UDP/IP/Ethernet IP Core

User Manual

Purpose

This document shows the features, interfaces and configuration options of the UDP/IP/Ethernet IP core and to provide the user with a comprehensive guide to understand and use the IP core.

Summary

This document first gives an overview of the UDP/IP/Ethernet IP core followed by a detailed description of its features and configuration options. In addition, it explains the functionality of the reference design and how to build it.

Product Information	Number	Name
Product	EN-UDP	UDP/IP/Ethernet IP Core
Base Development Project	106	UDP/IP/Ethernet IP Core

Document Information			Reference	Version	Date
Reference	Version	Date	D-0000-106-002	2020.07.08.001	08.07.2020

Approval Information	Name	Position	Date
Written by	Marc Oberholzer	Design Engineer	08.07.2020
Verified by	Christoph Glattfelder	Product Manager	08.07.2020
Approved by	Christoph Glattfelder	Product Manager	08.07.2020

Copyright Reminder

Document History

Version	Date	IP Core Version	Author	Comment
2020.07.08.001	08.07.2020	2020.07.08	C. Glattfelder	Bugfix
2020.05.13.001	13.05.2020	2020.05.13	C. Glattfelder	Added Destination MAC and IP per UDP interface
2019.11.29.001	29.11.2019	2019.11.29	C. Glattfelder	Bugfix in the UltraScale IMAC Interface
2019.09.20.001	20.09.2019	2019.09.20	C. Glattfelder	Updated tool version to Vivado 2019.1 and Quartus 18.1
2017.09.04.001	04.09.2017	2017.09.04	C. Glattfelder	Updated tool version to Vivado 2016.4 and Quartus 16.1
2017.08.28.001	28.08.2017	2017.08.28	C. Glattfelder	Added Xilinx SGMII interface
2016.10.28.001	28.10.2016	2016.10.28	C. Glattfelder	Some bug fixes, see IP Core Release History
2016.09.23.001	23.09.2016	2016.09.22	C. Glattfelder	Updated folder structure and simulation scripts
2016.04.05.001	05.04.2016	2016.04.05	C. Glattfelder	Added raw Ethernet port

Version	Date	IP Core Version	Author	Comment
2015.06.29.001	29.06.2015	2015.06.29	C. Glattfelder	Added checksum to UDP header mask and generic UseStreamingMode_g
2015.03.24.001	24.03.2015	2013.08.27	Ivan Vrbanek	Added Vivado IPI
2014.07.02.001	02.07.2014	2013.08.27	Marc Oberholzer	Added network setup to reference design section.
2014.01.10.001	10.01.2014	2013.08.27	André Schlegel	Added Xilinx Spartan-6 release
2013.08.27.001	27.08.2013	2013.08.27	Marc Oberholzer	Updated to IP core version 2013.08.27.
2013.08.15.001	13.08.2013	2013.08.15	Marc Oberholzer	Updated to IP core version 2013.08.13, updated product ordering codes, updated target hardware platforms, updated reference design synthesis and simulation guides.
2013.03.26.001	26.03.2013	2013.03.26	Marc Oberholzer	Updated to IP core version 2013.03.26.
2013.03.25.001	25.03.2013	2013.03.25	Marc Oberholzer	Updated to IP core version 2013.03.25.
2013.03.18.002	21.03.2013	2013.03.18	Marc Oberholzer	Added Xilinx 7 Series to target devices, updated product ordering codes, updated reference design simulation section, updated implementation constraints
2013.03.18.001	18.03.2013	2013.03.18	Marc Oberholzer	Updated to IP core version 2013.03.18, added section “Implementation Constraints”, clarified on available license option combinations.

Version	Date	IP Core Version	Author	Comment
2013.01.15.001	15.01.2013	2013.01.15	Marc Oberholzer	Updated to IP core version 2013.01.15.
2012.11.29.001	29.11.2012	2012.11.29	Marc Oberholzer	Updated to IP core version 2012.11.29. Added Virtex-6® to the target FPGA families, did minor document improvements.
2012.10.26.001	25.10.2012	2012.10.26	Marc Oberholzer	First release

IP Core Release History

Version	Date	Comment
2020.07.08	08.07.2020	Fixed issue when writing very fast to the RAW-ETH-TX register
2020.05.13	13.05.2020	Added destination MAC and IP per UDP interface
2019.11.29	29.11.2019	Bugfix in the UltraScale IMAC interfaces
2019.09.20	20.09.2019	Updated tool version to Vivado 2019.1 and Quartus 18.1 Fixed bug in MII interface
2017.09.04	04.09.2017	Updated tool version to Vivado 2016.4 and Quartus 16.1
2017.08.28	28.08.2017	Added Xilinx SGMII Interface
2016.10.28	28.10.2016	Fixed issue that an additional byte is transferred in some cases when using streaming mode. Updated data type of RxBufferCount_g in some wrapper files.
2016.09.22	23.09.2016	Updated folder structure and simulation scripts
2016.04.05	05.04.2016	Added raw Ethernet port
2015.06.29	29.06.2015	Added checksum to UDP header mask and generic UseStreamingMode_g

Version	Date	Comment
2013.08.27	27.08.2013	Fixed a bug in the transmit path that could damage a packet when UdpPortCount_g=1
2013.08.15	15.08.2013	Added RxBufferCount_g interface generic, removed “Version + IHL”, “Total Length” and “Header Checksum” bits from IpTxHdrMask_g interface generic, improved Ethernet and IP receive paths, added library generation TCL scripts for ModelSim®, Added Vivado™ tool flow for Xilinx® 7 Series, updated development tool versions.
2013.03.26	26.03.2013	Fixed an issue where a received IPv6 packet could corrupt the receive stream.
2013.03.25	25.03.2013	Optimized Xilinx® 7 Series RGMII 2.0 transmit timing.
2013.03.18	18.03.2013	Fixed a bug in the MII receive path, updated to Xilinx® ISE/Planahead™ 14.4
2013.01.15	15.01.2013	Added automatic ARP reply generation and header pass-through mode.
2012.11.29	29.11.2012	Version 2012.10.26 couldn't handle packets shorter than the IP header (e.g. ARP packets) with the effect that the subsequent packet got lost. This problem is fixed in this version.
2012.10.26	26.10.2012	First release

Table of Contents

1 Overview

1.1 General

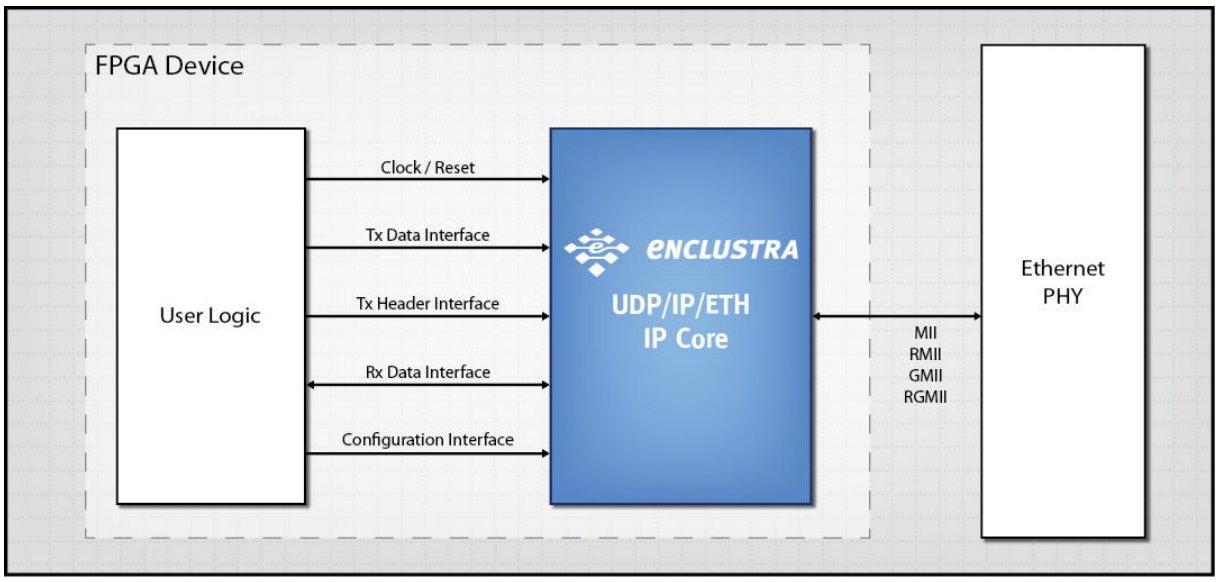


Figure 1: UDP/IP/Ethernet Core Overview

Enclustra's UDP/IP/Ethernet IP core easily enables FPGA-based subsystems to communicate with other subsystems via Ethernet, using the UDP protocol. The IP core is highly configurable and optimally implemented for the use in current Intel® and Xilinx® FPGA architectures. It provides a simple to use interface to the user logic and supports common media independent interfaces as MII, RMII, GMII and RGMII. The IP core, with its 8 bits wide transmit and receive interfaces running at 125 MHz, is able to operate at full 1 Gbit/sec wire speed. 100 Mbit/sec and 10 Mbit/sec operation is also supported.

