniAPI*. Moreover, the service is intended to follow a two-tiered execution methodology. Thus, on the first stage, the protocol needs to be initialized. This step generates all the necessary structures of the protocol and must compulsorily be run before launching the main synchronization service. The second stage follows the initialization and consists of the cyclic execution of the gPTP service with all the components depicted in the diagram.

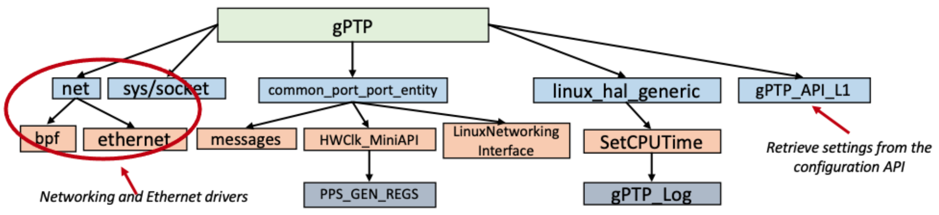


Figure 6.4
The main components of the gPTP synchronization service used for the TSN nodes built with the RTEMS OS.

6.3.6.2 *The configuration API*

The configuration API (gPTP_API_L1) is one of the complementary components to the synchronization service. It is intended to allow the user to supply the set of configuration parameters governing the behavior and execution of the protocol. Typical examples include the "*announce/sync*" message emission rate, which modulates the behavior and efficiency of the protocol, or parameters such as the *clock class*, the *clock identifier*, or its *priority*, which guide the operation of the best master clock algorithm. We have developed a streamlined interface allowing the user to conveniently supply all of these parameters before the gPTP service can be launched.

6.3.6.3 *The debug and supervision component*

The other complementary element to the gPTP service is its built-in debug and supervision process: the "gPTP_Mon" (gPTP Monitor). As the name implies, this is the process tasked with gathering traces and statistics from the execution of the synchronization service and then displaying them in a user-readable format. In order to reduce unnecessary overheads, its use is only recommended for debugging and troubleshooting purposes. Hence, the gPTP Monitor generates a comprehensive execution trace that system developers can use to conveniently track the performance of the protocol or help them locate the source of any issues arising during its execution. Moreover, it is highly versatile as well, since it allows the displaying of the execution trace results both locally on the user shell or remotely on a debug computer, where the results of the real-time trace on another node can be transmitted.

6.3.7 *High-accuracy timing services*

We have also worked with WR timing in some of our experiments, such as those described in Chapters 5 and 10, where we first intended to study the performance of this alternative synchronization method in the former, and then show its realistic application as an enhancement to a TSN system in the latter. Thus, the White Rabbit timing service is available on the WR-ZEN development board as a background Linux service, which is designated as "ppsi" and was integrated in the system image as a third-party package with *Buildroot*. The "ppsi" service is part of the collection of resources for White Rabbit from the Open Hardware Repository [157]. In this case, we have used an adapted version from Seven Solutions S.L. [160] that runs on the ARM processor architecture. Moreover, as is usually the case with analogous timing services, the "ppsi" is bundled with a host of supporting utilities and services apart from the White Rabbit synchronization itself. Hence, this implementation of "ppsi" makes use of additional user libraries for interfacing with the FPGA timing cores (e.g., TSUs, PHC) or for fetching the link calibration parameters [161] that are required for the correct operation of WR timing. Furthermore, it also features a lightweight debug mode in the style of that of our implementation of gPTP.

6.4 DESIGN OF THE NETWORKING SUBSYSTEM

The networking system is responsible for establishing the basic Ethernet communication service required for transmitting data frames from our TSN nodes. In the context of the system architecture from Fig. 6.2, it consists of the set of components that allows the movement of data frames from the ARM processor of the node and between its Ethernet ports, where they would eventually be injected or retrieved from the network. Hence, the Ethernet subsystem serves a crucial role in a TSN node, as it allows data transmission and reception from the user-level applications of the processor as well as data forwarding amongst its ports. In our implementation, we have used the following elements as its basic building blocks.

6.4.1 *The Xilinx DMA Engine*

The Xilinx DMA Engine [145] is an FPGA core implementing the functionality of a DMA controller. It takes on the role of a bridging device between the main system memory of the processor and the stream-oriented interfaces of the Ethernet MAC. Thus, it operates by transforming the data transactions from the *AXI-Full* memory-mapped bus domain that connects to the main system memory into data streams over the simpler *AXI-Stream* bus, which are bound to or originate from the MAC controller. Thus, in accordance with the Xilinx documentation, this IP core can be thought of as an integration of two simpler AXI Data Movers, which implement the bridge between the memory-mapped and the stream-oriented bus domains, a module containing the usual *AXI-Lite* configuration registers of the core, and an additional scatter/gather (SG) module to further offload the ARM CPU and handle high-throughput transactions more efficiently.

6.4.2 *The Ethernet medium access controller*

The Ethernet MAC is one of the central elements of the system, as it handles the access to the transmission medium of the Ethernet network. Specifically, the ordinary Ethernet MAC is tasked with formatting the data packets with the appropriate Ethernet frame format (e.g., adding delimiters and preamble words), checking and calculating the CRC on new data packets, and even implementing a simple congestion control mechanism in some cases by sending *pause frames* (IEEE 802.3-8, clause 31). In addition, the MAC also acts as a bridge between the AXI-Stream bus domain, which interfaces with the DMA core, and the underlying physical layer of the network link. This is achieved by transforming data transactions between the AXI-Stream and the Gigabit media-independent interface (GMII)[162], which allows connecting the MAC to different types of underlying physical layers (e.g., 1000Base-X, 1000Base-T, . . .). Moreover, some MAC implementations may contain a built-in MDIO controller for configuring their corresponding transceivers. We have used two different implementations for an Ethernet MAC throughout this thesis project.

- The Xilinx Ethernet Tri-Mode MAC [147], whose application was featured in our experiments with White Rabbit and in the early stages of prototyping of our TSN nodes.
- A lightweight MAC VHDL description from Libre Cores [148] that was enhanced with the updates for preemption that we studied in Chapters 8 and 9.

6.4.3 *The packet coding sublayer and physical medium attachment*

The packet coding sublayer and physical medium attachment modules (PCS/PMA) are responsible for implementing the physical layer protocol of the Ethernet link. In our experiments, we have used 1-Gb/s Ethernet interfaces over twisted-pair, copper-based cables, or optical fiber. Hence, the PCS/PMA module is tasked with bridging between the GMII interface and the serial interfaces for the appropriate link layer transceivers. Thus, these modules generate all the necessary signaling for maintaining the link active, even if no user data is present at the time, by sending the appropriate *comma characters*. In addition, they include an *auto-negotiation* unit that sets the operating mode of the link (e.g., 100Mbps, 1Gbps, full-duplex, . . .). Likewise, as was the case of the Ethernet MACs, we have used both external variants of the PCS/PMA and fully FPGA-based ones. The former were directly built internally within externally PCB-mounted chips, whereas the latter were the Xilinx PCS/PMA IP core [146] for the Zynq-7000 devices.

6.4.4 *The physical layer transceivers*

Lastly, the physical layer transceivers transform the serial data stream from the PCS/PMA modules into electrical signals adapted for transmission over the appropriate physical link layer. As we have used two main platforms for supporting our TSN nodes, the WR-ZEN and the MAIN Board, their corresponding architectures were paired with different types of transceivers. Hence, the WR-ZEN Boards for prototyping made use of the internal Xilinx GTP transceivers [149]. This allows for a simpler design of the PCB for the WR-ZEN Board and is also an inherited feature from its original application to White Rabbit synchronization nodes, which use a specific instantiation – a particularized VHDL *wrapper* – of the Xilinx GTPs to ensure that the transceivers behave deterministically and that their deserialization delay (*bitslide*) can be measured accurately. In contrast, the design of the MAIN Board uses an external PHY chip that supports its intended application to the avionics of an aerospace vehicle. Furthermore, the use of external PHYs can serve to lessen the resource requirements on the FPGA, as they only need a GMII-to-RGMII adaptor [163] to interface with the Ethernet MAC instead of the full FPGA-based implementation of a PCS/PMA module required by the internal GTPs. As a result, each type of PHY allows the use of different physical layers: The Xilinx GTPs provide the necessary serial interface with an SFP module, which in turn allows the use of either a 1000Base-X optical fiber Ethernet system or, by means of an *SFP-to-copper* adaptor, a twisted-pair 1000Base-T Ethernet link. Conversely, the external PHY of the MAIN Board interfaces directly with a twisted-pair 1000Base-T Ethernet link.

6.4.5 *Network drivers for embedded Linux environments*

The foregoing architectural elements (e.g., MAC, DMA, PHY) implement the core functionality for data transmission in our TSN nodes over an underlying Ethernet service; i.e., they provide the basic communication service. As mentioned previously, the TSN cores we have designed will then be integrated with the basic Ethernet cores to complement their functionality and, thus, enable a deterministic communication service. An important part of this process revolves around the configuration of the communication service itself: e.g., data frames have to be forwarded from the processor, the DMA core has to be able to handle the corresponding data transactions, the Ethernet MAC should be initialized, and the correct link speed and transmission mode should be selected during the auto-negotiation. That is the main job of the network drivers, which provide both an interface for the configuration and the operation of the Ethernet service from the OS kernel.

We have worked with two different versions for our network drivers: the Ethernet drivers for the WR-ZEN Board, and those for the RTEMS OS that we presented in the experiments of Chapters 8 and 9. Moreover, the Linux drivers we selected had two different variants depending on their underlying WR hardware: the upgrade over the original WR NIC (Chapter 5) or the new WR-ZEN Board design from Seven Solutions (Chapter 10). The drivers for the RTEMS OS were ported to this platform as a custom adaptation of the drivers for the WR-ZEN Board during a separate collaboration project with Seven Solutions.

We will focus on the description of the structure and the main features of the drivers for our Linux-based nodes, since the implementation of the RTEMS OS is beyond the main scope of this thesis project. Hence, the Linux kernel defines all the necessary data structures and functions (the driver *hooks*) for interfacing with the underlying hardware of different types of devices (e.g., networking interfaces, character devices, PCI bus peripherals, . . .). These are defined in the corresponding kernel headers and their use is well documented in the multiple resources for Linux kernel developers (e.g., training courses, reference books [85], . . .). As a result, this section will attempt to provide an overview of the basic elements that have to be considered for implementing the basic functionality of a network driver. Therefore, as a rule of thumb, a network driver requires the use of the following elements.

6.4.5.1 *A device tree entry*

Since we are working with an embedded Linux image for an ARM processor, the usual approach that the kernel has for dealing with the large variety of hardware platforms and their associated devices on ARM-based systems is to adhere to the device tree (DT) specification from the Open Firmware project. Thus, using a DT allows the kernel to discover the topology of the system at runtime and thus dynamically load the necessary kernel modules for handling the devices or peripherals associated with the architecture of a specific ARM-based platform. This approach also allows for a streamlined compilation of lighter kernel images, which would only get to include the necessary drivers for handling the architecture of a given platform as opposed to having "bloated" kernel images with a generality of unused drivers.

Device tree entries usually include a collection of fields such as the *"compatible"* property, the base memory address of the device, or other implementation-specific options such as the interrupt (IRQ) line numbers of the device. The "compatible" property identifies the kernel module version that the kernel should load for handling a specific type of device. Likewise, the rest of the properties in the device tree entry can supply the loaded kernel module with the corresponding parameters it should use for adjusting the operation of the device (e.g., the base memory address, the operation mode, the interrupt numbers, . . .) at runtime.

6.4.5.2 *The main driver code*

Generally, Linux kernel drivers are expected to implement a series of functions. In particular, our Linux network drivers include the following basic elements.

- A *"probe"* function. This function is one of the central elements of the driver, as it is the designated entry point into the kernel module. It usually specifies the sequence of operations needed for configuring the network interface and calls the appropriate tasks for accomplishing this goal. As a result, it contains a handler that matches to the "compatible" property of the DT entry, and makes a series of calls to functions for setting up parameters such as the auto-negotiation, the MAC address, . . .

- There is also a function for supplying the parameters for the auto-negotiation module of the PCS/PMA over the MDIO interface with the Ethernet MAC. This allows the kernel to specify the link speed (1 Gb/s, 100 Mb/s, 10 Mb/s) or the mode of operation (full-duplex, simplex).
- The usual driver *hooks* are also included and provide access to the main functionalities of the device. Thus, transmissions are initiated when an upper-layer module issues a call to the *"start_xmit()"* function, whereas receptions are associated with receiving a reception interrupt and the execution of its corresponding callback function ("cb_rx_..."). Furthermore, since our driver implementations are based on the Xilinx DMA, the drivers we have supplied make use of the necessary data structures and interfaces for using the scatter/gather (SG) mode of the DMA engine for handling transactions more efficiently.
- The implementation also includes two buffer descriptor linked lists, with each list devoted either to transmission or reception transactions. Moreover, each descriptor contains all the data structures associated with either a received or a transmitted packet, such as the packet buffer itself (sk_buff), the protocol, time-stamp data, ... Hence, the elements in each linked list represent the consecutive packet transactions handled by the DMA engine for transmissions or receptions; and the amount of units allocated to each class can have direct implications on the overall system performance. We verified this claim experimentally in our tests with the ZEN-CTA node (Appendix B).
- Most importantly given the nature of our application, our Ethernet drivers have support for **hardware time-stamping**. This is achieved by calling a specific function that accesses the time-stamping units of the timing subsystem upon reception or transmission of a gPTP protocol data frame. The time stamp so retrieved is then included in the *"sk_buff"* structure and passed to the upper-layer protocol applications.

6.4.5.3 *The driver variants*

We have worked with two different Linux driver variants throughout the thesis project. Thus, on the one hand, for the experiments of Chapter 5 we used a modified version of the WR-NIC and hence developed a customized version of the corresponding White Rabbit driver from the Open Hardware repository [131] with DMA enhancements. On the other hand, for the experiments where we used the WR-ZEN boards from Seven Solutions, the corresponding driver that supported their architecture was an adaptation from the original Xilinx Ethernet drivers for Linux ("xilinx_axienet_main.c"). These drivers were also ported to the RTEMS OS to allow the instantiation of an Ethernet communication service in our experiments with the Main Board (Chapters 8 and 9).

6.5 INTEGRATION WITH THE TIMING SUBSYSTEM

As indicated in the previous sections, the timing subsystem provides the common time base needed to allow the synchronous operation of distributed nodes. In the context of

TSN traffic-shaping, it supplies the time signal that drives the cyclic scheduling policies of the TSN TAS shapers and is eventually responsible for bringing about deterministic communications. This is achieved with a timing service that synchronizes the local PHC to that of a reference "Master" node, and through the use of the appropriate kernel-level drivers.

The timing services we have used throughout this project were a gPTP service for RTEMS, and the "ppsi" synchronization daemon for Linux for WR timing. Their development and implementation were beyond the scope of our work. Rather, we leveraged existing implementations of the aforementioned services that could work with the architectural blocks (e.g., TSUs, PHC, . . .) that we provided. In addition, since these services rely on the extensive use of hardware time-stamping to provide accurate synchronization, we had to enable time-stamping support where needed in the source code of our Ethernet network drivers.

7 DETAILED IMPLEMENTATION OF THE TSN SUBSYSTEM

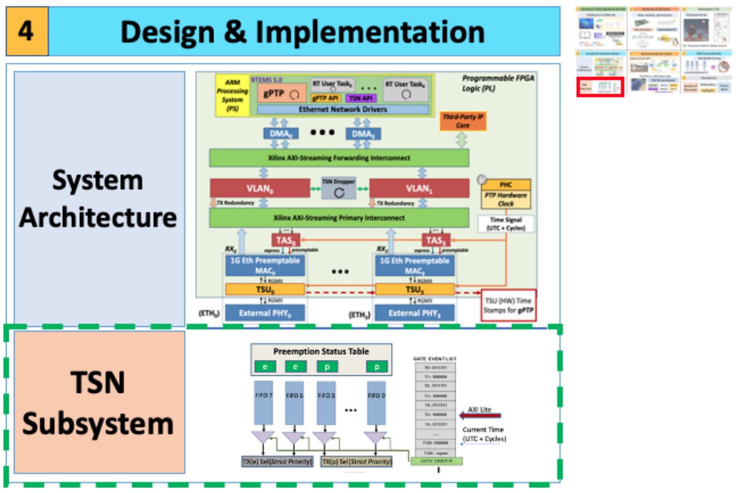


Figure 7.1
Overview of the contents of Chapter 7 detailing the implementation of the TSN subsystem.

This chapter is devoted to providing a thorough description of the implementation of the main modules of the TSN subsystem of our network nodes. Consequently, it expands upon the initial, high-level description that we provided in Chapter 6, which aimed to provide readers with generic system overview, by giving an in-depth description of the functionality, architectural implementation, operation, configuration, and system role of the main FPGA cores that we supplied for the design of the TSN subsystem. Hence, since our architecture can be customized to support the use of traffic identification (802.1Qcc) and VLAN-tagging (802.1Q) for routing and resource reservation, time-aware traffic shaping (802.1Qbv) with frame preemption (802.1Qbu & 802.3br), and seamless redundancy (802.1CB); we give a detailed explanation of the IP cores that we supplied for supporting these aforementioned features: a VLAN module, a TAS core for traffic-shaping that was also enhanced with frame preemption, a preemptable Ethernet MAC, and a Dropper module for discarding redundant duplicates. Also, we explain the role of the Xilinx crossbar switches for implementing the bridging functionality of our nodes. Lastly, we conclude by examining the resource usage of the nodes and with the considerations that led us to define a resource-conserving TSN architecture for FPGA.

Chapter contents

One of the essential steps in the implementation of a TSN node is the design and the construction of the TSN system itself. Since this is one of the main objectives of this thesis project, we have devoted an entire chapter to describing the implementation, design, and operation of the basic building blocks of our TSN nodes. Hence, as indicated in 6.1, our TSN devices can include time-aware traffic shapers with preemption, enhancements for redundancy, and a traffic identification mechanism. These are supported with the appropriate FPGA cores such as a VLAN module, a duplicate traffic dropper, or the TAS shaper core itself; and their operation and design are described in the corresponding sections throughout the chapter.

7.1 THE VLAN MODULE

The TSN VLAN Module has been designed to provide basic VLAN tagging and VLAN tag stripping (*untagging*) support for the TSN subsystem. Consequently, we have built support for the following basic operations described below.

- User-generated Ethernet frame encapsulation into user-configured VLAN streams with custom VLAN identifier (ID), VLAN priority (PRIO), destination MAC Address values.
- Multiplexing of different VLAN-encapsulated Ethernet frames to different TX interfaces, according to VLAN PRIO value specified in VLAN tag.
- VLAN tag stripping (*untagging*) on the RX Path for user-defined VLAN ID, VLAN PRIO, destination MAC Address combinations.
- RX data path multiplexing to PL-based AXI-Streaming peripherals, or DMA-based network interfaces based on programmable "*tdest*" flag associated with user-defined VLAN tags.
- Support for handling the emission and reception of redundant TSN streams over disjoint physical paths, as an enhanced protection mechanism for highly critical traffic.
- Shared management of multiple Ethernet Ports over a single TSN VLAN module in order to optimize FPGA logic resource usage.

7.1.1 *The VLAN block design diagram*

The TSN VLAN module supplies the VLAN-capable processing circuitry for both the TX and RX data paths of our nodes in order to handle the VLAN tagging and stripping operations required for the TSN subsystem. Hence, we provide an overview of the core by examining the implementation of its TX and RX data paths and how they can be used for supplying the traffic class identification, resource reservation, and routing features that are expected in a TSN system. In our design, we have accomplished this by combining three main state machines (FSMs) for each data path (a parser, a tagger/untagger module, and a central FSM configuration for each data path) with a configuration memory element (*CFG_MEM*).

7.1.1.1 *The TX data path*

On the TX data path, whose architecture can be examined in Fig. 7.2, the VLAN module parses the headers of the incoming user-generated Ethernet data frames. The header values are temporarily stored and then the TX FSM launches a comparison against the user-supplied configuration entries held in the VLAN Configuration Table (CFG_MEM). If a matching configuration is found, then the corresponding VLAN tag is applied and the resulting VLAN-encapsulated Ethernet frame is multiplexed based on its VLAN PRIO value to the corresponding output TX port for interfacing with the TAS module. In the event that no matching configuration can be retrieved from the Configuration Table, then the Ethernet frame is allowed through the VLAN Module "as-is" and multiplexed to the lowest priority queue of the TAS module (TX_1) (see Section 7.3).

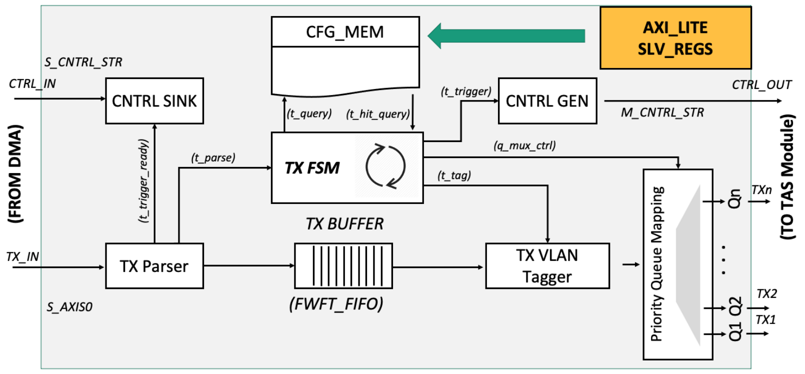


Figure 7.2
The TX data path of the VLAN module.

7.1.1.1.1 Considerations on the management of the gPTP traffic in the VLAN module

We have defined a special case for the handling of gPTP traffic on the TX path of the TSN VLAN module. Thus, we can detect the presence of incoming gPTP frames when their parsed ***Ether Type*** field matches the designated value of **0x88f7**, which indicates the presence of a gPTP protocol message. As a result, we have used this property to designate a preferential service for the gPTP messages, which are **forwarded through**

the VLAN module without applying any VLAN tag and multiplexed to the highest priority queue of the TAS module (TX_n).

7.1.1.2 *The RX data path*

On the RX data path, the VLAN module scans (*parses*) the headers of the received Ethernet frames to check for the presence of any VLAN tags. This can be seen in the architecture diagram of Fig. 7.3. If the received frame is found to contain a VLAN tag, then the RX FSM starts the comparison of its values against the user-supplied configuration entries held in the VLAN Configuration Table (CFG_MEM). If a matching configuration is found, the received frame is stripped of its VLAN tag and multiplexed towards the processing system (DMA-based network interface) or the FPGA programmable logic for interfacing with additional AXI-Streaming peripherals. Should there be no matching configuration entries for the received frame, then the frame is allowed through the VLAN module "as-is" and forwarded to the ARM processing system (DMA-based network interface).

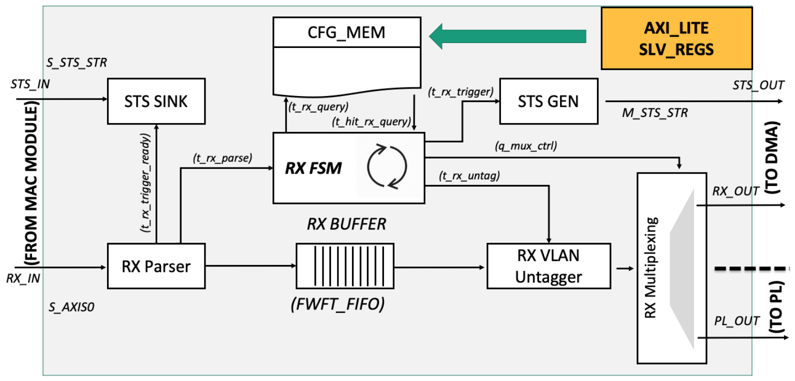


Figure 7.3
The RX data path of the VLAN module.

7.1.1.3 *Sharing of multiple Ethernet ports over a single TSN VLAN core*

We have fitted the TSN VLAN module with the optional capability of multiplexing the transmission or reception data paths of several Ethernet ports; i.e., a single VLAN core could potentially be shared amongst different ports for processing their traffic flows; rather than instantiating individual cores for handling each port. This feature is intended to minimize FPGA resource usage when implementing on relatively small devices or when tight FPGA resource constraints have to be considered in larger designs. Hence, since throughout our research we found that we had to build our designs with either small FPGA devices (the Z-7015 SoC) or under strict maximum resource usage limitations, as was the case of our implementations for aerospace, we concluded that this multi-port feature allowed us to overcome these hurdles. As a result, we found that not only can we save resources by having the VLAN module share its packet processing engine amongst multiple ports, but that we can also target FPGA devices and constraint sets successfully that would otherwise be unfeasible with a separate VLAN instance per port. This is one of the defining contributions of our architecture and, contrary to other implementations like that from Xilinx [164], it is one of the characteristics responsible for its moderate footprint.

We have indeed made extensive use of this new approach for supporting the system designs of the experiments in Chapters 8, 9, and 10. Furthermore, we provide an overview of the operation of this data path sharing feature amongst Ethernet ports in the following below.

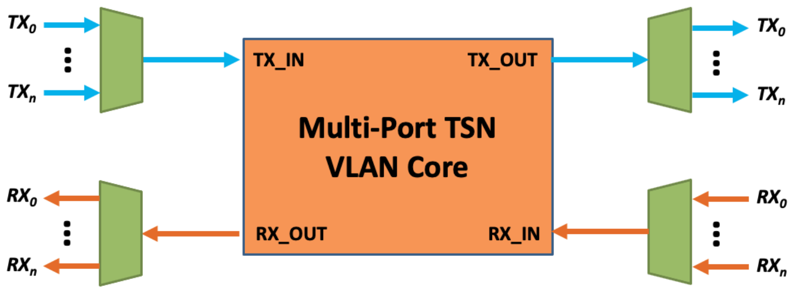


Figure 7.4
Simplified diagram of the multi-port sharing feature of the VLAN core to reduce FPGA resource usage.

7.1.1.3.1 Sharing on the TX data path.

On the transmission (TX) data path, the multi-port TSN VLAN module can identify the originating element for a transaction; be it either a DMA controller emitting Ethernet frames from the ARM processing system, or TSN frames forwarded from the reception (RX) path of a different Ethernet Port. This is accomplished by furnishing a specific identifier indicating the element that originated a TX transaction through the auxiliary ***tuser*** signal from the AXI-Stream system bus, which can specify user-defined signaling. As a result, the multi-port TSN VLAN module will in turn use the identifier supplied through the ***tuser*** signal to determine the appropriate Ethernet port that a specific Ethernet frame will be forwarded to by feeding it into its corresponding transmission data path.

7.1.1.3.2 Sharing on the RX data path.

Similarly, on the reception (RX) data path, the multi-port TSN VLAN module can also identify the Ethernet port over which a specific Ethernet frame was received using the same ***tuser*** identifier signaling mechanism outlined previously. Consequently, the multi-port TSN VLAN module will in turn use this identifier for determining the element that the Ethernet frame will be forwarded to in order to handle a specific reception transaction, which will usually consist of the DMA engine core associated with a specific Ethernet port.

7.1.2 *The VLAN configuration table*

The VLAN Configuration Table (CGF_MEM) is intended to hold user-supplied configuration entries, which the user specifies by uploading configuration parameters through the AXI Slave Configuration registers (see 7.1.3). These entries will be used either for applying a specific VLAN tag to transmitted Ethernet frames, or for stripping received Ethernet frames of their VLAN tag if a matching configuration is located in the table.

The VLAN Configuration Table can be represented logically as a number of user-configurable, 4-byte table entries (up to 128 4-byte entries) containing a set of VLAN Configuration Instances to be applied to transmitted and received Ethernet frames on the TX and RX data paths, respectively. Consequently, each Configuration Instance spans a block of 8 Configuration Table Entries; whereby the leading three entries contain a VLAN tag, and the remaining 5 entries hold the Ethernet header values associated with the aforementioned VLAN tag. As a result, the current implementation of the VLAN Configuration Table allows the user to configure up to 16 different Configuration Instances; i.e., 16 different *rules* for matching traffic to TSN streams. For a discussion on the actual FPGA implementation of the block, the reader may refer to Section 7.1.2.3.4. The logical structure for a Configuration Instance, replicated throughout the Configuration Table for all of the user-defined VLAN tagging/stripping configuration instances supplied, can be examined in Fig. 7.5. This structure allows us to differentiate between different classes of traffic by parsing key values of the Ethernet frame headers, such as the destination MAC address, IP address, source/destination port, protocol, . . . We can afford greater flexibility with this approach for telling apart the TSN flows in our system with finer granularity. It has also been of enormous assistance for achieving a successful configuration of our experiments in the Smart Grid and aerospace. This flexibility in configuration led to the design from Section 7.1.2.3.4 that had to juggle between conscious resource usage and traffic-processing speed.

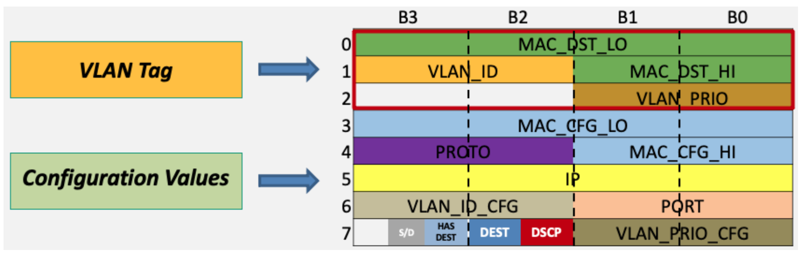


Figure 7.5
The logical layout of the configuration instances in the VLAN configuration table.

As can be observed in the figure, the Configuration Instance is divided into two different sections: the *VLAN tag section*, and the *Configuration Values section*. These are described next.

7.1.2.1 *The VLAN tag section*

The following fields are stored in the VLAN Tag section for each Configuration Instance.

- **MAC_DST_LO** (*32b*). Lower 32 bits of the destination MAC address (usually a multicast MAC address) to be applied alongside the VLAN tag.
- **MAC_DST_HI** (*16b*). Higher 16 bits of the destination MAC address (usually a multicast MAC address) to be applied alongside the VLAN tag.
- **VLAN_ID** (*16b*). VLAN ID field to be applied for the VLAN tag.
- **VLAN_PRIO** (*16b*). VLAN priority field to be applied for the VLAN tag.

- **FILTER** (*1b*). Enables the use of the FRER Filtering Function on the RX data path for redundant TSN streams that are consumed locally or forwarded to another node. This feature can only be used if the Multi-Port TSN VLAN module has been paired with a TSN FRER Dropper module (see Section 7.2), which will handle the detection and discarding of duplicate frames received over different Ethernet Ports when the Seamless Redundancy protection feature for TSN streams is used.

7.1.2.2 *The Configuration Values section*

The following fields are stored in the Configuration Values section. This section holds the Ethernet frame values that trigger the generation of a specific VLAN tag for a given Configuration Instance.

- **MAC_CFG_LO** (*32b*). Lower 32 bits of Ethernet frame Destination MAC Address.
- **MAC_CFG_HI** (*16b*). Higher 16 bits of the Ethernet frame destination MAC address.
- **PROTO** (*16b*). Transport protocol field indicated in the Ethernet frame (TCP or UDP).
- **IP** (*32b*). Destination IP address supplied in Ethernet frame.
- **PORT** (*16b*). Used in combination with D/S field. Denotes destination/source port contained within the Ethernet frame.
- **VLAN_ID_CFG** (*16b*). VLAN ID field of the Ethernet frame.
- **VLAN_PRIO_CFG** (*16b*). VLAN Priority field of the Ethernet frame.
- **DSCP** (*4b*). Differentiated services code point field of the Ethernet frame.
- **DEST** (*4b*). Destination interface for the received Ethernet frame (PL AXI-Streaming peripheral, or DMA-based network interface).
- **HAS_DEST** (*1b*). Indicates if a destination interface, other than the default DMA-based network interface, was supplied for forwarding the received Ethernet frames to a PL AXIS-based peripheral.
- **D/S** (*1b*). Indicates if the *PORT* entry refers to source (*"0"*) or destination (*"1"*) Port.
- **REDUNDANT** (*1b*). Indicates if the FRER Replication and Splitting functions are used for the current TSN stream on the TX data path. This configuration flag effectively enables the use of the enhanced protection mechanism of Seamless Redundancy for user-designated TSN streams on the transmission path (see Section 7.2).
- **RED_HANDLE** (*5b*). Internal unique integer value supplied for identifying a Redundant TSN stream and used for accessing its corresponding Sequence Number Generator (see Section 7.2).
- **RED_DEST** (*2b*). Destination interface supplied for forwarding duplicated TSN frames when the FRER Splitting function is used (see Section 7.2).

7.1.2.3 *Identifying traffic classes with the VLAN Configuration Table*

The Configuration Table holds the Configuration Instances that indicate whether a VLAN tag should be applied or removed from the user-generated Ethernet frames handled by the TSN node. Hence, in this section we provide an overview of how the traffic class identification mechanism operates on the basis of the VLAN Configuration Table.

7.1.2.3.1 The operation of the TX data path

On the TX data path, the header values of the transmitted Ethernet frames are parsed and compared against the Configuration Value sections of each one of the Configuration Instances stored in the Configuration Table. If any matching entries are found for the parsed header values in the Configuration Instances; then the corresponding VLAN tag values associated with a specific Configuration Instance are retrieved and applied for generating the VLAN tag on the transmitted frame. In the event that the same set of header values have partial matches for some entries across different Configuration Instances, then the first Configuration Instance that produces a match in the Configuration Table is selected for producing the VLAN tag. As a result, caution should be observed by the user configuring the TSN VLAN core since the priority of the different Configuration Instances for producing a VLAN tag on a transmitted frame would be determined by the loading order of the Configuration Instances onto the Configuration Table. The different types of header match events used for tag generation are outlined in Table 7.1.

Table 7.1
Priorities of header match events.

Priorities of header match events

Priority (Ascending)	**Header Match Event**
5	Destination MAC Address
4	DSCP (*Differentiated Services Code Point*).
3	Protocol
2	Destination IP
1	Source Port
0	Destination Port

As observed in the table, header match events for the entries held in each Configuration Section are ranked in ascending priority order. It can be observed that a given Configuration Instance will be selected for producing a VLAN tag in the event that there is a match on any of the header match events indicated in the table. It should be noted that the destination MAC address will be given the higher priority for determining whether the current Configuration Instance produces a match, whereas the destination port will be assigned the lower priority for determining a match event.

Should there be no matching entries for a certain set of header values in the Configuration Instances, the comparison circuitry of the Configuration Table generates a *"no-match"* condition signal to indicate that the Ethernet frame should be allowed through the VLAN Module "as-is" and multiplexed to the lowest priority TX port (TX_0).

7.1.2.3.2 Operation of the RX data path

On the RX data path, the VLAN tag values of the incoming Ethernet frames are parsed and compared against the VLAN tag Sections of each one of the Configuration Instances stored in the Configuration Table. If any matching entries are found for the parsed VLAN tag values in the Configuration Instances, then the corresponding Configuration Section is examined to determine if the *DEST* (destination) and *HAS_DEST* ("has destination") fields have been supplied. If the *HAS_DEST* field is present, then the VLAN tag is removed from the incoming Ethernet frame, its destination MAC address is replaced with that contained in the corresponding Configuration Section, and the frame is multiplexed to the destination indicated in the *DEST* field (PL AXI-Streaming); otherwise the frame is forwarded to the processing system (DMA-based network interface) once its VLAN tag has been removed.

The possible values of the *DEST*, the *HAS_DEST* fields, and their meanings are summarized in Table 7.2.

Table 7.2
Possible values of the "DEST" field.

Combinations with the destination field

HAS_DEST	DEST	Meaning
0	0-15	No Destination Information supplied (default). Forward to PS.
1	0-15	PL Destination Port Number. Forward to corresponding output port of TSN Redirector Module as indicated in DEST field.

In the event that no matching VLAN tags can be located in the Configuration Instances for a set of parsed VLAN tag values for a given incoming frame, then the frame is allowed through the VLAN module "as-is" and multiplexed to the processing system (DMA-based network interface).

7.1.2.3.3 The arrangement of the Configuration Instances

As indicated previously, a specific Configuration Instance will produce a match whenever any of the header match events indicated in Table 7.1 returns an affirmative value when comparing against an incoming Ethernet data frame. Since the TSN VLAN module assigns the higher priority value for determining a header match event to the destination MAC address, and the lower priority value to the destination port, it is recommended that the different Configuration Instances should be loaded onto the Configuration Table in reverse tagging priority order in order to ensure proper VLAN tagging operation. As a result, the configuration software for the VLAN module should guarantee that the different Configuration Instances are sorted through the Configuration table according to their preferred type of header match event in accordance with the suggested ordering depicted in Table 7.3.

Table 7.3
Recommended Loading Order for Different Types of Configuration Instances.

Suggested loading order for different match events.

Loading Order	**Types of Configuration Instance by Header Match Event**
1	Instance matches by **Destination Port**.
2	Instance matches by **Source Port**.
3	Instance matches by **Destination IP**.
4	Instance matches by **Protocol**.
5	Instance matches by **DSCP (*Differentiated Services Code Point*)**.
6	Instance matches by **Destination MAC Address**.

7.1.2.3.4 Implementation of the TSN VLAN Configuration Table

We have implemented the Configuration Table containing the different Configuration Instances with a series of distributed memory banks and DSP-based comparators. In practice, the VLAN Configuration Table is a pattern-matching engine: it has to retrieve the pointer that indicates the configuration that should be applied for encapsulating or removing the VLAN tag of a TSN flow. We considered several alternatives for implementing this engine. The first one consisted of the use of a content-associative memory (CAM). There are multiple CAM implementations, open-sourced or otherwise, that could fulfill this role, such as the Xilinx CAM implementation [165]. We also considered implementing our version of a CAM memory, but eventually opted for a different approach upon examining the resource usage consumption of the Xilinx CAM design. In this case, we found that a relatively large CAM for detecting the complex patterns that we needed would excessively tax our FPGA resources, thereby preventing us from complying with our maximum usage requirements or from fitting into the smaller FPGA devices altogether. Consequently, we chose a different approach that combined distributed memory elements arranged in a matrix-like fashion to store the patterns that we wanted our matching engine to detect. The pattern lookup would now have to be a sequential search for each memory block, as opposed to the nearly instant, same clock-cycle detection of the CAM; and the comparison between stored patterns and incoming Ethernet frames would have to be done with hardened DSP comparators to reduce LUT logic usage. This latter option, albeit slower and more complex than a CAM itself, allowed us to produce a reduced footprint design. Nonetheless, we propose the replacement of this pattern-matching system with a CAM-based module in our future work (Section 11.5).

Thus, in our present implementation, each pattern stored for matching (*recognition*) is called a *header match event*. Each distributed memory bank is intended to hold a different type of header match event for the Configuration Section and VLAN tag Section of the Configuration Instances stored in the VLAN Configuration Table. Hence, as shown in Fig. 7.6, this architectural design allows the data from the Configuration Instances to be searched in parallel upon reception of new Ethernet frames. Thus, this design provides

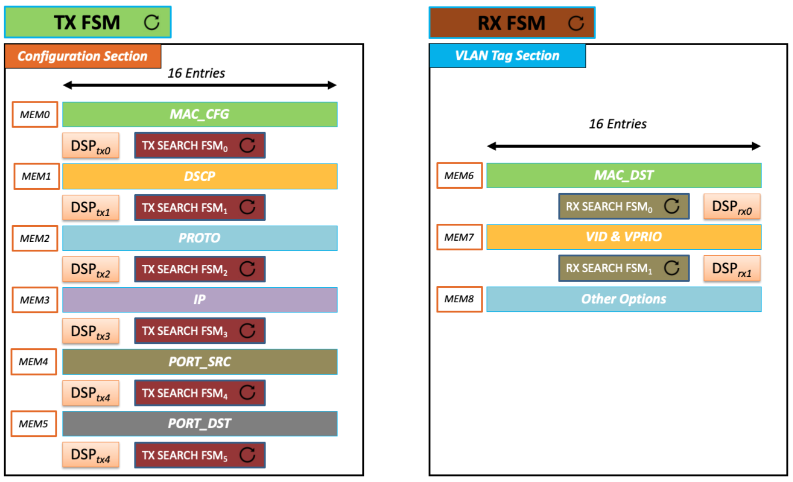


Figure 7.6
Implementation of the Configuration Table for the TSN VLAN core with distributed memory and DSP comparators.

a faster, more efficient logic for determining if a match event has been produced for applying/stripping a specific VLAN tag.

7.1.3 *Supplying the configuration: the Slave configuration registers*

The VLAN module operation is configured by furnishing the appropriate configuration values to its AXI Slave configuration registers from the ARM processing system. This allows the loading of different Configuration Instances to the Configuration Table for generating or stripping VLAN tags in user-produced Ethernet data frames. The registers implemented in the AXI Slave configuration module are shown in Table 7.4, along with a brief description of their functionality.

7.1.3.1 *Description of the configuration registers*

The following sections contain detailed descriptions of the Configuration Registers implemented for the TSN VLAN module, the fields present therein, as well as their functionality.

Table 7.4
The layout of the TSN VLAN slave configuration registers.

Slave register map for the TSN VLAN core.

Register name	Functionality	C offset
`CR`	Control Register	`0x0`
`SR`	Status Register	`0x4`
`CFG1`	Configuration Words for VLAN Tag Section of Configuration Instance (I)	`0x8`
`CFG2`	Configuration Words for VLAN Tag Section of Configuration Instance (II)	`0xc`
`CFG3`	Configuration Words for VLAN Tag Section of Configuration Instance (III)	`0x10`
`RUL1`	Configuration Words for Configuration Section of Configuration Instance (I)	`0x14`
`RUL2`	Configuration Words for Configuration Section of Configuration Instance (II)	`0x18`
`RUL3`	Configuration Words for Configuration Section of Configuration Instance (III)	`0x1c`
`RUL4`	Configuration Words for Configuration Section of Configuration Instance (IV)	`0x20`
`RUL5`	Configuration Words for Configuration Section of Configuration Instance (V)	`0x24`

7.1.3.1.1 The Control register

This manages the operation of the VLAN module. It allows the initiation of the load of an arbitrary number of Configuration Instances to the Configuration Table and controls the use of VLAN tagging/stripping functionalities or passthrough operation. Its structure can be examined in Table 7.5, and the meaning of its corresponding fields can be viewed in Table 7.6.

Table 7.5
The VLAN control register fields.

0 1 2	3	4	5 6 7	8	9	10 31
`Translation Rule`	`Done`	`Next Configuration Entry`	`Configuration Mode`	`Enable`	`Reset`	`Unused`

Table 7.6
Details of the VLAN control register fields.

Description of the CR register fields.

Bits	Description
31:10	Unused
9	**Reset. [RW]** Reset Configuration Table of VLAN Module. Usually recommended before loading a new set of Configuration Instances.
8	**Enable. [RW]** Enable VLAN Tagging/Stripping operation of the core. It is required that this bit be set to '1' before loading a new set of Configuration Instances.
7:5	**Configuration Mode. [RW]** Sets Configuration Mode for loading Configuration Instances. Must use default value of '0x1' for loading a new set of Configuration Instances.
4	**Next Configuration Entry. [RW]** Indicates that the current configuration values correspond to a new Configuration Instance.
3	**Done. [RW]** Indicates end of configuration cycle and that all Configuration Instances have been uploaded to the Configuration Table.
2:0	**Translation Rule. [RW]** Type of translation method used for generating VLAN tag. Must supply default value of "`0x1`".

7.1.3.1.2 The status register

This shows the current, user-supplied values of the Control Register. Its structure can be examined in Table 7.7, and the meaning of its corresponding fields can be viewed in Table 7.8.

Table 7.7
The VLAN status register fields.

0 1 2	3	4	5 6 7	8	9	10 31
Translation Rule	Done	Next Configuration Entry	Configuration Mode	Enable	Reset	Unused

Table 7.8
Details of the VLAN status register fields.

Description of the SR register fields.

Bits	Description
31:10	Unused
9	**Reset. [RO]** Reset Configuration Table of VLAN Module. Usually recommended before loading a new set of Configuration Instances.
8	**Enable. [RO]** Enable VLAN Tagging/Stripping operation of the core. It is required that this bit be set to '1' before loading a new set of Configuration Instances.
7:5	**Configuration Mode. [RO]** Sets Configuration Mode for loading Configuration Instances. Must use default value of '0x1' for loading a new set of Configuration Instances.
4	**Next Configuration Entry. [RO]** Indicates that the current configuration values correspond to a new Configuration Instance.
3	**Done. [RO]** Indicates end of configuration cycle and that all Configuration Instances have been uploaded to the Configuration Table.
2:0	**Translation Rule. [RO]** Type of translation method used for generating VLAN tag. Must supply default value of "0x1".

7.1.3.1.3 The Configuration register (I)

This holds the lower 32 bits of the destination MAC address that will be applied to the Ethernet data frames for generating their corresponding VLAN tag. Its structure can be examined in Table 7.9, and the meaning of its corresponding fields can be viewed in Table 7.10.

Table 7.9
The VLAN configuration register (I) fields.

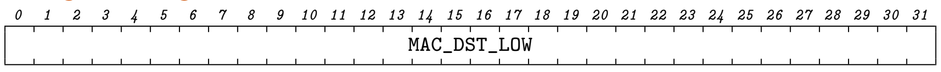


Table 7.10
Details of the VLAN configuration register (I) fields.

Description of the CFG1 register fields.

Bits	Description
31:0	**MAC_DST_LO. [RW]** Lower 32 bits of Multicast MAC Destination Address that will be generated along with the VLAN tag.

7.1.3.1.4 The Configuration register (II)

This holds VLAN ID and Multicast destination MAC address of the VLAN tag that will be applied to the user-generated data frames. Its structure can be examined in Table 7.11, and the meaning of its corresponding fields can be viewed in Table 7.12.

Table 7.11
The VLAN configuration register (II) fields.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MAC_DST_HI																VLAN_ID															

Table 7.12
Details of the VLAN configuration register (II) fields.

Description of the CFG2 register fields.

Bits	Description
31:16	**VLAN_ID. [RW]** VLAN Identifier supplied for generating the VLAN tag.
15:0	**MAC_DST_HI. [RW]** Higher 16 bits of Multicast MAC Destination Address that will be generated along with the VLAN tag.

7.1.3.1.5 The Configuration register (III)

This holds the VLAN priority value that will be applied to the user-generated Ethernet data frames. Its structure can be examined in Table 7.13, and the meaning of its corresponding fields can be viewed in Table 7.14.

Table 7.13
The VLAN configuration register (III) fields.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
VLAN_PRIO																RED_DEST		RED_HANDLE								Unused					

Table 7.14
Details of the VLAN configuration register (III) fields.

Description of the CFG3 register fields.

Bits	Description
31:26	Unused.
25:18	**RED_HANDLE. [RW].** Unique Internal value supplied for identifying a specific TSN redundant stream.
17:16	**RED_DEST. [RW]**. Indicates the user-supplied destination interface for forwarding duplicated TSN frames.
15:0	**VLAN_PRIO. [RW]** VLAN Priority value supplied for generating the VLAN tag.

7.1.3.1.6 The Configuration Rule register (I)

This holds the destination MAC address of the Configuration Section for a given Configuration Instance. Its structure can be examined in Table 7.15, and the meaning of its corresponding fields can be viewed in Table 7.16.

Table 7.15
The VLAN configuration rule (I) register fields.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
MAC_CFG_LO																															

Table 7.16
Details of the VLAN configuration rule (I) register fields.

Description of the RUL1 register fields.

Bits	Description
31:0	**MAC_CFG_LO. [RW]** Lower 32 bits of Destination MAC Address to be loaded for the Configuration Section of a specific Configuration Instance.

7.1.3.1.7 The Configuration Rule register (II)

This holds the destination MAC address and packet protocol fields of the Cofiguration Section for a given Configuration Instance. Its structure can be examined in Table 7.17, and the meaning of its corresponding fields can be viewed in Table 7.18.

Table 7.17
The VLAN configuration rule (II) register fields.

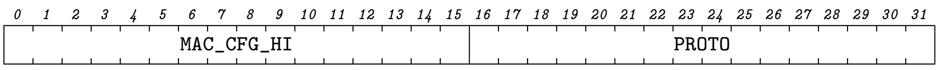


Table 7.18
Details of the VLAN configuration rule (II) register fields.

Description of the RUL2 register fields.

Bits	Description
31:16	**PROTO. [RW]** Packet Protocol field to be loaded for the Configuration Section of a specific Configuration Instance.
15:0	**MAC_CFG_HI. [RW]** Higher 16 bits of Destination MAC Address to be loaded for the Configuration Section of a specific Configuration Instance.

7.1.3.1.8 The Configuration Rule register (III)

This holds the destination IP address field of the Configuration Section for a given Configuration Instance. Its structure can be examined in Table 7.19, and the meaning of its corresponding fields can be viewed in Table 7.20.

Table 7.19
The VLAN configuration rule (III) register fields.

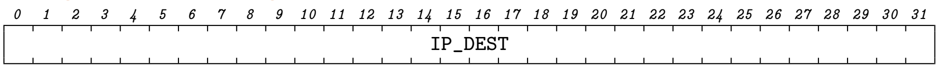


Table 7.20
Details of the VLAN configuration rule (III) register fields.

Description of the RUL3 register fields.

Bits	Description
31:0	**IP_DEST. [RW]** Destination IP Address to be loaded for the Configuration Section of a specific Configuration Instance.

7.1.3.1.9 The Configuration Rule register (IV)

This holds the VLAN ID and the port fields of the Configuration Section for a given Configuration Instance. Its structure can be examined in Table 7.21, and the meaning of its corresponding fields can be viewed in Table 7.22.

Table 7.21
The VLAN configuration rule (IV) register fields.

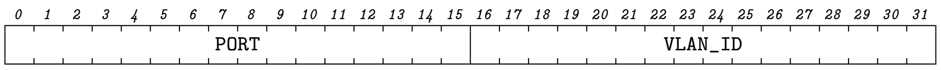


Table 7.22
Details of the VLAN configuration rule (IV) register fields.

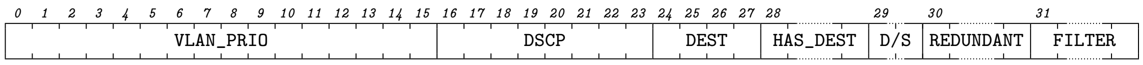
Description of the RUL4 register fields.

Bits	**Description**
31:16	**VLAN_ID. [RW]** VLAN_ID field to be loaded for the Configuration Section of a specific Configuration Instance.
15:0	**PORT. [RW]** Communication Port field to be loaded for the Configuration Section of a specific Configuration Instance. Used in combination with 'D/S' field in Configuration Rule (V) register to indicate whether the Port field refers to either Source Port ("0") or Destination Port ("1").

7.1.3.1.10 The Configuration Rule register (V)

This holds the VLAN_PRIO, DSCP, DEST, and D/S fields of the Configuration Section for a given Configuration Instance. Its structure can be examined in Table 7.23, and the meaning of its corresponding fields can be viewed in Table 7.24.

