Lynx: An Efficient and Flexible Communication Schema for On-Chip and Off-Chip Applications in Distributed Static Industrial Networks

Hipólito Guzmán-Miranda, *Senior Member, IEEE*, and Abraham Marquez Alcaide, *Member, IEEE*

***Abstract*—In this article, an efficient and flexible communication schema for on-chip and off-chip communication is presented. With a reduced header size, the proposal can achieve high communication efficiency among the devices of a distributed static network or the internal components in a system-on-chip. In addition, the proposal is very versatile in terms of network configurations and topologies. An FPGA implementation of the proposal has been developed and thoroughly verified, both in simulation and when implemented in two different FPGA devices. The prototypes are exhibiting good results which validate the expected performance, with an efficient use of device resources. As a result, the proposed schema can both improve existing applications whose performance is affected by communication overhead and enable new industrial applications that require efficient communications in distributed static networks.**

***Index Terms*—Efficient communications, FPGA, industrial networks, network on chip, system on chip.**

I. Introduction

IN MODERN industrial applications, communication protocols play a crucial role in enabling efficient and reliable data exchange between distributed devices and systems. Traditionally, complex control systems are based on one main controller that manages many secondary controllers, which act over the rest of the subsystems. In this sense, it is possible to affirm that the data flow is unidirectional, that is, from the main controller to secondary controllers and vice versa, but no interaction between secondary controllers is allowed. A good example of this is the electrical grid where at the beginning the commands and orders were given from the energy sources to the loads, but with the inrush of the smart-grid paradigm, this frontier is not so clear anymore [1], [2]. In this scenario, academia and industry support and explore the use of decentralized and coordinated control schemes where the communication protocol plays an essential role [3], [4], [5].

With the advent of Industry 4.0, the importance of such communication protocols has only increased, as smart factories and automated supply chains become increasingly commonplace [6], [7]. One of the key challenges in Industry 4.0 is the need for active maintenance and prognosis, which requires continuous monitoring and analysis of various data streams to detect anomalies and predict potential failures [8]. To this end, the digital twins and cyberphysical systems artifacts are used. This, in turn, requires a highly efficient and flexible communication protocol that can support real-time data transmission and processing across distributed networks [9].

In addition to these challenges, there is also a growing trend toward modularization and standardization of industrial systems. A good example is found in the power electronics field where the main manufacturers are investing time and resources to develop power converters following the modular power electronics building block paradigm [10]. This paradigm advocates the use of standardized power electronics modules that can be easily integrated and reconfigured to meet specific industrial requirements, allowing greater flexibility in industrial systems [11], [12], [13]. Fig. 1 shows a modern electric vehicle charging station (EVCS), which is an example application of this new power electronics design paradigm.

Also, with programmable system-on-chip devices and solutions appearing across many different industries and fields [14], the necessity of having efficient communication between different modules extends also to the inside of the chips [15]. Efficient and minimalist network-on-chip protocols are typically niche-specific and limited in practice, for example the SocWire Protocol [16] is tailored to the usage of reconfigurable modules in space applications, but is limited to a maximum of three switches per network, with a maximum of 16 ports per switch [16], [17].

The technical literature and the market portfolio present an extensive catalog of communication techniques based on the

Manuscript received 24 November 2023; revised 1 March 2024 and 10 June 2024; accepted 17 June 2024. Date of publication 17 July 2024; date of current version 5 November 2024. This work was supported in part by the MCIN/AEI/10.13039/501100011033 under Grant PID2020-115561RB-C31 and Grant TED2021-130613B-I00, in part by the 2021 Call for Grants aimed at research and development projects in artificial intelligence and other digital technologies and their integration into value chains from the Recovery and Resilience Facility, European Recovery Instrument (Next Generation EU), within the framework of the "PS/DS" project under Grant C005/21-ED, in part by the Andalusian Technological Corporation (CTA), and in part by ANID under Grant ANID/FONDAP/1523A0006 SERC Chile. Paper no. TII-23-4717. *(Corresponding author: Hipólito Guzmán-Miranda.)*

The authors are with the Department of Electronic Engineering, Universidad de Sevilla, 41092 Sevilla, Spain (e-mail: hguzman@us.es; amarquez@ieee.org).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2024.3422373.

Digital Object Identifier 10.1109/TII.2024.3422373

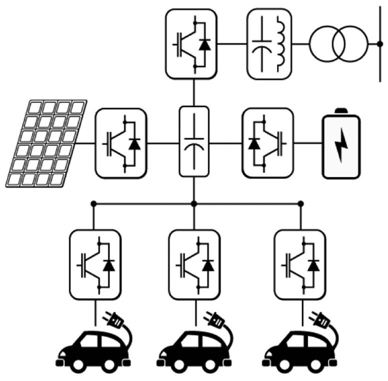


Fig. 1. Example of complex industrial system where the proposal would have a positive impact. DC-coupled EVCS including solar photovoltaic harvesting and energy storage system.

OSI stack, as well as the Internet stack [6]. For instance, communication methods, such as CAN [18], [19], Modbus [20], and TCP/IP, are very popular communication mechanisms in many industrial applications [6], [21], [22]. These protocols are very flexible since they allow multiple communication modes as well as other features, such as data integrity and data dispatching order, among others.

A. Common Extra Features in Industrial Communications

Each communication technique has its advantages and drawbacks from an industrial application point of view, and the selection of a communication schema for a specific environment is not a trivial task. In general, because of their extra features, these protocols and communication stacks present long header fields that limit the communication bandwidth as well as the data exchange efficiency.

Real-time communication protocols are designed to ensure timely and reliable data transmission in applications where timeliness is critical, such as control systems, robotics, and automation. These communication protocols are designed to transmit data in a specific time frame by time-division multiplexing or by using a priority-based system to ensure that high-priority data are transmitted first [21], [22], [23], [24].

In addition, low-latency communication is also a required feature in real-time communication. This is achieved through techniques, such as data compression, packet prioritization, and low-level optimizations to reduce processing time [23], [24].

Real-time communication protocols often incorporate fault-tolerant mechanisms to ensure that data transmission is reliable, even in the presence of network failures or other disruptions. This may include techniques, such as redundancy, error correction, and retransmission of lost packets [23], [24], [25].

Finally, these communication protocols should be scalable to support a large number of devices and high data rates. This is achieved through techniques, such as multicast communication, where data are transmitted to multiple devices simultaneously, and by using protocols that can handle large amounts of data traffic without degrading performance [21], [24].

In the case of [21], [22], [23], and [24], these protocols possess the mentioned characteristics, although one of them is proprietary [24] and only accessible under expensive licensing terms, and two of them require expensive hardware to be implemented [23], [24]. In the case of [24], its usage is intended for laboratory experiments and demonstrators, but not for actual deployment of industrial applications.

B. Need for an Efficient Communication Schema

While the aforementioned features are of interest for many applications, their implementation typically requires increasing the overhead of the protocols, which in turn increases communication delays and reduces their efficiency in utilizing the physical mediums, while also increasing their general complexity, their difficulty of implementation, and the resources they occupy in FPGA or ASIC devices. Also, some of the solutions mentioned in the previous section require specific hardware, such as optical fiber transceivers to reach the desired data transmission rates. In some cases, the selection of protocol forces a specific topology which may not be the desired one or may insert additional delays in the intended application. Moreover, some topologies such as ring, daisy-chain, or any topology that depends on a single master agent are especially vulnerable to failures.

For smaller, ad hoc networks, for example the ones that naturally appear inside a distributed power converter, all the aforementioned features introduce so much overhead that the feasibility of the network to be able to send the required data through it during a critical time interval (for example, the sampling time of the power converter) is put into question. Thus, it would be desirable to have a protocol with a very reduced overhead that could be quickly deployed for these *ad hoc* distributed static networks.

On the one hand, distributed computing may reduce the bandwidth requirements of specific applications due to the reduction of the quantity of information to be sent, since there is no central controller that needs to receive all the raw data because the processing capability is distributed among the agents. But, on the other hand, latency will always be a problem when real-time control is desired due to it depending exclusively on the path, the processing capability of the intermediate routing devices, and the amount of data to transmit, including protocol headers. This article presents a communication schema that solves the aforementioned issues, and its FPGA implementation. Since the schema is designed for static networks, its protocol overhead is automatically optimized according to the network parameters selected by the user.

