Remote TSN Frame Encapsulation by means of FPGA

Jaime Jiménez (jaime.jimenez@ehu.es)
University of the Basque Country University School of Industrial Technical Engineering: Universidad del Pais Vasco Escuela Universitaria de Ingenieria Tecnica Industrial https://orcid.org/0000-0002-3804-678X

Igor Rodríguez
University of the Basque Country University School of Industrial Technical Engineering: Universidad del Pais Vasco Escuela Universitaria de Ingenieria Tecnica Industrial

David Reguilón
University of the Basque Country University School of Industrial Technical Engineering: Universidad del Pais Vasco Escuela Universitaria de Ingenieria Tecnica Industrial

Aitzol Zuloaga
University of the Basque Country University School of Industrial Technical Engineering: Universidad del Pais Vasco Escuela Universitaria de Ingenieria Tecnica Industrial

Jesús Lázaro
University of the Basque Country University School of Industrial Technical Engineering: Universidad del Pais Vasco Escuela Universitaria de Ingenieria Tecnica Industrial

Research Article

Keywords: Smart Grid, TSN, IEC 61850

Posted Date: June 28th, 2021

DOI: https://doi.org/10.21203/rs.3.rs-481011/v1

Noname manuscript No.
(will be inserted by the editor)

Remote TSN frame encapsulation by means of FPGA

Jaime Jiménez · Igor Rodríguez · David Reguilón · Aitzol Zuloaga · Jesús Lázaro

Received: date / Accepted: date

Abstract TSN (Time-Sensitive Networking) has replaced outdated Fieldbus and non-deterministic Ethernet in the Industry 4.0. Field buses are not capable of providing neither connection for industry 4.0 IoT (Internet of Things) nor compatibility between different manufacturers. On the other hand, Ethernet is not able to ensure real-time. On the contrary, TSN guarantees real-time transmission, IoT and compatibility between devices. However, adapting to frequently changing needs makes TSN protocol evolve continuously. For this reason, devices for TSN analysis, such as PCs or not advanced frame analysis equipment are not able to process TSN packets at the speed that standard advances, discarding them as wrong frames. The integration of a System on Chip (SoC) that contains an FPGA (Field Programmable Gate Array) and a microcontroller, with capacity for reconfiguration and monitoring of the frames in the protocol, would be an ideal solution to this problem. This paper describes how to encapsulate TSN frames in Ethernet packets using an FPGA. Such Ethernet frames can subsequently be decapsulated, i.e. in a PC, and thus enable analysing TSN traffic in nonspecialized devices.

Keywords Smart Grid · TSN · IEC 61850

1 Introduction

Since industry 4.0 is now a reality, a type of communication that allows introducing IoT (Internet of Things) and CPS (Cyberphysical Systems) into the industry has been required [8]. At this time, although there are several industrial Ethernet protocols available on the market, in most cases the version sold by each manufacturer is different. This implies that only manufacturers offering equipment using the same protocol will be compatible with each

Jaime Jiménez · Igor Rodríguez · David Reguilón · Aitzol Zuloaga · Jesús Lázaro
UPV/EHU Plaza Ingeniero Torres Quevedo 1 E-mail: jaime.jimenez@ehu.eus