Table 7.23
The VLAN configuration rule (V) register fields.



7.1.4 *The configuration sequence in the Slave registers for the VLAN core*

In order to load a set of Configuration Instances onto the Configuration Table of the VLAN module and control the start of its tagging/untagging operation, we have devised the following sequence that programmers should observe when writing configuration parameters to the AXI-Slave registers of the core.

1. Supply the values of the first Configuration Instance to be loaded onto the Configuration Table and initialize the Control Register.

Table 7.24
Details of the VLAN configuration rule (V) register fields.

Description of the RUL5 register fields.

Bits	Description
31	**FILTER. [RW]**. Enables the application of the FRER Filtering function for a specific Configuration Instance in order to discard duplicate frames received as part of a redundant TSN stream.
30	**REDUNDANT. [RW]**. Used to indicate that TSN streams transmitted using a specific Configuration Instance have to be submitted to the enhanced protection feature of Seamless Redundancy for Reliability (FRER Stream Identification), thereby enabling the use of the FRER Splitting and FRER Sequence Number Generation functions for a specific TSN Stream.
29	**D/S. [RW]**. Used in combination with Port field of Configuration Rule (IV) register. A value of '1' indicates Source Port, and a value of '0' denotes Destination Port.
28	**HAS_DEST. [RW]**. Used in combination with DEST field of Configuration Rule (V) register. A value of '1' activates TSN Redirector to forward Ethernet frames to Port indicated in DEST field; and a values of '0' disables TSN Redirector operation.
27:24	**DEST. [RW]** RX Destination field to be loaded for the Configuration Section of a specific Configuration Instance.
23:16	**DSCP. [RW].** DSCP (Differentiated Services Code Point) field to be loaded for the Configuration Section of a specific Configuration Instance
15:0	**VLAN_PRIO. [RW]**. VLAN Priority field to be loaded for the Configuration Section of a specific Configuration Instance

a) Firstly, supply the configuration values for the Configuration and VLAN tag Sections to the applicable slave configuration registers.

b) Initialize the Control Register to enable operation of the VLAN module and load the first Configuration Instance onto the Configuration Table.

   i) Set *Enable* bit to "1".

   ii) Set *Translation Rule* and *Configuration Mode* to default value of *0x1*.

   iii) Set *Next Configuration Entry* to "1" in order to indicate first Configuration Instance entry.

   iv) Set *Done* to "0".

2. For the subsequent Configuration Instances, the following sequence should be applied.

   a) Supply configuration values for the Configuration and VLAN Tag Sections to the applicable slave configuration registers.

  b) Update Control Register to indicate that a new Configuration Instance is ready to be added to the Configuration Table.
    i) Set *Next Configuration Entry* to *modulo2* value of current Configuration Instance number (*#CfgInst%2*). Please, note that Configuration Instances are numbered from 1 onwards.

3. Once all Configuration Instances have been loaded, update the Control Register to indicate the end of the Programming Sequence.
   a) a. Set *Done* field of the Control Register to "1".

7.2 SEAMLESS REDUNDANCY FOR TSN: THE DROPPER CORE AND THE ENHANCEMENTS FOR THE VLAN MODULE

We have updated the TSN VLAN module to feature compliance with the specifications for Seamless Redundancy and Duplication for Reliability (802.1CB). As a result, the TSN system will implement a series of mechanisms for identifying highly critical TSN streams that will be shielded with an additional layer of protection allowing their redundant transmission over disjoint physical network paths.

Therefore, the mechanisms for Seamless Redundancy and Replication for Reliability (*FRER*) have to support four main features in order to enable this functionality for select TSN data streams: *a)* Identification of TSN streams to be submitted for Replication, *b)* Sequence Number Generation for Redundant Streams, *c)* Transmission over disjoint physical paths (*Splitting*), and *d)* Filtering of duplicates at the receiving end (*discarding duplicates*). All these aspects can be examined in the following points.

7.2.1 *Identification of TSN streams*

This function allows the unique identification of the user-configured TSN streams that are to be submitted to the Stream Replication process for enhanced reliability. Redundant TSN streams are internally identified using a user-designated "*stream_handle*", which will be associated with a unique MAC source address and VLAN ID pair. The resulting redundant stream will thus incorporate a Redundancy Tag (R-TAG) and a corresponding Sequence Number to the VLAN-tagged TSN stream to mark the use of Redundancy features.

7.2.2 *Sequence Number Generation*

Another major component for 802.1CB is the sequence generation function, which produces a unique Sequence Number for each TSN frame transmitted over a redundant stream. The Sequence Number will therefore be associated with a specific, user-supplied *stream_handle* that will identify each redundant TSN stream processed at the transmitting node. Consequently, this function will generate unique Sequence Numbers for each redundant TSN stream transmitted from the local node.

7.2.3 *Replication (Stream Splitting)*

The *splitting* function creates a duplicate copy of a redundant TSN stream and forwards it over the physical interface indicated by the user. This feature implements the Replication for Reliability functionality, since it allows the forwarding of redundant TSN streams over the disjoint physical paths configured by the used. Hence, the actual disjoint physical path for forwarding the redundant TSN streams is specified as a user-supplied configuration parameter associated with the *stream_handle* indicated during the configuration of the FRER features of the VLAN module for a given redundant stream.

7.2.4 *Elimination of duplicates (Stream Filtering)*

In the context of FRER, *filtering* refers to the function that discards the duplicates received at the TSN listener node. The filtering function will thus be tasked with recovering (i.e., *keeping track of*) the sequence number and eliminating the duplicate frames of a specific redundant stream marked for filtering at the receiving end by the user.

7.2.5 *Overview of the operation of the FRER features*

A functional implementation of a TSN networking system with seamless redundancy requires the combination of the foregoing functions. Thus, the TSN talker will usually implement the identification, splitting, and sequence number generation functions; whereas the listener will handle the filtering operations of any redundant duplicates. Hence, we provide an overview of how these mechanisms interact with each other and how we have combined them in our design for supplying a working implementation of 802.1CB FRER: with the use of a dedicated TSN Dropper core and by enhancing the VLAN core circuitry.

The use of Redundancy features as an added layer of protection for highly critical streams can be observed in the Fig. 7.7. Thus, we can observe that the Replication scheme proposed for TSN handles the creation of redundant TSN streams as the emission of duplicated TSN streams that are routed over disjoint physical paths across the TSN network. Duplicated TSN streams are in turn encapsulated using the VLAN-based tagging methodology defined for identifying streams. It should be noted that the VLAN ID and VLAN priority values generated for each duplicate stream are user-supplied and can change on a per-link basis, as required by the network scenario.

Consequently, in order to support frame replication, emitter nodes will need to implement additional frame replication and frame labeling circuitry on the TX data path of the VLAN module. The TSN streams marked for replication by a user-specified configuration will in turn be replicated and forwarded to a different TSN Ethernet port by means of the TSN Redirector (see Section 7.5) module. The duplicated TSN streams emitted over disjoint physical paths of the emitter node (different ports) will include additional tagging/labeling for further processing at the receiving nodes. Lastly, the receiving nodes will need to implement frame discarding logic to eliminate duplicates of frames received over disjoint paths.

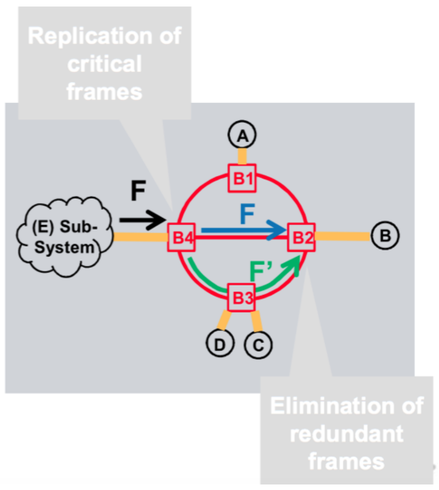

Replication of critical frames
F
(E) Sub-System
A
B1
F
B4
B2
B
F'
B3
D
C
Elimination of redundant frames

Figure 7.7
Use of FRER as added protection for TSN Streams. Reproduced from [166].

As a result, the mechanisms outlined previously that are required for supporting the Redundancy features have to be translated into a series of additional modules and upgrades for the stream reservation components of the TSN system (the VLAN Module, chiefly). As a rule of thumb, it can be stated that the TSN talker nodes used for emitting redundant TSN streams have to implement the FRER functions of Redundant Stream Identification, Sequence Number Generation, and Transmission over Disjoint Physical Paths (Stream Splitting). On the other hand, the TSN listener nodes consuming a user-specified redundant stream have to implement the FRER Filtering function for discarding the duplicate frames of a given redundant stream received over different physical paths. Therefore, this has implied the addition of several changes to the traffic-processing components of the system. We show the extent of these changes in Figs. 7.8 and 7.9, which depict the main upgrades applied to the VLAN core for supporting FRER alongside the inclusion of a new TSN Dropper core for implementing the filtering function at the receiving nodes.

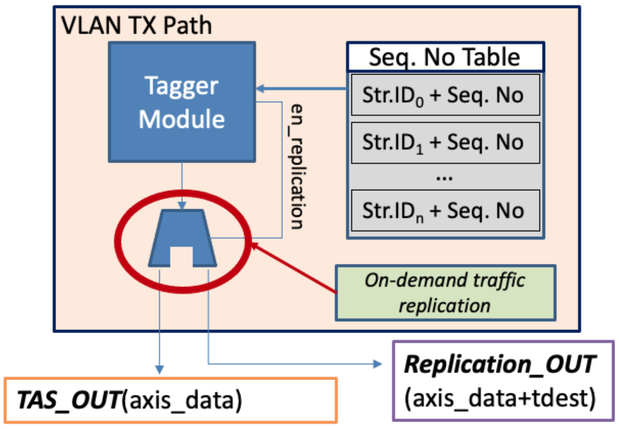


Figure 7.8
Changes to TX data path of VLAN Module to support use of FRER features.

As can be observed in the figures, we had to implement a number of upgrades around the core functionality of the VLAN Module in order to add support for FRER. Firstly, the TX data path (Fig. 7.8) needs to incorporate additional mechanisms for producing the

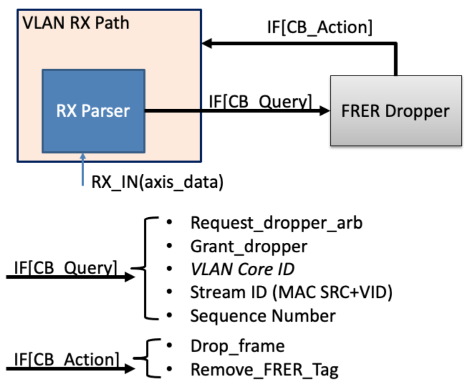
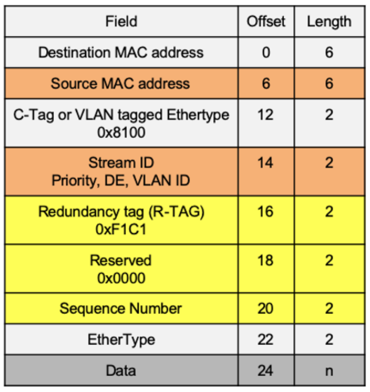


Figure 7.9
Changes to RX data path of TSN VLAN Module showcasing the use of a TSN Dropper Component to support discarding of duplicates for FRER.

R-TAG and the corresponding Sequence Number on a per-stream basis. These Sequence Numbers are in turn stored in an auxiliary Sequence Number Table that keeps track of the latest Sequence Number generated for each redundant TSN stream, identified by a unique *stream_handle* value. Thus, the TSN frame format is correspondingly modified as shown in Fig. 7.10 when FRER features are used.



Field	Offset	Length
Destination MAC address	0	6
Source MAC address	6	6
C-Tag or VLAN tagged Ethertype 0x8100	12	2
Stream ID Priority, DE, VLAN ID	14	2
Redundancy tag (R-TAG) 0xF1C1	16	2
Reserved 0x0000	18	2
Sequence Number	20	2
EtherType	22	2
Data	24	n

Figure 7.10
Modified TSN Frame Format when FRER Features are Enabled.

On the other hand, the RX data path (Fig. 7.9) needs to implement frame discarding logic for dropping duplicate frames received over different physical paths for a given redundant TSN stream. This is achieved by a separate module that interfaces with the VLAN Module for determining the redundant frames that are to be dropped: the TSN FRER Dropper module (Fig. 7.11).

7.2.6 *The TSN FRER Dropper module*

The TSN FRER Dropper module can be shared amongst different VLAN modules and keeps track of the redundant frames that are consumed at the local node for determining the frames from a given redundant stream that will either be dropped or have their R-TAG, Sequence Number and VLAN tag fields removed. The TSN Dropper module operates in a manner such that the first instance of a duplicate frame for a given redundant stream is allowed to proceed through the VLAN module; and any subsequent duplicate instances received for that frame are discarded. Thus, upon receiving a redundant TSN frame, the VLAN module submits a query to the Dropper module with the source MAC address, VLAN ID, and Sequence Number of the received redundant frame. Then, the TSN Dropper processes this information to determine if the Sequence Number for the redundant stream identified by a given pair of source MAC address and VLAN ID has been received previously. Once the Sequence Number for the redundant stream has been processed in accordance with the sliding window mechanism specified in IEEE 802.1CB, then the Dropper module generates a reply containing the action that the VLAN module should take for the received frame; i.e., drop the frame or allow it through the VLAN module. The main components of the TSN Dropper can be examined in the diagram included in Fig. 7.11.

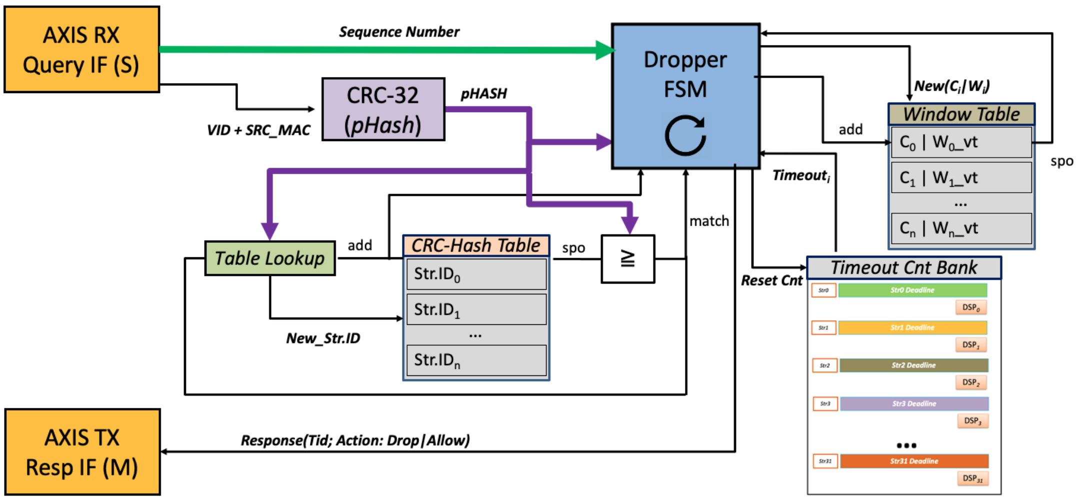


Figure 7.11
Block Diagram of the Implementation Supplied for the TSN FRER Dropper Module.

Next, we move on to describing the functionalities implemented in each one of the blocks included in the design of the TSN FRER Dropper in the following points.

7.2.6.1 *The AXI-Streaming RX query interface*

The AXI-Streaming RX query interface (*AXIS RX Query IF*) is interface allowing the reception of Dropper queries from the TSN Multi-Port VLAN module. Each query will contain a VLAN ID value, SRC MAC address, and the corresponding Sequence Number associated with the redundant TSN frame being processed. These values will be further processed by the rest of the TSN Dropper module engine to determine if a specific redundant frame is to be discarded as a duplicate or allowed to proceed through the VLAN Module.

Additionally, as the TSN FRER Dropper module can be shared amongst different TSN VLAN modules, the query received over the interface will also include a VLAN module identifier (*VLAN core ID*). This identifier will be used for determining the VLAN module that the TSN Dropper response to a specific query will be forwarded to.

7.2.6.2 *The CRC-32 pseudohash block*

In our design, we have implemented a simple mechanism for telling redundant streams apart by using a CRC-32 *pseudohash* block (*CRC-32 pHash*). Hence, we feed the VLAN ID and SRC MAC address values identifying a specific TSN Redundant stream into this device in order to calculate a CRC-based *pseudohash*. This *pseudohash* (*pHash*) value is in turn stored in a CRC-hash Table and used to retrieve a pointer to the Sliding Window associated with the Redundant stream being processed, which is stored in the *Window Table component*.

7.2.6.3 *The CRC-Hash Table and the Table Lookup Mechanism (pHash)*

The table containing the *pHash* values calculated by the CRC-32 component for each SRC MAC and VLAN ID tuple. Upon reception of a new query from the TSN VLAN module, the CRC-Hash Table will be scanned using the *Table Lookup Mechanism* until the position of the *pHash* value supplied by the CRC-32 component for given pair of VLAN ID and SRC MAC address is located. If the requested *pHash* value cannot be located on the Table, then it will be recorded in the next available position on the Table. This position will in turn be used as a pointer to the Window Table in order to retrieve the ***Sliding Window*** values associated with the processing of given TSN Redundant stream.

7.2.6.4 *The window table*

The table storing the ***Sliding Window*** values associated with a specific TSN Redundant Stream. Each table entry will thus be made up of a 32-bit word containing two different components: a 16-bit Window Origin, and a 16-bit History Vector. The Window Origin indicates the origin of the current position of the sliding window, while the History Vector spans the 16 subsequent positions after the Window Origin and is used for recording the reception of specific Sequence Numbers within the 16 positions covered by the sliding Window.

7.2.6.5 *The timeout counter bank*

This component can be instantiated optionally to provide an additional fail-safe mechanism in the event of a failure in the operation of the functions for seamless redundancy. Hence, we have built a DSP48-based counter bank implementing the generation of a timeout signal for each Sliding Window stored in the Window Table. Thus, this counter bank will enforce a deadline value by keeping track of the evolution of each Sliding Window from the Window Table. Consequently, this failsafe operates by incrementing the counter values of each window so long as the node keeps discarding redundant TSN frames before a maximum deadline of 10 ms can be reached, which would trigger the generation of a Window Reset signal indicating the expiration of the Sliding Window. This would prevent the TSN Dropper module from constantly discarding frames in the event that the Sequence Generation function from the transmitter (the TSN talker) becomes stuck in a particular value.

As observed in Fig. 7.11, the implementation we have supplied consists of an array of timeout counters ("*Timeout Cnt Bank*"), where each counter is tasked with enforcing the 10 ms deadline for each one of the Sliding Windows that the core could potentially be handling. For the sake of simplicity, this design can be thought of as a hardware implementation of a *watchdog timer* for each Sliding Window of the Dropper. Hence, the reception of a new Sequence Number corresponding to the first instance of a duplicated frame would cause the TSN Dropper to accept the reception of the new TSN frame and reset the deadline value associated with its Sliding Window; i.e., reset the *watchdog timer* for the window.

7.2.6.6 *The Dropper finite state machine*

This is the implementation of the TSN FRER Dropper Decision Engine, which we have supplied as a dedicated finite state machine (FSM). This component is therefore tasked with retrieving the Sliding Windows stored in the Window Table for each VLAN ID and MAC SRC address tuple received with the queries issued by each TSN VLAN module, and then generate a ***Decision*** based on the Sequence Number supplied with the query. The ***Decision*** value generated by the Dropper FSM will be either ***Drop*** or ***Allow***. A ***Drop*** decision is issued when a duplicate frame is detected and will signal the TSN VLAN module to reject a specific incoming frame, whereas an ***Allow*** decision is issued when the first instance of a duplicated frame is processed and will cause the TSN VLAN module to accept the new incoming frame.

7.2.6.7 *The AXI-Streaming TX response interface*

The AXI-Streaming TX response interface (*AXIS TX Resp IF*) is the master AXI-Stream Interface used for forwarding the Dropper ***Decision*** value generated by the Dropper FSM as a result of processing the VLAN ID, MAC SRC address, and Sequence Number of a redundant TSN frame. As the TSN Dropper module can be shared amongst different TSN VLAN Modules, the TSN VLAN module Identifier (*VLAN core ID*) originally supplied with the query will be used for forwarding the Dropper Response to the query – the Dropper ***Decision*** – to the appropriate TSN VLAN module.

7.3 THE TIME-AWARE TRAFFIC SHAPER

The time-aware traffic shaper is one the central components of our TSN implementation, as it allows the deterministic forwarding of data over the network. Moreover, the IEEE specification for TSN allows the use of different types of traffic shapers. Hence, to name a few, there is an asynchronous traffic shaper (802.1Qcr), a credit-based shaper (802.1Qav), and a time-aware shaper (802.1Qbv). The designer of the system should then choose the appropriate shaper for their implementation depending on the overall requirements of the network scenario that their deployment of a TSN system is intended to serve.

The time-aware traffic shaper (TAS) allows the processing of different data flows and traffic classes with a time-division multiplexing scheme, thus resulting in a more robust shaper alternative than the credit-based, asynchronous, or cyclic queueing schemes that are also contemplated for TSN. Hence, these latter shapers could still be used when less stringent constraints for end-to-end determinism are needed, as is the case of certain multimedia or automotive applications. In contrast, the TAS shaper can be selected when stricter end-to-end latency and determinism have to be enforced. That has been the case of the avionics and Smart Grid applications where we have conducted the experiments from Chapters 8 and 9.

The time-aware traffic shaper is described thoroughly in the IEEE 802.1Qbv [9] specification, and is meant to allow the cyclic forwarding of data according to a schedule – a gate control list (GCL) – supplied by the user of the network. This approach results in a "real-time-like" traffic forwarding system that sends data traffic only during certain time intervals – time *slots* – that the user has previously allocated to select traffic classes. Consequently, the implementation of this traffic shaping methodology requires that all the TSN nodes in the network be previously synchronized and that they implement an interface with the PHC time reference from the timing subsystem. These features allow the TAS core to be able to shape and forward data traffic synchronously and in coordination with the shapers of other nodes scattered throughout the network. Hence, the IEEE specification for TAS shaping can achieve these effects by building a time-driven queueing system with the architecture of Fig. 7.12.

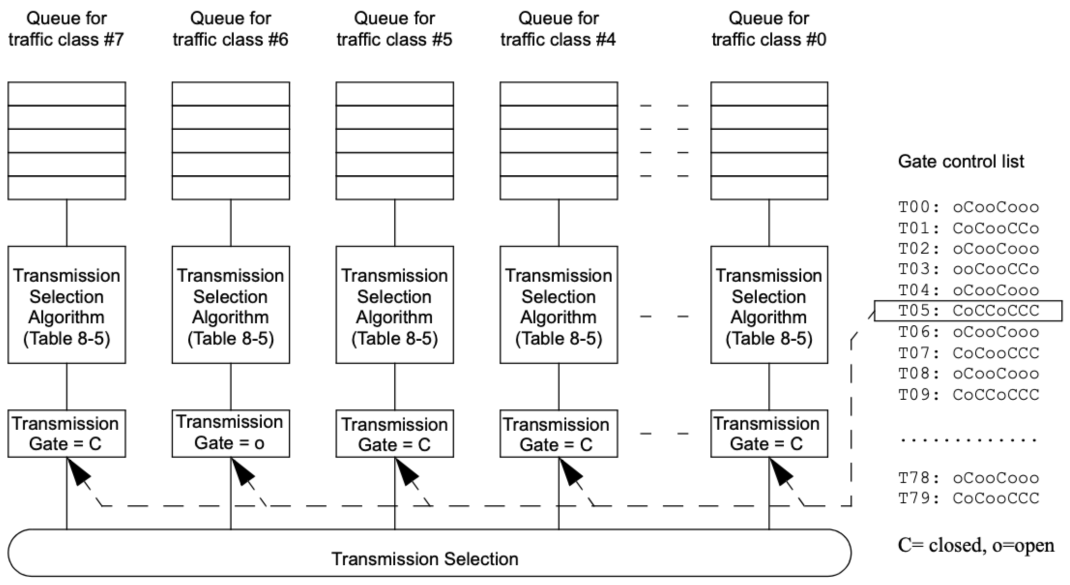


Figure 7.12
Block diagram of the architecture of the TAS shaper for implementing a time-driven traffic selection method with gates. Image from [9].

This architecture allows our TSN bridges and end systems to schedule transmissions relative to a known timescale from each one of the queues of the shaper, and we provide an overview of its operation in the following lines. Hence, as seen in the diagram, the forwarding of data from a given queue can be enabled by activating its corresponding *transmission gate*. These transmission gates, which are also termed *gate drivers*, can set the state of the queue to either *open* (**O**) or *closed* (**c**). Thus, this implies that the *open* queues can have their data frames selected for transmission, whereas the *closed* queues have to hold their data internally until their queue state can be updated to *open*.

The gate drivers are in turn steered by the execution of a so-called gate control list (GCL), which allows the user to specify a time-driven schedule with a cyclic execution period. Thus, the GCL is composed of a series of entries defining the different transmission slots for the traffic classes handled by the node, with each entry consisting of two separate components: an array of the applicable configuration values (O/c) for the gate drivers of the shaper during the execution of the current slot, and a *time offset* value indicating when the next entry of the schedule should be executed. In the event that several queues happened to be open during the execution of a given slot, then a strict priority selection algorithm would be used to select the queue which would get to forward its messages first, i.e., the queue would be set to the *open* state.

Our implementation of the TAS shaper is highly customizable. It was supplied by a collaborator from the Andalusian Institute of Astrophysics (IAA), with whom we collaborated closely in the block design specification and functional verification stages. This is the version that was used for the experiments of Chapters 8 and 9 with our collaborators. For the experiments of Chapter 10, we worked upon the initial design of our collaborator to allow the core to interface with a White Rabbit timing source. The flexibility of the design implies that it has multiple configuration options for its main parameters, as indicated next:

- The number of queues can range from a single element up to a maximum of eight; although our designs are generally customized to use four queues as a compromise between FPGA BRAM usage and performance.
- The amount of entries for the GCL. Likewise, the depth of the GCL list can be set to a reasonable value, such as the 12 entries we normally use, to reduce FPGA logic usage.
- We can also adjust the width of the configuration values array for each GCL entry as a function of number of queues used in a given TAS instantiation.
- The priorities for each queue can be assigned individually, with the design even allowing for the allocation of the same priority value to different queues. However, in the TAS implementation that we have supplied for this project, we have assigned different priorities to each queue in ascending order.

7.3.1 *Overview of the operation of the time-aware traffic shaper*

The deterministic traffic forwarding behavior of the TAS shaper can be attained as a result of coupling three different finite state machines: the "*list configurator*", the "*cycle timer*", and the "*list execute*" controller. This can be seen in Fig. 7.13. The functionalities, signals, and recommended implementation for each one of the preceding elements can be found in the applicable sections of the 802.1Qbv subcomponent of the TSN specification. Nonetheless, we provide the reader with an overview of their operation in the following points to further clarify the description of the TAS shaper.

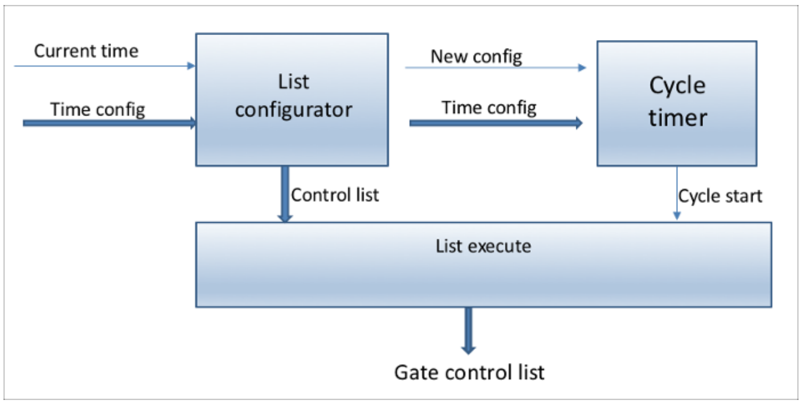


Figure 7.13
The different FSMs controlling the operation of the TAS shaper and the main signals that they exchange to attain a coupled behavior.

7.3.1.1 *The List Configurator FSM*

This is tasked with loading the GCL schedules supplied by the user. Thus, this module interfaces directly with the AXI slave configuration registers of the TAS shaper (see Section 7.3.2) to upload the corresponding settings to the rest of the modules of the core. Hence, as seen in the diagram, this module interfaces directly with its two main functional blocks: the *cycle timer* and the *list execute* FSMs. The duration of the scheduling cycle (*the cycle period*) and the relative time offset after which the scheduling policy should be applied are provided to the former; whereas the latter is supplied with the entries of the GCL schedule containing the different traffic-shaping slots and the corresponding arrays of gate driver state values. Once all of the settings for the user-defined traffic-shaping policy have been defined, then the configuration state machine signals the other FSMs of the core that a new GCL configuration has been supplied, thus triggering the start of the application of a given GCL shaping policy.

7.3.1.2 *The cycle timer FSM*

This is tasked with enforcing the scheduling cycle period that the network users specify when they define a new GCL schedule. Hence, this cycle period is the sum of the duration of all the individual entries that make up the GCL definition. Therefore, in order to ensure that the shaper gets to execute the scheduling cycle supplied through the GCL, this FSM has a direct interface with the local time signal generated at the PTP hardware clock. This time signal is in turn fed into the comparator logic of the module to determine the points in time when the next cycle of a given scheduling policy should be executed. This is the key feature on the foundation of the deterministic forwarding of a TSN system: since the PHC time signals of the TSN nodes are synchronized throughout the network, we can then leverage this property to get the shapers of each node to operate in coordination with each other and thus execute a given scheduling policy synchronously, as opposed to having free-running shapers executing a given cycle without any relative coordination. In our implementation, which follows the IEEE standard guidelines for TAS shaping, we have also included support for delaying the start of the execution of a given scheduling policy by a certain amount of offset. This allows the TAS shaper to initiate its shaping policy at a predefined offset with respect to the UTC origin that is designated as the *base time* in the standard terminology. Lastly, upon detection of the condition for starting a new cycle, the FSM signals the *list execute* module the occurrence of a new "*cycle start*" condition, which causes the "*list execute* FSM to restart the GCL and then sequentially apply all of its entries.

7.3.1.3 *The list execute FSM*

This is the module responsible for applying the different traffic-forwarding intervals (*slots*) defined in a given GCL scheduling policy. The module is activated upon reception of two main sets of signals from the previous FSMs: the configuration list (and configuration status) from the *list configurator*, and the new "cycle start" condition from the *cycle timer*. Hence, the former contains the GCL entries supplied by the user, whereas the latter is the external trigger that causes the FSM to start the execution of the GCL supplied through the *list configurator* signals. Consequently, this module will sequentially fetch each one of the entries contained in the GCL. Each entry, as mentioned previously, will contain an array of gate driver states for each one of the queues of shaper, and a relative time offset indicating when the next entry of the GCL should be fetched. This sequential execution of the GCL entries is responsible for creating a TDMA-like structure whereby each slot can be allocated to the transmission of specific traffic classes from the queues of the shaper.

7.3.1.4 *Relationship to the operation of the TAS core*

The relationship between the foregoing state machines and the operation of the TAS shaper is depicted thoroughly in the diagram of Fig. 7.14. Hence, as shown in the figure, the TAS core has to be programmed first with the appropriate user settings and parameters, which have to be uploaded to its AXI slave configuration registers in order to let the core execute a certain policy. Next, the policy itself (scheduling cycle, GCL entries) will be processed by the TAS FSMs as indicated in the previous points. Lastly, the gate driver states from the corresponding GCL entry fetched at the list execute machine will be applied to the queues of the shaper to form the appropriate queue configuration for a given slot within the scheduling cycle.

7.3.2 *Configuration of the time-aware traffic shaper*

As depicted in the diagram of Fig. 7.14, the different components of the TAS core have to be programmed before the shaper can start applying any traffic forwarding policy. In our design, this can be achieved by having the user upload the appropriate configuration parameters to the AXI-Lite slave configuration register. Hence, we have 14 different internal registers that the user should configure whilst adhering to a custom AXI write/read protocol to allow the core to operate properly and execute the supplied GCL policy as expected. Thus, an overview of the contents of the slave configuration registers for the TAS shaper can be examined in Table 7.25. The contents of each register from the AXI configuration slave can also be examined in the following points.

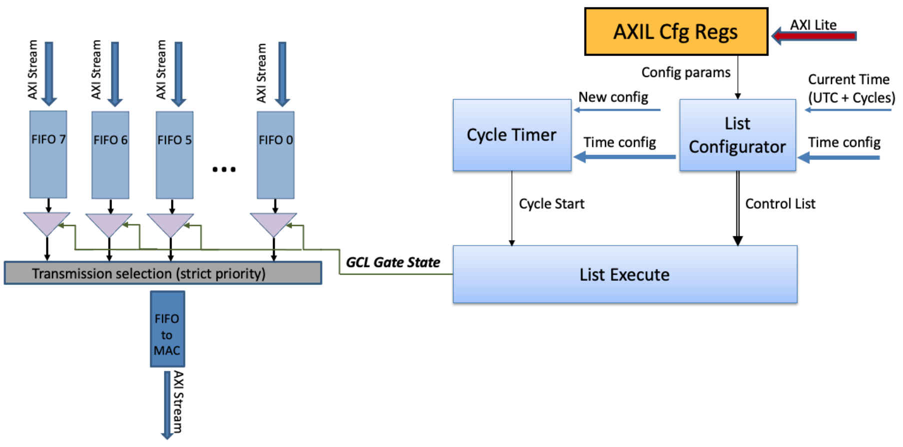


Figure 7.14
Relationship between the TAS architecture and its FSM modules for implementing a deterministic traffic forwarding component.

Table 7.25
The layout of the TSN TAS slave configuration registers.

Slave register map for the TSN TAS core.

Register name	Functionality	C offset
`Slave 0`: FIFO Configuration	Configuration of the queues of the traffic shaper.	0x0
`Slave 1`: Time interval configuration	Time interval length of a GCL slot.	0x4
`Slave 2`: Time interval used (current gate)	Report of the currently used time interval length during the execution of a GCL schedule.	0x8
`Slave 3`: Control list length	The number of entries contained in a GCL schedule.	0xc
`Slave 4`: Cycle time	The cycle time of a GCL schedule	0x10
`Slave 5`: Admin Cycle time extension	The sub-second value of the GCL cycle time	0x14
`Slave 6`: Operation cycle time extension	Report of the currently used cycle time during the execution of a GCL scheduling policy.	0x18
`Slave 7`: Base time (bits 31:0)	Bits 31:0 of the base time offset field before the start of the TAS core operation.	0x1c
`Slave 8`: Base time (bits 63:32)	Bits 63:32 of the base time offset field before the start of the TAS core operation.	0x20
`Slave 9`: Base time (bits 67:64)	Bits 67:64 of the base time offset field before the start of the TAS core operation	0x24
`Slave 10`: Operation base time (bits 31:0)	Report of bits 31:0 of the base time offset field before the start of the TAS core operation.	0x28
`Slave 11`: Operation base time (bits 63:32)	Report of bits 63:32 of the base time offset field before the start of the TAS core operation.	0x2c
`Slave 12`: Operation base time (bits 67:64)	Report of bits 67:64 of the base time offset field before the start of the TAS core operation	0x30
`Slave 13`: Error register	Error codes associated with the operation of the TAS core.	0x34

7.3.2.1 *Slave 0: the FIFO configuration register*

This configures several options for the different queues of the shaper. Its configuration fields can be examined in Table 7.26, whereas their corresponding meaning is outlined in Table 7.27.

Table 7.26
Slave 0: the configuration fields of the FIFO configuration register.

0 1 2	3 4 5	6 13	14 21	22	23	24	25	26	27 31
Unused	Tick Granularity	Oper Gate St	Admin Gate St	Gate En	Table Full	Prog Table	Reset FIFOs	New Input	Unused

Table 7.27
Slave 0: the contents of the fields of the FIFO configuration register.

Description of the FIFO configuration register fields.

Bits	Description
31:27	Unused
26	**New input. [RW].** When set to "1", the FSM knows there is a new gate control list ready to start operation. The GCL must be supplied **before** writing this bit.
25	**Reset FIFOs. [RW].**
24	**Programming Table. [RO].** Indicates that the GCL table is currently being programmed.
23	**Table Full. [RO].** Indicates that the programming of the GCL table has completed.
22	**Gate Enabled. [RW].** Enables operation of the TAS gate drivers for executing a GCL scheduling policy. Produces the following effects on the operation of the gates. – ***"1"***: Enabled. Time shaper operation allowed. – ***"0"***: Disabled. Gates are always open, no time-shaping processing is applied.
21:14	**Admin Gate State. [RW]**. Array of gate driver state values. These 8 bits define the driver state for each of the eight possible queues of the shaper. Thus, a *"1"* indicates an open queue, and a *"0"* refers to an inactive queue. If fewer queues are used, then only the least significant bits will be considered.
13:6	**Oper Gate State. [RO]**. Reports current gate driver state configuration.
5:3	**Tick Granularity. [RW]**. Sets the desired system clock granularity value.
2:0	Unused.

7.3.2.2 *Slave 1: the time interval configuration register*

This supplies the time interval, i.e., the slot duration for a given entry of the GCL schedule. The time interval is expressed in terms of clock *ticks*. As a result, a value for the clock granularity should be supplied during the programming process as well. Once the value for the time interval has been supplied, its value should be added to the GCL schedule by means of the FIFO control register. The structure of the register can be examined in Table 7.28.

Table 7.28
Slave 1: the fields of the time interval configuration register.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29	30 31
Time Interval	Unused

7.3.2.3 *Slave 2: the used time interval register*

This implements a core operation status register that can be read by the processor as required. This allows the processor software to determine the interval length being currently applied during the execution of a given GCL policy. As before, the returned value is given in terms of system clock *ticks*, thus implying that the system clock granularity should be known beforehand. The structure of the register can be examined in Table 7.29.

Table 7.29
Slave 2: the fields of the used time interval register.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Time Interval																														Unused	

7.3.2.4 *Slave 3: the control list length register*

This supplies the applicable length of a given GCL schedule. Its structure can be examined in Table 7.30, and the contents of each field are outlined in Table 7.31.

Table 7.30
Slave 3: the fields of the control list length register.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Oper Ctrl List Len								Admin Ctrl List Len								Unused															

Table 7.31
Slave 3: the contents of the fields of the control list length register.

Description of the control list length register fields.

Bits	Description
31:16	Unused
15:8	**Admin Ctrl List Length. [RW].** The value of the control list length (“*admin*” value) written by the processor. It must be programmed before writing the gate control list.
7:0	**Oper Ctrl List Length. [RO].** The length of the control list being currently executed by the processor (the operation – “*oper*” – value).

7.3.2.5 *Slave 4: the cycle time register*

This is used for specifying the duration of the traffic-shaping cycle of a given GCL policy. In accordance with the format of the TAI signal from the PHC, the cycle time of the TAS shaper is expressed as a combination of a seconds field and a “sub-second” field (the cycle time *extension*). Hence, this configuration register specifies the seconds field of the scheduling cycle duration. Its structure can be examined in Table 7.32, and the contents of its fields can be seen in Table 7.33.

Table 7.32
Slave 4: the fields of the control list length register.

0 1 2	3 4 5	6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Oper Cycle Time	Admin Cycle Time	Unused

Table 7.33
Slave 4: the contents of the fields of the cycle time register.

Description of the cycle time register fields.

Bits	Description
31:6	Unused
5:3	**Admin Cycle Time. [RW].** Admin cycle time written by the processor. The total time to execute the whole gate control list (in seconds). If programmed correctly, this will cause the periodic execution of the entries of the gate control list with the periodicity indicated in this field.
2:0	**Oper Cycle Time. [RO].** The cycle time for the gate control list being currently executed by the processor.

7.3.2.6 *Slave 5: the cycle time extension register*

As indicated previously, this register specifies the sub-second field of the scheduling cycle duration for a GCL policy. The cycle extension should be supplied in nanoseconds, which are internally translated into clock *ticks* according to the designated clock granularity. The structure of the register can be examined in Table 7.34.

Table 7.34
Slave 5: the fields of the cycle time extension register.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29	30 31
Admin Cycle Time Extension	Unused

7.3.2.7 *Slave 6: the operational cycle time extension register*

This is a status register for reporting the currently used cycle time extension value at the TAS shaper. The operation cycle time extension is reported in nanoseconds to the software modules accessing the configuration parameters of the TAS core from the processor. The structure of the register can be examined in Table 7.35.

Table 7.35
Slave 6: the fields of the operational cycle time extension register.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29	30 31
Oper Cycle Time Extension	Unused

7.3.2.8 *Slave 7: the administrative base time register (least significant bits)*

This register contains the lest significant bits (LSB) of the *base time* configuration parameter. As explained previously, this allows the user to specify a relative time offset before the TAS core can start running a given GCL. Its value is supplied in the standard UTC format (seconds and clock cycles). The contents of this register are shown in Table 7.36.

Table 7.36
Slave 7: the structure of the LSB administrative base time field contained in the register.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Admin Base Time (31:0)																															

7.3.2.9 *Slave 8: the administrative base time register (intermediate bits)*

This register contains the intermediate bits of the administrative base time of the traffic shaper, which is supplied in the standard UTC format (seconds and clock cycles). Its structure is shown in Table 7.37.

Table 7.37
Slave 8: the structure of the register with the intermediate bits of the administrative base time field.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Admin Base Time (63:32)																															

7.3.2.10 *Slave 9: the administrative base time register (most significant bits)*

This register contains the most-significant bit (MSB) field of the base time configuration parameter of the TAS shaper, which is supplied in the standard UTC format (seconds and clock cycles). Its structure is in turn depicted in Table 7.38.

Table 7.38
Slave 9: the structure of the configuration register containing the MSB field of the administrative base time.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Admin Base Time (67:64)					Unused																										

7.3.2.11 *Slave 10: the operational base time register (least significant bits)*

This is a status register for reporting the current base time at the TAS core. The register reports the initial 32 bits of the base time value (LSB part). Its structure is in Table 7.39.

Table 7.39
Slave 10: the structure of the status register containing the LSB field of the operational base time

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Oper Base Time (31:0)																															

7.3.2.12 *Slave 11: the operational base time register (intermediate bits)*

The intermediate bits of the operational base time value for reporting the currently used base time to the processor. Its structure can be examined in Table 7.40.

Table 7.40
Slave 11: the structure of the register containing the intermediate bits of the operational base time.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Oper Base Time (63:32)																															

7.3.2.13 *Slave 12: the operational base time register (most significant bits)*

This register contains the MSB field of the operational base time value for reporting the currently used base time to the processor. Its structure can be examined in Table 7.41.

Table 7.41
Slave 12: the structure of the register containing the MSB field of the operational base time.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Oper Base Time (67:64)					Unused																										

7.3.2.14 *Slave 13: the error register*

This contains the error condition field, which can signal the occurrence of different types of issues during the execution of a schedule or during the general operation of the TAS shaper. In our design, the error field contains 8 bits. This allows us to raise an error signal for any of the three possible conditions that we have contemplated in our implementation; although the unused bits in the field would still allow for the consideration of additional error conditions. The error codes considered for our current implementation are listed in Table 7.43, and the structure of the register can also be seen in Table 7.42.

Table 7.42
Slave 13: the structure of the error code register.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Error Code								Unused																							

7.3.2.15 *The configuration sequence of the time-aware traffic shaper*

The preceding registers have to be programmed with the TSN configuration API, which is tasked with supplying the traffic class identification parameters as well as uploading the corresponding GCL settings to each one of the TAS shapers in the system. Hence, in our design, the following sequence should be applied to the configuration registers of the TAS core from the corresponding configuration utility running on the processor to ensure its correct operation.

1. Write the control list length.

Table 7.43
Slave 13: Possible values of the TAS error codes and their meaning.

Possible TAS error codes.

Error Code	Description
`0x1`	Attempted write to a FIFO queue before the read operation has finished.
`0x2`	Transmission overrun condition.
`0x3`	Error while applying the core settings (*configuration error*).

2. Write the base time value.
3. Write the cycle time and its cycle time extension.
4. Supply all the rows of the GCL schedule in sequence. For each entry of the GCL, the interval (slot) duration must be written first.
5. Write the gate state array – the FIFO configuration of the shaper – associated with the GCL entry (bits 21:14, Slave_0 register). Bit 26 of the Slave_0 register must be set to "0" during this operation. Once the gate state array has been supplied, then the configuration FSM of the AXI Slave registers of the core is ready to accept the next GCL entry.
6. Repeat steps 4 through 5 until the entirety of the GCL has been uploaded to the core.
7. Once the full GCL has been written, the configuration utility should signal the list configurator FSM to start configuring the rest of the components of the TAS shaper and thus launch its operation. This is accomplished by writing a "1" to bit "26" of the FIFO configuration register. Likewise, bit "22" must also be set to "1" to enable the operation of the TAS module. In addition, caution should be exerted with regards to the clock granularity supplied during the configuration process, as the configuration FSM will only acknowledge the granularity value of the last GCL entry written to the FIFO configuration register, and then it will apply it to all the entries of the GCL while uploading the corresponding settings to the TAS core.
8. To signal the end of the configuration process, the configuration utility must write a "0" in bit 26 of the FIFO configuration register. Otherwise, the FSM will become stuck in a wait state for new configuration parameters. The rest of the bits in the register must be kept to their last value.
9. All set! Once all the foregoing steps have been applied, the *list configurator* will kickstart the operation of the *cycle timer* and *list execute* FSMs of the core to start the deterministic forwarding of all the TSN traffic classes that are present in the system.

7.4 TRAFFIC PREEMPTION: ENHANCEMENTS FOR THE TAS AND THE ETHERNET MAC CORES

TSN systems, especially those designed for industrial or avionics applications, often have to guarantee the delivery of highly critical data with a large degree of determinism. The guaranteed delivery service of these critical flows is taken care of by the use of the enhancements for seamless redundancy that we introduced in Section 7.2, which avoid the loss of highly critical data in the event of network congestion or even if the transmission path is severed given their use of a frame replication scheme with redundant routing. Thus, this is an enhancement for the system robustness. In this context, enforcing a deterministic delivery is the job of the time-aware shaper of the TSN system, which can provide bounded end-to-end latency by applying a cyclic time-driven schedule. Some select types of traffic such as those found in the highly critical applications of industrial automation or avionics may also require that the data not only be delivered within a specific time bound, but with minimized latency variation as well. This can be accomplished with the use of the enhancements for preemption and the interspersing of express traffic (802.1Qbu [41] & 802.2br [40]), which can enormously boost the degree of determinism of a TSN network.

The basic notion behind the use of frame preemption is that this mechanism should allow the interruption of lower priority Ethernet data frames to favor the transmission of highly critical – express – frames, as depicted in Fig. 7.15. In practice, this avoids the degradation of the deterministic delivery of the critical flows by mitigating the effects of interfering, lower priority traffic. Therefore, this methodology allows the segmentation and the subsequent reassembly of the lower priority frames directly over the link-layer Ethernet service when they come into conflict with the critical *express* traffic.

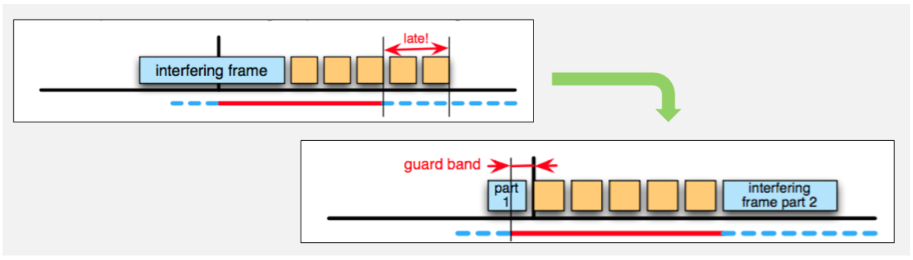


Figure 7.15
A representation of the effects of the frame preemption mechanism to avoid interfering traffic and the degradation of the system determinism. Adapted from [166].

The use of these features for preempting the lower priority traffic has several implications on the design of the main components of the TSN system, as this functionality was conceived as an incremental upgrade to the operation of the time-driven shaper of TSN (802.1Qbv) and to its underlying Ethernet link layer (IEEE 802.3). Consequently, in order to benefit from the use of frame preemption in our experiments, like those from Chapter 9, we had to introduce a number of modifications to our original implementations of the TAS shaper and the Ethernet MAC cores, which we present in the following points. These enhancements were supplied by our collaborators from the Andalusian Institute of Astrophysics (IAA) and our industrial partner Seven Solutions for the preemptable TAS shaper (802.1Qbu). Seven Solutions was also responsible for providing the design and implementation of the 1-Gb/s preemptable Ethernet MAC (802.3br). These components were used in the context of the experiments that we present in Chapters 8 and 9.

7.4.1 *The time-aware traffic shaper with frame preemption*

As stated previously, the use of the frame preemption feature has the objective of supplying a method whereby the transmissions of express traffic should always preempt the transmission of the lower priority messages; i.e., pause the forwarding of non-critical messages and resume their transmission at a later time once all the express traffic has been sent. In order to add support for this functionality to our initial implementation of the TAS shaper (Section 7.3), we had to introduce modifications to its queueing model, gate driver and queue selector behavior, as well as changes to its internal control logic and its interfaces with other external components. Some of these changes include the addition of a *preemption status table*, multiple changes to its internal FSMs, new configuration fields for the AXI slave registers, or the implementation of expanded communication interfaces with the Ethernet MAC, such as new signaling connections and bus interfaces for sending the express and preemptable data over separate transmission paths. The resulting architecture can be examined in Fig. 7.16.

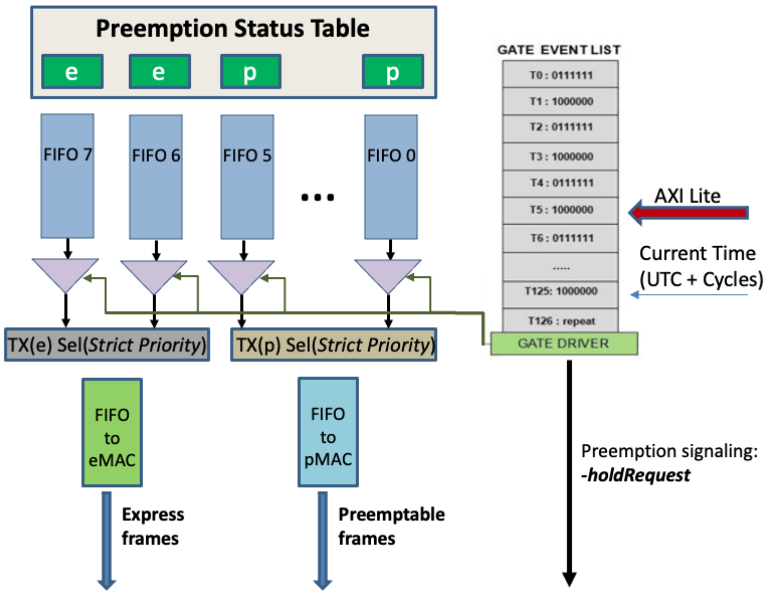


Figure 7.16
The resulting architecture of the TAS shaper core after implementing all the necessary changes for supporting the use of frame preemption.