Furthermore, the proposed communication protocol can also be used in on-chip communication applications, where efficient and reliable communication is essential to ensure the performance and reliability of complex systems-on-chip [26]. By utilizing a lightweight header format and a decentralized routing mechanism, the proposed protocol offers a highly efficient and flexible solution to transmit data within and between different components of a system-on-chip.

Through a series of experimental evaluations and simulations, the effectiveness and performance of the proposed

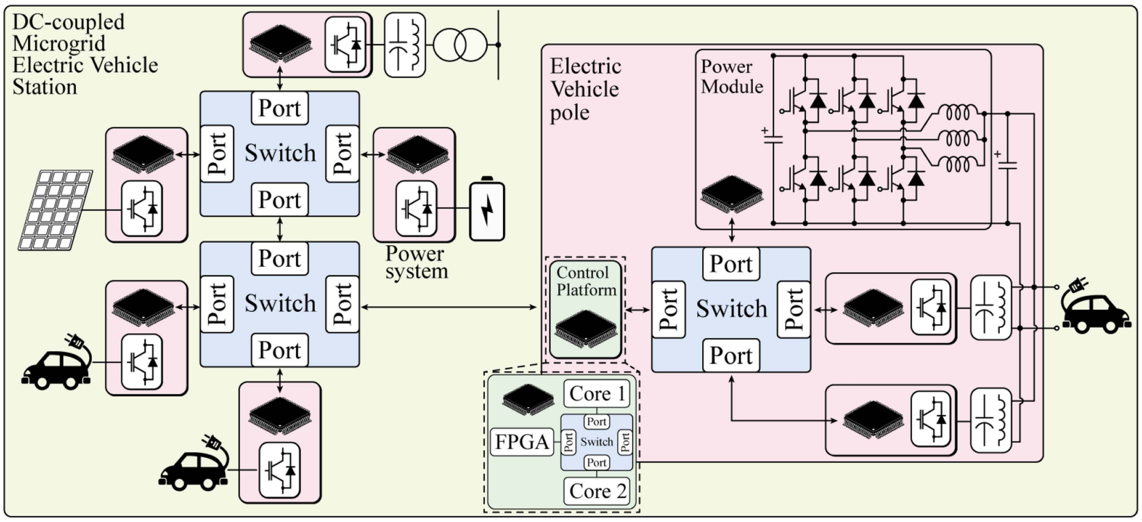


Fig. 2. Possible implementation of the EVCS of Fig. 1 using the proposed approach.

protocol in both industrial on-chip and off-chip communications are also demonstrated. Specifically, it is shown that the proposed protocol can enable efficient and reliable communication between different components of a distributed network or a system-on-chip, even under challenging network conditions and topologies.

Fig. 2 shows how the proposed approach could be used to implement the next-generation EVCS shown in Fig. 1. Lynx switch devices are drawn in blue, whereas power converters with their corresponding control platforms are drawn in red. Complex control platforms based on multicore CPU, FPGA, and/or SoC are drawn in green. In the figure, several examples where the proposed communication approach could be successfully used are illustrated. It must be noted that the communication networks shown could be, or not, isolated (this would depend on the final application). The isolation of the communication network allows to provide extra features to the industrial application, such as security, firewalling, or fault tolerance. In addition, the proposal also enables the communication of the devices inside networks-on-chip (NoC) or networks-on-board (NoB) with other modules outside the chip or board.

In summary, the proposed communication scheme is multipurpose and versatile. In particular, the proposal enables effective communications in the following scenarios, as shown in Fig. 2 (from highest to lowest communication level).

1) Communication exchange among different systems in a network. Off-chip communication by using the appropriate *frontend* (off-chip communication protocol, see Section III-A). This is the case of a conventional static industrial network.
2) Communication exchange among different elements within the same board by using the appropriate frontend. This is the case of the NoB.
3) Communication exchange in a hybrid context that means some agents are located inside the SoC and other agents are located outside of the SoC. This case can be found in NoB for instance.
4) Communication exchange on-chip. All the agents involved in the network are located inside the same SoC or NoC.

In general, the proposed communication schema offers a promising solution for addressing the challenges of communication in modern industrial and on-chip applications. With its high efficiency, flexibility, and scalability, the proposed protocol has the potential to significantly improve the efficiency, reliability, and performance of distributed systems in a wide range of domains.

II. Proposed Approach

In general, communication protocols usually have long headers, containing the multiple fields required for the implementation of the selected communication schema and other features that the protocol may provide, such as those discussed in Section I. However, such long headers are often not practical for distributed applications that send small amounts of data a high number of times per second, for example, in distributed power converters [27], [28]. For these kinds of applications, it commonly occurs that the header itself greatly exceeds the size of the useful data of the message, which wastes available bandwidth capacity and makes real-time communications difficult or impossible when the number of elements in the network increases.

In this article, an efficient and flexible communication protocol specifically designed to reduce overhead size is proposed. This communication protocol targets communication in statically defined networks, meaning networks with constant topology. Flexibility is achieved by making the protocol configurable according to the size of the network. Efficiency is achieved by reducing overhead to the minimum necessary to route the information within the specific network. Depending on the chosen configuration, the protocol headers have a size that varies between a minimum of 2 (for smaller networks) and a maximum of 5 bytes (for bigger networks).

The protocol defines the following terms.

Frame: A *frame* is the basic unit of information that can be transmitted by the protocol. This means that no less than a *frame* can be transmitted. *Frame* size is measured in bytes, which means that the number of total bits in each frame must always be a multiple of 8.

Module: A *module* is an end agent in the communication. This means that *modules* are the only devices who produce and consume data during normal operation.

Switch: A *switch* is an agent that routes *frames*. A *switch* routes *frames* by sending them to other elements in the *network* (either *switches* or *modules*). During normal operation, a *switch* can neither produce nor consume data: it must always output the exact same number of bytes it receives.

Network: A *network* is a group of agents that will send *frames* between them following this protocol.

Source: The *source* of a *frame* is the *module* from which a *frame* originates.

Destination: The *destination* of a *frame* is the *module* to which a *frame* should go.

Jump: A *jump* happens when a *frame* goes from one agent to another. For example, a *jump* occurs when a *frame* goes from a *module* to a *switch*, from a *switch* to a different *switch*, and finally when it goes from a *switch* to its *destination*.

Path: A *path* consists of a sequence of *jumps*. A *path* is used to route a *frame* through the *network* from a specific *source* to the desired *destination*. From a given *source*, a given *path* will always lead to the same *destination*. The same *path*, if starting on a different *source*, is not guaranteed to lead to the same *destination*. This means that *paths* are not absolute addresses to specific *modules:* they are relative to the position of the *source* in the network.

It should be noted that this protocol does not use addresses, but instead uses *paths*. This is by design, as this simplifies the computing cost of routing *frames*, making the *switches* and *modules* both simpler and faster. Since this protocol does not have absolute addresses, this means that the same *module* will have different *paths* leading to it, depending on who is sending the data to it. In *networks* with any kind of redundancy, there could be more than one possible *path* between a *source* and a *destination*. Since networks are static, this makes no difference for each *source* module: instead of storing an address for each of the *modules* to which it wants to send data, the *module* must store a *path*.

A. Frame Fields

The frame structure of the proposed protocol can be seen in Fig. 3. A *frame* in this protocol consists of two fields: the *header*, which contains the information to route the *frame* through the network, and the *data*, which is the information the modules want to transmit and receive. The size of the *header* is statically fixed, which means that it depends on the specific configuration of the protocol and cannot be changed after synthesis. This allows optimizing the *header* size to the minimum size necessary to properly route *frames* in the specific configured network. The size of the *data* field is dynamic, since it is specified in a sub-field of the *header*, so different *frames* may carry *data* of different sizes in the same network.