1.2 Key Features

- 1 Gbit/sec, 100 Mbit/sec and 10 Mbit/sec operation
- MII, RMII, GMII and RGMII media independent interfaces (full-duplex only)
- Complete UDP, IPv4 and Ethernet layer processing
- Automatic ARP reply generation (no local ARP caching)
- Destination UDP port, destination IP address and destination MAC address filtering
- UDP checksum calculation and check
- Ethernet frame check sequence calculation and check
- Header pass-through mode:
    - Selectively override default UDP/IP/Ethernet header field values with values embedded in the transmit data stream

    - Selectively embed UDP/IP/Ethernet header field values in the receive data stream for easy delivery to the user application
- Multiple UDP ports with dedicated receive and transmit interfaces for each port
- Streaming or packet oriented operation mode
- Optional receive data buffers
- Raw-Ethernet port for non UDP packets
- Vivado IPI and Intel QSYS components available
- Low FPGA resource usage
- Encrypted or plain VHDL delivery options
- Project or site license options

1.3 Product Options

Option	Values
Media Independent Interface Type	MII, RMII, GMII, RGMII, IGMII, SGMII
Target FPGA Vendor	Intel®, Xilinx®
Target FPGA Family	Cyclone IV®, Spartan-6®, Virtex-6®, Xilinx® 7 Series (Artix-7™, Kintex-7™, Virtex-7™, ZYNQ-7000™), Xilinx UltraScale(+)®
Delivery Type	Plain VHDL, Encrypted VHDL
License Type	Project, Site, Evaluation

Table 1: Product Options

Table 1 lists the available product options for the UDP/IP/Ethernet IP core. Only one value can be chosen per option. Please see section 8.1.1 for product ordering codes.

The evaluation version is delivered in Encrypted VHDL and stops working after one hour.

1.4 Deliverables

1.4.1 Overview

- UDP/IP/Ethernet IP core
    - VHDL source files (plain or encrypted, depending on product options)
    - Precompiled ModelSim® simulation libraries
    - User manual (PDF download)
- UDP/IP/Ethernet Xilinx® Vivado™ IPI core

- UDP/IP/Ethernet IP core reference design
  - Reference design top-level VHDL file (plain VHDL)
  - Host software to test the UDP loopback (Windows binary and C# source code)
  - UCF / XDC / SDC constraint files (depending on product options)
  - Xilinx® ISE / Xilinx® Vivado™ / Intel® Quartus® project files (depending on product options)
  - Top-level simulation test bench file (plain VHDL)
  - Top-level simulation ModelSim® project file
  - Documentation (integrated in UDP/IP/Ethernet IP core user manual)

1.4.2 Directory Structure

The IP core is delivered as a ZIP archive file with the following directory structure and contents:

- ***ip*** IP core base directory
  - ***hdl*** HDL version of the IP core
    - ***lib*** IP core library directory
      - ***en_udp_ip_eth_<mii>_<vnd>_<arch>_<lic>_cl*** IP core CL library files (encrypted or plain VHDL, depending on product options)
      - ***en_udp_ip_eth_<mii>_<vnd>_<arch>_<lic>_cn*** IP core CN library files (encrypted or plain VHDL, depending on product options)
      - ***en_udp_ip_eth_<mii>_<vnd>_<arch>_<lic>_cs*** IP core CS library files (encrypted or plain VHDL, depending on product options)
    - ***lic*** IP core license file (optional, depending on product options)
    - ***sim*** Precompiled simulation libraries (for encrypted versions only) and compile script
    - ***vhdl*** IP core implementation files (encrypted or plain VHDL, depending on product options) and instantiation wrapper (plain VHDL)
- ***rd*** Reference designs base directory
  - ***host_sw*** Windows application to test the UDP loopback
  - ***en_udp_ip_eth_<mii>_<vnd>_<arch>*** Specific reference design base directory
    - ***bitstream*** precompiled bitstream of the reference design
    - ***modelsim*** ModelSim® simulation scripts

- ***quartus*** Intel® Quartus® project directory (optional, depending on product options)
- ***vivado*** Xilinx® Vivado™ project directory (optional, depending on product options)
- ***vhdl*** Reference design implementation files (plain VHDL)

Some of the directory names are dependent on the product options as follows (see section 8.1.1 for code details):

- ***<mii>:*** MII interface type code
- ***<vnd>:*** Target FPGA vendor code (xil, alt)
- ***<arch>:*** Target FPGA architecture code (c4, c5, s6, x7)
- ***<lic>:*** License type (src, enc, eval)

2 Interface Description

2.1 Interface Generics

Name	Type	Range	Description
ClkFreq_g	positive	>0	Frequency in Hz of the system clock (Clk). Not required by all versions of the core.
UdpPortCount_g	positive	>0	Number of UDP ports. Each port has an individual receive and transmit interface as well as a dedicated receive buffer. Access to the transmit path is arbitrated between the individual UDP ports.
MaxPayloadSize_g	positive	>0	Maximum UDP payload size in bytes. This value is used to determine the required buffer size. No packets with bigger payload can be processed.
RxBufferCount_g	Natural	0 or >= 2	Defines the number of UDP packet that can be stored in the receive buffer (per UDP port). Set to 0 to disable the internal RX buffer.
EnableRawEth_g	boolean	True/false	Enables the optional raw Ethernet port.
RawEthRxBufferCnt_g	Natural	0 or >= 2	Defines the number of packet that can be stored in the receive buffer. Set to 0 to disable the internal RX buffer.

Name	Type	Range	Description
RawEthTxBufferCnt_g	Natural	0 or >= 2	Defines the number of packet that can be stored in the transmit buffer. Set to 0 to disable the internal TX buffer. Without TX buffer a frame must be transferred in one piece, so EthTx_Vld must not go low within one packet.
RawEthPayloadSize_g	positive	>0	Maximum packet size of a raw Ethernet frame.
TimeoutWidth_g	positive	>0	Width of the timeout vector in bits. The timeout duration is limited to $2^{TimeoutWidth\_g}$ clock cycles.
RoundRobin_g	boolean	True/false	Round robin arbitration enable flag True: A round robin arbiter is used to arbitrate between individual Tx UDP ports. False: A priority arbiter is used to arbitrate between individual Tx UDP ports. The UDP port with index 0 has the highest priority.
ArpReply_g	boolean	True/false	ARP reply enable flag True: An ARP reply packet is generated upon the reception of a valid ARP request packet. False: Received ARP request packets are ignored; no ARP reply packets are generated.

Name	Type	Range	Description
UseStreamingMode_g	boolean	True/false	Enables streaming mode True: Streaming mode: Packets are automatically sent when the timeout expires or Cfg_MTU bytes are in the buffer. The user can also start a transmit by assert Tx_Snd. Header pass-through mode is not supported. False: Packet mode: Packets are only sent after Tx_Last is asserted. Cfg_MTU and Cfg_Timeout have no impact and can be set to 0.
UdpRxHdrMask_g	std_logic_vector (0 to 3)	“0000”.. ”1111”	UDP receive header fields pass through enable mask Bit 0: Source UDP port number Bit 1: Destination UDP port number Bit 2: Length Bit 3: UDP checksum
UdpTxHdrMask_g	std_logic_vector (0 to 1)	“00”.. ”11”	UDP transmit header fields pass through enable mask Bit 0: Source UDP port number Bit 1: Destination UDP port number

Name	Type	Range	Description
IpRxHdrMask_g	std_logic_vector (0 to 9)	“0000000000”.. ”1111111111”	IP receive header fields pass through enable mask Bit 0: Version + IHL Bit 1: ECN+DCSP Bit 2: Total Length Bit 3: Identification Bit 4: Fragment Offset+Flags Bit 5: Time To Live Bit 6: Protocol Bit 7: Header Checksum Bit 8: Source IP Address Bit 9: Destination IP Address
IpTxHdrMask_g	std_logic_vector (0 to 6)	“0000000”.. ”1111111”	IP transmit header fields pass through enable mask Bit 0: ECN+DCSP (formerly TOS) Bit 1: Identification Bit 2: Fragment Offset+Flags Bit 3: Time To Live Bit 4: Protocol Bit 5: Source IP Address Bit 6: Destination IP Address
EthRxHdrMask_g	std_logic_vector (0 to 2)	“000”.. ”111”	Ethernet receive header fields pass through enable mask Bit 0: Destination MAC Address Bit 1: Source MAC Address Bit 2: Ethertype
EthTxHdrMask_g	std_logic_vector (0 to 2)	“000”.. ”111”	Ethernet transmit header fields pass through enable mask Bit 0: Destination MAC Address Bit 1: Source MAC Address Bit 2: Ethertype

Table 2: Interface Generics

2.2 Local Interface Ports

2.2.1 User Clock and Reset Ports

Name	Width	Direction	Description
Clk	1	Input	User clock >= 125 MHz for 1 Gbit/sec >= 25 MHz for 10/100 Mbit/sec operation
Rst	1	Input	Reset synchronous to user clock

Table 3: User Clock and Reset Ports

2.2.2 UDP Transmit Data Interface Ports

Name	Width	Direction	Description
Tx_Data	8* UdpPortCount_g	Input	Transmit data 8 bits per UDP port. Contains the UDP packet payload data to be sent.
Tx_Vld	UdpPortCount_g	Input	Transmit valid One bit per UDP port. Used to qualify *Tx_Data*, *Tx_Lst* and *Tx_Err*.
Tx_Lst	UdpPortCount_g	Input	Transmit last One bit per UDP port. Used to mark the last data byte of a packet in packet based operation. Must be driven constantly to '0' for streaming operation.
Tx_Rdy	UdpPortCount_g	Output	Transmit ready One bit per UDP port. Signalizes the user design whether the core is ready to accept transmit data or not.

