applied sciences

MDPI

Article

Hardware Communications: An Open-Source Ethernet Switch Implementing the Parallel Redundancy Protocol

André Torres [1,2,*], Rodrigo Piedade [1,†], Filipe Moutinho [2,3,*] and Luis Gomes [2,3]

[1] EFACEC Energia—Máquinas e Equipamentos Eléctricos S.A., Quinta da Fonte, 2770-229 Oeiras, Portugal; rodrigo.piedade@siemens.com
[2] NOVA School of Science and Technology, NOVA University Lisbon, 2829-516 Caparica, Portugal; lugo@fct.unl.pt
[3] Center of Technology and Systems (UNINOVA-CTS) and Associated Lab of Intelligent Systems (LASI), 2829-516 Caparica, Portugal
* Correspondence: andre.pimentel.torres@efacec.com (A.T.); fcm@fct.unl.pt (F.M.)
† Current address: Siemens, Südstr. 74, 04178 Leipzig, Deutschland.

Abstract: Ethernet communications are widely used in many areas, and redundant protocols like the Parallel Redundancy Protocol (PRP) and High-Availability Seamless Redundancy (HSR) were created to make these communications more reliable. These protocols' goal is to have a zero-delay network reconfiguration time. In order to achieve this, both protocols have a double network, and a copy of every packet will be present on the network. Upon packet reception, it will be verified if it is new or has been received before, being discarded if it is duplicated. This paper presents an open-source Ethernet switch, the REDSwitch, capable of implementing the PRP protocol, besides working as a regular switch. This Ethernet switch is based on an open-source project and was developed further, including a hash table with an aging mechanism, to store the received packets and allow duplication analysis. The REDSwitch is capable of dealing with the IPv4 protocol, IEEE 802.1Q, and implementing the PRP protocol. Specified in Verilog and SystemVerilog, it was designed to be implemented on field-programmable gate array (FPGA) devices.

Keywords: automation; Ethernet; switch; communications; redundancy; IEC 62439; Parallel Redundancy Protocol (PRP); IEEE 802.1Q; FPGA; hash table

Academic Editors: Amalia Miliou and Valery Salauyou

Received: 6 February 2025
Revised: 12 March 2025
Accepted: 21 March 2025
Published: 25 March 2025

Citation: Torres, A.; Piedade, R.; Moutinho, F.; Gomes, L. Hardware Communications: An Open-Source Ethernet Switch Implementing the Parallel Redundancy Protocol. *Appl. Sci.* **2025**, *15*, 3596. https://doi.org/10.3390/app15073596

1. Introduction

In industrial applications, such as automation, communications between computers and machines are critical, and any flaw can damage an entire system. Many of these communications are based on Ethernet networks, working at Gigabit speeds and having some protocols developed specifically for industrial communications, like EtherCAT and PROFINET [1].

Since these communications are crucial to many systems, they need to be as reliable as possible. This has inspired the creation of redundant protocols that could bring zero-delay network reconfiguration [2]. The Parallel Redundancy Protocol (PRP) and High-Availability Seamless Redundancy (HSR) are two redundant protocols introduced by the IEC 62439-3 standard [3]. Both protocols have a duplicated network but different topologies: the PRP uses a double local area network (LAN), while HSR uses a ring topology. Every packet sent will be duplicated on the network; this means that, if a flaw occurs in a network, the packet can still be delivered by the other one. These duplicated packets can be handled by Ethernet switches, which need to discard the second copies. A redundant Ethernet switch,

named RedBox (Redundant Box), can be used to attach any non-redundant device to a PRP/HSR network, executing the PRP/HSR protocol for them.

Field-programmable gate arrays (FPGAs) have multiple possible applications. Their reconfiguration characteristics make them suited for industrial Ethernet switch implementation [4]. FPGAs can be low-cost implementation platforms for real-time Ethernet switches [5]. Additionally, the network conditions of real-time Ethernet communications can be diagnosed by FPGA-based instruments [6]. In [7], a ring network adapter was implemented in an FPGA, whereas, in [8], an FPGA implementation of a real-time network adapter, serving as an Ethernet bridge, was presented. The evolution of FPGAs continues with the improvement of hardware resources like analog resources, floating point arithmetic, embedded processors, and hard memory control, as well as software development, including the availability of many IP cores that can simplify the design of complex systems [9]. This constant evolution positions FPGAs as an interesting option in terms of cost–performance, and they are used to implement controllers in many industrial areas, like robotic applications, power electronics, and drive applications.

The main goal of this work was the development of an open-source Ethernet switch implementing the PRP protocol, filling a gap in existing implementations. To make this possible without developing a switch from scratch, multiple open-source projects related to network communications and switches were analyzed, and one was selected to be used as a basis for development. Besides implementing the PRP protocol, the proposed Ethernet switch should also be capable of working as a non-redundant switch. The developed switch was named REDSwitch, and its source code is available online at https://github.com/TorresA/RedSwitch (accessed on 5 February 2025). It supports the IPv4 protocol and the IEEE 802.1Q norm and implements the PRP protocol. Developed in Verilog and SystemVerilog, it is, as far as we know, the only open-source Switch implementing any of the redundant protocols (PRP or HSR).

This paper is divided into five sections, having the following structure. Section 2 addresses some redundant switches available on the market and some open-source projects related to switches and networking and justifies the selection of the one that was used as a starting point. In Section 3, the proposed switch is described, along with how it differs from the open-source switch that was used as a starting point and with the developments in the used hash table. In Section 4, several simulations performed on the Ethernet parser, the hash table, and the complete switch are presented and discussed. Finally, Section 5 presents the conclusions and future work.

2. Related Work

This section presents some products available on the market that can implement the Parallel Redundancy Protocol (PRP) and some open-source Ethernet communication projects and switches. The main features of the commercial products are identified, showing the capabilities of those products that implement redundancy protocols. However, the main goal of this section is the analysis of the open-source projects that can be used as the starting point to develop an Ethernet switch capable of implementing the PRP.

2.1. Redundant Switches

In the market, there are already some products capable of implementing redundant protocols, which are briefly analyzed in this section. The analyzed products are the Flexibilis Redundant Switch (FRS) by Flexibilis [10], the High-Availability Seamless Redundancy and Parallel Redundancy Protocol Switch (HPS) by SoC-e [11], the RED25 [12] by Hirschmann, and, finally, the PRP-1 IP Core and the High-Availability Seamless Redundancy (HSR) IP Core [13] developed by Zurich University of Applied Sciences. All of them, except the

PRP-1 IP Core, can implement both redundant protocols (PRP and HSR). Most products can work at speeds of 10/100/1000 Mps, are capable of filtering by EtherType and/or MAC address, have a CPU interface for communication and control, and even have the Precision Time Protocol (PTP) implemented over the redundant protocol for better synchronization. RED25, by Hirschmann, stands out with its unique features as it is being prepared for application in harsh industrial environments, being very resilient. Table 1 summarizes the characteristics of the mentioned products.

Table 1. Redundant switches: main features.

	FRS	**HPS**	**RED25**	**PRP-1 IP Core**	**HSR IP Core**
Speed (Mbps)	10/100 /1000	10/100 /1000	10/100	100	100
Redundancy	PRP and HSR	PRP and HSR	PRP and HSR	PRP	PRP and HSR
Filtering by EtherType	No	Yes	No	No	No
Filtering by MAC	Yes	Yes	Yes	Yes	Yes
PTP	Yes	Yes	Yes	No	Yes
CPU Interface (Control)	Yes	Yes	No	Yes	Yes
CPU Interface (Communication)	Yes	Yes	No	Yes	Yes
Bandwidth Control	Yes	Yes	No	Yes	Yes

2.2. Open-Source Switches

Several open-source projects exist, implementing network communication systems and different types of Ethernet switches. This subsection presents these projects and identifies the best solution to use as a starting point for the development of the redundant switch.

Private Island [14] is a Gigabit Ethernet network communication project developed by Mind Chasers Inc. It implements network communication based on FPGA technology, which is capable of performing different parallel processes and executing operations in the physical layer, like filtering addresses and ports. This implementation can be used for packet processing, being used, for example, in the Internet of Things (IoT) to read sensor values and control motors/actuators.