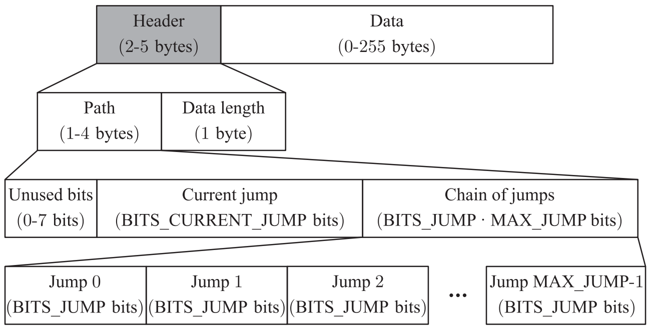


Fig. 3. Frame structure. The fields take the smallest necessary size to represent the possible jumps in the network, according to the number of ports of each switch and the maximum number of jumps a frame may perform.

At the same time, the *header* contains itself two fields: *path*, which contains the jump-by-jump route that the *frame* should follow, and *data length*, which is the length, in bytes, of the *data* field.

The *path* itself is also further subdivided into three fields: *unused bits*, which just pad the size of *path* to a multiple of 8 bits, *current jump*, which is used to keep score of how many *switches* the *frame* has passed through, and *chain of jumps*, which contains the sequence of jumps the *frame* should perform to arrive at its *destination*.

In this way, a *switch* does not need to know any information about the network to route a received *frame:* it can just read the *current jump* field, then use it as an index to select a specific *jump* from the *chain of jumps* field. The value of the selected *jump* contains which output port the *switch* should send the *frame* to. This also allows for very fast processing of the *frames*, increasing switch throughput.

It should be noted that a *frame* does not need to perform all *jumps* allowed by the *chain of jumps* field, if it reaches its *destination* before performing all possible *jumps*. For example, if two *modules* are connected to the same *switch*, these two *modules* can reach each other in a couple of *jumps*, even if the network allows for more.

B. Return Path

The reader may have noticed that the *frame* does not have any fields in which to note the *source* of the *frame*. In this case, how would the *destination* know which *module* originated the transmission? How would it know where to send its response, if the user application needs one to be sent? A naive solution to this problem would be to just add another field to the *header* that would identify the *source* of the *frame*, but that would impose a very inconvenient overhead: the *header* would almost duplicate its size, since we would need to add another field with the same size as *path*.

For this work, it is proposed that the information to identify the *source* is stored in the *path*, overwriting it, since the *destination* does not need to know how other *modules* actually reach it. The most important condition that must be met for this approach to be valid is to never overwrite any information that will be needed to route the *frame* towards the *destination*. This is achieved by overwriting the *jumps* that have already been performed. Each *switch* knows from which of its ports a *frame* came, so after getting the value of the output port to which to send the *frame* (determined by the *jump* in position *current jump* inside the *chain of jumps*), it can just overwrite that *jump* with the index of the input port from where the *frame* came. Since the switch will also increment the *frame*'s *current jump* by one, the overwritten *jump* will never be used by the *switches* that follow.

This modified *path*, that the *destination* sees in the received *frame*, is called the *return path*. The *destination* could now reverse this *return path* to determine which is the *source* of the *frame*, but it is typically more convenient to just store both the *path* to and the *return path* from the *modules* from which it expects to communicate with. If the data are correctly structured (for example, in a table), the *module* can look up the received *return path* to get the direct *path* for the same network element. Since increasing the *header* size would increase overhead throughout all the network, but duplicating path storage in the *modules* should be no issue, this is considered an acceptable tradeoff in this context. In the general case, each *module* must store two 1-to-4 byte *paths* per each *module* it wants to communicate with.

Of course, switches do not need to store any *paths*, since all the information needed to route *frames* through the correct output port is contained in the *header*.

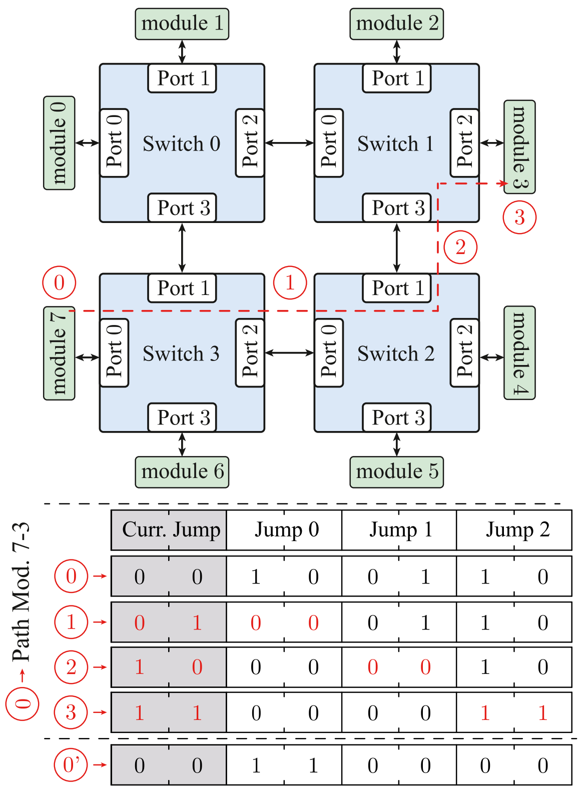
Fig. 4 shows a network with four four-port *switches*, over which an example is illustrated. The example assumes a maximum number of *jumps* of 4, since four *jumps* are enough to route a *frame* between each pair of *modules* in the diagram. In the figure, path 0 is the path to route a *frame* from *module* 7 to *module* 3, while paths 1, 2, and 3 show the modifications of the original *path* as the *frame* traverses the network. Finally, reversing path 3, which is the *return path* in this example, results in path 0' which allows *module* 3 to send data back to the sender if required. As it will be explained in Section III-A, only three jumps appear in the *paths* since the initial jump from *module* 7 to *switch* 3 does not need to be encoded.

A high number of agents would lead to an increase in transmission delays across the network and longer overheads to store very long *paths*, which is why the protocol limits the *header* size to 5 bytes (4 bytes of *path* and 1 byte of *data length*). Nevertheless, it must be noted that, when configured for example with eight-port switches and ten maximum jumps, a network could have thousands of modules, depending on the exact topology.

C. Impact of Header Size on the Communication Mechanism

It is possible to model a *switch* as a network of queues, as it is shown in Fig. 5. Let λ_i be the frame arrival rate for each input port, and μ_j be the service rate for each output port, whereas μ_a represents the service rate of the arbiter. The arbiter processes just the header of every frame and then connects the input queue to the relevant output queue, but from a queuing standpoint it is possible to consider that the complete message is quickly processed and sent to an output queue. From this queuing point of view, the arbiter splits the input flows according to the probability, p_j, that a frame is destined to a specific output port j.

While service times are commonly modeled using exponential probability distributions, the arbiter service time T_a for the proposed approach depends only on the clock frequency and the header size, and thus is deterministic, as will be shown in Section IV. Thus, $T_a = 1/\mu_a$. In contrast, the service times T_j of the output queues will depend on the clock frequency and the full frame size, so they are not deterministic. A typical assumption is to consider that frame sizes are exponentially distributed,



	Curr. Jump		Jump 0		Jump 1		Jump 2	
0	0	0	1	0	0	1	1	0
1	0	1	0	0	0	1	1	0
2	1	0	0	0	0	0	1	0
3	1	1	0	0	0	0	1	1
0'	0	0	1	1	0	0	0	0

Fig. 4. Network with four-port switches. Four jumps are enough to route a frame between each pair of modules.

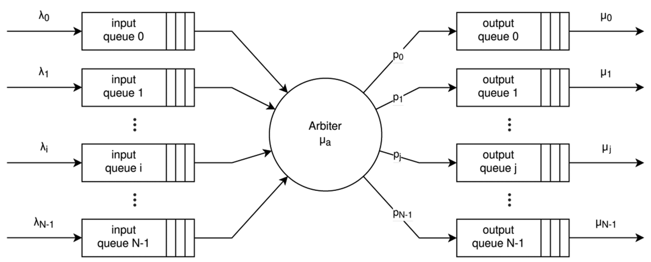


Fig. 5. Switch modeled as a network of queues. The arbiter processes just the header of every frame and then connects the input queue to the relevant output queue.

and thus T_j will also follow an exponential distribution, with service rate parameter μ_j, and mean service time $1/\mu_j$.