As seen in the figure, the changes to the TAS core can be summarized along the following lines.

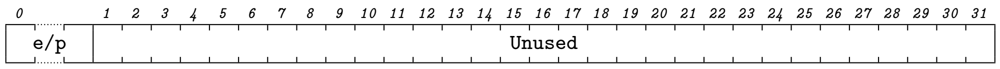
- **The use of a renewed queueing model**. With the use of frame preemption, the queues of the TAS shaper can potentially be designated as either express (*e*) or preemptable (*p*). The preemptable queues will handle the low-priority flows, whereas the express queues will be used for forwarding the highly critical data frames. Each type of queue will in turn interface with a revamped *strict priority selection mechanism* that will prioritize the selection of express frames when different types of traffic flows contend for access to the Ethernet link. This allows the execution of GCL scheduling policies that can activate both types of queues simultaneously throughout the execution of their respective entries.

- **The strict priority selection** mechanism of the TAS shaper will now be split into two different branches: one for the express queues and another one for the preemptable queues. As was the case with the original 802.1Qbv shaper, the strict priority mechanism is intended to select data for transmission from the higher priority queue amongst all the queues that it handles which are open at a given time. Therefore, the use of separate selectors for both types of queues will allow the simultaneous forwarding of the higher priority express and preemptable frames, respectively, to the underlying Ethernet MAC at any given time during the execution of a GCL schedule. As we will see in 7.4.2, the Ethernet MAC will then arbitrate between both types of packets (*e/p*) as needed.
- **The implementation of changes to the internal control logic** of the shaper, most notably the design of a **preemption status table**. Hence, the former will include changes to the design of the internal FSMs of the TAS shaper to allow the designation of its queues as either express or preemptable, as well as to allow their respective activation during the execution of a given GCL. The latter is an auxiliary table allowing users to supply an array of preemption status values for each queue of the shaper. This will result in each queue being assigned to one particular strict priority selector depending on the value of its preemption status. Furthermore, the queueing elements with an *express* designation will be given priority for forwarding their messages over those from the preemptable queues.
- **The implementation of additional interfaces with the Ethernet MAC module**. This can be seen in Fig. 7.16, where we show the main modifications that we built into the TAS shaper to enable the use of frame preemption. These changes are twofold and, thus, require the deployment of additional signaling and bus interfaces.
  - The use of a ***new signaling channel with the underlying MAC*** is visible in the figure. In our design, we used this channel for sending "*holdRequest*" signals to the preemptable Ethernet MAC. These signals are part of the IEEE specification for frame preemption and are meant to be sent during the execution of each GCL entry upon the reception of new critical frames bound for the express queues that are open during a given time slot. Thus, the "*holdRequest*" signal can take two possible values: when a value of "1" is issued, the shaper indicates the preemptable MAC that the forwarding of lower priority messages should be halted to allow the transmission of critical express frames. Conversely, a value of "0" signals that the forwarding of the lower priority frames should be allowed to proceed undisturbed. It can be seen that the use of this mechanism is largely responsible for the enhanced determinism of frame preemption, as it prevents the transmission of a lower priority frame from spilling onto the slots reserved for express traffic and thus clash with the critical messages of the system.
  - The use of duplicated bus interfaces for injecting Ethernet frames into the preemptable Ethernet MAC is also evident in the picture. Thus, as the new

architecture design for the TAS shaper effectively divides the processing of the express and preemptable data over two separate data paths, then each path has its own separate bus interface for connecting to the Ethernet MAC as well. Hence, as seen in the diagram, we have built an *express* AXI-Streaming bus interface for injecting the *express* critical frames from the priority selector for the express queues into the Ethernet MAC. Likewise, there is also another AXI-Streaming interface for delivering the low-priority preemptable data. This allows the TAS shaper to process both types of messages separately before sending them to the Ethernet MAC service, which would then arbitrate between and apply packet segmentation as needed before they can be granted access to the physical Ethernet link. Additionally, the enhancements for greater determinism and lower latency of frame preemption could also potentially allow for the implementation of an additional *cut-through* interface that could effectively bypass the queueing mechanism of the shaper. Even though this would further reduce the latency variation of highly critical flows, we opted to leave the implementation of this feature outside of the scope of our design to further simplify the architecture of the Ethernet MAC arbitration mechanism and thus avoid making excessive use of FPGA resources.

- Lastly, the TAS shaper also required changes to its internal configuration logic. This was the case of the AXI-Lite slave configuration registers, which had to expand their fields so that the values of the preemption status table could be supplied during the configuration of new GCL scheduling policies. Specifically, we have supported the definition of preemption status values for each queue by adding eight additional configuration registers, with each register dedicated to each queue of the shaper out of a possible maximum number of up to eight queues. The structure for each register is relatively simple and can be examined in Table 7.44.

Table 7.44
Slaves 14-21 : the preemption status registers.



7.4.2 *The preemptable Ethernet MAC*

The enhancements for preemption also extend to the overall design of the MAC core for accessing the physical Ethernet links of the system. These changes require a thorough redesign of the standard Ethernet MAC, which will now have to include an arbitration mechanism between the *express* and the *preemptable* traffic flows; as well as the necessary segmentation and reassembly logic for handling the fragments of preemptable messages on the fly. Hence, in order to use these features in our experiments, like those from Chapter 9, we had to replace the original Xilinx Ethernet Tri-Mode MAC from the prototyping stages with a customized version of a 1-Gb/s Ethernet MAC from the Open Cores repository [148] instead as mentioned in 6.4.2. This MAC module has a lower FPGA usage footprint, thus allowing us to bundle with it all the necessary additional elements for supporting frame preemption without making excessive use of FPGA resources. The resulting architecture of the new preemptable Ethernet MAC and its main features can be seen in the diagram of Fig. 7.17.

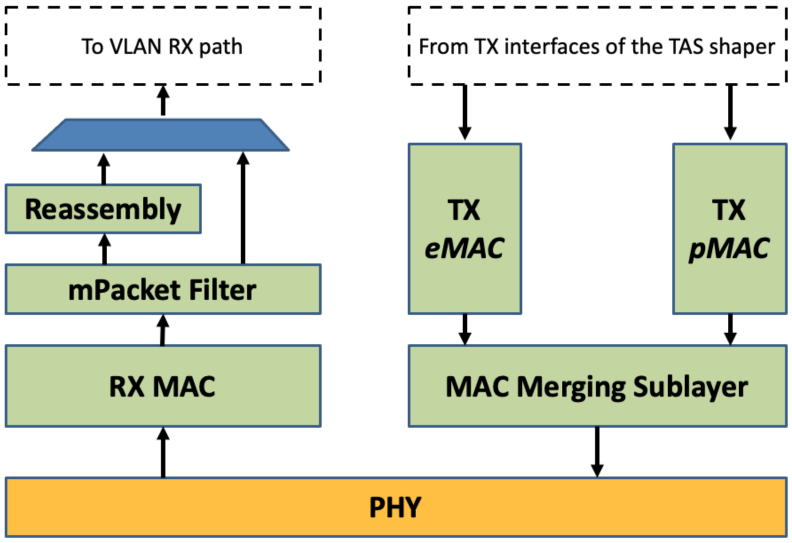


Figure 7.17
The block diagram of the new preemptable MAC, featuring two dedicated paths for transmitting express and preemptable traffic, arbitration and segmentation mechanisms between these paths, and the reassembly logic on the reception path.

As observed in the figure, the new MAC design consists of three main elements: the new transmission interfaces with the preemptable TAS shaper, a MAC merging sublayer, and the reassembly logic on the reception path. Hence, the transmission data path was duplicated with an interface for an express MAC (*eMAC*) and another interface for a preemptable MAC (*pMAC*), thus allowing us to interface with the corresponding express and preemptable interfaces of the TAS core, which had its internal data path duplicated as well. Consequently, this extended MAC service can provide differentiated access to the Ethernet transmission links by applying two different user-selectable forwarding methods: express low-latency forwarding, or low-priority preemptable forwarding. Yet, in the event that the user did not specify a preference for the Ethernet MAC forwarding method, then the TAS shaper would inject all data through the express port of our MAC core. This would result in the Ethernet MAC handling all transactions as express data to maintain backward compatibility with the ordinary Ethernet specification.

The arbitration mechanism resides in the MAC merging sublayer. This component is one of the main modifications implemented over the legacy Ethernet service as a result of adding support for the frame preemption features. Thus, in our design, the merging sublayer features a bus arbitration mechanism for selecting the transmission of either express or preemptable frames. Moreover, this arbiter has the ability to stop the transmissions of preemptable frames from the *pMAC* and start their subsequent segmentation to avoid interference with data injected from the *eMAC*. These fragmented frames are known as "*mPackets*" in the terminology of the IEEE standard. Overall, we have designed this arbiter system to prioritize the transmission of express traffic to reduce its delivery jitter.

As for the reception data path of the MAC, it is one of the few components that remain relatively unchanged with respect to the original MAC from Open Cores. Nonetheless, it also had to undergo minor changes as we had to implement a simple packet filter to detect fragmented frames and then supply the corresponding reassembly logic, which consisted of a simple FSM and a temporary buffer.

Thus, after providing a glimpse into the operation of the new preemptable MAC, its main components, and the relationship between them, we also give an in-depth explanation and a thorough overview of their operation in the following points.

7.4.2.1 *The preemptable message format*

The Ethernet messages injected from the TAS shaper core over the preemptable MAC could potentially be subjected to the use of preemption by segmenting their data into several fragments. Thus, according to the standard, this preemption and segmentation process can take place for a given frame as many times as necessary in the event that it is preempted by higher priority traffic so long as two main conditions hold true for the remaining of the preempted frame: that at least the initial 60 B of the frame have already been transmitted, and that more than 64 B of the frame still remain to be transmitted. Otherwise, segmentation will not occur.

The operation of the segmentation and reassembly tasks of the preemptable frames is controlled with the use of a signaling mechanism that combines the calculation of partial cyclic redundancy checks (CRCs) with the inclusion of new delimiters and fragment count fields into the Ethernet frame. Hence, this allows the reassembly logic to ensure the orderly reception of fragments or detect when the last fragment of a preempted message is received. Since the use of preemption requires that the Ethernet frames for the preemptable messages include these new fields, the resulting frames are designated "mPackets" and their fields can be examined in Fig. 7.18.

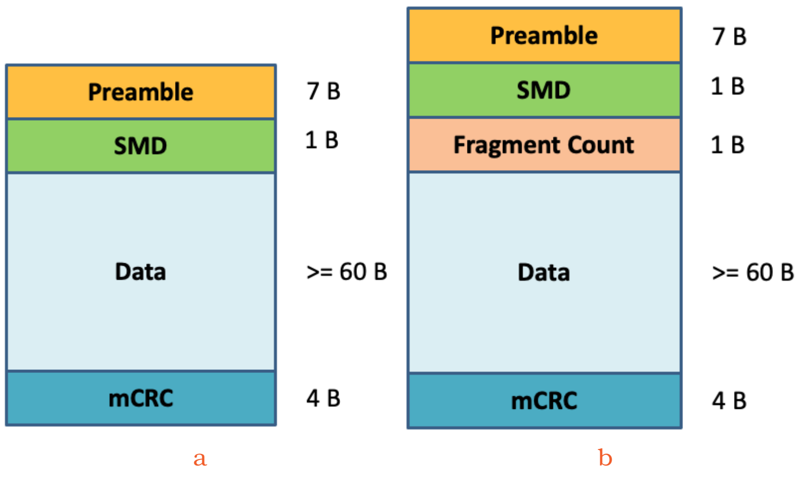


Figure 7.18
The internal structure of "mPackets" for the frames carrying the first segment of a preempted message (Fig. 7.18a), and the structure of the subsequent frames carrying the "continuation" segments to a fragmented message (Fig. 7.18b).

7.4.2.2 *Segmentation signaling for packet preemption*

We use a relatively simple signaling mechanism with frame preemption to keep track of all the segments of a preempted frame in the case that it had to segmented to avoid interference with express traffic. This signaling scheme uses custom delimiter values and a partial CRC for determining the reception of the last fragment of a given packet. This can be examined in Table 7.45, which contains the possible values that the different Ethernet preambles, fields, and delimiters are supposed to take when preemption is used.

Table 7.45
Values of the SMD delimiter for each type of "mPacket": the initial fragment of a preemptable message, or the subsequent "continuation" fragments of a segmented message.

Possible values for the SMD delimiter.

mPacket type	**SMD name**	**Frame count**	Value
Express packet	SMD-E	-	`0xD5`
Packet Start	SMD-S0	0	`0xE6`
	SMD-S1	1	`0x4C`
	SMD-S2	2	`0x7F`
	SMD-S3	3	`0xB3`
Continuation Fragment	SMD-C0	0	`0x61`
	SMD-C1	1	`0x52`
	SMD-C2	2	`0x9E`
	SMD-C3	3	`0x2A`

Since the preemptable Ethernet MAC is required to maintain backward compatibility with the ordinary Ethernet specification, which does not support *layer-2* (L2) data fragmentation, the core relies on the use on a particularized signaling technique that indicates the reception of the different parts of a segmented frame by means of supplying specific Ethernet frame delimiter values, other than the standard value of `0xD5` for ordinary Ethernet frames, for each part of the fragmented message. This can be seen in conjunction with the diagrams from Fig. 7.18. Hence, when the preemptable MAC indicates the transmission of the initial fragment of a preempted message by producing a frame with the format from Fig. 7.18a that is marked with the use of one of the corresponding segmentation start delimiters (*SMD-Si*). Likewise, the subsequent frames of a segmented message will use the format from Fig. 7.18b and will be marked by the use of their corresponding segment continuation delimiter (*SMD-Ci*). In addition, each segment continuation delimiter is in turn associated with a corresponding segmentation start delimiter (*SMD-Si*) for a given preempted message. Furthermore, the segment continuation frames are also fitted with an additional fragment counter field ("*Frag Count*") for use in the reassembly process and for keeping track of the number of segments that a given message is split into. The values of the counter field are also normalized, and we show them in Table 7.46.

Thus, given the maximum transmission unit (MTU) of standard Ethernet frames of up to 1500 B, the specification for frame preemption contemplates a maximum of up to

Table 7.46
The values of the fragment counter field as a function of the fragment number.

Values of the fragment counter.

Fragment Count	Value
0	0xE6
1	0x4C
2	0x7F
3	0xB3

four different fragments in practice for transmitting the segmented parts of a preempted message. Moreover, since this signaling scheme is based on the use of different values for the Ethernet delimiter field, it has the advantage of retaining backward compatibility with ordinary Ethernet frames, which are designated with the usual value of `0xD5` for their delimiter. Thus, when preemption is not used, the messages injected into the MAC are forwarded as express frames and therefore make use of the ordinary Ethernet frame format; i.e., the express messages use the ordinary Ethernet frame delimiter value of `0xD5` for the SMD field and are sent over an "*mPacket*" with the format from Fig. 7.18a.

7.4.2.3 *Arbitration on the transmission path: the MAC merging sublayer*

The last component on the transmission data path before the "*mPackets*" can be injected into the physical layer, which is unaware of the use of preemption, is the arbitration process of the merging sublayer. Hence, this module arbitrates between transactions originating from either the *pMAC* or *eMAC* submodules, thereby granting priority to the express messages for accessing the transmission medium. This implies that, in the event of a conflict between a preemptable message and an express frame, the former message, which is usually lower in priority as well, will be preempted in favor of the express frame and its segmentation will be triggered. Nonetheless, once the transmission of a preempted message is underway, it cannot be preempted again until the minimum Ethernet frame size has been reached.

7.4.2.4 *Reassembly on the reception data path*

Lastly, the reassembly of fragmented *mPackets* is performed on the reception data path of the Ethernet MAC, which we fitted with additional packet filtering and reassembly logic in our design. Thus, on the one hand, our implementation forwards the express messages directly to the upper layer components of the TSN node, as they do not require any additional post-processing. On the other hand, the different segments of the preempted messages received from the network are handled by the reassembly logic. This logic uses a temporary buffer to store the segments of mPackets with matching SMC-Si and SMD-Ci delimiter values; i.e., the start frame and all the subsequent continuation frames of a segmented message. The fragment count is then used internally to keep track of the size of all the reassembled fragments, and the last fragment is detected by inspecting the value of the CRC field. Thus, an express frame or a preemptable message that was transmitted without segmentation will contain the value of the CRC of the entire message in the corresponding CRC field. On the contrary, fragmented *mPackets* will replace the CRC with a temporary value (the *mCRC*) for all frames, except for the last fragment, which will contain the CRC of the whole preempted message. The *mCRC* consists of a partial checksum of the contents of the current fragment which is then applied to an `XOR` with the `0x0000FFFF` value. Hence, the internal reassembly logic can detect the reception of the last frame of a preempted message when the partial checksum of the reassembled data matches that of the current mPacket. This condition would signal that the reassembly operation is complete and that the reassembled frame can be forwarded to the upper-level cores for further processing.

7.5 THE SWITCHING ELEMENTS

Although they are not strictly a part of our TSN core design, our nodes are multi-port devices with bridging capabilities allowing them to forward data amongst their Ethernet ports, or between the ports and their embedded ARM processor. Thus, in our design, controlling the flow of traffic to determine how the TSN frames handled at a specific node should be routed is the responsibility of our TSN cores. The routing itself is taken care of by the use of dedicated switching elements, which in our implementation were the off-the-shelf switching cores from Xilinx [150]. These switches come with all the necessary bus adaptation and conversion stages between the subsets of the AXI bus protocol as needed. A schematic diagram of their implementation can be seen in Fig. 7.19.

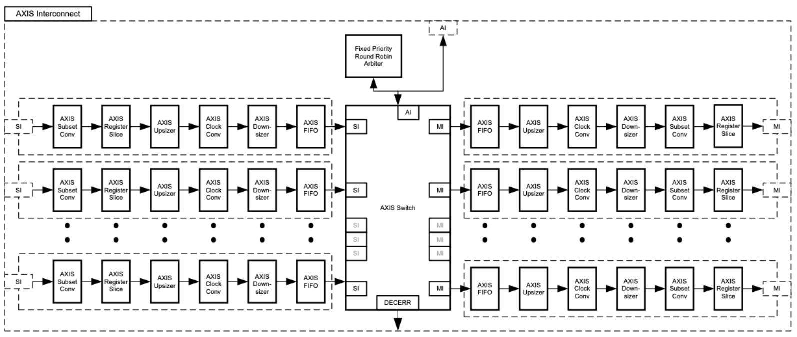


Figure 7.19
The internal structure of the AXI-Streaming crossbar interconnects used as the main switching elements of our nodes.

These switches are AXI-Streaming crossbar switches with a *round-robin* arbitration policy. They are responsible for routing data frames between their AXI master and AXI slave ports as indicated by the value of the "*tdest*" signal of the AXI-Streaming bus. This signal is in turn supplied by the routing logic of TSN VLAN module, which we have endowed with a traffic redirection feature that allows the implementation of the bridging functionality of our nodes.

Resource consumption can be an important issue with this type of switches. Hence, their resource usage is bound to grow exponentially when larger switches are required. Thus, their FPGA usage is usually driven by the logic elements demanded by the implementation of the crossbar switching matrix, which mainly uses LUTs and flip-flop (FF) primitives. Consequently, their resources are clearly a function of their size, expressed as $M \times N$ (M slaves to N master ports). Moderately sized crossbars such as 1x2 switches can fetch LUTs on the order of the hundreds of primitives, which is relatively low for the devices we have worked with, whereas matrices larger than 4×4 may quickly cannibalize a substantial amount of resources. That is the reason why we opted for a two-tier architecture, as shown in Fig. 6.2, where we have an initial switching stage for transmission and reception, and a forwarding stage – the "*Redirector*" – for implementing the bridge functionality or to send data between the processor, the node Ethernet ports, or even third party IP cores in the FPGA logic.

We aimed to produce small crossbars with this strategy, such as $1 \times M$ or $2 \times M$ switches, which usually result in a lower footprint. We prioritized this aspect throughout our work as having low resource usage was part of the requirements of the projects described in Chapters 8 and 9, where we had the opportunity to conduct our experiments. Since this was a major requirement, it had multiple implications that, in the case of the switching elements, materialized as the two-level switching architecture that we have mentioned. At any rate, this is effective for conserving resources, but the use of crossbars with arbiters may introduce undesired delays in the processing of the TSN messages that may degrade the deterministic performance of the system. This is one of the aspects for improvement that we may study during continuing work after this thesis project; such as the replacement of all the packet switching crossbars with shared memory switches.

7.6 comments on the firmware versions and their node variants

We have used the preceding elements to implement the TSN subsystems of the different types of nodes that we built for our experiments. As stated in previous sections, we have made use of two main node variants.

- The WR-ZEN board was our platform for prototyping, verification, and debugging. We also used it to support our tests for integrating White Rabbit timing and a TSN networking system from Chapter 10.
- The Main Board is the platform where we developed the low-cost avionics solution from Chapter 9 and the nodes for the Smart Grid from Chapter 8.

In both cases, we had to design TSN systems with moderate resource usage by combining the different elements of the TSN subsystem that we have introduced in this chapter. This requirement stemmed from different facts for both platforms.

- In the case of the WR-ZEN board, the moderate footprint was a consequence of the fact that this node uses the Z-7015 SoC from Xilinx, which is a low-cost, albeit relatively small, device.
- In the case of the Main board, since it was originally conceived for the development of the avionics of a microlauncher, it was a requirement that the TSN implementation should not exceed 50 % average utilization of the Z-7030 device. This would allow the inclusion of additional mission-related IP cores in the design.

As a result, we eventually designed three different firmware variants for these platforms under these requirements: one for the WR-ZEN board, and two for the Main board for its avionics and its Smart Grid versions, respectively. In all cases, we supplied the implementation of the TSN system with a lightweight design – TSN "*Lite*" – with moderate resources, although with some of the design tradeoffs that we have mentioned (e.g., VLAN sharing between ports, crossbars with arbitration, reduced VLAN tagging rules).

The architecture of each firmware variant is discussed in detail in the corresponding sections of Chapters 8 and 9 (for the Main board), and Chapter 10 for the WR-ZEN. Meanwhile, we outline the individual resource consumption of each component of the TSN system in Section 7.7.

7.7 RESOURCE CONSUMPTION

Lastly, to further contextualize our TSN implementation, we have gathered the raw utilization figures of the different cores of the TSN subsystem. This should give the reader a general idea of the broad range of Xilinx devices that could somehow incorporate either a full version of the TSN system or a partial version of it with support for a reduced subset of TSN (e.g., TAS shaping only without redundancy). Thus, the FPGA utilization of each core can be examined in Table 7.47, where we also compare them to their relative usage in a Z-7030 device. Furthermore, the overall utilization in the Z-7015 and Z-7030 devices that we used in our experiments is discussed in their corresponding chapters. Nonetheless, to estimate the resource consumption of our architecture on their platform, the reader should anticipate that the following elements are usually needed for an n-port design: $\frac{n}{2} \cdot (VLAN) + n \cdot (TAS + MAC) + Dropper + Crossbars$.

Table 7.47
The overall utilization of the TSN cores. Percentage usage figures with respect to a Z-7030 device for reference.

Module	**Slice LUTs**		**Slice Registers**		**Multiplexers**		**BRAM**		**DSPs**	
TSN VLAN Core (incl. TX Redundancy)	4220	5.37%	4624	2.94%	55	0.093%	3.5	1.32%	8	2%
Dropper Module (RX Redundancy) with timeout counters	854	1.09%	1697	1.08%	6	0.010%	0	0 %	34	8.50%
Time-Aware Traffic Shaper (TAS) with preemption	1813	2.31%	3089	1.97%	99	0.168%	20	7.55%	0	0%
Time-Aware Traffic Shaper (TAS) w/o preemption	1458	1.85%	2468	1.57%	34	0.058%	9	3.4%	0	0%
1G Ethernet MAC & Buffering & Preemption	1581	2.01%	1948	1.24%	12	0.020%	6	2.26%	0	0%
1G Xilinx Tri-Mode Ethernet MAC & Buffering & No Preemption	3304	4.20%	5594	3.56%	45	0.076%	4	1.51%	0	0%
Switching Crossbars – 4 Ethernet ports	1796	2.28%	1798	1.14%	0	0%	9	3.40%	0	0%

The resource usage of the implementation of the crossbar switches is highly dependent on the number of ports. The figures that we show in the table are illustrative for a 4-port device, and a particularized record of the consumption for each of the node architectures that we have implemented can be found in the corresponding sections of Chapters 8, 9, and 10.

7.8 HIGHLIGHTS OF THE TSN ARCHITECTURE

We have implemented a versatile TSN subsystem for our network nodes that is based on a collection of configurable FPGA IP cores. As a summary of the main features of our design, we enumerate our cores, their role in the system, and their main configuration options in the following points.

– We have supplied an implementation of the VLAN-tagging module for identifying the different traffic classes that will be handled in the TSN system (802.1Q). It uses our DSP-based Configuration Table, which can be configured to detect up to 16 different traffic classes. It is also tasked with redirecting TSN flows between the different Ethernet ports of the node.

– We have included enhancements to increase the robustness of the system with the use of seamless redundancy (802.1CB). These features are supported through improvements to the VLAN core and with an additional *Dropper* module for discarding duplicates. The use of redundancy can be optionally enabled in our design.

– The deterministic data forwarding is taken care of by the time-aware traffic shaper (802.1Qbv), which was jointly developed with our collaborators. The TAS shaper can be customized with a choice of forwarding queues and length of the GCL.

– The TAS shaper has been enhanced with frame preemption (802.1Qbu). This also called for the development of a preemptable Ethernet MAC (802.2br) that could interface with the TAS shaper. The use of preemption is optional in our implementations and was provided by our partners from the IAA and Seven Solutions.

– Lastly, our TSN nodes have bridging capabilities that are controlled by the corresponding VLAN modules. Our current design uses the Xilinx crossbar switches with an approach that uses small cascaded crossbars to reduce resource consumption. They may be replaced with a shared memory switch in future versions of our architecture.

Part III

Experimental Use Cases for TSN and Timing Boards

8 APPLICATION OF TSN FOR THE SMART GRID

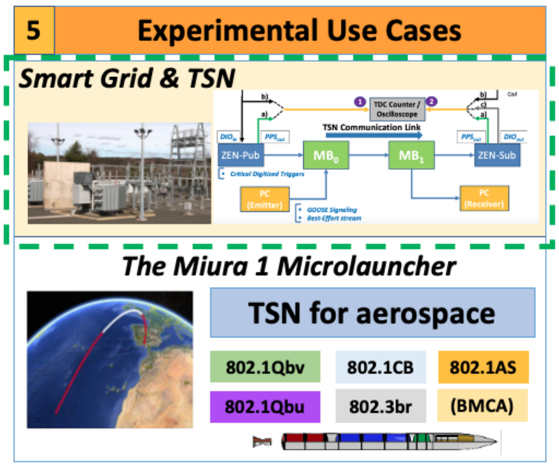


Figure 8.1
Overview of the contents of Chapter 8, where we present our use case for the Smart Grid.

This chapter presents one of the main use cases of our TSN system. Hence, we show how it can be successfully applied to support the deployment of the Smart Grid and its features by providing a unified communication interface for all the data flows in an electrical substation. This is a clear example of an application for one of the typical industrial use cases that TSN networks are intended for. We show the feasibility of this application by building a proof-of-concept system that we deployed in an actual substation facility, where we verified that our TSN network was capable of handling critical data, GOOSE substation signaling, and best-effort video simultaneously while also prioritizing the delivery of critical data with greater efficiency than the legacy electromechanical systems of the substation. After that, we performed a thorough laboratory characterization of the system to study the accuracy of its gPTP synchronization, and to learn about the effects that the specification of different types of GCL policies may exert on the attainable determinism and PDV values of the critical flows in the network. As a result of these experiments, we concluded that the end-to-end latency of a TSN flow was a parameter that the user could control precisely through the specification of different cycle times and slot structures in a given GCL.

We performed this study in the framework of an industrial transfer project with our industrial partner Seven Solutions, where we had the opportunity of collaborating with a local electricity provider from Granada (Grupo Cuerva S.L. [167]) in the development of a prototype TSN-capable switch for the Smart Grid. It was a pioneering project that showed one of the most promising applications for TSN systems in industrial scenarios. Consequently, it caused a substantial interest in the community, as evidenced by several appearances in the local media such as in [168]. Moreover, the results of the study have had a significant scientific and engineering value, which led to the production of a research article [169] that summed them up as a journal contribution. Thus, the rest of the chapter presents a transcription of the journal article, whose reference is also provided below, to illustrate the development and application of our TSN system to streamline the deployment of the Smart Grid.

J. Sanchez-Garrido, A. Jurado, L. Medina, R. Rodriguez, E. Ros and J. Diaz, "Digital Electrical Substation Communications Based on Deterministic Time-Sensitive Networking Over Ethernet," in IEEE Access, vol. 8, pp. 93621-93634, 2020, doi: 10.1109/ACCESS.2020.2995189.

Chapter contents

8.1 ABSTRACT

This work presents a novel use case with Time-Sensitive Networks (TSN) for implementing a deterministic system allowing the joint transmission of all substation communications over the same Ethernet-based infrastructure. This approach streamlines the transition to Smart Grid by simplifying the typically complex architecture of electrical substations, characterized by multiple field buses and bridging devices. Thus, Smart Grid represents a disruptive innovation advancing substations to an "all-digital" environment with a uniform interface to access, manage, and update their communications and variables. TSN can serve as its underlying foundation as it is based on open, interoperable standards and enhancements for Ethernet that can establish deterministic communications with bounded end-to-end latency. This is shown with a TSN Proof of Concept (PoC) in a real-life substation that can integrate its most usual signals: digitized analog triggers for critical events or interlocks, GOOSE signaling (IEC 61850), and Best-Effort "Internet-like" traffic. This TSN PoC is shown to be versatile enough to propagate digitized critical events around 160 μs earlier than legacy substation equipment while preserving the integrity of background traffic. Furthermore, its flexibility was characterized in-depth in controlled laboratory tests, thereby confirming TSN as a viable alternative for supporting Smart Grid so long as the appropriate configuration is supplied.

8.2 INTRODUCTION. SMART GRID AND THE MOTIVATION FOR THE APPLICATION OF TSN NETWORKS

This paper describes the application of a Time-Sensitive Networking (TSN) system to support the deployment of Smart Grid features in electrical substation environments. Time-Sensitive Networking is conceived as a number of enhancements for regular Ethernet networks [30] allowing the coexistence of data flows with different levels of criticality, bounded end-to-end latency, guaranteed bandwidth, and ensured determinism for critical traffic. Hence, these networks, which are based on interoperable open standards, implement so-called convergent communication systems that allow the aggregation of different flows according to user-defined criteria.

The concept of Smart Grid emerges from the application of new techniques and processes to legacy power grids replacing their traditional hierarchy with a completely integrated environment to efficiently process system services, exchange process data, and forward system transactions. Hence, its implementation is an ongoing transformation on all the levels of the power grid (from the generation to the distribution stages) that is being driven by the need to provide an efficient communication layer for accommodating the management of new energy sources. This results in power supply infrastructure with improved fault tolerance, and enhanced safety and quality [170, 171].

On the distribution level, this upgrade is aimed at digitizing the different control and communication equipment of substations handling the operation of the power distribution subsystems, including their sensors and actuators. A fundamental aspect of this transformation is the communication technology amongst the different substation processes, which will be replaced by Ethernet networks (either on optical fiber or copper-based links), instead of using vendor-specific solutions.

These Ethernet networks have the potential of allowing a streamlined exchange of data and the establishment of redundant communication paths for critical flows but cannot enforce any delivery guarantees or ensure end-to-end determinism. Thus, in a substation scenario, this traffic is usually handled with a three-tier architecture supported by dedicated field buses. These tiers are the *Field Level*, which implements lower-level interfaces with sensors and actuators; the *Bay Level*, which includes the equipment controlling the operation of the substation; and the *Substation Level*, which defines communication interfaces with other elements from the power grid.

Thus, the Field-level processes usually have to handle time-critical traffic, such as event alarms, trigger signals, or the interlocking mechanisms of closed control loops. This type of traffic is usually propagated using the IEC 61850 standard [172] protocol, which carries these messages from the Merging Units (*MUs*) of the substation to the Intelligent Electronic Devices (*IEDs*), where the corresponding monitoring, control, protection, and diagnostics tasks of the substation are implemented.

For its application in Smart Grid, the IEC 61850 protocol needs to be complemented with additional features; such as node discovery, reconfiguration, aggregation of different priority flows, or guaranteed determinism as bounded end-to-end latency. Additionally, its messages have to be transmitted using Ethernet-based networks. There are several Ethernet-based approaches that can be applied for Smart Grid, such as PRP (*Parallel*

Redundancy Protocol) or White Rabbit (WR) HSR (*High-Availability Seamless Redundancy*) [173]. The former allows the creation of redundant network topologies, whereas the latter uses a customized implementation of the White Rabbit synchronization stack to distribute timing information and to implement redundant ring topologies with $\sim 3\ \mu$s recovery times [171]. However, none of these systems can enforce any type of message delivery guarantee. Hence, TSN represents the ideal alternative to support the migration to Smart Grid, as it can implement a deterministic communication network with support for redundant ring topologies for select flows, as mandated by IEC 61850, with zero switchover time using its 802.1CB [44] component.

Consequently, this paper shows that TSN is a viable alternative for supporting substation communication flows in accordance with the specifications of IEC 61850 [174]. The operation of the system is characterized in-depth in the following sections, but further reliability studies can still be conducted following the guidelines proposed in [175]. Therefore, after introducing the motivations for the adoption of TSN, its main functionalities and components are described in Section 8.3. Next, the architecture of the TSN network nodes used in this work is explained in Section 8.4. Then, the feasibility of applying a TSN network for integrating Smart Grid communications is shown in Section 8.5 with the deployment of an experimental Proof-of-Concept (PoC) setup in a real-life substation. The results of the initial PoC are further characterized with a laboratory test bench in Section 8.6, where the effects of applying different configuration parameters can be studied. Lastly, Section 8.7 concludes by showing the viability of applying TSN for Smart Grid. The effects of user-driven configuration on achievable determinism are also outlined, and future lines of work are presented, such as the development of a specific TSN profile for Smart Grid.

8.3 TIME-SENSITIVE NETWORKING (TSN)

This section is intended to give the reader a brief overview of the concept of Time-Sensitive Networking (TSN) and its application on Ethernet-based networks. Readers already familiar with the concept and operation of TSN technologies may directly proceed to Section 8.4.

TSN emerges as a set of enhancements for Ethernet networks set forth by the IEEE Standardization Committees. Traditionally, Ethernet has been considered a robust technology that can fulfill the communication needs of general-purpose applications and, as a result, its use became pervasive alongside the widespread deployment of the Internet. It is thus a well-known standard that has become the foundation of the so-called *"open world"* communications. These are the IP-based flows that make up the bulk of most of today's popular Internet applications; such as HTTP web browsing or audio/video streaming. Typical Internet traffic is Best-Effort (BE) in nature and is usually supported with Ethernet networks, which share the same service philosophy.

The scenarios where timing and determinism guarantees are required usually rely on specialized, vendor-specific solutions. This is the case of industrial plants, avionics systems, or sensor-actuator control loops in general. These implement *"closed world"* communications and typically make use of field buses, such as ModBus or CAN, to guarantee that their timing and real-time constraints can be met. In heterogeneous environments, their use can lead to complex architectures with different bus domains, hence making their integration and maintenance highly costly. Furthermore, some scenarios may even require a separate data network, which adds up to system complexity.

TSN allows the combination of the aforementioned Internet-like, "open world" communications, with the "closed world" traffic associated with industrial monitoring and control using the same standard-based Ethernet network. An initial approach to this was the development of AVB (Audio/Video Bridging) [176], which was geared towards professional multimedia environments. Later on, AVB was superseded and targeted to broader applications including industrial automation, hence giving rise to Time-Sensitive Networking. Its components are outlined next.

- **System-wide Synchronization** provides a common time reference for the network. This allows the nodes of a distributed TSN system to work synchronously and forward messages consistently with the time of the network. This reference is maintained by the gPTP service (802.1AS) [6], which implements a PTP profile tailored for TSN. In the context of Smart Grid, gPTP timing can provide synchronization on the order of tens of nanoseconds (experiment 8.6.3.1) for the substation equipment, far exceeding the ~ 1 μs specification of IEC 61850 Part 5 [177]. As this reference will be distributed over Ethernet links, it is a safer alternative than the GPS-based systems that synchronize Phasor Measurement Units in some substations, adding protection against accidental or intentional GPS malfunction [178].
- **Bounded End-to-End Latency**. TSN networks define different queueing and forwarding mechanisms for guaranteeing end-to-end latency. The Time-Aware Traffic Shaper (TAS) [802.1Qbv] [9] stands out, as it forwards traffic according to a Gate Control List (*GCL*) schedule supplied by the user. This is complemented with the enhancements for preemption of lower priority messages in favor of critical, express traffic (802.1Qbu & 802.3br) [40, 41]. Their action allows TSN systems to guarantee system-level determinism for critical messages and the coexistence with Best-Effort data in the same Ethernet network.
- **System Configuration and Traffic Identification**. TSN handles traffic according to its *Traffic Class*. Thus, it implements several mechanisms for specifying the network topology, the criteria for assigning messages to a given traffic class, or the GCL Schedule of the traffic shapers. This is accomplished with resource reservation protocols (802.1Qcc, 802.1BA) [36, 47], which disseminate these parameters throughout the system. The traffic classes identified with these parameters will be denoted with a VLAN-tagged TSN stream (802.1Q) [46], whose associated priority will indicate the forwarding queue of the TAS module that it will be assigned to.
- **Reliability. Seamless Redundancy**. TSN networks can make use of the 802.1CB [44] component for protecting highly critical messages by allowing their transmission over disjoint physical paths in the network using a standard-defined frame replication scheme. This feature is often required in scenarios where the delivery of time-critical messages has to be guaranteed. This is the case of Smart Grid, where the use of redundancy mechanisms is expected in IEC 61850.

This work shows the application of a TSN approach for Smart Grid using two purpose-built nodes, whose architecture is described in Section 8.4.

8.4 IMPLEMENTATION OF A TSN-CAPABLE NETWORK NODE

The experiments presented in this work make use of two different TSN nodes: the WR-ZEN board and the MAIN TSN Switch. They are both based on the Zynq-7000 devices from Xilinx [7]. These devices are *programmable Systems-on-Chip (SoCs)*, featuring a dual-core ARM Processing System (PS) for running an embedded OS (e.g., Linux, RTEMS [152]), and FPGA Programmable Logic (PL) for implementing HDL coprocessors that can interface with the PS. These two TSN nodes share the same underlying architecture but use different SoC devices depending on their role in the system.

8.4.1 *The TSN Network Nodes*

- **The WR-ZEN Node** implements a two-port network device that can operate both as a TSN *Listener* (Receiver) or a TSN *Talker* (Transmitter) of highly critical messages. Hence, it implements a reduced TSN system in the relatively small, low-cost Z-7015 programmable SoC. Additionally, this node includes a dedicated HDL coprocessor (*DIO*) for digitizing analog substation triggers in order to forward them over the network as high priority TSN messages. This node repurposes the original architecture of the WR-ZEN board [8], which was originally used for distributing White Rabbit timing [74].
- **The MAIN TSN Switch**. This node implements a four-port *TSN bridge* that forwards different TSN messages amongst its ports in accordance with user-specified settings. The bridge functionality requires the use of a greater amount of FPGA resources to support a multi-port implementation. This led to a design that uses the larger Z-7030 Xilinx device, which was integrated in the purpose-built MAIN circuit board.

8.4.2 *TSN Node Architecture*

Fig. 8.2 shows the common architecture for the foregoing nodes. It features both FPGA subsystems and software-based components. The FPGA subsystems are implemented in the PL of the corresponding Zynq-7000 device and consist of several units that interact with one another. These are the Ethernet Networking Subsystem (blue), the Timing Distribution Subsystem (orange), the switching cores (green), and the TSN Subsystem (red). The *DIO coprocessor* is an exclusive component of the WR-ZEN node and is highlighted with a blue box. The software components run on the ARM-based PS and control the operation of the aforementioned subsystems. These components include the RTEMS OS, the gPTP service, configuration APIs, or other user tasks.

- **The Ethernet Networking Subsystem** provides the underlying Ethernet service for establishing a functioning TSN network. It consists of several off-the-shelf components allowing the instantiation of ordinary Ethernet ports in the PL of the Zynq-7000 device. These components include DMA controllers, a lightweight 1G Ethernet MAC ported from an open core design [148], the Xilinx PCS/PMA core,

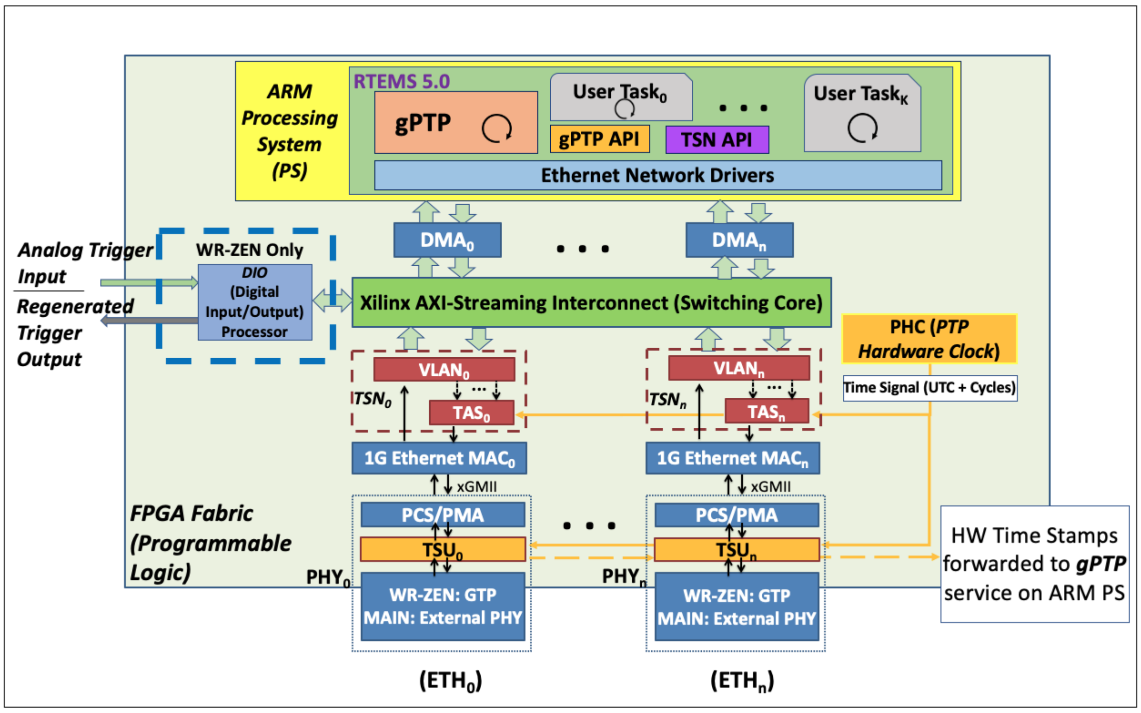


Figure 8.2
General system architecture diagram for the TSN nodes used in this work, highlighting the different subsystems within the FPGA Programmable Logic and their interactions. Image from [169].

and Ethernet transceivers (GTP blocks for the WR-ZEN or a dedicated PHY chip for the MAIN node).

- **The Timing Distribution System (gPTP)** provides the crucial timing synchronization required for any TSN system as specified in the 802.1AS subcomponent [6]. This component defines an implementation of PTP particularized for TSN, which was applied to Smart Grid using an FPGA-based design like that shown in [179]. Thus, the subsystem requires two main cores in the FPGA logic that will be coordinated by a system service of the PS implementing the gPTP protocol: The PTP Hardware Clock (PHC) and the Time-Stamping Units (TSUs). The PHC contains the internal time representation of the node, which is steered by the execution of the gPTP synchronization service. The TSUs will be used for retrieving time stamps associated with the exchange of gPTP protocol messages.
- **The TSN Subsystem** implements the essential elements allowing the establishment of deterministic communication flows. In Fig. 8.2, these elements, which are instantiated on a per port basis, are the TSN VLAN Core and the Time-Aware Traffic Shaper (*TAS*). The VLAN Core operates as an input and forwarding stage: it identifies different types of traffic according to user-defined criteria, which are then encapsulated into VLAN-tagged TSN streams and delivered to the appropriate port and queue over the AXI Switching Core. The Time-Aware Traffic Shaper works synchronously with the time reference supplied from the PHC. It forwards messages deterministically by periodically activating its priority queues according to a user-defined Gate Control List (GCL) Schedule.
- **The Software Environment of the TSN nodes**. The TSN nodes make use of Zynq-7000 devices with a dual-core Processing System that supports the execution of

the embedded real-time RTEMS 5.0 OS [152] environment. This OS provides the framework to support the various software components, modules and services to handle the operation of the TSN system: a gPTP synchronization service, Ethernet network drivers, and configuration APIs. The synchronization service updates the time representation of the PHC with the gPTP protocol (802.1AS [6]) and was developed as a custom RTEMS port of the OpenAvnu Project [52]. The network drivers initialize the Ethernet Subsystem and were adapted into RTEMS from the Xilinx Ethernet Network drivers [151]. They were additionally customized to allow their interaction with the Timing Subsystem. Lastly, the system uses two different APIs. The gPTP API is used to pass configuration parameters to the gPTP synchronization service (e.g., oscillator quality). The TSN API is used to indicate the traffic classes and applicable GCL schedules to the TSN Subsystem.

8.4.3 *Considerations on FPGA Footprint*

These experiments make use of two different flavors of TSN nodes: The WR-ZEN node and the MAIN TSN node. These are based on Zynq-7000 devices and have different resource requirements depending on their role in the network. Thus, while the former implements an End-System with just two Ethernet ports, the latter is a four-port bridge with advanced switching capabilities. The system architecture of Fig. 8.2 shows that some cores are instantiated on a per port basis, and that others have FPGA slice usage dependent on their number of bus interfaces, as is the case of the AXI Switching Core. Hence, resource consumption is bound to be dependent on the number of ports instantiated in a particular node. Consequently, the WR-ZEN node will require relatively reduced FPGA logic and can be fitted in the small, low-cost Z-7015 device (70% overall usage), whereas the MAIN node will require greater resources and will have to use the larger Z-7030 device (60% overall usage) to support its four-port design. These figures were achieved with a design that prioritized moderate FPGA footprint, which led to some compromises in the implementation, like the use of relatively modest 4 kB buffers for each queue in the traffic shapers to reduce the utilization of Block RAM primitives.

8.5 EXPERIMENTAL VALIDATION. ELECTRICAL SUBSTATION FIELD TESTS

The main premise of this work is to show that a TSN system can manage to successfully integrate all the data flows of electrical substations, especially the critical ones, over Ethernet networks extending the capabilities of the communication buses currently described in the IEC 61850 standard. Specifically, the proposed TSN system can implement the underlying communication layer of a Smart-Grid-enabled substation and, as a key difference from existing approaches, it allows for the joint transmission over shared physical links of internal control signaling [180], monitoring messages, and digitized analog triggers associated with critical events, which typically required the use of dedicated analog interfaces. Consequently, a Proof-of-Concept system demonstrating the feasibility of the application of TSN for Smart Grid was deployed in the real-life substation facility of the local electricity provider Grupo Cuerva S.L. [167] in Granada (Spain). This facility features a typical substation environment for performing a number of field tests for the TSN PoC, and its components can be examined in Fig. 8.3.

8.5.1 *The Substation Environment*

The substation architecture consists of a central control element, the CPX Substation Central Unit from ZIV Automation [181], that interfaces with the rest of the equipment in the facility. The CPX Substation Central Unit operates on the Substation Level and is therefore tasked with implementing a number of supervisory and control processes over the rest of the equipment operating on the lower Bay Level. Units such as the Transformer Control Unit (ZIV RTN) or the Line Protection Unit (7IRD) stand out amongst the elements in this latter level. The supervisory tasks over these units are supported using dedicated TX/RX optical fiber link pairs. These links do not implement a proper Ethernet network, but are rather used to efficiently implement an electrically isolated communication channel. Additionally, the entire substation can be remotely managed using a dedicated RF link that interfaces with the CPX unit.

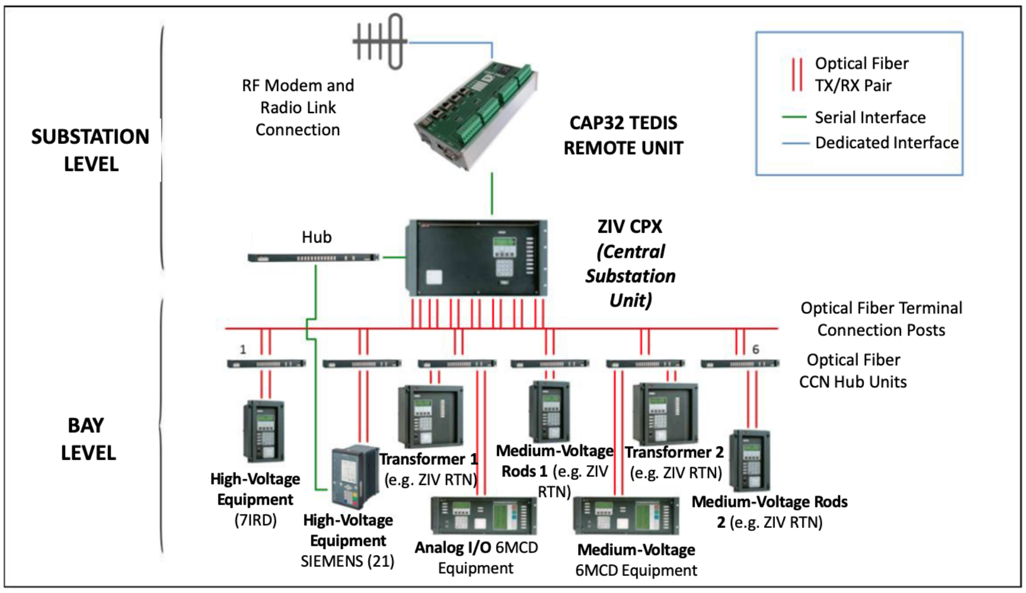

SUBSTATION LEVEL
RF Modem and Radio Link Connection
CAP32 TEDIS REMOTE UNIT
Optical Fiber TX/RX Pair
Serial Interface
Dedicated Interface
Hub
ZIV CPX (Central Substation Unit)
Optical Fiber Terminal Connection Posts
Optical Fiber CCN Hub Units
1
6
BAY LEVEL
High-Voltage Equipment (7IRD)
High-Voltage Equipment SIEMENS (21)
Transformer 1 (e.g. ZIV RTN)
Analog I/O 6MCD Equipment
Medium-Voltage Rods 1 (e.g. ZIV RTN)
Transformer 2 (e.g. ZIV RTN)
Medium-Voltage 6MCD Equipment
Medium-Voltage Rods 2 (e.g. ZIV RTN)

Figure 8.3
Simplified system diagram of components and architecture in an electrical substation from Grupo Cuerva S.L., where the field tests were conducted. Image from [169].