other. In addition, it forces clients to stock up from just one supplier or to perform multiple protocol conversions. However, Industry 4.0 requires a full degree of automation and this entails interconnection between different devices [7,4]. The current set of Time-Sensitive Networking (TSN) standards, in development within the IEEE standardization group, emerges as a solution to the above mentioned problem, because of its real-time communications, bounded latency, and the guarantee of being a set of international rules [2]. Frame processing cards, based either on microcontroller or ASIC (Application Specific Integrated Circuit), have partially implemented TSN. In the case of microcontroller-based ones, as all the process is performed in the microprocessor, which leads to longer processing time compared to an FPGA (Field Programmable Gate Array) based system, when performing algorithm solutions. The reason is that an FPGA allows algorithm parallelization, which means solutions are obtained in less clock cycles. On the other hand, ASIC-based ones are neither reconfigurable nor programmable, so they cannot be updated according to TSN standards; therefore, sometime ASIC will stop working as intended. To solve the current limitations that other devices have, a SoC on an FPGA could be a suitable solution. This SoC integrates a hardware processor and a programmable logic part that adds flexibility and ensures tight integration of TSN between both parts. In addition, the reprogrammable features allows upgrading, to handle TSN frames or to be updated to evolving standards. Additionally, the operating system allows to be reconfigured remotely through the Internet. Section II introduces TSN and analyses its most remarkable characteristics. After that, in section III, the state of the art of FPGA remote configuration is studied. To finish, in the section IV, our solution based on remote configuration and frame encapsulation of TSN for FPGA is presented.

2 Time-Sensitive Networking

2.1 Introduction

TSN is a set of standards and published projects develop by the TSN group, which is part of the IEEE 802.1 Working Group [5]. This is responsible of defining mechanisms that guarantee deterministic services for IEEE 802 networks [1].

2.2 TSN characteristics

The different standards add certain features or improvements listed below [1]:

– Timing and Synchronization. The 802.1AS-2011 IEEE standard enables precise timing between network nodes. It is capable of an accuracy better than 1 μ s. However, the loss of an active master is solved by choosing a new one again, and this can lead to time jumps in some nodes.

- To solve this, the 802.1AS-Rev IEEE standard was developed to keep time synchronization for time sensitive applications. In this standard, redundancy is improved to achieve continuous low latency transfer and a quick recovery of time synchronization from failures.
- Bounded low latency. TSN improves the real-time operation by two standards. The first one, IEEE 802.1Qbv, defines programmed communication to leverage synchronized time in transmissions, and deciding to forward messages in the network. And the second one, IEEE 802.1Qbu, allows prioritized transfers and may interrupt transmissions of lower priority.
- Reliability. Various TSN standards contribute to communication reliability in different ways: duplication of packages to reduce the probability of packet loss, control of the path and the bandwidth reservation, detection of streams that do not behave properly and the use of mitigation actions.
- Resource management. The reservation and management of resources is the key to get deterministic networks. Multiple TSN standards contribute in a different way to the reservation and management of the bandwidth, thus distributing the traffic load and making the transmission more efficient.

2.3 Implementation

Authors have designed an FPGA core to encapsulate TSN frames in Ethernet packets, so that a PC or similar equipment can receive them. We used an FPGA, since it allows to be reconfigured, as the protocols of communication are being developed and it will be necessary to use a system that could be upgraded. Moreover, this hardware allows real-time transmission with precision enough to satisfy the implementation of TSN requirements.

3 Remote reconfiguration of the FPGA

The current SoC development allows to integrate a microprocessor with an FPGA in the same chip. Furthermore, it is capable of running Linux on ARM architecture (Advanced RISC Machines) and, at the same time, making dynamic reconfigurations of the FPGA from the microprocessor. The remote connection to Linux system, to reconfigure dynamically the FPGA, allows updating the frame handling devices in a very short time. Thus, cards can always be updated with the latest changes of the protocol.

3.1 Dynamic reconfiguration

Reconfiguration of an FPGA is the process to load a hardware design into the device by transferring the bitstream which describes it. It is equivalent to programming it. However, this process of reprogramming takes between microseconds and milliseconds, and in applications that need real time, such as TSN, reprogramming for protocol updates was not possible due to the loss