Assuming the input arrivals can be modeled as independent Poisson processes, we can calculate the total arrival rate to the arbiter as

$$\lambda_{\text{total}} = \sum_i \lambda_i. \tag{1}$$

Since the arrivals are determined by a Poisson process and the service time of the arbiter is deterministic, it is possible to consider the first part of the switch as an M/D/1 queue (Markov arrivals, deterministic service time, single server), with input arrival rate λ_{total} and service rate μ_a. Defining the utilization factor as $\rho_a = \lambda_{\text{total}}/\mu_a$, the average time spent by a frame in the first part of the switch is

$$T_1 = \frac{1}{\mu_a} + \frac{\rho_a}{2\mu_a(1-\rho_a)}. \tag{2}$$

When the utilization ρ_a is high, the output of this first part of the switch can be approximated to a Poisson distribution, which is then split by the arbiter into N flows, according to the probabilities p_j. The resulting flows will also be approximate Poisson processes with arrival rates $\lambda_j = p_j\lambda_{\text{total}}$. Each of those flows goes to an M/M/1 queue (Markov arrivals, Markov service time, single server), where the average time spent by a frame is as follows:

$$T_2 = \frac{1}{(\mu_j - \lambda_j)}. \tag{3}$$

These two times can be combined to estimate the average time a frame spends inside the switch, assuming it is is routed to the output queue j

$$T = T_1 + T_2 = \frac{1}{\mu_a} + \frac{\lambda_{\text{total}}/\mu_a}{2\mu_a(1-\frac{\lambda_{\text{total}}}{\mu_a})} + \frac{1}{\mu_j - p_j * \lambda_{\text{total}}}. \tag{4}$$

The header size affects multiple elements of the previous equation: a smaller header size increases the rate at which the arbiter can process frames, increasing μ_a, which appears in the denominator of the two first terms of (4). The first term expresses the actual time spent by the arbiter processing the frame headers; whereas the second term expresses the time the frame spends waiting in the input queues for the arbiter to process it. But also, a smaller header size affects the rate at which the output queues can process frames—since these output queues must also transmit the full headers—, thus increasing μ_j, the service rate of the output queues.

On the other hand, the required transmission time in a real-time communication scheme is determined by the amount of data to be transmitted and the baud-rate of the physical links. For the sake of simplicity, an example is illustrated in the following paragraphs.

Assuming the need for a real-time communication procedure between two machines, A and B, in a network connected by a switch device, as shown in Fig. 6, and considering a target communication time constraint T_{obj}, the required communication time to send a message from A to B can be expressed as

$$T_A^B = T_b^A(H_A + D_A) + T_b^A(H_A) + \delta_1$$
$$+ T_b^B(H_A + D_A) + T_b^B(H_A) + \delta_2 \tag{5}$$

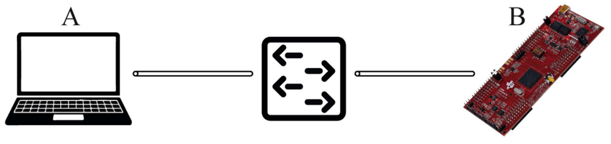


Fig. 6. Example of communication topology used to determine the minimum transfer for a real-time application.

where terms $T_b^A(H_A + D_A) + T_b^A(H_A) + \delta_1$ represent the time to transmit from A and the computation time required in the switch device to decide the next step. In a similar way, the terms $T_b^B(H_A + D_A) + T_b^B(H_A) + \delta_2$ represent the data transmission from the switch device to the final destination and the computation time required to process the message. H_A and D_A represent the header and data size expressed in bytes, respectively. T_b^X is the byte-time in the physical link and δ_x is a guard time to ensure real-time communication.

In a similar way, the required communication time for the response from B to A can be expressed as

$$T_B^A = T_b^B(H_B + D_B) + T_b^B(H_B) + \delta_3$$
$$+ T_b^A(H_B + D_B) + T_b^A(H_B) + \delta_4 \tag{6}$$

following the same notation.

Then, to guarantee real-time communications, the complete communication process should be completed inside the target time constraint, that is

$$T_A^B + T_B^A \leq T_{\text{obj}}. \tag{7}$$

In addition, for the sake of simplicity, we can assume header sizes are the same ($H_A = H_B = H$), and that the transmission time in both links can be considered equal ($T_b^A = T_b^B = T_b$), as well as the computational times $\delta_x = \delta$, then it follows:

$$T_b(8H + 2(D_A + D_B)) + 4\delta \leq T_{\text{obj}}$$
$$T_b \leq \frac{T_{\text{obj}} - 4\delta}{8H + 2(D_A + D_B)}. \tag{8}$$

It is clear from (8) that to fulfill the communication time (T_{obj}) for a particular industrial application only two actions can be performed: either increase the transmission rate or reduce as much as possible the data to transmit. Assuming the data to transmit is already optimized, only the header size could be reduced.

This discussion is general, but does not include the details of any particular communication protocol. For instance, in CAN-bus the size of the frame is set to 8 bytes, where the payload is limited to 48 bits.

III. FPGA Implementation

A. Design Architecture

A key decision in the architecture has been to define a clear separation between the on-chip and off-chip functionalities. For this, a simple data/valid/ready entity interface has been defined for internal communication purposes. In this way, the proposal

```
package User_Constants is
  constant Max_Number_of_Jumps: integer:= 6;
  constant Number_of_Ports: integer:= 4;
end User_Constants;
```

Fig. 7. Constants for protocol configuration are set in a single VHDL file which is just four lines long. All internal constants for the specific protocol configuration are automatically derived, from these ones, in synthesis time.

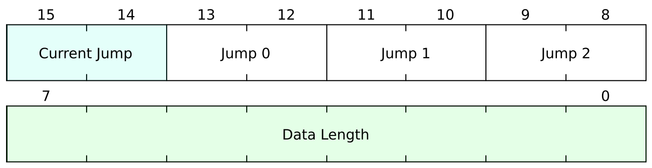


Fig. 8. Protocol header when configured for a maximum of four jumps and four-port switches. In this case, the size of the header is the minimum possible, 2 bytes.

can be used for on-chip communications, where a network-on-chip could be implemented by instancing and connecting one or more switches in a single FPGA device, while off-chip communication requires the use of *frontends*, which convert from the internal data/valid/ready interface to a protocol more suitable for communicating with the outside of a chip, such as UART or SPI. By defining a standardized interface, frontends can be swapped without modifying the internal behavior of the switch entities. Also, new frontends can be developed in order to support different off-chip communication protocols, and the only requirement is that they support the data/valid/ready interface.

Another key decision in the architecture has been to make the protocol as easy as possible to configure. To configure the size of the network, the user just has to decide how many *jumps* can a *frame* perform through the network, and how many *ports* each switch in the network will have. While using VHDL generics would be the default strategy to achieve this configurability, the complexity of the design, which requires defining custom datatypes with sizes that depend on the number of *jumps* and *ports*, implies that using VHDL generics would require the code to be written in VHDL-2008 [29]. Due to the status of the current landscape of both proprietary and free and open-source (FOSS) tools for FPGA design and verification [14], [30], it is clear that, in order to achieve maximum portability of the design, it is much desirable to code the design in the VHDL'93 [31] version of the standard. Thus, the decision was made of defining a package with only the user-settable constants, which can be very easily modified by the user, without requiring any knowledge of the VHDL language.

Fig. 7 shows the contents of this package. This approach does not cause any loss of generality, because all elements in the same network will use the same value for *Max_Number_of_Jumps* and *Number_of_Ports*.

From these two user-defined constants, all other sizes and internal constants are statically derived in synthesis time. In the case that the user specifies constants that would result in a *header* longer than 5 bytes, an assertion warns the user and prevents simulation and implementation.

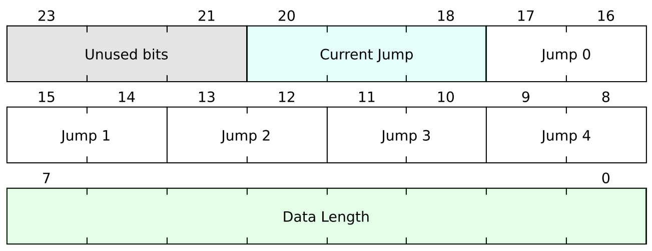


Fig. 9. Protocol header when configured for a maximum of six jumps and four-port switches.