NetFPGA [15] is an open-source development platform for both hardware and software implementations; it is available for use by the academic community, allowing not only researchers but also students to build and develop networking systems in hardware [16]. Several boards have been developed, namely NetFPGA Plus, NetFPGA SUME, and NetFPGA 1G, for network and high-speed communication. Both of them can be used to implement a regular Ethernet switch, with the NetFPGA Plus and the NetFPGA SUME boards being the most recent ones. Regarding software implementations, there are two Ethernet switches: the Real-Time Switch [17] and OpenFlow Switch [18]. The Real-Time Switch is an Ethernet switch based on a time-triggered mechanism that has two working modes: soft mode, where it works like a regular Ethernet switch, performing switching through best-effort approximation, and hard mode, where time triggers are defined to

perform switching. Both modes can be applied simultaneously on the same network. The other implementation is the OpenFlow Switch, which is a switch based on flow tables. These tables have information regarding the priority and destinations of the packets. The switch and controller communicate with each other, and actions are made taking into consideration the values present in the flow tables. This implementation was expanded to the OpenFlow MPLS Switch and became capable of dealing with multiple protocols at once. To allow this, two tuples containing information about the existing protocols were added to the OpenFlow framework.

ONetSwitch is a platform that started as a crowdfunding project on Kickstarter [19]. The goal was to create an open-source software-defined network (SDN) [20]. Multiple boards, based on FPGAs, were developed: ONetCard, ONetSwitch20, ONetSwitch30, and ONetSwitch45. In addition to these hardware boards, open-source software was also developed; this is how the REF Switch [21] project emerged. The REF Switch is a basic Ethernet switch that works at speeds of 10/100/1000 Mbps; it can parse the MAC addresses and EtherType from a packet header and forward packets using a MAC address table that is capable of learning and mapping new addresses. This implementation also has CPU control and communication interfaces.

2.3. Selected Project

The analysis of the open-source projects, summarized in Table 2, allowed us to identify the best candidate to start the development of an open-source Ethernet switch with the PRP. The REF Switch project, from ONetSwitch, was the selected project. It could work at the three desired speeds (10/100/1000), which the other options did not permit, and already had CPU interfaces for control and communication, the latter of which was not available in the other options and would have needed considerable effort to develop. Before the PRP implementation, this project required some further development: it was necessary to ensure that it was capable of dealing with frames according to the IPv4 protocol format and the 802.1Q IEEE norm [22], regarding VLANs. In this way, it would be possible to implement different priorities depending on the EtherType, MAC addresses, and VLAN values.

Table 2. Open-source projects.

	Private Island	**NetFPGA**	**ONetSwitch**
Speed (Mbps)	1000/10,000	1000	10/100/1000
Redundancy	No	No	No
Filtering by EtherType	No	No	No
Filtering by MAC	Yes	No	Yes
PTP	No	Yes	No
CPU Interface (Control)	Yes	Yes	Yes
CPU Interface (Communication)	No	No	Yes
Bandwidth Control	Yes	Yes	Yes

3. REDSwitch

The proposed switch, named REDSwitch, is presented in this section. A PRP network containing two REDSwitches is presented in Figure 1. The REDSwitches connect two

single attached nodes (SANs) to the network and execute the PRP protocol for them, since SANs are not capable of duplicating packets and discarding duplicates. On the other hand, dual attached nodes (DANs) are devices capable of executing the PRP protocol, which is why they do not need to be connected to any RedBox (or REDSwitch, in this scenario) and are directly connected to the PRP network.

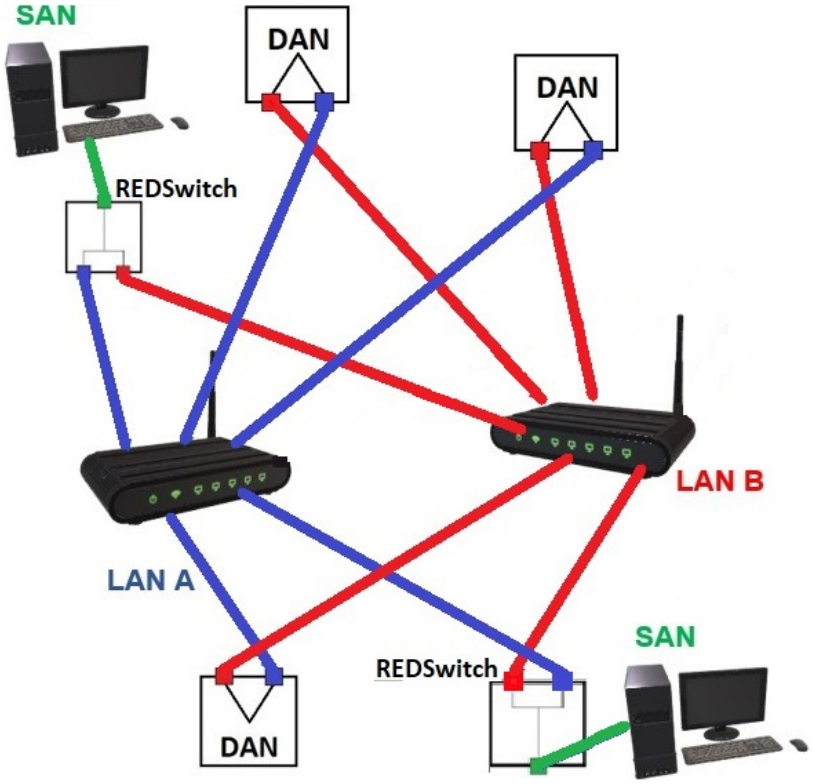

Figure 1. PRP topology with REDSwitch integrated.

3.1. Overview

As previously mentioned, the REF Switch, by ONetSwitch, was the chosen project to start the development of the new redundant Ethernet switch (REDSwitch). The REF Switch is a traditional Ethernet layer 2 switch, capable of working at speeds of 10/100/1000 Mbps. It uses a round robin on the input arbiter to satisfy the queue requests, as it does on the output queues when sending the packets. It is capable of parsing 32- or 64-bit words, but the Ethernet parser has basic functionalities, being capable only of reading the MAC addresses (source and destination) and EtherType from a packet header. The MAC addresses and respective ports are saved in a table that works with a look-up engine connected to the Ethernet parser. Thus, we can see that, before implementation regarding the redundancy (PRP protocol), this switch must be developed further, mainly in terms of the Ethernet parser and the MAC address table. The table used to store MAC addresses has a limited search mechanism, so it is necessary to develop a hash table.

Besides implementing the PRP protocol, the REDSwitch should be capable of working as a regular Ethernet switch when the PRP mode is not active, so it is necessary to preserve the actual implementation while adding what is needed to implement the PRP protocol. When receiving a packet in PRP mode, the switch should check if it is a duplicate or not, accepting and forwarding only the first, as we can see in Figure 2. When sending a packet in PRP mode, the switch needs to duplicate it, sending it to both LANs, as presented in Figure 3.

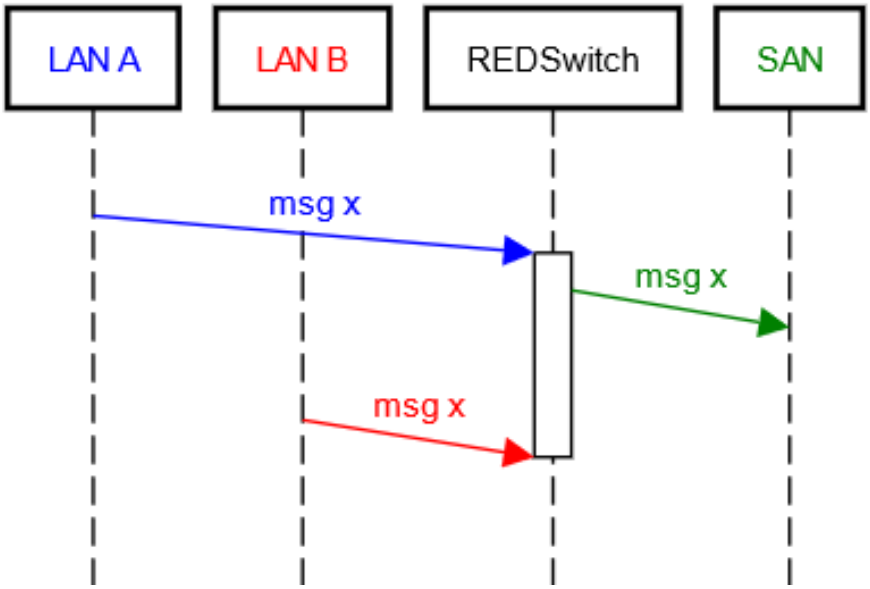

Figure 2. UML sequence diagram: REDSwitch duplicate packet discard.