8.5.2 *The Substation Field Tests*

The field tests conducted in this work consisted of the deployment of a TSN system operating on the Bay level of the substation facility in order to evaluate the performance of the application of a TSN network for the transmission of highly critical data, GOOSE control signaling, and Best-Effort messages. To this end, the experimental TSN demonstrator interconnected the 7IRD Line Protection Unit with other substation equipment, as shown in the diagram of the PoC system of Fig. 8.4 or the picture of the actual deployment in the substation facility of Fig. 8.5.

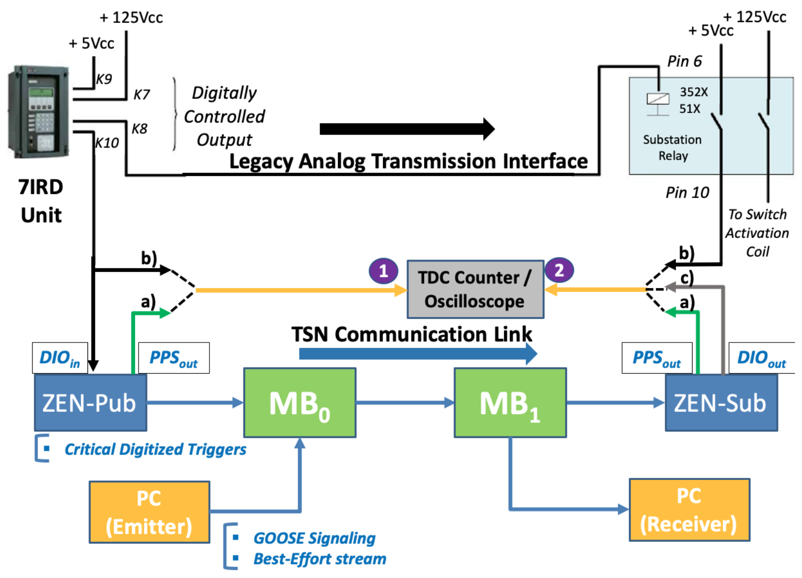


Figure 8.4
Diagram of the experimental TSN network deployed in the substation, highlighting its application for transmitting digitized signals originating from the 7IRD unit as high-priority TSN messages. Image from [169].

These tests made use of two main switching devices (the MAIN TSN nodes), and two end-systems (the WR-ZEN nodes) for establishing the demonstrator TSN network. In this setup, one of the WR-ZEN nodes assumes a TSN Talker role (ZEN-Pub) and is tasked with digitizing the analog triggers produced at the 7IRD unit and forwarding them as critical TSN messages. These messages will be received at the other Listener end-system (ZEN-Sub), which regenerates the analog trigger on reception of these messages. The digitalization and analog conversion of the critical messages is supported by the DIO feature of the WR-ZEN nodes. The network also uses two laboratory PCs for producing additional background traffic in the TSN system. These flows will be injected from the Emitter PC and forwarded to Receiver PC over the two MAIN nodes (MB0 and MB1). These flows include GOOSE substation signaling, which is generated with a custom application developed at CIRCE Foundation [182], and Best-Effort traffic, which is emitted with a general-purpose traffic generator [183].

8.5.3 *Communication Flows defined for the Substation Field Tests*

Consequently, the following communication flows were established during the experiments:

- **A high priority flow (*VLAN Priority: 3/2*).** The messages resulting from the digitalization of the analog triggers originating from the 7IRD node which are forwarded from the ZEN-Pub to ZEN-Sub nodes.
- **A medium priority GOOSE signaling stream (*VLAN Priority: 1*).** The TSN stream between the two laboratory PCs simulating the presence of background substation signaling with the GOOSE protocol.
- **A Best-Effort flow (*VLAN Priority: 0*).** Bulk Ethernet messages simulating the presence of additional, Internet-like traffic, typically non-critical monitoring.

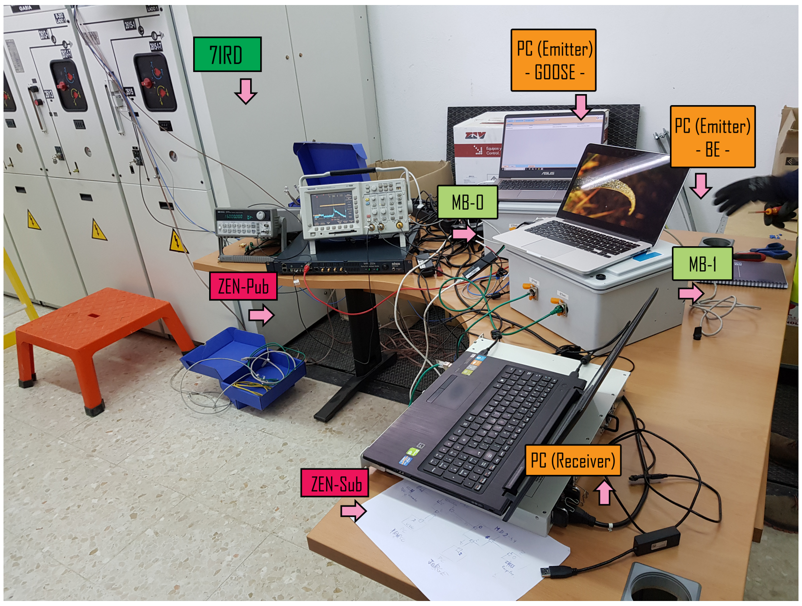


Figure 8.5
Picture of the experimental setup used in the electrical substation where the field tests were conducted, highlighting the actual test equipment, measuring instruments, and the connection to the 7IRD Line Protection Unit. Image from [169].

8.5.4 *Configuration Parameters for the substation field tests*

The substation field tests have the objective of comparing the performance of the legacy transmission mechanism of critical events in substation facilities to what is achievable with a TSN Proof-of-Concept system. To this end, two different experiments, whose settings are outlined in Table 8.1, were devised: *Performance of the Legacy Analog System*, for studying the former, and the *TSN Proof of Concept*, for the latter.

8.5.5 *Experimental Validation at the Substation Facility*

This section presents the field tests carried out at the substation facility to demonstrate the feasibility of the application of TSN networks for supporting Smart Grid deployments. These tests translate into the implementation of two major experiments with the goals of characterizing the legacy systems of the substation and showing the potential application of a TSN network in this environment, respectively.

An overview of these experiments is contained in Table 8.2, where each experiment is introduced, its goals are presented, its experimental setup and measurements are briefly described, and its corresponding outcome is outlined. The results of each experiment are also discussed in greater detail in the points 8.5.5.1 and 8.5.5.2.

8.5.5.1 *Performance of the Legacy Analog System*

This experiment was meant to establish the reference, baseline performance of the legacy substation equipment that is used for propagating high priority signals and events. In the substation facility, this mechanism consisted of a dedicated analog link that transmitted

Configuration settings applied for the substation field tests		
Configuration Parameters	**Performance of the Legacy Analog System**	**TSN Proof of Concept**
Traffic Classes & Priority	N/A (Analog triggers for signaling critical events)	– gPTP: PCP 3 – Critical: PCP 3 – GOOSE: PCP 1 – Best-Effort: PCP 0
Scheduling Policy	N/A	Configuration Set I (Table 8.5)
Network Routing	Analog link from the 7IRD Unit to the Substation Relay	Default Routing in Table 8.4
Timing Distribution	N/A	ZEN-Pub Node operating as Grand Master for all Nodes in the TSN System

Table 8.1
Configuration parameters applied for the characterization of the analog system and the TSN Proof of Concept. The PCP (Priority Code Point) applied for the VLAN tag of each traffic class also denotes their corresponding *TAS* queue. Table from [169].

critical events in the form of simple analog triggers. These triggers, which originate from the 7IRD Unit, are then delivered to a substation relay that interfaces with the appropriate controller.

Thus, the experiment characterized their propagation time along this analog circuit. As a result, it was found that the transmission latency was around $\sim 209\ \mu$s, as can be observed in Fig. 8.6a (Left). It can be seen that the dedicated analog link is a reliable means for transporting critical signals; however, this comes at the expense of forfeiting transmission speed, as the circuit incurs additional latency by activating an electromechanical substation relay, and larger deployment costs, as these links have to be set up separately for each type of event handled at the substation.

8.5.5.2 *Feasibility of the TSN Proof of Concept*

This experiment was used to demonstrate the feasibility of applying a TSN network to support the transmission of critical process data over shared Ethernet interfaces while other substation traffic is also present in the background, such as GOOSE or Best-Effort flows. This was shown with a Proof-of-Concept setup that measured the propagation time of critical events that were digitized and forwarded over the TSN system. It was found that the same critical events originating from the 7IRD Unit could now be delivered within 30 μs, that is, around $\sim 169\ \mu$s faster delivery than the legacy analog system. This can be seen in Figs. 8.6b and 8.6c (Center and Right). This delivery time corresponds to the propagation time along the three hops of the network.

These results are a significant Proof of Concept of the application of TSN for Smart Grid-capable substations. It is also proof of its scalability and versatility, as multiple flows with different levels of criticality and priority can now be deployed sharing the same physical TSN link, thereby removing the need for costly dedicated analog interfaces

Experiments performed for the substation field tests				
Configuration Parameters	**Objective**	**Laboratory Setup**	**Characterization Measurements**	**Results**
Performance of the Legacy Analog System	Assess the performance of the analog-based transmission mechanism at the substation	Injection of single analog trigger from the 7IRD Unit.	Measure time difference between rising edges of analog triggers at points 1b) & 2b) (Fig. 8.4).	– 209 μs TX latency using the legacy analog system. – Fig. 8.6a (Left). – Discussion in 8.5.5.1.
TSN Proof of Concept	– Measure the performance of the TSN System. – Compare against that of the legacy, analog system.	– Inject single analog trigger from the 7IRD Unit into the DIO input of the ZEN-Pub node. – Generate GOOSE from Emitter PC (~0.6 Mbps). – Produce Best-Effort video from the Emitter PC (50 Mbps).	– Measure end-to-end latency for the critical TSN messages as the time difference between 1b) & 2c) (Fig. 8.4). – Characterize propagation time over the TSN System.	– 30 μs propagation time over the TSN system for critical messages. – Faster than the legacy system with the substation relay (Figs. 8.6b [Center] and 8.6c [Right]). – Successful integration of Best-Effort, GOOSE, and critical messages (digitized triggers) over TSN. – Discussion in 8.5.5.2.

Table 8.2
Overview of the experiments at the substation facility characterizing the Legacy System and the TSN PoC. Table from [169].

with complex electromechanical components that were often required for propagating critical signals.

The level of performance of this TSN flow aggregation mechanism is largely determined by the configuration parameters supplied by the user, which can enormously impact variables such as the packet loss ratio or the end-to-end latency. The influence of these parameters could not be studied at the substation where the TSN PoC demonstrator was deployed, as the electromechanical elements of the 7IRD Unit and the substation relay can only be safely activated a limited number of times before causing excessive wear. Hence, this characterization was performed thoroughly in a controlled laboratory environment, where the TSN demonstrator was replicated and larger data sets could be compiled.

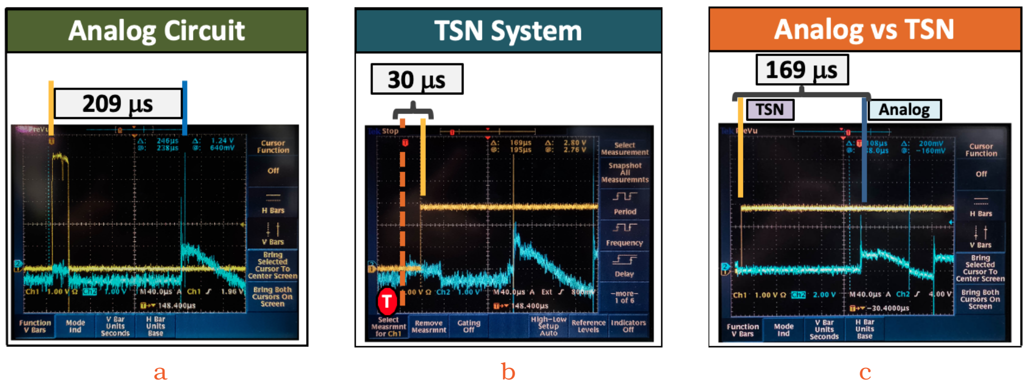


Figure 8.6
Results obtained in the tests carried out at the substation; *Figs. 8.6a (left), 8.6b (center), and 8.6c (right)*. Signal probing points referred to Fig. 8.4. **Left (Fig. 8.6a)**: Propagation latency through the analog circuit of triggers generated at the 7IRD: $\sim$ 209 μs. Yellow trace: injection at (*1-b)*). Blue trace: output at substation relay (*2-b)*). **Center (Fig. 8.6b)**: Propagation delay of digitized critical trigger through the TSN system: $\sim$ 30 μs. (T) indicates the oscilloscope trigger at *1-b)*, the Yellow trace contains the regenerated trigger at *2-c)*, and the Blue trace shows the output of the analog system at *2-b)*. **Right (Fig. 8.6c)**: The TSN network propagates event messages around 169 μs sooner than the legacy equipment. Yellow trace: regenerated trigger at *2-c)*. Blue trace: output of the analog system at *2-b)*. Image from [169].

8.6 SYSTEM CHARACTERIZATION. LABORATORY EXPERIMENTS AND RESULTS

As introduced at the end of Section 8.5, different settings can vastly affect the outcome of the data processed and propagated in a TSN system. In light of this, an in-depth characterization of the system was carried out in a controlled laboratory setup that replicated the original environment of the substation. This was accomplished by characterizing the different elements of the system, namely the performance of the timing distribution mechanism or the end-to-end latency and packet-loss ratio attainable under different settings. As a result, it is expected that these experiments will provide some valuable insight into the production of meaningful configuration designs for TSN systems at Smart Grid substations.

8.6.1 *Elements of the Experimental Laboratory Test Bench*

The laboratory test bench presented in this section is intended to replicate that of the electrical substation field tests so that the system can be characterized thoroughly. Thus, the setup used for conducting the laboratory experiments will be similar to that presented in Section 8.5, but will introduce a few modifications to suit the new laboratory testing environment. These are introduced below.

- **The TSN network nodes**. As before, the TSN network will be formed by two different types of devices: The *End-Systems*, which use the **WR-ZEN nodes**, and the *main switching nodes*, which are implemented with the two **TSN MAIN nodes**.
- **The substation event simulator**. As opposed to the field tests, which used the 7IRD unit for producing critical event analog triggers, the laboratory characterization made use of an off-the-shelf signal generator instead. This signal generator can

produce analog triggers at varying rates, which will be used for producing a larger number of events for the system characterization than could otherwise be generated at the substation environment.

- **The measurement instrumentation**. The experiments characterize the performance of the system in terms of its timing distribution accuracy, propagation delay of critical messages, packet loss ratio, and attainable bandwidth. These two latter parameters will be measured using a regular network sniffing tool [92], while the first two ones will be measured with a TDC Counter instrument [88]. The TDC Counter will replace the oscilloscope given its efficiency to perform multiple back-to-back measurements of end-to-end latency values, which are stored as large data sets that can be used to easily derive statistical indicators.
- **The substation traffic simulators**. The laboratory validation uses the same two laboratory PCs for emitting and receiving both Best-Effort traffic and GOOSE signaling. Traffic integrity statistics will be derived at the Receiver PC.

This experimental setup is only concerned with characterizing the performance of the TSN system and hence the analog transmission circuit found at the substation is not replicated for these experiments. A picture showing the laboratory setup that was built for conducting the experiments presented in this section is included in Fig. 8.7.

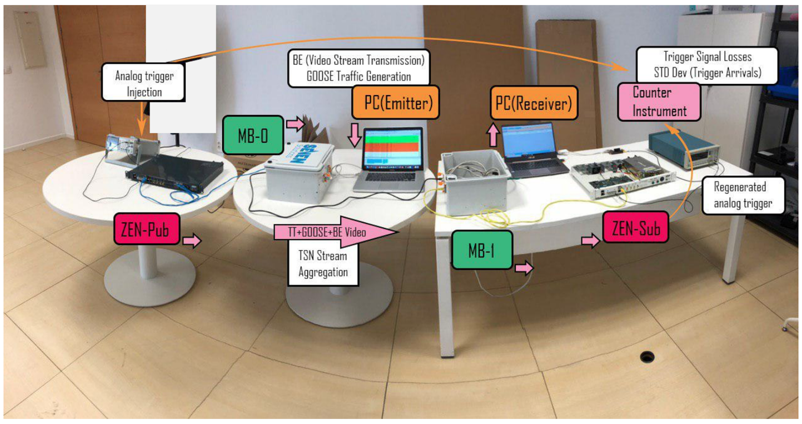

Analog trigger Injection
BE (Video Stream Transmission) GOOSE Traffic Generation
PC(Emitter)
PC(Receiver)
Trigger Signal Losses STD Dev (Trigger Arrivals)
Counter Instrument
MB-0
Regenerated analog trigger
ZEN-Pub
TT+GOOSE+BE Video
TSN Stream Aggregation
MB-1
ZEN-Sub

Figure 8.7
Picture of the laboratory setup replicating the substation field test environment. In the image, the network nodes, traffic flows, and connections used for conducting the laboratory validation tests are highlighted. Image from [169].

8.6.2 *Configuration Parameters for the Experimental Characterization*

The laboratory characterization has the goal of determining the effects that the application of different configuration parameters can pose on the achievable determinism of critical messages as well as the integrity of the lower priority flows traversing the network. To this end, five different experiments have been conducted to further analyze the operation of the elements of the TSN system, such as the traffic aggregation mechanism or its timing distribution component, by replicating the environment of the substation field tests in a controlled laboratory setup.

Each experiment will thus make use of different configuration sets aimed at producing diverse effects in specific aspects of the system; namely on its ability to guarantee timely deliveries of critical messages or the integrity of the lower priority messages. In general, TSN systems need to be supplied with two main sets of parameters for their operation: a set of Traffic Classes for associating different types of traffic with VLAN-tagged TSN streams and a traffic shaping policy (GCL). Traffic classes can be identified in the presented solution by providing specific Ethernet header fields, like the *"Destination MAC Address"*, and an associated priority value (*PCP code*). In order to minimize resource usage, this implementation was customized to handle priorities ranging from *0* (Best-Effort) to *3* (critical). The traffic shaping policy will be defined by a main scheduling cycle time (T_{cyc}) divided into constituting intervals (I_n) and will have to be applied on the shapers of the egress ports of the nodes in the network. The combined action of the traffic classes and the scheduling policies are the main drivers of the achievable determinism, but the user also needs to provide routing information for each traffic class and configure at least one of the nodes in the system to operate as a gPTP synchronization Master. Table 8.3 contains a summary of the configuration parameters that were tapped for each subsystem of the TSN network in order to perform each characterization experiment.

Configuration settings applied for the laboratory validation tests

System Component	Experiment I (Timing)	Experiment II (Baseline)	Experiment III (Highest Attainable Rate)	Experiment IV (Moderate Use)	Experiment V (Worst Case)
Traffic Classes & Priority	gPTP: PCP 3 Critical: PCP 3 GOOSE: PCP 1 Best-Effort: PCP 0			gPTP: PCP 3 Critical: PCP 2 GOOSE: PCP 1 Best-Effort: PCP 0	
Scheduling Policy	All Queues Open All the Time	Configuration Set I		Configuration Set II	Configuration Set III
Network Routing	No TSN flows routed	Default Routing in Table 8.4			
Timing Distribution	ZEN-Pub Node operating as Grand Master for all Nodes in the TSN System				

Table 8.3
Configuration parameters applied for performing each experiment in the laboratory validation tests. The PCP (*priority*) values indicated for the VLAN tag of each type of message also denote their corresponding *TAS* queue. Table from [169].

In particular, the Routing settings can be examined in Table 8.4, which indicates that the critical traffic is exchanged between the ZEN nodes, whereas the GOOSE and Best-Effort flows are exchanged between the two laboratory PCs.

Furthermore, the scheduling policies applied throughout the experiments can be seen in the Tables for the Configuration Sets I (Table 8.5), II (Table 8.6), and III (Table 8.7). Sets I and II share the same structure, as they define a 4 ms periodic schedule divided into three different intervals, with the main difference between the two of them being that Set II enforces a more restrictive policy that limits the transmission of critical frames to a designated slot with a segregated queue ($Q2$) from that of gPTP synchronization. Set III studies the effects on the variation of the end-to-end latency of critical messages and the

TSN Stream Routing Settings			
Start Node	Hop#0	Hop#1	End Node
ZEN-Pub.[Critical TSN Stream]	MB0	MB1	ZEN-Sub
PC Emitter.[GOOSE Signaling & Best-Effort]			PC Receiver

Table 8.4
Routing configuration used in the laboratory validation experiments. All the TSN flows are forwarded over the MAIN TSN nodes to the appropriate recipient. Table from [169].

integrity of the rest of the traffic in the network when a number of scheduling policies are iteratively applied to the system.

Configuration Set I			
Interval No.	Duration (ms)	Queue Settings [Q0|Q1|Q2|Q3]	Description
I0	2	1001	BE & Critical & gPTP
I1	1	0101	GOOSE & Critical & gPTP
I2	1	0001	Critical & gPTP

Table 8.5
Configuration applied to Experiments II (8.6.3.2) and III (8.6.3.3). The table defines a 4 ms periodic cycle divided into three different intervals where the critical flow and the gPTP messages share the same queue (*Q3*). Table from [169].

Configuration Set II			
Interval No.	Duration (ms)	Queue Settings [Q0|Q1|Q2|Q3]	Description
I0	2	1001	BE & gPTP
I1	1	0101	GOOSE & gPTP
I2	1	0011	Critical & gPTP

Table 8.6
Configuration applied to Experiment IV (8.6.3.4). The table defines a 4 ms periodic cycle divided into three different intervals where the critical flow gets its own separate queue (*Q2*) from the gPTP messages (*Q3*). Table from [169].

8.6.3 *Characterization Experiments*

This section covers the laboratory characterization that expands on that performed at the substation facility of the TSN system. This is achieved through a series of experiments with a twofold goal. On the one hand, they will aim to characterize the attainable performance and operation of the system (Experiments I through IV), whereas, on the other hand, they will also attempt to delimit the effects that the application of different settings can exert on the achievable determinism of critical TSN flows (Experiment V).

Configuration Set III							
Scheduling Policy		Interval 0		Interval 1		Interval 2	
Iteration No.	Total Cycle (μs)	Duration (μs)	Configuration [Q0|Q1|Q2|Q3]	Duration (μs)	Configuration [Q0|Q1|Q2|Q3]	Duration (μs)	Configuration [Q0|Q1|Q2|Q3]
0	48	24	BE & gPTP [1001]	12	GOOSE & gPTP [0101]	12	Critical & gPTP [0011]
1	96	48		24		24	
2	192	96		48		48	
3	384	192		96		96	
4	768	384		192		192	
5	1536	768		384		384	
6	3072	1536		768		768	
7	6144	3072		1536		1536	
8	12288	6144		3072		3072	

Table 8.7
Scheduling policy for the iterative sweep of different scheduling cycle times performed during Experiment V (8.6.3.5). Each row corresponds to a specific iteration, which is associated with a given cycle time that is divided into three separate intervals. Table from [169].

A general overview of these experiments is presented in Table 8.8, where each experiment is introduced, along with its goals, the experimental setup that was applied, the measurements that were performed, and the corresponding outcome that resulted from the experiment. These outcomes are subsequently discussed in greater detail in the corresponding discussion sections for each experiment (Sections 8.6.3.1 through 8.6.3.5).

8.6.3.1 *Performance of the Timing Distribution System*

The timing distribution is one of the crucial components required to ensure proper operation of a Time-Sensitive Networking system. Thus, its task is to propagate an accurate time base that will be shared amongst all the nodes that are part of the TSN network to guarantee the deterministic forwarding of critical messages, as it enables the synchronous operation of the different time-aware queues throughout the network. Overall, the achievable determinism of the TSN system is limited by the robustness of its timing distribution service, that is, the gPTP component, which is characterized in this experiment.

This characterization was carried out by means of deriving the Allan Deviation (ADEV) [96], which was calculated by recording PPS time differences between the ZEN-Pub and ZEN-Sub nodes for 65 hours. The ADEV indicator was then derived and represented using the *AllanTools* toolset [97], as shown in Fig. 8.8. The results indicate that the system remains stable in the long term and behaves linearly as its phase noise corresponds to that of a Gaussian process. Furthermore, the plot shows that the timing distribution has a degree of accuracy in the vicinity of tens of nanoseconds ($\sim$ 10 ns) for averaging times on the order of one second, which is in the same range as other commercial PTP-based solutions.

8.6.3.2 *TSN Flow Aggregation over Ethernet Links. Baseline Scenario*

This test defines the baseline experimental case as it characterizes the system using a trivial configuration similar to that used during the substation field tests. The results can

Experiments performed for the laboratory tests				
Configuration Parameters	**Objective**	**Laboratory Setup**	**Characterization Measurements**	**Results**
Experiment I (Timing)	Measure the performance of the timing distribution system.	– ZEN-Pub acting as Timing Master. – MB0, MB1, and ZEN-Sub are Timing Slaves.	-Measure PPS time difference between the ZEN-Pub and ZEN-Sub nodes (*1a) & 2a)* in Fig. 8.4).	– Calculation of the Allan Deviation (ADEV) for time synchronization stability. – ADEV Plot (Fig. 8.8). – ∼10 ns, PTP-like accuracy. – Discussion in 8.6.3.1.
Experiment II (Baseline)	Characterize the TSN Link Aggregation mechanism with trivial configuration.	– Inject 1 Hz triggers into the DIO of the ZEN-Pub. – Generate GOOSE from Emitter PC (@ ∼0.6 Mbps). – Produce Best-Effort video from the Emitter PC (4 Mbps).	– Measure end-to-end latency variation for the critical flow (*1b) & 2c)* in Fig. 8.4). – Study level of traffic integrity preservation for the GOOSE and Best-Effort flows.	– Critical Traffic Latency in Table 8.9. – Traffic Integrity in Table 8.10. – Verified end-to-end determinism of critical messages and aggregation of other flows (BE, GOOSE). – Discussion in 8.6.3.2.
Experiment III (Highest Attainable Rate)	Characterize resilience of the TSN Link Aggregation mechanism by determining the highest attainable transmission rate of critical traffic.	– Incremental sweep on trigger generation rates injected into the DIO of the ZEN-Pub. – Generate GOOSE from Emitter PC (@ ∼0.6 Mbps). – Produce Best-Effort video from the Emitter PC (4 Mbps).	– Determine trigger rate that causes critical packet losses during network transmission (*1b) & 2c)* in Fig. 8.4). – Detected as non-zero totalization condition in the Counter instrument.	– Highest achievable rate under current configuration: 670 Hz. – Subsequent experiments will generate critical messages at 100 Hz. – Discussion in 8.6.3.3.
Experiment IV (Moderate Use)	Study the effects of the application of a more restrictive scheduling policy with separate queues for critical and gPTP traffic.	– Inject 100 Hz trigger signals for producing critical traffic into the DIO input of the ZEN-Pub. – Generate GOOSE from Emitter PC (@ ∼0.6 Mbps). – Produce Best-Effort flow from the Emitter PC (50 Mbps).	-Measure impact of new scheduling on the end-to-end latency of critical traffic (*1b) & 2c)* in Fig. 8.4).	– Critical traffic Latency in Table 8.11. – End-to-end latency as a function of cycle & interval time. – Discussion in 8.6.3.4.
Experiment V (Worst Case)	In-depth characterization of the influence on determinism of different scheduling designs by applying an iterative sweep of policies with growing cycle times and interval lengths.		– Measure end-to-end latency of critical traffic for each iteration of Configuration Set III (*1b) & 2c)* in Fig. 8.4). – Measure integrity of the GOOSE and Best-Effort flows for each iteration of Configuration Set III.	– Critical traffic Latency in Table 8.12. – Traffic Integrity in Table 8.13. – Determinism for critical flows can be set by the scheduling policy. – Found effects of cycle & interval in (8.1) for this scenario. – Discussion in 8.6.3.5.

Table 8.8
Overview of the experiments in the laboratory validation environment. Table from [169].

be examined in Tables 8.9 and 8.10, which show the transmission latency of critical TSN messages and the level of integrity of the lower priority flows in the network (GOOSE and Best-Effort), respectively.

The experiment measured the operation of the system for 300 seconds and found that the applied scheduling policy yielded minimized end-to-latency for the critical messages transmitted over the network. This minimized latency oscillates between 25.8 μs, which is associated with the message propagation time, and a peak of 38.7 μs, which is accounted for by the effect of interfering gPTP traffic sharing the same queue ($Q3$) as the critical traffic, as can be seen in Table 8.9. Hence, in this experiment the main variable affecting the determinism of the critical flow is its associated priority regardless of the interval distribution in the scheduling cycle, which in this case was set to a 4 ms cycle (T_{cyc}) with a 1 ms interval for the critical traffic (I_2). As the traffic shapers implement a strict priority

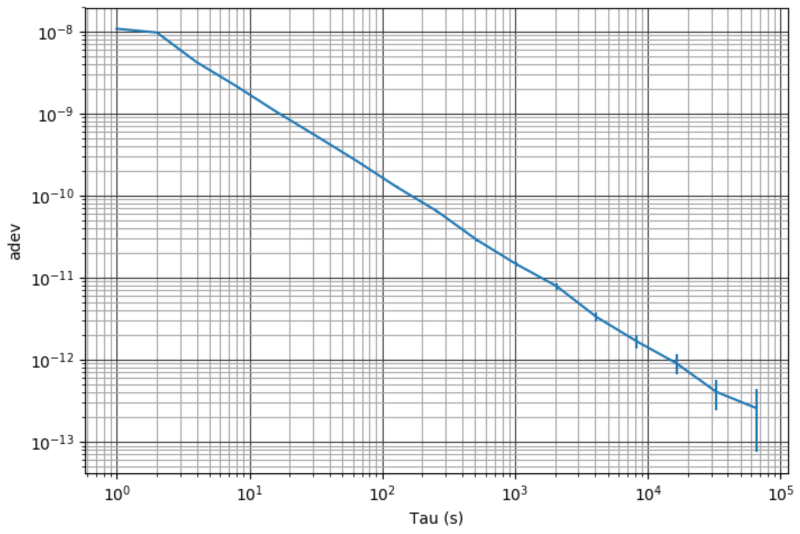


Figure 8.8
Allan Deviation plot showing the degree of accuracy and timing distribution stability calculated for the gPTP synchronization component of the TSN system after recording PPS difference samples for approximately 65 hours. Image from [169].

selection mechanism when several queues are active during the same interval, having the critical flow share the higher priority queue of gPTP (priority 3), which must always be open, would yield this minimized end-to-end latency.

Furthermore, it was shown that this particular policy allows the preservation of the integrity of the GOOSE substation signaling messages at the expense of the Best-Effort traffic, which undergoes minor degradation ($< 1\%$) under this configuration (Table 8.10).

Critical Traffic Delivery Jitter			
MAX (μs)	min (μs)	Peak-to-Peak (MAX-min) (μs)	Std.Dev. (μs)
38.70	25.80	12.938	1.173

Table 8.9
Transmission latency associated with the highly critical traffic in the baseline characterization scenario. Table from [169].

	GOOSE				Best-Effort (BE)				Results	
Cycle (μs)	NTX [pckts]	NRX [pckts]	BW(TX) [Mbps]	BW(RX) [Mbps]	NTX [pckts]	NRX [pckts]	BW(TX) [Mbps]	BW(RX) [Mbps]	GOOSE PL (%)	BE PL (%)
4000	129493	129493	0.573	0.573	698250	693252	3.9	3.8	0	0.716

Table 8.10
Summary of the traffic integrity level for the medium priority and Best-Effort flows traversing the network in experiment 8.6.3.2, as indicated in the Packet Loss (*PL*) entries. In the table *NTX* and *NRX* indicate the number of packets sent/received, and *BW(TX)* and *BW(RX)* indicate the measured bandwidth on transmission/reception. The integrity of the medium priority GOOSE signaling is preserved at the expense of the Best-Effort messages, which undergo minor degradation ($< 1\%$). Table from [169].

8.6.3.3 *TSN Flow Aggregation over Ethernet Links. Highest attainable rate for critical traffic*

The goal of the experiment was to determine the highest generation rate of critical messages under the baseline configuration that allows their transmission without incurring any data losses. This was measured by totalizing the number of critical messages forwarded from the ZEN-Pub node to the ZEN-Sub node, and then detecting packet losses as *non-zero counter totalization*.

It was determined that the highest attainable critical message transmission rate was *670 Hz* for this experiment and configuration set in particular, and that higher rates gave rise to congestion losses. Consequently, the subsequent experiments will generate critical messages at 100 Hz in order to produce larger data sets for deriving statistical indicators.

8.6.3.4 *TSN Flow Aggregation over Ethernet Links. Moderate Link Utilization with Best-Effort Traffic.*

This experiment studies the effects of the application of a more restrictive scheduling policy, whereby critical messages will now have a separate, designated queue ($Q2$) which only gets activated during a specific interval of the Configuration Set II (Table 8.6). The synchronization queue ($Q3$) is always active. Thus, it is expected that critical messages will be prone to be severely impacted by the user-designed schedule applied in the experiment. This is shown in Table 8.11, where it can be seen that the end-to-end latency variation is directly related to the length of the scheduling cycle.

Critical Traffic Delivery Jitter			
MAX (μs)	**min (μs)**	**Peak-to-Peak (MAX-min) (μs)**	**Std.Dev. (μs)**
3030	26	2999.013	900.075

Table 8.11
Transmission latency associated with the critical traffic from experiment 8.6.3.4. The test uses a segregated queue for the critical traffic ($Q2$) and a long scheduling cycle with a small service slot for $Q2$, resulting in significant end-to-end latency variation (~3 ms). Table from [169].

Specifically, this policy still allows the realization of a minimum ~26 μs latency, which corresponds to the minimum propagation and processing time through the Ethernet links. However, the maximum end-to-end latency will now be determined by the relationship between the cycle time of the scheduling policy of the Configuration Set II and the length of time that the queue for critical traffic remains idle. Hence, it should be noted that unlike the case of Experiment 8.6.3.2, the choice of a priority value other than 3 for the critical messages will result in the fact that the scheduling cycle and its internal interval distribution will now be the decisive factors for establishing the end-to-end latency of a given flow.

This is confirmed in the experimental data, where the ~3 ms latency variation measured corresponds to the length of time that the critical message queue is inactive, as the applied policy defines a 4 ms cycle (T_{cyc}) with a 1 ms slot when Q2 is active (I_2). Besides, as the critical traffic generator is not synchronized to the TSN system time, the 900 μs standard deviation of the end-to-latency of the critical messages indicates that most critical frames are forwarded within the 1 ms slot when Q2 is active. This leads to the conclusion that a poorly designed schedule could result in catastrophic system operation, causing critical data to miss delivery deadlines or even experience congestion losses.

8.6.3.5 *TSN Flow Aggregation over Ethernet Links. Worst-case End-to-End Latency for Critical Traffic and Effects on the Integrity of Lower Priority Flows.*

The tests carried out in this section set out to delimit the influence of different scheduling policies on the attainable determinism for critical messages and the integrity of the lower priority flows in the network. This was achieved by applying the different policies defined in each iteration of the Configuration Set III (Table 8.7).

The results of the end-to-end latency for the critical messages under each iteration can be examined in Table 8.12, where it can be seen that the relationship determining the maximum latency variation between the schedule cycle time and the duration of the slot for critical traffic that was pointed out in 8.6.3.4 still holds, and can be described with the expression in (8.1).

$$Lat_{max} = T_{cyc} - I_2 + t_{prop} + t_{del} \tag{8.1}$$

Critical Traffic Delivery Jitter for each Iteration					
It. No.	Cycle (μs)	MAX (μs)	min (μs)	Peak-to-Peak (MAX-min) (μs)	Std.Dev. (μs)
0	48	81.225	27.101	54.124	13.814
1	96	117.201	27.019	90.182	25.168
2	192	187.464	27.052	160.412	50.147
3	384	332.210	27.082	305.128	97.818
4	768	619.064	27.053	592.01	193.117
5	1536	1192.346	27.085	1165.261	382.413
6	3072	2340.611	27.072	2313.539	764.557
7	6144	4644.351	27.057	4617.294	1525.328
8	12288	9251.809	26.992	9224.817	3050.670

Table 8.12
Values for the end-to-end latency associated with the critical flow obtained for each iteration defined in Table 8.7. Table from [169].

This expression describes the effect that the configuration applied for the experiment has on the latency variation. Hence, it was found that the maximum end-to-end latency was determined by the duration of the scheduling cycle (T_{cyc}) and the length of the slot for critical traffic (I_2), with the additional contributing delays of the propagation time (27 μs) through the Ethernet links of the TSN system (t_{prop}), and a peak processing time of 17 μs (t_{del}). This empirical derivation of (8.1) is meant to show that the user would be able

	GOOSE				Best-Effort (BE)				Results (Packet Losses %)	
Cycle (μs)	NTX [pckts]	NRX [pckts]	BW(TX) [Mbps]	BW(RX) [Mbps]	NTX [pckts]	NRX [pckts]	BW(TX) [Mbps]	BW(RX) [Mbps]	GOOSE PL (%)	BE PL(%)
48	24371	24371	0.099	0.098	1233030	1233026	49	49	0	0.000324
96	31549	31549	0.129	0.129	1232509	1232505	49	49	0	0.000325
192	32298	32296	0.133	0.132	1206781	1206777	49	49	0.006192	0.000332
384	142503	142503	0.577	0.577	1234170	1234169	49	49	0	$8.1026 \cdot 10^{-5}$
768	142309	142309	0.577	0.577	1235835	1235835	49	49	0	0
1536	141115	141115	0.569	0.568	1228597	1228593	49	49	0	0.000326
3072	141771	141771	0.577	0.576	1241574	1013543	49	40	0	18.366283
6144	141044	141044	0.577	0.577	1229649	809817	49	32	0	34.142426
12288	139467	139467	0.576	0.575	1233620	716424	49	28	0	41.925066

Table 8.13
Summary of the traffic integrity level for the medium priority and Best-Effort flows traversing the network throughout the iterations of experiment 8.6.3.5, as indicated in the Packet Loss (*PL*) entries. In the table, *NTX* and *NRX* indicate the number of packets sent/received, and *BW(TX)* and *BW(RX)* indicate the measured bandwidth on transmission/reception. GOOSE signaling is preserved, even though some of the policies applied cause significant degradation for Best-Effort traffic ($> 20\%$). Table from [169].

to adjust the end-to-end determinism by supplying different values for T_{cyc} and I_2 for the scenario under evaluation with the current choices of priority for the critical flow and network topology. For instance, a 384 μs cycle (T_{cyc}) combined with a 96 μs interval for the critical traffic (I_2) should yield a maximum latency of 332 μs, which closely resembles the experimental data for *iteration 3* in Table 8.12 (MAX: 332.210 μs). Other scenarios with different network topologies or a greater number of flows and priorities might lead to different expressions that the user should evaluate and leverage to design configurations that can target the desired levels of determinism.

The level of integrity of the lower priority flows in the network for each iteration of the experiment was also examined in parallel, as shown in Table 8.13. These results have a twofold implication. On the one hand, the determinism of critical flows is a parameter that can be designed for to meet the requirements of a given system (e.g., delivery deadlines). On the other hand, there is a compromise with the integrity of the lower priority flows, which could experience congestion losses with growing cycle times and comparatively short service slots. This is the case of the iterations under study, where the Best-Effort can undergo significant degradation in some cases, while the GOOSE signaling remains protected given its lower bandwidth usage and higher priority.

It is important to note that the buffer depth (4 kB per queue) of the current implementation is an important factor in the integrity results. The Best-Effort traffic is generated at a constant rate of 50 Mbps using 1500 B frames and, since a general-purpose OS is used, occasional bursts may occur. As the Best-Effort queue can only hold two 1500 B frames at a time when it is idle, we have found that there are cycle/interval combinations where the occasional frame may be dropped ($\sim$4 frames on average) for cycles shorter than 1536 μs, and others, like 768 μs, that manage to avoid dropping any messages altogether. GOOSE traffic uses a higher priority queue with frames of 168 B (26 messages per queue), but it could still be affected if the traffic were to be emitted in sufficiently large bursts. This was the case of the 192 μs cycle iteration. These effects would be mitigated using larger buffers.

Lastly, it is worth noting that the application of the scheduling policies of Set III produces the maximum end-to-end latency described in the expression in (8.1) when applied to the current system. However, more complex topologies with a greater number of flows may require the derivation of more complicated policies. These policies should be able to take into account the fact that TSN messages may have different associated forwarding times as a result of internal processing delays, impact of arbitration mechanisms, or the use of redundant paths (802.1CB). This latter case is especially sensitive, as the propagation times of redundant messages over different paths can be widely different. In the literature there are several works for calculating meaningful policies under such complex scenarios, and the work in [184] provides a useful framework for assessing the synchronization error in message forwarding associated with the application of a certain policy in systems with time-varying delays.

8.7 CONCLUSIONS AND FUTURE WORK

This work has shown the feasible application of a TSN system to a substation environment to enhance the new Ethernet-based networks that are being deployed in these facilities during the transition to Smart Grid. It is expected that the use of Smart Grid will provide a unified framework for managing and handling all the different data flows of the substation. Ethernet-based networks already allow the transmission of the signaling data and non-critical messages that are usually propagated in these environments. In this context, critical event data messages, which cannot be handled by regular Ethernet networks, could benefit from the use of the proposed TSN system, which would allow their deterministic transmission alongside the other flows of the substation over the same Ethernet-based infrastructure. Hence, this would provide Smart Grid-enabled substations with a flexible networking stack allowing simultaneous propagation of ordinary IP flows, GOOSE signaling (IEC 61850), or the critical traffic typically found in the supervisory and control processes on the Bay Level equipment of the substation.

This latter point was proven by devising and deploying a Proof-of-Concept TSN system in an actual electrical substation from a local power utility [167], where several field tests managed to show that a TSN system could be successfully applied to unifying all the communications in the substation over a shared Ethernet-based bus: critical messages carrying digitized trigger data, medium priority GOOSE signaling from the IEC 61850 standard, and Best-Effort flows.

Next, this work performed an in-depth characterization of the influence of different configuration parameters on the performance of the system in a controlled laboratory environment. Thus, this stage started with the evaluation of the gPTP synchronization, which was pegged to the tens of nanoseconds. After that, the ability of the TSN system to combine background substation traffic (GOOSE, Best-Effort) with critical flows was assessed, and it was verified that a deterministic delivery for the critical messages could be enforced. In this context, the influence of the scheduling policy on the end-to-end latency of the critical flows was determined to be the result of the combined effect of the application of different cycle times and interval durations in the traffic shapers: bounded latencies between 81 μs and 9.251 ms could be achieved with cycles (T_{cyc}) and critical intervals (I_2) between 48 μs and 12 ms. Hence, it was found that the design of the traffic-shaping schedule is the chief parameter governing the attainable determinism for a TSN flow, allowing the user to target application-specific requirements. Overall, this has shown that the system is highly versatile and scalable, given its multiple user configuration options for handling different types of traffic, and flexible enough to allow the deployment of distributed applications supported with highly accurate gPTP synchronization (tens of nanoseconds accuracy).

After this characterization work, we have planned a number of future actions, like the development of a user-directed utility for the centralized configuration of the entire system or the design of larger TSN switches. Further applications of TSN for Smart Grid domains could also be considered, like the definition of a Smart Grid Profile for TSN, its application for monitoring low-voltage grids [185], or the implementation of an OPC-UA interface over TSN for managing substation equipment.

9 APPLICATION OF A TSN ETHERNET INTERFACE FOR SPACE MICROLAUNCHERS WITH A MODERATE-FOOTPRINT FPGA DESIGN

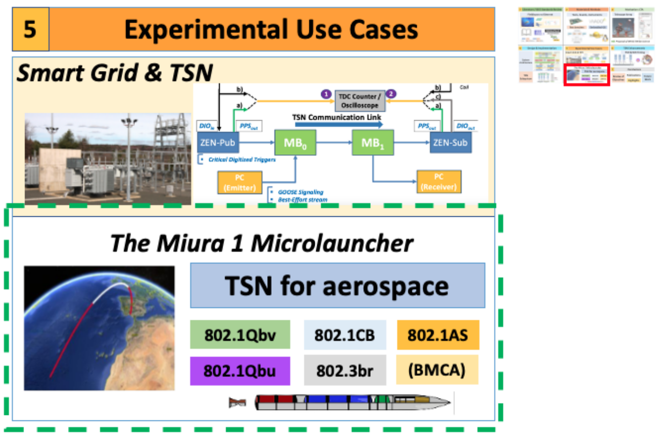


Figure 9.1
Overview of the contents of Chapter 9, where we present the application of our TSN system to the avionics of the Miura 1 microlauncher.

This chapter examines the development of a TSN system to support the avionics of a microlauncher vehicle. One of our main contributions is that we have demonstrated how COTS components can successfully be applied to the construction aerospace platforms. Specifically, we have shown this by deploying an Ethernet-based TSN system to handle the communication requirements of the Miura 1 sounding rocket. We have validated this claim with extensive system testing and by building a demonstrator of our proposed TSN-based avionics network. These tests show that our implementation features the necessary robustness and determinism to support the communication requirements of the Miura 1 sounding rocket.

This study is the result of a joint collaboration with Seven Solutions S.L. and GMV Aerospace SAU; and the results thereof presented in this chapter are a preliminary version of a submitted journal contribution that is currently under review for future publication.

Chapter contents

9.1 AEROSPACE MICROLAUNCHERS AND THE CASE FOR TSN IN SPACE.

Microlaunchers are versatile vehicles that have gained notoriety in the context of the so-called "*New Space*" endeavors [186]. These are launch vehicles that are usually meant to deploy relatively reduced payloads of up to 300 kg on most occasions to the lower Earth orbits (up to ∼700 km above the Earth's surface). Hence, these platforms are built with the goal of providing direct, convenient, and affordable access to space for different actors, such as businesses, corporations, or governments. They are an alternative to traditional vehicles like Ariane [187]. As a result, these vehicles follow a different paradigm for their design from that of traditional space missions: they need faster development cycles, they usually attempt to pursue the reusability of the vehicles themselves, and fast project development turnarounds. Hence, these projects prioritize these aspects over the ability of carrying larger payloads or the greater power of traditional space launchers.

One of the main results of this new design methodology is that manufacturers are overwhelmingly opting for COTS solutions as a means of reducing development time or for enhancing interoperability. Furthermore, the use of COTS could also bring in additional benefits, such as increased computing power or a simplified integration with components from other suppliers. Although the process for procuring and integrating COTS elements has to follow strict guidelines for aerospace projects, their use for building the high-performance computing systems of space vehicles is well-documented, as was the case of the Demeter microsatellite [188]. Another notable example is the Ariane 6 launch vehicle, which uses an adaptation for space of the COTS-based time-triggered Ethernet protocol from TTTech [71, 189].

Thus, the design of space vehicles has often relied on the traditional fieldbuses for aerospace, which are the de-facto standards for the development of space missions. Some well-known examples, as mentioned in the State of the Art (Chapter 3) are the MIL-STD-1553B [18] or the Spacewire [62] buses. The former originated in 1978 and has been widely used; although it is now mostly superseded in favor of Spacewire. Both solutions could deliver the deterministic, robust, and redundant communications that are expected in space missions, albeit at the expense of requiring non-interoperable, vendor-locked equipment.

Consequently, since the dominant trend in the design of "New Space" vehicles is to opt for interoperable, COTS-based standards and solutions, there is an interest in replacing these space fieldbuses with Ethernet-based communications. Hence, Ethernet networks can provide decentralized topologies with interoperable devices, as the standard is well-known in the engineering community and supported by a large vendor base. Nonetheless, we had not been able to use these systems for aerospace vehicles up until recently on account of their best-effort nature. However, with the advent of TSN [30], the replacement of the multiple legacy fieldbuses for aerospace with an Ethernet-based interface is technically feasible. As we mentioned in Chapter 3, this can be seen as part of a larger trend that seeks to replace fieldbuses at large with one "single, true, and open" standard. Its application to aerospace would be but another one of its profiles. In fact, given the broad areas of application of TSN technologies for aerospace, there is a growing interest in the aerospace industry in the development of a standardized specification of a

TSN profile for space, as evidenced by the ongoing standardization efforts at the TSN WG [65].

In this context, the study that we present in this chapter looks into the construction and application of a TSN system to support the avionics of the Miura 1 [190] microlauncher from GMV [191] and PLD [192]. We use this project to examine the feasibility of applying TSN to aerospace. We accomplished this by designing a suitable hardware platform for the avionics nodes of the microlauncher and, in the process, we fitted the nodes with all the elements of TSN that we posited were needed for supporting the deterministic communications of an avionics system. Hence, we could consider this design an early proposal of a TSN profile for aerospace, which we claim should support the use of 802.1AS timing, 802.1Qbv time-aware traffic shaping with frame preemption (802.1Qbu, 802.3br), and seamless redundancy (802.1CB). Furthermore, we supported this with a real-time OS (RTEMS).

9.2 THE PROPOSAL OF A TSN PROFILE FOR AEROSPACE

As stated previously, there is a growing interest in the development of a TSN profile for aerospace. Indeed, its specification is currently underway at the TSN WG [30]. We have proposed a possible profile that could be applied to space and avionics systems. Hence, given the main components – the "pillars" (Section 3.3) – of TSN, we analyze how we could deploy TSN systems for these applications. Ultimately, our goal is to ensure that our avionics implementation uses *a)* system-wide synchronization, *b)* bounded end-to-end latency, *c)* system management and traffic class identification, and *d)* robust message forwarding.

a) The **system-wide synchronization**, as expected in the specification of TSN, can be supplied through the gPTP timing described in the 802.1AS [6] standard. In addition, to enhance the system robustness, we also contemplate the use of the best master clock algorithm (BMCA). This allows the system to quickly switch over to a secondary timing source in the event of a failure in the grand master or if the network path to the time source is severed or becomes unavailable.

b) The **bounded end-to-end latency** can be supplied with the time-aware traffic shaper (TAS) defined in 802.1Qbv [9] for the cyclic forwarding of TSN *streams* during the time slots of a given GCL schedule. It is also a requirement that we should reduce the end-to-end jitter of the flows in the microlauncher. As a result, we selected the feature of frame preemption (802.1Qbu [41] & 802.3br [40]) to achieve this.

c) The **system-wide configuration and management** can be achieved by supplying the necessary configuration parameters that the system will in turn use to identify the different types of traffic (802.1Q [46]), enforce a certain topology, activate the use of redundant paths, or execute a given GCL schedule. We can achieve this with resource reservation protocols (802.1Qcc [47]). In this project, we designed a custom API to this end that could supply these parameters (see 6.3.4).

d) The **robust message forwarding** that is expected in avionics systems can be achieved with the corresponding TSN feature of seamless redundancy (802.1CB [44]). Hence, we could use this component to carry duplicated versions of the most critical messages of the system over disjoint physical paths and, thus, ensure the reception of the most critical data with the use of redundant rings.