Name	Width	Direction	Description
Tx_Err	UdpPortCount_g	Input	Transmit error One bit per UDP port. Used to mark a packet as erroneous. Must be asserted together with *Tx_Vld* and *Tx_Lst*. A packet marked as erroneous is discarded within the core and is not transmitted to the Ethernet physical.
Tx_Snd	UdpPortCount_g	Input	Transmit send now One bit per UDP port. Used to initiate the transmission of the current content of the transmit buffer in streaming operation.

Table 4: UDP Transmit Data Interface Ports

2.2.3 UDP Receive Data Interface Ports

Name	Width	Direction	Description
Rx_Data	8* UdpPortCount_g	Output	Receive data 8 Bits per UDP port. Contains the received UDP packet payload data.
Rx_Vld	UdpPortCount_g	Output	Receive valid One bit per UDP port. Used to qualify *Rx_Data*, *Rx_Lst* and *Rx_Err*.
Rx_Rdy	UdpPortCount_g	Input	Receive ready One bit per UDP port. Asserting *Rx_Rdy* while *Rx_Vld* is '1' reads one byte from the receive buffer. *Rx_Rdy* is not used when the receive buffer is bypassed.
Rx_Lst	UdpPortCount_g	Output	Receive last One bit per UDP port. Used to mark the last data byte of a packet. Can be ignored in streaming operation.

Name	Width	Direction	Description
Rx_Err	UdpPortCount_g	Output	Rx error One bit per UDP port. Used to mark a packet as erroneous when the receive buffer is bypassed. Is always zero when the receive buffer is used, as erroneous packets are discarded in the receive buffer.

Table 5: UDP Receive Data Interface Ports

2.2.4 Raw Ethernet Transmit Data Interface Ports

Name	Width	Direction	Description
EthTx_Data	8	Input	Transmit data Contains the raw Ethernet packet payload data to be sent.
EthTx_Vld	1	Input	Transmit valid Used to qualify *EthTx_Data*, *EthTx_Lst* and *EthTx_Err*.
EthTx_Lst	1	Input	Transmit last Used to mark the last data byte of a packet.
EthTx_Rdy	1	Output	Transmit ready Signalizes the user design whether the core is ready to accept transmit data or not.
EthTx_Err	1	Input	Transmit error Used to mark a packet as erroneous. Must be asserted together with *EthTx_Vld* and *EthTx_Lst*. A packet marked as erroneous is discarded within the core and is not transmitted to the Ethernet physical.

Table 6: Raw Ethernet Transmit Data Interface Ports

2.2.5 Raw Ethernet Receive Data Interface Ports

Name	Width	Direction	Description
EthRx_Data	8	Output	Receive data Contains the received raw Ethernet packet payload data.
EthRx_Vld	1	Output	Receive valid Used to qualify *EthRx_Data*, *EthRx_Lst* and *EthRx_Err*.
EthRx_Rdy	1	Input	Receive ready Asserting *EthRx_Rdy* while *EthRx_Vld* is '1' reads one byte from the receive buffer. *EthRx_Rdy* is not used when the receive buffer is bypassed.
EthRx_Lst	1	Output	Receive last Used to mark the last data byte of a packet.
EthRx_Err	1	Output	Rx error Used to mark a packet as erroneous when the receive buffer is bypassed. Is always zero when the receive buffer is used, as erroneous packets are discarded in the receive buffer.

Table 7: Raw Ethernet Receive Data Interface Ports

2.2.6 Header Interface Ports

Name	Width	Direction	Description
Hdr_UdpTxSrc	16* UdpPortCount_g	Input	UDP transmit source port number Static, 16 bits per UDP port. This value is used for the UDP header's source port field.

Name	Width	Direction	Description
Hdr_UdpTxDst	16* UdpPortCount_g	Input	UDP transmit destination port number Static, 16 bits per UDP port. This value is used for the UDP header's destination port field.
Hdr_UdpRxPort	16* UdpPortCount_g	Input	UDP receive destination port number Static, 16 bits per UDP port. This value is used for the UDP receive destination port filtering.
Hdr_UdpRxPortMask	16* UdpPortCount_g	Input	UDP receive destination port mask Static, 16 bits per UDP port. This value is used for the UDP receive destination port filtering.
Hdr_IpSrc	32	Input	Source IP address Static, global. This value is used for the IP header's source IP address field.
Hdr_IpDst	32* UdpPortCount_g	Input	Destination IP address Static, per UDP interface. This value is used for the IP header's destination IP address field.
Hdr_IpDscp	8	Input	Differentiated services code point Static, global. This value is used for the IP header's DSCP field.
Hdr_IpId	16	Input	Identification Static, global. This value is used for the IP header's Identification field.
Hdr_IpFlags	3	Input	Flags Static, global. This value is used for the IP header's Flags field.

Name	Width	Direction	Description
Hdr_IpTtl	8	Input	Time to live Static, global. This value is used for the IP header's TTL field.
Hdr_MacSrc	48	Input	Source MAC address Static, global. This value is used for the Ethernet header's source MAC address field and for MAC address filtering on the receive path.
Hdr_MacDst	48* UdpPortCount_g	Input	Destination MAC address Static, per UDP interface. This value is used for the Ethernet header's destination MAC address field.
Hdr_RawEthMac	48	Input	MAC address to filter raw Ethernet packets Static, global. This value is used to filter frames received on raw Ethernet port.

Table 8: Header Interface Ports

The header interface ports are intended for static configuration. The assigned values should thus not be changed during operation. If some of the UDP/IP/Ethernet header fields need to be changed during operation, they should be enabled for header pass-through mode.

2.2.7 User Configuration Interface Ports

Name	Width	Direction	Description
Cfg_TxEn	1	Input	Transmit path enable 0: Transmit path disabled 1: Transmit path enabled
Cfg_RxEn	1	Input	Receive path enable 0: Receive path disabled 1: Receive path enabled

Name	Width	Direction	Description
Cfg_CheckRawEthMac	1	Input	MAC address filter enable for raw Ethernet port 0: MAC address filter disabled 1: MAC address filter enabled. Only Ethernet frames with destination MAC address equal to *Hdr_MacSrc,* multicasts and broadcasts are accepted.
Cfg_CheckMac	1	Input	MAC address filter enable (global) 0: MAC address filter disabled. Ethernet frames to arbitrary destination MAC addresses are accepted. 1: MAC address filter enabled. Only Ethernet frames with destination MAC address equal to *Hdr_RawEthMac,* multicasts and broadcasts are accepted.
Cfg_CheckIp	1	Input	IP address filter enable (global) 0: IP address filter disabled. IP packets to arbitrary destination IP addresses are accepted. 1: IP address filter enabled. Only IP packets with destination IP address equal to *Hdr_IpSrc,* multicasts and broadcasts are accepted.
Cfg_CheckUdp	1	Input	UDP port filter enable (global) 0: UDP port filter disabled. UDP packets to arbitrary destination UDP ports are accepted. 1: UDP port filter enabled. Only UDP packets with destination UDP port equal to *Hdr_UdpSrcPort* are accepted.

Name	Width	Direction	Description
Cfg_Mtu	14* UdpPortCount_g	Input	Maximum transmission unit, 14 bits per UDP port, value in bytes. The transmission is started after Cfg_Mtu bytes are in the buffer. Only used in streaming mode, can be set to zero in packet mode.
Cfg_Timeout	TimeoutWidth_g	Input	Transmit timeout The transmission of the current transmit buffer content is triggered *Cfg_Timeout* clock cycles after the last write access to the transmit buffer. This feature should only be enabled for streaming applications. A value of 0 disables the transmit timeout feature.

Table 9: User Configuration Interface Ports

The user configuration interface ports are intended for static configuration. The assigned values should thus not be changed during operation.