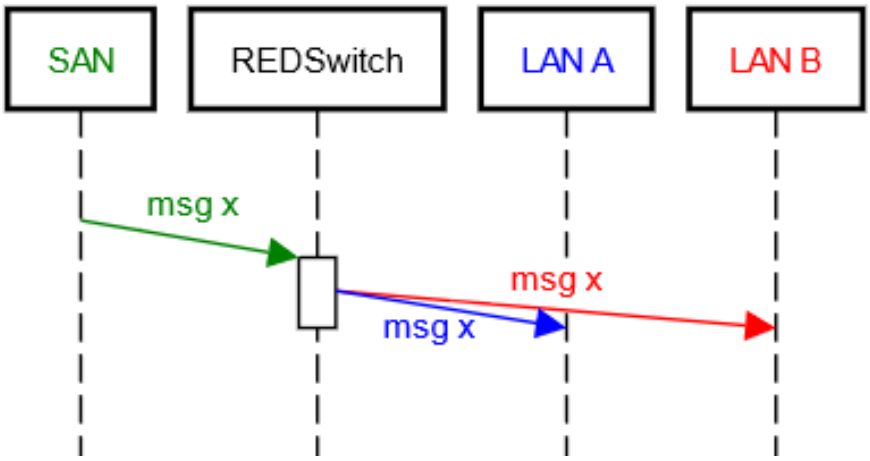

Figure 3. UML sequence diagram: REDSwitch packet duplication.

3.2. Ethernet Parser Supporting IPv4 and IEEE 802.1Q

Before implementing the redundancy (PRP protocol), the chosen project needs to support additional features, mainly its Ethernet parser. The Ethernet parser of the REF Switch is only capable of parsing two words (32/64 bits), where it obtains the MAC addresses (source and destination) and EtherType values. Then, it waits for the end of the packet. The REF Switch is able to identify the EtherType, determining whether it is in IPv4 format or not, but it is not capable of dealing with IPv4 headers.

The REDSwitch Ethernet parser must deal with IPv4 headers and the IEEE 802.1Q (VLANs) [22] standard. Being capable of identifying VLAN tags enables the implementation of priorities in the future. This Ethernet parser must read different formats of packet headers: IPv4 format, depending on the number of VLANs (0 to 2), and without a standard EtherType, where this field represents the length instead. The state diagram of the Ethernet parser is presented in Figure 4.

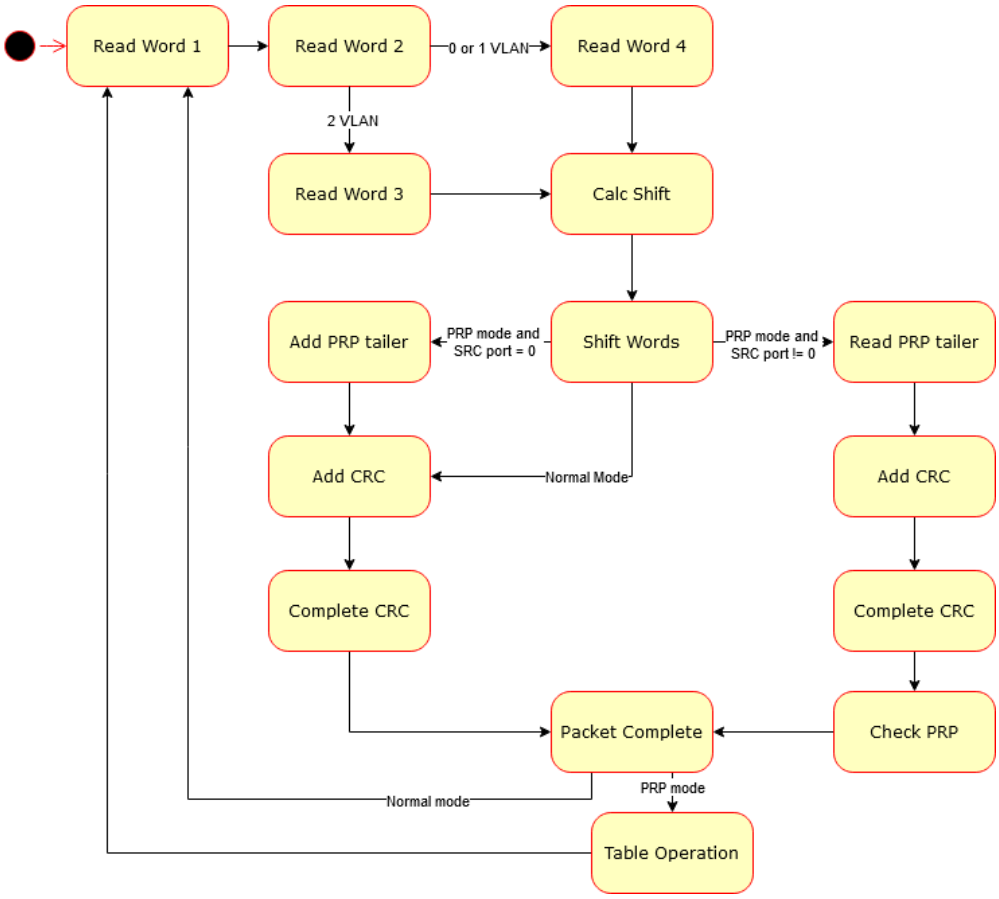

Figure 4. Ethernet parser state diagram.

Regarding the IPv4 header, the "length" field will be read and used to calculate the number of 64-bit words in the packet and the remaining bytes, in the "Calc Shift" state shown in Figure 4. Using these values, the payload will be shifted until its last byte, in the "Shift Words" state. In this way, it is possible to read the destination port at the end of the packet and write the PRP fields necessary for packet duplication.

3.3. Ethernet Parser Supporting Redundancy

The REDSwitch, when working in PRP mode, can face two different scenarios: receiving a packet coming from the onboard CPU port or receiving a packet from an outside LAN. The scenario where it receives a packet coming from an outside LAN can be separated into two sub-scenarios: a packet coming from a DAN or from a SAN, since both can be connected to a PRP LAN. The tasks performed by the Ethernet parser in these two different scenarios, as well as a third scenario, where the REDSwitch works as a regular switch, can be observed in the state diagram presented in Figure 4, where the "Shift Words" state has three output arcs to three different branches of the diagram.

The packets that come from the onboard CPU (SRC port = 0) will need to have the PRP fields (LAN, length of the link service data unit (LSDU), sequence number, and PRP ID) at the end of the packet. These fields must be added before sending it to both LANs. This task is performed in the "Add PRP Tailer" state of the left branch of the state diagram presented in Figure 4.

When a packet comes from one of the LAN networks (SRC port != 0), it is necessary to receive the entire packet to check the PRP ID field value at the end. This is required to understand if it was sent by an SAN or DAN. When parsing the packet (in the "Read PRP" state in the right branch of the state diagram presented in Figure 4), the PRP tailer will be stored on a temporary register until the PRP ID is received and verified in the "Check PRP"

state. If the PRP ID confirms that the packet comes from a DAN, the information stored on the temporary register will be sent out by the Ethernet parser. If the PRP ID field does not match, it means that the packet was sent by an SAN, which means that the parser must discard the information stored on the temporary register and calculate and add the PRP tailer, as in the scenario where a packet comes from the onboard CPU.

In any of the presented scenarios, the REDSwich needs a cyclic redundancy check (CRC) calculation tool. This tool will progressively calculate the packet CRC by calculating the CRC of each word and updating the CRC value after every new word. In the REDSwitch implementation, we integrated a CRC calculation tool developed by easics [23].