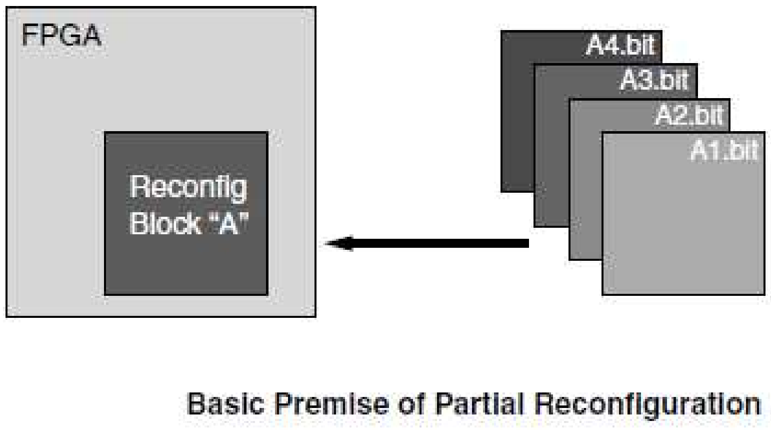

Fig. 1 Demonstration of partial reconfiguration.

of frames involved. Also, to reconfigure an FPGA so far, it was necessary to do it from a microcontroller or external memory, which required the presence of a physical connection with the device, to download the bitstream, which limited remote reconfigurations.

Because of the necessary time to write the bitstream, manufacturers have developed the dynamic reconfiguration instead of the usual static one of designs in the FPGA:

– Static reconfiguration (off-line): The load of a new bitstream, which requires to reboot the device after loading the new design. This reconfiguration implies to paralyze the tasks in execution. It allows loading a new bitstream at the start.
– Dynamic reconfiguration (runtime): This is the modification of the design loaded in an FPGA at runtime. Additionally, it allows dynamic partial reconfiguration (DPR), modifying only a part of the design written in the previous load of the FPGA, while the rest of the design continues working without interruption.

It is possible to partially reconfigure the Zynq 7000 series from the ARM cores integrated in the SoC (or from Linux running on these cores), unlike the previous devices, in which the reconfiguration came from external memories.

Fig. 1 shows how partial dynamic reconfiguration works. The FPGA design keeps fixed interfaces while the different partial bitstreams of the same size (A1, A2,. . .) can be loaded in the reconfiguration field defined as A.

The bitstream loading mode varies depending on the used interface configuration, either parallel or serial; but the determining characteristic is whether the load is made from an internal processor or externally. For this project, the reconfiguration performed by the microprocessor is done through a PCAP interface (processor configuration access port) and an AXI (advanced extensible interface). In this way, it has to be taken into account the time to load will be directly proportional to the size of the bitstream.

3.2 Loading methods of bitstream

Partially reconfiguring an FPGA demands to access its memory configuration. The way to load dynamically a bitstream into a Xilinx FPGA can be done in 2 different ways [3]:

- Externally: It is the traditional method that requires to access physically to the pins of the device and it can be done with the JTAG (Joint Test Action Group) and through the Select Map interfaces. In addition, as it is performed externally, a PC or a microcontroller is needed to load the partial bitstream.
- Internally: It is done through the ICAP interface (Internal Configuration Access Port), the only one which allows a reconfiguration without accessing external pins. ICAP is accessible from software processors, like Microblaze, or from hardware processors, such as PowerPC or ARM. Different projects of partial dynamic reconfiguration from PowerPC (Recon OS, BEE2 based on the Linux kernel) or from Microblaze (Ullman, Blodget) have been carried out.

So far, current partial dynamic reconfiguration systems required a microcontroller or an external microprocessor to load the partial bitstream. Currently, the new Xilinx Zynq 7000 series chips have a PCAP interface (Processor Configuration Access Port) (Fig. 2), which allows connecting the ARM architecture to the FPGA, and reconfigure it from the ARM architecture. This PCAP interface can be considered as an evolution of the previous ICAP. Hence, running Linux over ARM allows to reconfigure the FPGA.