2.3 Ethernet Physical Interface Ports

2.3.1 Media Independent Interface (MII) Ports

Name	Width	Direction	Description
Receive Interface Ports			
Mii_RxClk	1	Input	Receive clock 25 MHz for 100 Mbit/sec and 2.5 MHz for 10 Mbit/sec operation.
Mii_RxD	4	Input	Receive data
Mii_RxDv	1	Input	Receive data valid
Mii_RxEr	1	Input	Receive error
Transmit Interface Ports			
Mii_TxClk	1	Input	Transmit clock 25 MHz for 100 Mbit/sec and 2.5 MHz for 10 Mbit/sec operation
Mii_TxD	4	Output	Transmit data
Mii_TxEn	1	Output	Transmit enable

Table 10: Media Independent Interface (MII) Ports

2.3.2 Reduced Media Independent Interface (RMII) Ports

Name	Width	Direction	Description
Common Ports			
Rmii_RefClk	1	Output	Reference clock 50 MHz
Receive Interface Ports			

Name	Width	Direction	Description
Rmii_RxD	2	Input	Receive data
Rmii_RxCrsDv	1	Input	Carrier sense / receive data valid (multiplexed). Dv is active every second clock cycle for 100 Mbit/sec and every 20th clock cycle for 10 Mbit/sec.
Rmii_RxEr	1	Input	Receive error
Transmit Interface Ports			
Rmii_TxD	2	Output	Transmit data
Rmii_TxEn	1	Output	Transmit enable

Table 11: Reduced Media Independent Interface (RMII) Ports

2.3.3 Gigabit Media Independent Interface (GMII) Ports

Name	Width	Direction	Description
Receive Interface Ports			
Gmii_RxClk	1	Input	Receive clock 125 MHz for 1 Gbit/sec, 25 MHz for 100 Mbit/sec and 2.5 MHz for 10 Mbit/sec operation.
Gmii_RxD	8	Input	Receive data
Gmii_RxDv	1	Input	Receive data valid
Gmii_RxEr	1	Input	Receive error
Transmit Interface Ports			
Gmii_TxClk	1	Output	Transmit clock 125 MHz for 1 Gbit/sec, 25 MHz for 100 Mbit/sec and 2.5 MHz for 10 Mbit/sec operation.
Gmii_TxD	8	Output	Transmit data

Name	Width	Direction	Description
Gmii_TxEn	1	Output	Transmit enable
Gmii_TxEr	1	Output	Transmit error (used to corrupt a packet)

Table 12: Gigabit Media Independent Interface (GMII) Ports

2.3.4 Reduced Gigabit Media Independent Interface (RGMII) Ports

Name	Width	Direction	Description
Receive Interface Ports			
Rgmii_RxClk	1	Input	Receive clock 125 MHz for 1 Gbit/sec, 25 MHz for 100 Mbit/sec and 2.5 MHz for 10 Mbit/sec operation.
Rgmii_RxD	4	Input	Receive data (double data rate)
Rgmii_RxCtl	1	Input	Receive control
Transmit Interface Ports			
Rgmii_TxClk	1	Output	Transmit clock 125 MHz for 1 Gbit/s, 25 MHz for 100 Mbit/s and 2.5 MHz for 10 Mbit/s operation
Rgmii_TxD	4	Output	Transmit data (double data rate)
Rgmii_TxCtl	1	Output	Transmit enable

Table 13: Reduced Gigabit Media Independent Interface (RGMII) Ports

2.3.5 Internal Gigabit Media Independent Interface (IGMII) Ports

The IGMII interface is synchronous to the system clock.

Name	Width	Direction	Description
Receive Interface Ports			
IGMII_RxD	8	Input	Receive data
IGMII_RxDv	1	Input	Receive data valid **Attention: at least 2 words with RxDv low are required at the end of a frame!**
IGMII_RxEr	1	Input	Receive data error
IGMII_RxVld	1	Input	Receive handshake signal, indicates that the values on RxD, RxDv and RxEr are valid
Transmit Interface Ports			
IGMII_TxD	8	Output	Transmit data
IGMII_TxEn	1	Output	Transmit enable
IGMII_TxEr	1	Output	Transmit data error
IGMII_TxRdy	1	Input	Receive handshake signal, indicates that the receiver accepts data

Table 14: Internal Gigabit Media Independent Interface (IGMII) Ports

2.3.6 Serial Gigabit Media Independent Interface (SGMII) Ports

The ports of the interface to vendor specific SGMII cores as well as a guide how to connect and configure the according IP-Core are found in a separate document delivered with the UDP/IP/Ethernet core if needed.

2.3.7 Internal MAC Interface (ZYNQ UltraScale+ only)

The Enclustra UDP/IP/Ethernet IP Core also supports the external FIFO Interface of the ZYNQ UltraScale+ EMAC. This allows to use the PS-MAC with a PHY connected to PS-MIO or SGMII.
Please note that the ZYNQ UltraScale EMAC only supports 1G operation if it is used with the external FIFO interface and SGMII.

The external FIFO Interface needs to be enabled in the ZYNQ settings:
PS-PL Configuration->General->Others->GEM[x]

Further the PS-MAC needs to be configured before it can be used with the UDP/IP/Ethernet core. An example of this configuration can be found in the software example of the reference design.

The following things need to be configured:

- Enable the external FIFO interface
- Set the TX clock divider (for 100M operation)
- Set speed, enable full duplex, enable CRC removal
- Configure the MAC address and MAC address filtering
- Enable the MAC

2.4 Version and Debug Interface Ports

Name	Width	Direction	Description
Debug	128	Output	Debug output (for Enclustra-internal use only)
Version	32	Output	IP core version 31:16: Year 15:8: Month 7:0: Day

Table 15: Version Interface Ports

2.5 Local Interface Waveforms

2.5.1 UDP Data Transmission (Packet-Oriented Application)

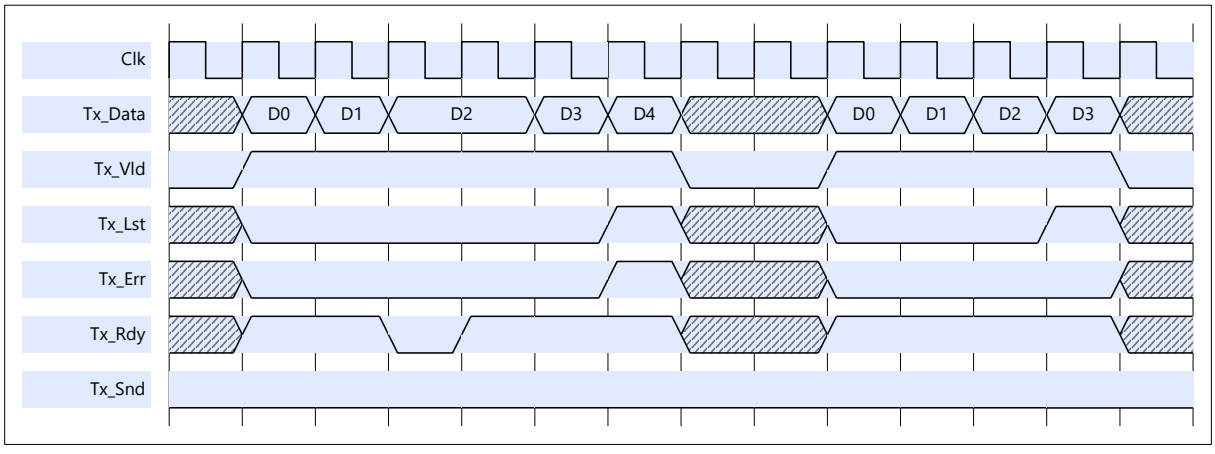


Figure 2: UDP Data Transmission (Packet-Oriented Application)

Figure 2 shows a UDP data transmission example for packet-oriented applications. A UDP packet payload data byte is written into the transmit buffer if *Tx_Vld* = '1' and *Tx_Rdy* = '1'. Please note that the transmit buffer may not be able to accept data at any time, which is signaled to the user design with *Tx_Rdy* = '0'. Also, the user design is allowed to temporarily suspend data delivery by setting *Tx_Vld* = '0'. An erroneous packet is signaled to the UDP/IP/Ethernet core with *Tx_Vld* = '1' and *Tx_Lst* = '1' and *Tx_Err* = '1'. The end of a valid packet is signaled to the UDP/IP/Ethernet core with *Tx_Vld* = '1' and *Tx_Lst* = '1' and *Tx_Err* = '0'. *Tx_Snd* must not be asserted in packet-oriented applications.