To support packet duplication and discarding, the Ethernet parser will have to store the packets that were already sent and received. These packets will be stored in a hash table, which allows the verification of previously received packets and duplicate identification.

When the parser is in the "Packet Complete" state (Figure 4), it checks if it is in PRP mode. If it is not in PRP mode, it returns to the initial state; otherwise, it moves to the "Table Operation" state. In this state, the Ethernet parser will search the hash table, using the combination of the source MAC and sequence number, to verify whether the packet is a duplicate or not, inserting it if it is new.

3.4. Hash Table

The developed REDSwitch includes a hash table, based on an open-source project [24]. In this table, all elements are connected to each other and are identified by two fields: a key and value. In this table, it is possible to insert, delete, and search for elements. The elements are inserted at the first empty address of the table, and, upon deletion, the respective address will be stored in the empty addresses list. After being inserted, each element will have added two new fields, one identifying whether there is an element next to it and the other with the respective address. Having all elements connected in this way will allow a search through the table without using specific addresses.

This hash table will be used for the redundant packets; when working in normal mode, the switch will use the original MAC table. To identify the packet in the hash table, a key field consisting of a combination of 48 bits is used, using the lowest 40 bits from the source MAC address and the lowest 8 bits from the sequence number, because these are the bits that will change the most. Regarding the field value, it will be used to store the age of the packet, which will be used by the aging algorithm. The field value will start at zero.

In this work, the delete algorithm of the hash table was changed to implement the proposed aging algorithm. Originally, the delete algorithm would have searched for a given key in the table, deleting the element if there was a match. The proposed aging algorithm, presented in Listing 1, keeps cycling through all elements, incrementing the value field (the age) of each one. When this value reaches a defined threshold (very old), it is deleted. Therefore, it is crucial to choose an appropriate threshold value to ensure that the first packet is not discarded before the second one arrives. Otherwise, the second packet may be incorrectly treated as a new packet. It is important to note that the aging algorithm starts when a delete command is received with a key with all bits equal to zero. Since no element can be inserted using zero as a key, this value was used to start the execution of the aging algorithm.

Listing 1. Pseudocode of aging algorithm.

```
threshold = 10 (customizable parameter)

cycle through all table elements
calculate difference between aging counter and
element value
if difference < threshold
update the element value (+1)
else
check current state (to find element position)
if state = READ_HEAD_S
delete elem. (state KEY_MATCH_IN_HEAD_S)
else
if state = GO_ON_CHAIN_S
check if element is in middle/end
if got tail
delete elem. (state KEY_MATCH_IN_TAIL_S)
else
delete elem. (state KEY_MATCH_IN_MIDDLE_S)

set next state to analyze the next element:
if got tail
state next = READ_HEAD_S
else
state next = GO_ON_CHAIN_S
```

It is possible to observe, in Listing 1, that there are three distinct key match states when an element needs to be deleted. The difference between these three key match states is related to the position of the element to be deleted, changing how the algorithm deals with the previous element. When an element is deleted, the algorithm needs to update the next pointer field of the previous element, except when deleting the first element (in the state KEY_MATCH_IN_HEAD_S). If it is a middle element (KEY_MATCH_IN_MIDDLE_S state), the algorithm updates the next pointer field with the current element's address. For the last element (in the state KEY_MATCH_IN_TAIL_S), it deletes the next pointer field value and changes the field's next valid element to zero, since the previous element is now the last one.

3.5. REDSwitch Core

The proposed REDSwitch architecture is presented in Figure 5. The input arbiter is the original one from the REF Switch [21]. The REDSwitch "Ethernet Parser" includes the "Header Parser" connected to the "Hash Table", so it can search for duplicates and insert the packets when sending. The REF Switch Ethernet parser only had one data output, whereas the developed Ethernet parser has two outputs ("data A" and "data B") to support packet duplication. As a consequence, the "Output Queues" from the REF Switch were modified to include two data inputs instead of one. The original eight outputs (first-in, first-out (FIFO) queues) were divided into two groups of three (even-numbered FIFOs and odd-numbered FIFOs), each one associated with a LAN ("data out A" connected to LAN A and "data out B" connected to LAN B). In this way (when working in PRP mode), each group of FIFOs will send the packet to the respective LAN. FIFO 0 will be associated with the onboard CPU, whereas FIFO 7 will not be used in PRP mode to keep the groups with three FIFOs each.

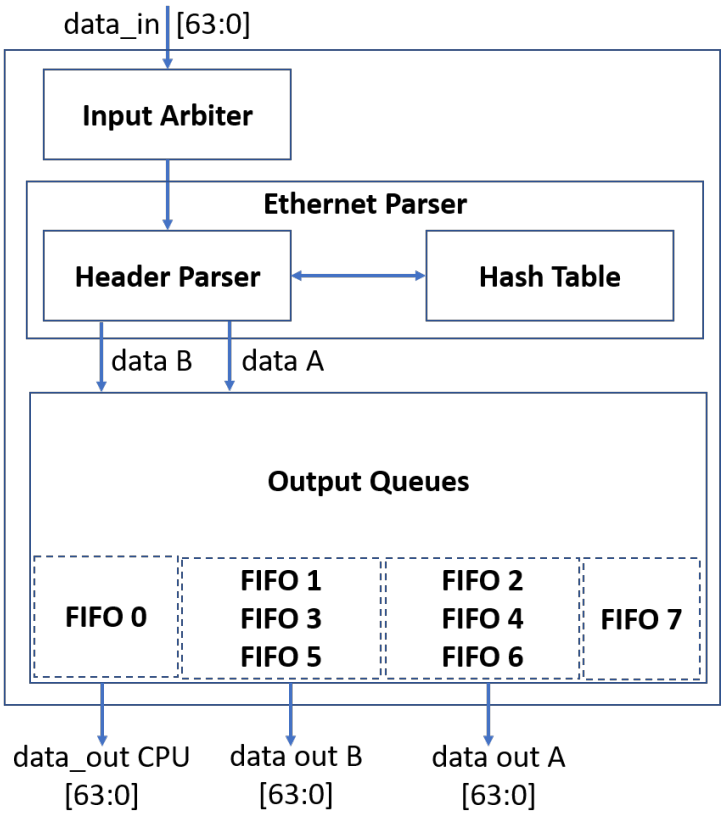

Figure 5. REDSwitch core block diagram.

4. Validation

This section presents several simulations that were performed in Xilinx Vivado 2019.2, considering the Zynq-7000 XC7Z020CLG484-1 board (Xilinx, San Jose, CA, USA), to validate the Ethernet parser, the hash table, and the complete switch. For the Ethernet parser, simulations were performed to validate the reading of the IPv4 protocol and IEEE 802.1Q fields; the reading of packets with a PRP tailer; and the addition of the PRP tailer to the packets to be duplicated. Regarding the hash table, the aging algorithm was tested to validate its capability of deleting and updating the correct elements and also how the insertion of an element would affect the aging algorithm while it was running. Finally, the REDSwitch core (Figure 5) was simulated to verify that the destination port was correctly codified, so that it could select one output FIFO from each node (LAN A and LAN B), duplicating the packet.

The packets used in the following simulations were generated using the software CatKarat (version 1.51.200) [25], except in one of the simulations, where a real packet was used. In this way, we could have packets respecting the desired parameters to perform the required simulations.

To confirm that the CRC was correctly calculated, an online CRC calculator tool was used [26]. Different polynomials can be used to calculate a packet CRC, with CRC-32 being the most commonly used on Ethernet networks.

4.1. Ethernet Parser

The first two simulations represent the scenario where the switch receives a packet with one VLAN tag that is not in the PRP format. In the first simulation, the switch is in normal mode, whereas, in the second simulation, the switch is in PRP mode. In Table 3, we show the field values used to generate the packet; moreover, in the last two lines of the table, we show the expected CRC values, which must be calculated and added at

the end of the packet, for the switch in normal mode and in PRP mode. To simplify the reading of the following time diagrams, it is important to note that the packet payload is "aaaaaaaaaaaaaaaaaaaa" and that it is followed by the destination port ("6") and by random values ("8bc2…") that will be overwritten with the CRC value.

Table 3. Values for simulation with generated packet.