3.3 Zynq 7000 system architecture

The Xilinx Zynq 7000 platform includes in the same SOC an ARM Cortex-A9 dual-core processor in PS (processing system) with an FPGA in PL (Programmable logic). The PS part has a unit of device configuration (DevC) that allows configuring the PL part at startup and while the system is running. For this purpose, from PS and through the AXI bus (from ARM) partial/full bitstream is transferred to the DevC unit. This unit is connected to the PCAP interface on PL, from which the FPGA is reconfigured. DevC incorporates an AXI / PCAP bridge, which helps to convert messages from AXI bus to PCAP and vice versa (AXI is an ARM bus connected to PS). Consequently, AXI allows to communicate PS with PL while Linux is running, enabling the frame encapsulation control. The theoretical maximum speed of reconfiguration will be 3.2 Gbps when PCAP interface works at 100 MHz [1].

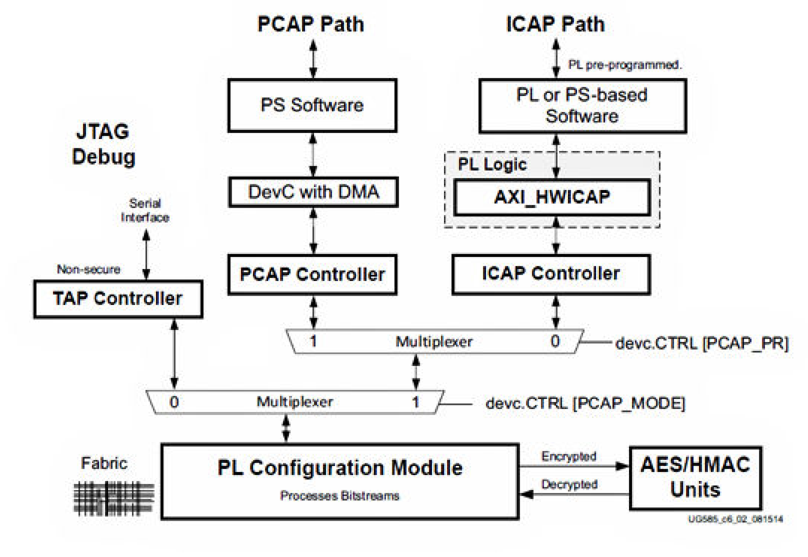

Fig. 2 PCAP in Zynq 7000 devices.

3.4 AXI on Zynq 7000

AXI (in its different versions) is the protocol of short-distance communication between microcontrollers. This bus interconnects PS and PL, which are integrated in the same SOC. AXI has been developed by ARM as an evolution of the AMBA bus, which provides microcontroller communication (advanced microcontroller bus architecture), launched in 1996. In 2003 AMBA, launched its version 3 and the first version of AXI, which was renamed as AXI 3 in 2010, when AMBA 4 and AXI 4 were presented.

ARM includes compatibility with this bus, providing bandwidth, synchrony and high frequency. Xilinx has made it compatible with their FPGAs, so the communication between the two chips contained in the SOC does not imply a limitation of performance. In the Zynq 7000 series (Fig. 3), AXI3 is used in PS and AXI4 in PL, compatible through the IP "AXI interconnect".

Having an operating system in PS, with an FPGA without communication limitations, allows the use of reconfigurable hardware in this application, providing TSN frames monitoring and the configuration of the device from Linux, once the design is loaded.

As can be seen in Fig. 3, the outputs and inputs to peripherals are multiplexed (MIO, Multiplexed input-output) and can be extended (EMIO, Extended MIO) to make them accessible from PL. From this GPIO (General Purpose input-output), the controller can provide 54 MIO (Multiplexed input-output) pins or 64 EMIO (extended MIO) pins. The pins of input/output can be routed for use in PL if required. However, in this case, the PCAP/AXI interface, through the DevC unit, will transfer bitstream from PS to PL, and some Ethernet ports will send and receive frames.