The foregoing specification should result in the definition of a minimum working set of components for implementing an avionics network that is based on TSN-capable interfaces. Not only is this a major step towards showing a potential application of a COTS solution to space, it also goes a long way towards starting the definition of an aerospace profile for TSN. Hence, we claim that this initial specification should contemplate the use of **802.1AS** timing, time-aware traffic shaping (**802.1Qbv**) with frame preemption (**802.1Qbu & 802.3br**), seamless redundancy (**802.1CB**) for greater robustness, and the **BMCA** for improved resilience. We set out to verify this hypothesis by implementing, and then verifying, the Miura 1 microlauncher that we present in the following section.

9.3 OVERVIEW OF THE MIURA 1 MICROLAUNCHER. THE SYSTEM ARCHITECTURE.

The Miura 1 is a suborbital sounding rocket that falls into the scope the new generation of low-cost, affordable space vehicles. Its construction is intended to showcase a demonstration of the technologies that will eventually be deployed in the larger Miura 5 vehicle, which will be able to lift payloads of up to 300 kg to the low Earth orbit (∼400 km). Thus, the Miura 1 has been developed to adhere to the principles of affordability and reusability that can be expected of the "New Space" vehicles. We have implemented the avionics systems of the launcher in accordance with this paradigm. We carried out this project in the framework of an industrial transfer action with collaborators from the Andalusian Institute of Astrophysics (IAA) and from our industrial partner Seven Solutions. We gave a presentation describing our preliminary results and the system that we built in collaboration with GMV Aerospace and Defense SAU in [193]. Thus, from the standpoint of the avionics nodes that we had to build, this paradigm placed a series of requirements that we outline in Section 9.3.1.

9.3.1 *System Requirements for the Miura Avionics*

Our TSN platform for supporting the avionics of Miura had to enforce several key requirements. Chief among them was the need to support robust and deterministic data transmission. We outline the main requirements of our system in the points below, which will drive the definition of the specifications that we present in Section 9.3.3.

– **The communication interface** had to be based on the standard, Gigabit Ethernet links over twisted pair wiring. Furthermore, the TSN system should be capable of handling flows with multiple priorities, including critical data, and reduce the overall PDV with frame preemption. In this context, the attainable determinism had to be better that 50 μs over 10 hops when sending 512-B frames. In addition, we needed to ensure a robust delivery of data with the use of seamless redundancy.

– **The network nodes** would be built out of a programmable SoC and feature two different versions: one with support for a four-port switching device – the Main

Board –, and a lightweight one – the Secondary Board – for a dual-port node. Moreover, we make an emphasis on the use of COTS components for its production and on the moderate resource consumption of the design, which had to allow the integration with other third-party cores from our partner GMV SAU for performing mission-related tasks. The nodes should be designed for a suborbital flight with relaxed radiation resistance constraints.

– **The software environment** would be based on a real-time OS: RTEMS v5 [152]. This environment would therefore allow us to support the execution of the real-time user-level tasks for our avionics nodes. Furthermore, we can also provide an execution environment with enhanced safety with a static memory paradigm for handling the internal variables of our applications. The RTEMS OS should also be able to handle the communications over the TSN interface with custom driver support. In this context, it should also integrate the necessary configuration APIs for TSN.

9.3.2 *An Introduction to the Avionics of Miura 1*

The Miura 1 launcher will need to implement a redundant TSN system to handle three different flows that are commonly found in avionics systems: command and control traffic, housekeeping telemetry data, and best-effort flows. The first two types of traffic are generally high-priority and, thus, require the use of redundancy protection. The latter class is usually composed of either best-effort video feeds or monitoring data. This specification drove the design of the network architecture that we show in Fig. 9.2.

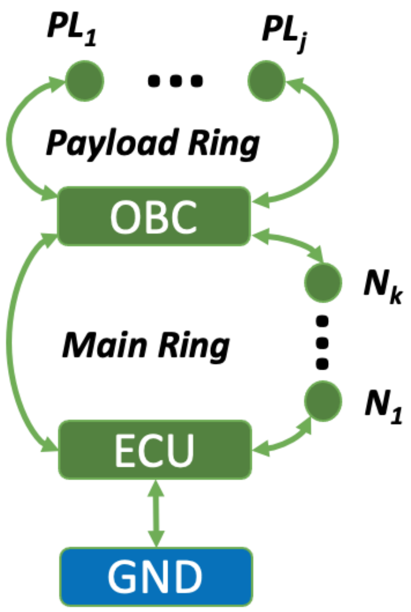


Figure 9.2
Expected high-level topology of the avionics network of the Miura 1 microlauncher.

This is a ring-style topology with the five different types of nodes that we describe below.

- The **on-board computer (*OBC*)** handles most flows in the system. It emits and receives control and telemetry data, and works as a TSN *listener* (a data sink) for all the video streams from the launcher. It is built with the four-port Main node and comes with additional built-in modules, such as a radio frequency interface (RF) for sending data back to ground control facilities.
- The **engine control unit (*ECU*)** handles the telemetry and control traffic. It is built with the four-port Main board to act as a bridge between the ground node and the rest of the network of the launcher.
- The **sensor and actuator boards (N_i *nodes*)** are intended to interface with the sensing and control elements of the rocket. Hence, their role is usually to operate as either TSN *listeners* or *talkers* for the processing of commands, telemetry or the occasional best-effort video feed. In our topology, they are built out of the dual-port Secondary boards and are placed on the main ring of the network in a daisy-chain layout.
- The **payload nodes (PL_k)**, as implied by their designation, interface with the payload modules of the launcher. They are placed on their own separate ring, where they operate as TSN *listeners* or *talkers* for handling telemetry or command messages.
- The **ground (GND)** node is one of the IC-317x series modules from National Instruments [194]. It supplies the time synchronization source before the start of the mission.

As for the classes of traffic that are expected to coexist on the TSN network of the Miura 1 avionics, we provide a short overview in the following points. Nonetheless, a more complete description can be found in [195].

- **The critical command and control flows** normally originate from the sensor/actuator nodes or the payload nodes; and are routed towards the OBC and ECU units for further processing or for relaying them to a ground station. These messages may be associated with control loops (and hence are periodic) or critical alarms. They must be handled with seamless redundancy (802.1CB) and traffic shapers with frame preemption (802.1Qbu & 802.3Qbr) to ensure their timely delivery without any data losses.
- **The medium priority housekeeping data** are the telemetry messages that are emitted from all the nodes in the system. They are usually forwarded to the OBC over redundant paths (802.1CB) to ensure their reception.
- **The best-effort flows** are the low priority video streams originating from several sensor/actuator nodes. They consist of different video feeds emitted at a constrained rate of 8 Mb/s.

9.3.3 *Design of the TSN Avionics Nodes*

We have supported the requirements for our use case by supplying a design based on the Z-7030 device from the Zynq-7000 family [7] from Xilinx. These are devices that feature a dual-core ARM processor with a separate section of programmable FPGA logic (PL).

Thus, we can run software components such as an operating system (OS) on the former and implement custom FPGA cores on the latter that could potentially interface with the ARM processor.

The architecture of our TSN nodes can be examined in Fig. 9.3. This design is parameterizable to allow us to target different versions of the architecture with different FPGA footprint. Thus, we could produce a four-port version for the design of the Main node, or a lightweight dual-port version for the firmware of the Secondary node. This architecture is highly versatile and lends itself to multiple applications, such the experiments with the Smart Grid that we presented in [169].

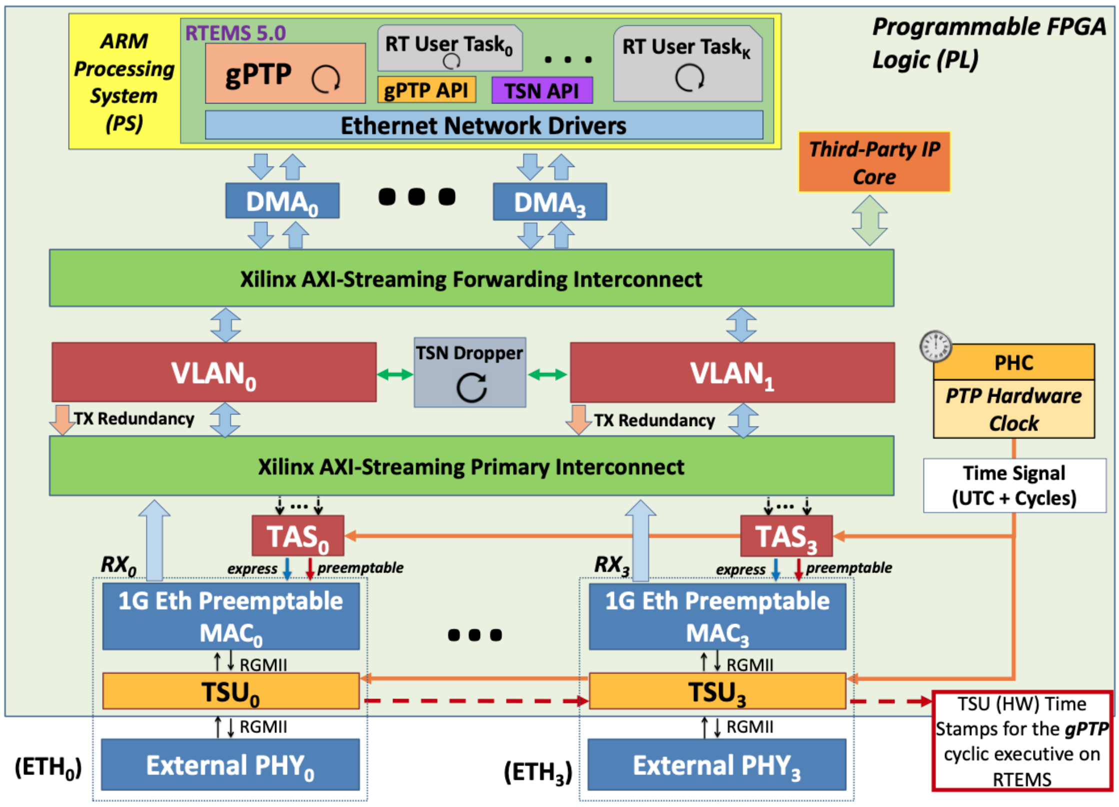


Figure 9.3
The general, system-level design of the Miura 1 avionics nodes, highlighting its RTEMS-based environment on the ARM processor and its FPGA-based elements.

9.3.3.1 *FPGA firmware of the Avionics Nodes*

The design of our TSN nodes relies on the combination of the following subsystems.

9.3.3.1.1 The Ethernet subsystem

This comprises the blocks tasked with supporting the underlying 1-Gb/s Ethernet communication service of the TSN system. These include several off-the-shelf and custom elements: The Xilinx DMA [145] for forwarding Ethernet data frames, a preemptable Ethernet MAC (802.3br) adapted from [148], RGMII/GMII bus infrastructure, and an external transceiver (PHY).

9.3.3.1.2 The timing subsystem

This ensures the deterministic operation of the TSN node by providing a common time reference for synchronizing the TSN traffic shapers. It is supported by the PTP hardware clock (PHC), which holds this time signal internally. It is in turn disciplined by the gPTP software on the ARM PS. This also includes the time-stamping units (TSUs) of the node for producing the hardware time stamps used by the gPTP protocol.

9.3.3.1.3 The TSN and switching subsystem

This takes care of the core functionality of the nodes: the deterministic forwarding amongst multiple ports. Consequently, the subsystem features data switching interconnects ("*Primary*" and "*Forwarding*"); time-aware traffic shapers for each port (802.1Qbv) that are complemented with frame preemption (802.1Qbu); VLAN modules (802.1Q) for traffic identification, flow routing, and redundant transmissions; and a Dropper Module for discarding duplicate packets when the seamless redundancy feature is in use (802.1CB).

9.3.3.2 ***FPGA Resource Utilization***

It is a requirement in our design to conserve resources and maintain overall usage below 60% to allow the integration with other mission FPGA cores. Hence, our architecture represents a compromise between deterministic performance and the correspondingly sensible usage of resources that this requirement entails.

We found that the main drivers of resource consumption were the TSN TAS traffic shapers, the VLAN modules, and the packet forwarding stages. They demand large amounts of block RAM (BRAM) primitives and general look-up table (LUT) logic for implementing buffers, finite state machines and traffic identification engines. As a result, we adhered to the approach that we outlined in Section 7.7 for implementing this architecture and preserving resources at the same time. Thus, this involved design decisions such as the use of shared VLAN modules between every two ports, support for 16 rules in the VLAN matching engine, the customization of the TAS to handle just four different priorities (queues), the use of small buffers (4 kB) for the TAS queues, or replacing the main comparators in the VLAN module with DSPs. The internal transceivers of our Z-7030 device were left idle, as we used an external PHY.

The main result of this design is that we can fit our architecture into the Z-7030 device whilst also allowing for the integration of other mission-related IP cores. This has resulted in an architecture that uses comparably fewer resources than other alternatives, like that of Xilinx [164]. In fact, the TSN architecture from Xilinx can only be targeted to larger Zynq-7000 or UltraScale devices to attain a similar level of features. The results of Table 9.1 show how our architecture can comply with these requirements by achieving overall LUT usage of ~60% and ~45% for the Main and the Secondary nodes, respectively. We obtained these utilization figures by combining our IP cores according to the approach that we outlined in Section 7.7.

Table 9.1
Resource consumption of the TSN implementation for the avionics nodes on the Z-7030 device.

Resource usage of the FPGA subcomponents

Module	**Slice LUTs**		**Slice Registers**		**Multiplexers**		**BRAM**		**DSPs**	
Xilinx DMA	2128	2.71%	3681	2.34%	1	0.002%	2	0.75%	0	0%
1G Ethernet MAC & Buffering	1581	2.01%	1948	1.24%	12	0.020%	6	2.26%	0	0%
GMII to RGMII Bridge	62	0.08%	132	0.08%	0	0%	0	0%	0	0%
*Ethernet Transceiver**	0	0%	0	0%	0	0%	0	0%	0	0%
Time-Stamping Unit (TSU)	1463	1.86%	2016	1.28%	0	0%	2	0.75%	0	0%
PTP Hardware Clock (PHC)	980	1.25%	1159	0.74%	2	0.003%	0.5	0.19%	0	0%
Switching Interconnects **(a)**	1796	2.28%	1798	1.14%	0	0%	9	3.40%	0	0%
Switching Interconnects **(b)**	545	0.69%	767	0.49%	0	0%	4.5	1.70%	0	0%
TSN VLAN Core	4220	5.37%	4624	2.94%	55	0.093%	3.5	1.32%	8	2%
Dropper Module	854	1.09%	1697	1.08%	6	0.010%	0	0 %	34	8.50%
Time-Aware Traffic Shaper (TAS)	1813	2.31%	3089	1.97%	99	0.168%	20	7.55%	0	0%
AMBA AXI Bus Infrastructure **(a)**	6150	7.82%	5221	3.32%	0	0%	0	0%	0	0%
AMBA AXI Bus Infrastructure **(b)**	3488	4.44%	3110	1.98%	0	0%	0	0%	0	0%
Implementation Totals										
(a) Main: 4 Ethernet Ports	***46408***	***59.04%***	***62587***	***39.81%***	***566***	***0.960%***	***136.5***	***51.51%***	***50***	***12.5%***
(b) Secondary: 2 Ethernet Ports	***34649***	***44.08%***	***48643***	***30.94%***	***313***	***0.53%***	***88.5***	***33.40%***	***42***	***10.50%***

*The nodes use an external transceiver chip.

Implementation totals from combining the FPGA cores as indicated in Section 7.7.

9.3.3.3 *The Software Components*

The development of the software environment was beyond the scope of this thesis project. Instead, it was supplied by our industrial partner Seven Solutions. This is the framework that will include the main software elements of our avionics nodes, which will be executed on the ARM processor of the Z-7030 device that our platform is based on. A diagram of the main elements of this software ecosystem can be examined in Fig. 9.3. Hence, we can observe that the entire software stack is built on top of the RTEMS OS. We made this choice of OS since it can provide a deterministic execution environment, and hence it is commonly used for avionics systems. In our implementation, we used the RTEMS OS to build a safe real-time execution environment with a static memory paradigm. The main elements that it integrates are the implementation of the gPTP protocol itself, which we adapted from [52] into a cyclic executive service, its custom Ethernet drivers for TSN ported from the Xilinx repositories, configuration APIs for TSN, and a real-time scheduling interface for launching custom user-level tasks with real-time execution constraints. To the best of our knowledge, this is one of the first implementations of a real-time OS (RTEMS) to feature an integration with the gPTP protocol for aerospace and, as a result, this implementation is suitable for receiving an ESA certification for space flight.

9.3.3.4 *The Embedded Platform for the Avionics Nodes*

We built a customized embedded platform to support the avionics nodes of the Miura 1 microlauncher. Thus, our design uses a PCB with a 160 mm by 160 mm form-factor, which simplifies its integration into the structure of the vehicle. It prominently features multiple interfaces for interacting with the main sensors and controllers of the launcher, such as a CAN interface, FMCs with low-pin count connectors, 20 generic I/O pins (GPIOs), and four Gigabit Ethernet ports. Furthermore, since the Miura 1 is a sounding rocket

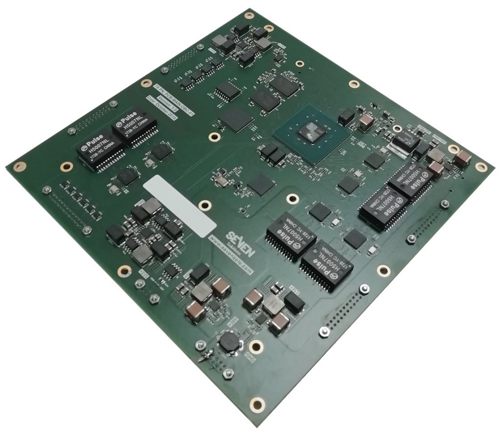


Figure 9.4
A picture of the hardware platform supporting the two variants (Main and Secondary) of the avionics nodes.

intended for a suborbital flight for demonstrating the application of new technologies. This implies that the requirement of radiation resistance on the electronics of the launcher will be more relaxed than that of a traditional space mission [196]. That is the reason why we based the construction of the embedded platform on an automotive-grade variant of the Z-7030 device, which would be enough for withstanding the thermal and vibration stresses of the suborbital flight. A picture of the resulting PCB for our nodes can be seen in Fig. 9.4. This is the common platform that supports both the Main and Secondary nodes of the Miura 1 avionics.

9.4 SYSTEM CHARACTERIZATION AND PERFORMANCE EVALUATION

We have characterized the operation of our avionics nodes through several test benches studying the performance of the TSN system under network scenarios comparable to those expected during the Miura 1 mission. We performed this evaluation with an incremental methodology: Firstly, we measured the performance of key elements of the system separately, such as its forwarding efficiency or the timing synchronization component. Next, we studied the behavior of the system in a larger system integration test that simulated the type of network environment that would be expected in a realistic avionics environment.

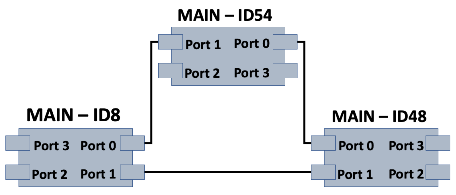
9.4.1 *Performance of the gPTP timing distribution with ring network topologies*

A correct operation of the gPTP timing service is essential for guaranteeing the synchronous operation of the traffic shapers in the TSN network, so that they can operate in a coordinated manner for sending data deterministically. Thus, we claimed that not only does its application to an avionics environment require a high level of accuracy, but a heightened level of resilience is desirable as well. We have assessed these aspects by studying the accuracy of the gPTP synchronization and its support for ring topologies in a laboratory environment (Fig. 9.5a).

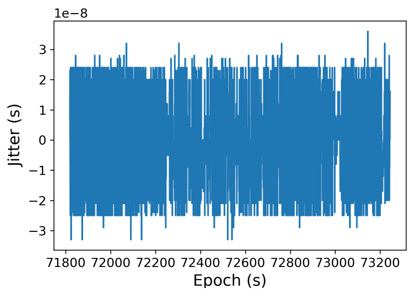
The use of ring topologies is a requirement for building an aerospace avionics network, which usually mandates that additional safeguards be provided in the form of redundant systems. From the perspective of the gPTP timing service, we can achieve this with the use of the best master clock algorithm (BMCA). Hence, the BMCA supports redundant timing paths and automatically adjusts the role of each port to either *Master*, *Slave*, or *Passive*. As a result, the configuration of the gPTP service at each node can adapt dynamically to changes in the system topology originating from events such as link or node failure. Thus, with the use of the BMCA we can assure that synchronization would not be lost so long as there is an alternative path for reaching the timing source from the gPTP slave device.

The test that we present in this section has two main objectives. On the one hand, we aimed to verify the action of the BMCA by by supplying the Main nodes in Fig. 9.5a with the appropriate configuration whereby nodes ID54 and ID48 get to operate as timing slaves of ID8. On the other hand, we also assessed the timing distribution accuracy of our gPTP implementation for RTEMS between the master time source (ID8) and the slave nodes (ID54 and ID48) in the ring.

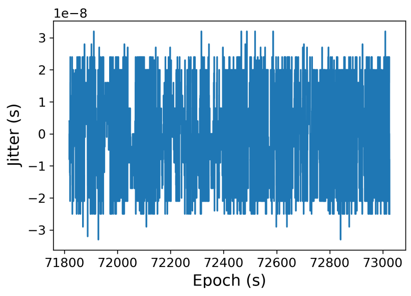
We verified that the BMCA operated correctly by examining the log files generated at each slave node. Hence, we noted that both the ID54 and ID48 were taking node ID8 as their timing source and that the role of their ports was adjusted accordingly (textitSlave or *Passive*) to ensure the reception of the protocol messages from the timing source. As for the attainable accuracy of the system, we assessed it through the synchronization jitter value that the gPTP service reported at each node. This jitter value is the slave-to-master time difference between the PHCs at the master and slave nodes in the network. Hence, we found that we could support ring topologies with accuracy levels below 100 ns.



a



b



c

Figure 9.5
Characterization of the gPTP synchronization performance with ring configurations. Assessment of the gPTP synchronization efficiency using a ring topology configuration. ***Fig. 9.5a*** is the diagram of the experiment, which measures the synchronization accuracy between the master node *ID8* and the slaves at *ID54* (***Fig. 9.5b***) and *ID48* (***Fig. 9.5c***).

9.4.2 *The Baseline Determinism Efficiency for Data Forwarding*

The forwarding efficiency of the underlying FPGA switching fabric in our nodes is one the decisive aspects impacting the overall determinism of the TSN nodes. Hence, if the underlying switching fabric presents large PDV swings under relatively idle conditions, it may be difficult to apply a GCL schedule with the TSN TAS shapers that manages to attain a low delivery variation. Thus, in order to estimate the baseline performance of the system, we studied the attainable PDV of our nodes in test scenario that is fully devoted to the transmission of probe frames.

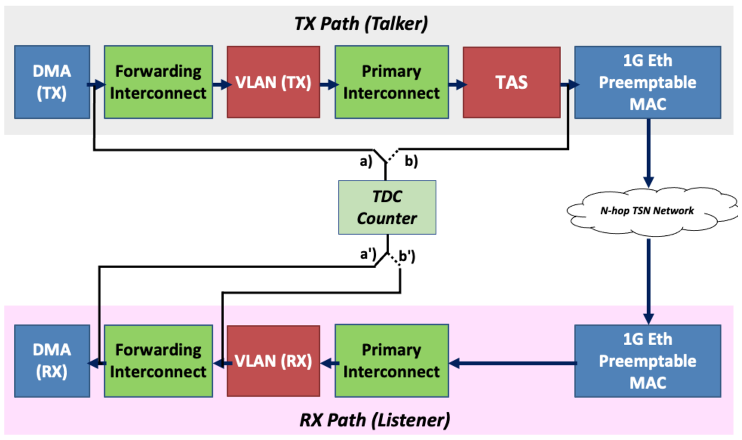


Figure 9.6
The probing points along the transmission and reception data paths of our nodes for measuring the performance of the TSN system with the assistance of a TDC counter.

This test would therefore allow us to estimate the attainable baseline determinism of our nodes. Our setup used a one-hop network (a daisy-chain of nodes) to perform an experimental characterization which consisted of the injection of more than a million probe frames between two Main nodes. The nodes were in turn fitted with a diagnostics core that could produce an analog trigger upon detecting the start of an Ethernet frame, as shown in Fig. 9.6. Next, we used a time-to-digital converter (TDC) counter [88] connected between the points *a)* (egress path of the emitter) and *a′)* (ingress path of the receiver) to calculate the flight time of the packets across the network and, thus, calculate their main statistical indicators, such as the PDV or the average end-to-end latency.

As a result, this test provides an estimation of the expected packet delay variation (PDV) of the system under ideal conditions. Hence, for 500-B probe frames, the average single-hop delay was pegged to 17.49 μs with a PDV of ∼301 ns, as observed in Fig. 9.7.

9.4.3 *Evaluation of Determinism in a Demonstration Avionics Setup*

We concluded the characterization phase by studying the attainable determinism of our TSN solution in a simulated avionics network that implemented a representative layout of the type of system that would eventually be built into the Miura 1 launcher. The layout of the test bench can be seen in Fig. 9.8, and it is in turn a simplified version of general network architecture that we introduced in Fig. 9.3. This is a representative test that we also demonstrated as part of our presentation in [193].

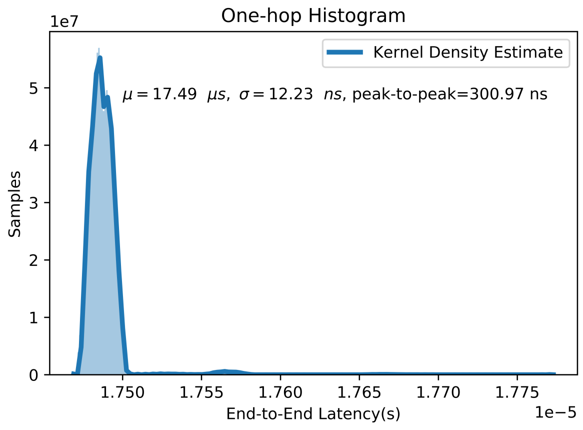


Figure 9.7
The end-to-end latency of a single-hop TSN network shows that the average latency (determinism) in this ideal test remains around 17.49 μs, with a PDV of 300.97 ns and a standard deviation of 12.23 ns.

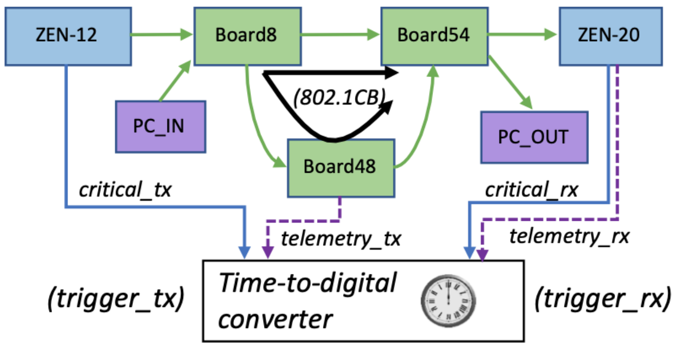


Figure 9.8
Diagram of the test bench for simulating the traffic landscape of a typical microlauncher mission.

The test uses three Main nodes (Boards 8, 48 and 54) to create a ring topology, and two TSN end-points for injecting and receiving time-critical flows based on the dual-port ZEN Board from Seven Solutions S.L. [8], which we have used as a development platform in a previous project [169]. The TSN flows handled by the system are indicated in Table 9.2, where we can observe that our test will make use of command and control (*CC*) messages, payload telemetry (*Tel*), bulk best-effort (*BE*) video, and gPTP synchronization messages.

9.4.3.1 *Configuration of the Demonstration Setup*

In our test, we forward the commands and control messages from the TSN ZEN talker towards the ZEN listener using a redundant network path. These messages are deemed high-priority critical messages, and we inject them into the network upon receiving analog triggers produced at a signal generator with a variable rate that is connected to the ZEN talker. Hence, we can vary the frequency of the signal generator to simulate the production of critical messages at different rates. The bulk, best-effort video is injected through one of the available Ethernet ports of the ZEN talker and sent to a receiving PC attached to the ZEN listener. As for the payload telemetry, we simulated its presence in the system by having an RTEMS task inject these messages into the network at the

Table 9.2
The traffic classes considered for the demonstration of the TSN avionics system, alongside their main characteristics.

TSN Flows in the Demonstration

Traffic Class	Priority Level	Traffic Profile	Critical	Redundant
gPTP (Synchronization)	3	60-B gPTP protocol messages	*e*	(BMCA)
CCl	VLAN PCP 2	400 B at variable rate	*e*	Yes (802.1CB)
Tel	VLAN PCP 1	500 B at 1 frame/ms	*e*	-
BE (Video)	VLAN PCP 0	750 B at 20 Mb/s	*p*	-

Main Board 48. The medium-priority telemetry would then be routed towards the ZEN listener. Additionally, all the nodes in the system would get to exchange gPTP protocol messages amongst one another (on a point-to-point basis).

In our configuration, we considered that the synchronization, the commands and control, and the telemetry flows should be considered time-critical. Therefore, they required non-preemptable, express (*e*) forwarding and redundancy. The redundancy was in turn implemented either through the features of 802.1CB for TSN flows or by the action of the BMCA for synchronization. Moreover, since the best-effort video is the lower priority flow, we designated it as a preemptable (*p*) and assigned it with the lowest priority queue of the TAS shaper.

The assignment of TSN flows to the TAS queues follows the convention that they are associated with the queue number matching their VLAN priority code point (PCP). This is true for all TSN flows, except for the gPTP messages, which are always assigned to the higher priority queue (*Q3*). The traffic shaping policy applied to the entire network can be examined in Table 9.3, which describes the structure of the GCL policy that we used for the test: the slots designated for forwarding the different flows, their associated queue (*Qk*) settings, and the corresponding relative offsets (“*base time*”) of the shapers in each node to compensate for forwarding delays in the network. Although we determined these settings analytically for this relatively simple setup, we acknowledge that there exist different methodologies and algorithms that could be put to use when more complex topologies with aperiodic flows are used. An example of such a methodology is discussed in [197].

9.4.3.2 *Experimental Results*

We have assessed the overall determinism that could be achieved in our avionics test-bed by measuring the end-to-end latency for both *CC* and *Tel* messages in our test bench. Their end-to-end latency is defined as the flight time through the network, and we used their latency results to calculate the corresponding PDV for each flow. Thus, we measure the end-to-end latency of our flows as the difference between the time at the egress paths (*b*)) of the Zen talker and Board 48 for the *CC* and *Tel* flows, respectively, and that at the ingress path (*b′*)) of the ZEN listener, which is the designated end-point for both flows. It

Table 9.3
The GCL configuration applied during the demonstration. The table shows the configuration for each slot within the 1.012 ms cycle of the GCL by showing its corresponding contents, queue (O/c) and preemption status (e/p). The base time for each node in the network is also indicated.

Traffic Shaping Policy for the Demonstration TSN System

Interval (μs)	Q3(e)	Q2(e)	Q1(e)	Q0 (p)	Slot Contents
28	O	O	O	O	gPTP, CC, Tel, BE
478			c	c	gPTP, CC
28			O	O	gPTP, CC, Tel, BE
478			c	c	gPTP, CC
Node ID	ZEN 12	Board 8	Board 48	Board 54	ZEN 30
Base Time (μs)	0	10.5	21	21	-

should be noted that although both flows are designated as express, non-preemptable flows, their respective levels of priority for the TSN shapers are different. This resulted in the end-to-end latency values obtained in Fig. 9.9, where we can observe how this GCL succeeded at avoiding interference amongst the *CC* flow, the *Tel* traffic injected at Board 48, and the background video traffic.

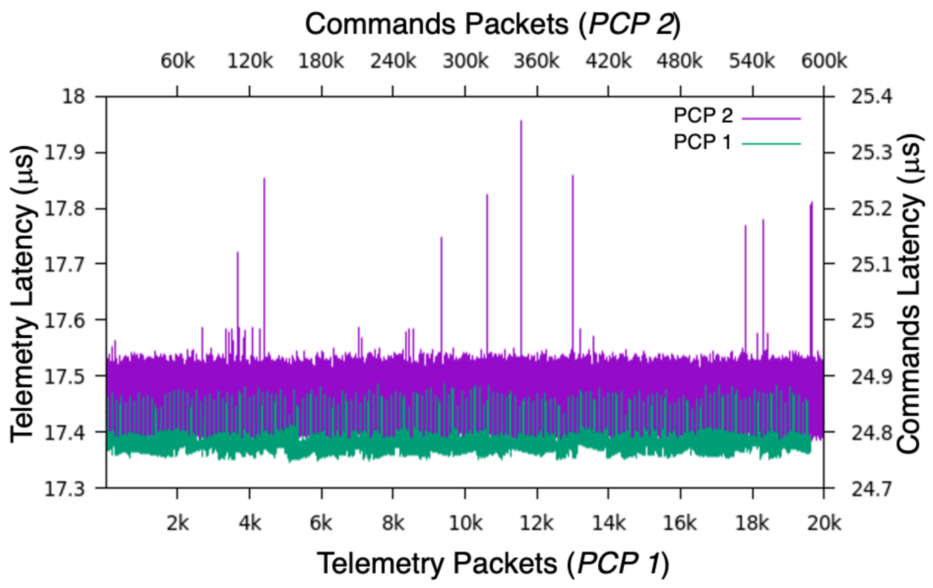


Figure 9.9
The end-to-end latency between the *CC* and *Tel* critical flows of the demonstration. We verified that the latency and PDV remain stable as mutual interference is averted by the TSN system.

Hence, in our results, the *CC* flow attained an end-to-end latency centered about ∼24.85 μs, with a PDV of 600 ns, whereas the *Tel* flow, which takes a shorter path through the network, achieved a PDV of 141 ns and a lower latency of ∼17.4 μs. Nonetheless, we observed that the PDV of the *CC* traffic would remain below ∼150 ns during most of the test, and that it presented occasional peaks of up to 600 ns. We have attributed this effect to the larger generation rate of *CC* messages, which would make interference with the higher priority gPTP messages more likely. We have speculated that this effect is a result of the compromises that we made in the implementation of our FPGA architecture to conserve resources. Specifically, these peaks are probably the result of contention at the internal switching crossbars. Therefore, we may be able to obtain better results architecture with larger switching and buffering elements in enhanced, future versions of our architecture.

Lastly, we studied the system reliability by characterizing the *CC* TSN flow during the experiment, which was protected with the use of redundant transmissions (802.1CB) given its classification as a time-critical flow. Our methodology for estimating the reliability in the reception of this flow consisted of totalizing the number of *CC* frames at the egress path (*b)*) of the ZEN talker, and those at the ingress path (*b'*)) of the ZEN listener after all the duplicates transmitted over the redundant ring in Fig. 9.9 are discarded at Board 54. The totalization operation is one of the built-in functions of the TDC counter and, as implied by its designation, it is a simple difference between the number of analog triggers detected at its two inputs. In our tests, since the ZEN boards generate an analog trigger when they transmit or receive a high-priority frame, we can use these signals in combination with the TDC to estimate the robustness of the system. Hence, during the experiment, we found that the difference between the number of transmitted and received *CC* frames remained constant at zero, even one of the redundant links (direct path or fallback path through Board 48) was severed. This is indicative that the integrity of the critical flow could be preserved with the use of the seamless redundancy feature.

9.5 CONCLUSION

We have shown that our TSN platform can be applied successfully to the resource-constrained scenario of the Miura 1 microlauncher. We have confirmed this claim through several experiments, which have validated our hypothesis that the use of COTS components, which is a prevalent trend in the new space vehicles, can also be applied to the implementation of their avionics systems.

We achieved this goal by supplying a TSN interface that makes considerate use of the available FPGA resources in our platform, while maintaining the required level of determinism and reliability that is expected in the forwarding of the time-critical flows handled in avionics systems. Furthermore, we could even consider this implementation an initial proposal of a TSN profile for aerospace applications and, given our results, we can confidently claim that any working application of TSN for space should at least contemplate the use of 802.1AS timing, 802.1Qbv traffic-shaping with frame preemption (802.1Qbu & 802.3br), and seamless redundancy (802.1CB). We think that our results indicate that this design is a promising solution with the capability of superseding the many legacy space fieldbuses that had commonly been used in space missions so far, such as Spacewire or MIL-STD-1553B, given the comparable levels of determinism and reliability of our solution. In addition, since our design is supported with a COTS platform with standard-based Ethernet physical interfaces, it is highly interoperable, and it can also achieve a far superior bandwidth of up to 1 Gb/s.

We have supported the development of our TSN avionics nodes with a platform that is based on the automotive-grade Z-7030 device, which provides the necessary flexibility and robustness needed to withstand the thermal, mechanical and radiation stresses of the suborbital flight that the Miura 1 sounding rocket will cover during its mission. Thus, this device features a powerful, highly modular architecture with an FPGA framework for supporting the design of the TSN communication cores, and an ARM microprocessor for running the RTEMS OS environment where we implemented the gPTP synchronization. Furthermore, to the best of our knowledge, our platform has been one of the first successful implementation to feature a fully functional gPTP timing stack for space in combination with the RTEMS OS. All of these elements have allowed us to attain a PDV value below 1 μs for the handling of the critical flows in the system, although we acknowledge that this figure could be improved if we repurposed our design to prioritize performance, rather than conserve resources. This would imply the implementation of larger buffers, faster packet processing stages, or more efficient switching interconnects, like the FPGA router design in [198].

In conclusion, we have shown a viable application of a TSN system for the implementation of the avionics of the Miura 1 microlauncher with a solution based on commercially available components that is highly flexible, interoperable, and easily upgradeable. Furthermore, we have supplied a real-time OS (RTEMS) to leverage the deterministic TSN interfaces of the avionics nodes of the launcher. This RTEMS implementation allows us to schedule real-time user tasks that can inject critical messages into the deterministic TSN system for further processing and forwarding. In addition, the RTEMS environment comes with one of the first successful applications of gPTP for aerospace. As a result, we can conclude by stating that our TSN-based avionics nodes for the Miura 1 can indeed deliver the deterministic performance required for avionics scenarios and could be considered a plausible COTS-based alternative for similar microlauncher platforms.

Part IV

Enhancements for TSN

10 INTEGRATION OF TSN WITH WHITE RABBIT TIMING

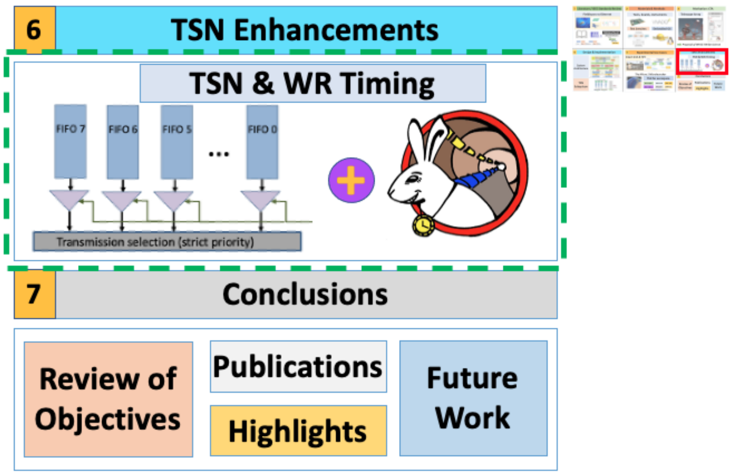


Figure 10.1
Overview of the contents of Chapter 10, where we feature an experimental integration of our TSN cores with WR timing.

The contents of this chapter are devoted to presenting our experimental integration with White Rabbit timing. We start by outlining the motivation for integrating our TSN cores with a WR system, which revolves around the fact that WR can deliver a more robust and precise form of synchronization that could pave the way for the use of TSN in scientific projects or for its integration in even larger industrial networks. Furthermore, the PDV could be enormously reduced. As a result, we developed a "hybrid" architecture combining our TSN cores with a WR timing system. This is the first aspect that we introduce in the chapter, where we also indicate that we used the WR-ZEN node [8] for prototyping the new architecture. Part of this study was conducted jointly with collaborators from the Technical University of Denmark (DTU) during a research visit in 2019. This is presented next, as we introduce our joint experimentation whereby we characterized the internal processing delays of the WR-capable TSN system and used this information for exploring the automatic generation of GCLs with an experimental tool from DTU. We conclude by assessing the determinism of the system, identifying its "bottlenecks", and proposing an upgrade path to enhance the attainable determinism as future work.

Chapter contents

10.1 MOTIVATION FOR ENHANCING THE TIMING STACK OF TSN

After characterizing our TSN system and demonstrating its applicability to the Smart Grid (Chapter 8) and as a low-cost solution for space microlaunchers (Chapter 9), we set out to implement an enhancement on its timing subsystem by replacing its components with an implementation of White Rabbit timing [101]. Indeed, we verified that gPTP could offer a degree of performance on the order of the tens of nanoseconds, which is in line with the results of other generic PTP implementations [179] that use hardware time-stamping, and this translates into a corresponding PDV on a similar scale. This was evidenced in our results in point 9.4.3.2. Hence, our rationale for seeking this upgrade was that since our traffic shaping elements are directly steered by a timing signal synchronized by the gPTP service, we would be able to achieve a lower PDV if the timing signal that we fed into the TAS shaper were more precise; i.e., the queues of the shaper would be activated with greater accuracy. WR timing can provide sub-nanosecond synchronization [75] and could therefore allow us to benefit from a substantial enhancement in the TSN system determinism; i.e., the TAS shapers could be synchronized on a sub-nanosecond scale theoretically.

To achieve this, we tapped into an existing design of this timing protocol which was implemented on the WR-ZEN Board from Seven Solutions [121]. This has also been our development and prototyping platform throughout most of this thesis project, and we introduced its main features in Section 4.1. This platform already features a working WR design on top of which we integrated our implementation for the IP cores of our TSN subsystem. Even though WR timing has not been contemplated officially by the IEEE standardization committees, the fact is that the latest specification for the high-accuracy PTP synchronization in the IEEE 1588-2019 [199] standard is heavily based on WR timing. Hence, we have posited that its use could substantially improve the performance of a TSN system and pave the way for its application to new scenarios, such as the data gathering and control networks of scientific infrastructure. Therefore, we outline the goals that we pursue with this experimental integration of TSN and WR in the points below. We will also characterize their degree of compliance with the tests that we present throughout this section to propose possible areas of improvement in future projects. Hence, our main goals with this experimental integration are:

- To improve the robustness of the TSN system. Since WR timing can support the establishment of large network topologies without a substantial loss of accuracy [139], it could be used to replace the standard gPTP synchronization for TSN, which may be unable to fulfill the requirements of large industrial networks with hundreds of elements [32]. These networks are usually mandated to provide synchronization accuracy better than 1 μs, as is the case of industrial automation or automotive networks, and the inaccuracies of large chains of nodes can eventually build up and degrade the gPTP synchronization up to a point when it can no longer fulfill the requirements of a particular network scenario. Hence, we have proposed the use of the more robust WR timing to circumvent this issue.
- To assess the influence on determinism of the use of a more precise timing solution. That is one of the reasons why we opted for an integration with WR timing, as its enhanced accuracy may allow for the definition of more precise GCL schedules.

Hence, this could result in better control of the end-to-end latency of the critical flows or even a reduced latency variation.

- To identify the components of the data path of our TSN nodes that represent determinism "bottlenecks" and propose the corresponding improvements to our architecture for reducing internal processing time jitter and latency.
- To discuss the feasibility of applying this enhanced TSN system for supporting the networks of scientific facilities, such as particle accelerators or telescope arrays; i.e., **the use of TSN for science**.

10.2 THE ARCHITECTURE OF OUR DEVELOPMENT PLATFORM

Our prototype of a WR-enhanced TSN system is based on the WR-ZEN timing board, which uses the Z-7015 SoC. This is a low-cost device, albeit fairly limited in its FPGA resources. We presented it in Section 4.1, where we introduced some of its main features, such as its dual-core ARM processing system for running an embedded Linux OS, or its built-in implementation of a functioning WR timing stack from our collaborator Seven Solutions. Therefore, to conduct our experiments with this new platform, what we did was to integrate our components for TSN networking, i.e., our FPGA IP cores for TSN, with the baseline WR timing design of the WR-ZEN Board. This resulted in the new hybrid, dual-port "*TSN & WR*" architecture that we show in Fig. 10.2.

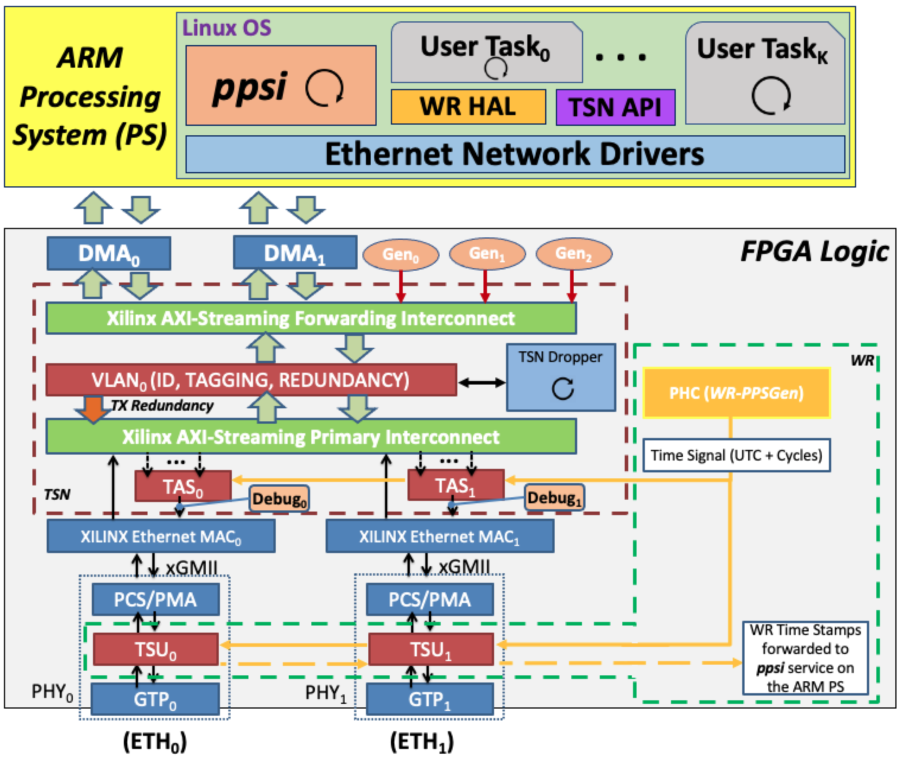


Figure 10.2
The system architecture of the experimental integration of TSN and WR timing on the Z-7015-based WR-ZEN Board.

As shown in the figure, the design includes both hardware and software components. The hardware comprises the original PCB design from Seven Solutions and the modified FPGA firmware with our IP cores for including the TSN functionality. Since the Z-7015 device comes with a reduced-size FPGA, we had to consider several tradeoffs to ensure that our proposed architecture could fit in the available resources of the device. We indicate these design compromises alongside the description of the main components of our implementation in the points below.

- The version of the TSN subsystem that we included in this architecture featured one single VLAN tagging/untagging module, a TAS shaper without preemption, and the enhancements for redundancy. As indicated previously, since the FPGA resources were highly constrained, we opted for a shared VLAN module between the two ports of the board, and we intentionally left out the use of frame preemption to maintain a lightweight TAS shaper.
- The networking subsystem is largely unchanged from the original architecture of the WR-ZEN board and provides a 1-Gb/s service. It uses the Xilinx cores for Ethernet networking by combining the Xilinx AXI-Streaming DMA engine and the Xilinx tri-mode Ethernet MAC with the PCS/PMA cores. Moreover, since the WR-ZEN Boards feature WR timing, they make use of a customized instantiation of the internal GTP transceivers of the Z-7015 device to ensure their deterministic behavior.
- As for the timing system, it contains the usual elements of a PTP hardware clock, designated as *PPSGen* in WR implementations, and time-stamping units. The use of WR timing results in the introduction of additional components for the timing system as well; such as a VCO for correcting phase differences, or a digital dual-mixer time difference (DDMTD) phase comparator for keeping track of small (sub-clock cycle) phase offset variations.
- Since our WR-ZEN nodes are dual-port devices with bridging capabilities, we implemented all the necessary crossbars to allow the forwarding of TSN packets between their ports. Nonetheless, in order to conserve FPGA resources from excessively large crossbar matrices, we used a setup with the same two-tier switching layout that we discussed in 7.5.
- Lastly, since the development of a prototype requires extensive validation, we also included several experimental debug cores within the architecture as needed to assist with the debugging process. These cores are usually meant to produce an analog trigger upon detection of a specific type of traffic to help measure the end-to-end latency or the latency variation with an external time counter. We also made use of traffic generators for injecting probe TSN frames during our experiments.

The software part of the design is run on an ARM processor for the most part; except for the process for steering the VCO oscillator with a PID controller, which is run as a standalone utility on an LM32 processor embedded within the WR logic (PPSGen). Admittedly, the development of the software stack for supporting a WR implementation is beyond the scope of this study. Nonetheless, we outline its main components in the following points and emphasize the elements that we added to complement our design.

- The embedded Linux OS of the WR-ZEN board was originally part of the system design from Seven Solutions, and hence most of its content was left unchanged.
- The Ethernet network drivers are a version of the original Xilinx Ethernet drivers and we have patched them to support the use of hardware time-stamping.
- A hardware abstraction layer (HAL) for managing the system peripherals of the board.
- The WR timing synchronization daemon: *ppsi*.
- We have also added a TSN configuration API (see Section 6.3.4) for interfacing with our cores.

Lastly, we have compiled the FPGA resource consumption of the different components of our architecture in Table 10.1. Thus, upon examining the usage figures of our implementation, we can highlight that one of the main outcomes of our design is that we have managed to fit a functioning TSN system with bridging and end-point capabilities into a small, low-cost cost device with support for WR timing. Larger FPGA devices are usually needed for running a TSN system, and hence we can conclude that this architecture showcases the versatility of our design, which we could fit into the Z-7015 SoC. In contrast, the Xilinx-based implementations of TSN [164], albeit more feature-rich than ours, usually target the higher end range of the Zynq-7000 SoCs or even the UltraScale devices; i.e., the range of SoCs above the Z-7030 device.

Table 10.1
The overall utilization figures of our experimental TSN node enhanced with WR timing on the Z-7015-based WR-ZEN Board.

