RedHsrPrp

Reference Manual

Product Info	
Product Manager	Sven Meier
Author(s)	Sven Meier
Reviewer(s)	-
Version	2.4
Date	27.02.2020

Copyright Notice

Disclaimer

The information available to you in this document/code may contain errors and is subject to periods of interruption. While NetTimeLogic GmbH does its best to maintain the information it offers in the document/code, it cannot be held responsible for any errors, defects, lost profits, or other consequential damages arising from the use of this document/code.

NETTIMELOGIC GMBH PROVIDES THE INFORMATION, SERVICES AND PRODUCTS AVAILABLE IN THIS DOCUMENT/CODE "AS IS," WITH NO WARRANTIES WHATSOEVER. ALL EXPRESS WARRANTIES AND ALL IMPLIED WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT OF PROPRIETARY RIGHTS ARE HEREBY DISCLAIMED TO THE FULLEST EXTENT PERMITTED BY LAW. IN NO EVENT SHALL NETTIMELOGIC GMBH BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, CONSEQUENTIAL, SPECIAL AND EXEMPLARY DAMAGES, OR ANY DAMAGES WHATSOEVER, ARISING FROM THE USE OR PERFORMANCE OF THIS DOCUMENT/CODE OR FROM ANY INFORMATION, SERVICES OR PRODUCTS PROVIDED THROUGH THIS DOCUMENT/CODE, EVEN IF NETTIMELOGIC GMBH HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

IF YOU ARE DISSATISFIED WITH THIS DOCUMENT/CODE, OR ANY PORTION THEREOF, YOUR EXCLUSIVE REMEDY SHALL BE TO CEASE USING THE DOCUMENT/CODE.

Overview

The HSR&PRP Core from NetTimeLogic is a standalone Network Redundancy Core according to IEC62439-3 rev3. It allows to connect to a redundant network supporting either the Parallel Redundancy Protocol (PRP) or the High-availability Seamless Redundancy Protocol (HSR). It can either run as a Dual Attached Node (DAN) being an endpoint or it can run as a Redundancy Box (RedBox) bridging between a redundant and a non-redundant network.
The core has three ports: A redundant pair named Ports A&B and an uplink named Port C. The basic principal is the same for both protocols: duplicating and tagging frames on the path from Port C to Ports A&B and duplicate rejection and untagging on the path from Ports A&B to Port C. It can support a configurable number of nodes on the redundant side and also on the non-redundant network (when run as a RedBox). It makes a basic supervision of the redundant network and sends supervision frames for its non-redundant connected nodes. The core learns the connected nodes itself and needs no further configuration.

The HSR&PRP Core is intercepting the path between two Ethernet PHYs and an Ethernet core that forwards or handles Ethernet frames (MAC or Switch).

All tables, protocols and algorithms are implemented in the core, no CPU is required. This allows running network redundancy completely independent and standalone from the user application. The core can be configured either by signals or by an AXI4Light-Slave Register interface.

Key Features:

- Supports the HSR and PRP redundancy protocol according to IEC62439-3 rev 3
- Can run as Dual Attached Node (DAN) or as Redundancy Box (RedBox)
- Supports HSR Mode H and Mode X and PRP Duplicate Discard Mode
- PTP aware for use with PTP Utility Profile
- Intercepts path between MAC and two PHYs (DAN) or three PHYs (RedBox)
- Configurable number of nodes supported on Port C and Promiscuous Mode
- Full line speed
- AXI4 Light register set or static configuration
- MII/RMII/GMII/RGMII Interface support
- Hardware supervision handling
- Optional frame and error counters per Port

- Optional VLAN tagging and filtering
- Optional Tail Tagging mode
- Optional cut through frame processing
- Optional HSR-PRP or HSR-HSR RedBox mode support

Revision History

This table shows the revision history of this document.

Version	Date	Revision
0.1	16.06.2017	First draft
1.0	28.07.2017	First release
1.1	17.08.2017	NO mode added
2.0	24.07.2018	Adaptations after merge with TSN and added Port Status and Promiscuous Mode
2.1	14.09.2018	Reworked and Link added
2.2	11.12.2018	VLAN mode changed, tail tagging added and PRP untagging as option
2.3	07.08.2019	HSR-PRP and HSR-HSR Mode added
2.4	27.02.2020	Added cut through mode

Table 1: Revision History

Content

Definitions

Definitions	
Redundancy Box	Redundancy Box according to IEC62439-3
Single Attached Node	A node that does not support redundancy
Dual Attached Node	A node that does support redundancy with HSR or PRP
Virtual Dual Attached Node	A node connected via a Redundancy Box to the redundant network

Table 2: Definitions

Abbreviations

Abbreviations	
AXI	AMBA4 Specification (Stream and Memory Mapped)
IRQ	Interrupt, Signaling to e.g. a CPU
SAN	Single Attached Node
DAN	Dual Attached Node
VDAN	Virtual DAN
RedBox	Redundancy Box
QuadBox	Two RedBoxes bridging between two redundant Networks
PTP	Precision Time Protocol according to IEEE1588
PRP	Parallel Redundancy Protocol according to IEC62439-3
HSR	High-availability Seamless Redundancy Protocol according to IEC62439-3
TB	Testbench
LUT	Look Up Table
LSDU	Link Service Data Unit (Data in a frame)
LPDU	Link Protocol Data Unit (Data in a frame)
FF	Flip Flop

FCS	Frame Check Sequence also known as CRC
FIFO	First In First Out Buffer
RAM	Random Access Memory
RCT	Redundancy Control Trailer
ROM	Read Only Memory
FPGA	Field Programmable Gate Array
VHDL	Hardware description Language for FPGA's

Table 3: Abbreviations

1 Introduction

1.1 Context Overview

The HSR&PRP Redundancy Core is meant as a co-processing handling network redundancy. intercepts the Media Independent Interface (MII) on the Ethernet path between the MAC and PHY where it handles all redundancy traffic. This means it duplicates and tags frames coming from the Ethernet MAC and does duplicate discarding and untagging on the paths from the Ethernet PHYs. If run in HSR mode it also does the forwarding according to IEC62439.3 between the Ethernet PHYs. It also generates and processes HSR&PRP Supervision frames directly in hardware using the same data path as normal traffic coming from or going to the Ethernet MAC. This also means that it uses a small amount (around 1 frame every 2 seconds per node) of the bandwidth on the MII so if 100% Network traffic shall be constantly sent by the Ethernet MAC it would eventually drop some frames to still handle HSR&PRP. The HSR&PRP Redundancy Core is designed to work in cooperation with the Counter Clock core from NetTimeLogic (not a requirement). It contains an AXI4Light slave for configuration from a CPU, this is however not required since the HSR&PRP Core can also be configured statically via signals/constants directly from within the FPGA.
The HSR&PRP Core can be combined with a PTP Transparent Clock from NetTimeLogic to handle also time synchronization.

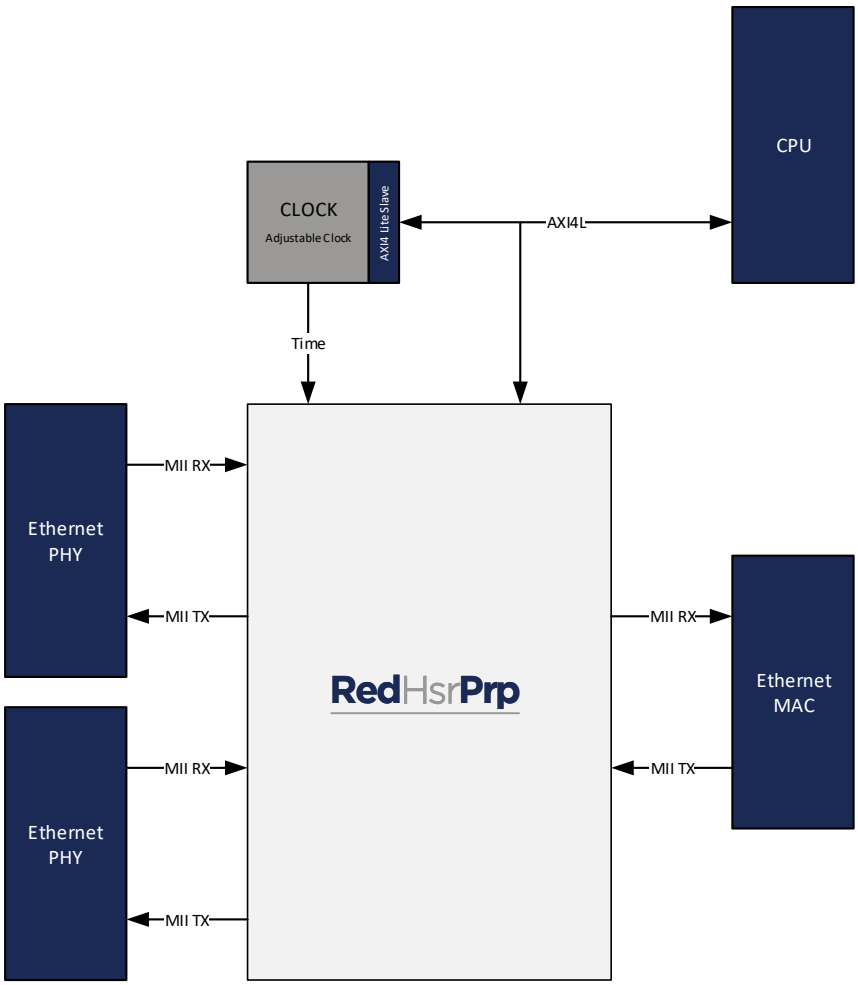

Figure 1: Context Block Diagram

1.2 Function

The HSR&PRP core handles the redundancy according to IEC63439-3. It duplicates and tags frames in the path to the PHYs and discards duplicates and untags frames on the path to the MAC. It optionally contains a Table which contains all nodes connected on the MAC side when run as RedBox. It sends and analyzes HSR/PRP Supervision frames and can do forwarding of frames between the PHY ports if run in HSR mode.

1.3 Architecture

The core is split up into different functional blocks for reduction of the complexity, modularity and maximum reuse of blocks. The interfaces between the functional blocks are kept as small as possible for easier understanding of the core.

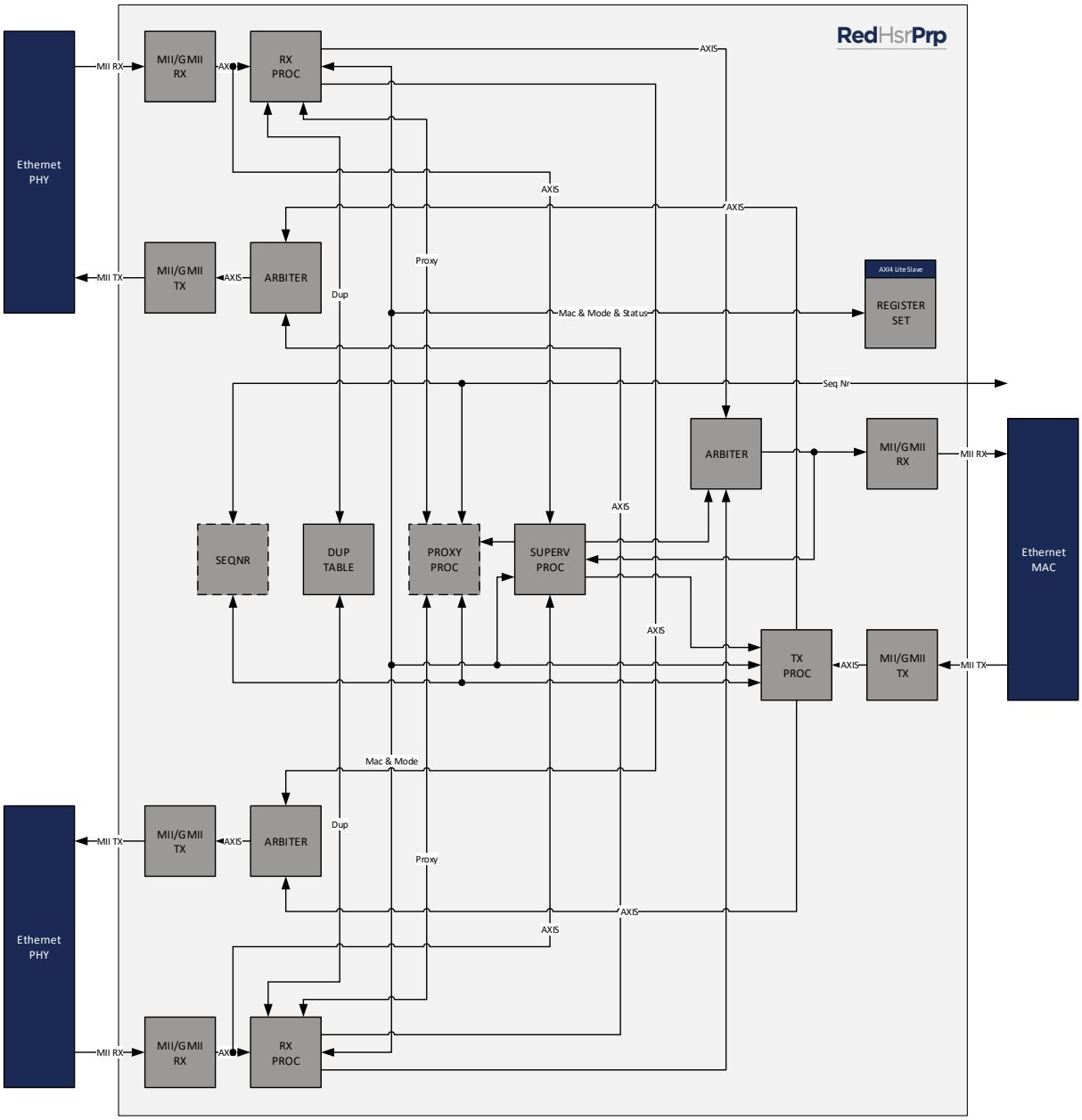

Figure 2: Architecture Block Diagram

Rx Processor

This is one of the core modules, it buffers, parses and untags frames as well as taking the forwarding decision whether to forward frames on Ports A/B or Port C. It gets the duplicate decision from the Duplicate Processor and if in RedBox mode also the source Decision from the Proxy Node Processor

Tx Processor

This is the other core module, it buffers, parses, tags frames as well as duplicating frames and forwards them to Ports A & B. It gets the sequence number information from the Sequence Number Processor or if in RedBox mode from the Proxy Node Processor.

Supervision Processor

This block sends Supervision frame for this node and if in RedBox mode also for all nodes in the Proxy Node Table. It also parses the incoming Supervision frames and signals the reception to the Supervision block within which shows the logical link status based on a timer which signals whether supervision frames have been received in the last 5 Supervision frame intervals or not.

Duplicate Processor

This block is in charge of detecting duplicates, it contains a Duplication Table which is a pooled Hash Table for each frame received in the redundant Network. Per default it has 2**14 entries, meaning it can handle up to 16k frames in the redundant network. The number of entries is configurable at compile time. Whenever a tagged frame is received it makes an entry in the table for that frame. When the duplicate is received, the entry is cleared. When no duplicate is received within 400ms the entry is aged out.

Proxy Node Processor

This block is only available when run in RedBox mode, it contains entries of all nodes (per default 64, but can be changed at compile time) connected to Port C. It stores the MAC address and a Sequence Number which is used for Supervision frames sending, tagging of frames on sending and in the forward decision when run in HSR mode (frames that are source or destination of a node in the Proxy Node table shall be removed from the ring). Nodes in the table are searched in a linear manner but parallelized to achieve the required speed.

Sequence Number Processor

This module is only available when not run in RedBox mode, it generates Sequence Number for the tagging.

Frame Arbiter

This block multiplexes the frames coming from Ports A & B towards Port C or from Ports A/B &C towards Ports A/B.

(R)(G)MII Receive/Transmit Interface Adapter

These blocks convert the data stream from the (R)(G)MII to a 32bit AXI stream and back form 32bit AXI stream to (R)(G)MII.