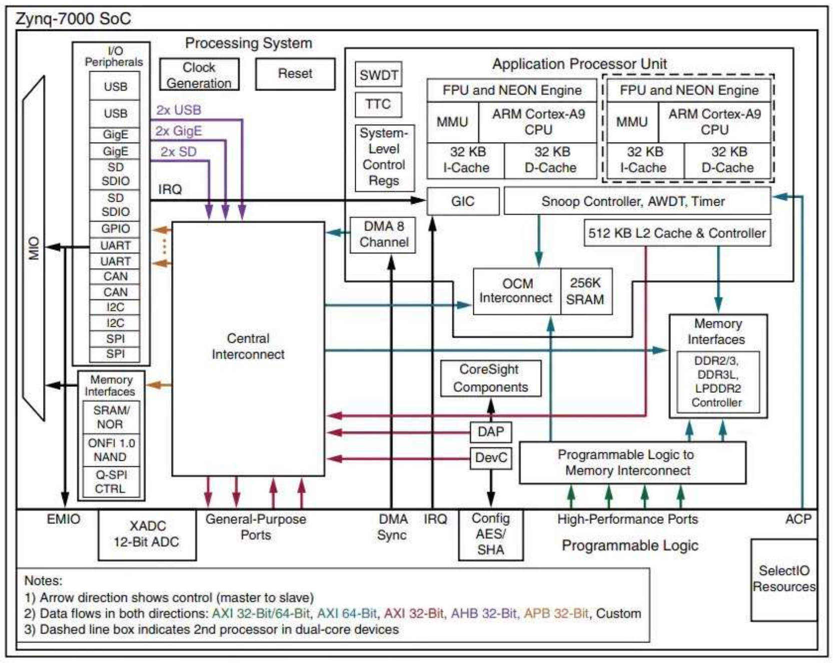

Fig. 3 Input output structure of Zynq 7000 devices.

3.5 PS-PL bitstream transfer

Xilinx manufacturer provides an operating system based on Linux kernel to run on ARM. From this, it starts the xdevcfg driver provided by Digilent manufacturer to allow a total or partial reconfiguration. For its operation, a virtual file system is used, and the configuration is done by writing the bitstream through AXI to a certain address in the Linux filesystem. Once the bitstream is transferred, it is necessary to wait an event indicating the end of the reconfiguration. The PL part is considered as reconfigurable hardware, a peripheral with its own address map. These addresses are connected to the PS AXI interface with a 100 MHz clock. The bitstream, which is generated and stored in the flash storage (SD card), is transferred to the PL part. Reconfiguration is done through PCAP, writing the bitstream to /Dev/xdevcfg [6].

3.6 Implementation

Once the bitstream has been transferred through the PCAP interface, the reconfiguration of the TSN frame encapsulation card is done. In this transmission, Linux system only knows, in addition to the bitstream, the addresses which should be typed into the AXI interface. This interface, through DevC, will write in PCAP. The driver used to access to these addresses is "Uio_pdrv". To configure the system and allow reconfiguration by the described methods it is necessary to follow two steps:

1. Bitstream generation: All the bitstream is necessary in addition to partial bitstreams, because the full bitstream contains the current layout, that is routed in the FPGA. This can be obtained using the tool PlanAhead. All bitstreams must be found in the SD, saved after conversion to binary format. This conversion is necessary for the driver configuration, to be able to read them.
2. Device tree of Linux: When the compilation and boot-up of Linux kernel is done, inside the device tree, the information of all the peripherals connected to the system boot is hosted. This may be a limitation, if it is not provided before loading the OS. To fix this, there is a possibility of including the device tree with the range of physical addresses that will be used in the Linux. This indicates in the I/O driver the assigned address ranged to the peripheral, so that it has read/write access to those physical addresses [5].

In addition, for control and communication with the FPGA from Linux, it is necessary to install a driver in the OS. This can be done in two ways:

- Integrating it into the Linux kernel, which can give problems, if you are not sure about what function will perform the partial reconfiguration. This way requires to recompile the kernel each new application that is integrated in the design.
- Operating as a kernel module. These modules can be added or removed at any moment. In this way, each partial bitstream loaded must have a kernel module associated with it. Core modules can be added from the kernel version of Linux 2.X. In our work, the Linux that Xilinx integrates can transfer a bitstream. However, it is necessary to provide Internet connectivity to the system, as well as the installation of Apache server on Linux.