Usage of the experimental TSN & WR architecture

Module	**Slice LUTs**		**Slice Registers**		**Multiplexers**		**BRAM**		**DSPs**		**GTPE**	
Xilinx DMA engine	2267	4.91%	3707	4.01%	0	0.00%	5	5.26%	0	0.00%	0	0 %
Xilinx 1G Ethernet Tri-Mode MAC & Ethernet Subsystem	3304	7.15%	5594	6.05%	45	0.13%	4	4.21%	0	0.00%	0	0%
White Rabbit Transceivers	756	1.64%	1062	1.15%	0	0.00%	1	1.05%	0	0.00%	2	50%
White Rabbit timing core	3160	6.84%	3205	3.47%	32	0.09%	31,5	33.16%	3	1.88%	0	0%
Time-stamping units	1373	2.97%	1975	2.14%	0	0.00%	2	2.11%	0	0.00%	0	0%
Switching Crossbars	857	1.85%	1299	1.41%	47	0.14%	8	8.42%	0	0.00%	0	0%
TAS shaper (no preemption)	1458	3.16%	2468	2.67%	34	0.10%	9	9.47%	0	0.00%	0	0%
VLAN (9 tagging rules)	4801	10.39%	5528	5.98%	55	0.16%	3,5	3.68%	8	5.00%	0	0%
Dropper (no timeout counters)	583	1.26%	688	0.74%	3	0.01%	0	0.00%	0	0.00%	0	0%
Debug Core	17	0.04%	9	0.01%	0	0.00%	0	0.00%	0	0.00%	0	0%
Traffic Generator	131	0.28%	118	0.13%	0	0.00%	0	0.00%	0	0.00%	0	0%
Common AXI4 Bus Infrastructure	4342	9.40%	4264	4.61%	0	0.00%	0	0.00%	0	0.00%	0	0%
Implementation Totals – Dual-port WR-ZEN												
Dual-port TSN & WR	*31730*	*68.68%*	*43906*	*47.52%*	*295*	*0.85%*	*84*	*88.42%*	*11*	*6.88%*	*2*	*50%*

The utilization figures in the table show the feasibility of fitting a working TSN design into a Z-7015 device with the application of our node architecture from Fig. 10.2 and our FPGA IP cores. This architecture offers a lightweight TSN implementation with some of the main features expected in these types of systems: time-aware traffic shaping (802.1Qbv), traffic identification with VLAN-tagging (802.1Q), resource reservation (802.1Qcc), and seamless redundancy (802.1CB). Furthermore, since our implementation consisted of a development platform, we also integrated debug cores and traffic generators into the design of the node to assist in the validation of its operation.

The overall usage figures show that the combined FPGA occupancy of all the subsystems in the node is relatively high: close to ∼69 % LUT usage or up to ∼88 % block RAM primitives. Nonetheless, this result goes a long way to show how our TSN architecture is flexible enough to allow different block connectivity combinations and customizations to target FPGA devices so constrained in their resources that they would easily fall out of the synthesizable scope of other larger TSN IP solutions from other manufacturers/implementers. The Z-7015 with WR timing could be considered a borderline use case for our architecture with a baseline set of features for TSN. Hence, this would allow us to target even larger devices where we could gradually expand our TSN architecture features as more FPGA "*real estate*" becomes available. In the case of the TSN and WR design that we present in this section, we had to make several implementation decisions to ensure that we could make such a tight fit. We enumerate some of these implementation decisions alongside the main characteristics of our design in the points below.

- Since the existing implementation of the WR timing and Ethernet subsystems already took up a substantial amount of FPGA resources on their own (e.g. ∼33 % and ∼4 % of all the BRAM primitives for the WR timing core and the Ethernet subsystem, respectively), we had to reduce the features of our TSN cores and share some resources between the data paths of the two Ethernet ports of the WR-ZEN. Specifically, our architecture makes use of the following approach for instantiating the cores:
  - 2× Ethernet subsystems (one per port): Xilinx DMA, Xilinx 1G Ethernet Subsystem & Tri-mode MAC, WR Transceivers.
  - 1× WR timing core for the entire node.
  - 2× time-stamping units (one per port).
  - 1× VLAN-tagging module (shared between the two ports).
  - 1× switching crossbar split into a primary and a forwarding stage between the two ports.
  - 2× TAS shapers (one per port).
  - 1× TSN Dropper module shared between the two ports.
  - 2× debug cores and 3× traffic generators for validation purposes for the entire node.
- Besides instantiating the system cores using the aforementioned methodology, we also trimmed their occupancy by leaving out some of their features from our implementation, as indicated next.
  - For the TAS shaper, we left out the use of the frame preemption mechanism to conserve BRAM primitives.

– For the VLAN module, we reduced the maximum number of translation rules that its comparison logic can handle internally down to nine rules. This reduced LUT and comparator usage.

– For the TSN Dropper module for discarding duplicates, we omitted the fail-safe timeout counter mechanism of its internal sliding windows. This reduced the use of DSP comparators and the associated routing congestion of their implementation.

The foregoing considerations have thus allowed us to successfully implement a TSN system with WR timing on a relatively small device with the utilization values of Table 10.1.

10.3 EXPERIMENTAL CHARACTERIZATION

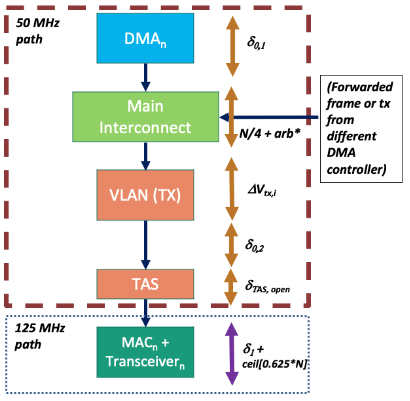
We have characterized the new architecture by firstly carrying out an evaluation of its efficiency for forwarding packets. Then, we assessed its attainable determinism as the resulting end-to-end latency after applying an automatically generated GCL schedule that could take the peculiarities of our architecture into account. This latter part was part of the joint characterization studies of the system that we carried out during a research visit at the Technical University of Denmark (DTU) in Köngens Lyngby (Denmark).

10.3.1 *The evaluation of the forwarding delays*

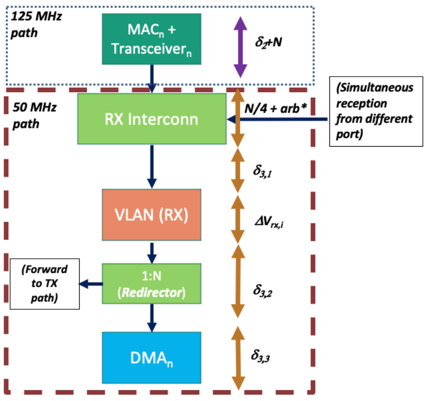
The characterization of the new platform starts off by examining the forwarding delays for packets of varying sizes. An accurate characterization of these values is essential for producing optimum GCL schedules analytically or even manually, as the routing and schedule generation algorithms for TSN usually require a sound knowledge of the internal routing delays and processing delays in the node logic to produce a meaningful time slot structure for handling the TSN flows with a time-driven schedule.

Our goals with this characterization were twofold: on the one hand, we wanted to evaluate if the use of the high-accuracy WR timing could pose any benefit for our TSN system. On the other hand, since these experiments were conducted in the framework of a research visit at DTU with Prof. Paul Pop, we wanted to show that the production of GCLs could be automatized with their metaheuristics-based tools [156]. Specifically, we aimed to validate that the automatically generated GCL policies we worked with could fulfill the requirements of a given network scenario, provided that our forwarding delay model of the node is correct.

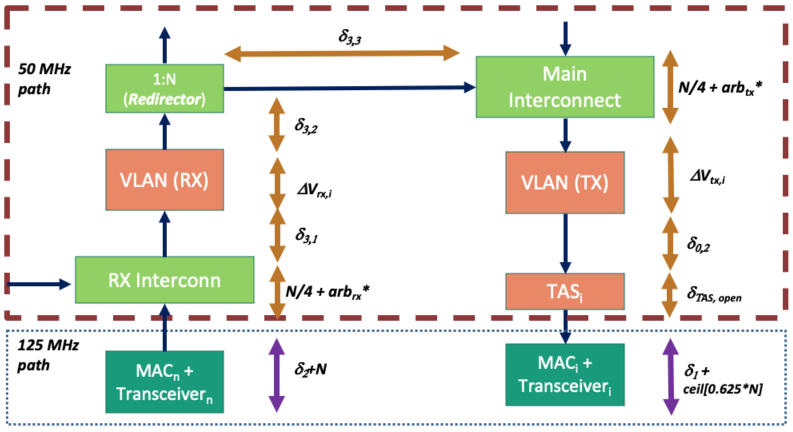
Hence, the reason why we performed this characterization was because the automated GCL-producing tools for TSN need to take several parameters into consideration: the desired network topology, the types of flows that will be forwarded (including frame size, emission period, deadline, . . .), and the expected processing delays at each bridge node. The first two parameters are usually specified by the user or implied by the application, whereas the latter parameters are implementation-dependent and should be supplied by the implementer. As a result, we set out to study the processing delays of our nodes associated the transmission (TX), reception (RX), and forwarding (FWD) operations of the different types of Ethernet frames handled in the system. We accomplished this by building a model of the corresponding data path and then deriving an analytical expression for estimating the delays of frames with different sizes. The model for the TX data path can be seen in Fig. 10.3a, that of the RX path is in Fig. 10.3b, and that of the FWD path is depicted in Fig. 10.3c. The corresponding analytical expressions for each path are in the equations in (10.1), (10.2), and (10.3), respectively.



a



b



c

Figure 10.3
The data path models for estimating the inherent forwarding and processing delays of our architecture for its integration with an automated configuration tool from DTU. The models study the delays of TX (Fig. 10.3a), RX (Fig. 10.3b), and FWD (Fig. 10.3c) data paths.

Aside from the fact that some of our IP cores can incur different processing delays as a function of the type of frame (i.e., gPTP, TSN, or ordinary Ethernet), some of the main sources of internal delays stem from the use of AXI-Streaming crossbar switches. These switches need to implement arbitration mechanisms between their ingress ports and may also include internal buffering elements that, in some cases, were configured to operate in "packet mode" forwarding; i.e., they accumulate the entire frame before it is injected into the crossbar switch. These characteristics can further increase the internal forwarding delays of the node or even contribute to an undesired increase in packet delay variation.

Thus, our delay models have two main components: a fixed delay part, and a variable delay parameter. We found that this latter parameter was a function of the frame size and the type of message. This can be seen in the following expressions for the estimated TX (10.1), RX (10.2), FWD (10.3) delays.

$$\delta_{tx} = \left[\delta_{0,1} + \delta_{0,2} + \delta_{TAS,open} + \Delta V_{tx,i} + \frac{N}{4} + arb(t)\right] t_{50MHz} + [\delta_1 + ceiling(0.625N)]t_{125MHz} \tag{10.1}$$

$$\delta_{rx} = [\delta_2 + N]\, t_{125MHz} + \left[\delta_{3,1} + \delta_{3,2} + \delta_{3,3} + \Delta V_{rx,i} + \frac{N}{4} + arb(t)\right] t_{50MHz} \tag{10.2}$$

$$\begin{aligned}\delta_{rxn,txi} = [\delta_2 + N + \delta_1 + ceiling(0.625N)]\, t_{125MHz} + \Big[\frac{N}{2} + arb(t)_{tx} + arb(t)_{rx} + \delta_{3,1} + \delta_{3,2} + \\ + \delta_{3,3} + \Delta V_{rx,i} + \Delta V_{tx,i} + \delta_{0,2} + \delta_{TAS,open}\Big]\, t_{50MHz}\end{aligned} \tag{10.3}$$

In the preceding expressions, the fixed delay terms of $\delta_{0,i}$, δ_1, δ_2, $\delta_{3,k}$, and $\delta_{TAS,open}$ are associated with the propagation of a given message through the logic fabric of the node. During our characterization, we determined that these elements had a fixed latency of a few clock cycles and were usually associated with the packet going through several concatenated data registers along its path. In contrast, the variable delay terms are associated with clock domain crossings (the "*ceiling(*)*" terms), the internal buffering elements, or the processing of different types of packets on the TX or RX paths of the VLAN module: $\Delta V_{tx,i}$ and $\Delta V_{rx,i}$ respectively. Thus, they may have a dependence on the packet size (N) or the type of traffic (TSN, gPTP, Ethernet).

As shown in Fig. 10.3, our architecture deals with two main clock domains for data transmission: 50 MHz and 125 MHz. Consequently, we estimated the values of the preceding parameters experimentally by inserting ILA debug cores into our design to measure the propagation time through our logic as clock cycles in their corresponding domain. Other parameters such as the delay of clock domain crossings were estimated by transmitting packets of varying sizes for building a linear regression estimation. For the processing delays of the VLAN core, we used the values of its behavioral simulation to quantify its processing delay for different types of messages. We outline the meaning of each parameter in the following points, and the values that we determined experimentally for each one can be found in Tables 10.2 for the fixed delay terms, 10.3 for the VLAN TX/RX latency.

- The system clock domains present along the data path are indicated by t_{50MHz} (20-ns clock cycle), and t_{125MHz} (8-ns clock cycle).
- For the TX data path, the following parameters apply:
  - $\delta_{0,1}$ is the propagation delay from the DMA controller to the main interconnect.
  - $\delta_{0,2}$ is the path delay from the TX VLAN tagging module through to the TAS interface.

  - $\delta_{TAS,open}$ is the propagation time through the TAS module when no GCL is supplied and all of its queues are open.
  - δ_1 is the constant path delay through the MAC and the Ethernet transceiver modules.
  - $\Delta V_{tx,i}$ is the processing latency of the TX path of the VLAN module for different types of packets: Regular Ethernet frame/PTP or TSN-tagged Ethernet frame.
- As for the RX data path, the following parameters apply.
  - δ_2 is the constant RX path delay through the MAC and the Ethernet transceiver modules.
  - $\delta_{3,1}$ is the path delay from the RX interconnect to the RX path of the VLAN processing module.
  - $\delta_{3,2}$ is the propagation delay from the VLAN Module to the *Redirector Interconnect* (the forwarding crossbar).
  - $\delta_{3,3}$ is the path delay from the redirector (*forwarding*) interconnect to the RX interface of the DMA controller.
  - $\Delta V_{rx,i}$ is the VLAN RX processing delay, dependent on message type: Regular Ethernet Frame/PTP or TSN-tagged Ethernet frame.
- The arbitration delays incurred at the different crossbar switches on account of the effects of their arbitration mechanisms are designated as $arb(t)$. This is a non-deterministic parameter dependent on network occupation and the size of the crossbar switches. For this characterization, we estimated it as the worst-case value in (10.4), where N_p is the largest possible Ethernet frame size of 1500 B, and n_p is the number of input ports into the crossbar.

$$arb(t) = 2 + (n_p - 1)\,\frac{N_p}{4} \tag{10.4}$$

Table 10.2
The fixed delay terms on the TX and RX paths of the experimental node integrating TSN & WR timing.

Fixed delay terms

Parameter	δ_1	δ_2	$\delta_{0,1}$	$\delta_{0,2}$	$\delta_{TAS,open}$	$\delta_{3,1}$	$\delta_{3,2}$	$\delta_{3,3}$
Cycles	39	28	1	15	27	5	10	9
Clock Domain	*125 MHz*		*50 MHz*					

As a result, we calculated the expected processing delays at the node for different types of messages and packet sizes by plugging the values of this characterization into our expressions in (10.1), (10.2), and (10.3). The results of these estimates are shown in Tables 10.4, 10.5, and 10.6 for the expected TX, RX, and FWD latencies of the node, respectively. Thus, after conducting this characterization, we supplied these results to the scheduling tools from DTU [156] to proceed with the next phase of our experiments where we assess the overall performance and determinism of the system (see Section 10.3.2).

Table 10.3
The processing time on the TX and RX paths of the VLAN module for different types of traffic.

$\Delta V_{tx,i}$ **(50 MHz)**		
PTP	**Ethernet**	**TSN**
42 *cycles*	55 *cycles*	62 *cycles*

$\Delta V_{rx,i}$ **(50 MHz)**		
PTP	**Ethernet**	**TSN**
26 *cycles*		40 *cycles*

10.3.2 *The GCL scheduling tests. The collaboration of the DTU research visit*

We assessed the overall system-level performance of our WR-enhanced TSN prototype during the next phase of our characterization experiments. This involved the definition of an experimental test bench where we could deploy several of our WR-ZEN prototypes to try out the effects of different configuration profiles produced with the tools from DTU.

This led to the deployment of a test bench like the one shown in Fig. 10.4. As seen in the picture, the setup consisted of three WR-ZEN boards fitted with our experimental design that were connected back-to-back in a daisy chain. For the purpose of the experiment, each board was termed as a TSN "switch" (SW) to emphasize their role as bridging devices, even though our implementation could only work as a dual-port device at the time (future versions of our prototype may come with an expansion card for additional Ethernet interfaces). Thus, our experiment will feature three different daisy-chained switches whereby the switches at either edge of the network will be forwarding different TSN flows to each other. The intermediate switching node will therefore operate as a bridge between the switches at the edges.

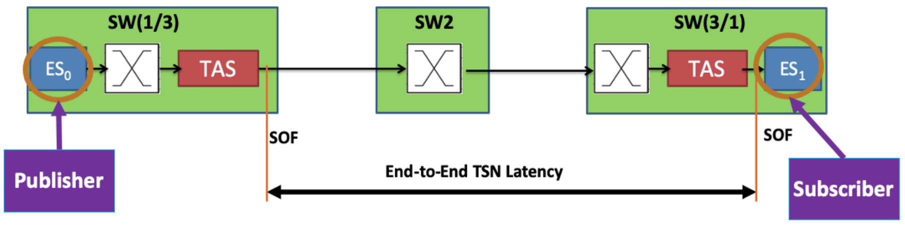


Figure 10.4
Schematic representation of the experimental test bench for evaluating the performance of our prototype implementation of TSN with WR timing.

As shown in Fig. 10.4, the switches SW1 and SW3 will be tasked with exchanging probe TSN messages between each other. These messages will be produced with the traffic generator IP cores that we introduced when we presented the architecture of the system in 10.2. We modeled this transmission as an exchange between two end systems (ES_i): a traffic generator working as a publisher at the transmitting switching node (ES_0), and the receiving switch itself (ES_1) in the role of a subscriber.

Table 10.4
The estimated TX forwarding latency of the TSN & WR architecture.

Estimated TX forwarding latency

N (Bytes)	PTP [t(ns)]	Ethernet [t(ns)]	TSN [t(ns)]
60	4116	4376	4516
100	-	5776	5916
200	-	9272	9412
300	-	12776	12916
400	-	16272	16412
500	-	19776	19916
600	-	23272	23412
700	-	26776	26916
800	-	30272	30412
900	-	33776	33916
1000	-	37272	37412
1100	-	40776	40916
1200	-	44272	44412
1300	-	47776	47916
1400	-	51272	51412
1500	-	54776	54916

To assess the performance of the system, we measured the determinism of each flow handled by the TSN network as the flight time between the egress path of the transmitting switch and the ingress path of the subscriber (the egress path of the receiving switch). As observed in the figure, both probing points are located after a TAS shaper to synchronize the frames injected into the network to the GCL schedule, since the traffic generator at ES_0 is not synchronized to the TAS traffic-shaping schedule. Thus, the main indicators of the attainable performance of the system will be the end-to-end latency of each flow during the experiments, and its associated packet delay variation (PDV). A reduced value for the PDV will therefore result in enhanced determinism.

The configuration for our tests was driven by the use of the automated configuration tool from DTU. As mentioned previously, the use of the tool takes advantage of our forwarding model (Section 10.3.1) to produce a meaningful configuration that we could use for assessing the overall determinism of the system. For our tests, we had the tool produce a configuration with four flows (two flows emitted per switch) that were emitted periodically within a *hyperperiod* (T_p) interval. The message emission schedule of each flow at the end systems is summarized in Table 10.7, whereas the resulting GCL policy that the automated derived for each TSN switch is shown graphically in Fig. 10.5. The study that we have presented for this thesis project contains preliminary results of our characterization for *hyperperiod* values of $T_p = (4800, 9600)\mu s$.

Table 10.5
The estimated RX forwarding latency of the TSN & WR architecture.

Estimated RX forwarding latency

N (Bytes)	PTP [t(ns)]	Ethernet [t(ns)]	TSN [t(ns)]
60	2912	2912	3192
100	-	3632	3912
200	-	5432	5712
300	-	7232	7512
400	-	9032	9312
500	-	10832	11112
600	-	12632	12912
700	-	14432	14712
800	-	16232	16512
900	-	18032	18312
1000	-	19832	20112
1100	-	21632	21912
1200	-	23432	23712
1300	-	25232	25512
1400	-	27032	27312
1500	-	28832	29112

This is a relatively trivial case, but useful for proving the usefulness of the tool for producing a meaningful configuration, which is driven by the user-supplied constraints for the traffic classes of the system (periodicity, payload size, deadline, ...) and the corresponding system latency model. We discuss our results in Section 10.3.2.1, and the insights on the production of GCL schedules that we derived from these experiments are outlined in 10.3.2.2.

10.3.2.1 *Preliminary experimental results*

The automated scheduling tool from DTU produced the settings with the time slot structure that we have shown in Fig. 10.5. This time slot structure is valid for fulfilling the requirements of our experimental scenario, but, given the experimental nature of the DTU configuration tool, we realized that the time length for each one of the slots in the GCL schedule was actually larger than the frame transmission time for each type of message; i.e., the slot was longer than the time it took to transmit the messages of a given traffic class. Since the traffic generators of our nodes were not synchronized to the TAS scheduling cycle, this had the undesired effect of producing a large latency variation. Hence, we adjusted the slot lengths manually for those GCL entries intended for the transmission of TSN messages so that the corresponding queues would only remain open long enough to allow the transmission of one single TSN frame at a time.

Table 10.6
The estimated FWD bridging latency of the TSN & WR architecture. Bridging is only allowed for TSN messages.

Estimated FWD bridging latency

N (Bytes)	TSN [t(ns)]
60	7080
100	9200
200	14496
300	19800
400	25096
500	30400
600	35696
700	41000
800	46296
900	51600
1000	56896
1100	62200
1200	67496
1300	72800
1400	78096
1500	83400

In consequence, we conducted these validation experiments with a modified version of the configuration produced by the DTU scheduling tool. This modified configuration retained the original time structure of Fig. 10.5, which indicates that our processing delay model for each node from Section 10.3.1 is valid, but used a shorter slot duration for the GCL entries reserved for the transmission of TSN frames. We determined that this modification helped reduce the PDV in our experimental setup (see 10.3.2.2), and the new slot duration for these entries can be easily derived from the characteristics of our system implementation, as shown in (10.5). In the expression, N_i indicates the frame length of the corresponding traffic class, and t_{50MHz} is the clock cycle time of the system bus in the 50 MHz clock domain of the TAS shaper (Fig. 10.3a). Since we are using a traffic shaper without preemption, the transmission of a frame cannot be halted once it is underway and the value of (10.5) can be taken as the maximum slot duration.

$$t_s \leq ceiling \left[\frac{N_i}{4} \right] \cdot t_{50MHz} \tag{10.5}$$

We show the preliminary results of these characterization experiments in Table 10.8. This table summarizes the behavior of the end-to-end latency observed for each one of the TSN flows in the experiment, where we generated two sets of GCL schedules for the two test cases with *hyperperiod* values of 4800 μs and 9600 μs, respectively.

Table 10.7
The constraints on the traffic classes involved in the experiment: period, frame size, priority, and routing.

The traffic classes for the test

Flow ID	Size (B)	Priority	Period	Routing
f_1	400	1	$\frac{T_p}{4}$	SW1 $\rightarrow$ SW3
f_2	100	2	$\frac{T_p}{2}$	
f_3	300	1	$\frac{T_p}{3}$	SW3 $\rightarrow$ SW1
f_4	60	2	T_p	

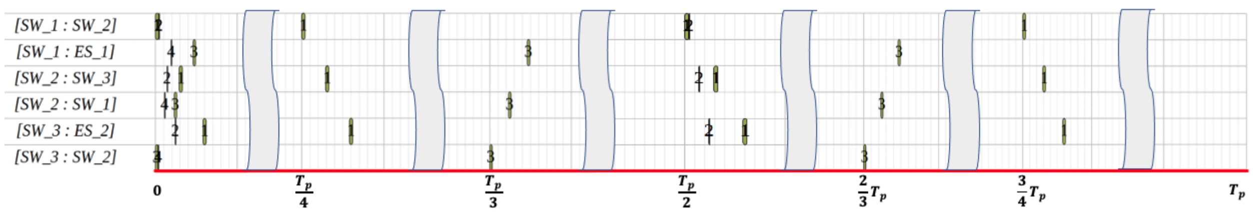


Figure 10.5
The GCL policy generated by the automated tool to fulfill the requirements in Table 10.7.

In the results, the "peak-to-peak" column is of special importance, as it shows the attainable determinism of the system and corresponds to the packet delay variation of a given flow. Moreover, these results show how the application of a scheduling policy with the traffic shaper can be used to tightly control the delivery times of a given TSN flow within bounded latency values. In the experiment, we achieved this by combining the effects of the application of the "base time" shaping offset of the TAS core with the definition of slot intervals with the duration specified in (10.5). Thus, provided that our model of the worst-case internal processing delays at each node is correct, this configuration has allowed us to control the end-to-end latency of the TSN flows in the experiment and, in addition, minimize their latency variation to enhance the overall degree of determinism of the system. The attainable PDV in our results sits below the microsecond threshold for most of the flows and remains around the hundreds of nanoseconds; except for the flow f_3, which had a PDV of $\sim$1.80 μs. This difference may be a consequence of an unforeseen arbitration issue or the effect of recurrent interference with the PTP messages of the WR protocol exchanged between the nodes, which have a higher priority ($Q3$) than those of flow f_3 ($Q1$). These results led us to derive the insights that we discuss in Section 10.3.2.2 on different strategies for producing GCL schedules.

10.3.2.2 *Insights on GCL schedule production*

The manual tuning process that we had to perform on the results of the DTU scheduling tool to minimize the PDV provides some clues as to how the generation of a GCL schedule can be directed to attain different effects. Indeed, we have observed that the time structure of the slots in the GCL policy can be adjusted to *a)* optimize the bandwidth of rate constrained flows, *b)* minimize the end-to-end latency, or *c)* target highly deterministic applications by minimizing the delivery jitter.

Table 10.8
The preliminary results of our evaluation of the attainable determinism of the experimental design of our WR-enhanced TSN nodes.

End-to-end latency for the test cases

	MAX		**min**		**Std.Dev.**		**Peak-to-Peak**	
Flow ID	$T_p = 4800\,\mu s$	$T_p = 9600\,\mu s$	$T_p = 4800\,\mu s$	$T_p = 9600\,\mu s$	$T_p = 4800\,\mu s$	$T_p = 9600\,\mu s$	$T_p = 4800\,\mu s$	$T_p = 9600\,\mu s$
f_1	58.34 μs	58.33 μs	57.86 μs	57.84 μs	163.78 ns	167.16 ns	487.22 ns	493.21 ns
f_2	19.89 μs	19.88 μs	19.75 μs	19.76 μs	29.85 ns	26.97 ns	135.84 ns	119.88 ns
f_3	46.27 μs	46.27 μs	44.46 μs	44.47 μs	128.65 ns	135.81 ns	1.82 μs	1.80 μs
f_3	15.82 μs	15.82 μs	15.76 μs	15.76 μs	9.72 ns	9.48 ns	63.13 ns	57.41 ns

a) The TSN system can be optimized to implement a simple time-division multiplexing scheme to prioritize the bandwidth of the lower priority, rate-constrained flows. In these cases, the duration of the slot for forwarding the rate-constrained traffic should be proportional to its emission rate and its maximum burst size.

b) Likewise, we could also minimize the end-to-end latency of a given flow by ensuring that its corresponding queue is assigned a higher priority and remains open during a substantial portion of the GCL schedule. Moreover, no "base time" scheduling offset should ideally be provided for these cases.

c) Lastly, for those scenarios requiring a service with "hard" determinism; i.e., a known delivery deadline and the reception of the TSN frames within a narrow window, we could apply a similar strategy to the one we deployed during the experimental characterization of our TSN nodes in the previous section. Hence, this methodology relies on the use of an accurate processing delay model that can characterize the worst-case internal propagation delay through the system nodes. Then, as shown in Fig. 10.6, we could use the "base time" offset setting of the TAS shaper at each node to compensate for the worst-case switching and processing delays of each node. We can further complement this strategy by applying the expression in (10.5) to reduce the length of the GCL entries forwarding the TSN traffic down to the minimum transmission time for emitting a particular frame. Since the TSN flows handled by the system are usually not synchronized to the TAS cyclic schedule, this step helps to reduce the PDV by synchronizing the TSN frames as they are injected into the network, as seen in Fig. 10.7.

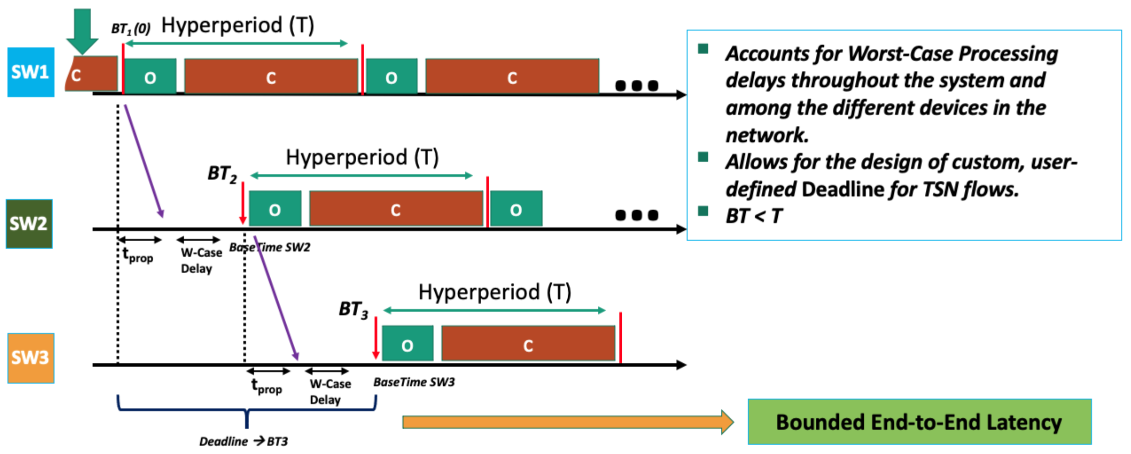

SW1
SW2
SW3
BT1(0) Hyperperiod (T)
BT2 Hyperperiod (T)
BT3 Hyperperiod (T)
C
O
t_{prop}
W-Case Delay
Base Time SW2
BaseTime SW3
Deadline → BT3
Accounts for Worst-Case Processing delays throughout the system and among the different devices in the network.
Allows for the design of custom, user-defined Deadline for TSN flows.
BT < T
Bounded End-to-End Latency

Figure 10.6
The application of the "base time" offset to deliver flows within a deadline and thus compensate for internal processing delays and worst-case forwarding jitter at each node.

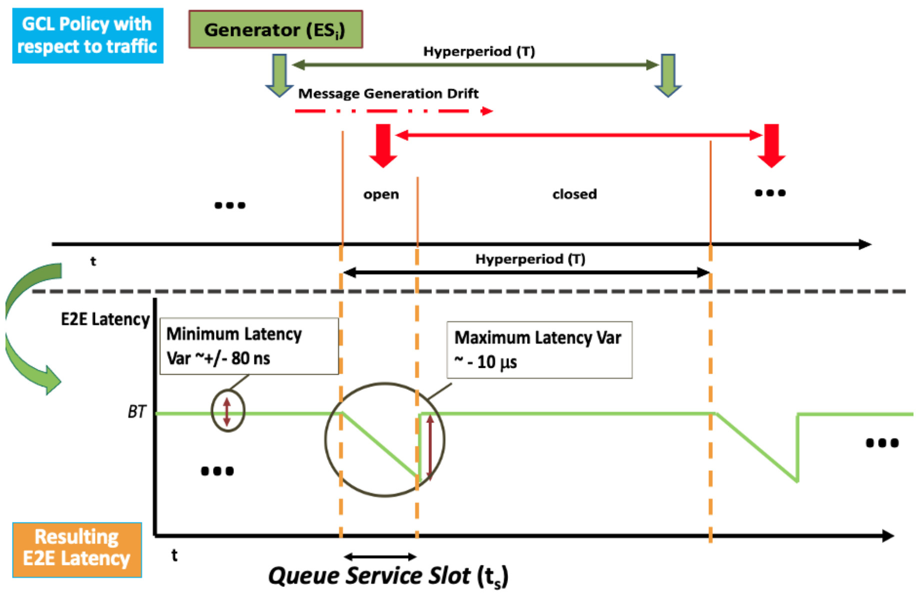

GCL Policy with respect to traffic
Generator (ES_i)
Hyperperiod (T)
Message Generation Drift
open
closed
t
Hyperperiod (T)
E2E Latency
Minimum Latency Var ~+/- 80 ns
Maximum Latency Var ~ - 10 µs
BT
Resulting E2E Latency
t
Queue Service Slot (t_s)

Figure 10.7
An example showing the relationship between the slot length for forwarding TSN messages and the attainable PDV.

10.4 FUTURE IMPROVEMENTS TO THE INTEGRATION WITH WR TIMING

Even though we have managed to implement a working TSN system that replaced its gPTP synchronization service with a WR timing stack, the experimental results that we obtained in 10.3.2.1 are not substantially better (i.e., showing a greater degree of determinism) than those we showcased in the experimental validation of our use case for an avionics network (Section 9.4.3). In fact, the end-to-end latency variation values of both cases are comparable and on the order of the hundreds of nanoseconds. This leads us to conclude that, although the use of WR timing can be beneficial given its robustness for transferring timing information with high accuracy over large networks [32], there does not seem to be an evident advantage to its use for enhancing the determinism of a TSN flow for our architecture in its current form.

White Rabbit timing can provide a highly accurate, sub-nanosecond synchronization service. As we mentioned in the strategy *c)* from our GCL generation insights (see 10.3.2.2), if we applied the suggested method of combining "base time" offsets with short intervals and steered the internal FSMs of the TAS shaper with a WR-synchronized timing signal simultaneously, we would then be able to synchronize the injection of data packets into the network on the same scale as WR timing. Since we are observing a worse level of performance than expected, we set out to identify its possible root causes and the actions for future improvement in the following points.

10.4.1 *Suggested optimizations to the TAS shaper*

We reviewed our implementation of the TAS shaper for TSN to identify possible sources of jitter in its operation that could degrade the overall determinism of the system. As WR timing is so highly precise, the presence of any inaccuracy in the design of our IP cores could easily be comparable in magnitude to the uncertainty of WR synchronization and, thus, degrade the determinism of the system. This does not necessarily imply a flaw in the design of the TAS shaper, which we built with the guidelines of the IEEE 802.1Qbv standard; but rather that its architecture was devised to be used jointly with gPTP timing, which usually has an accuracy on the order of the hundreds of nanoseconds. In the standard description of the TAS shaper, the use of elements such as synchronization chains for implementing crossings between different clock domains is implied. This should not pose an issue for gPTP timing but can be a major source of jitter if WR-like accuracy is to be targeted.

We verified this claim by measuring the synchronization jitter between the shapers at SW1 and SW3 in the experimental test bench of Fig. 10.4. The synchronization between any pair of shapers can be readily examined by comparing the relative points in time when their respective "*Cycle Start*" signals are triggered. As explained in 7.3.1.2, this signal is triggered periodically when the logic of the core detects that a new execution of the GCL scheduling cycle should be performed. Thus, we used the debug cores from our WR-enhanced TSN nodes to output the respective "*Cycle Start*" signals from nodes SW1 and SW3 into a TDC counter to quantify the amount of jitter that could be attributed to the implementation of the TAS shaper. The results of the experiment can be seen in Fig. 10.8.

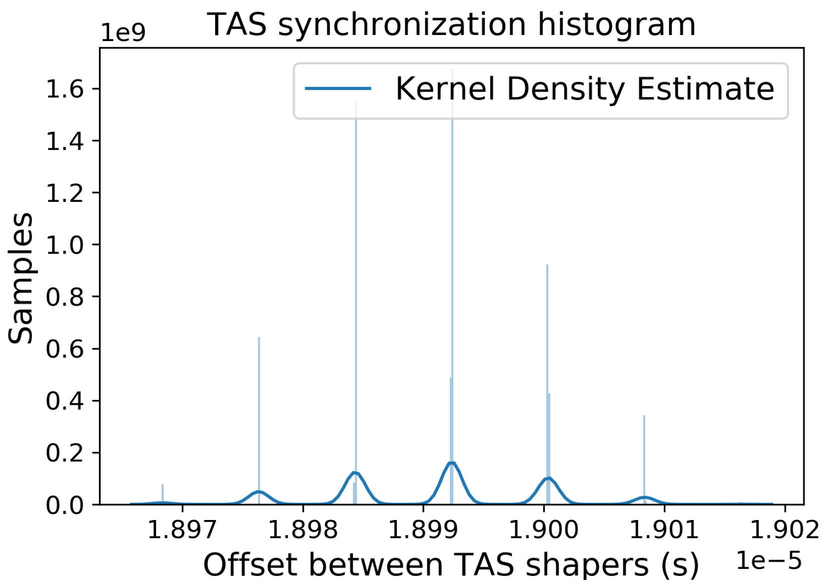


Figure 10.8
The distribution of the synchronization jitter between the TAS shapers in our experiment.

In the experiment, we applied a "base time" offset between the shapers of $\sim$19 μs, which is visible in the figure. We can also appreciate that the offset distribution shows evenly spaced peaks every $\sim$ 8 ns, **with a peak-to-peak variation of 48.105 ns**. This is a visualization of indeterminism stemming from the use of clock-crossing logic. This performance matches that of a gPTP service, as we saw in the Smart Grid experiments from Section 8.6.3.1. Nonetheless, it completely prevents us from taking full advantage of WR to enhance the determinism of our solution. As stated previously, we have identified the clock domain crossings at the interfaces between the TAS FSMs as the main culprits of this behavior, and hence we will work to optimize this aspect in future versions of our cores after this thesis project.

10.4.2 *Other optimizations for the node logic*

As seen during the modeling phase of the processing delays in our architecture, some of main contributors to the relatively large processing latency of our data path are the VLAN core and the effects of the crossbar arbitration mechanisms. Some of these issues stem from the fact that the same VLAN core for packet processing had to be shared between the two Ethernet ports of our nodes in order to conserve resources so that the design may fit into a small FPGA device.

This sharing of resources can produce contention between packets originating from several sources simultaneously, such as a message emitted from the processor and a redirected TSN frame from another Ethernet port. Since the TX/RX data paths have to be shared between the ports, the use of an arbiter between transactions becomes mandatory. This could in turn increase the worst-case switching latency, which would also become dependent on the overall network utilization. On top of this, the VLAN module could also take a relatively long time to produce a tagged/untagged TSN frame. This could further contribute to increasing the packet processing delay of the core.

This is the reason why we have also proposed the implementations of several improvements on the data path of the nodes. These improvements could include the development of a *lightweight* version of the VLAN module that is faster for processing data frames, the use of a shared memory switch to mitigate the contention and arbitration issues associated with the use of crossbars, or the design of a low-latency MAC module.

In conclusion, we can state that even though WR timing can provide a more robust type of synchronization, the architectural limitations of our current TSN cores hinder us from achieving lower packet delay variation. Consequently, we will attempt to implement alternative designs in future versions of these cores to overcome the architectural issues that prevent us from taking full advantage of the high degree of determinism of WR timing for shaping traffic.

10.4.3 *Interoperability with standardized TSN equipment and gPTP timing*

We also performed an initial evaluation of the compatibility between WR and gPTP. To do this, we configured our WR-enhanced TSN nodes to exchange traffic with third-party equipment from KONTRON [51]. KONTRON is known for manufacturing a PCI-based TSN evaluation board that can be fitted into an industrial PC enclosure to assist in the deployment of TSN networks in industrial environments. This simple test with the KONTROL device allowed us to realize that, even though the data exchange of different TSN flows could proceed between our experimental nodes and the standards-compliant KONTRON device given their use of a standard Ethernet link for communication, our WR nodes could not exchange synchronization information with the KONTRON device. The WR implementation of the WR-ZEN boards from Seven Solutions relies on the *ppsi* synchronization daemon (see Sections 6.3.6 and 6.3.7), whereas the KONTRON node uses the well-known *ptp4l* [50] stack for Linux. The WR protocol needs the execution of several additional steps to those of standard PTP, of which gPTP is just another profile. This makes both protocols incompatible. As a matter of fact, both gPTP and WR share the same "raw" (*level-two*) message transmission mode, but the WR link delay model is inherently different (more accurate) than that of gPTP, and it also introduces additional operations to compensate for the master-to-slave offset and the sub-clock cycle offset differences. As a result, the accompanying implementation of *ppsi* for WR, albeit highly accurate, does not currently contemplate the use of an alternative degraded mode in its state machines for supporting a standard PTP (or gPTP) protocol.

We consider that enabling interoperability between both protocols could eventually simplify the integration with equipment from different manufacturers, thus allowing the deployment of more complex scenarios. This is one of the tasks that we will propose as continuing work after the completion of this thesis project.

Part V

Conclusions

11

THE CONCLUSIONS

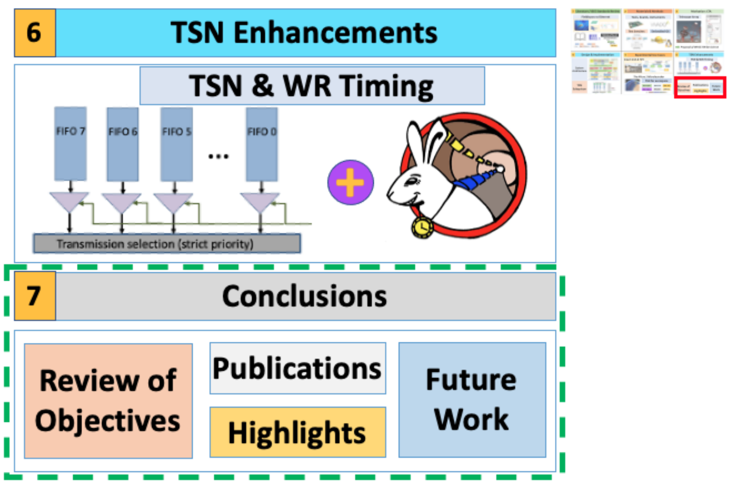


Figure 11.1
Overview of the contents of Chapter 11 with the conclusions of our research.

This chapter contains the summary of the main results of this thesis project, alongside its main objectives and the future work. Hence, we start by providing the reader with a review of the main objectives for the thesis project that we outlined in the Introduction. In the review, we also clarify the degree of compliance that we could achieve for each objective and, when needed, we indicate any possible future research that will follow up the results of a specific objective of the thesis. Once the review of the objectives is complete, we move on to stating and reviewing the main capabilities, attributes, characteristics, and highlights of the functional TSN systems that we have implemented. After that, we recount the scientific production, including journal and conference contributions, that we have published as a result of the studies and experiments that we have presented throughout the manuscript. Lastly, we enumerate the main areas of the system that could be the subject of further improvement in the section devoted to the future work.

Chapter contents

11.1 OVERVIEW OF THE CONCLUSIONS

The main result of this project is that we have managed to implement a TSN network system with a resource-conserving architecture and multiple customization options. Our initial study with the CTA telescopes provided a useful reference framework for establishing the foundation of our TSN design. Hence, our architecture can be adapted to target different FPGA devices with a moderate footprint utilization and still deliver a deterministic performance, as evidenced by the experimental use cases for the Smart Grid and avionics that we have documented. These are novel use cases for the TSN technologies that will likely gain traction in the future, as the IEEE, other major standardization committees (SAE), and leading industry forums (AVNU, TSN/A Conferences) ready the corresponding specifications for industrial and aerospace profiles for TSN. We also had the opportunity to explore advanced topics for TSN during a research visit at the Technical University of Denmark (DTU) in Köngens Lyngby (Copenhagen). During the research visit, we looked at WR for improving the determinism of TSN systems. Furthermore, we combined this with the development of device models for their use with constraint-programming, metaheuristics tools to explore the production of automated settings for our nodes.

Hence, in this chapter we will proceed to review the main objectives, claims, and assumptions that guided us during the design, implementation, and experimental stages of the project. During this review, we will clarify the degree of compliance that we reached for each main objective and, when needed, we will also indicate the future lines of work for continuing some aspects of the research that we have presented in this thesis project. Thus, we start this review process by reminding the reader of the main goals of the project in the points below.

11.2 A REVIEW OF THE OBJECTIVES OF THE PROJECT

As a reminder to the reader, we review the main objectives and motivations driving the development of the TSN-capable network nodes that we have introduced throughout this thesis study. Hence, these are outlined in the following points, where we also ascertain the degree of compliance that we have achieved for each item.

- **Objective 1**: We have shown how White Rabbit timing and the transfer of ordinary Ethernet payloads can coexist over the same physical link during our study for the small-sized telescopes in the Cherenkov Telescope Array (CTA) in Chapter 5.
  - This motivational experiment has confirmed that if we had applied our TSN enhancements for our development in CTA, we would still have been able to transmit our scientific data payload of time stamps, although we would have had a guaranteed bandwidth.
  - The use of TSN would make the link characterization for transmission integrity (section 5.6.2.2) in CTA redundant, as the sole method for guaranteeing integrity and determinism in a legacy Ethernet network is by overprovisioning its resources; i.e., we would not have had to ensure that the system would be

able to withstand a rate of up 7.14 MHz for producing time stamps, which is unrealistic for the small-sized telescopes in CTA (they required far slower rates), to be confident that the underlying Ethernet communications would provide the responsiveness (bandwidth and latency) that we required for CTA (latency lower than 100 ms) even under the worst-case conditions.

  – TSN and WR could therefore be applied to scientific facilities to provide timing and deterministic data forwarding over an open, standards-based interface.

- **Objective 2**: The primary goal of the project was to develop a working TSN system based on a collection of different FPGA IP cores. Since TSN functionalities can be added as an enhancement on top of an ordinary Ethernet network link, we fulfilled this objective by supplying the following elements:
  – We developed a set of FPGA cores that can support the use of some of the main functionalities of a TSN system. These included a VLAN module for resource reservation and routing management (802Qcc), and for traffic tagging and identification (802.1Q). We also supplied designs for the use of redundancy (802.1CB) with a TSN Dropper core, traffic shaping with preemption (802.1Qbv & 802.1Qbu), and a preemptable MAC core (802.3br).
  – In addition, we have complemented these elements with the necessary software and management APIs. This is in line with points *2.1)* and *2.2)* of the general objectives of the project, whereby we should supply methodologies for identifying different types of traffic.
- **Objective 3**: Another one of the objectives of the design was to provide a highly versatile and configurable TSN solution. Moreover, our solution had to be highly parameterizable to tailor its features and resource usage to different devices, constraints, and designs.
  – We accomplished this by designing a generic architecture for the implementation of TSN nodes, as shown in Fig. 6.2 (Chapter 6). This is the template that we have successfully applied for implementing functioning TSN systems across different devices, for multiple use cases, and under different requirements of maximum utilization.
  – In the avionics use case our proposed architecture would not take up more than half of the available resources of a Z-7030 device. This was a major design requirement that called for moderate resource usage to allow for the integration of other third-party cores alongside our TSN system. We achieved a fit into the Z-7030 SoC with an overall utilization of ∼59% LUT slice logic, ∼52% BRAM primitives, and 12.5% DSPs under the most demanding four-port configuration.
  – The flexibility of our cores allowed us to fit our architecture into the much smaller Z-7015 device in a completely different context: the experimental integration of WR timing and TSN. In this case, the application of our architectural template and the customization options of our IP cores allowed us to deploy a functioning TSN system within a very small device. We achieved overall use of ∼69% LUTs, 84 % BRAMs, and 7% DSPs. This factors in the usage of our architecture and that of the WR timing service.

- **Objective 4**: We have characterized the performance of our implementation as we explored its application to new scenarios; such as the Smart Grid (Chapter 8) or aerospace microlaunchers (Chapter 9).
  - Our experiments with the Smart Grid use case showed that the TSN system could successfully aggregate substation flows of different nature (point *4.1)*), while avoiding interference with other traffic (point *4.2)*) and enforcing a deterministic service with bounded latency and PDV dependent upon user-specified settings (point *4.3)*).
  - The test bench for the Smart Grid use case represents a novel use of TSN for industrial networks, demonstrating that is a preferrable alternative for aggregating substation flows such as GOOSE, critical messages, and best-effort messages. In this context, we verified that critical messages could be delivered more reliably and faster than with the legacy of the substation (up to ∼169 μs sooner than the legacy analog electromechanical signaling). We also verified that the bounded latency was a function of the GCL cycle time (points *4.2)* and *4.3)*).
  - We also verified that our gPTP timing service for TSN delivered synchronization on the order of the tens of nanoseconds (see ADEV plot in Fig. 8.8), which can keep up with the industrial requirements for the Smart Grid (1 μs) and aerospace (50 μs).
  - Our experiments with the aerospace avionics of the Miura 1 microlauncher have shown that COTS components can successfully be applied to aerospace and guarantee the robust delivery of critical data with the enhancements for seamless redundancy (802.1CB – point *4.4)*). Furthermore, we could provide a deterministic delivery service with low reception jitter (up to 600 ns) with the assistance of frame preemption (802.1Qbu & 802.3br).
  - We have verified that the application of different TSN cores and with their corresponding settings can heavily impact the transmission of data. Hence, the configuration of the GCL can impact the upper bound of the latency of the TSN flows, as we saw during the experiments with the Smart Grid. In the experiments for the aerospace avionics system, we realized how the use of preemption contributed to minimizing the packet delay variation, whereas the use of redundancy ensured a robust communication with a minimized possibility of data losses.
- **Objective 5**: We have explored the application of timing enhancements for TSN by replacing its standard gPTP synchronization with the more robust WR timing protocol. This could potentially allow the use of TSN for supporting the data and control planes of scientific facilities.
  - We verified that implementing a "hybrid" architecture with TSN traffic-processing blocks and WR timing was feasible on the small Z-7015 device.
  - We performed an initial assessment of the attainable determinism resulting from the application of WR timing to our TSN system (point *5.1)*). We conducted this assessment during a research visit at DTU, where we used a hardware model of the system plugged into an automated GCL-production

tool. The results of the experiments showed that the attainable determinism when WR timing was in use (Table 10.8) was on the order of the hundreds of nanoseconds, which is comparable to the behavior of a standard TSN implementation with gPTP, as we saw in our experiments for aerospace (Fig. 9.9).