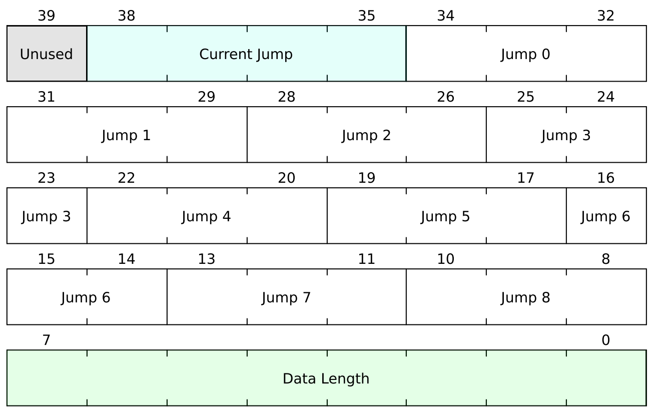


Fig. 10. Protocol header when configured for a maximum of ten jumps and eight-port switches. In this case, the size of the header is the maximum allowed, 5 bytes.

In Figs. 8–10, it is shown how the header changes for different configurations. It must be noted that the initial jump that occurs from the *source* to the first *switch* in the *network* does not need to be encoded in the header, since there is no possible ambiguity in that operation: the *module* is directly connected to the *switch* and it decides to send the *frame* to it. Even in the case of a module being connected to more than one switch (for example, a module that acted as a bridge between two Lynx networks), there is still no ambiguity since it would be connected to each switch through a different interface. By sending data through one or other interface the module is implicitly selecting that initial jump in the path.

The connection with the *frontends* is straightforward: a *switch* can send data to one of its *frontends* if the *frontend*'s `ready` signal is active, and the *switch* must assert its `valid` signal to the *frontend* so it knows that it should capture the `data` that comes from the *switch*. Another set of `ready`, `valid`, and `data` signals allows a *frontend* to send data to the *switch* to which it is connected.

A network-on-chip implementation may connect multiple *switches* inside the same chip. Two adjacent *switches*, meaning that they are directly connected through one or more of their ports, can communicate using their `ready`, `valid`, and `data` signals, without the need for any *frontends*.

B. Switch Architecture

The switch has been designed so all of its channels work independently and in parallel. This allows data channels to be

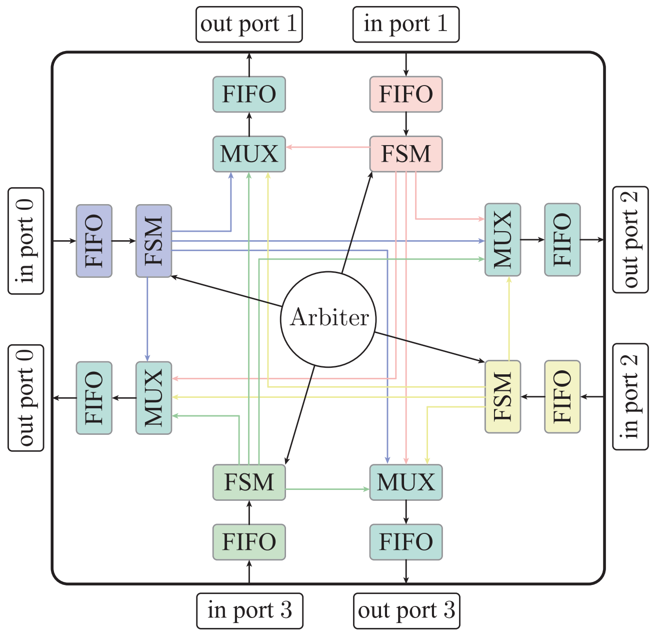


Fig. 11. Internal architecture of the switch. Note that the multiple communication channels can process data simultaneously: data can go, for example, from port 0 to port 2 at the same time that data goes from port 1 to port 3. Furthermore, ports are full duplex: each input port can work simultaneously and independently from its corresponding output port.

established between different pairs of ports without impacting the performance of other channels that may also be established. Furthermore, all ports are full duplex, with their input and output logic being independent, so it makes sense in this section to differentiate between input ports and output ports for clarity. Fig. 11 shows the architecture of a four-port *switch*, which has four input ports and four output ports. Each input port has a finite state machine (FSM) that, for each received *frame*, decodes the *path* inside the *header* and decides to which output port it should send the *frame*. When the FSM knows which output port it will send the data to, it petitions the arbiter for access to that specific output port. If the port is free, the arbiter immediately grants the FSM its ownership, but if it is not, the arbiter uses a round robin system for granting access to the output ports, so a single input port cannot exclusively monopolize an output port: before sending a second *frame*, it must wait until all the other input ports which want to send to that specific output port have finished sending one *frame* each. After the FSM finishes sending the *frame*, it releases the output port so the arbiter is free to assign it to another FSM if needed. It must be noted that, in order to avoid bottlenecks, the data to be transmitted does not cross the arbiter: the arbiter is only used to establish a communication channel between an input port and an output port.

On the other hand, each output port has a multiplexer (MUX in Fig. 11), which selects between the output of all FSMs in the switch, according to which FSM the arbiter has determined is the current owner of the output port.

C. Verification Procedure and Results

The system has been thoroughly verified by simulation and FPGA testing. Self-checking testbenches have been developed for the switch and also for the transmitter and receiver modules of each of the implemented frontends. These self-checking testbenches include a test sequencer, protocol drivers for the inputs, protocol monitors for the outputs, predictors to predict the expected outputs, and checkers to verify that the actual outputs match the expected ones. VUnit [32] has been used for test management and transaction message passing, and constrained random testing has been performed using the constrained random features of OSVVM [33]. Using VUnit, the different tests have been parameterized for different values of the VHDL generics of the modules under test, and also for multiple values of the total number of transactions processed in each test.

The testbench for the switch adapts to the protocol configuration chosen by the user, so the switch is always stimulated with *frames* that comply with the protocol, with the exact values from the *header* and *data* fields selected randomly, with different constraints depending on the specific test case. Furthermore, all the tests for the switch store all input and output *frames* in `.csv` files, so the exact same stimuli can be used in field tests over an FPGA device. A script has been developed that reads all the stimuli generated by the tests, sends them to the FPGA board, and afterwards checks that the FPGA outputs match the simulation outputs.

The complete set of switch tests has been run multiple times, automatically rewriting the *User_Constants.vhd* file for all possible configurations between *[Max_Number_of_Jumps=2, Number_of_Ports=2]* and *[Max_Number_of_Jumps=12, Number_of_Ports=12]*, to ensure functionality is preserved when changing the configuration of the network. Unsupported configurations, meaning those that result in a *header* longer than 5 bytes, are marked as such and thus the testbenches are not executed.

Line coverage for the synthesizable sources, as reported by GHDL [34] version 2.0.0-dev (1.0.0.r144.g68a7f85c) is 93.4%.

IV. Experimental Results and Discussion

The switch has been implemented for two different FPGA boards: the LX9 Microboard (using Xilinx ISE 14.7) and the PYNQ-Z2 (using Xilinx Vivado 2020.2). Two switch configurations have been implemented for each device: six jumps and four ports, and ten jumps and eight ports. For ease of testing, UART frontends have been used. The UART frontends have been configured with a speed of 1 Mbaud, with 1 stop bit and no parity, which is equivalent to 100 KB/s, although slower speeds can be used if needed. The functionality of the implemented designs has been checked using the inputs and outputs generated by the testbenches and it matches the expected behavior in all cases.