2.5.2 UDP Data Transmission (Stream-Oriented Application)

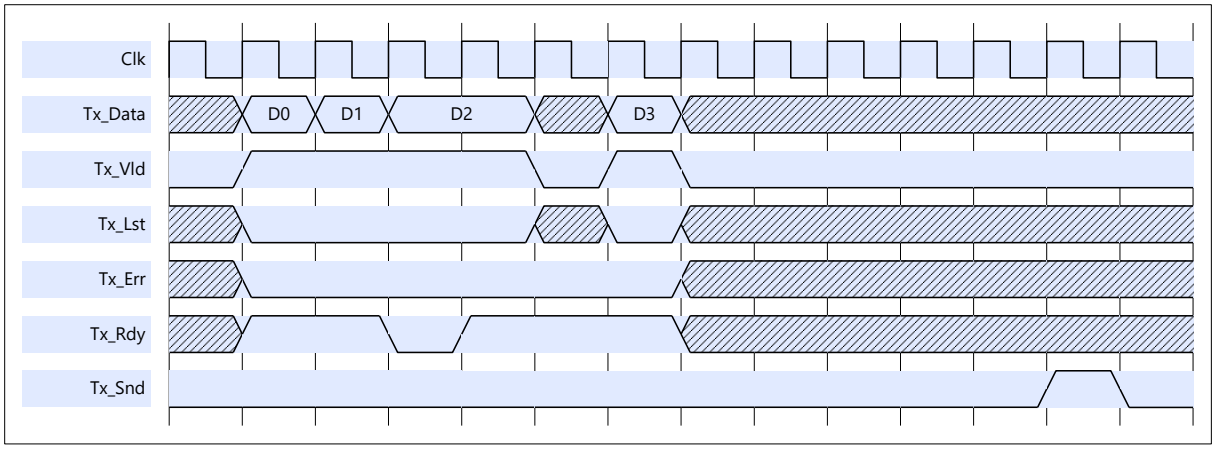


Figure 3: UDP Data Transmission (Stream-Oriented Application)

Figure 3 shows a UDP data transmission example for stream-oriented applications. A UDP packet payload data byte is written into the transmit buffer if *Tx_Vld* = '1' and *Tx_Rdy* = '1'. Please note that the transmit buffer may not be able to accept data at any time, which is signaled to the user design with *Tx_Rdy* = '0'. Also, the user design is allowed to temporarily suspend data delivery by setting *Tx_Vld* = '0'. The actual data transmission can be triggered by three events; either the transmit buffer gets full, the transmit timeout is reached, or the user design asserts *Tx_Snd* = '1'. *Tx_Lst* must not be asserted in stream-oriented applications. Furthermore, transmit header pass-through mode is not supported for stream-oriented applications.

2.5.3 UDP Data Reception (Receive Buffer enabled)

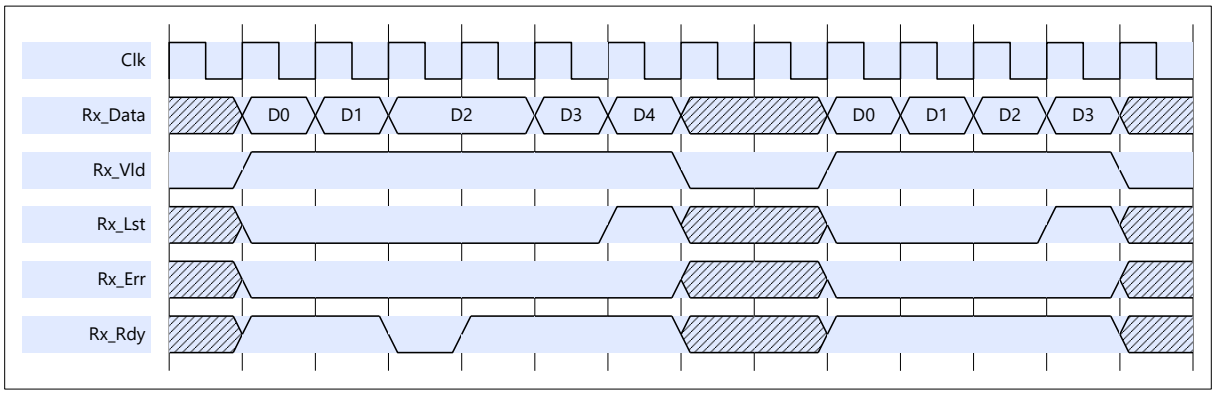


Figure 4: UDP Data Reception (Receive Buffer enabled)

Figure 4 shows a UDP data reception example with enabled receive buffer. A UDP packet payload data byte is read from the receive buffer if *Rx_Vld* = '1' and *Rx_Rdy* = '1'. Please note that the receive buffer may not be able to deliver data at any time, which is signaled to the user design with *Rx_Vld* = '0'. Also, the user design is allowed to temporarily suspend data acceptance by setting *Rx_Vld* = '0'. The end of a packet is signaled to the user design with *Rx_Vld* = '1' and *Rx_Lst* = '1'. Erroneous packets are discarded by the receive buffer, *Rx_Err* is thus never asserted to '1'.

2.5.4 UDP Data Reception (Receive Buffer bypassed)

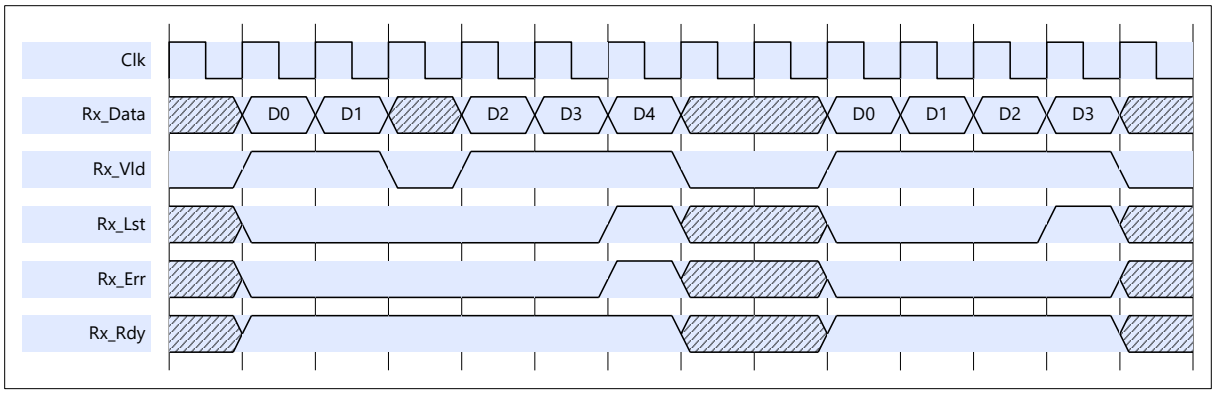


Figure 5: UDP Data Reception (Receive Buffer bypassed)

Figure 5 shows a UDP data reception example with bypassed receive buffer. A UDP packet payload data byte is pushed to the user design if *Rx_Vld* = '1'. Please note that the UDP/IP/Ethernet core may not be able to deliver data at any time, which is signaled to the user design with *Rx_Vld* = '0'. The user design must be able to accept data at any time, so *Rx_Rdy* must be asserted to '1' at any time *Rx_Vld* is asserted to '1'. An erroneous packet is signaled to the user design with *Rx_Vld* = '1' and *Rx_Lst* = '1' and *Rx_Err* = '1'. The end of a valid packet is signaled to the user design with *Rx_Vld* = '1' and *Rx_Lst* = '1' and *Rx_Err* = '0'. Because of erroneous packets passing through the whole receive path, streaming applications must not use the receive buffer bypass mode.

2.6 Header Pass-Through Mode

The header pass-through mode can only be used in packet mode.