  - Nonetheless, our experiments have demonstrated that TSN systems can be used as deterministic forwarding methods for high-priority traffic.
  - In spite of the lack of a substantial improvement in the determinism of the system when WR timing is used, we think that it would still make sense to deploy this hybrid architecture (TSN & WR) from the standpoint of industrial applications, as WR timing is more stable than gPTP timing, even over large topologies. This would make it possible to deploy larger networks that maintain the expected levels of sub-microsecond accuracy over long chains of devices for industrial cases (e.g., Smart Grid, industrial automation).
  - Moreover, the experiments with WR and TSN during the research visit at DTU provided some insight as to how the generation of a GCL policy can be directed towards achieving different effects. In light of our results, we suggested an upgrade path for our architecture incorporating changes for the crossbar switches and the TAS shaper (point *5.1)*). This would allow us to fully take advantage of the use of WR timing alongside TSN.

- **Objective 6**: Lastly, we have modeled the internal processing delays of our architecture. We supplied this model to the automated configuration tool from DTU during our research visit to study different strategies for producing GCLs. In addition, the results of this characterization pointed to possible bottlenecks in our data path that we will optimize during future work to reduce the latency and enhance the determinism of the system. The upgrade path for enhancing the overall determinism of the solution includes:
  - The main items flagged for improvement: the VLAN core, the TAS shaper, and the internal crossbar switches (point *6.1)*).
  - In this context, we found that the clock-domain crossing logic of the TAS shaper was one the main reasons why we could not take full advantage of WR timing: the interfaces between the WR clock domain and those of the FSMs of the shapers meant that we could only synchronize our TAS shapers with a peak-to-peak synchronization of 48.105 ns, which is far larger than the sub-nanosecond performance of WR.
  - We verified that our TSN architecture had some interoperable elements with other third-party designs, such as its VLAN-tagging routing for TSN streams, but that compatibility between WR timing and gPTP synchronization still required work (point *6.2)*).

11.3 THE CAPABILITIES OF OUR SYSTEM

As a result of the development tasks that we have carried out during this project, we have managed to build different types of TSN systems based on a common architecture template and a shared set of building blocks. The capabilities afforded by these elements allow us to deploy TSN network nodes that can fulfill the most usual requirements of a real-time system. These are outlined in the following points.

- Our nodes can guarantee the aggregation of traffic flows of different nature and apply a differentiated treatment for each traffic class. This traffic processing policy avoids interference amongst flows with different criticality and priority levels. This is one of the main features that make TSN stand out as an alternative to field buses.
- Our system includes support for the most usual features of TSN networking. Each feature is the domain of the corresponding IEEE standard, and we have supplied the necessary components to attain the basic functionality expected of a TSN system: traffic identification and VLAN-tagging (802.1Q), resource reservation on a node-by-node basis with software APIs (802.2Qcc), traffic shaping (802.1Qbv) with preemption (802.1Qbu & 802.3br), and seamless redundancy (802.1CB).
- We can target multiple applications and different networking scenarios by applying customized versions of the foregoing elements. Moreover, since they can be readily configured to selectively enable their different features, we can use them to target the multiple scenarios; such as the industrial networks for our experiments with the Smart Grid, or even the avionics of a microlauncher vehicle. These scenarios have different requirements. Thus, our experiments with the Smart Grid only needed to ensure the deterministic forwarding of data, and hence they only made of a TAS shaper for running a GCL schedule. In contrast, our experiments for aerospace required the configuration of a redundant network with seamless redundancy and frame preemption to minimize the packet delay variation.
- Our TSN architecture can be integrated with other third-party scheduling tools to produce meaningful settings driven from a set of user constraints, as shown in our experiments at DTU (Chapter 10). This streamlines the design and configuration of complex network scenarios with multiple flows and paves the way for integrating our solution with centralized network configuration utilities.
- The integration with automated GCL generation tools requires a previous modeling phase. We have produced a reference model that could consistently generate meaningful scheduling policies when used in combination with the tool from DTU. Furthermore, it allowed us to identify areas and components of our data path that will be the subject of future improvement work after the completion of this thesis project.
- Our implementation of TSN can be deployed incrementally on top of an ordinary Ethernet link. Not only does this comply with the specifications of the standard, but it also ensures that our TSN nodes are backward-compatible with legacy Ethernet equipment.
- Our design is modular enough as to allow the substitution of its gPTP timing component for the more robust WR synchronization stack. This allows the use of

our TSN system for supporting larger networks or for its application to scientific facilities. Nonetheless, on account of the limitations of our resource-conserving design, we could not achieve enhanced levels of determinism as compared to those of our experiments with industrial-level scenarios when WR timing was in use. We could still achieve a deterministic delivery of the data flows in the system, but some parts of our architecture should be redesigned to fully take advantage of the use of WR timing, as we mention in 11.5.

- Also in the context of our integration of WR timing with our traffic-processing cores of the TSN system, we verified that even though we could assemble a network of TSN-capable WR devices; we cannot build at present a hybrid network combining both gPTP and WR within the same network subsystem domain. As a rule of thumb, generic implementations of PTP are largely incompatible with the design of the WR protocol that was implemented in our prototyping WR-ZEN nodes. Enabling compatibility between these two timing stacks will be the focus of future work (see 11.5).

11.4 PUBLICATIONS

We provide a list of the main publications that have resulted from the experiments and developments of this thesis in the points below.

11.4.1 *Journal contributions*

J.1) J. Sanchez-Garrido, A. Jurado, L. Medina, R. Rodriguez, E. Ros and J. Diaz, "Digital Electrical Substation Communications Based on Deterministic Time-Sensitive Networking Over Ethernet," in IEEE Access, vol. 8, pp. 93621-93634, 2020, doi: 10.1109/ACCESS.2020.2995189.

J.2) J. Sanchez-Garrido et al., "A White Rabbit-synchronized accurate time-stamping solution for the small-sized cameras of the Cherenkov Telescope Array.," in IEEE Transactions on Instrumentation and Measurement, doi: 10.1109/TIM.2020.3013343.

J.3) *(Submitted, under review).* J. Sanchez-Garrido et al., "Resource-optimized FPGA implementation of a TSN Ethernet bus for space microlaunchers."

11.4.2 *Conference Papers*

C.1) J. Sánchez-Garrido, A. M. López-Antequera, M. Jiménez-López and J. Díaz, "Sub-nanosecond Synchronization over 1G ethernet data links using white rabbit technologies on the WR-ZEN board," 2017 40th International Conference on Telecommunications and Signal Processing (TSP), Barcelona, 2017, pp. 688-693, doi: 10.1109/TSP.2017.8076075.

C.2) J. Sanchez-Garrido, A. Jurado, J. M. Machado-Cano, M. Fuentes-Garcia, A. Sanchez-Perez, J. Cuadros-Vilchez, et al., "TSN en smart grids–comunicaciones deterministas para operaciones críticas", Proc. 6th Congreso Smart Grids, pp. 36-41, Dec. 2019, [online] Available: https://static.smartgridsinfo.es/media/2020/01/6-congreso-smart-grids-libro-comunicaciones.pdf.

11.4.3 *Lectures and conference presentations*

P.1) Jorge Sánchez Garrido, Luis Medina Valdés, Rafael Rodríguez, Javier Díaz, "Cost-optimized TSN platform for aerospace applications based on RTEMS OS," presented at TSN/A Conference, [Online], Oct. 7-8, 2020.

11.5 THE FUTURE WORK

As evidenced during our experiments with the WR-enhanced TSN node prototypes that we developed for the research visit at DTU (Chapter 10), the architecture we have devised for our nodes has succeeded at providing a low-footprint design that is flexible enough for its integration in multiple FPGA devices: we could potentially target any of the Zynq-7000 family devices starting from the small, low-cost Z-7015, which we could consider our "corner" case for a feasible system synthesis. Although this flexibility and modularity have been useful for building our system on multiple platforms for different use cases with differing caps on resource usage, we have also found that it is this very same flexibility that can be the source of some of the main limitations of our design.

The effort to preserve FPGA usage led to a compromise between performance and resources in the design of our cores. The VLAN module is a clear example of this. Hence, we designed this core with a limited number of traffic identification rules (up to *16* rules) and its frame-matching engine was based on a sequential lookup mechanism over a memory bank. Even though it was relatively flexible and could identify different types of traffic according to parameters such as their destination MAC address, IP address, ports, or protocol; it could still take up to $\sim 1\ \mu$s to appropriately identify a TSN stream and tag it with its corresponding VLAN values. This delay, as seen in Table 10.3, is one of the causes for the large overall processing latency of our TSN data path. Moreover, it could also introduce additional processing jitter and thus prevent us from enhancing the system determinism when we combine our TSN cores with WR. That is the reason why the future work awaiting this core will mainly consist of implementing several redesigns for making its packet-processing engine faster: We have envisioned the use of a content-addressable memory (CAM) [165] for much faster rule matching, potentially within two clock cycles, and increasing the number of traffic identification rules. Other changes to the VLAN core could include the implementation of a faster, "*lightweight*" version of its current architecture that uses fewer resources and produces a faster response.

We are also considering some others of our cores for enhancements; such as the TAS shaper or the internal crossbar switches. Implementing an upgrade for the TAS is paramount if we intend to increase the level of determinism of the system for data forwarding. This improvement will consist of a redesign of the different interfaces between the FSMs of the TAS and the main system bus of our data path. This redesign should therefore eliminate the use of unnecessary clock domain crossings and circumvent the issue that we detected during our characterization in the point 10.4.1.

We also claim that the crossbar switches should be replaced in an enhanced version of our architecture. The Xilinx crossbars that we have used have provided a convenient way of implementing a node with switching capabilities with promptness and agility. Nonetheless, the use of crossbars poses a hurdle for building large TSN switches, as resource consumption can quickly grow exponentially with the number of ports. This issue is compounded by the presence of arbitration issues or effects such as *head-of-line* blocking (*HOL*). HOL can prevent packets from being forwarded to an output port because their ingress path into the crossbar is currently being halted by the arbitration mechanism, even if their output port is idle at the moment. Hence, large crossbars can be a major source of indeterminism in a TSN system, and their replacement for more efficient designs such as shared memory switch should be contemplated.

Another line of future work lies in the system modeling domain of our TSN nodes. We aim to build a more accurate model of the internal architecture of our TSN system, even possibly combine it with the results from simulation tools, to build a thorough characterization of the system that could be supplied as a specific hardware profile for automating the production of GCL schedules and other configuration parameters with constraint-driven tools. An accurate characterization and modeling of our nodes could also work as a previous stage to the development of a centralized configuration system for TSN. This would involve an improvement to the configuration APIs of our nodes so that their parameters can be exposed in a canonical form to a centralized network configurator (CNC) for distributing configuration parameters from a single point of control and management, as opposed to the node-by-node approach we have used so far.

Lastly, after our initial characterization and preliminary results of the experimental integration of WR timing with our TSN nodes, we intend to revise our results to build a WR-enhanced TSN system that can fully take advantage of WR for improving the determinism of data forwarding. The upgrades we have proposed for some of our major cores should be enough to achieve better results than those we saw in Table 10.8. However, we would also like to explore the use of faster Ethernet links, such as 10 Gb Ethernet, with frame preemption, as we claim that the use of a faster link layer with finer clock granularity should make the benefits of using WR instead of gPTP more evident by comparison to the use of slower versions of the Ethernet link layer.

12 CONCLUSIONES

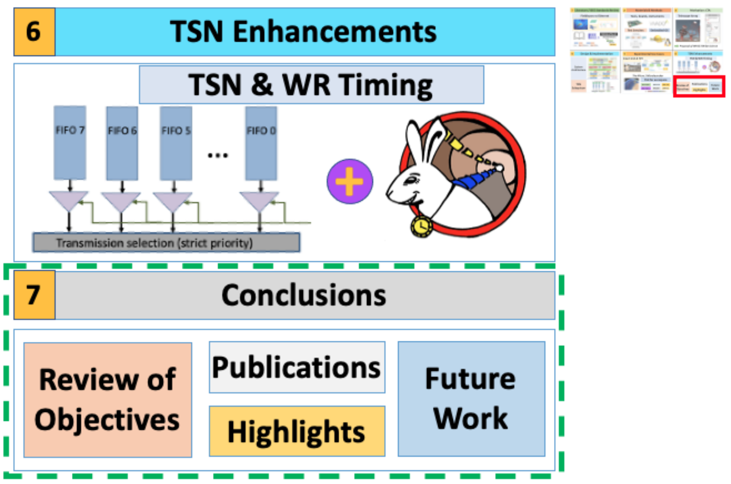


Figura 12.1
Visión general de los contenidos del Capítulo 12 de conclusiones de nuestra investigación en castellano.

Este capítulo contiene el resumen de los principales resultados de este proyecto de tesis, junto con sus principales objetivos y el trabajo futuro. Por lo tanto, se comienza proporcionando una revisión de los principales objetivos del proyecto de tesis, según los elementos que describimos en la Introducción. En esta revisión también aclaramos el grado de cumplimiento que se ha alcanzado para cada objetivo y, según se requiera, también indicamos cualquier posibles líneas futuras de investigación que retomen los resultados de objetivos específicos de la tesis. Una vez finalizada la revisión de los objetivos, pasamos a enunciar y revisar las principales capacidades, atributos, características y aspectos destacados de los sistemas TSN que hemos desarrollado. Posteriormente, repasamos la producción científica, incluyendo las contribuciones a revistas y conferencias, que se han publicado como resultado de la realización de los estudios y experimentos que hemos presentado a lo largo del trabajo de tesis. Por último, enumeramos las principales áreas del sistema que podrían ser objeto de mejoras adicionales en la sección del trabajo futuro.

Índice del Capítulo

12.1 VISIÓN GENERAL DE LAS CONCLUSIONES

El principal resultado de este proyecto es que se ha conseguido implementar un sistema de red con TSN con una arquitectura que reduce el uso de recursos y que presenta múltiples opciones de parametrización. En nuestro estudio inicial con los telescopios de CTA establecimos un marco de referencia útil y sentamos las bases para abordar nuestro diseño de TSN. Por lo tanto, hemos diseñado una arquitectura que puede adaptarse con éxito para implementarse en dispositivos FPGA de distinto tipo con un consumo de recursos moderado, al tiempo que se consigue ofrecer un rendimiento determinista, tal como se demuestra en los casos de uso experimentales para Smart Grid y aviónica que se incluyen en esta memoria. Se trata de casos de uso novedosos para las tecnologías TSN que probablemente ganarán cada vez más terreno en el futuro; tal como evidencia el trabajo del IEEE, junto con otros organismos importantes de estandarización (SAE) y los principales foros de la industria (AVNU, las conferencias de la serie TSN/A, ...); que se encuentran ultimando las especificaciones de los perfiles industrial y de aeroespacial para TSN. También tuvimos la oportunidad de explorar temas avanzados en relación con TSN durante una estancia de doctorado en la Universidad Técnica de Dinamarca (DTU) en Köngens Lyngby (Copenhague). Durante la estancia, analizamos el papel de la sincronización de WR como un medio para mejorar el determinismo de los sistemas TSN. Además, este estudio se hace en combinación con un proceso de modelado de nuestros dispositivos TSN de modo que puedan integrarse con herramientas de programación con restricciones y otras metaheurísticas para explorar la generación de configuraciones de manera automática para nuestros nodos.

En consecuencia, en este capítulo procederemos a revisar los objetivos principales, atributos, afirmaciones y supuestos que nos guiaron durante las etapas de diseño, implementación y experimentación del proyecto. Durante esta revisión, aclararemos el grado de cumplimiento que alcanzamos para cada objetivo principal y, cuando sea necesario, indicaremos también las líneas de trabajo futuro sobre las que continuaremos algunos aspectos de la investigación del proyecto de tesis. Por lo tanto, comenzamos este proceso de revisión con un recordatorio de los objetivos principales del proyecto, que se desglosan a continuación.

12.2 REVISIÓN DE LOS OBJETIVOS DEL PROYECTO

A modo de recordatorio, se procede a revisar los principales objetivos y motivaciones que han propiciado el desarrollo de los nodos para TSN que se han introducido a lo largo de la exposición de esta tesis. Por tanto, dichos objetivos aparecen detallados en los puntos siguientes, en los que también constatamos el grado de cumplimiento asociado que se ha logrado en cada caso.

- **Objetivo 1**: Hemos mostrado cómo la sincronización de White Rabbit y la transferencia de tramas Ethernet pueden coexistir sobre el mismo enlace físico con nuestro estudio para los telescopios de pequeño tamaño en el Array de Telescopios Cherenkov (CTA), tal como se muestra en el Capítulo 5.

- Éste fue un experimento con el que se establece una motivación para el desarrollo de TSN; al confirmar que de haber aplicado nuestras mejoras para TSN en el desarrollo de los módulos para CTA, la transmisión de la carga útil científica de sellos de tiempo seguiría siendo posible, pero se habría visto favorecida con la característica del ancho de banda garantizado.
- Con el uso de TSN, la caracterización que fue necesario realizar de la capacidad del enlace Ethernet para verificar la integridad de las transmisiones (sección 5.6.2.2) en los experimentos con la ZEN-CTA sería accesoria, ya que el único método del que se dispone para garantizar la integridad y el determinismo en una red Ethernet convencional es el sobredimensionamiento de sus recursos. Por lo tanto, al usar TSN, no hubiéramos tenido que asegurarnos de que el sistema era capaz de soportar una tasa de hasta 7.14 MHz para producir marcas de tiempo, lo cual no es realista para los telescopios de pequeño tamaño en CTA (necesitan tasas mucho menores), para dar una garantía de que el sistema de comunicaciones Ethernet sería capaz de dar el nivel de respuesta (ancho de banda y latencia) que sería necesario para CTA (latencia inferior a 100 ms) aún en el peor de los casos.
- En consecuencia, vemos que TSN y la sincronización WR podrían aplicarse a infraestructuras científicas para proporcionar servicios de temporización y de transmisión deterministas que operen sobre interfaces abiertas y basadas en estándares.

• **Objetivo 2**: El objetivo principal del proyecto ha consistido en desarrollar un sistema TSN funcional basado en la combinación de una serie de *cores IP* diversos para FPGA. Dado que las funcionalidades TSN pueden incorporarse de forma incremental sobre la especificación base para los enlaces Ethernet, pudimos cumplir este objetivo con el desarrollo de los elementos siguientes:

  - Desarrollamos un conjunto de *cores IP* para FPGA que proporcionan las principales funcionalidades del sistema TSN. Entre ellos se incluyen un módulo VLAN para la reserva de recursos y la gestión de enrutamiento (802.1Qcc), así como para el etiquetado e identificación del tráfico (802.1Q). También suministramos diseños para el uso de redundancia (802.1CB) con el *core* TSN Dropper, e incluimos módulos de clasificadores de tráfico dependientes de la temporización con las mejoras de interrupción de tramas – *frame preemption* – (802.1Qbv & 802.1Qbu), así como un diseño de MAC para FPGA compatible con *frame preemption* (802.3br) que prioriza la transmisión del tráfico más crítico.
  - De modo adicional, el diseño de los elementos anteriores se complementa con todos los componentes software y de gestión (APIs) necesarios. Esto está en línea con los puntos *2.1)* y *2.2)* de los objetivos generales del proyecto, según los cuales deben especificarse metodologías con las que se identifiquen y gestionen diferentes tipos de tráfico.

• **Objetivo 3**: Otro de los objetivos del diseño fue proporcionar una solución TSN altamente versátil y configurable. Además, nuestra solución tenía que ser altamente parametrizable para adaptar sus características y uso de recursos a diferentes dispositivos, restricciones y diseños.

  - Esto se logró con el diseño de una arquitectura genérica para la implementación de los nodos TSN, como se muestra en la Fig. 6.2 (Capítulo 6). Esta arquitectura supone la plantilla de diseño que hemos aplicado con éxito para implementar sistemas TSN funcionales para distintos tipos de dispositivos, en múltiples casos de uso, y bajo diferentes requisitos de máxima utilización.
  - En el caso de uso de aviónica, la arquitectura que propusimos no ocupa más de la mitad de los recursos disponibles de un dispositivo Z-7030. Éste era un requisito de diseño importante en este caso, ya que se imponía un uso moderado de recursos que permitiese que nuestra solución TSN pudiera integrarse junto con otros *cores IP* para FPGA de terceros. De este modo, se consiguió implementar el diseño en el SoC Z-7030 con una utilización general del ∼ 59 % de lógica general (LUT), ∼ 52 % de las primitivas BRAM, y un 12.5 % de los DSPs cuando se parametrizaba la arquitectura con la configuración más exigente de cuatro puertos Ethernet.
  - La alta flexibilidad de nuestros módulos para FPGA también nos permitió "encajar" el diseño en el dispositivo Z-7015, que es mucho más pequeño, en un contexto completamente diferente al del punto anterior: la integración experimental de la sincronización de WR con la red TSN. En este caso, al aplicar nuestra arquitectura maestra para el diseño de los nodos en combinación con las opciones de parametrización de los *cores IP*, pudimos implementar un sistema TSN funcional en un dispositivo muy pequeño. Se logró un uso general del ∼ 69 % de LUTs, 84 % de BRAMs, y un 7 % de DSPs. Estos resultados tienen en cuenta el consumo de nuestra arquitectura junto con el de los módulos para la sincronización de WR.

- **Objetivo 4**: Hemos caracterizado el rendimiento de nuestra implementación conforme íbamos explorando su aplicación en nuevos escenarios; tales como en las Smart Grids (Capítulo 8) o para los microlanzadores en aeroespacial (Capítulo 9).
  - Nuestros experimentos con el caso de uso de Smart Grid han demostrado con éxito que la aplicación de un sistema TSN puede servir para agregar todos los flujos de distinta naturaleza que están presentes en la subestación (punto *4.1)*), al tiempo que se evita la interferencia con otros tipos de tráfico (punto *4.2)*), y se hace cumplir en estos casos un servicio determinista con latencia acotada y PDV dependientes de la configuración suministrada por el usuario del sistema (punto *4.3)*).
  - El banco de pruebas para el caso de uso de Smart Grid representa un uso novedoso de TSN en redes industriales, demostrando que supone una alternativa preferible para agregar los flujos de datos presentes en subestaciones eléctricas, tales como GOOSE, mensajes críticos y mensajes de baja prioridad (*best-effort*) con entrega de mejor esfuerzo. En este contexto, verificamos que es posible hacer la entrega de los mensajes más críticos de forma más rápida, robusta, y fiable que con el equipamiento convencional de la subestación (hasta ∼ 169 μs antes que los mecanismos electromecánicos de señalización tradicionales). También verificamos que era posible controlar el valor de la latencia extremo a extremo acotada en función del tiempo de ciclo que se especificara en el definición del GCL (puntos *4.2)* y *4.3)*).

  - También verificamos que nuestro servicio de temporización gPTP para TSN era capaz de proporcionar un nivel de sincronización del orden de las decenas de nanosegundos (véase el gráfico de la ADEV en la Fig. 8.8), que es capaz de cumplir con los requisitos esperados en sistemas industriales para Smart Grid (1 μs) y para aeroespacial (50 μs).
  - En nuestros experimentos con la aviónica para aeroespacial del microlanzador Miura 1, hemos demostrado que es factible aplicar componentes COTS para desarrollos de espacio (y aeroespacial en general) con las garantías de robustez y fiabilidad en la entrega de datos críticos que proporcionan las mejoras de redundancia para TSN ("seamless redundancy" – 802.1CB - punto *4.4)*). Además, vimos que el determinismo en estos casos puede mejorarse enormemente con un sistema de entrega determinista con bajo *jitter* en la recepción (hasta 600 ns) con ayuda del componente de interrupción de tramas de baja prioridad de "*frame preemption*" (802.1Qbu & 802.3br).
  - Hemos verificado que la aplicación de diferentes núcleos TSN junto con el uso de parámetros de configuración distintos puede impactar enormemente la transmisión de datos. Por lo tanto, la configuración que se suministre para el GCL puede afectar directamente al límite superior de la latencia de los flujos de TSN, tal como se verificó durante los experimentos en Smart Grid. En los experimentos para el sistema de aviónica en aeroespacial, pudimos constatar cómo el empleo de los mecanismos de interrupción de tramas menos prioritarias (*frame preemption*) podía contribuir a minimizar la variación del retardo de los paquetes, mientras que el uso de la redundancia aseguraba al mismo tiempo que se establecía una comunicación robusta en la que se minimizaba, además, la probabilidad de pérdida de datos.

- **Objetivo 5**: Hemos explorado la aplicación de mejoras en el sistema de sincronización para TSN al reemplazar su implementación por defecto de gPTP, tal como contempla el estándar, con el protocolo de sincronización WR, que proporciona mayor robustez y rendimiento. Esto podría permitir el uso de TSN como la tecnología de base de los planos de control y datos en las infraestructuras científicas.
  - Verificamos que implementar esta arquitectura "híbrida" con bloques de WR y de procesado TSN era factible, incluso usando el dispositivo Z-7015 de recursos reducidos.
  - Realizamos una evaluación inicial del determinismo alcanzable como resultado de la inclusión de la sincronización WR en nuestro sistema TSN (punto *5.1)*). Esta evaluación se realizó durante una estancia de doctorado en DTU, para la cual usamos un modelo del hardware del sistema en combinación con una herramienta de generación de configuraciones GCL automatizada. Los resultados de los experimentos nos han mostrado que el determinismo alcanzable cuando se usaba la sincronizacióin de WR (véase la Tabla 10.8) era del orden de los cientos de nanosegundos, que es comparable al comportamiento de una implementación estándar de TSN con gPTP; tal se vio en los experimentos para el caso de aeroespacial (véase la Fig. 9.9).

- Sin embargo, sí que hemos demostrado con nuestros experimentos que los sistemas TSN se pueden usar como un método para la transmisión determinista del tráfico de alta prioridad.
- A pesar de no poder observar una mejora sustancial en el determinismo del sistema cuando se utiliza la sincronización de WR, creemos que todavía tendría sentido implementar esta arquitectura híbrida (TSN & WR) desde el punto de vista de las aplicaciones industriales, dado que la sincronización de WR puede comportarse de forma más estable y determinista que gPTP, incluso en cadenas largas. Esto permitiría desplegar redes más grandes que mantuvieran los niveles esperados de precisión en rangos inferiores al microsegundo cuando se emplean topologías con cadenas muy extensas de dispositivos, como suele el caso en muchas aplicaciones industriales (por ejemplo, Smart Grid o automatización industrial).
- Además, gracias a la realización de los experimentos con WR y TSN de la estancia de doctorado en DTU, hemos llegado a la conclusión de que existen formas diferentes de ajustar la generación de las configuraciones de los GCLs para conseguir efectos distintos en el tipo de tratamiento que recibe el tráfico. Tomando en consideración nuestros resultados en estas pruebas, hemos sugerido una hoja de ruta para actualizar nuestra arquitectura de modo que incorpore cambios para los elementos conmutadores de barras cruzadas y el clasificador de tráfico TAS (punto *5.1*)). Esto nos permitiría aprovechar al máximo el uso de la sincronización WR junto con TSN para mejorar su determinismo.

• **Objetivo 6**: Por último, hemos modelado los retardos de procesamiento interno de nuestra arquitectura, para pasar a combinar este modelo con las herramientas de configuración automatizadas de DTU durante la estancia de doctorado y, de este modo, realizar el estudio de diferentes estrategias para producir GCLs. Además, los resultados de esta caracterización señalaron posibles cuellos de botella en el camino de datos de nuestra arquitectura. Éstos serán los principales puntos que trabajaremos para optimizar durante el trabajo futuro con el fin de reducir la latencia y mejorar el determinismo del sistema. Esta hoja de ruta de trabajo futuro y actualizaciones contempla los elementos siguientes:

  - La realización de mejoras en los principales elementos que hemos identificado que necesitan actualizarse: el módulo de VLAN, el clasificador de tráfico dependiente de la temporización (TAS), y los conmutadores internos de barras cruzadas (punto *6.1*)).
  - En este contexto, se determinó que los elementos lógicos que implementan los cruces de reloj entre los distintos dominios del clasificador de TSN (TAS) eran los principales responsables de impedir que se pudiera sacar provecho al máximo del rendimiento de la sincronización de WR. Esto se debía a que las interfaces entre el dominio de reloj de WR y las máquinas de estado (FSM) del clasificador TAS únicamente permitían que los distintos TAS presentes en la red se pudieran sincronizar entre sí con una diferencia pico a pico de 48.105 ns, que es mucho peor que el rendimiento que es capaz de proporcionar WR,

el cual es mucho más preciso y capaz de alcanzar un rendimiento por debajo del nanosegundo.

– Hemos comprobado que nuestra arquitectura TSN era capaz de ser interoperable con algunos elementos y diseños de terceros, tales como el mecanismo de etiquetado y encapsulado del VLAN para flujos TSN (*streams*). No obstante, se hace necesario continuar trabajando en mejorar la compatibilidad entre la sincronización de WR y temporización gPTP de TSN (punto *6.2)*).

12.3 capacidades del sistema

Como resultado de las tareas de desarrollo que hemos llevado a cabo durante este proyecto, hemos logrado construir distintas variantes de sistemas TSN siguiendo un patrón maestro para su arquitectura arquitectura junto con una serie de módulos funcionales básicos que se han compartido en todos nuestros diseños. Las capacidades que brindan estos elementos nos permiten implementar nodos de red TSN que pueden cumplir con los requisitos más habituales de los sistemas de tiempo real. A modo de resumen, proporcionamos una recapitulación de las principales capacidades del sistema en los puntos siguientes.

- Nuestros nodos pueden garantizar la agregación de flujos de tráfico de diferente naturaleza y proporcionar un nivel de tratamiento y procesado diferenciado para cada clase de tráfico. De este modo, al aplicar distintas políticas para el procesado de los distintos flujos, la red evita la interferencia entre flujos con niveles de criticidad y prioridad distintos. Ésta es una de las principales características que hacen que TSN se destaque como una alternativa a los buses de campo.
- Nuestro sistema incluye soporte para las funciones más habituales de las redes TSN. En este tipo de sistemas, cada característica funcional se especifica dentro del subestándar del IEEE correspondiente. De este modo, hemos implementado los componentes del estándar necesarios para proporcionar las principales funcionalidades de un sistema TSN: identificación de tráfico y etiquetado de VLAN (802.1Q), reserva de recursos de nodo a nodo mediante APIs de configuración software (802.2Qcc), clasificación de tráfico (802.1Qbv) con interrupción de tramas de baja prioridad (802.1Qbu & 802.3br), y transmisiones redundantes (802.1CB).
- Se hace posible abordar con éxito aplicaciones diversas y los requisitos de múltiples escenarios de red mediante la generación de versiones parametrizadas a medida de nuestra arquitectura (combinando elementos de los puntos anteriores). Además, dado que pueden configurarse fácilmente y permiten la habilitación selectiva de sus diferentes prestaciones, podemos producir diseños especializados que se adapten a escenarios distintos; como las redes industriales para nuestros experimentos con las Smart Grids, o incluso la aviónica de un vehículo microlanzador. Estos escenarios tienen diferentes requisitos. Por lo tanto, nuestros experimentos con Smart Grid solo necesitaban asegurar la transmisión determinista de datos y, de este modo, únicamente requerían del uso de un clasificador de tráfico TAS que ejecutase un GCL determinado para procesar los flujos de red. Por el contrario, nuestros experimentos del caso de aeroespacial requirieron la configuración de

una red redundante (con el componente de redundancia de TSN) y del uso del mecanismo de interrupción de tramas de baja prioridad para minimizar la PDV.

- Nuestra arquitectura TSN se puede integrar con herramientas automáticas de generación de configuraciones de terceros; que produzcan parámetros de configuración significativos en base a una serie de requisitos y condicionantes que defina el usuario del sistema, tal como se muestra en nuestros experimentos en DTU (Capítulo 10). Esto puede agilizar el diseño y la configuración de escenarios de red complejos con múltiples flujos, además de allanar el camino para integrar nuestra solución con sistemas de configuración de red centralizados.
- La integración con herramientas de generación de GCLs automatizadas necesita de una fase de modelado previa. Para ello, hemos generado un modelo de referencia que puede usarse para producir configuraciones y políticas significativas de procesado de tráfico de manera consistente al combinarse con la herramienta de DTU. Además, como fruto de esta integración, pudimos identificar áreas y componentes de nuestro camino de datos que son susceptibles de mejora en el trabajo futuro que continúe la investigación de esta tesis.
- Nuestra implementación de TSN se puede añadir de forma incremental sobre cualquier tipo de enlace Ethernet convencional. Esto no solo cumple con las especificaciones del estándar, sino que también asegura que nuestros nodos TSN sean compatibles con equipamiento que incorpore versiones convencionales del estándar Ethernet.
- Nuestro diseño es lo suficientemente modular como para permitir la sustitución de su componente de sincronizacióngPTP por el sistema WR, que proporciona una sincronización más robusta. Esto permite que se emplee nuestro sistema TSN para dar soporte a redes más amplias o para su aplicación a infraestructura científica. No obstante, dadas las limitaciones en torno al uso reducido de recursos que se han impuesto sobre nuestro diseño, no pudimos lograr niveles de mejoría en el determinismo con la sincronización de WR que superasen a aquéllos de nuestros experimentos en escenarios industriales o en aviónica. En estos casos, verificamos que seguía siendo posible lograr una entrega determinista de los flujos de datos en el sistema, pero algunas partes de nuestra arquitectura necesitaban de un rediseño para que pudieran aprovechar completamente el uso de la sincronización WR para mejorar el determinismo del sistema, tal como se menciona en 12.5.
- Asimismo, en el contexto de nuestra integración de la sincronización de WR con nuestros módulos FPGA para TSN, pudimos verificar que, a pesar de que se ha podido realizar el montaje de una red de varios dispositivos TSN mejorados con WR, no ha sido posible por el momento montar una red híbrida en la que se combinen tanto dispositivos gPTP como con nodos WR dentro del mismo subsistema de red. En general, llegamos a comprobar que las implementaciones genéricas de PTP son incompatibles en gran medida con el diseño del protocolo WR, que se encuentra presente en nuestros nodos WR-ZEN de prototipado. Habilitar la compatibilidad entre estos dos protocolos de sincronización será uno de los principales objetivos del trabajo futuro (véase 12.5).

12.4 PUBLICACIONES

A continuación, proporcionamos una lista de las principales publicaciones en sus diferentes modalidades que la realización de los experimentos y trabajos de desarrollo de esta tesis ha dado como resultado. Estos trabajos aparecen referidos en los puntos a continuación.

12.4.1 *Publicaciones de revista*

J.1) J. Sanchez-Garrido, A. Jurado, L. Medina, R. Rodriguez, E. Ros and J. Diaz, "Digital Electrical Substation Communications Based on Deterministic Time-Sensitive Networking Over Ethernet," in IEEE Access, vol. 8, pp. 93621-93634, 2020, doi: 10.1109/ACCESS.2020.2995189.

J.2) J. Sanchez-Garrido et al., "A White Rabbit-synchronized accurate time-stamping solution for the small-sized cameras of the Cherenkov Telescope Array.," in IEEE Transactions on Instrumentation and Measurement, doi: 10.1109/TIM.2020.3013343.

J.3) *(Enviado a revista, actualmente en proceso de revisión)*. J. Sanchez-Garrido et al., "Resource-optimized FPGA implementation of a TSN Ethernet bus for space microlaunchers."

12.4.2 *Publicaciones en conferencias*

C.1) J. Sánchez-Garrido, A. M. López-Antequera, M. Jiménez-López and J. Díaz, "Subnanosecond Synchronization over 1G ethernet data links using white rabbit technologies on the WR-ZEN board," 2017 40th International Conference on Telecommunications and Signal Processing (TSP), Barcelona, 2017, pp. 688-693, doi: 10.1109/TSP.2017.8076075.

C.2) J. Sanchez-Garrido, A. Jurado, J. M. Machado-Cano, M. Fuentes-Garcia, A. Sanchez-Perez, J. Cuadros-Vilchez, et al., "TSN en smart grids–comunicaciones deterministas para operaciones críticas", Proc. 6th Congreso Smart Grids, pp. 36-41, Dec. 2019, [online] Available: `https://static.smartgridsinfo.es/media/2020/01/6-congreso-smart-grids-libro-comunicaciones.pdf`.

12.4.3 *Presentaciones y charlas en conferencias*

P.1) Jorge Sánchez Garrido, Luis Medina Valdés, Rafael Rodríguez, Javier Díaz, "Cost-optimized TSN platform for aerospace applications based on RTEMS OS," presented at TSN/A Conference, [Online], Oct. 7-8, 2020.

12.5 TRABAJO FUTURO

Tal como se ha demostrado con nuestros experimentos con los prototipos de nodo TSN mejorados con WR de la estancia de doctorado en DTU (Capítulo 10), la arquitectura que hemos diseñado para nuestros nodos ha sido capaz de proporcionar con éxito un diseño con un consumo moderado de recursos que es lo suficientemente flexible como para permitir su integración en distintos tipos de dispositivos FPGA; esto es, la flexibilidad es tal que potencialmente podríamos implementar nuestro diseño en cualquiera de los dispositivos de la familia Zynq-7000, tomando el dispositivo Z-7015 de tamaño reducido y bajo coste como punto de partida. Éste sería el caso extremo con los requisitos más exigentes a partir del cual sería factible realizar la síntesis del sistema. Aunque esta flexibilidad y modularidad han sido útiles para implementar nuestro sistema en múltiples plataformas en casos de uso distintos en los que se imponían límites en el uso de recursos diferentes, también hemos podido constatar que esta misma flexibilidad es precisamente la que puede ser la fuente de algunas de las principales limitaciones del rendimiento del diseño.

Como hemos realizado un esfuerzo importante en preservar y reducir el uso de FPGA, en nuestro diseño llegamos a establecer un equilibrio entre el máximo rendimiento aprovechable y la cantidad de recursos asociada con la implementación de cada uno de nuestros *cores IP*. El módulo de VLAN representa un ejemplo claro. De este modo, su diseño usa un número limitado de reglas de identificación de tráfico (hasta 16 reglas distintas), y su motor de comparación, que identifica las tramas Ethernet como pertenecientes a distintas clases de tráfico, está basado en el uso de un mecanismo de búsqueda secuencial en una zona de bancos de memoria. Aunque este mecanismo es relativamente flexible y permite identificar distintos tipos de tráfico atendiendo a parámetros tales como la dirección MAC de destino, dirección IP, el número de puerto o de protocolo; durante el modelado caracterizamos que el retardo que introduce esta aproximación podría ser del orden de hasta $\sim 1\ \mu$s antes de que el bloque sea capaz de identificar a un flujo TSN correctamente y de aplicarle el encapsulamiento VLAN correspondiente. Este retraso, como se ve en la Tabla 10.3, es una de las principales causas de la gran latencia en el procesado de los paquetes en el camino de datos de nuestra arquitectura TSN. Además, también podría contribuir a introducir *jitter* adicional en la transmisión y, de este modo, evitar que se pueda mejorar el determinismo del sistema cuando combinamos nuestros núcleos TSN con la sincronización de WR. Ésta es una de las razones principales por las que el trabajo futuro que se emprenderá sobre el módulo de VLAN versará principalmente en rediseñar su motor de procesamiento y comparación de paquetes para hacerlo más rápido. En esta línea, hemos previsto el uso de una memoria direccionable por contenido (CAM) [165] que permita la detección de clases de tráfico con mayor rapidez, posiblemente en menos de dos ciclos de reloj, y que ,además, facilite la definición de un mayor número de reglas para detectar aún más flujos TSN. Otros cambios en el núcleo de VLAN podrían orientarse por la vertiente de proporcionar una versión más *"liviana"* y rápida del mismo con su arquitectura actual, pero que use menos recursos y que mejore su velocidad de respuesta.

También se considera la aplicación de mejoras en el resto de nuestros módulos para TSN; tales como el clasificador TAS o los conmutadores internos de barras cruzadas. Implementar una mejora para el módulo del TAS resultaría esencial para mejorar el nivel de determinismo que presenta el sistema al transmitir datos. Esta mejora consistirá en un rediseño de las interfaces entre las máquinas de estado (FSMs) del TAS con el bus principal del sistema en el camino de datos. Por lo tanto, el principal objetivo de esta mejora debería ser evitar el uso de lógica de cambio de dominio de reloj innecesaria y, de esta forma, eludir el problema que se detectó durante la caracterización del TAS en el punto 10.4.1.

Asimismo, también afirmamos que los conmutadores de barras cruzadas deberían reemplazarse totalmente en una versión mejorada de nuestra arquitectura. Los conmutadores de barras cruzadas de Xilinx son un bloque estructural importante que nos ha permitido diseñar nuestra arquitectura de forma ágil y cómoda para cumplir con las funcionalidades esperadas en los objetivos del trabajo. No obstante, el uso estos elementos da lugar a limitaciones importantes para diseñar conmutadores (*switches*) de gran tamaño con un elevado número de puertos, ya que la implementación en FPGA de *switches* de barras cruzadas puede disparar exponencialmente el consumo de recursos cuando se usan parametrizaciones para un número elevado de puertos. El problema se agrava todavía más en estos casos con los efectos de los mecanismos de arbitraje, como el bloqueo de cabeza de línea (*head-of-line blocking* - *HOL*). Este efecto es un problema típico en estructuras de barras cruzadas. De este modo, puede llegar a causar el bloqueo momentáneo en la transmisión de ciertos paquetes hacia su puerto de salida, aunque éste se encuentre inactivo, porque su ruta de entrada al conmutador está detenida actualmente por acción del arbitraje entre otros paquetes anteriores destinados a otro puerto. Por lo tanto, el uso de conmutadores de barras cruzadas puede ser una fuente importante de indeterminismo en un sistema TSN. En consecuencia, con el fin de producir un sistema con mayor rendimiento, se debe contemplar su reemplazo por otros diseños más eficientes, como los conmutadores de memoria compartida, que eliminan este problema.

Otra línea de trabajo futuro radica en el área del modelado de nuestros nodos TSN. En este contexto, tenemos como objetivo producir un modelo más preciso de la arquitectura interna de nuestro sistema TSN, e incluso posiblemente combinarlo con los resultados de herramientas de simulación para aumentar su precisión. De este modo, podríamos construir una caracterización completa del sistema que podría proporcionarse como un perfil de hardware específico del mismo a herramientas automáticas de generación de los GCLs y de los otros parámetros de configuración del sistema, al tiempo que tengan en cuenta las restricciones y condicionantes especificados por el usuario de la red. Estas caracterizaciones y modelados precisos también podrían servir como una etapa previa al desarrollo de un sistema de configuración centralizado para TSN. Este paso conllevaría una mejora en las APIs de configuración de nuestros nodos para que sus parámetros se puedan exponer de forma canónica a un elemento central de gestión de red (CNC). El CNC distribuiría en este caso los parámetros de configuración específicos de cada nodo como un único punto de control y gestión, en contraposición a la filosofía de programación individual nodo a nodo que hemos venido empleando hasta ahora.

Por último, tras haber realizado una caracterización inicial y haber obtenido resultados preliminares de la integración entre WR y nuestros nodos TSN, tenemos la intención de revisar nuestros resultados para construir sistema TSN con WR en el que se pueda aprovechar de manera completa el uso de la nueva sincronización para mejorar el determinismo en las comunicaciones. Las mejoras que hemos propuesto para los principales módulos FPGA del sistema deberían permitir obtener resultados bastante mejores que los que presentamos en la Tabla 10.8. Sin embargo, también nos gustaría explorar el uso de enlaces Ethernet más rápidos, como Ethernet a 10 Gb combinado con el mecanismo de interrupción de tramas menos prioritarias (*frame preemption*), ya que postulamos que con una capa de enlace Ethernet a mayor velocidad y, por lo tanto, con una mayor granularidad de reloj, debería hacer que las ventajas de usar WR en sustitución de gPTP se hagan patentes en comparación con el uso de versiones de menor ancho de banda de Ethernet.

Parte VI

Annexes and Bibliography

A ZEN-CTA SOFTWARE ENVIRONMENT

This section shows part of the supplementary material from [1]. We use this appendix to provide the reader with an overview of the software ecosystem of the ZEN-CTA node. Since this node is a customized version of the WR-ZEN boards [8] that we have extensively used for prototyping our TSN solutions, the features and characteristics that we present in this section are also applicable to the WR-ZEN boards that we used during the thesis project. Moreover, some of the concepts and features that we outline in the appendix have guided our development efforts in the areas of Linux kernel configuration and driver development, integration of new user-level applications, and use of third-party system services. Hence, its inclusion with the manuscript is illustrative and should help the reader get a broader overview of the typical development workflow in a generic embedded environment, like those of the ZEN-CTA or the WR-ZEN nodes.

Chapter contents

A.1 THE SOFTWARE ENVIRONMENT OF THE ZEN-CTA NODES

White Rabbit Zynq (WRZ) devices, including the ZEN-CTA node, feature a dual-core embedded processing system (a microprocessor) that allows the running of several software tasks. In this context, a complete Linux ecosystem has been generated using embedded tools such as Buildroot [83]. This tool can be configured to create a specific image containing different applications, a Linux kernel and a device bootloader. For the Linux Operating System (OS) and bootloader, we use the Linux kernel and U-boot from Xilinx GitHub repositories ([200] and [201]). Upon determining the OS and bootloader for the WRZ device, specific software development tasks have been performed to design and implement user space tools and kernel drivers for use in CTA [99]. The WRZ software ecosystem is shown in Fig. A.1 and the main blocks are briefly discussed in the following lines. Additionally, a complete description can be found in [132].

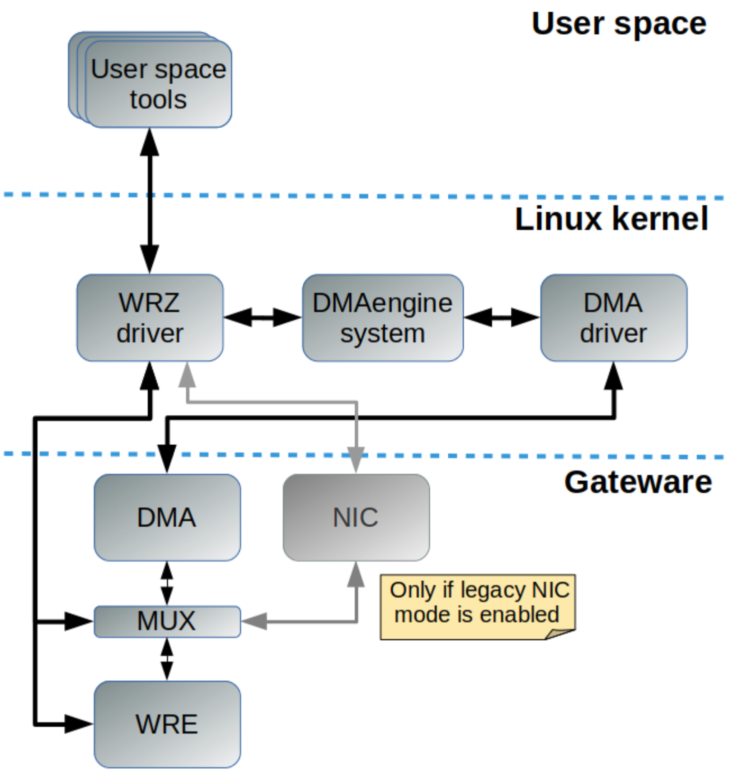


Figure A.1
Diagram of the Software Ecosystem for the ZEN-CTA node, based on the generic environment of the WR-ZEN family of nodes. © **2020 IEEE.**

A.2 USER SPACE TOOLS

Some utilities have been implemented to program the FPGA device and access the hardware resources of the platform.

- **zen-ts**. This configures the TDC module inside the FPGA to retrieve time-stamp data from new incoming events using either a software-assisted or a coprocessor-enhanced mode: The former sends time stamps from the TDC to the ARM processing system, where they are transformed into a human-readable format and forwarded over the DMA engine (Section 5.5.1); whereas the latter mode uses an FPGA coprocessor (Section 5.5.2) to transmit raw time-stamp data directly over the new data path without processor intervention.

– **wrz-cl**. This programs the soft-processor inside the FPGA device. This soft-processor is in charge of implementing the WR protocol for the high accuracy timing synchronization.

– **wrz-date**. This reads/writes the time counters from the time baseline core inside the FPGA device. It is especially useful if a network time protocol (NTP) server is used as time reference.

– **wrz-fwloader.sh**. This is responsible for programming the FPGA device with a specific gateware bitstream.

– **wrz-mem**. This allows other applications to access hardware registers of different Intellectual Property (IP) blocks inside the FPGA gateware.

– **wrz-vuart**. This acts as virtual universal asynchronous receiver-transmitter (VUART) terminal between the main CPU and the soft-processor inside the FPGA gateware.

A.3 KERNEL DRIVERS

In addition to the userspace tools, a kernel driver has been implemented to handle the WRZ device. This module is responsible for accessing hardware resources by reading/writing relevant information to the registers in the IP blocks. Moreover, a Linux network driver has been developed to expose the optical fiber ports of the WR End-Points (WRE) as conventional Linux network interfaces for the rest of the applications in the user space domain. WRZ can configure the underlying hardware to use a DMA module for high-bandwidth data transfers (Section 5.5) or switch over (MUX) to the legacy NIC [131], which is disabled in our implementation. With regards to the DMA mode, the WRZ driver uses the *Linux DMAengine* Application Programming Interface (API) to request DMA descriptors for the Xilinx AXI DMA driver. Consequently, the WRZ driver presents a generic implementation that handles the DMA module through a kernel-maintained abstraction layer.

A.4 SYSTEM SERVICES

The system implements a lightweight Linux environment with several useful services; such as a dynamic host configuration protocol (DHCP) client for acquiring local network settings, an NTP client for retrieving Internet time, or a secure shell (SSH) server for providing management shell access to the end-user. Additional components, such as an Object Linking and Embedding for Process Control: Unified Architecture (OPC-UA) server for slow control could also be installed from resource repositories if needed.

B

LINUX KERNEL-LEVEL SUPPORT FOR NETWORK DRIVERS

This section shows part of the supplementary material from [1]. We use this appendix to introduce some advanced topics on network driver development and optimization that we encountered initially during our study and development efforts with the ZEN-CTA board. Hence, we discuss some of the main considerations applicable to the optimization of our network drivers for the ZEN-CTA, which we supplied as an upgrade over the legacy WR-NIC [131]. These optimizations allowed our network driver to take advantage of the available 1-Gb/s link bandwidth by applying a series of customizations and configuration options to our code. These considerations are generally common for network driver development and, thus, we introduce them in the appendix to provide the reader with greater context on the usual engineering challenges that are normally associated with the implementation of kernel modules for handling interfacing with FPGA-based network interfaces or other generic peripherals.

Chapter contents

B.1 LINUX KERNEL-LEVEL SUPPORT FOR THE NEW DMA-BASED UPGRADE

The implementation of the DMA-based upgrade for the WR-compatible interfaces of the ZEN-CTA requires the development of the necessary kernel-level support to integrate their new data path with the Linux networking stack. As explained in Appendix A, a kernel driver has been used to expose the WR ports of the WRPC as regular Ethernet interfaces for the user applications and the Linux OS of the Zynq-7000 processing system. Hence, this allows the convergence over the same physical link of deterministic WR messages from the WRPC and ordinary data traffic from user applications in the processing system of the Zynq-7000 SoC transmitted over the high-speed DMA path developed in this design.