Both simulation and measurement over the FPGA implementations show that the switch, in absence of any contention, requires two clock cycles to send a single byte, plus an extra two clock cycles per *header* byte to begin processing a *frame*. This result is summarized in the following:

$$T_{\text{frame}} = 2 \cdot T_{\text{clk}} \cdot (2 \cdot \texttt{size\_header} + \texttt{data\_length}). \quad (9)$$

By contention we mean when a frame has to be routed through a port that is currently being used and thus it has to wait until the

TABLE I
MAXIMUM CLOCK FREQUENCIES (IN MHZ) FOR DIFFERENT CONFIGURATIONS OF THE SWITCH IN THE SPARTAN-6 AND ZYNQ-7000 FAMILIES

		Jumps				
	Number of ports	4	6	8	10	12
Spartan-6	3	98.74	94.29	103.78	103.10	104.60
	4	88.93	89.15	89.74	88.83	81.35
	6	73.29	73.06	69.71	70.18	–
	8	62.15	69.85	58.20	36.33	–
	10	36.99	40.03	38.97	–	–
	12	30.40	29.85	32.97	–	–
Zynq-7000	3	98.74	119.55	115.34	119.92	115.57
	4	112.35	118.29	115.97	120.57	121.11
	6	87.14	85.18	85.66	72.80	–
	8	62.15	69.85	58.20	36.33	–
	10	43.26	43.93	42.69	–	–
	12	36.35	35.64	35.41	–	–

previous frame has been fully transmitted. Equation (9) suggests that, as the *data_length* of the frame increases, the maximum throughput in bytes per second becomes approximately half the clock frequency at which the switch operates. For example, if a switch operates at 50 MHz, the upper bound for its throughput per port will be around 25 MB/s, which is a good rule of thumb to approximate the maximum capacity for a single switch port. The goodput (useful throughput) in bytes per second is actually given by the following:

$$\text{Goodput} = \frac{1}{2} F_{\text{clk}} \cdot \frac{\texttt{data\_length}}{2 \cdot \texttt{size\_header} + \texttt{data\_length}}. \quad (10)$$

The leftmost fraction of (10) is fixed, but the right fraction of that equation will take different values depending on both the protocol configuration (which determines *size_header*) and the size of the specific data that is sent in each frame. When the size of the payload is not zero, the value for that fraction is bounded between its minimum, 0.09 $(1/11)$, which happens when sending *frames* that only contain 1 byte of data in networks with 5-byte *headers*, and its maximum, 0.98 $(255/259)$, which occurs when *header_size* is 2 and *data_length* is 255.

It must be noted that this capacity is per port, so for example and in the absence of contention, all ports of an eight-port switch with a 50 Mhz clock could be running at 0.98×25 MB/s, which would result in a total goodput of almost 200 MB/s.

Table I shows the maximum clock frequencies the switch can have at different configurations, for two different FPGA families, using two different implementation toolchains (Spartan-6 family with Xilinx ISE 14.7, and Zynq-7000 family with Xilinx Vivado 2020.2, respectively). Data have been obtained from the post-place and route timing analyses for each of the implementations. Configurations marked with a hyphen (–) are unsupported due to the *header* being longer than 5 bytes. In Table I, the configurations with ten and 12 ports do not fit inside the lx9 device that the LX9 Microboard includes, so a different device from the same family and the same speed factor (–2) has been used for those. For the Zynq-7000, all the implementations have been made for the same device, specifically the xc7z020-1clg400c.

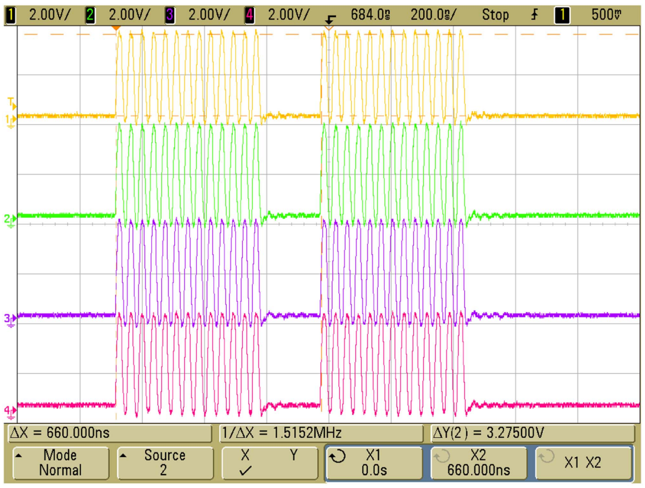


Fig. 12. Oscilloscope capture of the transmission of two frames (per port) with *data_length* = 8 in the switch configured for ten jumps and eight ports, on the Pynq-z2 development board. The measured time to send the 5 + 8 byte frame is 660 ns.

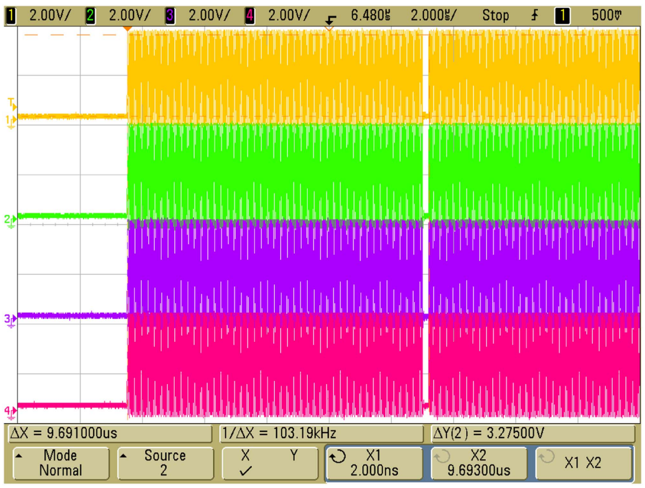


Fig. 13. Oscilloscope capture of the transmission of two frames (per port) with *data_length* = 255 in the switch configured for ten jumps and eight ports, on the Pynq-z2 development board. The measured time to send the 5 + 255 byte frame is 9.69 us.

FPGA resource utilization for different switch configurations is also shown in Table II. It must be noted that the implementations contain an UART frontend for each switch port. The resource utilization table shows that the switch implementations occupy a small percentage of a modern FPGA such as the Zynq-7020. The designs have a reasonable resource utilization even in the Spartan-6 lx9, which is the second smallest device of a low cost FPGA family first released in 2009. In the case of the Zynq-7020, many switches can be implemented in the FPGA, so a network-on-chip with multiple switches can be easily implemented in the device. Since the results suggest that the main limitation is the Block RAM (BRAM) capacity of the FPGA devices, an optimization that can be performed is to reduce the size of the internal FIFOs so they are synthesized into distributed memories instead of BRAMs.

Figs. 12 and 13 show four ports of an eight-port, ten-max-jumps switch outputting data in parallel. The pulsing signals

TABLE II
DEVICE UTILIZATION FOR DIFFERENT SWITCH CONFIGURATIONS IN THE TWO TESTED FPGA DEVICES

Switch conf.	Spartan-6 (xc6lx9)						Zynq-7000 (xc7020)					
	Flip-flops	[%]	LUTs	[%]	BRAMs	[%]	Flip-flops	[%]	LUTs	[%]	BRAMs	[%]
Four ports, four jumps	810	7.08	1738	30.28	16	25.00	800	0.75	1451	2.73	16	5.71
Four ports, six jumps	854	7.47	1852	32.38	16	25.00	840	0.79	1694	3.18	16	5.71
Six ports, four jumps	1275	11.15	2682	46.89	24	37.50	1278	1.20	2832	5.32	24	8.57
Six ports, six jumps	1356	11.85	2861	50.02	24	37.50	1350	1.27	3113	5.85	24	8.57
Eight ports, eight jumps	1827	15.97	3867	67.60	32	50.00	1832	1.72	4514	8.48	32	11.43
Eight ports, ten jumps	1883	16.46	4040	70.63	32	50.00	1896	1.78	5038	9.47	32	11.43

Due to the design including also the frontends, half of the occupied BRAMs are used by the switch, while the other half are used by the frontends.