2.6.1 Transmit Header Pass-Through Mode

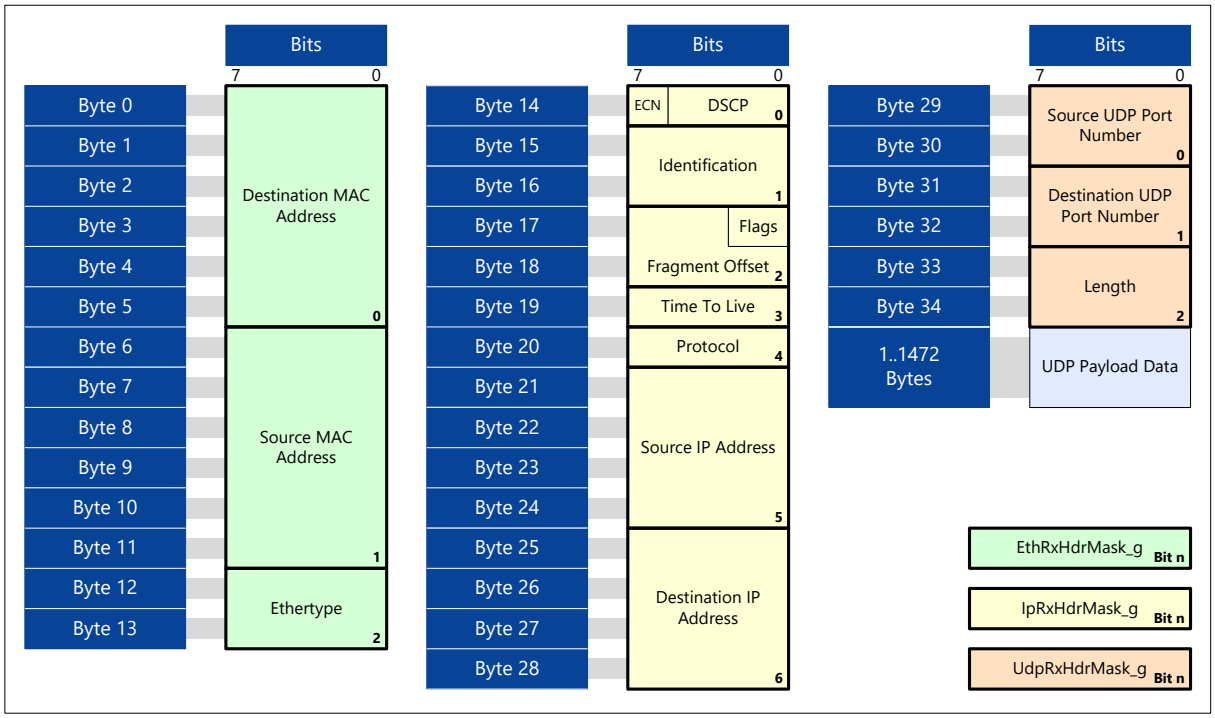


Figure 6: Transmit Header Pass-Through Mode

Figure 6 shows the transmit data byte sequence to be presented to the user transmit data interface for a UDP packet with all supported header fields enabled for transmit header pass-through mode (*UdpTxHdrMask_g* = “11”, *IpTxHdrMask_g* = “1111111”, *EthTxHdrMask_g* = “111”).

If any of the transmit header mask bits is set to zero, the corresponding transmit data byte(s) must be omitted in the transmit data byte sequence.

2.6.2 Receive Header Pass-Through Mode

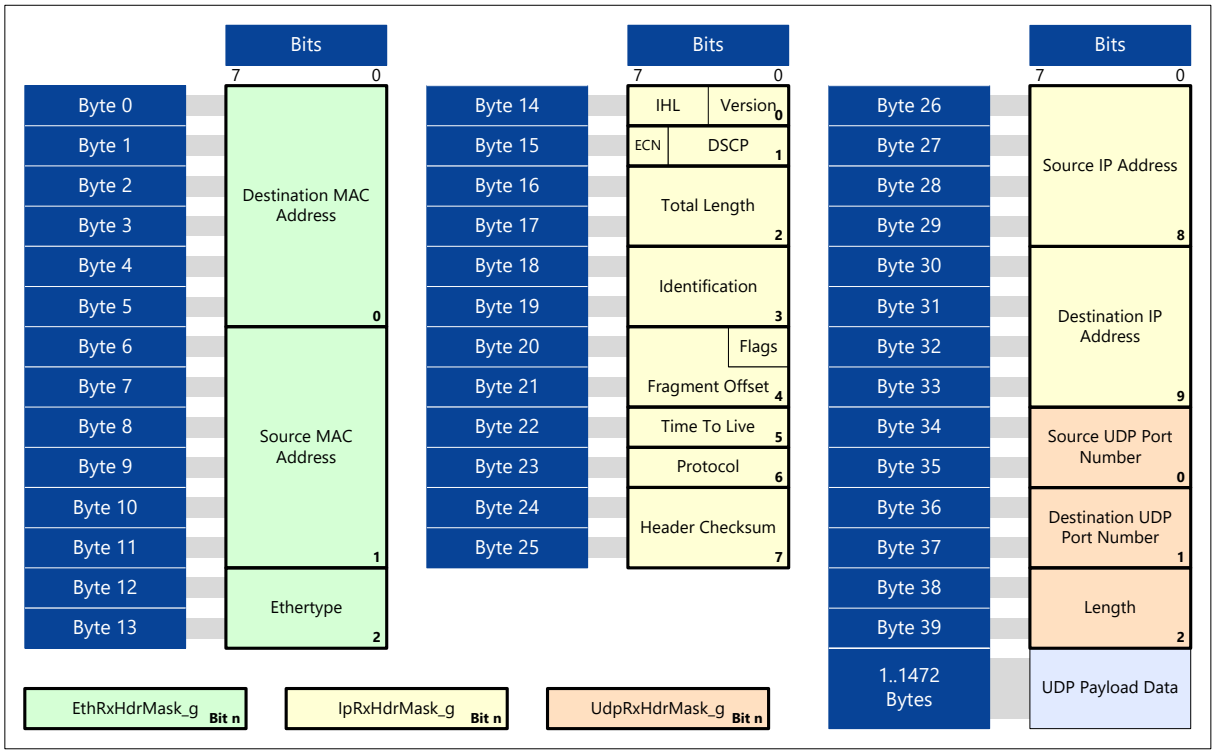


Figure 7: Receive Header Pass-Through Mode

Figure 7 shows the receive data byte sequence provided at the user receive data interface for a UDP packet with all supported header fields enabled for receive header pass-through mode (*UdpRxHdrMask_g* = “111”, *IpRxHdrMask_g* = “1111111111”, *EthRxHdrMask_g* = “111”).

If any of the receive header mask bits is set to zero, the corresponding receive data byte(s) are not inserted into the receive data byte sequence.

2.7 Raw Ethernet port

The raw Ethernet port can be used to send and receive non-UDP packets. It has the same FIFO interface as the UDP interfaces, but always a full Ethernet frame starting with the MAC addresses and ending with the last data byte is transferred. The CRC is generated and checked by the UDP core.

The raw Ethernet port can have the same or another MAC address as the UDP interfaces. The MAC address applied to port Hdr_RawEthMac is only used to filter incoming packets. The settings from the UDP interfaces have no influence to the raw Ethernet port.

There is also a Linux driver available that mounts the raw Ethernet port of the UDP core as a normal network adapter. This allows the Linux to communicate through the same Ethernet interface as UDP core for configuration and other purpose.

3 Functional Description

3.1 Architecture

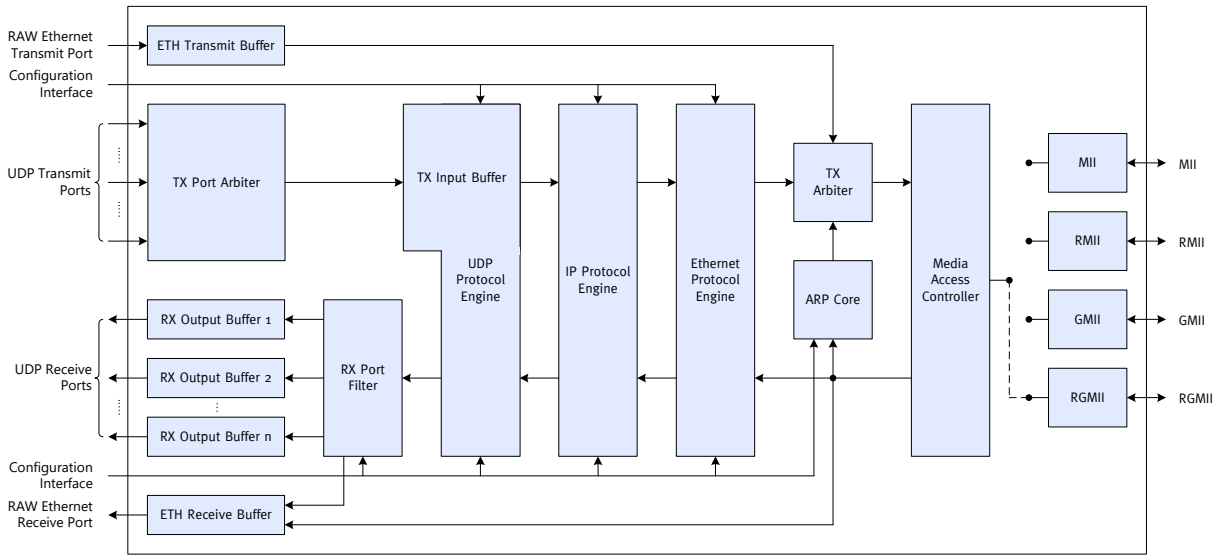


Figure 8: Core Architecture

Figure 8 shows a simplified block diagram of the UDP/IP/Ethernet IP core's internal architecture. Please note that only one type of MII interface is comprised, depending on the selected product options.

3.2 Functional Blocks Description

3.2.1 UDP TX Port Arbiter

This block provides one transmit data interface for each of the individual UDP ports. It arbitrates between requests on the transmit data interfaces, accepts UDP packet payload data on the currently active transmit data interface and forwards the data to the UDP protocol engine. The arbitration scheme can either be round robin or priority arbitration, depending on the configuration. The UDP port with index zero has the highest priority in case of priority arbitration.

3.2.2 UDP RX Output Buffer

This block provides one receive data buffer and an associated receive data interface for each of the individual UDP ports. It accepts received UDP packet payload data from the UDP protocol engine, stores the data in the appropriate receive data buffer and, upon request, forwards the data to the associated receive data interface.