Register Set

This block allows reading status values and writing configuration.

1.4 Deviations from the Standard or Limitations

The deviations and limitations below apply to the core; however, the core is fully compatible with IEC62439-3:

- No Duplicate detection for frames looping in the ring more than once, no strict mode H, can be partly handled by mode X though
- No special SAN handling for PRP, sending always to both ports tagged
- No Supervision counters and Error counters
- No Node Table for Supervision
- PTP Traffic is handled special
- No padding between RCT and CRC for PRP allowed
- No padding of frames when frames are less than 64 bytes after untagging or before tagging
- No IP for core
- Optional cut through frame processing
- No Duplicate Accept mode, only Duplicate Discard
- No wait after reboot
- Allows frame size up to 2044 bytes, but no Jumbo frames (which is according to standard)
- No priority queues
- Only one VLAN supported
- No SNMP support by the core
- No QuadBox support or coupling with other HSR or PRP networks, only PRP/HSR ⇔ SAN per default HSR-PRP or HSR-HSR mode can be enabled optionaly
- Frames are sent independent of the physical link status
- No Transparent Reception for PRP
- No Bridging Mode for PRP
- DAN or RedBox own MAC has to be configured, no self-learning
- No Multicast Filter Table

2 PRP, HSR and Tail Tagging Basics

2.1 Network Redundancy (HSR/PRP) Basics

The two redundancy protocols: Parallel Redundancy Protocol (PRP) and High-availability Seamless Redundancy Protocol (HSR) are closely linked and both standardized in IEC62439-3. The goal of the two Network Redundancy Protocols is to get rid of single point of failures in the network connections, it does not avoid single point of failures in the devices. They use similar principals to achieve redundancy by tagging and untagging frames and duplicating and discarding duplicate frames and mostly differ on the network topology used. The tags contain a Protocol Identifier, a Size Field, a LAN Identifier and most important a Sequence Number to identify frames. Each frame has an individual sequence number per source MAC which is used to do duplicate discarding. They also send periodically so called Supervision frames which allows to supervise the status of the redundant network, e.g. broken links. Each node contains of at least three Ports: The redundant pair Ports A&B and the uplink Port C.
There is also a close coupling to time synchronization according to IEEE1588, which defines a combination of the redundancy with time synchronization in IEC61850-9-3 (Utility Profile). NetTimeLogic's PTP cores can be combined with this core to get a redundant time synchronization solution.

2.1.1 PRP

The Parallel Redundancy Protocol (PRP) is defined in chapter 4 of IEC62439-3. The principal is quite simple, it duplicates and tags frames on sending (from Port C to Ports A&B) and does duplicate discarding and untagging on reception (from Ports A&B to Port C). PRP uses two individual networks (LAN A&B) of any topology to achieve redundancy. There is no forwarding between the two networks but the two networks shall have a similar delay. As mentioned frames are tagged on sending, this tag is added at the end of the frame right before the FCS. In case of padding this is inserted between the LPDU and the PRP tag, so the tag is always at the same position in the frame (right before the FCS). Tagging at the end has another advantage, all network nodes in the two individual networks do not have to be PRP aware and just treat the tag as padding. This also means that Single Attached Nodes (SAN) which are only connected to one of the networks can communicate directly with DANs and vice versa ignoring the tags and not creating tags (See 2.1.4, 2.1.5, 2.1.6 and 2.1.7 for details).

A frame is always sent to both networks and the frames are forwarded in the two networks according to switching rules. On reception of a tagged frame for which this node is destination the duplicate discard algorithm is run and only the first frame received forwarded to Port C.

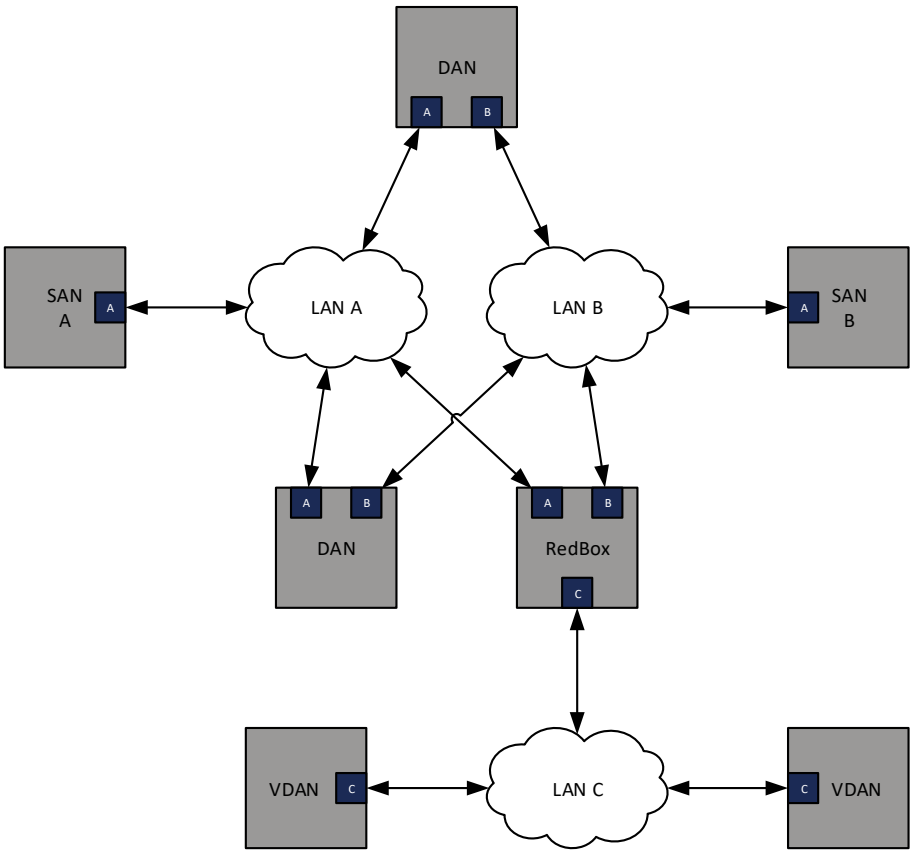

Figure 3: PRP

The frame format of a PRP tagged frame looks as following:

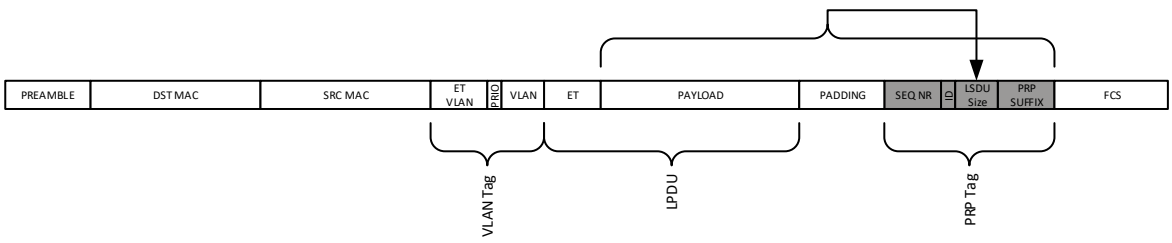

Figure 4: PRP frame

Fields of the PRP tag:

- SEQ NR: 16 bit monotonous increasing (by 1 wrapping through zero) Sequence Number
- ID: 4bit LAN Identifier, 0xA for LAN A, 0xB for LAN B
- LSDU Size: 12bit LSDU Size in bytes including payload and PRP Tag
- PRP Suffix: 16bit PRP Tag Identifier: 0x88FB

WARNING! Frame payload of a non-PRP-tagged frame can always be wrongly detected as a PRP Tag (e.g. Frames from a SAN). This is not critical for the duplication rejection but is critical when untagging, since when the wrongly detected PRP Tag is removed the frame is corrupted because it actually cuts of the last 6 bytes of real payload

See IEC62439-3 chapter 4 for details.

2.1.2 HSR

The High-availability Seamless Redundancy Protocol (HSR) is defined in chapter 5 of IEC62439-3. The principal of HSR is very similar to the one of PRP. It also duplicates and tags frames on sending (from Port C to Ports A&B) and does duplicate discarding and untagging on receiving from Ports A&B to Port C). In contrast to PRP it uses a ring structure for redundancy rather than two individual networks and inserts the tags before the Ethertype of the LPDU rather than appending it at the end of the frame. This means on one hand that there is frame forwarding between the Ports A&B and each node in the redundant network needs to support HSR. There is no scheme to connect Single Attached Nodes (SAN) directly to the ring, they can only be connected via a RedBox to the ring, this can be either a single node or a whole network (See 2.1.4, 2.1.5, 2.1.6 and 2.1.7 for details).
A frame is always sent into both directions of the ring and each node forwards the frame to the next node if it is not destination or source of the frame. If it is the destination (unicast) it removes the frame from the ring and runs the duplicate discard algorithm and forwards the first occurrence of the frame to its Port C. For multicast frames the frame is forwarded on the ring but also the duplicate discard algorithm run towards Port C, the node who injected the frame on the ring is the one that removes the frame from the ring again. There is a special mode called Mode X which does also duplicate rejection on the forwarding path, meaning that only the first occurrence of a frame is forwarded and the duplicate rejected, since it has already passed the ring in both directions. This mode reduces the overall net-

work load since multicast frames do not pass through the whole ring, on the other hand frame discarding gets non-deterministic.

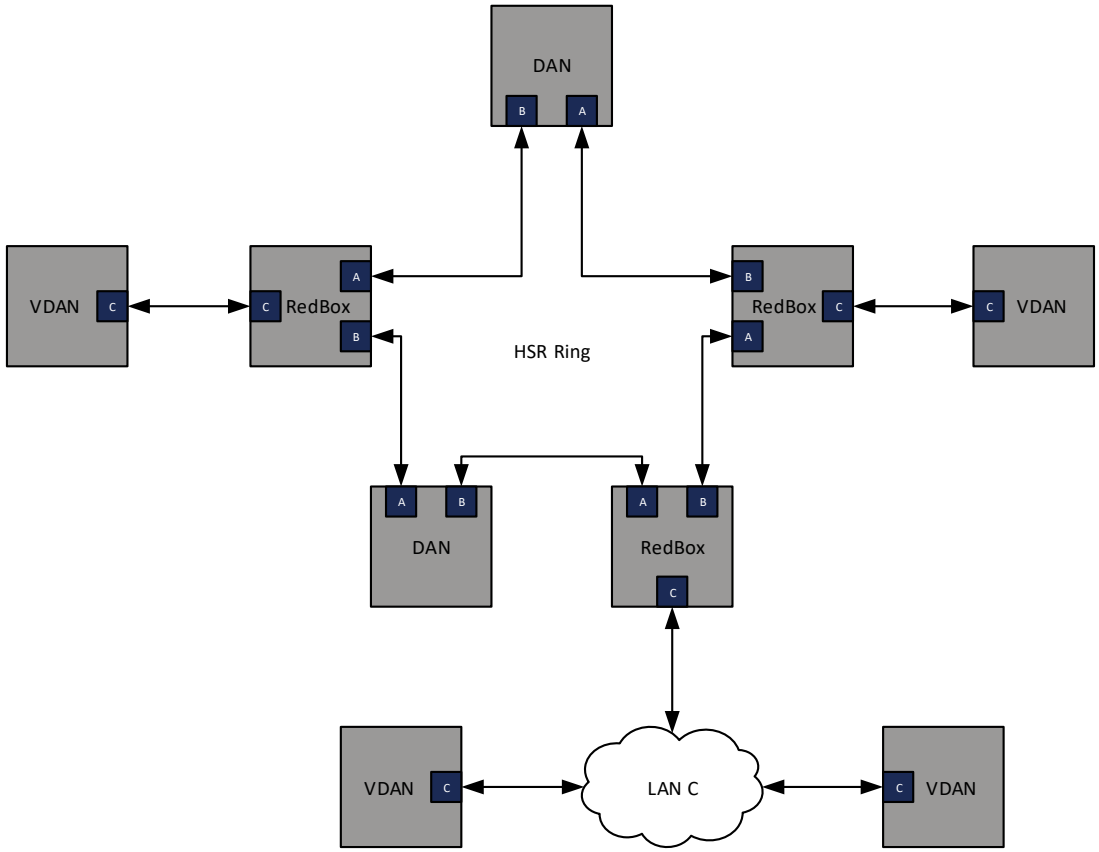

Figure 5: HSR

The frame format of an HSR tagged frame looks as following:

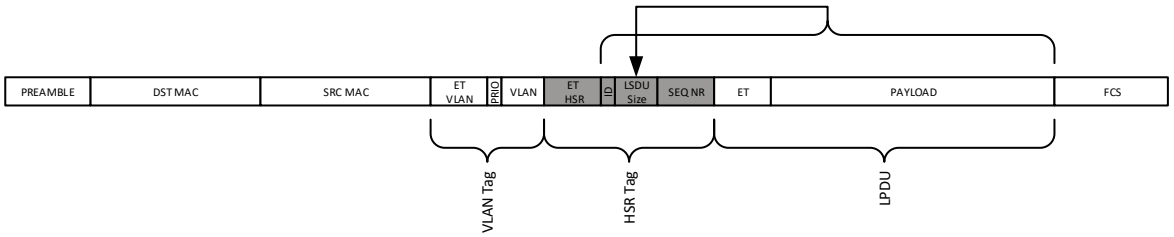

Figure 6: HSR frame

Fields of the HSR tag:

- HSR Ethertype: 16bit HSR Tag Identifier: 0x892F
- SEQ NR: 16 bit monotonous increasing (by 1 wrapping through zero) Sequence Number

- ID: 4bit Path Identifier, 0x0 for Path A, 0x1 for Path B
- LSDU Size: 12bit LSDU Size in bytes including payload and Ethertype and parts of the HSR Tag

See IEC62439-3 chapter 5 for details.

2.1.3 NO

The NO mode is basically the same like HSR but without tagging and duplicate rejection. So, it duplicates frames on the path from Port C to Port A&B and forwards all frames from Port A&B to Port C (which match the Destination or Multicast). It also forwards frames between the Ports A&B, removing frames on the forwarding where it is Destination or Source.
In this mode no Supervision frames are sent or checked. No tagging or untagging is done.

2.1.4 DAN

A Dual Attached Node (DAN) is an end node participating directly in the redundancy protocols (duplication/duplicate discarding and tagging/untagging). A DAN has only one MAC and sends Supervision frames on behalf of this node only. In case of HSR it removes frames from the ring only for this MAC. Often a DAN is built from a RedBox supporting only one MAC on Port C.

2.1.5 VDAN

A Virtual Dual Attached Node (VDAN) is also an end node with only one MAC, but connected true a RedBox to the redundant network. It does not participate in the redundancy protocols, this is taken care of in the RedBox, so in particular it does no duplication and duplicate discarding or tagging and untagging neither does it send Supervision frames. A VDAN has a single point of failure at the interlink between the RedBox and the node itself.

2.1.6 SAN

A Single Attached Node (SAN) is also an end node which does not participate in the redundancy protocols. So it does neither duplication and duplicate discarding nor tagging and untagging. In case of PRP it can be directly connected to either of the networks (LAN A or B) but only to one at the time. A SAN has a single point of failure since it is connected to only one of the LANs. If a SAN is connected behind a RedBox it is called a VDAN.

2.1.7 RedBox

A Redundancy Box (RedBox) is a network node acting as bridge between the redundant networks and a non-redundant network. It has its own MAC address but it's main task is to act as DANs for the VDANs connected on the non-redundant network. It contains a so called ProxyNodeTable which has entries for each VDAN connected to it. It does duplication of the frames received from Port C towards Port A&B and tags them with an individual monotonous incrementing Sequence Number for each node (MAC) in the ProxyNodeTable. It also creates Supervision frames for each node in the ProxyNodeTable, marking the frames that they were sent via a RedBox. On reception, it does run the duplicate discard algorithm and untagging for all frames, not only the one it is destination. In case of HSR it removes the frames, which are addressed to or sourced by a node in the ProxyNodeTable from the ring acting like multiple DANs in a single device.
In addition to this functionality, a RedBox acts as DAN for frames sourced by or addressed to the MAC of the RedBox itself.