TABLE III
COMPARISON TABLE BETWEEN DIFFERENT COMMUNICATION SCHEMAS

Type	Proposal	Overhead [bytes]	Speed	Minimum payload ratio	Available topologies	Maximum number of nodes
On-chip	Kim [35]	19	> 589 kbps	$1/20$	Star	Very high
	Ben [36]	8	> 100 kbps	$1/9$	Any	Very high
	SoCWire Protocol [37]	24	> 54.61 Mbps	$1/25$	Any	44
Off-chip	Modbus [20]	7	> 112.5 kbps	$1/8$	Bus	< 247
	CAN bus [38]	6	1 Mbps	$1/7$	Bus	110
	Profinet [22], (Real Time mode)	30	100 Mbps	$1/31$	Star, tree, daisy-chain, ring	Very high
	Profinet [22], (Non-Real Time mode)	138	100 Mbps	$1/139$	Star, tree, daisy-chain, ring	Very high
	EtherCAT [21], [39]	52	125 Mbps	$1/53$	Star, ring, bus	< 1000
	PESnet [25], [40]	4	125 Mbps	$1/5$	Ring only	32
	SyCCo Bus [23]	N/A	1.25 Gbps	N/A	Ring only	< 50
	RealSync [24]	Not Public	6.6 Gbps	Not Public	Tree	4095
On-chip Off-chip	Lynx	2 to 5	> 200 Mbps (consult Table I)	1/3 to 1/6	Any	Very high

are the `valid` signal of the output of each channel, which is activated once each two clock cycles. In this case, the clock of the switch is running at 54.67 MHz.

Table III presents a comparison between different communication schemas. The table compares different characteristics of the approaches, such as overhead, speed, minimum payload ratios, and maximum number of devices in the network, among others. The minimum payload ratio is the effective ratio, between user data and total frame size that corresponds to a frame that only sends a single data byte. Maximum payload ratios have not been included, since all protocols tend to achieve a payload ratio close to 1 when sending frames full of information. Profinet and EtherCAT are Internet-based protocols, while PESnet, SyCCo Bus, and RealSync are specifically designed for modular power converters. It must be noted that SyCCo and RealSync require specialized hardware (such as optical fiber connections) to achieve the specified communication rates. For the proposed approach, maximum capacity per port has been approximated as explained before: to check how the number of ports and jumps affects capacity, the reader can check Table I, divide the frequency by two, and multiply it by 8 to obtain the speed, in Mbps, per port.

V. CONCLUSION

A flexible communications protocol that enables efficient communications in distributed static networks has been proposed, implemented and verified both in simulation and when implemented in two different FPGAs. Single switches can be implemented in inexpensive FPGA devices, or multiple switches can be implemented in bigger, midrange FPGAs to easily create complex NoC. The endpoints of the communication can be either FPGAs or microprocessors, since the only condition they need to meet is to use the same physical communications layer that is implemented in the frontend connected to the specific port of the switch they are connected to. The protocol header size is automatically optimized to the minimum number of bytes necessary to establish effective communications, and the switches have been verified and characterized for many configurations. This allows the user to decide which network topology will be better suited for their application, according to their application needs and the maximum available throughput of the specific switch configurations they may be considering.

It must be noted that the limitation of a maximum of 5 bytes per header is a self-imposed restriction, which was adopted to guarantee that the headers do not grow too much, ensuring that the reduction of header sizes achieved by the proposed protocol is effective. If required, this self-imposed restriction could be removed for specific communication scenarios, at the cost of bigger header sizes.

The design is flexible so a single switch could have different frontends on different ports: in the case of distributed networks with one module acting clearly as a main module that directs the operation of other, secondary modules, having a faster frontend for the main module (such as optical communication link) and slower frontends for the secondary modules (such as UART or SPI) optimizes deployment costs and ensures that the main module has enough bandwidth to communicate with all the secondary modules.

Finally, although the main target of the proposed communication schema are static industrial networks which have typically small to medium sizes, when the protocol is configured with a high number of jumps and ports per switch, networks with hundreds or even thousands of modules can be implemented, depending on the specific network topology. Network traffic control, which would probably be needed in that case, could be deployed at a higher level network protocol, which we are currently considering as future work.

In summary, the authors believe that the minimalist communication protocol proposed in this work is a good candidate for static industrial communication networks, for several reasons given as follows.

1) The payload ratio is high due to the minimalist header length, which is automatically optimized depending on network configuration.
2) The return path mechanism allows receivers to identify the senders and send responses back to them without requiring space to store an origin address in the frame, reducing the frame space necessary to store addresses by a factor of two.
3) The decentralized routing mechanism prevents extra information exchange between modules.
4) The decentralized routing mechanism allows to share the loads between communication switches. In this sense, communication does not depend on a single element in the network.
5) The minimalist communication proposal is independent of the network topology structure.

The authors expect that this communication protocol, with its FPGA implementation, will enable both the improvement of existing applications that currently suffer from performance issues due to communication overhead and the creation of new distributed applications in various industrial settings. The proposal does not intend to replace well-established state-of-the-art protocols: it strives instead to solve a very specific problem which is the overhead introduced by communication protocols in the case of static industrial networks.

Future work will include development of frontends of higher capacity, implementing error detection and correction, and developing higher level network functionalities by building over the ones provided by the switch. In order to achieve that, two main approaches can be combined: a) reserving space in the payload to implement control values, and b) adding new types of blocks to the network, connected to the switches. For example, independent network discovery and management blocks could be connected to each switch, and these blocks could talk between themselves to implement self-discovery of the network and congestion awareness. In addition, communication control blocks could be inserted between frontends and switches to implement error detection and correction, message retransmission when unrecoverable errors are detected, and tolerance to misbehaving modules. Extra security features, such as encryption or frame tampering detection, could be implemented peer-to-peer in higher communication layers.

Acknowledgment

The authors would like to thank Mario Ríos Pérez for his contributions to the first VHDL implementation of the system. The authors would also like to thank the Xilinx University Program (XUP) for kindly providing ISE and Vivado licenses. The authors would also like to thank the developers of the FOSS software they have used for this article: Tristan Gingold and the rest of the contributors to the GHDL simulator, Lars Asplund and the rest of the contributors to the VUnit HDL test framework, and Olof Kindgren and the rest of contributors to the Fusesoc package manager and build abstraction tool; your work makes the FPGA world more awesome, accessible, and fun. The Spartan-6 LX9 microboards were procured back in the day by the teaching innovation project "Integración de herramientas y procesos típicos de la educacion online para la mejora de asignaturas presenciales centradas en el aprendizaje basado en proyectos," funded by "II Plan Propio de Docencia (Acuerdo 5/Consejo de Gobierno de la Universidad de Sevilla de 20/12/12), en su Accion 1.3.1.1. (Ayudas de Innovación y Mejora Docente)" of Universidad de Sevilla.

References

[1] H. Farhangi, "The path of the smart grid," *IEEE Power Energy Mag.*, vol. 8, no. 1, pp. 18–28, Jan./Feb. 2010.
[2] X. Fang, S. Misra, G. Xue, and D. Yang, "Smart grid - the new and improved power grid: A survey," *IEEE Commun. Surv. Tut.*, vol. 14, no. 4, pp. 944–980, Oct.–Dec. 2012.
[3] J. M. Guerrero, J. C. Vasquez, J. Matas, L. G. de Vicuna, and M. Castilla, "Hierarchical control of droop-controlled AC and DC microgrids - a general approach toward standardization," *IEEE Trans. Ind. Electron.*, vol. 58, no. 1, pp. 158–172, Jan. 2011.
[4] V. C. Gungor et al., "Smart grid technologies: Communication technologies and standards," *IEEE Trans. Ind. Informat.*, vol. 7, no. 4, pp. 529–539, Nov. 2011.
[5] Z. Li and Q. Liang, "Performance analysis of multiuser selection scheme in dynamic home area networks for smart grid communications," *IEEE Trans. Smart Grid*, vol. 4, no. 1, pp. 13–20, Mar. 2013.
[6] M. Wollschlaeger, T. Sauter, and J. Jasperneite, "The future of industrial communication: Automation networks in the era of the Internet of Things and industry 4.0," *IEEE Ind. Electron. Mag.*, vol. 11, no. 1, pp. 17–27, Mar. 2017.
[7] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, opportunities, and directions," *IEEE Trans. Ind. Informat.*, vol. 14, no. 11, pp. 4724–4734, Nov. 2018.
[8] F. Tao, H. Zhang, A. Liu, and A. Y. C. Nee, "Digital twin in industry: State-of-the-art," *IEEE Trans. Ind. Informat.*, vol. 15, no. 4, pp. 2405–2415, Apr. 2019.
[9] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang, and W. Zhao, "A survey on Internet of Things: Architecture, enabling technologies, security and privacy, and applications," *IEEE Internet Things J.*, vol. 4, no. 5, pp. 1125–1142, Oct. 2017.
[10] C. Burgos-Mellado, J. Pereda, A. Mora, R. Cardenas-Dobson, and T. Dragicevic, "Distributed control for modular multilevel cascaded converters: Toward a fully modular topology," *IEEE Ind. Electron. Mag.*, vol. 18, no. 1, pp. 32–45, Mar. 2024.
[11] V. G. Monopoli et al., "Applications and modulation methods for modular converters enabling unequal cell power sharing: Carrier variable-angle phase-displacement modulation methods," *IEEE Ind. Electron. Mag.*, vol. 16, no. 1, pp. 19–30, Mar. 2022.