3.2.3 UDP Protocol Engine

The UDP protocol engine handles the UDP protocol layer for incoming and outgoing traffic. It consists of two fully independent sub-blocks, namely the transmit and the receive path.

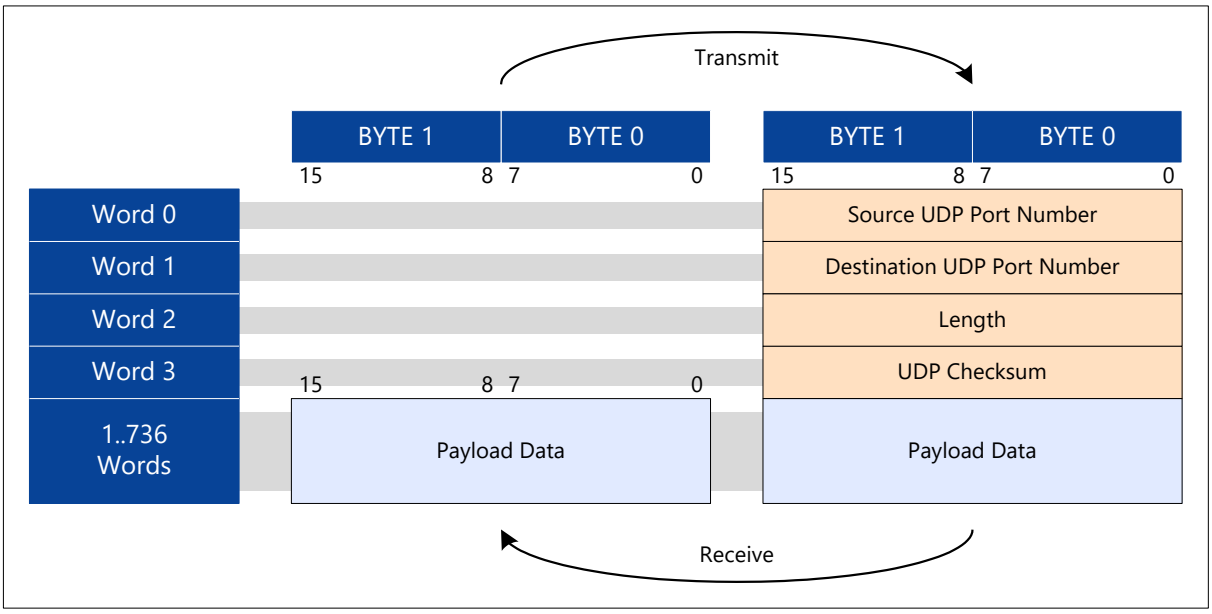


Figure 9: UDP Protocol Engine: UDP Header Handling

Figure 9 shows how the UDP header is prepended to the outgoing payload data (transmit path) and removed from the incoming UDP packets (receive path).

3.2.3.1 Transmit Path

The transmit path accepts UDP packet payload data from the Tx UDP port arbiter. It then prepends the UDP header and forwards the complete UDP packet to the IP protocol engine. The individual UDP header fields are populated as follows:

- **Source UDP Port Number:**
  If *UdpTxHdrMask_g(0)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is populated with the value of *Hdr_UdpTxSrc*.
- **Destination UDP Port Number:**
  If *UdpTxHdrMask_g(1)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is populated with the value of *Hdr_UdpTxDst*.
- **Length:**
  The value of this field is calculated internally.
- **UDP Checksum:**
  The value of this field is calculated internally.

3.2.3.2 Receive Path

The receive path accepts complete UDP packets from the IP protocol engine. It then checks the UDP packet for validity and forwards the payload data of valid packets to the appropriate receive buffer. The following fields are checked for validity and/or are embedded in the receive data stream:

- **Source UDP Port Number:**
  This field is not checked for validity. It is embedded in the receive data stream if *UdpRxHdrMask_g(0)* = ‘1’.
- **Destination UDP Port Number:**
  If *Cfg_CheckUdp* is set to '1', the value of this field and (bit-wise logical and) *Hdr_UdpRxPortMask* must match the value of (*Hdr_UdpRxPort and Hdr_UdpRxPortMask).* If *Cfg_CheckUdp* is set to '0', the value of this field is not checked. In either case, this field is used to determine the appropriate receive buffer to write the payload data to. This field is embedded in the receive data stream if *UdpRxHdrMask_g(1)* = ‘1’.
- **Length**
  This field is not checked for validity. It is embedded in the receive data stream if *UdpRxHdrMask_g(2)* = ‘1’.
- **UDP Checksum:**
  If *CalcCheckSum_g* is set to true, the checksum of the received UDP packet is re-calculated, and the re-calculated value must match the value of this field.

Packets identified as invalid are discarded, the payload data is thus not forwarded to the receive buffer.

3.2.4 IP Protocol Engine

The IP protocol engine handles the IPv4 protocol layer for incoming and outgoing traffic. It consists of two fully independent sub-blocks, namely the transmit and the receive path.

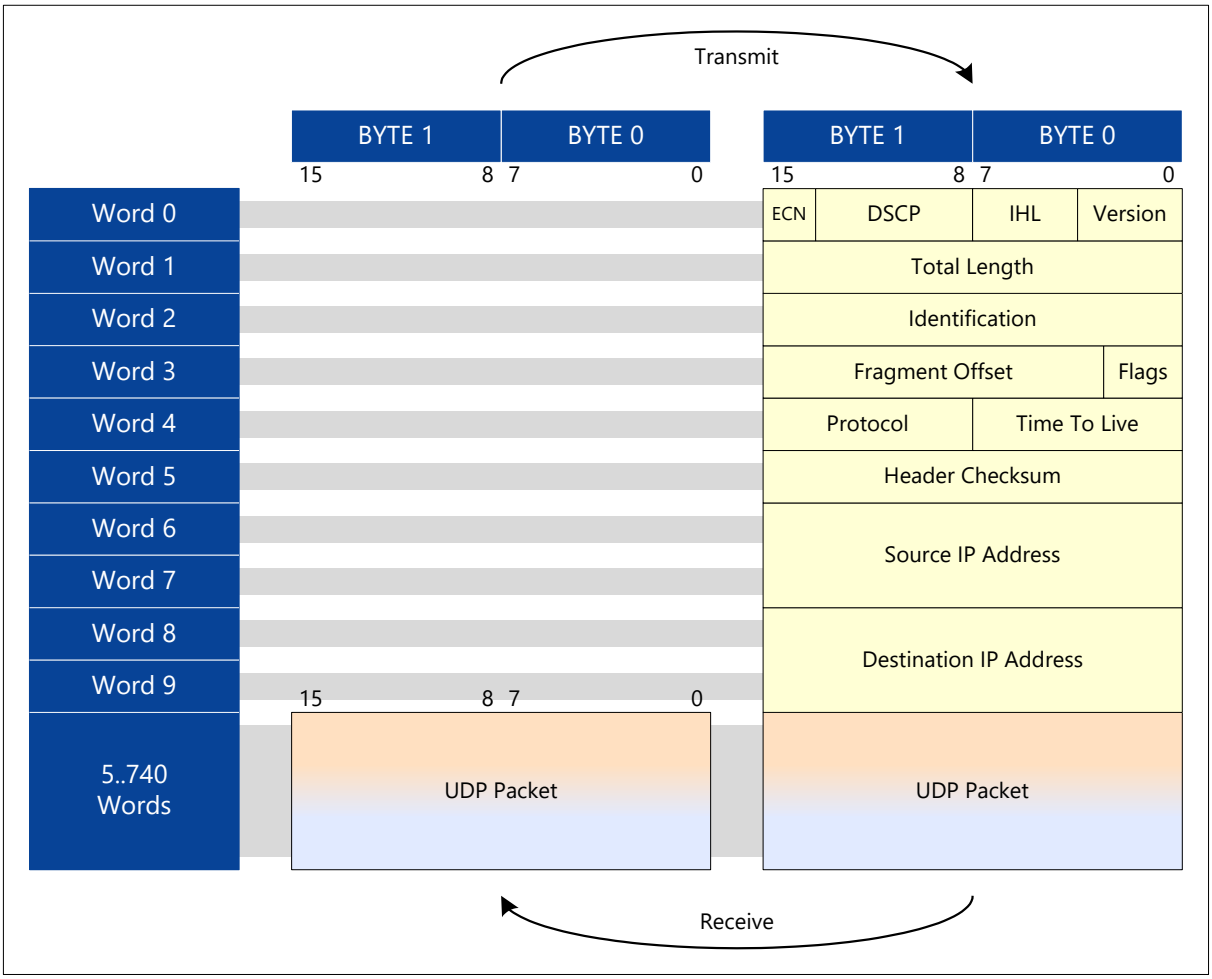


Figure 10: IP Protocol Engine: IP Header Handling

Figure 10 shows how the IP header is prepended to the outgoing UDP packets (transmit path) and removed from the incoming IP packets (receive path).