2.1.8 Supervision

Both protocols define so called Supervision frames (with basically the same format) which have a reserved multicast destination MAC range (01:15:4E:00:01:XX) and Ethertype (0x88FB). These frames are sent periodically which allows other nodes, and the node itself to supervise the status of the redundant network. If not both of the Supervision frames are received from a node, a logical or physical link is broken. In case of HSR the node can supervise its own status by checking if it receives back the supervision frames sent by itself. In case of PRP the Supervision frames also identify DANs from SANs which is needed for untagging frames.

2.2 Tail Tagging Basics

Tail Tagging is used to know from which Port (A or B) the frame was received and if it was redundancy tagged when forwarded to Port C and to define when a frame is sent to Port C on which Port (A and/or B) shall be sent (bitmask: only A, only B or both) and if the frame shall be redundancy and/or VLAN tagged or not.
Tail Tagging is often used by switches for e.g. LLDP frames (for e.g. RSTP).
The Tail Tag is a single Byte which is inserted right before the FCS, which means it is part of the payload and the FCS is generated over the Tail Tag.
The Tail Tag will be removed before the frame is forwarded to Ports A & B and it is added before the frame is forwarded to Port C.

The format of the Tail Tag is as following:

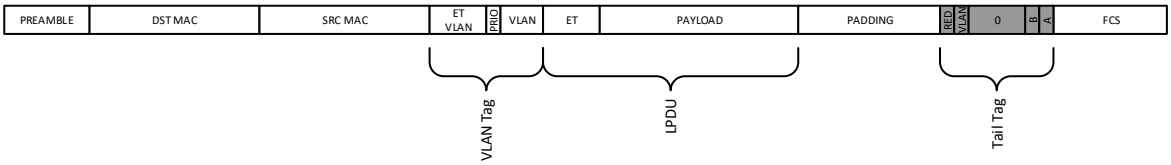

Figure 7: Tail Tagged frame

Fields of the Tail tag:

- RED: Bit 7, indicating if the frame was redundancy tagged on reception (from Port A & B) or if the frame shall be redundancy tagged on transmission (to Port A & B)
- VLAN: Bit 6, indicating if the frame was VLAN tagged on reception (from Port A & B) or if the frame shall be VLAN tagged on transmission, if VLAN tagging is enabled (to Port A & B)
- Port B: Bit 1, indicating if the frame is coming from Port B on reception or if the frame shall be sent to Port B on transmission
- Port A: Bit 0, indicating if the frame is coming from Port A on reception or if the frame shall be sent to Port A on transmission

WARNING! When Tail Tagging is enabled every frame going to Port C, has to be Tail Tagged, otherwise the last byte before the FCS will be treated as Tail Tag which will result in an undefined behavior.

3 Register Set

This is the register set of the HSR&PRP Core. It is accessible via AXI4 Light Memory Mapped. All registers are 32bit wide, no burst access, no unaligned access, no byte enables, no timeouts are supported. Register address space is not contiguous. Register addresses are only offsets in the memory area where the core is mapped in the AXI interconnects. Non existing register access in the mapped memory area is answered with a slave decoding error.

3.1 Register Overview

Registerset Overview			
Name	Description	Offset	Access
Red HsrPrpControl Reg	HsrPrp Enable Control Register	0x00000000	RW
Red HsrPrpStatus Reg	HsrPrp Error Status Register	0x00000004	WC
Red HsrPrpVersion Reg	HsrPrp Version Register	0x0000000C	RO
Red HsrPrpFrameCountControl Reg	HsrPrp Status Control Register	0x00000010	RW
Red HsrPrpRxFrameCountA Reg	HsrPrp Received Frames Count Port A Register	0x00000020	RO
Red HsrPrpRxErrCountA Reg	HsrPrp Received Error Frames Count Port A Register	0x00000024	RO
Red HsrPrpTxFrameCountA Reg	HsrPrp Transmitted Frames Count Port A Register	0x00000030	RO
Red HsrPrpTxErrCountA Reg	HsrPrp Transmitted Error Frames Count Port A Register	0x00000034	RO
Red HsrPrpRxFrameCountB Reg	HsrPrp Received Frames Count Port B Register	0x00000040	RO
Red HsrPrpRxErrCountB Reg	HsrPrp Received Error Frames Count Port B Register	0x00000044	RO
Red HsrPrpTxFrameCountB Reg	HsrPrp Transmitted Frames Count Port B Register	0x00000050	RO
Red HsrPrpTxErrCountB Reg	HsrPrp Transmitted Error Frames Count Port B Register	0x00000054	RO
Red HsrPrpRxFrameCountC Reg	HsrPrp Received Frames Count Port C Register	0x00000060	RO
Red HsrPrpRxErrCountC Reg	HsrPrp Received Error Frames Count Port C Register	0x00000064	RO

Red HsrPrpTxFrameCountC Reg	HsrPrp Transmitted Frames Count Port C Register	0x00000070	RO
Red HsrPrpTxErrCountC Reg	HsrPrp Transmitted Error Frames Count Port C Register	0x00000074	RO
Red HsrPrpConfigControl Reg	HsrPrp Configuration Control Register	0x00000080	RW
Red HsrPrpConfigMode Reg	HsrPrp Configuration Mode Register	0x00000084	RW
Red HsrPrpConfigVlan Reg	HsrPrp Configuration VLAN Register	0x00000088	RW
Red HsrPrpMacControl Reg	HsrPrp MAC Control Register	0x00000100	RW
Red HsrPrpMac1 Reg	HsrPrp MAC 0-3	0x00000104	RW
Red HsrPrpMac2 Reg	HsrPrp MAC 4-5	0x00000108	RW

Table 4: Register Set Overview

3.2 Register Descriptions

3.2.1 General

3.2.1.1 RED HsrPrp Control Register

Used for general control over the HSR&PRP Core, all configurations on the core shall only be done when disabled.

RED HsrPrpControl Reg

Reg Description

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

Reset: 0x00000000

Offset: 0x0000

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:1	RO
ENABLE	Enable	Bit: 0	RW

3.2.1.2 RED HsrPrp Status Register

Shows the current status of the RED HsrPrp core, status bits of supervision timeouts.

RED HsrPrpStatus Reg																												
Reg Description																												
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																					LINK_C	LINK_B	LINK_A	-						TIMEOUT_B	TIMEOUT_A
RO																					RO	RO	RO	RO						WC	WC
Reset: 0x00000000																															
Offset: 0x0004																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:11	RO
LINK_C	Link state of Port C	Bit: 10	RO
LINK_B	Link state of Port B	Bit: 9	RO
LINK_A	Link state of Port A	Bit: 8	RO
-	Reserved, read 0	Bit:7:2	RO
TIMEOUT_B	No Supervision frames were received on B for the defined number supervision intervals	Bit: 1	WC
TIMEOUT_A	No Supervision frames were received on A for the defined number supervision intervals	Bit: 0	WC

3.2.1.3 RED HsrPrp Version Register

Version of the IP core, even though is seen as a 32bit value, bits 31 down to 24 represent the major, bits 23 down to 16 the minor and bits 15 down to 0 the build numbers.

RED HsrPrpVersion Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
VERSION																															
RO																															
0xXXXXXXXX																															
Offset: 0x000C																															

Name	Description	Bits	Access
VERSION	Version of the core	Bit: 31:0	RO

3.2.1.4 RED HsrPrp Frame Count Control Register

Used for clearing all statistic counters

Only available if the generic PortStatusSupport_Gen is true.

RED HsrPrpFrameCountControl Reg																													
Reg Description																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															CLEAR
RO																															RW
Reset: 0x00000000																															
Offset: 0x0010																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:1	RO
CLEAR	Clear all counters (self cleared)	Bit: 0	RW

3.2.1.5 RED HsrPrp RX Frame Count Registers

Received number of frames on the specific Port [X]: X= A, B, C. Offsets: Port A: 0x0020, Port B: 0x0040, Port C: 0x0060.
Includes also erroneous frames.
Only available if the generic PortStatusSupport_Gen is true.

Red HsrPrpRxFrameCount[X] Reg																														
Reg Description																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX_FRAMES																														
RO																														
0x00000000																														
Offset: 0x0020, 0x0040, 0x0060																														

Name	Description	Bits	Access
RX_FRAMES	Received number of frames on Port [X] X=A,B,C	Bit: 31:0	RO

3.2.1.6 RED HsrPrp RX Error Count Registers

Received number of erroneous frames on the specific Port [X]: X= A, B, C. Offsets: Port A: 0x0024, Port B: 0x0044, Port C: 0x0064. Only available if the generic PortStatusSupport_Gen is true.

Red HsrPrpRxErrCount[X] Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX_ERR																															
RO																															
0x00000000																															
Offset: 0x0024, 0x0044, 0x0064																															

Name	Description	Bits	Access
RX_ERR	Received number of erroneous frames on Port [X] X=A,B,C	Bit: 31:0	RO

3.2.1.7 RED HsrPrp TX Frame Count Registers

Transmitted number of frames on the specific Port [X]: X= A, B, C. Offsets: Port A: 0x0030, Port B: 0x0050, Port C: 0x0070.
Includes also erroneous frames.
Only available if the generic PortStatusSupport_Gen is true.

Red HsrPrpTxFrameCount[X] Reg																														
Reg Description																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX_FRAMES																														
RO																														
0x00000000																														
Offset: 0x0030, 0x0050, 0x0070																														

Name	Description	Bits	Access
RX_FRAMES	Transmitted number of frames on Port [X] X=A,B,C	Bit: 31:0	RO

3.2.1.8 RED HsrPrp TX Error Count Registers

Transmitted number of erroneous frames on the specific Port [X]: X= A, B, C. Offsets: Port A: 0x0034, Port B: 0x0054, Port C: 0x0074.
Only available if the generic PortStatusSupport_Gen is true.

Red HsrPrpTxErrCount[X] Reg																														
Reg Description																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RX_ERR																														
RO																														
0x00000000																														
Offset: 0x0034, 0x0054, 0x0074																														

Name	Description	Bits	Access
RX_ERR	Transmitted number of erroneous frames on Port [X] X=A,B,C	Bit: 31:0	RO

3.2.1.9 RED HsrPrp Config Control Register

Configuration valid bits, used to mark the corresponding fields as valid.

RED HsrPrpConfigControl Reg																														
Reg Description																														
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																														VLAN_VAL	MODE_VAL
RO																														RW	RW
Reset: 0x00000000																															
Offset: 0x0080																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:2	RO
VLAN_VAL	VLAN valid (autocleared)	Bit: 1	RW
MODE_VAL	Mode valid (autocleared)	Bit: 0	RW

3.2.1.10 RED HsrPrp Config Mode Register

Mode to run the core in, Either HSR, PRP or NON. All other modes all traffic is dropped. If Promiscuous mode is enabled all traffic will be forwarded to Port C, duplicates rejected but all destinations. If no-forwarding is enabled frames will not be forwarded between Ports A & B. If tail-tagging is enabled each frame going to Port C needs to have a tail-tag, each frame coming from Port C will have a tail-tag containing the source port and if the frame was redundancy tagged or not. If PRP untagging is enabled, PRP tagged frames will be untagged, this shall only be enabled if it can be guaranteed that all frames in the redundant network contain a PRP tag, otherwise the frame might get corrupted if the payload matches the PRP trailer pattern which will then be removed (frame shortened by 6 bytes which was actual payload)

RED HsrPrpConfigMode Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-											CUT_THROUGH	PRP_UNTAGGING	TAIL_TAGGING	NO_FORWARD	PROMISCUOUS	REDBOX_ID				-	NET_ID		-					MODE		
RO											RW	RW	RW	RW	RW	RW				RO	RW		RO					RW		
Reset: 0x00000000																															
Offset: 0x0084																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:21	RO
CUT_THROUIGH	Enable cut through frame processing	Bit:20	RW
PRP_UNTAGGING	Enables untagging for PRP tagged frames, only set if sure that the incoming frames always have a PRP tag (otherwise payload might be intrepreted as PRP tag which will end in a corrupt frame (6bytes cut off)	Bit:19	RW
TAIL_TAGGING	Enables Tail Tagging, all Frames coming to Port C shall be tail tagged, all Frames comming from Port C will contain a tail tag	Bit:18	RW
NO_FORWARD	Disables forwarding between Port A&B	Bit:17	RW
PROMISCUOUS	Promiscuous Mode	Bit:16	RW
REDBOX_ID	RedBox identifier, used for HSR-PRP mode (0xA or 0xB)	Bit:15:12	RW
-	Reserved, read 0	Bit:11	RO
NET_ID	Network identifier, used for HSR-PRP mode	Bit:10:8	RW
-	Reserved, read 0	Bit:7:3	RO
MODE	Profile (0 Non, 1 PRP, 2 HSR, 5 HSR-PRP, 6 HSR-HSR)	Bit:2:0	RW

3.2.1.11 RED HsrPrp Config Vlan Register

VLAN for 802.3q priority tagging or virtual networks. VLAN can be enabled or disabled.

RED HsrPrpConfigVlan Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-															VLAN_EN	VLAN															
RO															RW	RW															
Reset: 0x00000000																															
Offset: 0x0088																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:17	RO
VLAN_EN	VLAN enable (0 disabled, 1 enabled)	Bit: 16	RW
VLAN	VLAN	Bit: 15:0	RW

3.2.2 Mac

3.2.2.1 RED HsrPrp Mac Control Register

Used for general control over the HSR&PRP Core, all configurations on the core shall only be done when disabled.

RED HsrPrpMacControl Reg																															
Reg Description																															
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
																															MAC_VAL
RO																															RW
Reset: 0x00000000																															
Offset: 0x0100																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:1	RO
MAC_VAL	MAC address valid (selfclearing)	Bit: 0	RW

3.2.2.2 RED HsrPrp MAC 1 Register

MAC address of the node. LSB is transferred first on the network.
E.g. 0x01234567 => MAC: 67:45:32:01:XX:XX.

RED HsrPrpMac1 Reg																													
Reg Description																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MAC(3)								MAC(2)								MAC(1)								MAC(0)							
RW								RW								RW								RW							
Reset: 0x00000000																															
Offset: 0x0104																															

Name	Description	Bits	Access
MAC(3)	MAC Byte 3	Bit:31:24	RW
MAC(2)	MAC Byte 2	Bit:23:16	RW
MAC(1)	MAC Byte 1	Bit:15:8	RW
MAC(0)	MAC Byte 0	Bit:7:0	RW

3.2.2.3 RED HsrPrp MAC 2 Register

MAC address of the node. LSB is transferred first on the network.

E.g. 0x0004567 => MAC: XX:XX:XX:67:45.

RED HsrPrpMac2 Reg																													
Reg Description																													
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
-																MAC(5)								MAC(4)							
RO																RW								RW							
Reset: 0x00000000																															
Offset: 0x0108																															

Name	Description	Bits	Access
-	Reserved, read 0	Bit:31:16	RO
MAC(5)	MAC Byte 5	Bit:15:8	RW
MAC(4)	MAC Byte 4	Bit:7:0	RW

4 Design Description

The following chapters describe the internals of the RED HsrPrp Core: starting with the Top Level, which is a collection of subcores, followed by the description of all subcores.

4.1 Top Level – RED HsrPrp

4.1.1.1 Parameters

The core must be parametrized at synthesis time. There are a couple of parameters which define the final behavior and resource usage of the core.