Field	Value
Destination MAC	48003B31C2D7
Source MAC	5292F4174E30
VLAN ID	0fff
Protocol ID	8100
EtherType	0800
Length	46
IHL	5
Source Port	0
Destination Port	6
CRC (Normal)	50C29332
CRC (PRP)	0363F597

The result of the first simulation, in normal mode, is presented in Figure 6. It is possible to see that the values in Table 3 match the values read by the Ethernet parser. Moreover, in the last word of output_A_FINAL, it is possible to see that, after the first four bits (value 6, in hexadecimal—the destination port), the CRC was correctly added.

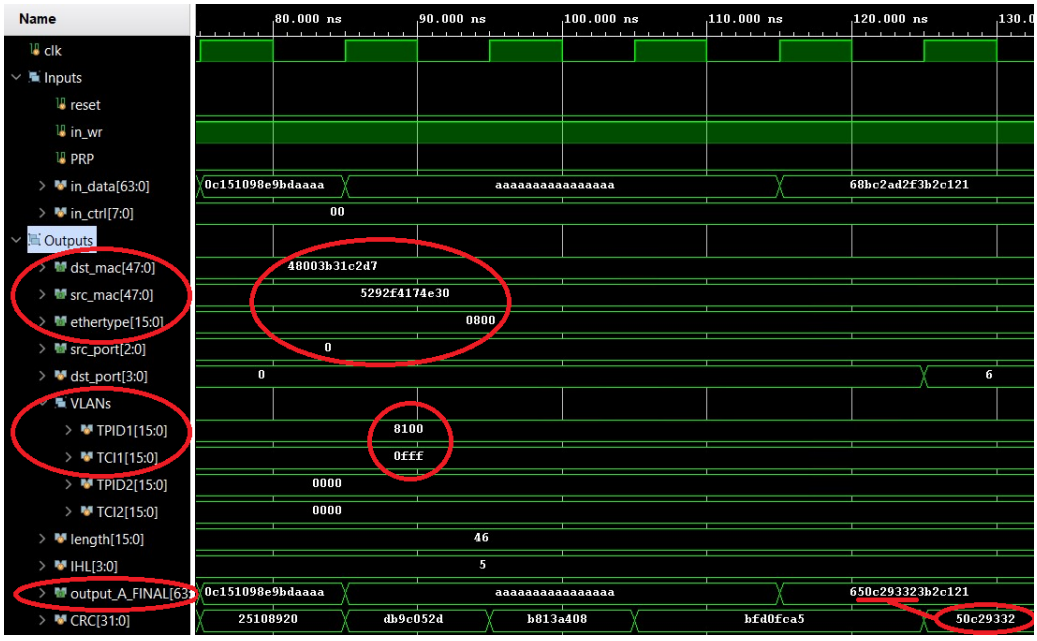

Figure 6. Simulation of a packet with 1 VLAN in normal mode.

The same packet was used in the second simulation in PRP mode. In this mode, besides parsing the previous fields, the switch must also calculate the PRP fields and add them at the end of the packet, followed by the CRC. The sequence number starts as 1 and is incremented for each pair of packets sent; the LAN is codified according to the destination port; the LSDU will be calculated (the sum of the size of the payload and PRP fields except

the PRP ID); and the PRP ID is a constant that identifies the presence of the PRP. Figure 7 presents the second simulation, where it is possible to see that the fields from Table 3 were also correctly parsed and that the PRP fields and CRC (PRP) were correctly inserted at the end of the packet.

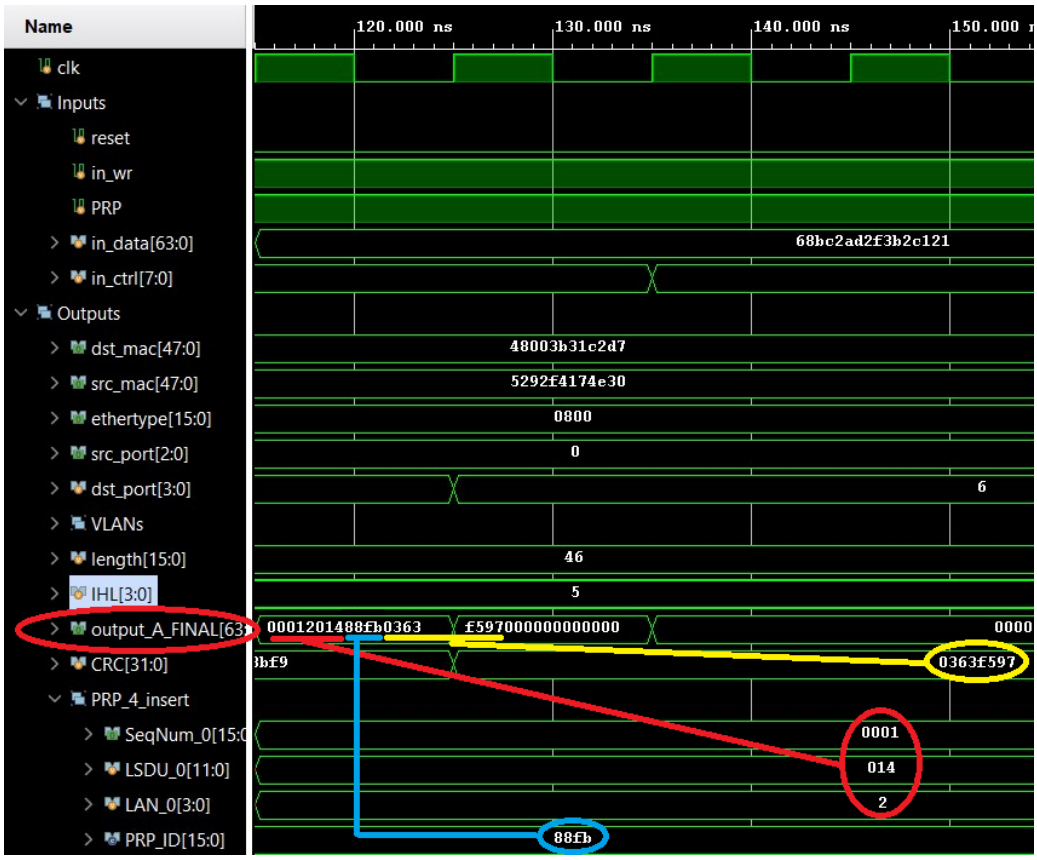

Figure 7. Simulation of a packet with 1 VLAN in PRP mode.

In the next simulation, the switch receives a packet already in PRP mode. In this simulation, a real example of a PRP packet was used, and its fields are presented in Table 4. Given that this is a PRP packet, the destination port is not at the end of the payload, and the CRC must be inserted after the PRP tailer. The simulation result is presented in Figure 8, where it is possible to observe that the parsed values match the ones presented in the table and that the CRC was correctly inserted at the end of the packet.

Table 4. Values for simulation with real PRP packet.

Field	Value
Destination MAC	010ccd040000
Source MAC	4a8a9da72b34
EtherType	88ba
Length	213
Source Port	1
Sequence Number	46016
LAN	10
LSDU	221
PRP Suffix	88fb
CRC	1C16C543