3.7 Remote reconfiguration for TSN update

The device, in which the Zynq 7000 SoC is integrated, also enables the access to the internet through Ethernet ports. So, given that the Zynq 7000 platform works with Linux, it is possible to plug it into the network. For this purpose, packages that allow the self-assignment in Linux must be installed. Furthermore, it is possible to install tools to host a web server in the system. There are multiple alternatives, such as Apache or Nginx, which provide services under

IP protocol. These servers allow to store remotely the bitstream, through the driver and a script from Linux system, and to reprogram the FPGA, updating the TSN protocol integrated in it.

In this case, the Apache server is one of the options that has been installed on the Linux contained in Zynq7000. This web server contains a page where the bitstream should be loaded with the design modifications that are wanted to be implement in the FPGA. Once stored, a Linux script reconfigures the FPGA with the new updated design, and when required, it makes the FPGA compatible with the new TSN frames.

4 Frame encapsulation

4.1 General description

Our project aims to design an FPGA core that could be integrated into an industrial embedded switch module. The design's objective is to encapsulate TSN frames that are sent between two industrial machines in Ethernet frames, and redirect them into a PC that is connected to the Internet. In this way, TSN frames that would normally be discarded, by non specialized frame analyzers, will instead be received on a PC or a similar platform and extracted there. Such TSN traffic could later be analyzed by specialized software.

4.2 IP frame encapsulation

For this purpose, the IP has been integrated into the system, as described in Fig. 4, which shows the interconnection between PS and PL through an AXI lite bus interface. In addition, this will also allow to reprogram some of the parameters of the IP. On the other hand, the interface of communication used for frame trading is the RGMII (Reduced Gigabit Media Independent Interface) [11], which is an alternative to the Gigabit Media Independent Interface (GMII). The reduced interface lowers the number of needed pins for the communication, furthermore, it works at 1Gbit/s and fits industrial needs. Therefore, and to guarantee a consistent frequency, it's introduced a 125MHz clock that is connected to all modules.

The input of the frames to the IP is received through three signals. The first, RX_clk_i, is the communication clock signal that defines the frequency at which the frame is going to be received. The RX_data_i signal is a 4-bit signal that carries the data of the frame itself. The RX_data_i is synchronized with the clock signal and transmits a new nibble on each rising and falling edge. However, as the DDR (Double-Data-Rate) flip-flops are used at the input of the circuit, the data received from 4 bits at each clock event is converted into a data of 8-bits at every positive edge. Finally, the RX_en_i signal defines when frame transmission is enabled.

Once the signal is ready to be processed, the following points have to be carried out to complete the IP:

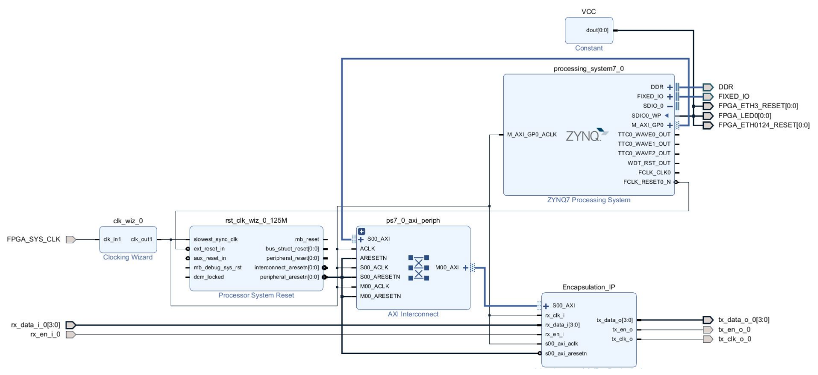

Fig. 4 Block diagram of the designed IP.