Name	Type	Size	Description
NoSupport_Gen	boolean	1	Support for NO (ust forward no dropping, but duplication)
HsrSupport_Gen	boolean	1	Support for HSR
PrpSupport_Gen	boolean	1	Support for PRP
HsrPrpSupport_Gen	boolean	1	Support for HSR-PRP RedBox
HsrHsrSupport_Gen	boolean	1	Support for HSR-HSR RedBox
CutThrough_Gen	boolean	1	Support for Cut Through frame handling
ProxyNodeTable Support_Gen	boolean	1	Whether the core shall be run as a RedBox
VlanSupport_Gen	boolean	1	Support for VLAN tagging and untagging
TailTagging_Gen	boolean	1	Support for Tail tagging and untagging to define to send to specific ports and to know from which port it was received
PtpSupport_Gen	boolean	1	Support for PTP
NrOfProxyNodes_Gen	natural	1	How many nodes it supports on the non-redundant network when run as a RedBox
NrOfEntries_Gen	natural	1	How many duplication entries shall be stored, if this number is lowered, chances for non-

			discarded duplicates is higher
HsrModeX Support_Gen	boolean	1	If run in HSR mode if it shall support Mode X (duplicate discarding on the ring)
SeqNumber-OfPorts_Gen	natural	1	How many external cores can request a sequence number from the core. The sequence number is always from the Sequence number pool of the local MAC
LinkSpeed Support_Gen	natural	1	Shall be either 100 or 1000. For 1000 duplication is paral-lelized to achieve the required throughput.
PortStatus Support_Gen	boolean	1	If frame and error counters shall be available in registers
ClockClkPeriod Nanosecond_Gen	natural	1	Clock Period in Nanosecond: Default for 50 MHz = 20 ns
StaticConfig_Gen	boolean	1	If Static Configuration or AXI is used
PortAIoFf_Gen	boolean	1	If Port A shall contain a IO Flip Flop.
PortBIoFf_Gen	boolean	1	If Port B shall contain a IO Flip Flop.
PortCIoFf_Gen	boolean	1	If Port C shall contain a IO Flip Flop.
AxiAddressRange Low_Gen	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	std_logic_vector	32	AXI Base Address plus Regis-terset Size Default plus 0xFFFF
Sim_Gen	boolean	1	If in Testbench simulation mode: true = Simulation, false = Synthesis

Table 5: Parameters

4.1.1.2 Structured Types

4.1.1.2.1 Red_SeqReq_Type

Defined in Red_Package.vhd of library RedLib
This is the type used for sequence number requesting by thirdparty cores.

Field Name	Type	Size	Description
SrcId	std_logic_vector	4	Source Id of the port requesting a sequence number, for external requesters alsways 0x0

Table 6: Red_SeqReq_Type

4.1.1.2.2 Red_SeqReqVal_Type

Defined in Red_Package.vhd of library RedLib
This is the type used for valid falgs of sequence number requesting by thirdparty cores.

Field Name	Type	Size	Description
Req_Val	std_logic	1	Request valid

Table 7: Red_SeqReqVal_Type

4.1.1.2.3 Red_SeqResp_Type

Defined in Red_Package.vhd of library RedLib
This is the type used for sequence number requesting by thirdparty cores.

Field Name	Type	Size	Description
SeqNr	std_logic_vector	16	The sequence number to be used by the thirdparty cores

Table 8: Red_SeqReq_Type

4.1.1.2.4 Red_SeqRespVal_Type

Defined in Red_Package.vhd of library RedLib

This is the type used for valid falgs of sequence number requesting by thirdparty cores.

Field Name	Type	Size	Description
Resp_Val	std_logic	1	Response valid

Table 9: Red_SeqReqVal_Type

4.1.1.2.5 Red_HsrPrpStaticConfig_Type

Defined in Red_HsrPrpAddrPackage.vhd of library RedLib
This is the type used for static configuration.

Field Name	Type	Size	Description
OwnMac	Common_ Byte_Type	6	The MAC of the node. Used for supervision frames and dis-carding
RedMode	Red_Mode_Type	1	Redundancy Mode: Hsr_E Prp_E HsrPrp_E HsrHsr_E No_E
RedBoxNetId	std_logic_vector	3	RedBox Network Identifier for HSR-PRP mode
RedBoxId	std_logic_vector	4	RedBox Identifier for HSR-PRP mode
PromiscuousMode	std_logic	1	If Promiscuous mode shall be active for Port C
NoForward	std_logic	1	If forwarding between Ports A & B shall be disabled
TailTagging	std_logic	1	If tail tagging shall be used for frames to and from Port C
PrpUntagging	std_logic	1	If frames from Port A and B shall be untagged when for-warded to Port C
CutThough	std_logic	1	If frames shall be processed in cut through mode

Vlan	Red_Vlan_Type	1	The Pcp,Dei and Vid of the VLAN
VlanEnable	std_logic	1	If VLAN shall be used

Table 10: Red_HsrPrpStaticConfig_Type

4.1.1.2.6 Red_HsrPrpStaticConfigVal_Type

Defined in Red_HsrPrpAddrPackage.vhd of library RedLib
This is the type used for valid flags of the static configuration.

Field Name	Type	Size	Description
Vlan_Val	std_logic	1	If the VLAN shall be set
Enable_Val	std_logic	1	Enables the RED HsrPrp

Table 11: Red_HsrPrpStaticConfigVal_Type

4.1.1.2.7 Red_HsrPrpStaticStatus_Type

Defined in Red_HsrPrpAddrPackage.vhd of library RedLib
This is the type used for static status supervision.

Field Name	Type	Size	Description
Enabled	std_logic	1	If the core is enabled
RedMode	Red_Mode_Type	1	Redundancy Mode: Hsr_E Prp_E HsrPrp_E HsrHsr_E No_E
RedBoxNetId	std_logic_vector	3	RedBox Network Identifier for HSR-PRP mode
RedBoxId	std_logic_vector	4	RedBox Identifier for HSR-PRP mode
SupervisionTimeout PortA	std_logic	1	No supervision frames received on Port A
SupervisionTimeout PortA	std_logic	1	No supervision frames received on Port B

Table 12: Red_HsrPrpStaticStatus_Type

4.1.1.2.8 Red_HsrPrpStaticStatusVal_Type

Defined in Red_HsrPrpAddrPackage.vhd of library RedLib
This is the type used for valid flags of the static status supervision.

Field Name	Type	Size	Description
Dummy_Val	std_logic	1	Dummy flag (unused)

Table 13: Red_HsrPrpStaticStatusVal_Type

4.1.1.3 Entity Block Diagram

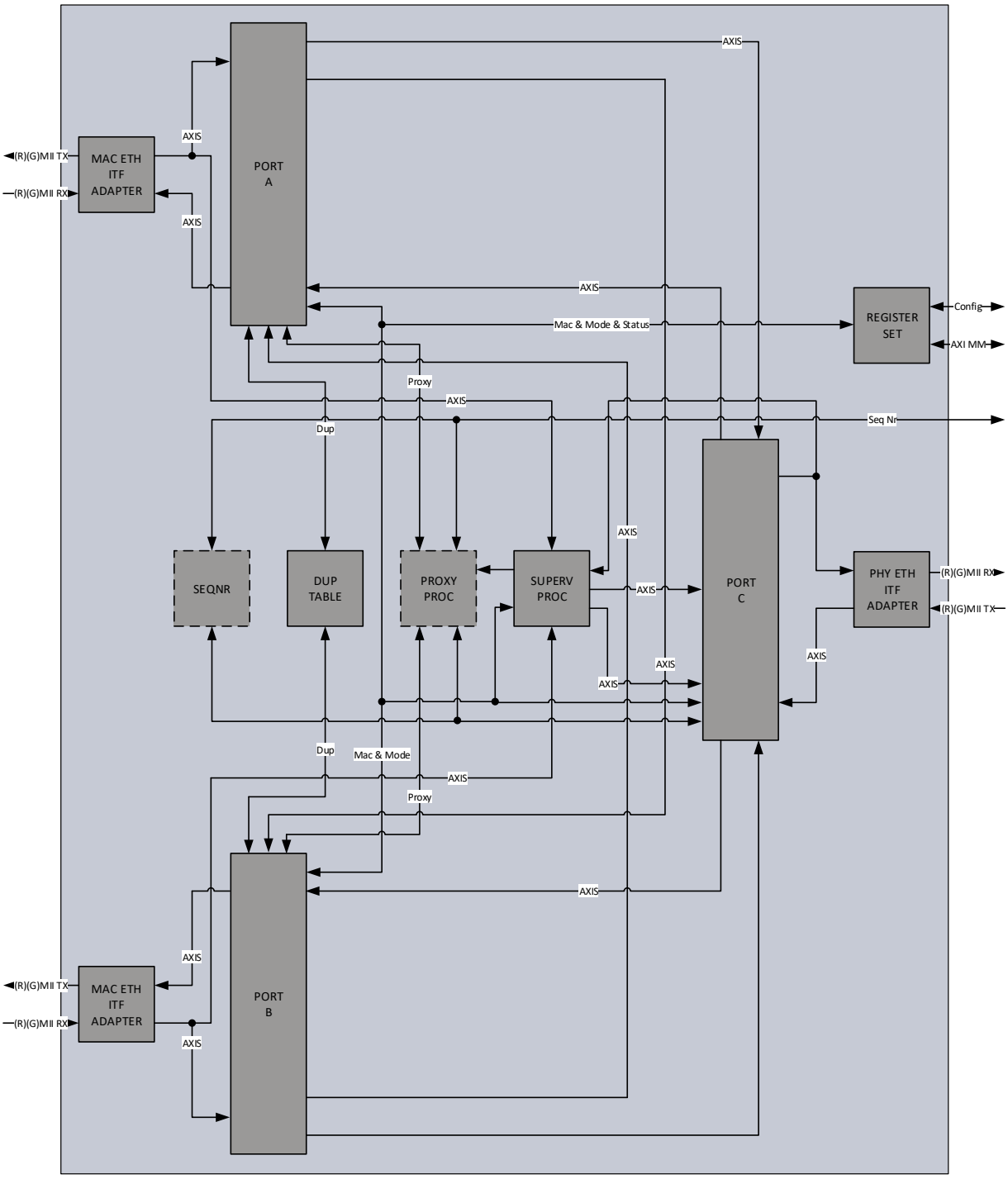

Figure 8: RED HsrPrp Core

4.1.1.4 Entity Description

Port A & B

This module arbitrates between the forwarding Ports A&B and the uplink Port C in the direction to the PHY. It buffers and extracts frame information and takes the

forwarding decision based on information from the Duplicate Processor and Proxy Node Processor. It also untags the frames if required. Only valid frames and no duplicates frames are forwarded to either Port A&B or Port C.
See 4.2.1 for more details.

Port C

This module arbitrates between the Ports A&B in the uplink direction towards the MAC. It also arbitrates between the Supervision Processor and the MAC towards Ports A&B. It buffers and extracts frame information and tags the frames based on information from the ProxyNodeTabel or Sequence Number generator depending on the mode (DAN or RedBox). It then buffers the frame in two individual FIFOs before forwarding them to the Ports A&B.
See 4.2.2 for more details.

Proxy Node Processor

This module learns what nodes are connected on Port C. It is used to generate Sequence Numbers on a per node base. It is also used for the forwarding decision when run in HSR mode. This module is only present if the Core is running as a RedBox otherwise the Sequence Number Processor is taking care of this.
See 4.2.3 for more details.

Duplicate Processor

This module checks if a duplicate is received. It contains a pooled hash table for the duplicate detection.
See 4.2.4 for more details.

Supervision Processor

This module sends Supervision frames based on the nodes stored in the Proxy Node Table and the local MAC. It also supervises the reception of Supervision frames, to detect if a link is broken. In case of failure it will signal Supervision frame timeouts on the specific Ports.
See 4.2.5 for more details.

Sequence Number Processor

This module generates Sequence Numbers which are used by the tagger or third-party cores which require a Sequence Number because they send their own tagged frames. This module is only present if the Proxy Node Processor is not present which is the case when the Core is running as a DAN.

See 4.2.6 for more details.

MAC & PHY Ethernet Interface Adapter

This module converts the Media Independent Interface (MII) to AXI stream and vice versa. It is also in charge of generating correct Interframe Gaps and a Preamble with SFD.
See 4.2.7 for more details.

Registerset

This module is an AXI Light Memory Mapped Slave. It provides access to all Registers and allows to configure the HSR&PRP Core. It can be configured to either run in AXI or StaticConfig mode. If in StaticConfig mode, the configuration of the Registers is done via signals and can be easily done from within the FPGA without CPU. If in AXI mode, a AXI Master has to configure the Datasets with AXI writes to the registers, which is typically done by a CPU
See 4.2.8 for more details.

4.1.1.5 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
NoSupport_Gen	-	boolean	1	Support for NO (ust forward no dropping, but duplication)
HsrSupport_Gen	-	boolean	1	Support for HSR
PrpSupport_Gen	-	boolean	1	Support for PRP
CutThrough_Gen	-	boolean	1	Support for Cut Through frame handling
ProxyNodeTable Support_Gen	-	boolean	1	Whether the core shall be run as a RedBox
VlanSupport_Gen	-	boolean	1	Support for VLAN tagging and untagging

TailTagging_Gen	-	boolean	1	Support for Tail tagging and untagging to define to send to specific ports and to know from which port it was received
PtpSupport_Gen	-	boolean	1	Support for PTP
NrOfProxyNodes_Gen	-	natural	1	How many nodes it supports on the non-redundant network when run as a RedBox (shall be a power of 2)
NrOfEntries_Gen	-	natural	1	How many duplication entries shall be stored, if this number is lowered, chances for non-discarded duplicates is higher (shall be a power of 2)
HsrModeX Support_Gen	-	boolean	1	If run in HSR mode if it shall support Mode X (duplicate discarding on the ring)
SeqNumber OfPorts_Gen	-	natural	1	How many external cores can request a sequence number from the core. The sequence number is always from the Sequence number pool of the local MAC
LinkSpeed Support_Gen	-	natural	1	Shall be either 100 or 1000. For 1000

				duplication is parallelized to achieve the required throughput.
ClockClkPeriodNanosecond_Gen	-	natural	1	Integer Clock Period
PortStatus Support_Gen	-	boolean	1	If frames and error counters shall be available in the registerset
StaticConfig_Gen	-	boolean	1	If Static Configuration or AXI is used
PortAIoFf_Gen	-	boolean	1	If Port A shall contain a IO Flip Flop.
PortBIoFf_Gen	-	boolean	1	If Port B shall contain a IO Flip Flop.
PortCIoFf_Gen	-	boolean	1	If Port C shall contain a IO Flip Flop.
AxiAddressRange Low_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset Size Default plus 0xFFFF
Sim_Gen	-	boolean	1	If in Testbench simulation mode: true = Simulation, false = Synthesis
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Config				
StaticConfig_DatIn	in	Red_HsrPrp StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Red_HsrPrp StaticConfigVal_Type	1	Static Configuration valid

Status				
StaticStatus_DatOut	out	Red_HsrPrp StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Red_HsrPrp StaticStatusVal _Type	1	Static Status valid
Timer				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the Clock
SeqNr Info Input				
SeqReq_DatIn	in	Red_SeqReq Array_Type	SeqNum-berOfPort s_Gen	Sequence Number Request
SeqReq_ValIn	in	Red_SeqReqVal Array_Type	SeqNum-berOfPort s_Gen	Sequence Number Request valid
SeqNr Info Output				
SeqResp_DatOut	out	Red_SeqResp Array_Type	SeqNum-berOfPort s_Gen	Sequence Number Response
SeqResp_ValOut	out	Red_SeqRespVal Array_Type	SeqNum-berOfPort s_Gen	Sequence Number Response valid
Port(A/B) Link Input				
Port(A/B) Link_DatIn	in	std_logic		Link state
Port(A/B)(R)(G)Mii RX Clk/Rst Input				
Port(A/B) (R)(G)MiiRxClk_ClkIn	in	std_logic	1	RX Clock
Port(A/B) (R)(G)MiiRxRstN_Rst In	in	std_logic	1	Reset aligned with RX Clock
Port(A/B)(R)(G)Mii TX Clk/Rst Input				
Port(A/B) (R)(G)MiiTxClk_ClkIn	in	std_logic	1	TX Clock
Port(A/B) (R)(G)MiiTxRstN_Rst In	in	std_logic	1	Reset aligned with TX Clock
Port(A/B)(R)(G)Mii RX Data Input				
Port(A/B) (R)(G)MiiRxDv_EnaIn	in	std_logic	1	RX Data valid
Port(A/B) (R)(G)MiiRxErr_EnaIn	in	std_logic	1	RX Error
Port(A/B) (R)(G)MiiRxData_Dat In	in	std_logic_vector	2-8	RX Data MII:4, RMII:2, GMII:8, RGMII:4
Port(A/B) (R)(G)MiiCol_DatIn	in	std_logic	1	Collision
Port(A/B) (R)(G)MiiCrs_DatIn	in	std_logic	1	Carrier Sense