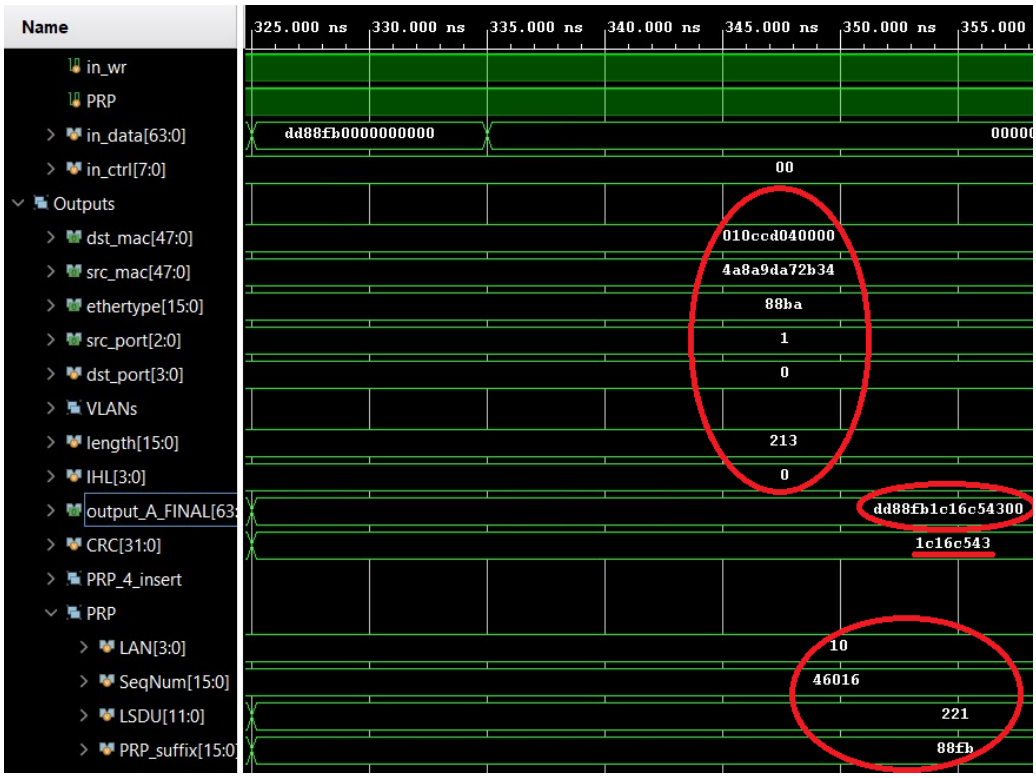

Figure 8. PRP packet simulation.

4.2. Hash Table

To test the aging algorithm, several elements were inserted, updated, and deleted from the table. The following three simulations present the scenarios where elements were automatically deleted or updated. Before these simulations, seven elements were inserted into the table, with keys and values that were incremented from 1 to 7, and the threshold value (the age), which determined whether the element was updated or deleted, was set to 10. It is important to note that, for simulation purposes, the aging counter was reduced to 4 bits instead of 16 bits.

In the first simulation, presented in Figure 9, the aging algorithm deletes the first element. It is possible to see that the aging counter is equal to "c" (in hexadecimal, which is equal to 12 in decimal) and that the aging difference is equal to "b" (in hexadecimal, which is equal to 11 in decimal), since the value of the first number is 1. Considering that the aging difference is larger than the defined threshold (10), this element must be deleted. It is possible to see that the "next_sate" is "KEY_MATCH_IN_HEAD_S", meaning that the element will be deleted.

The scenario where an element must be updated can be seen in Figure 10. The aging counter of this element (the second one) is equal to 1, such as the aging difference (below the defined threshold), meaning that it will be updated and not deleted, as shown in the next_sate value ("UPDATE_VAL_WR"). When an element is updated, its value will be increased by one and its position will be saved like a checkpoint; this means, that after this element, whenever another element is deleted, the algorithm will come back to the updated element to update its next pointer value.

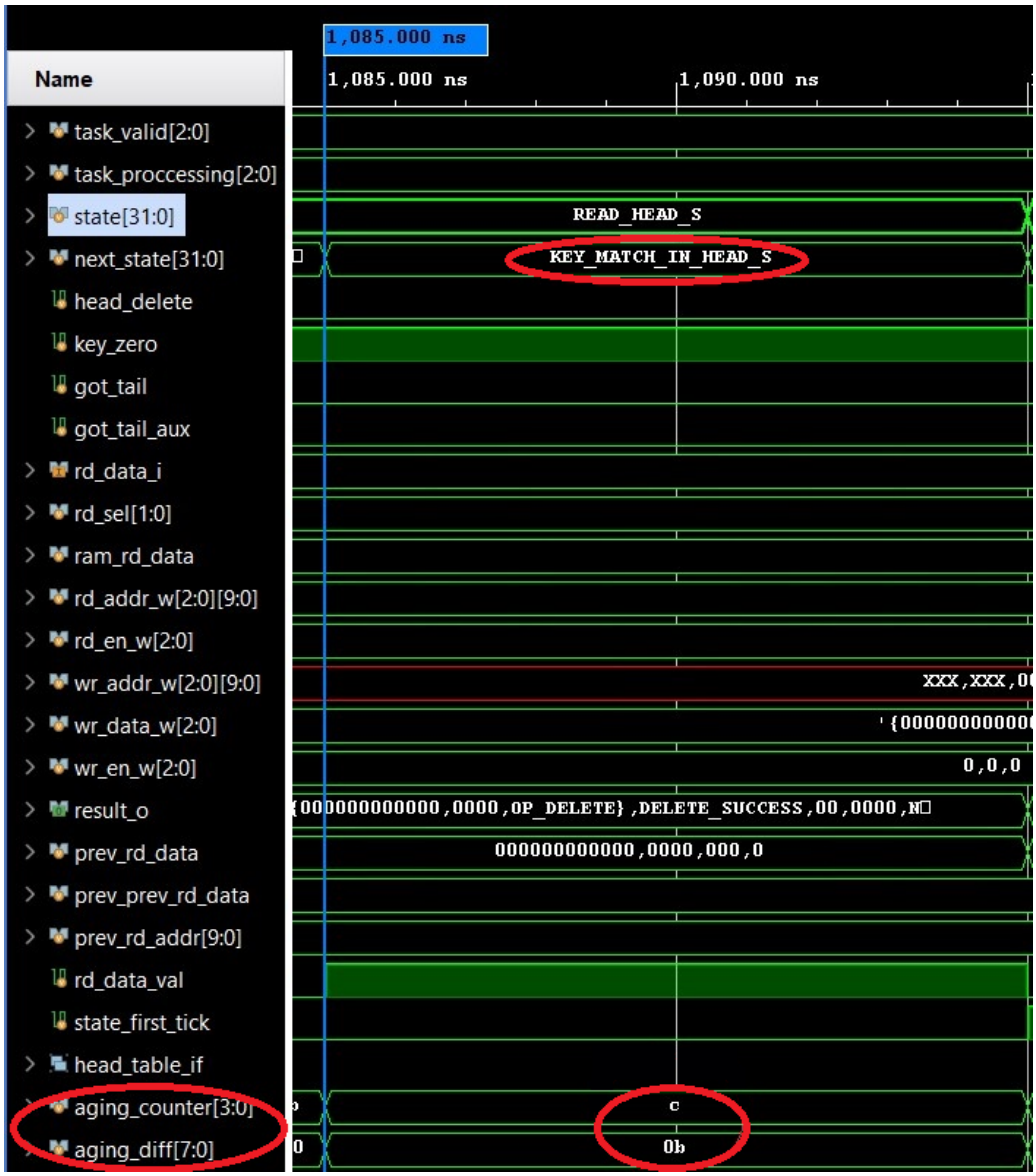

Figure 9. Analyzing the first element in the table that will be deleted.

The third simulation, shown in Figure 11, presents the value update of the second element, analyzed in the previous simulation. During the UPDATE_VAL_WR state (the update state where the data are written), the "prev_rd_data" value is equal to "0002", whereas the "wr_data_w" value (second field) is equal to 0003, meaning that the element value is updated by 1.

4.3. REDSwitch

Finally, the simulation of the REDSwitch core (Figure 5) is presented in Figure 12. This simulation shows that the packet was correctly parsed and duplicated, being sent to the correct pair of FIFOs in the output queues. In this case, the packet needs to be duplicated, and it comes with only one destination port, which would activate a single-output FIFO. This destination port is then modified to activate two output FIFOs—one for each LAN. Each bit in the destination port field corresponds to a specific output FIFO ([FIFO0, FIFO1, FIFO2, FIFO3, FIFO4, FIFO5, FIFO6, FIFO7]). The destination port with the value of 4 "0100" will be codified into "00110000" to activate output FIFOs 2 and 3. Table 5 presents the used port codification. It is possible to see in Figure 12 that the "in_data_A/B" values will go out through "out_data_2/3", as expected.