[12] F. Flores-Bahamonde, H. Renaudineau, A. M. Llor, A. Chub, and S. Kouro, "The DC transformer power electronic building block: Powering next-generation converter design," *IEEE Ind. Electron. Mag.*, vol. 17, no. 1, pp. 21–35, Mar. 2023.
[13] C. Restrepo, C. González-Castaño, and R. Giral, "The versatile buck-boost converter as power electronics building block: Changes, techniques, and applications," *IEEE Ind. Electron. Mag.*, vol. 17, no. 1, pp. 36–45, Mar. 2023.
[14] J. She et al., "A cross-disciplinary outlook of directions and challenges in industrial electronics," *IEEE Open J. Ind. Electron. Soc.*, vol. 3, pp. 375–391, May 2022.
[15] J. Silva, V. Sklyarov, and I. Skliarova, "Comparison of on-chip communications in Zynq-7000 all programmable systems-on-chip," *IEEE Embedded Syst. Lett.*, vol. 7, no. 1, pp. 31–34, Mar. 2015.
[16] H. Michel et al., "The socwire protocol (socp): A flexible and minimal protocol for a network-on-chip," in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, 2012, pp. 1–8.
[17] H. Michel, "Integration of SRAM-FPGAs for hardware acceleration of a data processing module for space instruments," Ph.D. dissertation, Inst. Comput. Netw. Eng., Technische Universität Braunschweig, Braunschweig, Germany, 2017.
[18] Bosch, "Communication area network FD," 2012. [Online]. Available: https://web.archive.org/web/20151211125301/http://www.bosch-semiconductors.de/media/ubk_semiconductors/pdf_1/canliteratur/can_fd_spec.pdf
[19] Bosch, "Communication area network XL," 2022. [Online]. Available: https://www.bosch-semiconductors.com/ip-modules/can-protocols/can-xl/
[20] Bosch, "Modbus reference guide," 1996. [Online]. Available: https://www.modbus.org/docs/PI_MBUS_300.pdf
[21] H. Büttner, D. Janssen, and M. Rostan, "Ethercat - the ethernet fieldbus," *PC Control Mag.*, vol. 3, pp. 14–19, 2003.
[22] "Profinet description," 2014. [Online]. Available: http://us.profinet.com/wp-content/uploads/2012/11/PROFINET_SystemDescription_ENG_2014_web.pdf
[23] "Sycco communication protocol," 2022. [Online]. Available: https://hpe.ee.ethz.ch/research/syyco-bus.html#info
[24] "Realsync protocol by imperix," 2021. [Online]. Available: https://imperix.com/technology/low-latency-communication/
[25] I. Milosavljevic, "Power electronics system communications," Master thesis, Virginia Power Electron. Center, Virginia Tech, Blacksburg, VA, USA, 1999.
[26] L. Benini and G. De Micheli, "Networks on chips: A new soc paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.
[27] S. Huang, R. Teodorescu, and L. Mathe, "Analysis of communication based distributed control of MMC for HVDC," in *Proc. 15th Eur. Conf. Power Electron. Appl.*, 2013, pp. 1–10.
[28] L. Mathe, P. D. Burlacu, and R. Teodorescu, "Control of a modular multilevel converter with reduced internal data exchange," *IEEE Trans. Ind. Informat.*, vol. 13, no. 1, pp. 248–257, Feb. 2017.
[29] *IEEE Standard VHDL Language Reference Manual*, IEEE Std 1076-2008 (Revision of IEEE Standard 1076-2002), pp. 1–640, Jan. 6, 2009, doi: 10.1109/IEEESTD.2009.4772740.
[30] B. Marshall, "Hardware verification in an open source context," in *Proc. 1st Worskhop Open Source Des. Automat., 10th Workshop Des. Automat. Test Europe Conf.*, 2019. [Online]. Available: https://osda.gitlab.io/19/1.2.pdf
[31] *IEEE Standard VHDL Language Reference Manual*, ANSI/IEEE Std 1076-1993, pp. 1–288, Jun. 6, 1994, doi: 10.1109/IEEESTD.1994.121433.
[32] VUnit, a test framework for HDL, 2014. [Online]. Available: https://vunit.github.io/
[33] J. Lewis et al., "Open source VHDL verification methodology (OSVVM)," 2020. [Online]. Available: https://github.com/OSVVM
[34] T. Gingold et al., "GHDL: Free and open-source analyzer, compiler, simulator and (experimental) synthesizer," 2002. [Online]. Available: https://ghdl.github.io/ghdl/
[35] B. Kim, Y. Kim, D. Lee, and S. Tak, "A reconfigurable noc platform incorporating real-time task management technique for H/W-S/W codesign of network protocols," in *Proc. Int. Symp. Ubiquitous Multimedia Comput.*, 2008, pp. 238–243.
[36] A. Ben Achballah and S. Ben Saoud, "The design of a network-on-chip architecture based on an avionic protocol," in *Proc. World Symp. Comput. Appl. Res.*, 2014, pp. 1–5.
[37] T. Lange, B. Fiethe, H. Michel, H. Michalik, K. Albert, and J. Hirzberger, "On-board processing using reconfigurable hardware on the solar orbiter phi instrument," in *Proc. NASA/ESA Conf. Adaptive Hardware Syst.*, 2017, pp. 186–191.
[38] Bosch, "Communication area network FD," 2008. [Online]. Available: https://www.eecs.umich.edu/courses/eecs461/doc/CAN_notes.pdf
[39] D. Jansen, "Real-time ethernet: The ethercat solution," *Comput. Control Eng.*, vol. 15, pp. 16–21, Feb. 2004. [Online]. Available: https://digital-library.theiet.org/content/journals/10.1049/cce_20040104
[40] C. Carstensen, R. Christen, H. Vollenweider, R. Stark, and J. Biela, "A converter control field bus protocol for power electronic systems with a synchronization accuracy of NS," in *Proc. 17th Eur. Conf. Power Electron. Appl.*, 2015, pp. 1–10.

Hipólito Guzmán-Miranda (Senior Member, IEEE) received the degree in telecommunications engineering with a specialization in electronics and the Ph.D. degree in telecommunication engineering (with Hons. and European mention) through the Doctorate Program in electronic engineering, signal processing, and communications from Universidad de Sevilla, Seville, Spain, in 2006 and 2010, respectively.

He is an Associate Professor (Professor Titular de Universidad) with Universidad de Sevilla, where he teaches undergraduate and master's subjects mainly related to FPGA design and verification, system-on-chip, and project management applied to electronics. He has authored or coauthored more than 15 JCR journal papers and more than 40 conference contributions. His research interests include programmable logic applications, high-throughput visual classification systems, wireless for critical applications, and emulation and mitigation of radiation effects in digital devices.

Dr. Guzmán-Miranda has been Chair of the IEEE IES Technical Committee on Electronic Systems on Chip (term 2020–2021), and Cluster Delegate for IEEE IES Cluster 4 (Cross-Disciplinary Cluster, term 2021–2022).

Abraham Marquez Alcaide (Member, IEEE) was born in Huelva, Spain, in 1985. He received the B.S., M.S., and Ph.D. degrees in telecommunications engineering from Universidad de Sevilla, Seville, Spain in 2014, 2016, and 2019, respectively.

He has coauthored more than 60 journal articles and participated in more than 25 R&D projects. His research interests include advanced modulation techniques, multilevel converters, modular converters, model-based predictive control of power converters and drives, renewable energy sources, thermal modeling of power converters, and power device lifetime extension.

Dr. Marquez Alcaide was the recipient as coauthor of the 2015, 2021, and 2023 Best Paper Award of the IEEE Industrial Electronics Magazine.