– A FIFO type stack, which will be storing the frame as it enters.
– Different registers with frame part information: Preamble, SFD (Start of Frame Delimiter), MAC destination, MAC source, EtherType/length and FCS (Frame Check Sequence).
– A circuit of parallel CRC-32 with a data input of 8-bits for the calculation of the CRC that goes into the new frames FCS.
– A counter that counts until the inter-frame gap is completed.
– A state machine that controls the transmission of each part of the frame and the CRC calculation.

The encapsulation process occurs as follows: initially, the IP is in a resting state and as soon as a new frame arrives, it begins to load it on the stack. Simultaneously, the preamble and SFD begin to be transmitted. But, for this process to fulfill the needs of RGMII, the inverse conversion of the one performed on entry must be performed, whereby a signal of 8-bits every rising edge is converted into a 4-bits signal that transmits every clock event.

Once the SFD's last bit is sent, MAC and Type/length headers begin to be transmitted. Simultaneously, the CRC-32 begins to be calculated, as they have to take these headers, as well as the consequent payload, into account. For this model, a parallel CRC-32 design is chosen, which allows calculations to be carried out during a single clock cycle [12].

Following this, the stack with the received frame begins to be read and transmitted by TX_data_o, which specifies the content of the payload of the new frame.

Once the full stack is empty, the CRC calculation is done and the FCS is sent. Once this has ended, the transmission is completed and a countdown of the interframe gap will start. Finally, when all the processes have been completed, the model returns to the initial resting state while waiting for a new frame to be encapsulated and sent.

4.3 Simulation and results

The performed simulation (Fig. 5) contains the input signals created in the testbench; RX_clk_i signal (set to 125 MHz), a TSN frame (RX_data_i), and a frame transmission enable signal RX_en_i. As well as the signals obtained through the IP; an Ethernet standard frame, which has as payload the entire TSN frame (TX_data_o), the clock (TX_clk_o, set to 125MHz) to synchronize the data and the enabling signal of the Ethernet frame transmission.

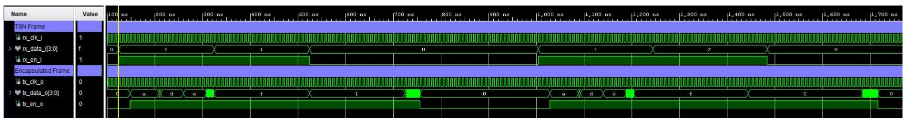

Fig. 5 Simulation results.

The simulation shows a successful result with a response delay on arrival of only 3 clock cycles.

In addition, our simulations identify a couple of limitations. Firstly, as the sent frame is always longer than the received one, some issues on transmissions with low latency might occur, as it will be impossible to respect the interframe gap. The second identified limitation regards the maximum length of the frame to be encapsulated, as this cannot be larger than the payload's maximum length.

The resources used for the implemented IP in Zynq 7000 have also been analyzed. The FIFO stack has been described in such a way that it is implemented in a BRAM memory, thus saving resources while creating a simpler circuit. As a result, we obtained a circuit design with a small percentage of resource utilization.

5 Conclusions

Having analyzed the state of the art for TSN as well as for remote dynamic reconfigurations, it can be concluded that the FPGAs are a possible solution for the challenges arising out of continuous standard renovation, due to the flexibility and reconfiguration options that these offer. Other devices such as PCs and non-advanced frame analysis devices are not able to process TSN as it continues to evolve, which may result some frames being discarded by such devices as if they were corrupted.

This paper describes a design based on FPGA that encapsulates TSN traffic in standard Ethernet packets and redirects it to the PC or other devices, as a way to process and analyze TSN frames. This creates a solution that is able to manage and analyze TSN frames in non-specialized frame analysis devices.

The leaps in evolution that remote reconfiguration technology has made during the last years have made it suitable even for real-time applications, such as TSN processing.

The IP design created for this paper, shown in the recourse results, is simple enough to make it useful for a partial remote reconfiguration. In addition to this, as it is built in an AXI lite interface, it's possible to reconfigure parameters from the IP. In this way, the source and destiny MAC directions can be changed if a redirection of the frame transmission is needed.