Port(A/B)(R)(G)Mii TX Data Output				
Port(A/B)(R)(G)MiiTxEn_EnaOut	out	std_logic	1	TX Data valid
Port(A/B)(R)(G)MiiTxErr_EnaOut	out	std_logic	1	TX Error
Port(A/B)(R)(G)MiiTxData_DatOut	out	std_logic_vector	2-8	TX Data MII:4, RMII:2, GMII:8, RGMII:4
PortC Link Input				
PortC Link_DatIn	in	std_logic		Link state
PortC(R)(G)Mii RX Clk/Rst Output				
PortC(R)(G)MiiRxClk_ClkOut	out	std_logic	1	RX Clock
PortC(R)(G)MiiRxRstN_RstOut	out	std_logic	1	Reset aligned with RX Clock
PortC(R)(G)Mii TX Clk/Rst Output				
PortC(R)(G)MiiTxClk_ClkOut	out	std_logic	1	TX Clock
PortC(R)(G)MiiTxRstN_RstOut	out	std_logic	1	Reset aligned with TX Clock
PortC(R)(G)Mii RX Data Output				
PortC(R)(G)MiiRxDv_EnaOut	out	std_logic	1	RX Data valid
PortC(R)(G)MiiRxErr_EnaOut	out	std_logic	1	RX Error
PortC(R)(G)MiiRxData_DatOut	out	std_logic_vector	2-8	RX Data MII:4, RMII:2, GMII:8, RGMII:4
PortC(R)(G)MiiCol_DatOut	out	std_logic	1	Collision
PortC(R)(G)MiiCrs_DatOut	out	std_logic	1	Carrier Sense
PortC(R)(G)Mii TX Data Input				
PortC(R)(G)MiiTxEn_EnaIn	in	std_logic	1	TX Data valid
PortC(R)(G)MiiTxErr_EnaIn	in	std_logic	1	TX Error
PortC(R)(G)MiiTxData_DatIn	in	std_logic_vector	2-8	TX Data MII:4, RMII:2, GMII:8, RGMII:4
AXI4 Light Slave				

AxiWriteAddrValid_ValIn	in	std_logic	1	Write Address Valid
AxiWriteAddrReady_RdyOut	out	std_logic	1	Write Address Ready
AxiWriteAddrAddress_AdrIn	in	std_logic_vector	32	Write Address
AxiWriteAddrProt_DatIn	in	std_logic_vector	3	Write Address Protocol
AxiWriteDataValid_ValIn	in	std_logic	1	Write Data Valid
AxiWriteDataReady_RdyOut	out	std_logic	1	Write Data Ready
AxiWriteDataData_DatIn	in	std_logic_vector	32	Write Data
AxiWriteDataStrobe_DatIn	in	std_logic_vector	4	Write Data Strobe
AxiWriteRespValid_ValOut	out	std_logic	1	Write Response Valid
AxiWriteRespReady_RdyIn	in	std_logic	1	Write Response Ready
AxiWriteResp Response_DatOut	out	std_logic_vector	2	Write Response
AxiReadAddrValid_ValIn	in	std_logic	1	Read Address Valid
AxiReadAddrReady_RdyOut	out	std_logic	1	Read Address Ready
AxiReadAddrAddress_AdrIn	in	std_logic_vector	32	Read Address
AxiReadAddrProt_DatIn	in	std_logic_vector	3	Read Address Protocol
AxiReadDataValid_ValOut	out	std_logic	1	Read Data Valid
AxiReadDataReady_RdyIn	in	std_logic	1	Read Data Ready
AxiReadData Response_DatOut	out	std_logic_vector	2	Read Data
AxiReadDataData_DatOut	out	std_logic_vector	32	Read Data Response

Table 14: RED HsrPrp Core

4.2 Design Parts

The RED HsrPrp Core core consists of a couple of subcores. Each of the subcores itself consist again of smaller function block. The following chapters describe these subcores and their functionality. The Core is used for TSN as well only the relevant interfaces for HSR&PRP are described, this means there are more generics and signals on the entities which are not relevant for HSR&PRP.

4.2.1 Port A&B

4.2.1.1 Entity Block Diagram

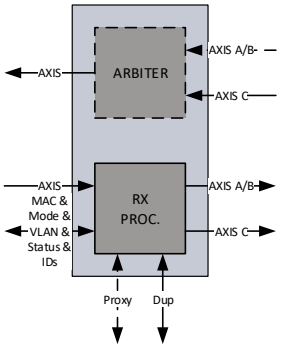

Figure 9: Port A&B

4.2.1.2 Entity Description

Arbiter

This module merges the AXIS forwarding path from the Ports A&B (Port B when Port A and vice versa) with the AXIS path from Port C. The merged stream is then forwarded to the interface adapter.

RX Processor

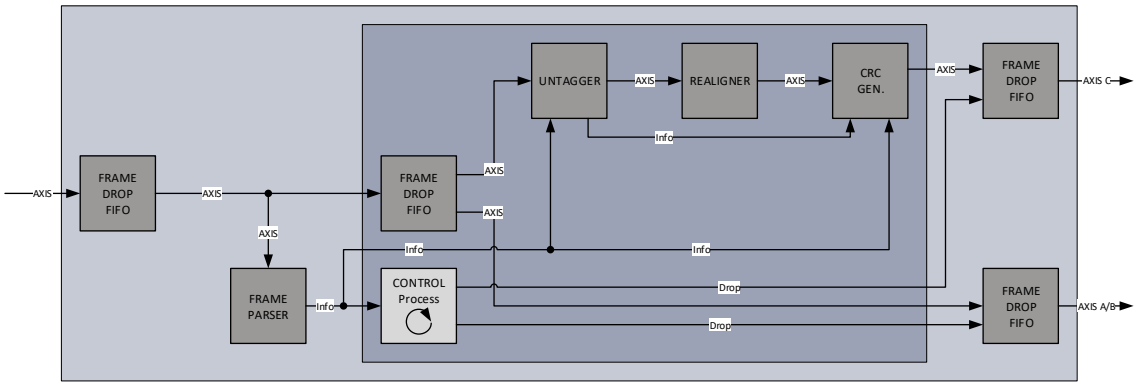

Figure 10: Rx Processor

This module is one of the core modules of the design. It parses incoming frames, does the duplicate discarding, untagging (retagging), CRC recalculation and split into paths to Ports A&B or Port C. To do so it requests from the Proxy Node Processor if the frame received is from/for a Proxy Node and drops it accordingly on the forwarding path of Ports A&B. For duplicate discarding it requests from the Duplicate Processor if the frame is a duplicate or not. Depending on the mode and result a frame is then forwarded to the Ports A&B and/or Port C.
If VLAN is enabled it also checks the VLAN and only forwards matching VID (and VID 0) frames to Port C, the VLAN tag is then removed. If there are multiple VLAN tags only the outer VLAN is removed.
If Tail Tagging is supported and enabled, it will add the Tail Tag to the frame before the FCS to indicate from which port the frame came and if it was VLAN and/or redundancy tagged.

4.2.1.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
NoSupport_Gen	-	boolean	1	Support for NO (ust forward no dropping, but duplication)
HsrSupport_Gen	-	boolean	1	Support for HSR
PrpSupport_Gen	-	boolean	1	Support for PRP
HsrPrpSupport_Gen	-	boolean	1	Support for HSR-PRP RedBox
HsrHsrSupport_Gen	-	boolean	1	Support for HSR-HSR RedBox
CutThrough_Gen	-	boolean	1	Support for Cut Through frame handling
VlanSupport_Gen	-	boolean	1	Support for VLAN
TailTagging_Gen	-	boolean	1	Support for Tail tagging and untagging to define to send to specific

				ports and to know from which port it was received
Sim_Gen	-	boolean	1	If in Testbench simulation mode: true = Simulation, false = Synthesis
Ports				
ProxyNodeTable Support_Gen	-	boolean	1	Whether the core shall be run as a RedBox
PtpSupport_Gen	-	boolean	1	Support for PTP
HsrModeXSup-port_Gen	-	boolean	1	If run in HSR mode if it shall support Mode X (duplicate discarding on the ring)
LinkSpeed Support_Gen	-	natural	1	Shall be either 100 or 1000. For 1000 duplication is paral-lelized to achieve the required throughput.
PortStatus Support_Gen	-	boolean	1	If frames and error counters shall be available in the registerset
SrcId_Gen	-	std_logic_vector	4	Port identifier: 0xA for Port A 0xB for Port B
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Own MAC Input				
OwnMac_DatIn	in	Common_Byte_Type	6	MAC address of the node
VLAN Input				
Vlan_DatIn	in	Red_Vlan_Type	6	VLAN

VlanEnable_DatIn	in	std_logic	1	VLAN mode enabled
Mode Input				
RedMode_DatIn	in	Red_Mode_Type	1	Redundancy Mode: Hsr_E Prp_E HsrPrp_E HsrHsr_E No_E
RedBoxNetId_DatIn	in	std_logic_vector	3	RedBox Network Identifier for HSR-PRP mode
RedBoxId_DatIn	in	std_logic_vector	4	RedBox Identifier for HSR-PRP mode
Promiscuous-Mode_DatIn	in	std_logic	1	If in Promiscuous mode
CutThrough_ValIn	in	std_logic	1	If cut through shall be enabled
NoForward_DatIn	in	std_logic	1	If forwarding between Ports A & B shall be disabled
TailTagging_DatIn	in	std_logic	1	If Tail Tagging shall be done
PrpUntagging_DatIn	in	std_logic	1	If PRP frames shall be untagged
Link Input				
Link_DatIn	in	std_logic	1	Link state of the Port
LinkSpeed_DatIn	in	Common_LinkSpeed_Type	6	Link spped of the node
Proxy Info Output				
ProxyReq_DatOut	out	Red_ProxyReq_Type	1	Proxy Table Processor request
ProxyReq_ValOut	out	Red_ProxyReqVal_Type	1	Proxy Table Processor request valid
Proxy Info Input				
ProxyResp_DatIn	in	Red_ProxyResp_Type	1	Proxy Table Processor response

ProxyResp_ValIn	in	Red_ProxyRespVal_ Type	1	Proxy Table Proces- sor response valid
Duplicate Info Output				
DupReq_DatOut	out	Red_DupReq_Type	1	Duplicate Processor request
DupReq_ValOut	out	Red_DupReqVal_ Type	1	Duplicate Processor request valid
Duplicate Info Input				
DupResp_DatIn	in	Red_DupResp_ Type	1	Duplicate Processor response
DupResp_ValIn	in	Red_DupRespVal_ Type	1	Duplicate Processor response valid
Port Status Output				
PortStatus_DatOut	out	Red_Port Status_Type	1	Port Status
Axi Input				
AxisRxValid_ValIn	in	std_logic	1	AXI Stream frame input
AxisRxReady_ValOut	out	std_logic	1	
AxisRxData_DatIn	in	std_logic_vector	32	
AxisRxStrobe_ValIn	in	std_logic_vector	4	
AxisRxKeep_ValIn	in	std_logic_vector	4	
AxisRxLast_ValIn	in	std_logic	1	
AxisRxUser_DatIn	in	std_logic_vector	3	
Axi Output				
AxisTxValid_ValOut	out	std_logic	1	AXI Stream frame output
AxisTxReady_ValIn	in	std_logic	1	
AxisTxData_DatOut	out	std_logic_vector	32	
AxisTxStrobe_ValOut	out	std_logic_vector	4	
AxisTxKeep_ValOut	out	std_logic_vector	4	
AxisTxLast_ValOut	out	std_logic	1	
AxisTxUser_DatOut	out	std_logic_vector	3	
Axi TX Port A/B Input				
AxisTxPortAB Valid_ValIn	in	std_logic	1	AXI Stream frame input
AxisTxPortAB Ready_ValOut	out	std_logic	1	
AxisTxPortAB Data_DatIn	in	std_logic_vector	32	
AxisTxPortAB Strobe_ValIn	in	std_logic_vector	4	
AxisTxPortAB Keep_ValIn	in	std_logic_vector	4	

AxisTxPortAB Last_ValIn	in	std_logic	1	
AxisTxPortAB User_DatIn	in	std_logic_vector	3	
Axi TX Port C Input				
AxisTxPortC Valid_ValIn	in	std_logic	1	AXI Stream frame input
AxisTxPortC Ready_ValOut	out	std_logic	1	
AxisTxPortC Data_DatIn	in	std_logic_vector	32	
AxisTxPortC Strobe_ValIn	in	std_logic_vector	4	
AxisTxPortC Keep_ValIn	in	std_logic_vector	4	
AxisTxPortC Last_ValIn	in	std_logic	1	
AxisTxPortC User_DatIn	in	std_logic_vector	3	
Axi RX Port A/B Output				
AxisRxPortAB Valid_ValOut	out	std_logic	1	AXI Stream frame output
AxisRxPortAB Ready_ValIn	in	std_logic	1	
AxisRxPortAB Data_DatOut	out	std_logic_vector	32	
AxisRxPortA BStrobe_ValOut	out	std_logic_vector	4	
AxisRxPortAB Keep_ValOut	out	std_logic_vector	4	
AxisRxPortAB Last_ValOut	out	std_logic	1	
AxisRxPortAB User_DatOut	out	std_logic_vector	3	
Axi RX Port C Output				
AxisRxPortC Valid_ValOut	out	std_logic	1	AXI Stream frame output
AxisRxPortC Ready_ValIn	in	std_logic	1	
AxisRxPortC Data_DatOut	out	std_logic_vector	32	
AxisRxPortC BStrobe_ValOut	out	std_logic_vector	4	
AxisRxPortC Keep_ValOut	out	std_logic_vector	4	
AxisRxPortC Last_ValOut	out	std_logic	1	
AxisRxPortC User_DatOut	out	std_logic_vector	3	
Enable Input				
Enable_EnaIn	in	std_logic	1	Enable core

Table 15: Port A&B

4.2.2 Port C

4.2.2.1 Entity Block Diagram

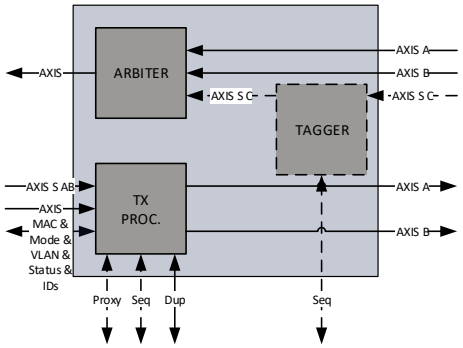

Figure 11: Port C

4.2.2.2 Entity Description

Arbiter

This module merges the AXIS paths from the Ports A&B. The merged stream is then forwarded to the interface adapter.

Tagger

This module tags supervision frames towards Port C in case of a HSR-PRP RedBox.