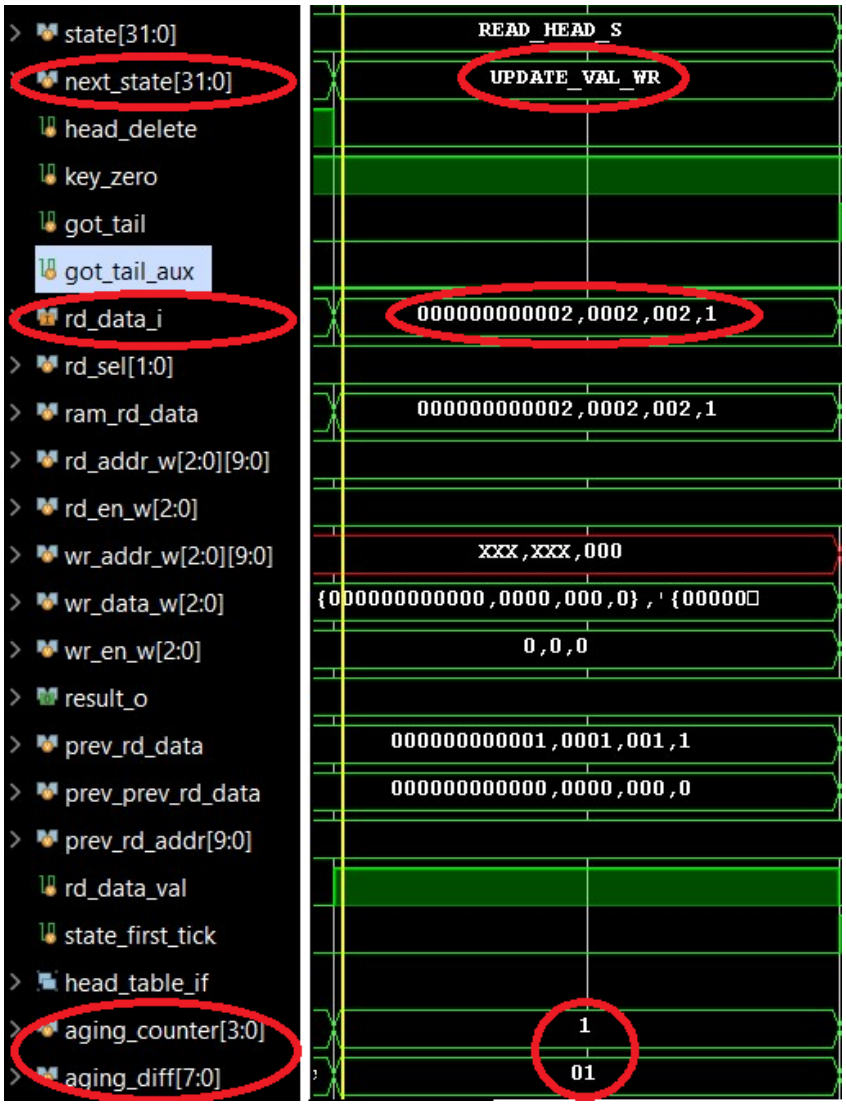

Figure 10. Analyzing the second element in the table that will be updated.

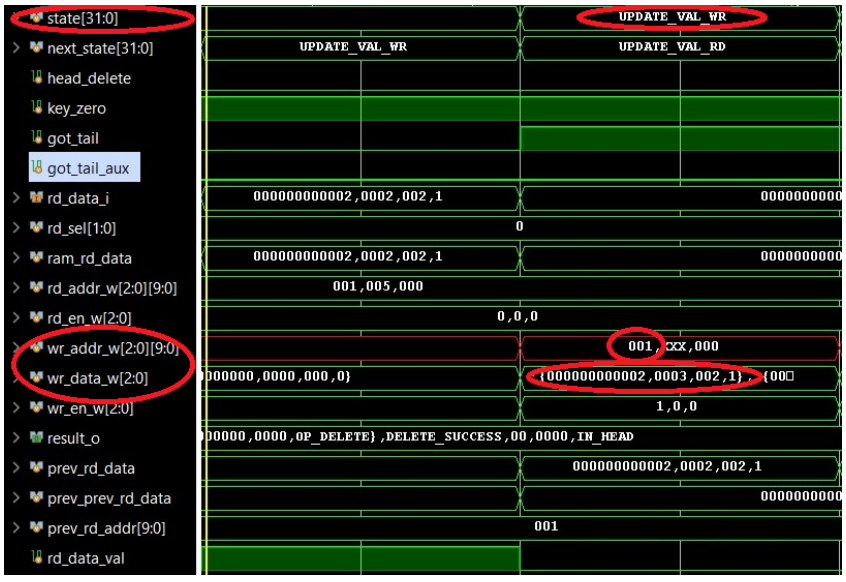

Figure 11. Updating the value of the second element.

Table 5. Port codification.

Port	dst_port_next (Binary)
1	8'b00000110
2	8'b00001100
3	8'b00011000
4	8'b00110000
5	8'b01100000
6	8'b01000010
7	8'b00010010
8	8'b00100100

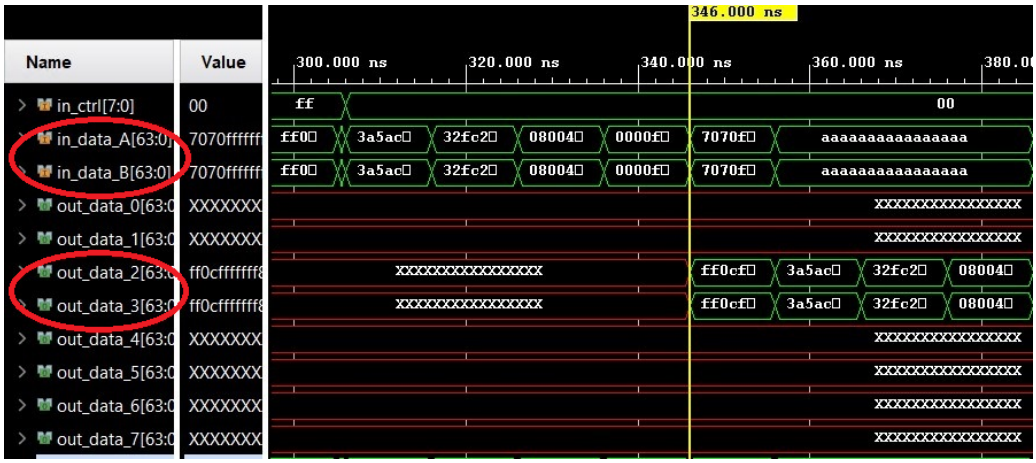

Figure 12. REDSwitch core simulation.

4.4. Ethernet Parser Resource Utilization

Regarding the hardware resources used by the developed switch (REDSwitch), and considering that the main modification to the ONetSwitch (which served as the starting point) was the Ethernet parser, Table 6 presents the post-synthesis results of resource utilization on the Zynq-7000 XC7Z020CLG484-1 board (which has 106,400 flip-flops, 53,200 LUTs, 200 I/Os, and 32 BUFGs) for the Ethernet parsers in both ONetSwitch and REDSwitch. The global buffer (BUFG) utilization increased from 3% to 16%, whereas the look-up table (LUT) usage increased from 1% to 3%. The input/output (I/O) utilization appears excessively high in both cases (96% and 160%) because it refers to a block extracted from within the switch, where multiple internal connections exist. This I/O count does not correspond to the number of board-connected I/O but reflects internal switch connections.

Table 6. Ethernet parser resource utilization.

	ONetSwitch	REDSwitch
FF	1%	3%
LUT	1%	3%
I/O	96%	160%
BUFG	3%	16%

5. Conclusions

This project could be divided into three stages. The first stage was the search and selection of an open-source project that could be used as a starting point for development; the second stage was the development of the Ethernet switch's capabilities; the third stage was the development of the chosen redundant protocol: the PRP.

By using an open-source project, some of the development burden for the Ethernet switch could be relieved, with the redundant protocol (PRP) being the focus. After the analysis of multiple open-source projects related to network communication and Ethernet switches, the REF Switch (by ONetSwitch) was found to be the best candidate to be used as a starting point. The REF Switch filled some of the decided minimum requirements for the Ethernet switch but also had some room for improvement and needed changes to implement the PRP.

The second stage of the project was to develop the Ethernet switch's capabilities further. Thus, it was developed to be able to deal with packets according to the IPv4 protocol and the IEEE 802.1Q standard.