Despite the limitations of the described design; i.e. the fact that the frame to be encapsulated cannot be bigger than the payloads maximum size, we do not foresee any major problem with the implementation of our design, as industrial TSN communication frame are typically relatively short [13]. Nevertheless, the latency limitation shown through our simulation should be something to take into account; and could be a reason for future study.

Acknowledgements This work has been supported by the Ministerio de Economía y Competitividad of Spain within the project TEC2017-84011-R and FEDER funds, Doctorados Industriales program DI-15-07857 and by the Department of Education, Linguistic Policy and Culture of the Basque Government within the fund for research groups of the Basque university system IT978-16.

Declarations

- Funding: This work has been supported by the Ministerio de Economía y Competitividad of Spain within the project TEC2017-84011-R and FEDER funds, Doctorados Industriales program DI-15-07857 and by the Department of Education, Linguistic Policy and Culture of the Basque Government within the fund for research groups of the Basque university system IT978-16.
- Conflicts of interest/Competing interests: On behalf of all authors, the corresponding author states that there is no conflict of interest.
- Availability of data and material: Not applicable
- Code availability: Not applicable

References

1. Bello, L.L., Steiner, W.: A perspective on IEEE time-sensitive networking for industrial communication and automation systems. Proceedings of the IEEE **107**(6), 1094–1120 (2019). DOI 10.1109/jproc.2019.2905334. URL `https://doi.org/10.1109%2Fjproc.2019.2905334`
2. Brooks, S., Brooks, S.: Time-sensitive networking: From theory to implementation in industrial automation. Tech. rep. URL `https://www.intel.com/content/dam/www/public/us/en/documents/product-briefs/wp-01279-time-sensitive-networking-theory-to-implementation-in-industrial-automation.pdf`
3. Kadi, M.A., Rudolph, P., Gohringer, D., Hubner, M.: Dynamic and partial reconfiguration of zynq 7000 under linux. In: 2013 International Conference on Reconfigurable

Computing and FPGAs (ReConFig). IEEE (2013). DOI 10.1109/reconfig.2013.6732279. URL `https://doi.org/10.1109%2Freconfig.2013.6732279`

4. Lu, Y.: Industry 4.0: A survey on technologies, applications and open research issues. Journal of Industrial Information Integration **6**, 1–10 (2017). DOI 10.1016/j.jii.2017.04.005. URL `https://doi.org/10.1016%2Fj.jii.2017.04.005`
5. Machidon, O., Sandu, F., Zaharia, C., Cotfas, P., Cotfas, D.: Remote SoC/FPGA platform configuration for cloud applications. In: 2014 International Conference on Optimization of Electrical and Electronic Equipment (OPTIM). IEEE (2014). DOI 10.1109/optim.2014.6850986. URL `https://doi.org/10.1109%2Foptim.2014.6850986`
6. Melo, R.A., Valinoti, B., Amador, M.B., Garcia, L.G., Cicuttin, A., Crespo, M.L.: Study of the data exchange between programmable logic and processor system of zynq-7000 devices. In: 2019 X Southern Conference on Programmable Logic (SPL). IEEE (2019). DOI 10.1109/spl.2019.8714328. URL `https://doi.org/10.1109%2Fspl.2019.8714328`
7. Silva, L., Pedreiras, P., Fonseca, P., Almeida, L.: On the adequacy of SDN and TSN for industry 4.0. In: 2019 IEEE 22nd International Symposium on Real-Time Distributed Computing (ISORC). IEEE (2019). DOI 10.1109/isorc.2019.00017. URL `https://doi.org/10.1109%2Fisorc.2019.00017`
8. Wollschlaeger, M., Sauter, T., Jasperneite, J.: The future of industrial communication: Automation networks in the era of the internet of things and industry 4.0. IEEE Industrial Electronics Magazine **11**(1), 17–27 (2017). DOI 10.1109/mie.2017.2649104. URL `https://doi.org/10.1109%2Fmie.2017.2649104`