TX Processor

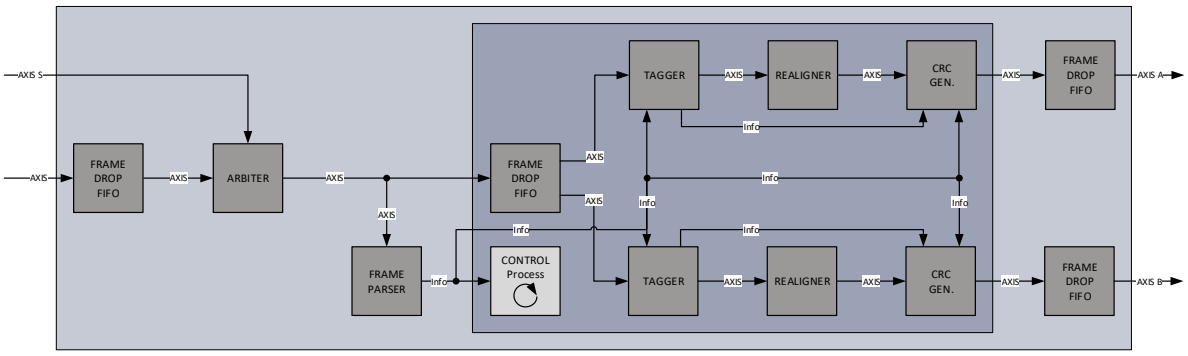

Figure 12: Tx Processor

This module is one of the core modules of the design. It parses incoming frames, does the duplication, tagging, CRC recalculation and split into paths to Ports A&B. To do so it requests from the Proxy Node Processor if in RedBox mode, or from the Sequence Number generator if in DAN mode, a sequence number for tagging. Then

the two tagged frames are forwarded to the two individual Ports. It also merges the path from the interface adapter with the path from the Supervision Processor. If VLAN is enabled, the frame is tagged if not already VLAN tagged; if already tagged the VLAN tag is forwarded untouched. Also, it merges the Supervision frames into the stream
If Tail Tagging is enabled, it extracts the information from the Tail Tag and forwards the frames only to the Ports indicated in the Tail Tag. If the Tail Tag states to not redundancy tag the frame or to not VLAN tag the frame it will skip this as well. This requires that each frame coming to this module is Tail Tagged, otherwise the last byte before the FCS is interpreted wrongly as Tail Tag.

4.2.2.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
NoSupport_Gen	-	boolean	1	Support for NO (ust forward no dropping, but duplication)
HsrSupport_Gen	-	boolean	1	Support for HSR
PrpSupport_Gen	-	boolean	1	Support for PRP
HsrPrpSupport_Gen	-	boolean	1	Support for HSR-PRP RedBox
HsrHsrSupport_Gen	-	boolean	1	Support for HSR-HSR RedBox
CutThrough_Gen	-	boolean	1	Support for Cut Through frame handling
VlanSupport_Gen	-	boolean	1	Support for VLAN
TailTagging_Gen	-	boolean	1	Support for Tail tagging and untagging to define to send to specific ports and to know from which port it was received

Sim_Gen	-	boolean	1	If in Testbench simulation mode: true = Simulation, false = Synthesis
Ports				
ProxyNodeTable Support_Gen	-	boolean	1	Whether the core shall be run as a RedBox
PtpSupport_Gen	-	boolean	1	Support for PTP
HsrModeXSup-port_Gen	-	boolean	1	If run in HSR mode if it shall support Mode X (duplicate discarding on the ring)
LinkSpeed Support_Gen	-	natural	1	Shall be either 100 or 1000. For 1000 duplication is paral-lelized to achieve the required throughput.
PortStatus Support_Gen	-	boolean	1	If frames and error counters shall be available in the registerset
SrcId_Gen	-	std_logic_vector	4	Port identifier: 0xA for Port A 0xB for Port B
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Own MAC Input				
OwnMac_DatIn	in	Common_Byte_Type	6	MAC address of the node
VLAN Input				
Vlan_DatIn	in	Red_Vlan_Type	6	VLAN
VlanEnable_DatIn	in	std_logic	1	VLAN mode ena-bled
Mode Input				

RedMode_DatIn	in	Red_Mode_Type	1	Redundancy Mode: Hsr_E Prp_E HsrPrp_E HsrHsr_E No_E
RedBoxNetId_DatIn	in	std_logic_vector	3	RedBox Network Identifier for HSR-PRP mode
RedBoxId_DatIn	in	std_logic_vector	4	RedBox Identifier for HSR-PRP mode
CutThrough_ValIn	in	std_logic	1	If cut through shall be enabled
TailTagging_DatIn	in	std_logic	1	If Tail Tagging shall be done
Link Input				
Link_DatIn	in	std_logic	1	Link state of the Port
LinkSpeed_DatIn	in	Com-mon_LinkSpeed_Type	6	Link spped of the node
Proxy Info Output				
ProxyReq_DatOut	out	Red_ProxyReq_Type	1	Proxy Table Proces-sor request
ProxyReq_ValOut	out	Red_ProxyReqVal_Type	1	Proxy Table Proces-sor request valid
Proxy Info Input				
ProxyResp_DatIn	in	Red_ProxyResp_Type	1	Proxy Table Proces-sor response
ProxyResp_ValIn	in	Red_ProxyRespVal_Type	1	Proxy Table Proces-sor response valid
Sequence Number Info Output				
SeqReq_DatOut	out	Red_SeqReq Array_Type	2	Sequence Number Processor request
SeqReq_ValOut	out	Red_SeqReqVal Array_Type	2	Sequence Number Processor request valid
Sequence Number Info Input				
SeqResp_DatIn	in	Red_SeqResp	2	Sequence Number

		Array_Type		Processor response
SeqResp_ValIn	in	Red_SeqRespVal Array_Type	2	Sequence Number Processor response valid
Duplicate Info Output				
DupReq_DatOut	out	Red_DupReq_Type	1	Duplicate Processor request
DupReq_ValOut	out	Red_DupReqVal_ Type	1	Duplicate Processor request valid
Duplicate Info Input				
DupResp_DatIn	in	Red_DupResp_ Type	1	Duplicate Processor response
DupResp_ValIn	in	Red_DupRespVal_ Type	1	Duplicate Processor response valid
Port Status Output				
PortStatus_DatOut	out	Red_Port Status_Type	1	Port Status
Axi Input				
AxisTxValid_ValIn	in	std_logic	1	AXI Stream frame input
AxisTxReady_ValOut	out	std_logic	1	
AxisTxData_DatIn	in	std_logic_vector	32	
AxisTxStrobe_ValIn	in	std_logic_vector	4	
AxisTxKeep_ValIn	in	std_logic_vector	4	
AxisTxLast_ValIn	in	std_logic	1	
AxisTxUser_DatIn	in	std_logic_vector	3	
Axi Output				
AxisRxValid_ValOut	out	std_logic	1	AXI Stream frame output
AxisRxReady_ValIn	in	std_logic	1	
AxisRxData_DatOut	out	std_logic_vector	32	
AxisRxStrobe_ValOut	out	std_logic_vector	4	
AxisRxKeep_ValOut	out	std_logic_vector	4	
AxisRxLast_ValOut	out	std_logic	1	
AxisRxUser_DatOut	out	std_logic_vector	3	
Axi RX Port A Input				
AxisRxPortA Valid_ValIn	in	std_logic	1	AXI Stream frame input
AxisRxPortA Ready_ValOut	out	std_logic	1	
AxisRxPortA Data_DatIn	in	std_logic_vector	32	
AxisRxPortA Strobe_ValIn	in	std_logic_vector	4	

AxisRxPortA Keep_ValIn	in	std_logic_vector	4	
AxisRxPortA Last_ValIn	in	std_logic	1	
AxisRxPortA User_DatIn	in	std_logic_vector	3	
Axi RX Port B Input				
AxisRxPortB Valid_ValIn	in	std_logic	1	AXI Stream frame input
AxisRxPortB Ready_ValOut	out	std_logic	1	
AxisRxPortB Data_DatIn	in	std_logic_vector	32	
AxisRxPortB Strobe_ValIn	in	std_logic_vector	4	
AxisRxPortB Keep_ValIn	in	std_logic_vector	4	
AxisRxPortB Last_ValIn	in	std_logic	1	
AxisRxPortB User_DatIn	in	std_logic_vector	3	
Axi TX Port S AB Input				
AxisTxPortSAB Valid_ValIn	in	std_logic	1	AXI Stream frame input
AxisTxPortSAB Ready_ValOut	out	std_logic	1	
AxisTxPortSAB Data_DatIn	in	std_logic_vector	32	
AxisTxPortSAB Strobe_ValIn	in	std_logic_vector	4	
AxisTxPortSAB Keep_ValIn	in	std_logic_vector	4	
AxisTxPortSAB Last_ValIn	in	std_logic	1	
AxisTxPortSAB User_DatIn	in	std_logic_vector	3	
Axi TX Port S C Input				
AxisTxPortSC Valid_ValIn	in	std_logic	1	AXI Stream frame input
AxisTxPortSC Ready_ValOut	out	std_logic	1	
AxisTxPortSC Data_DatIn	in	std_logic_vector	32	
AxisTxPortSC Strobe_ValIn	in	std_logic_vector	4	
AxisTxPortSC Keep_ValIn	in	std_logic_vector	4	
AxisTxPortSC Last_ValIn	in	std_logic	1	
AxisTxPortSC User_DatIn	in	std_logic_vector	3	
Axi TX Port A Output				
AxisTxPortA Valid_ValOut	out	std_logic	1	AXI Stream frame output
AxisTxPortA Ready_ValIn	in	std_logic	1	

AxisTxPortA Data_DatOut	out	std_logic_vector	32	
AxisTxPortA BStrobe_ValOut	out	std_logic_vector	4	
AxisTxPortA Keep_ValOut	out	std_logic_vector	4	
AxisTxPortA Last_ValOut	out	std_logic	1	
AxisTxPortA User_DatOut	out	std_logic_vector	3	
Axi TX Port B Output				
AxisTxPortB Valid_ValOut	out	std_logic	1	AXI Stream frame output
AxisTxPortB Ready_ValIn	in	std_logic	1	
AxisTxPortB Data_DatOut	out	std_logic_vector	32	
AxisTxPortB BStrobe_ValOut	out	std_logic_vector	4	
AxisTxPortB Keep_ValOut	out	std_logic_vector	4	
AxisTxPortB Last_ValOut	out	std_logic	1	
AxisTxPortB User_DatOut	out	std_logic_vector	3	
Enable Input				
Enable_EnaIn	in	std_logic	1	Enable core

Table 16: Port C

4.2.3 Proxy Node Processor

4.2.3.1 Entity Block Diagram

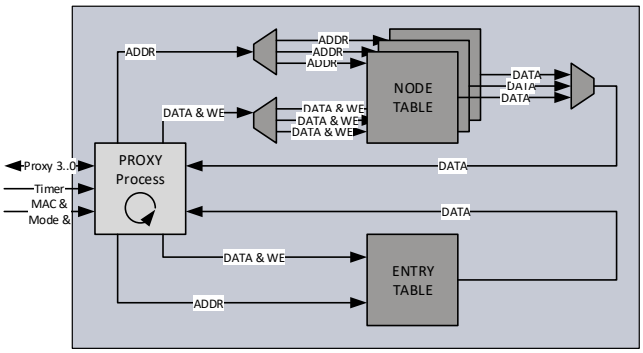

Figure 13: Proxy Node Processor

4.2.3.2 Entity Description

The Proxy Process sequentiallizes incoming request and answers them respectively.
There are three different kind of request:

- Node request: Returns the MAC of a specific address in the table
- SeqNr request: Returns and increments the Sequence Number of a specific node
- Node request: Checks if a specific node is in the table

It also does aging of the nodes so if a node is not there anymore the entry is cleared.

Node Tables

This module is the storage for the nodes connected to Port C. Per entry it contains the MAC address and a timeout value for aging the nodes from the table.
There are multiple instances of the Node Table depending on the link speed and number of nodes supported on port C. The reason for multiple instances is to reduce the search duration in the table. A node is searched with linear search. This is used by the Supervision Processor to figure out for which nodes Supervision frames have to be sent, by Port A&B to decide where to forward a frame and by Port C for tagging frames on a per node base.

Entry Table

Per node there is exactly one entry in the Entry Table which stores the Sequence Number for this node. This is only used by Port C for tagging.

4.2.3.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
NoSupport_Gen	-	boolean	1	Support for NO (ust forward no dropping, but duplication)
HsrSupport_Gen	-	boolean	1	Support for HSR
PrpSupport_Gen	-	boolean	1	Support for PRP
HsrPrpSupport_Gen	-	boolean	1	Support for HSR-PRP RedBox
HsrHsrSupport_Gen	-	boolean	1	Support for HSR-HSR RedBox
ClockClkPeriodNanosecond_Gen	-	natural	1	Integer Clock Period
Sim_Gen	-	boolean	1	If in Testbench simulation mode: true = Simulation, false = Synthesis
Proxy Node table				
NumberOfPorts_Gen	-	natural	1	How many requestors are connected
NrOfProxyNodes_Gen	-	natural	1	How many nodes it supports on the non-redundant network when run as a RedBox
LinkSpeedSupport_Gen	-	natural	1	Either 100 or 1000
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Own MAC Input				

OwnMac_DatIn	in	Common_Byte_Type	6	MAC address of the node
Timer				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the Clock
Mode Output				
RedMode_DatIn	in	Red_Mode_Type	1	Redundancy Mode: Hsr_E Prp_E HsrPrp_E HsrHsr_E No_E
Proxy Info Input				
ProxyReq_DatIn	in	Red_ProxyReqArray_Type	Number OfPorts _Gen	Proxy Table Processor request
ProxyReq_ValIn	in	Red_ProxyReqValArray_Type	Number OfPorts _Gen	Proxy Table Processor request valid
Proxy Info Output				
ProxyResp_DatOut	out	Red_ProxyRespArray_Type	Number OfPorts _Gen	Proxy Table Processor response
ProxyResp_ValOut	out	Red_ProxyRespValArray_Type	Number OfPorts _Gen	Proxy Table Processor response valid
Enable Input				
Enable_EnaIn	in	std_logic	1	Enable core

Table 17: Proxy Node Processor

4.2.4 Duplicate Processor

4.2.4.1 Entity Block Diagram

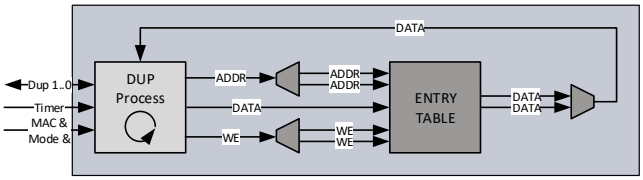

Figure 14: Duplicate Processor

4.2.4.2 Entity Description

The Duplicate Process sequentiallizes incoming request and answers them respectively.
There is only one kind of requests:

- Duplicate request: Returns if an entry is a duplicate or not

It checks if a frame with a specific Sequence Number from a specific node has been seen on the other LAN already and signals that this is a duplicate. There are a couple of other conditions which lead to signaling of a non-duplicate.
It also does aging of the entries so after 400ms when no duplicate is received the entry is cleared.

Entry Table

This is a Hash table with Pools. The Pool size is 16 entries. The Hash is calculated over the Source MAC and Sequence Number. Also, a Hash is stored for only the MAC addresses. Per Entry it stores the Sequence Number, a part of the MAC Hash, a Timeout and flags for this frame. First the Pool is addressed via the Hash and then linear searched within the Pool. If no entry with the same MAC hash and Sequence Number is found a new entry is added. If an entry was found and the frame was already received on the other LAN the entry is cleared. If Sequence Number and MAC Hash match but the flags do not match the entry is overwritten.