The third and final stage was to implement the PRP protocol on the developed Ethernet switch. The PRP protocol's implementation could be divided into multiple functionalities: the capability of reading/writing the PRP fields at the end of the packets, the duplication of packets, and the acceptance of packets and rejection of the respective copy. Besides the previously mentioned capabilities, it was necessary to have a table to store the sequence numbers (present in the PRP tail) of the received packets, making it possible to identify duplicates. Thus, another open-source project emerged as a good candidate to be used as a hash table. This hash table could insert, search, and delete elements, but other functionalities were needed, namely an aging algorithm, which was developed.

The simulations were performed individually for the second and third stages mentioned above. It was possible to validate the development performed on the Ethernet switch (before the PRP); the duplication, acceptation, and rejection of packets; the operation on the hash table; and the aging algorithm. After testing the functionalities individually, it was also possible to validate the connection between the Ethernet switch and the hash table.

The use of open-source projects was a benefit. The chosen projects proved to be versatile enough to be adapted for different purposes and helped to save time in the development stage, even when facing the challenge of integrating two projects that were not designed to work together.

As future work, we plan to extend the REDSwitch capabilities by adding support for VLANs, integrating the HSR protocol, and implementing the PTP for synchronization.

Author Contributions: Conceptualization, A.T., R.P. and F.M.; methodology, A.T., R.P. and F.M.; software, A.T.; validation, A.T., R.P., F.M. and L.G.; formal analysis, A.T. and R.P; investigation, A.T. and R.P.; data curation, A.T. and R.P.; writing—original draft preparation, A.T.; writing—review and editing, R.P., F.M. and L.G.; visualization, A.T., R.P., F.M. and L.G.; supervision, R.P. and F.M. All authors have read and agreed to the published version of the manuscript.

Funding: This work was funded in part by the Portuguese FCT program, Center of Technology and Systems (CTS) CTS/00066.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article; further inquiries can be directed to the corresponding author.

Conflicts of Interest: André Torres and Rodrigo Piedade were employed by the company EFACEC. At the time of submission, Rodrigo Piedade was affiliated with Siemens. The remaining authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

References

1. Jasperneite, J.; Schumacher, M.; Weber, K. Limits of increasing the performance of Industrial Ethernet protocols. In Proceedings of the 2007 IEEE Conference on Emerging Technologies and Factory Automation, Patras, Greece, 25–28 September 2007; pp. 17–24. [CrossRef]
2. Hansen, K. Redundancy Ethernet in industrial automation. In Proceedings of the 2005 IEEE Conference on Emerging Technologies and Factory Automation, Catania, Italy, 19–22 September 2005; Volume 2, pp. 941–947. . [CrossRef]
3. *IEC 62439-3*; Industrial Communication Networks—High Availability Automation Networks—Part 3: Parallel Redundancy Protocol (PRP) and High-Availability Seamless Redundancy (HSR). International Electrotechnical Commission: Geneva, Switzerland, 2016.
4. Li, Z.; Wan, H.; Deng, Y.; Zhao, X.; Gao, Y.; Gu, M.; Song, X. A Flattened-Priority Framework for Mixed-Criticality Systems. *IEEE Trans. Ind. Electron.* **2020**, *67*, 9862–9872. [CrossRef]
5. Flatt, H.; Jasperneite, J.; Schewe, F. An FPGA based cut-through switch optimized for one-step PTP and real-time Ethernet. In Proceedings of the 2013 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control and Communication (ISPCS) Proceedings, Lemgo, Germany, 22–27 September 2013.
6. Ferrari, P.; Flammini, A.; Marioli, D.; Taroni, A. A Distributed Instrument for Performance Analysis of Real-Time Ethernet Networks. *IEEE Trans. Ind. Inform.* **2008**, *4*, 16–25. [CrossRef]
7. Khalilzadegan, A.; Zarei, M.; Hadidi, K. A New Adaptive Prioritization and Fail-Over Mechanism for Ring Network Adapter. *IEEE Access* **2020**, *8*, 99070–99082. [CrossRef]
8. Kanta, K.; Toumasis, P.; Tokas, K.; Stratakos, I.; Papatheofanous, E.A.; Giannoulis, G.; Mesogiti, I.; Theodoropoulou, E.; Lyberopoulos, G.; Lentaris, G.; et al. Demonstration of a Hybrid Analog–Digital Transport System Architecture for 5G and Beyond Networks. *Appl. Sci.* **2022**, *12*, 2122. [CrossRef]
9. Rodríguez-Andina, J.J.; Valdés-Peña, M.D.; Moure, M.J. Advanced Features and Industrial Applications of FPGAs—A Review. *IEEE Trans. Ind. Inform.* **2015**, *11*, 853–864. [CrossRef]
10. TTTech Flexibilis Oy. HSR/PRP IP Core for FPGA (GbE): Flexibilis Redundant Switch (FRS). Available online: https://www.flexibilis.com/products/frs/ (accessed on 7 June 2024).
11. HPS - HSR-PRP Switch IP Core - SoC-e. Available online: https://soc-e.com/products/hsr-prp-switch-ip-core-all-hardware-low-latency-switch-for-fpgas/ (accessed on 7 June 2024).
12. RED25 Fast Ethernet Entry-Level Switches. Available online: https://www.belden.com/products/industrial-networking-cybersecurity/managed-switches/din-rail-compact/red25 (accessed on 24 January 2024).
13. Seamless Redundancy with PRP and HSR | ZHAW Institute of Embedded Systems InES. Available online: https://www.zhaw.ch/en/engineering/institutes-centres/ines/communication-network-engineering/seamless-redundancy-with-prp-and-hsr (accessed on 24 January 2024).
14. Private Island: Open Source FPGA-Based Network Processor for Privacy, Security, and AI Applications. Available online: https://mindchasers.com/dev/private-island (accessed on 24 February 2024).
15. NetFPGA. Available online: https://netfpga.org (accessed on 7 June 2024).
16. Lockwood, J.W.; McKeown, N.; Watson, G.; Gibb, G.; Hartke, P.; Naous, J.; Raghuraman, R.; Luo, J. NetFPGA–An Open Platform for Gigabit-Rate Network Switching and Routing. In Proceedings of the 2007 IEEE International Conference on Microelectronic Systems Education (MSE'07), San Diego, CA, USA, 3–4 June 2007; pp. 160–161. [CrossRef]
17. RealTimeSwitch · NetFPGA/netfpga Wiki · GitHub. Available online: https://github.com/NetFPGA/netfpga/wiki/RealTimeSwitch (accessed on 7 June 2024).
18. OpenFlowNetFPGA100 · NetFPGA/netfpga Wiki · GitHub. Available online: https://github.com/NetFPGA/netfpga/wiki/OpenFlowNetFPGA100 (accessed on 7 June 2024).
19. ONetSwitch: Open Source Hardware for Networking by MeshSr - Kickstarter. Available online: https://www.kickstarter.com/projects/onetswitch/onetswitch-open-source-hardware-for-networking (accessed on 7 June 2024).
20. Hu, C.; Yang, J.; Zhao, H.; Lu, J. Design of All Programable Innovation Platform for Software Defined Networking. In Proceedings of the Open Networking Summit 2014 (ONS 2014), Santa Clara, CA, USA, 2–4 March 2014.
21. REF Switch · MeshSr/ONetSwitch Wiki · GitHub. Available online: https://github.com/MeshSr/ONetSwitch/wiki/REF-Switch (accessed on 7 June 2024).
22. IEEE 802.1: 802.1Q-Virtual LANs. Available online: https://www.ieee802.org/1/pages/802.1Q.html (accessed on 7 June 2024).
23. ASICS.ws-Solutions for Your ASIC/FPGA and System Design Needs. Available online: https://www.asics.ws/tools_sub.html (accessed on 8 October 2024).
24. GitHub-johan92/fpga-Hash-Table: Simple Hash Table on Verilog (SystemVerilog). Available online: https://github.com/johan92/fpga-hash-table (accessed on 7 June 2024).

25. Cat Karat Packet Builder. Available online: https://www.softpedia.com/get/Network-Tools/Network-Testing/Cat-Karat-Packet-Builder.shtml (accessed on 8 October 2024).
26. Online CRC-8 CRC-16 CRC-32 Calculator. Available online: https://crccalc.com/ (accessed on 8 October 2024).