We determined that the highest achievable throughput performance of the newly developed driver was highly dependent on a number of configuration parameters; such as the number of buffer descriptors reserved for handling transmission or reception transactions from the DMA module, the amount of allocated RX/TX memory for the UDP and transmission control protocol (TCP) kernel components, disabling the use of the TCP SACK (*Selective Acknowledgements*) mechanism, or even the Processor Affinity assigned to each user space task transmitting data over the WR-compatible network interfaces. As a result, a number of kernel configuration parameters were modified iteratively using simple *sysctl* commands in accordance with some of the recommendations explained in [202]. These parameters are summarized in Table B.1.

Configuration Parameters	
txqueuelen	5000
netdev_backlog	5000
wmem_max, rmem_max	212992 B
tcp_mem	"285108 380145 570216" B
tcp_sack	*Disabled*
Processor Affinity	Adjusted Processor Affinity value to the second core out of the two cores available at the ARM-based processing system **for the user space applications producing TX transactions** on the WR-compatible Ethernet interfaces
Optimization Flags	Enabled optimization flags for compiling the necessary kernel modules implementing the WR-compatible network interfaces (e.g.,"***-Ofast***"), as well as the Embedded Linux environment running on the processing system.

Table B.1
Overview of the Linux kernel configuration parameters used for fine-tuning the highest attainable throughput of the DMA-based, WR-compatible network interfaces. © **2020 IEEE**.

The foregoing points summarize the main optimization strategies used to enhance the stability and throughput of the kernel modules handling the operation of the DMA-based, WR-capable Ethernet interfaces. Further optimizations could include customizing the kernel with TCP modules tailored to support high-speed links [203], or fine-tuning the number of buffer descriptors allocated to the Xilinx DMA module for processing transactions. This latter approach is fairly effective and could achieve an optimum throughput of 426 Mbps (TX) and 564 Mbps (RX) when 55 and 80 buffer descriptors were reserved for TX and RX transactions, respectively. The attainable bandwidth is ultimately limited by the CPU clocking rate of the ZEN-CTA node, but is nonetheless in line with the performance of the Xilinx 1G Ethernet drivers for Zynq-7000 devices [151]. Even though this is a significant improvement over the original WR-NIC, the transmission of TDC time stamps from user applications could only at best use approximately half of the capacity of the 1-Gigabit Ethernet links.

BIBLIOGRAPHY

[1] J. Sanchez-Garrido, A. Jurado, M. Jimenez-Lopez, A. K. Balzer, H. Prokoph, M. Stephan, D. Berge, M. Rodriguez-Alvarez, and J. Diaz. "A White Rabbit-synchronized accurate time-stamping solution for the small-sized cameras of the Cherenkov Telescope Array." In: *IEEE Transactions on Instrumentation and Measurement* (2020). © 2020 IEEE. Reprinted, with permission, from J. Sanchez-Garrido et al., "A White Rabbit-synchronized accurate time-stamping solution for the small-sized cameras of the Cherenkov Telescope Array", IEEE Transactions on Instrumentation and Measurement, July 2020, pp. 1–1. DOI: 10.1109/TIM.2020.3013343.

[2] D. Jansen and H. Buttner. "Real-time ethernet the EtherCAT solution". In: *Computing Control Engineering Journal* 15.1 (2004), pp. 16–21. DOI: 10.1049/cce:20040104.

[3] *ISO 11898-1:2015. Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling*. ISO standards. URL: https://www.iso.org/standard/63648.html.

[4] *IEC 61158-1:2019. Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*. IEC standards. URL: https://webstore.iec.ch/publication/59890.

[5] *ISO 17458-1:2013. Road vehicles – FlexRay communications system – Part 1: General information and use case definition*. ISO standards. URL: https://www.iso.org/standard/59804.html.

[6] "IEEE Standard for Local and Metropolitan Area Networks - Timing and Synchronization for Time-Sensitive Applications in Bridged Local Area Networks". In: *IEEE Std 802.1AS-2011* (2011), pp. 1–292. DOI: 10.1109/IEEESTD.2011.5741898.

[7] *DS190. Zynq-7000 SoC Data Sheet: Overview*. Xilinx. July 2018. URL: https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf.

[8] *WR-ZEN TP Product Site*. Seven Solutions. 2019. URL: https://sevensols.com/index.php/products/wr-zen-tp/.

[9] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks - Amendment 25: Enhancements for Scheduled Traffic". In: *IEEE Std 802.1Qbv-2015 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, and IEEE Std 802.1Q-2014/Cor 1-2015)* (2016), pp. 1–57. DOI: 10.1109/IEEESTD.2016.8613095.

[10] *IEC 61158-1:2019. Industrial communication networks – Fieldbus specifications – Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series*. IEC standards. URL: https://webstore.iec.ch/publication/59890.

[11] *IEC61158 Technology Comparison. State of the Bus.* English. Fieldbus Inc. Nov. 2020. URL: http://www.fieldbusinc.com/downloads/fieldbus_comparison.pdf.

[12] *Beckhoff Automation company site.* English. Beckhoff Automation GmbH & Co. KG. Nov. 2020. URL: https://www.beckhoff.com.

[13] *IEC 61784-1:2019. Industrial communication networks - Profiles Part 1: Fieldbus profiles.* IEC standards. URL: https://webstore.iec.ch/publication/59887.

[14] *Aircraft Data Network Part 7 Avionics Full-Duplex Switched Ethernet (AFDX) Network, ARINC Specification 664, Part 7.* English. Aeronautical Radio, Inc., June 2006.

[15] *FOUNDATION Fieldbus Engineering Guidelines.* English. Fieldbus Foundation. Sept. 2012. URL: https://fieldcommgroup.org/sites/default/files/technologies/ff/techspecs/system_engineering_guidelines_version_3.2.1.pdf.

[16] *PROFIBUS Technology and Application – System Description (online).* English. PROFIBUS Nutzerorganisation e.V. Nov. 2020. URL: https://www.profibus.com/download/profibus-technology-and-application-system-description/.

[17] *KNX Specifications.* English. KNX Association cvba. Nov. 2020. URL: https://my.knx.org/en/shop/knx-specifications.

[18] United States Department of Defense. *MIL-STD-1553B: Aircraft Internal Time Division Command/Response Multiplex Data B.* United States Department of Defense, Sept. 1978.

[19] Steve Parkes. "ESA ExoMars". In: *SpaceWire User's Manual.* STAR-Dundee Limited, 2012. Chap. 2, pp. 36–37. ISBN: 978-0-9573408-0-0. URL: https://www.star-dundee.com/wp-content/star_uploads/2019/05/SpaceWire-Users-Guide.pdf.

[20] SAE International. *Time-Triggered Ethernet AS6802.* English. Online. SAE International, Nov. 2016. URL: https://www.sae.org/standards/content/as6802/.

[21] *IEC 61784-2:2019. Industrial communication networks – Profiles – Part 2: Additional fieldbus profiles for real-time networks based on ISO/IEC/IEEE 8802-3.* IEC standards. URL: https://webstore.iec.ch/publication/59888.

[22] *EtherCAT TSN Communication Profile – project site.* English. EtherCAT Technology Group. Nov. 2020. URL: https://www.ethercat.org/en/downloads/downloads_254672A7CED54910B655F565B974F5AD.htm.

[23] *Roadmap of PROFINET over TSN.* English. PROFIBUS Nutzerorganisation e.V. Nov. 2020. URL: https://www.profibus.com/technology/industrie-40/profinet-over-tsn/.

[24] *Avnu Certification Testing Services website.* English. Interoperability Laboratory – University of New Hampshire. URL: https://www.iol.unh.edu/testing/switching/avnu.

[25] Gary Lee. "Chapter 3 - Switch Fabric Technology". In: *Cloud Networking.* Ed. by Gary Lee. Boston: Morgan Kaufmann, 2014, pp. 37–64. ISBN: 978-0-12-800728-0. DOI: https://doi.org/10.1016/B978-0-12-800728-0.00003-5. URL: http://www.sciencedirect.com/science/article/pii/B9780128007280000035.

[26] L. Montalvo, G. Mace, C. Chapel, S. Defrance, T. Tapie, and J. Le Roux. "Implementation of a TV studio based on Ethernet and the IP protocol stack". In: *2009 IEEE International Symposium on Broadband Multimedia Systems and Broadcasting.* 2009, pp. 1–7. DOI: 10.1109/ISBMSB.2009.5133736.

[27] A. Fredette. *"How Big do my pipes need to be?" – Traffic Shaping & Infrastructure planning*. English. AVnu Alliance Broadcast Advisory Council. May 2013. URL: https://avnu.org/wp-content/uploads/2014/05/AVnu-AABAC_Traffic-Shaping-Infrastructure-Planning_Andre-Fredette.pdf.

[28] J. Farkas. "Bounded Low Latency in IEEE 802.1 and IETF DetNEt". In: *TSN/A Conference 2017*. Ericsson. Stuttgart, Germany, Sept. 2017.

[29] *IEEE 802.3 Residential Ethernet Study Group*. IEEE standards. URL: http://grouper.ieee.org/groups/802/3/re_study/.

[30] IEEE. *Time-Sensitive Networking (TSN) Task Group*. English. IEEE. Apr. 24, 2020. URL: https://1.ieee802.org/tsn/.

[31] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. English. Online. IEEE. URL: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4579760.

[32] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat. "Synchronization Quality of IEEE 802.1AS in Large-Scale Industrial Automation Networks". In: *2017 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*. 2017, pp. 273–282. DOI: 10.1109/RTAS.2017.10.

[33] "IEEE Standard for Local and metropolitan area networks– Virtual Bridged Local Area Networks Amendment 12: Forwarding and Queuing Enhancements for Time-Sensitive Streams". In: *IEEE Std 802.1Qav-2009 (Amendment to IEEE Std 802.1Q-2005)* (2010), pp. 1–72. DOI: 10.1109/IEEESTD.2010.8684664.

[34] G. Bechtel, J. Specht, S. Maydiga, S. Samii, and M. Potts. "Ethernet Traffic Shapers to Support In-Vehicle Automotive Networking". In: *TSN/A Conference 2017*. Stuttgart, Germany, Sept. 2017.

[35] "IEEE Standard for Local and metropolitan area networks–Virtual Bridged Local Area Networks Amendment 14: Stream Reservation Protocol (SRP)". In: *IEEE Std 802.1Qat-2010 (Revision of IEEE Std 802.1Q-2005)* (2010), pp. 1–119. DOI: 10.1109/IEEESTD.2010.5594972.

[36] "IEEE Standard for Local and metropolitan area networks–Audio Video Bridging (AVB) Systems". In: *IEEE Std 802.1BA-2011* (2011), pp. 1–45. DOI: 10.1109/IEEESTD.2011.6032690.

[37] D. Panell. "Data Reliability and Redundancy with TSN". In: *TSN/A Conference 2017*. Stuttgart, Germany, Sept. 2017.

[38] "IEEE Standard for Local and Metropolitan Area Networks–Timing and Synchronization for Time-Sensitive Applications". In: *IEEE Std 802.1AS-2020 (Revision of IEEE Std 802.1AS-2011)* (2020), pp. 1–421. DOI: 10.1109/IEEESTD.2020.9121845.

[39] S. Vitturi, C. Zunino, and T. Sauter. "Industrial Communication Systems and Their Future Challenges: Next-Generation Ethernet, IIoT, and 5G". In: *Proceedings of the IEEE* 107.6 (2019), pp. 944–961. DOI: 10.1109/JPROC.2019.2913443.

[40] "IEEE Standard for Ethernet Amendment 5: Specification and Management Parameters for Interspersing Express Traffic". In: *IEEE Std 802.3br-2016 (Amendment to IEEE Std 802.3-2015 as amended by IEEE St802.3bw-2015, IEEE Std 802.3by-2016, IEEE Std 802.3bq-2016, and IEEE Std 802.3bp-2016)* (2016), pp. 1–58. DOI: 10.1109/IEEESTD.2016.7900321.

[41] "IEEE Standard for Local and metropolitan area networks – Bridges and Bridged Networks – Amendment 26: Frame Preemption". In: *IEEE Std 802.1Qbu-2016 (Amendment to IEEE Std 802.1Q-2014)* (2016), pp. 1–52. DOI: 10.1109/IEEESTD.2016.7553415.

[42] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks - Amendment 34:Asynchronous Traffic Shaping". In: *IEEE Std 802.1Qcr-2020 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018, IEEE Std 802.1Qcc-2018, IEEE Std 802.1Qcy-2019, and IEEE Std 802.1Qcx-2020)* (2020), pp. 1–151. DOI: 10.1109/IEEESTD.2020.9253013.

[43] *IEC 62439-3:2016 RLV. Industrial communication networks – High availability automation networks – Part 3: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy (HSR)*. IEC standards. URL: https://webstore.iec.ch/publication/24438.

[44] "IEEE Standard for Local and metropolitan area networks–Frame Replication and Elimination for Reliability". In: *IEEE Std 802.1CB-2017* (2017), pp. 1–102. DOI: 10.1109/IEEESTD.2017.8091139.

[45] "IEEE Standard for Local and metropolitan area networks–Bridges and Bridged Networks–Amendment 28: Per-Stream Filtering and Policing". In: *IEEE Std 802.1Qci-2017 (Amendment to IEEE Std 802.1Q-2014 as amended by IEEE Std 802.1Qca-2015, IEEE Std 802.1Qcd-2015, IEEE Std 802.1Q-2014/Cor 1-2015, IEEE Std 802.1Qbv-2015, IEEE Std 802.1Qbu-2016, and IEEE Std 802.1Qbz-2016)* (2017), pp. 1–65. DOI: 10.1109/IEEESTD.2017.8064221.

[46] "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks - Redline". In: *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014) - Redline* (2018), pp. 1–3654.

[47] "IEEE Standard for Local and Metropolitan Area Networks–Bridges and Bridged Networks – Amendment 31: Stream Reservation Protocol (SRP) Enhancements and Performance Improvements". In: *IEEE Std 802.1Qcc-2018 (Amendment to IEEE Std 802.1Q-2018 as amended by IEEE Std 802.1Qcp-2018)* (2018), pp. 1–208. DOI: 10.1109/IEEESTD.2018.8514112.

[48] *EVAL-RapID-TSNEK (formerly from Innovasic)*. English. Analog Devices. Nov. 2020. URL: https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/eval-rapid-tsnek.html#eb-overview.

[49] *Slate XNS*. English. TTTech Industrial Automation AG. Nov. 2020. URL: https://www.tttech-industrial.com/products/slate/slate-xns/.

[50] *PTP for Linux (ptp4l) project site*. English. The Linux PTP Project. Mar. 2020. URL: http://linuxptp.sourceforge.net.

[51] *KBox C-102-2 TSN Starterkit*. English. Kontron. Mar. 2020. URL: https://www.kontron.com/products/systems/ethernet-solutions/network-interfaces-tsn/kbox-c-102-2-tsn-starterkit.html.

[52] *OpenAvnu Project: an Avnu sponsored repository for Audio/Video Bridging and Time Sensitive Networking technology*. Avnu Alliance. URL: http://avnu.github.io/OpenAvnu/.

[53] *AVNU Alliance official site*. English. AVNU Alliance. Nov. 2020. URL: https://avnu.org.

[54] *Site of the TSN/A Conference. Technology and Applications*. English. AVNU Alliance. Sept. 2020. URL: https://events.weka-fachmedien.de/tsna-conference/home/.

[55] *IEC/IEEE 60802 TSN Profile for Industrial Automation*. IEEE standards. URL: https://1.ieee802.org/tsn/iec-ieee-60802/.

[56] G. Steindl. *An example selection of standards, standard attributes and standard extensions to cover an use cases and requirements selection*. English. Siemens AG. Jan. 2019. URL: https://www.ieee802.org/1/files/public/docs2019/60802-Steindl-ExampleSelections-0119-v02.pdf.

[57] A. Ademaj, G. Steindl, and T. Koskiahde. *Redundant Clock Synchronization*. English. TTTech. Nov. 2019. URL: https://www.ieee802.org/1/files/public/docs2019/60802-Ademaj-et-al-Redundant-Clocks-1119-v6.pdf.

[58] S.B. Carlson, J. Farkas, N. Finn, D. Panell, M. Potts, M. Seaman, N. Wienckowski, and G. Zimmerman. *IEEE 802 Ethernet Networks for Automotive*. English. IEEE 802 Plenary. 2017. URL: https://www.ieee802.org/802_tutorials/2017-07/tutorial-Automotive-Ethernet-0717-v02.pdf.

[59] *Octal 10/100BASE-TX Ethernet BroadR-Reach Transceiver*. English. Broadcom. URL: https://docs.broadcom.com/doc/1211168574381.

[60] "IEEE Standard for Local and metropolitan area networks – Time-Sensitive Networking for Fronthaul - Amendment 1: Enhancements to Fronthaul Profiles to Support New Fronthaul Interface, Synchronization, and Syntonization Standards". In: *IEEE Std 802.1CMde-2020 (Amendment to IEEE Std 802.1CM-2018* (2020), pp. 1–35. DOI: 10.1109/IEEESTD.2020.9228956.

[61] *CPRI – Common Public Radio Interface. Specification site*. English. Ericsson, Huawei, NEC, Nokia. Nov. 2020. URL: http://www.cpri.info/spec.html.

[62] European Cooperation for Space Standardization (European Space Agency). *SpaceWire–Links, Nodes, Routers, and Networks, ECSS-E-ST-50-12C Rev.1*. English. Online. European Cooperation for Space Standardization (European Space Agency), May 2019. URL: https://ecss.nl/standard/ecss-e-st-50-12c-rev-1-spacewire-links-nodes-routers-and-networks-15-may-2019/.

[63] J. Farkas, L. Winkel, C. Gunther, and G. Masini. "Standards Update". In: *TSN/A Conference 2020*. online, Sept. 2020.

[64] *SAE Standards site*. English. SAE Standards. Nov. 2020. URL: https://www.sae.org/standards/.

[65] *Application of TSN*. English. IEEE 802.1 Working Group. Nov. 2020. URL: https://1.ieee802.org/tsn/application-of-tsn/#Automotive_networks.

[66] J. Farkas, B. Varga, G. Myklós, and J. Sachs. *5G-TSN integration meets networking requirements for industrial automation*. English. Ericsson. Nov. 2020. URL: https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/5g-tsn-integration-for-industrial-automation.

[67] *Industrial Internet Consortium – official site*. English. Industrial Internet Consortium. Nov. 2020. URL: https://www.iiconsortium.org.

[68] *TTTech official site*. English. TTTech Computertechnik AG. Nov. 2020. URL: https://www.tttech.com.

[69] *TSN Starter Package site*. English. TTTech Computertechnik AG. Nov. 2020. URL: https://www.tttech-industrial.com/products/slate/.

[70] *Slate Edge IP solution*. English. TTTech Computertechnik AG. Nov. 2020. URL: https://www.tttech-industrial.com/products/slate/edge-ip-solution/.

[71] *Hi-Rel Solutions for Space Launch Vehicles*. TTTech. URL: https://www.tttech.com/markets/space/projects-references/ariane-6/.

[72] *Layerscape LS1028A Reference Design Board*. English. NXP Semiconductors. Nov. 2020. URL: https://www.nxp.com/design/qoriq-developer-resources/layerscape-ls1028a-reference-design-board:LS1028ARDB.

[73] *TSN-capable CompactRIO Controllers*. English. National Instruments. URL: http://www.ni.com/pdf/product-flyers/compactrio-controller.pdf.

[74] *White Rabbit Project*. Open Hardware Repository. URL: https://www.ohwr.org/project/white-rabbit/wikis/home.

[75] Pedro Moreira, Javier Serrano, Tomasz Wlostowski, Patrick Loschmidt, and Georg Gaderer. "White rabbit: Sub-nanosecond timing distribution over ethernet". In: *2009 International Symposium on Precision Clock Synchronization for Measurement, Control and Communication*. IEEE, Oct. 2009. DOI: 10.1109/ispcs.2009.5340196.

[76] G ITU. "8261-Timing and synchronization aspects in packet networks". In: *ITU Std* (2008).

[77] *IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. IEEE Std 1588-2008. July 2008.

[78] P. Moreira and I. Darwazeh. "Digital femtosecond time difference circuit for CERN's timing system". In: *Proc. London Commun. Symp.* 2011, pp. 1–4.

[79] Open Hardware Repository. *Simple PCIe FMC carrier (SPEC)*. English. Open Hardware Repository. URL: https://www.ohwr.org/project/spec/wikis/home.

[80] Sebastiano Aiello and et al. "KM3NeT front-end and readout electronics system: hardware, firmware, and software". In: *Journal of Astronomical Telescopes, Instruments, and Systems* 5.4 (2019). DOI: 10.1117/1.JATIS.5.4.046001, pp. 1–15. DOI: 10.1117/1.JATIS.5.4.046001. URL: https://doi.org/10.1117/1.JATIS.5.4.046001.

[81] *Zynq-7000 SoCs with Hardware and Software Programmability.* English. Xilinx. Nov. 2020. URL: https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html.

[82] *Vivado Design Suite - HLx Editions*. English. Xilinx. Nov. 2020. URL: https://www.xilinx.com/products/design-tools/vivado.html.

[83] *Buildroot - Making Embedded Linux Easy*. Buildroot. 2019. URL: https://buildroot.org/.

[84] *Busybox project site*. Busybox. 2019. URL: https://buildroot.org/.

[85] Alessandro Rubini. *Linux device drivers*. eng. 2nd ed. Sebastopol: O'Reilly & Associates. ISBN: 0-596-00008-1.

[86] *AXGE-1254 Datasheet: 1.25Gbps Single Fiber Bi-directional SFP, ONU Transceiver.* English. AXCEN Photonics Corporation. Nov. 2020. URL: https://www.axcen.com.tw/upload/product/202004281724140.pdf.

[87] *SFP-1GBT-06 copper-based SFP*. English. Belfuse Inc. Nov. 2020. URL: https://www.belfuse.com/product/part-details?partn=SFP-1GBT-06.

[88] *53230A RF Counter. Brochure for the 53200 Series of RF and Universal Frequency Counter/Timers*. Keysight. URL: https://www.keysight.com/us/en/assets/7018-02654/technical-overviews/5990-6339.pdf.

[89] *Vivado Hardware Debug*. English. Xilinx. Nov. 2020. URL: https://www.xilinx.com/products/design-tools/vivado/debug.html.

[90] Morion US. *MV89, Double Oven Controlled Crystal Oscillator*. URL: https://www.morion-us.com/catalog_pdf/mv89.pdf.

[91] ISO-TECH. *Arbitrary function generator – AFG 2100 Series*. Nov. 2020. URL: https://docs.rs-online.com/f4eb/0900766b8145f857.pdf.

[92] *Official Wireshark Site*. Wireshark. URL: https://www.wireshark.org/.

[93] *TCPDUMP & LIBPCAP*. English. The Tcpdump Group. Nov. 2020. URL: https://www.tcpdump.org/index.html.

[94] *IPERF3 project site*. ESnet/Lawrence Berkeley National Laboratory. 2020. URL: https://software.es.net/iperf/.

[95] *Calnex Paragon-x. Prove Ethernet Sync to 10GbE.* English. Calnex. Nov. 2020. URL: https://www.calnexsol.com/en/docman/brochures/6-paragon-x-brochure/file.

[96] Michael Lombardi. "Fundamentals of Time and Frequency". In: *The Mechatronics Handbook*. DOI: 10.1201/9781420037241.ch10. CRC Press, Jan. 2002. Chap. 17. DOI: 10.1201/9781420037241.ch10.

[97] Anders E.E. Wallin, Danny Price, Cantwell G. Carson, Frédéric Meynadier, Yan Xie, Erik Benkler. *AllanTools. A python library for calculating Allan deviation and related time & frequency statistics*. English. URL: https://github.com/aewallin/allantools.

[98] A. Farris, R. Marson, and J. Kern. "The ALMA Telescope Control System". In: *10th ICALEPCS Int. Conf. on Accelerator & Large Expt. Physics Control Systems, October, 2005, Geneva* (Geneva). Oct. 10, 2005. URL: https://accelconf.web.cern.ch/ica05/proceedings/pdf/O3_009.pdf.

[99] *CTA. Cherenkov Telescope Array Site*. English. CTA. URL: https://www.cta-observatory.org/.

[100] Niko Neufeld. "Future of DAQ Frameworks and Approaches, and Their Evolution towards the Internet of Things". In: *Journal of Physics: Conference Series* 664 (Dec. 2015), p. 082038. DOI: 10.1088/1742-6596/664/8/082038.

[101] *The White Rabbit Project*. English. Open Hardware Repository. URL: https://ohwr.org/projects/white-rabbit.

[102] *Common Object Request Broker Architecture – Project Site*. English. Object Management Group (OMG). URL: https://www.corba.org/.

[103] *ZeroMQ. An open-source universal messaging library*. English. ZeroMQ. URL: https://zeromq.org/.

[104] Serguei Kolos and Reiner Hauser. *The new inter process communication middle-ware for the ATLAS Trigger and Data Acquisition system*. Tech. rep. ATL-DAQ-PROC-2016-042. Geneva: CERN, Dec. 2016. URL: https://cds.cern.ch/record/2239746.

[105] *MIDAS Middleware Site*. English. MIDAS. URL: https://midas.triumf.ca/MidasWiki/index.php/Main_Page.

[106] *DAQ-Middleware Site*. Japanese. KEK (High Energy Accelerator Research Organization). URL: https://daqmw.kek.jp/.

[107] *Overview of LabView Features – National Instruments Site*. English. National Instruments). URL: https://www.ni.com/en-us/shop/labview.html.

[108] *Google Protocol Buffers for Serializing Structured Data*. English. Google. URL: https://developers.google.com/protocol-buffers.

[109] E. Lyard and R. Walter. *End-to-end data acquisition pipeline for the Cherenkov Telescope Array*. 2017. arXiv: 1709.04203 [astro-ph.IM].

[110] *TDC7200 Time-to-Digital Converter for Time-of-Flight Applications*. English. Texas Instruments. URL: http://www.ti.com/lit/ds/symlink/tdc7200.pdf?ts=1588691655081.

[111] PMT (Precision Measurement Technologies). *TDC- Time-to-Digital Converters*. English. URL: https://www.pmt-fl.com/tdc-time-to-digital-converters.

[112] L. Arpin, M. Bergeron, M. A. Tetrault, R. Lecomte, and R. Fontaine. "A Sub-Nanosecond Time Interval Detection System Using FPGA Embedded I/O Resources". In: *IEEE Transactions on Nuclear Science* 57.2 (Apr. 2010), pp. 519–524. ISSN: 0018-9499. DOI: 10.1109/TNS.2009.2039804.

[113] R. G. Baron. "The Vernier Time-Measuring Technique". In: *Proceedings of the IRE* 45.1 (1957). DOI: 10.1109/JRPROC.1957.278252, pp. 21–30. DOI: 10.1109/JRPROC.1957.278252.

[114] Claudio Favi and E. Charbon. "A 17ps Time-to-digital Converter Implemented in 65nm FPGA Technology". In: Jan. 2009, pp. 113–120. DOI: 10.1145/1508128.1508145.

[115] C. Champion, M. Punch, R. Oger, S. Colonges, and Y. Moudden. "TiCkS: A Flexible White-Rabbit Based Time-Stamping Board". In: *Proc. of International Conference on Accelerator and Large Experimental Control Systems (ICALEPCS'17), Barcelona, Spain, 8-13 October 2017*. https://doi.org/10.18429/JACoW-ICALEPCS2017-TUPHA090. Oct. 19, 2017, pp. 622–626. DOI: https://doi.org/10.18429/JACoW-ICALEPCS2017-TUPHA090. arXiv: 1710.07128v1 [astro-ph.IM]. URL: http://jacow.org/icalepcs2017/papers/tupha090.pdf.

[116] Jacques Marteau, Jean de Bremond d'Ars, Dominique Gibert, Kevin Jourde, Jean-Christophe Ianigro, and Bruno Carlus. "DIAPHANE: Muon tomography applied to volcanoes, civil engineering, archaelogy". In: (Dec. 12, 2016). DOI: 10.1088/1748-0221/12/02/C02008. arXiv: 1612.03905v1 [physics.geo-ph].

[117] *Science with the Cherenkov Telescope Array*. Vol. arXiv:1709.07997. CTA Consortium. 2017. DOI: 10.1142/10986,2019.

[118] *Seeing the High-Energy Universe with the Cherenkov Telescope Array - The Science Explored with the CTA*. Astropart. Phys. Vol. Vol 43 1-356. 1-356. CTA Consortium, 2013.

[119] K. Bernlöhr et al. "Monte Carlo design studies for the Cherenkov Telescope Array". In: *Astroparticle Physics* 43 (2013). Seeing the High-Energy Universe with the Cherenkov Telescope Array - The Science Explored with the CTA, pp. 171–188. ISSN: 0927-6505. DOI: https://doi.org/10.1016/j.astropartphys.2012.10.002. URL: http://www.sciencedirect.com/science/article/pii/S0927650512001867.

[120] M. Jimenez-Lopez; J.M. Machado-Cano; M. Rodriguez-Alvarez; M. Stephan; G. Giavitto; D. Berge; J. Diaz. "Optimized framegrabber for the Cherenkov telescope array". In: *Journal of Astronomical Telescopes, Instruments, and Systems* 5.1 (2019), pp. 1 - 11 –11. DOI: 10.1117/1.JATIS.5.1.014001. URL: https://doi.org/10.1117/1.JATIS.5.1.014001.

[121] M. Jiménez-López, J. L. Gutiérrez-Rivas, J. Díaz, E. López-Marín, and R. Rodríguez. "WR-ZEN: Ultra-accurate synchronization SoC based on Zynq technology". In: *2016 European Frequency and Time Forum (EFTF)*. Apr. 2016, pp. 1–4. DOI: 10.1109/EFTF.2016.7477790.

[122] J. Zorn et al. "Characterisation and testing of CHEC-M—A camera prototype for the small-sized telescopes of the Cherenkov telescope array". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 904 (2018), pp. 44–63. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2018.06.078. URL: http://www.sciencedirect.com/science/article/pii/S0168900218308143.

[123] Miguel Jimenez-Lopez, Felipe Torres-González, José Gutiérrez Rivas, Manuel Rodriguez-Alvarez, and Javier Díaz. "A Fully Programmable White-Rabbit Node for the SKA Telescope PPS Distribution System". In: *IEEE Transactions on Instrumentation and Measurement* PP (July 2018), pp. 1–10. DOI: 10.1109/TIM.2018.2851658.

[124] R. White. "CHEC: a Compact High Energy Camera for the Cherenkov Telescope Array". In: *Journal of Instrumentation* 12.12 (Dec. 2017), pp. C12059–C12059. DOI: 10.1088/1748-0221/12/12/c12059. URL: https://doi.org/10.1088%2F1748-0221%2F12%2F12%2Fc12059.

[125] U. Schwanke, M. Shayduk, K.-H. Sulanke, S. Vorobiov, and R. Wischnewski. "A versatile digital camera trigger for telescopes in the Cherenkov Telescope Array". In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 782 (2015), pp. 92–103. ISSN: 0168-9002. DOI: https://doi.org/10.1016/j.nima.2015.01.096. URL: http://www.sciencedirect.com/science/article/pii/S0168900215001382.

[126] D. Russo and S. Ricci. "FPGA Implementation of a Synchronization Circuit for Arbitrary Trigger Sequences". In: *IEEE Transactions on Instrumentation and Measurement* (2019), pp. 1–1. ISSN: 1557-9662. DOI: 10.1109/TIM.2019.2952478.

[127] *7-Series FPGAs SelectIO Resources*. English. Xilinx. URL: https://www.xilinx.com/support/documentation/user_guides/ug471_7Series_SelectIO.pdf.

[128] Xilinx. *7-Series FPGAs Data Sheet: Overview*. English. https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf. Xilinx. URL: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf.

[129] *WR-ZEN Board Official Site*. Seven Solutions. URL: http://sevensols.com/index.php/products/wr-zen-board/.

[130] M. Jimenez-Lopez; J.L. Gutierrez; J. Diaz Javier; E. Marin; and R. Rodriguez. "WR-ZEN: Ultra-accurate Synchronization SoC Based On Zynq Technology". In: *European Frequency and Time Forum (EFTF)* (2016).

[131] M. Jimenez-Lopez; J.L. Gutierrez; and J. Diaz. "A White-Rabbit Network Interface Card for Synchronized Sensor Networks". In: *IEEE Sensors* 2.5 (Nov. 2014), pp. 2000–2009. DOI: 10.1109/ICENS.2014.6985426.

[132] J. Sanchez-Garrido, A. M. Lopez-Antequera, M. Jimenez-Lopez, and J. Diaz. "Sub-nanosecond Synchronization over 1G Ethernet data links using White Rabbit technologies on the WR-ZEN board". In: *2017 40th Int. Conf. on Telecommunications and Signal Processing (TSP)*. July 2017, pp. 688–693. DOI: 10.1109/TSP.2017.8076075.

[133] ARM Ltd. *ARM Cortex-A9 Technical Reference Manual*. URL: https://static.docs.arm.com/100511/0401/arm_cortexa9_trm_100511_0401_10_en.pdf?_ga=2.238924928.1418237512.1581792457-1175236830.1581792457.

[134] *Xilinx AXI DataMover v5.1 Product Guide*. English. Xilinx. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_datamover/v5_1/pg022_axi_datamover.pdf.

[135] *Xilinx DMA Controller Documentation*. English. Xilinx. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf.

[136] *White Rabbit PTP Core - Open Hardware Repository*. English. Open Hardware Repository. URL: http://www.ohwr.org/projects/wr-cores/wiki/wrpc_core.

[137] J. Lopez-Jimenez, M. Jimenez-Lopez, J. Diaz, and J. L. Gutierrez-Rivas. "White-rabbit-enabled data acquisition system". English. In: *2017 Joint Conf. of the Eu. Frequency and Time Forum and IEEE Int. Frequency Control Symp., EFTF/IFC 2017 - Proceedings*. 2017, pp. 410–416.

[138] Seven Solutions. *WR-ZEN Time-Provider Data Sheet*. URL: https://sevensols.com/index.php/download/brochure-white-rabbit-zen-tp/?wpdmdl=1005.

[139] F. Torres-González, J. Díaz, E. Marín-López, and R. Rodriguez-Gómez. "Scalability analysis of the white-rabbit technology for cascade-chain networks". English. In: *IEEE Int. Symp. on Precision Clock Synchronization for Measurement, Control, and Communication, ISPCS*. Vol. 2016-September. 2016.

[140] David Calvo. "1 ns time to digital converters for the KM3NeT data readout system". In: *AIP Conference Proceedings* 1630 (Nov. 2014), pp. 98–101. DOI: 10.1063/1.4902781.

[141] J. Marteau, Jean de Bremond d'Ars, Dominique Gibert, Kevin Jourde, Serge Gardien, Claude Girerd, and J. Ianigro. "Implementation of sub-nanoseconds TDC in FPGA: applications to time-of-flight analysis in muon radiography". In: *http://lanl.arxiv.org/abs/1310.4281* (Oct. 2013). DOI: 10.1088/0957-0233/25/3/035101.

[142] E. Houtzager, R. Hornecker, and G. Rietveld. "Compact Distributed Digitizers With Metrological Precision". In: *IEEE Transactions on Instrumentation and Measurement* 68.6 (June 2019), pp. 1653–1658. ISSN: 1557-9662. DOI: 10.1109/TIM.2018.2878090.

[143] *EISCAT Project Site*. English. EISCAT. URL: https://eiscat.se.

[144] *South Pole Neutrino Observatory*. English. ICECUBE. URL: https://icecube.wisc.edu/.

[145] *Xilinx DMA Controller Documentation*. English. Xilinx. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_dma/v7_1/pg021_axi_dma.pdf.

[146] *1G/2.5G Ethernet PCS/PMA or SGMII v16.2*. English. Xilinx. Nov. 2020. URL: https://www.xilinx.com/support/documentation/ip_documentation/gig_ethernet_pcs_pma/v16_2/pg047-gig-eth-pcs-pma.pdf.

[147] *Tri-Mode Ethernet MAC v9.0*. English. Xilinx. Nov. 2020. URL: https://www.xilinx.com/support/documentation/ip_documentation/tri_mode_ethernet_mac/v9_0/pg051-tri-mode-eth-mac.pdf.

[148] Alex Forencich. *Verilog Ethernet Components. A collection of 1G, 10G, 25G Packet Processing Data Paths*. English. URL: https://gitlab.com/alex.forencich/verilog-ethernet.

[149] *7 Series FPGAs GTP Transceivers*. English. Xilinx. Nov. 2020. URL: https://www.xilinx.com/support/documentation/user_guides/ug482_7Series_GTP_Transceivers.pdf.

[150] *AXI Interconnect v2.1 PG059 product guide*. English. Xilinx. Dec. 2017. URL: https://www.xilinx.com/support/documentation/application_notes/xapp1151_Param_CAM.pdf.

[151] *Linux AXI Ethernet Driver*. Xilinx. URL: https://xilinx-wiki.atlassian.net/wiki/spaces/A/pages/18842485/Linux+AXI+Ethernet+driver.

[152] *RTEMS Real-Time Operating System (RTOS) Site*. The RTEMS Project. URL: https://www.rtems.org/.

[153] *Overview of the NCURSES library*. English. Invisible Island. Nov. 2020. URL: https://invisible-island.net/ncurses/announce.html#h2-overview.

[154] *AXI 1G/2.5G Ethernet Subsystem v7.0. Product guide*. English. Xilinx. Nov. 2020. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_ethernet/v7_0/pg138-axi-ethernet.pdf.

[155] L. Zhao, P. Pop, and S. S. Craciunas. "Worst-Case Latency Analysis for IEEE 802.1Qbv Time Sensitive Networks Using Network Calculus". In: *IEEE Access* 6 (2018), pp. 41803–41815. DOI: 10.1109/ACCESS.2018.2858767.

[156] V. Gavriluţ and P. Pop. "Traffic-Type Assignment for TSN-Based Mixed-Criticality Cyber-Physical Systems". In: *ACM Trans. Cyber-Phys. Syst.* 4.2 (Jan. 2020). ISSN: 2378-962X. DOI: 10.1145/3371708. URL: https://doi.org/10.1145/3371708.

[157] *PPSi project repository site.* English. Open Hardware Repository. Nov. 2020. URL: https://ohwr.org/project/ppsi.

[158] *AD9516-5: 14-Output Clock Generator (product site).* English. Analog Devices. Nov. 2020. URL: https://www.analog.com/en/products/ad9516-5.html#product-overview.

[159] *RTEMS Source Builder documentation.* English. RTEMS Documentation Project. Nov. 2020. URL: https://docs.rtems.org/branches/master/user/rsb/index.html.

[160] *Overview of White Rabbit technology.* English. Seven Solutions S.L. Nov. 2020. URL: https://sevensols.com/index.php/projects/white-rabbit-technology/.

[161] *White Rabbit calibration and the link delay model.* English. Open Hardware Repository. Nov. 2020. URL: https://ohwr.org/project/white-rabbit/wikis/Calibration.

[162] A. Iqbal, D. Fifield, and J. Kadambi. *GMII Timing and Electrical Specification.* English. IEEE 802.3z Task Force. Nov. 1996. URL: https://www.analog.com/en/products/ad9516-5.html#product-overview.

[163] *GMII to RGMII v4.0.* English. Xilinx. URL: https://www.xilinx.com/support/documentation/ip_documentation/gmii_to_rgmii/v4_0/pg160-gmii-to-rgmii.pdf.

[164] *100M/1G TSN Subsystem.* Xilinx. URL: https://www.xilinx.com/products/intellectual-property/1gtsn.html.

[165] K. Locke. *Parameterizable Content-Addressable Memory.* English. Xilinx. Mar. 2011. URL: https://www.xilinx.com/support/documentation/application_notes/xapp1151_Param_CAM.pdf.

[166] Bob Noseworthy. *Time Sensitive Networks (TSN) Overview.* English. University of New Hampshire – InterOperability Laboratory. Nov. 2015. URL: https://www.iol.unh.edu/sites/default/files/knowledgebase/UNH-IOL_TSN-Overview.pdf.

[167] *Grupo Cuerva S.L. Electrical Utilities. Official Site (English Version).* Grupo Cuerva. URL: https://www.grupocuerva.com/en/.

[168] *OnGranada coordina un proyecto de redes eléctricas inteligentes para mejorar la industria.* Spanish. GranadaHoy. Mar. 2019. URL: https://www.granadahoy.com/granada/OnGranada-coordina-electricas-inteligentes-industria_0_1333966867.html.

[169] J. Sanchez-Garrido, A. Jurado, L. Medina, R. Rodriguez, E. Ros, and J. Diaz. "Digital Electrical Substation Communications Based on Deterministic Time-Sensitive Networking Over Ethernet". In: *IEEE Access* 8 (2020), pp. 93621–93634.

[170] H. Farhangi *et al.* "The path of the smart grid". In: *IEEE Power Energy Mag.* 8 (1 Dec. 22, 2009). DOI: 10.1109/mpe.2009.934876, pp. 18–28. DOI: 10.1109/mpe.2009.934876. URL: https://ieeexplore.ieee.org/document/5357331/ (visited on 04/24/2020).

[171] Red Eléctrica de España. *Red21.* Spanish. Red Eléctrica de España. Dec. 2018. URL: https://www.ree.es/es/red21/redes-inteligentes/que-son-las-smartgrid.

[172] R. E. Mackiewicz. “Overview of IEC 61850 and Benefits”. In: *Overview of IEC 61850 and Benefits. IEEE PES Power Syst. Conf. and Expo.* IEEE PES Power Syst. Conf. and Expo. DOI: 10.1109/PSCE.2006.296392. IEEE, 2006, pp. 623–630. ISBN: 1-4244-0177-1. DOI: 10.1109/PSCE.2006.296392.

[173] J. L. Gutiérrez-Rivas, J. López-Jiménez, E. Ros and J. Díaz. “White Rabbit HSR: A Seamless Subnanosecond Redundant Timing System With Low-Latency Data Capabilities for the Smart Grid”. In: *IEEE Trans. Ind. Informat.* 14.8 (Aug. 1, 2018). DOI: 10.1109/TII.2017.2779240, pp. 3486–3494. DOI: 10.1109/TII.2017.2779240.

[174] J. Sanchez-Garrido, A. Jurado, J.M Machado-Cano, M. Fuentes-Garcia, A. Sanchez-Perez, J. Cuadros-Vilchez, A. Alcantara-Lopez, J. Torres-Tenor, E. Ros, and J. Diaz. “TSN en Smart Grids – Comunicaciones Deterministas para Operaciones Críticas”. Spanish. In: *VI Congreso Smart Grids, 2019 Madrid*. Dec. 12, 2019. URL: https://static.smartgridsinfo.es/media/2020/01/6-congreso-smart-grids-libro-comunicaciones.pdf.

[175] H. Lei, C. Singh, and A. Sprintson. “Reliability Modeling and Analysis of IEC 61850 Based Substation Protection Systems”. In: *IEEE Trans. Smart Grid* 5.5 (Sept. 2014). DOI: 10.1109/TSG.2014.2314616, pp. 2194–2202. ISSN: 1949-3061. DOI: 10.1109/TSG.2014.2314616.

[176] IEEE. *Audio Video Bridging Task Group*. English. URL: http://grouper.ieee.org/groups/802/1/pages/avbridges.html.

[177] *IEC 61850-5. Communication networks and systems for power utility automation - Part 5: Communication requirements for functions and device models*. Online. IEC standards, Jan. 2013.

[178] P. Risbud, N. Gatsis, and A. Taha. “Vulnerability Analysis of Smart Grids to GPS Spoofing”. In: *IEEE Trans. Smart Grid* 10.4 (July 2019). DOI: 10.1109/TSG.2018.2830118, pp. 3535–3548. ISSN: 1949-3061. DOI: 10.1109/TSG.2018.2830118.

[179] N. Moreira, J. Lázaro, U. Bidarte, J. Jimenez, and A. Astarloa. “On the Utilization of System-on-Chip Platforms to Achieve Nanosecond Synchronization Accuracies in Substation Automation Systems”. In: *IEEE Trans. Smart Grid* 8.4 (July 2017). DOI: 10.1109/TSG.2015.2512440, pp. 1932–1942. ISSN: 1949-3061. DOI: 10.1109/TSG.2015.2512440.

[180] D. Mashima, P. Gunathilaka, and B. Chen. “Artificial Command Delaying for Secure Substation Remote Control: Design and Implementation”. In: *IEEE Trans. Smart Grid* 10.1 (Jan. 2019). DOI: 10.1109/TSG.2017.2744802, pp. 471–482. ISSN: 1949-3061. DOI: 10.1109/TSG.2017.2744802.

[181] *ZIV Automation Document Library*. ZIV Automation. URL: https://www.zivautomation.com/downloads/#t2-p7.

[182] *CIRCE Foundation for Renewable Energy. Official Site*. CIRCE Foundation. URL: http://www.fcirce.es/que-es-circe.

[183] *PackETH Ethernet Packet Generator Project website*. PackETH project. URL: http://packeth.sourceforge.net/packeth/Home.html.

[184] Bohui Wang, Weisheng Chen, Bin Zhang, and Yu Zhao. "Regulation cooperative control for heterogeneous uncertain chaotic systems with time delay: A synchronization errors estimation framework". In: *Automatica* 108 (2019). DOI: https://doi.org/10.1016/j.automatica.2019.06.038, p. 108486. ISSN: 0005-1098. DOI: https://doi.org/10.1016/j.automatica.2019.06.038. URL: http://www.sciencedirect.com/science/article/pii/S0005109819303383.

[185] S. Lu, S. Repo, D. D. Giustina, F. A. Figuerola, A. Löf, and M. Pikkarainen. "Real-Time Low Voltage Network Monitoring – ICT Architecture and Field Test Experience". In: *IEEE Trans. Smart Grid* 6.4 (July 2015). DOI: 10.1109/TSG.2014.2371853, pp. 2002–2012. ISSN: 1949-3061. DOI: 10.1109/TSG.2014.2371853.

[186] M. Tugnoli, M. Sarret, and M. Aliberti. "Overview on Micro Launchers". In: *European Access to Space: Business and Policy Perspectives on Micro Launchers*. Cham: Springer International Publishing, 2019, pp. 5–28. ISBN: 978-3-319-78960-6. DOI: 10.1007/978-3-319-78960-6_2. URL: https://doi.org/10.1007/978-3-319-78960-6%5C_2.

[187] *Ariane 5*. ESA (European Space Agency). URL: http://www.esa.int/Enabling_Support/Space_Transportation/Launch_vehicles/Ariane_5.

[188] T. Cussac, M.A. Clair, P. Ultré-Guerard, F. Buisson, G. Lassalle-Balier, M. Ledu, C. Elisabelar, X. Passot, and N. Rey. "The Demeter microsatellite and ground segment". In: *Planetary and Space Science* 54.5 (2006). First Results of the DEMETER Micro-Satellite, pp. 413–427. ISSN: 0032-0633. DOI: https://doi.org/10.1016/j.pss.2005.10.013. URL: http://www.sciencedirect.com/science/article/pii/S0032063305002047.

[189] R. Clavier, P. Sautereau, and J.F. Dufour. "TTEthernet, a promising candidate for Ariane 6". In: *DASIA - DAta Systems In Aerospace, Proceedings* 725 (2014), p. 34.

[190] *MIURA 1 information site from PLD Space*. English. PLD Space. 2020. URL: https://www.pldspace.com/en/miura-1.

[191] *GMV – Official Company Site*. English. GMV. 2020. URL: https://www.gmv.com/en/.

[192] *PLD Space – Official Company Site*. English. PLD Space. 2020. URL: https://pldspace.com/en/.

[193] L. Medina-Valdes, M. Melara, and L. Cercos. "MIURA 1: Data Handling System". English. In: *13th ESA Workshop ADCSS 2019, Nov. 12-14, 2019 ESA/ESTEC, Noordwijk*. Nov. 12, 2019. URL: https://indico.esa.int/event/323/contributions/5043/attachments/3745/5201/12.30%5C_-%5C_An%5C_IPCORE%5C_for%5C_Deterministic%5C_Ethernet%5C_via%5C_TSN%5C_...%5C_.pdf.

[194] National Instruments. *IC-317x Industrial Controller with Reconfigurable I/O. User Manual*. National Instruments. URL: http://www.ni.com/pdf/manuals/375285b.pdf.

[195] M. Melara, C. Domínguez, L. Cercós, G. Ramírez, R. Rodríguez, E. González, J. Bru, and J.P. Préaud. "MIURA 1: Data Handling System". Spanish. In: *Int. Space Syst. Conf. DASIA, June 4-6, 2019 Torremolinos*. June 4, 2019. URL: https://eurospace.org/conferences-events/.

[196] M. Amrbar, F. Irom, S. M. Guertin, and G. Allen. “Heavy Ion Single Event Effects Measurements of Xilinx Zynq-7000 FPGA”. In: *2015 IEEE Radiation Effects Data Workshop (REDW)*. 2015, pp. 1–4.

[197] M. Hu, J. Luo, Y. Wang, and B. Veeravalli. “Scheduling periodic task graphs for safety-critical time-triggered avionic systems”. In: *IEEE Transactions on Aerospace and Electronic Systems* 51.3 (2015), pp. 2294–2304.

[198] S. Janković, A. Smiljanić, M. Vesović, H. Redžović, M. Bežulj, A. Radošević, and S. Moro. “High-capacity FPGA Router for Satellite Backbone Network”. In: *IEEE Transactions on Aerospace and Electronic Systems* (2019), pp. 1–1.

[199] “IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems”. In: *IEEE Std 1588-2019 (Revision ofIEEE Std 1588-2008)* (2020), pp. 1–499. DOI: 10.1109/IEEESTD.2020.9120376.

[200] *Linux kernel from Xilinx*. Xilinx. 2019. URL: https://github.com/Xilinx/linux-xlnx.

[201] *U-boot from Xilinx*. Xilinx. 2019. URL: https://github.com/Xilinx/u-boot-xlnx.

[202] *How to achieve Gigabit speeds with Linux*. English. DataTAG. URL: https://datatag.web.cern.ch/datatag/howto/tcp.html.

[203] D. X. Wei, C. Jin, S. H. Low, and S. Hegde. “FAST TCP: Motivation, architecture, algorithms, performance”. English. In: *IEEE/ACM Transactions on Networking* 14.6 (2006), pp. 1246–1259.

DECLARACIÓN

El doctorando Jorge Sánchez Garrido y los directores de la tesis Antonio Javier Díaz Alonso y Eduardo Ros Vidal garantizamos, al firmar esta tesis doctoral, que el trabajo ha sido realizado por el doctorando bajo la dirección de los directores de la tesis y hasta donde nuestro conocimiento alcanza, en la realización del trabajo, se han respetado los derechos de otros autores a ser citados, cuando se han utilizado sus resultados o publicaciones.

Granada, Noviembre 2020

[illegible]

Jorge Sánchez Garrido	Antonio Javier Díaz Alonso	Eduardo Ros Vidal