4.2.4.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
NoSupport_Gen	-	boolean	1	Support for NO (ust

				forward no dropping, but duplication)
HsrSupport_Gen	-	boolean	1	Support for HSR
PrpSupport_Gen	-	boolean	1	Support for PRP
HsrPrpSupport_Gen	-	boolean	1	Support for HSR-PRP RedBox
HsrHsrSupport_Gen	-	boolean	1	Support for HSR-HSR RedBox
PtpSupport_Gen	-	boolean	1	Support for PTP
ClockClkPeriodNanosecond_Gen	-	natural	1	Integer Clock Period
Sim_Gen	-	boolean	1	If in Testbench simulation mode: true = Simulation, false = Synthesis
Proxy Node table				
NumberOfPorts_Gen	-	natural	1	How many requestors are connected
NrOfEntries_Gen	-	natural	1	How many entries for duplicate detection shall be supported
LinkSpeedSupport_Gen	-	natural	1	Either 100 or 1000
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Own MAC Input				
OwnMac_DatIn	in	Common_Byte_Type	6	MAC address of the node
Timer				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the Clock
Mode Output				
RedMode_DatIn	in	Red_Mode_Type	1	Redundancy Mode: Hsr_E

				Prp_E HsrPrp_E HsrHsr_E No_E
Link Input				
PortALink_DatIn	in	std_logic	1	Link state of Port A
PortALinkSpeed _DatIn	in	Common_LinkSpeed _Type	1	Link speed of Port A
PortBLink_DatIn	in	std_logic	1	Link state of Port B
PortBLinkSpeed _DatIn	in	Common_LinkSpeed _Type	1	Link speed of Port B
Duplicate Info Input				
DupReq_DatIn	in	Red_DupReqArray _Type	Number OfPorts _Gen	Duplicate Processor request
DupReq_ValIn	in	Red_DupReqVal Array_Type	Number OfPorts _Gen	Duplicate Processor request valid
Proxy Info Output				
DupResp_DatOut	out	Red_DupRespArray _Type	Number OfPorts _Gen	Duplicate Processor response
DupResp_ValOut	out	Red_DupRespVal Array_Type	Number OfPorts _Gen	Duplicate Processor response valid
Enable Input				
Enable_EnaIn	in	std_logic	1	Enable core

Table 18: Duplicate Processor

4.2.5 Supervision Processor

4.2.5.1 Entity Block Diagram

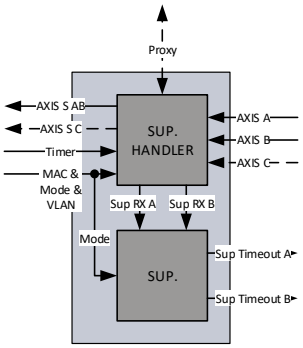

Figure 15: Supervision Processor

4.2.5.2 Entity Description

Supervision Handler

This module receives frames from Ports A&B individually and checks it they are Supervision frames. If a Supervision frame was detected on either of the ports it signals this to the Supervision module. It also generates Supervision frames periodically for the node itself and if run in RedBox mode it sends supervision frames for the nodes in the Proxy Node Table. The frames sent are sent untagged and with cleared FCS since this is added and calculated in the Port C module.

Supervision

This module runs two individual timeout counters (one for Port A and B), which get reset whenever a Supervision frame was received on the respective Port. If the timeout value is reached, it signals this with the respective timeout signal.

4.2.5.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
NoSupport_Gen	-	boolean	1	Support for NO (ust forward no dropping, but duplica-

				tion)
HsrSupport_Gen	-	boolean	1	Support for HSR
PrpSupport_Gen	-	boolean	1	Support for PRP
HsrPrpSupport_Gen	-	boolean	1	Support for HSR-PRP RedBox
HsrHsrSupport_Gen	-	boolean	1	Support for HSR-HSR RedBox
VlanSupport_Gen	-	boolean	1	Support for VLAN
Sim_Gen	-	boolean	1	If in Testbench simulation mode: true = Simulation, false = Synthesis
Supervision Processor				
NrOfProxyNodes_Gen	-	natural	1	How many nodes it supports on the non-redundant network when run as a RedBox
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Own MAC Input				
OwnMac_DatIn	in	Common_Byte_Type	6	MAC address of the node
VLAN Input				
Vlan_DatIn	in	Red_Vlan_Type	6	VLAN
VlanEnable_DatIn	in	std_logic	1	VLAN mode enabled
Mode Output				
RedMode_DatIn	in	Red_Mode_Type	1	Redundancy Mode: Hsr_E Prp_E HsrPrp_E HsrHsr_E No_E
Timer				
Timer1ms_EvtIn	in	std_logic	1	Millisecond timer adjusted with the

				Clock
Proxy Info Output				
ProxyReq_DatOut	out	Red_ProxyReq_Type	1	Proxy Table Processor request
ProxyReq_ValOut	out	Red_ProxyReqVal_Type	1	Proxy Table Processor request valid
Proxy Info Input				
ProxyResp_DatIn	in	Red_ProxyResp_Type	1	Proxy Table Processor response
ProxyResp_ValIn	in	Red_ProxyRespVal_Type	1	Proxy Table Processor response valid
Supervision TimeoutOutput				
SupervisionTimeout PortA_DatOut	out	std_logic	1	Supervision frame timeout on Port A
SupervisionTimeout PortB_DatOut	out	std_logic	1	Supervision frame timeout on Port B
Axi RX Port A Input				
AxisRxPortA Valid_ValIn	in	std_logic	1	AXI Stream frame input
AxisRxPortA Ready_ValIn	in	std_logic	1	
AxisRxPortA Data_DatIn	in	std_logic_vector	32	
AxisRxPortA Strobe_ValIn	in	std_logic_vector	4	
AxisRxPortA Keep_ValIn	in	std_logic_vector	4	
AxisRxPortA Last_ValIn	in	std_logic	1	
AxisRxPortA User_DatIn	in	std_logic_vector	3	
Axi RX Port B Input				
AxisRxPortB Valid_ValIn	in	std_logic	1	AXI Stream frame input
AxisRxPortB Ready_ValIn	in	std_logic	1	
AxisRxPortB Data_DatIn	in	std_logic_vector	32	
AxisRxPortB Strobe_ValIn	in	std_logic_vector	4	
AxisRxPortB Keep_ValIn	in	std_logic_vector	4	
AxisRxPortB Last_ValIn	in	std_logic	1	
AxisRxPortB User_DatIn	in	std_logic_vector	3	
Axi RX Port C Input				
AxisRxPortC Valid_ValIn	in	std_logic	1	AXI Stream frame

AxisRxPortC Ready_ValIn	in	std_logic	1	input
AxisRxPortC Data_DatIn	in	std_logic_vector	32	
AxisRxPortC Strobe_ValIn	in	std_logic_vector	4	
AxisRxPortC Keep_ValIn	in	std_logic_vector	4	
AxisRxPortC Last_ValIn	in	std_logic	1	
AxisRxPortC User_DatIn	in	std_logic_vector	3	
Axi TX Port S AB Output				
AxisTxPortSAB Valid_ValOut	out	std_logic	1	AXI Stream frame output
AxisTxPortSAB Ready_ValIn	in	std_logic	1	
AxisTxPortSAB Data_DatOut	out	std_logic_vector	32	
AxisTxPortS BStrobe_ValOut	out	std_logic_vector	4	
AxisTxPortSAB Keep_ValOut	out	std_logic_vector	4	
AxisTxPortSAB Last_ValOut	out	std_logic	1	
AxisTxPortSAB User_DatOut	out	std_logic_vector	3	
Axi TX Port S C Output				
AxisTxPortSC Valid_ValOut	out	std_logic	1	AXI Stream frame output
AxisTxPortSC Ready_ValIn	in	std_logic	1	
AxisTxPortSC Data_DatOut	out	std_logic_vector	32	
AxisTxPortSC BStrobe_ValOut	out	std_logic_vector	4	
AxisTxPortSC Keep_ValOut	out	std_logic_vector	4	
AxisTxPortSC Last_ValOut	out	std_logic	1	
AxisTxPortSC User_DatOut	out	std_logic_vector	3	
Enable Input				
Enable_EnaIn	in	std_logic	1	Enable core

Table 19: Supervision Processor

4.2.6 Sequence Number Processor

4.2.6.1 Entity Block Diagram

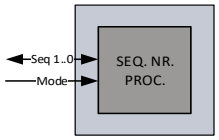

Figure 16: Sequence Number Processor

4.2.6.2 Entity Description

Sequence Number Processor

This module answers Sequence Number requests sequentially, so that each requestor gets a unique Sequence Number. The Sequence Number is incremented by 1 for each request wrapping through 0 (16bit).

4.2.6.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
NoSupport_Gen	-	boolean	1	Support for NO (ust forward no dropping, but duplication)
HsrSupport_Gen	-	boolean	1	Support for HSR
PrpSupport_Gen	-	boolean	1	Support for PRP
HsrPrpSupport_Gen	-	boolean	1	Support for HSR-PRP RedBox
HsrHsrSupport_Gen	-	boolean	1	Support for HSR-HSR RedBox
Proxy Node table				
NumberOfPorts_Gen	-	natural	1	How many requestors are connected
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset

Mode Output				
RedMode_DatIn	in	Red_Mode_Type	1	Redundancy Mode: Hsr_E Prp_E HsrPrp_E HsrHsr_E No_E
Sequence Number Info Input				
SeqReq_DatIn	in	Red_SeqReqArray_Type	Number OfPorts _Gen	Sequence Number Processor request
SeqReq_ValIn	in	Red_SeqReqVal Array_Type	Number OfPorts _Gen	Sequence Number Processor request valid
Sequence Number Info Output				
SeqResp_DatOut	out	Red_SeqRespArray_Type	Number OfPorts _Gen	Sequence Number Processor response
SeqResp_ValOut	out	Red_SeqRespVal Array_Type	Number OfPorts _Gen	Sequence Number Processor response valid

Table 20: Sequence Number Processor

4.2.7 Ethernet Interface Adapter

4.2.7.1 Entity Block Diagram

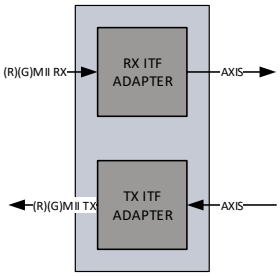

Figure 17: Ethernet Interface Adapter

4.2.7.2Entity Description

RX Interface Adapter

This module convert the Media Independent Interface (R)(G)MII data stream (2/4/8bit) into a 32bit AXI stream. First bytes on the cable are mapped to the AXI MSB of the data array. It contains an asynchronous Fifo to on one hand do clock domain crossing from the external clock to the system clock and on the other hand also to minimal buffer data for speed differences. The Fifo size is kept quite small to assure correct timestamp alignment with the frame. It converts the different data widths into a 32bit block AXI stream. The Preamble and SFD are removed on reception.

TX Interface Adapter

This module convert the 32bit AXI stream into a Media Independent Interface (R)(G)MII data stream (2/4/8bit) which is continuous. The MSB of the AXI data array is mapped to the first byte on the cable. It contains an asynchronous Fifo to on one hand do clock domain crossing from the system clock to the external clock and on the other hand also to minimal buffer data for speed differences. The Fifo size is kept quite small to assure correct timestamp alignment with the frame. It converts the 32bit block AXI stream into the different data widths. The Preamble and SFD are added before transmission. It also assures the correct interframe gap between frames.

4.2.7.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
Interface Adapter				
IoFf_Gen	-	boolean	1	Shall IO flip flops be instantiated
ClockClkPeriodNano-second_Gen	-	natural	1	Integer Clock Period
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
(R)(G)Mii RX Clk/Rst Input				
(R)(G)MiiRxClk_ClkIn	in	std_logic	1	RX Clock
(R)(G)MiiRxRstN_RstIn	in	std_logic	1	Reset aligned with RX Clock
(R)(G)Mii TX Clk/Rst Input				
(R)(G)MiiTxClk_ClkIn	in	std_logic	1	TX Clock
(R)(G)MiiTxRstN_RstIn	in	std_logic	1	Reset aligned with TX Clock
(R)(G)Mii RX Data Input/Output				
(R)(G)MiiRxDv_Ena	In/ out	std_logic	1	RX Data valid
(R)(G)MiiRxErr_Ena	In/ out	std_logic	1	RX Error
(R)(G)MiiRxData_Dat	In/ out	std_logic_vector	2-8	RX Data MII:4, RMII:2, GMII:8, RGMII:4
(R)(G)MiiCol_Dat	In/ out	std_logic	1	Collision
(R)(G)MiiCrs_Dat	In/ out	std_logic	1	Carrier Sense
(R)(G)Mii TX Data Input				
(R)(G)MiiTxEn_Ena	In/ out	std_logic	1	TX Data valid
(R)(G)MiiTxErr_Ena	In/ out	std_logic	1	TX Error
(R)(G)MiiTxData_Dat	In/	std_logic_vector	2-8	TX Data

	out			MII:4, RMII:2, GMII:8, RGMII:4
Axi Input				
AxisValid_ValIn	in	std_logic	1	AXI Stream frame input
AxisReady_ValOut	out	std_logic	1	
AxisData_DatIn	in	std_logic_vector	32	
AxisStrobe_ValIn	in	std_logic_vector	4	
AxisKeep_ValIn	in	std_logic_vector	4	
AxisLast_ValIn	in	std_logic	1	
AxisUser_DatIn	in	std_logic_vector	3	
Axi Output				
AxisValid_ValOut	out	std_logic	1	AXI Stream frame output
AxisReady_ValIn	in	std_logic	1	
AxisData_DatOut	out	std_logic_vector	32	
AxisStrobe_ValOut	out	std_logic_vector	4	
AxisKeep_ValOut	out	std_logic_vector	4	
AxisLast_ValOut	out	std_logic	1	
AxisUser_DatOut	out	std_logic_vector	3	

Table 21: Ethernet Interface Adapter

4.2.8 Registerset

4.2.8.1 Entity Block Diagram

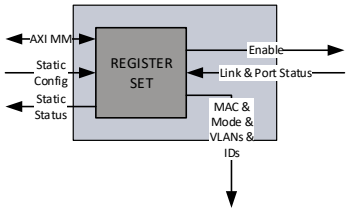

Figure 18: Registerset

4.2.8.2 Entity Description

Register Set

This module is an AXI Light Memory Mapped Slave. It provides access to all registers and allows configuring the RED HsrPrp Core. AXI4 Light only supports 32 bit wide data access, no byte enables, no burst, no simultaneous read and writes and no unaligned access. It can be configured to either run in AXI or StaticConfig mode. If in StaticConfig mode, the configuration of the registers is done via signals and can be easily done from within the FPGA without CPU. For each parameter a valid signal is available, the enable signal shall be set last (or simultaneously). To change configuration parameters the core has to be disabled and enabled again. If in AXI mode, an AXI Master has to configure the registers with AXI writes to the registers, which is typically done by a CPU. Parameters can in this case also be changed at runtime.
In addition it counts frames and errors of all ports to allow status supervision. This only when the generic PortStatusSupport_Gen is true

4.2.8.3 Entity Declaration

Name	Dir	Type	Size	Description
Generics				
General				
NoSupport_Gen	-	boolean	1	Support for NO (ust forward no dropping, but duplication)

HsrSupport_Gen	-	boolean	1	Support for HSR
PrpSupport_Gen	-	boolean	1	Support for PRP
HsrPrpSupport_Gen	-	boolean	1	Support for HSR-PRP RedBox
HsrHsrSupport_Gen	-	boolean	1	Support for HSR-HSR RedBox
CutThrough_Gen	-	boolean	1	Support for Cut Through frame handling
VlanSupport_Gen	-	boolean	1	Support for VLAN
TailTagging_Gen	-	boolean	1	Support for Tail tagging and untag-ging to define to send to specific ports and to know from which port it was received
Register Set				
StaticConfig_Gen	-	boolean	1	If Static Configura-tion or AXI is used
PortStatus Support_Gen	-	boolean	1	If frames and error counters shall be available in the registerset
AxiAddressRange Low_Gen	-	std_logic_vector	32	AXI Base Address
AxiAddressRange High_Gen	-	std_logic_vector	32	AXI Base Address plus Registerset Size
Ports				
System				
SysClk_ClkIn	in	std_logic	1	System Clock
SysRstN_RstIn	in	std_logic	1	System Reset
Config				
StaticConfig_DatIn	in	Red_HsrPrp StaticConfig_Type	1	Static Configuration
StaticConfig_ValIn	in	Red_HsrPrp StaticConfigVal	1	Static Configuration valid

		_Type		
Status				
StaticStatus_DatOut	out	Red_HsrPrp StaticStatus_Type	1	Static Status
StaticStatus_ValOut	out	Red_HsrPrp StaticStatusVal _Type	1	Static Status valid
Port Status Input				
PortStatus PortA_DatIn	in	Red_Port Status_Type	1	Port Status
PortStatus PortB_DatIn	in	Red_Port Status_Type	1	Port Status
PortStatus PortC_DatIn	in	Red_Port Status_Type	1	Port Status
AXI4 Light Slave				
AxiWriteAddrValid _ValIn	in	std_logic	1	Write Address Valid
AxiWriteAddrReady _RdyOut	out	std_logic	1	Write Address Ready
AxiWriteAddrAddress _AdrIn	in	std_logic_vector	32	Write Address
AxiWriteAddrProt _DatIn	in	std_logic_vector	3	Write Address Protocol
AxiWriteDataValid _ValIn	in	std_logic	1	Write Data Valid
AxiWriteDataReady _RdyOut	out	std_logic	1	Write Data Ready
AxiWriteDataData _DatIn	in	std_logic_vector	32	Write Data
AxiWriteDataStrobe _DatIn	in	std_logic_vector	4	Write Data Strobe
AxiWriteRespValid _ValOut	out	std_logic	1	Write Response Valid
AxiWriteRespReady _RdyIn	in	std_logic	1	Write Response Ready
AxiWriteResp Response_DatOut	out	std_logic_vector	2	Write Response
AxiReadAddrValid _ValIn	in	std_logic	1	Read Address Valid
AxiReadAddrReady _RdyOut	out	std_logic	1	Read Address Ready
AxiReadAddrAddress _AdrIn	in	std_logic_vector	32	Read Address
AxiReadAddrProt _DatIn	in	std_logic_vector	3	Read Address

				Protocol
AxiReadDataValid_ValOut	out	std_logic	1	Read Data Valid
AxiReadDataReady_RdyIn	in	std_logic	1	Read Data Ready
AxiReadData Response_DatOut	out	std_logic_vector	2	Read Data
AxiReadDataData_DatOut	out	std_logic_vector	32	Read Data Re-sponse
Timeout Input				
SupervisionTimeout PortA_DatIn	in	std_logic	1	Whether Port A has a Supervision timeout
SupervisionTimeout PortB_DatIn	in	std_logic	1	Whether Port B has a Supervision timeout
Own MAC Output				
OwnMac_DatOut	out	Common_Byte_Type	6	MAC address of the node
VLAN Output				
Vlan_DatOut	out	Red_Vlan_Type	6	VLAN
VlanEnable_DatOut	out	std_logic	1	VLAN mode ena-bled
Mode Output				
RedMode_DatOut	out	Red_Mode_Type	1	Redundancy Mode: Hsr_E Prp_E HsrPrp_E HsrHsr_E No_E
RedBoxNetId_DatOut	out	std_logic_vector	3	RedBox Network Identifier for HSR-PRP mode
RedBoxId_DatOut	out	std_logic_vector	4	RedBox Identifier for HSR-PRP mode
Promiscuous-Mode_DatOut	out	std_logic	1	If in Promiscuous mode
CutThrough_ValOut	out	std_logic	1	If cut through shall be enabled
NoForward_DatOut	out	std_logic	1	If forwarding be-tween Ports A & B

				shall be disabled
TailTagging_DatOut	out	std_logic	1	If Tail Tagging shall be done
PrpUntagging_ DatOut	out	std_logic	1	If PRP frames shall be untagged
Enable Output				
RedHsrPrp Enable_DatOut	out	std_logic	1	Enables the core

Table 22: Registerset

4.3 Configuration example

In both cases the enabling of the core shall be done last, after or together with the configuration.

4.3.1 Static Configuration

```
    constant RedStaticConfigHsrPrp_Con : Red_HsrPrpStaticConfig_Type := (
        OwnMac                                          => (
            0                                           => x"00",
            1                                           => x"01",
            2                                           => x"02",
            3                                           => x"03",
            4                                           => x"04",
            5                                           => x"05",
        RedMode                                         => Hsr_E,
        RedBoxNetId                                     => "010",
        RedBoxId                                        => x"A",
        NoForward                                       => '0',
        PromiscuousMode                                 => '0',
        PrpUntagging                                    => '0',
        TailTagging                                     => '0',
        CutThrough                                      => '1',
        Vlan                                            => (
            Pcp                                         => "100",
            Dei                                         => '0',
            Vid                                         => x"004"),
        VlanEnable                                      => '1'
  );

    constant RedStaticConfigValHsrPrp_Con : Red_HsrPrpStaticConfigVal_Type := (
        Vlan_Val                                        => '1',
        Enable_Val                                      => '1'
  );
```

Figure 19: Static Configuration

4.3.2 AXI Configuration

The following code is a simplified pseudocode from the testbench: The base address of the RED HsrPrp Core is 0x10000000.

```
-- RED HSR/PRP
-- Config
-- Mode Hsr
AXI WRITE 10000084 00000002
-- VLAN valid and 4
AXI WRITE 10000088 00010004
```

```
-- Mode and VLAN valid
AXI WRITE 10000080 00000003
-- Own MAC 00:01:02:03:04:05
AXI WRITE 10000104 03020100
AXI WRITE 10000108 00000504
-- MAC valid
AXI WRITE 10000100 00000001
-- Enable
AXI WRITE 10000000 00000001
```

Figure 20: AXI Configuration

In the example the VLAN is enabled and Promiscuous Mode is disabled.

4.4 Clocking and Reset Concept

4.4.1 Clocking

To keep the design as robust and simple as possible, the whole RED HsrPrp Core, including the Counter Clock and all other cores from NetTimeLogic are run in one clock domain. This is considered to be the system clock. Per default this clock is 50MHz. Where possible also the interfaces are run synchronous to this clock. For clock domain crossing asynchronous fifos with gray counters or message patterns with meta-stability flip-flops are used. Clock domain crossings for the AXI interface is moved from the AXI slave to the AXI interconnect.

Clock	Frequency	Description
System		
System Clock	50MHz (Default)	System clock where the RED HsrPrp runs on as well as the counter clock etc.
(R)(G)MII Interface		
PHY (R)(G)MII RX Clock	2.5/25/125MHz	Asynchronous, external receive clock from the PHY also used for the MAC. Depending on the interface not all frequencies apply.
PHY (R)(G)MII TX Clock	2.5/25/125MHz	Asynchronous, external transmit clock to/from the PHY also used for the MAC. Depending on the interface not all frequencies apply.
AXI Interface		
AXI Clock	50MHz (Default)	Internal AXI bus clock, same as the system clock

Table 23: Clocks

4.4.2Reset

In connection with the clocks, there is a reset signal for each clock domain. All resets are active low. All resets can be asynchronously set and shall be synchronously released with the corresponding clock domain. All resets shall be asserted for the first couple (around 8) clock cycles. All resets shall be set simultaneously and released simultaneously to avoid overflow conditions in the core. See the reference designs top file for an example of how the reset shall be handled.

Reset	Polarity	Description
System		
System Reset	Active low	Asynchronous set, synchronous release with the system clock
(R)(G)MII Interface		
PHY (R)(G)MII RX Reset	Active low	Asynchronous set, synchronous release with the (R)(G)MII RX clock
PHY (R)(G)MII TX Reset	Active low	Asynchronous set, synchronous release with the (R)(G)MII TX clock
AXI Interface		
AXI Reset	Active low	Asynchronous set, synchronous release with the AXI clock, which is the same as the system clock

Table 24: Resets

5 Resource Usage

Since the FPGA Architecture between vendors and FPGA families differ there is a split up into the two major FPGA vendors.

5.1 Altera (Cyclone V)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (PRP, DAN, no VLAN, 16k entries, static config, 100Mbit)	5591	15259	96	0
Maximal (HSR&PRP, RedBox, VLAN, 32k entries, 128 proxy, AXI, 1000Mbit)	6881	18710	181	0

Table 25: Resource Usage Altera

5.2 Xilinx (Kintex 7)

Configuration	FFs	LUTs	BRAMs	DSPs
Minimal (PRP, DAN, no VLAN, 16k entries, static config, 100Mbit)	5471	13283	23	0
Maximal (HSR&PRP, RedBox, VLAN, 32k entries, 128 proxy, AXI, 1000Mbit)	7540	19626	56	0

Table 26: Resource Usage Xilinx

6 Delivery Structure

```
AXI                                 -- AXI library folder
 |-Library                          -- AXI library component sources
 |-Package                          -- AXI library package sources

CLK                                 -- CLK library folder
|-Package                           -- CLK library package sources

COMMON                              -- COMMON library folder
 |-Library                          -- COMMON library component sources
 |-Package                          -- COMMON library package sources

PTP                                 -- PTP library folder
 |-Package                          -- PTP library package sources

RED                                 -- RED library folder
|-Core                              -- RED library cores
 |-Doc                              -- RED library cores documentations
 |-Library                          -- RED library component sources
 |-Package                          -- RED library package sources
 |-Refdesign                        -- RED library cores reference designs
 |-Testbench                        -- RED library cores testbench sources and sim/log

SIM                                 -- SIM library folder
 |-Doc                              -- SIM library command documentation
 |-Package                          -- SIM library package sources
 |-Testbench                        -- SIM library testbench template sources
 |-Tools                            -- SIM simulation tools
```

7 Testbench

The HSR&PRP Core testbench consist of 2 parse/port types: AXI and ETH. Multiple instances exist. PHY A&B ETH ports are connected to the port going to the PHY from the DUT (which acts like a MAC). MAC C is connected to the port going to the MAC from the DUT (which acts like a PHY). A Clock instance provides the 1ms timer event for the DUT.
In addition for configuration and result checks an AXI read and write port is used in the testbench and for accessing more than one AXI slave also an AXI interconnect is required.

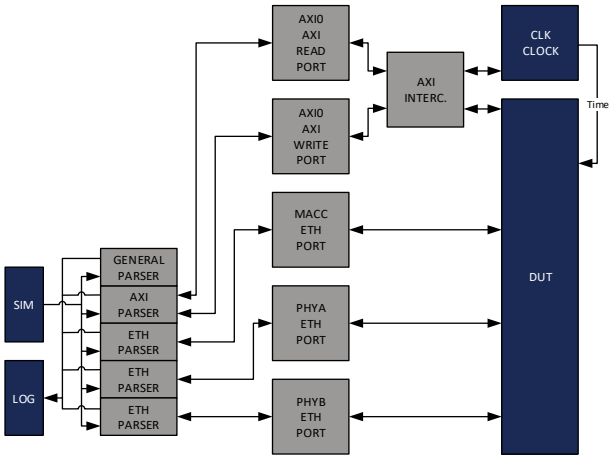

Figure 21: Testbench Framework

For more information on the testbench framework check the Sim_ReferenceManual documentation.

With the Sim parameter set the time base for timeouts are divided by 1000 to 100000 to speed up simulation time.

7.1 Run Testbench

1. Run the general script first

```
source XXX/SIM/Tools/source_with_args.tcl
```

2. Start the testbench with all test cases

```
src XXX/RED/Testbench/Core/RedHsrPrp/Script/run_Red_HsrPrpMii_Tb.tcl
```

3. Check the log file LogFile1.txt in the XXX/RED/Testbench/Core/RedHsrPrp/Log/ folder for simulation results.

8 Reference Designs

The HSR&PRP Core reference design contains a PLL to generate all necessary clocks (cores are run at 50 MHz), an instance of the RED HsrPrp Core IP core and an instance of the Adjustable Counter Clock IP core (needs to be purchased separately). The Reference Design is intended to be connected to any HSR or PRP device handling redundancy according to IEC 62439-3 or it can be set to NON mode and a daisy chain can be built.
All generics can be adapted to the specific needs.

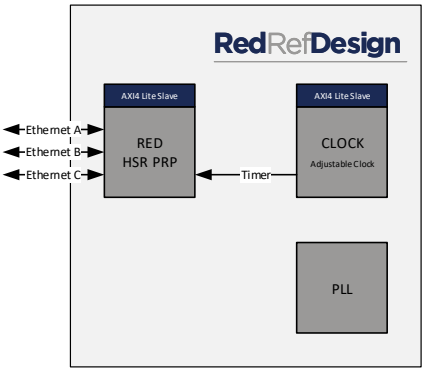

Figure 22: Reference Design

8.1 Xilinx: Digilent NetFpga

The NetFpga board is an FPGA board from Digilent Inc. with a Xintex7 FPGA from Xilinx. (http://store.digilentinc.com/netfpga-1g-cml-kintex-7-fpga-development-board)

1. Open Vivado 2017.4
2. Run TCL script
   /RED/Refdesign/Xilinx/NetFpga/RedHsrPrpMii/RedHsrPrp.tcl
   a. This has to be run only the first time and will create a new Vivado Project
3. If the project has been created before open the project and do not rerun the project TCL
4. Rerun implementation
5. Download to FPGA via JTAG

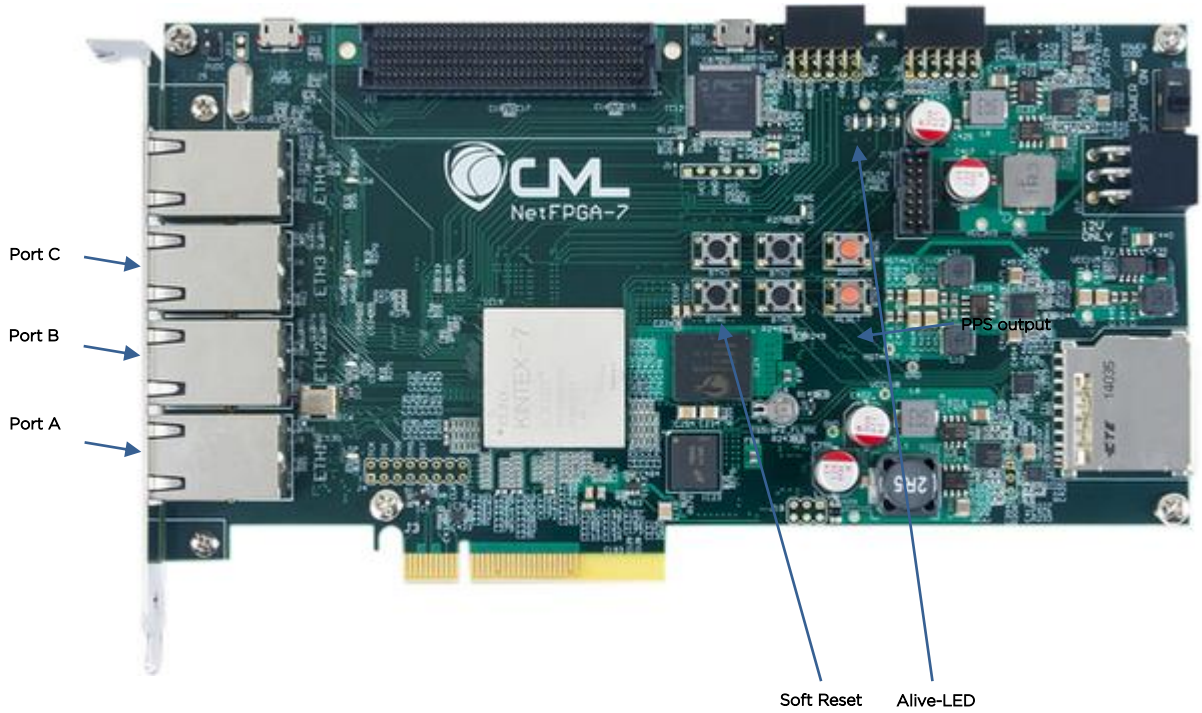

Figure 23: NetFPGA (source Digilent Inc)

A List of tables

B List of figures