3.2.4.1 Transmit Path

The transmit path accepts UDP packets from the UDP protocol engine. It then prepends the IP header and forwards the complete IP packet to the Ethernet protocol engine. The individual IP header fields are populated as follows:

- **Version:**
  This field is set to 0x4 (IPv4).
- **Internet Header Length (IHL):**
  This field is set to 0x5 (no option fields).

- **Differentiated Services Code Point (DSCP):**
  If *IpTxHdrMask_g (0)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is set to the value of *Hdr_IpDscp(5:0)*.
- **Explicit Congestion Notification (ECN):**
  If *IpTxHdrMask_g (0)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is set to the value of *Hdr_IpDscp(7:6).*
- **Total Length:**
  The value of this field is calculated internally.
- **Identification:**
  If *IpTxHdrMask_g (1)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is set to the value of *Hdr_IpId*.
- **Flags:**
  If *IpTxHdrMask_g (2)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is set to the value of *Hdr_IpFlags*.
- **Time To Live (TTL):**
  If *IpTxHdrMask_g (3)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is set to the value of *Hdr_IpTtl*.
- **Protocol:**
  If *IpTxHdrMask_g (4)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is set to 0x11 (UDP).
- **Header Checksum:**
  The value of this field is calculated internally.
- **Source IP Address:**
  If *IpTxHdrMask_g (5)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is set to the value of *Hdr_IpSrc*.
- **Destination IP Address:**
  If *IpTxHdrMask_g (6)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is set to the value of *Hdr_IpDst*.

3.2.4.2 Receive Path

The receive path accepts complete IP packets from the Ethernet protocol engine. It then checks the IP packet for validity and forwards the payload data of valid packets to the UDP protocol engine. The following fields are checked for validity and/or are embedded in the receive data stream:

- **Version:**
  The value of this field must be 0x4 (IPv4). It is embedded in the receive data stream if *IpRxHdrMask_g(0)* = '1'.
- **Internet Header Length (IHL)**
  This field is not checked for validity. It is embedded in the receive data stream if *IpRxHdrMask_g(0)* = '1'.
- **Differentiated Services Code Point (DSCP):**
  This field is not checked for validity. It is embedded in the receive data stream if *IpRxHdrMask_g(1)* = '1'.

- **Explicit Congestion Notification (ECN):**
  This field is not checked for validity. It is embedded in the receive data stream if *IpRxHdrMask_g(1)* = ‘1’.
- **Total Length:**
  The total length of the IP packet is checked against this field’s value. It is embedded in the receive data stream if *IpRxHdrMask_g(2)* = ‘1’.
- **Identification:**
  This field is not checked for validity. It is embedded in the receive data stream if *IpRxHdrMask_g(3)* = ‘1’.
- **Flags:**
  This field is not checked for validity. It is embedded in the receive data stream if *IpRxHdrMask_g(4)* = ‘1’.
- **Time To Live (TTL):**
  This field is not checked for validity. It is embedded in the receive data stream if *IpRxHdrMask_g(5)* = ‘1’.
- **Protocol:**
  The value of this field must be 0x11 (UDP). It is embedded in the receive data stream if *IpRxHdrMask_g(6)* = ‘1’.
- **Header Checksum:**
  The header checksum of the received IP packet is re-calculated, and the re-calculated value must match the value of this field. The received value is embedded in the receive data stream if *IpRxHdrMask_g(7)* = ‘1’.
- **Source IP Address:**
  This field is not checked for validity. It is embedded in the receive data stream if *IpRxHdrMask_g(8)* = ‘1’.
- **Destination IP Address:**
  If *Cfg_CheckIp* is set to '1', the value of this field must match the value of *Hdr_IpSrc*. *Cfg_CheckIp* is set to '0', the value of this field is not checked. The received value is embedded in the receive data stream if *IpRxHdrMask_g(9)* = ‘1’.

Packets identified as invalid are discarded; the payload data is thus not forwarded to the UDP protocol engine.

3.2.5 Ethernet Protocol Engine

The Ethernet protocol engine handles the Ethernet protocol layer for incoming and outgoing traffic. It consists of two fully independent sub-blocks, namely the transmit and the receive path.

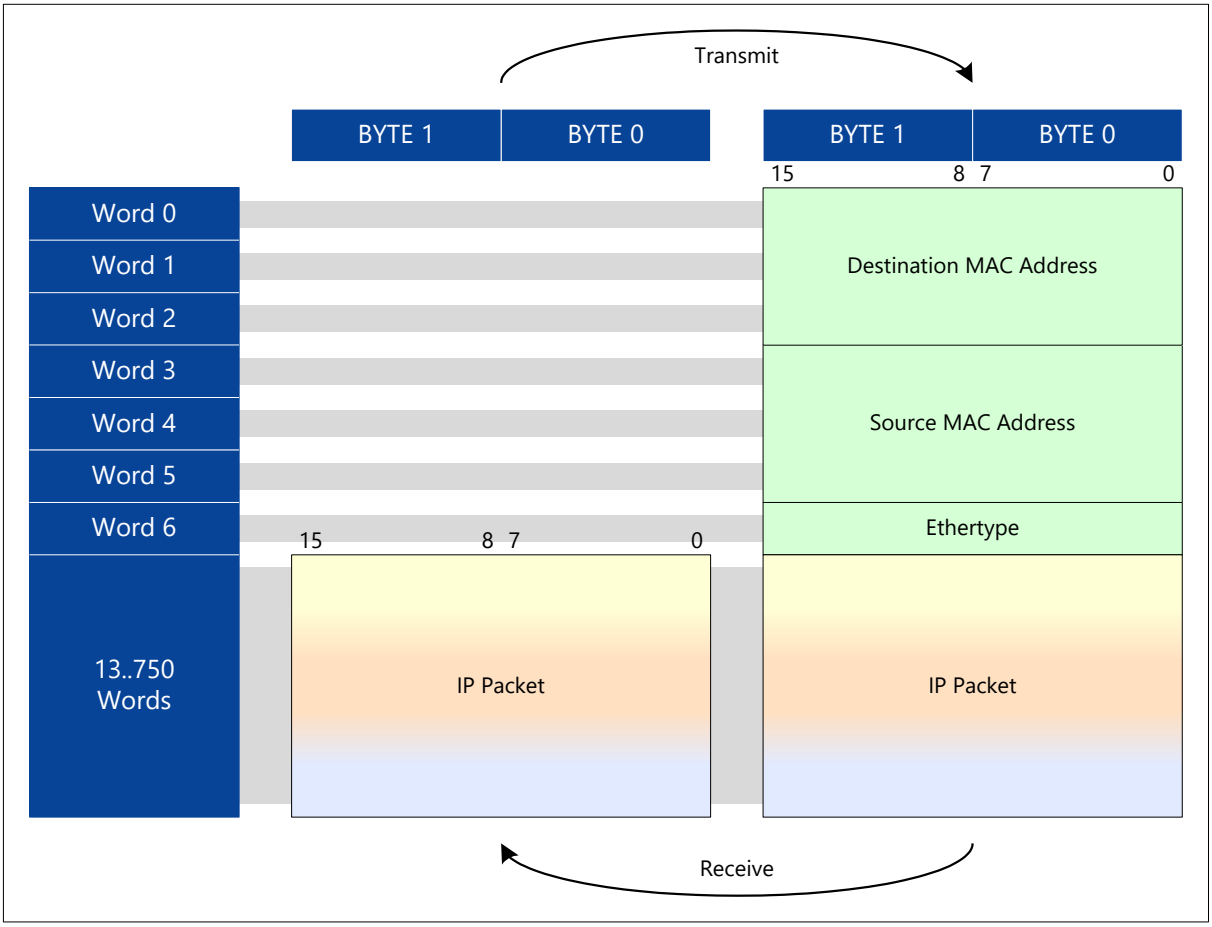


Figure 11: Ethernet Protocol Engine: Ethernet Header Handling

Figure 11 shows how the Ethernet header is prepended to the outgoing IP packets (transmit path) and removed from the incoming Ethernet packets (receive path).

3.2.5.1 Transmit Path

The transmit path accepts IP packets from the IP protocol engine. It then prepends the Ethernet header and forwards the complete Ethernet packet to the media access controller. The individual Ethernet header fields are populated as follows:

- **Destination MAC Address:**
  If *EthTxHdrMask_g (0)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is set to the value of *Hdr_MacDst*.
- **Source MAC Address:**
  If *EthTxHdrMask_g (1)* = '1', this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is set to the value of of *Hdr_MacSrc*.

- **Ethertype:**
  If *EthTxHdrMask_g (2)* = ‘1’, this field is populated with the corresponding value embedded in the transmit data stream. Otherwise this field is set to 0x0800 (IPv4).

3.2.5.2 Receive Path

The receive path accepts complete Ethernet packets from the media access controller. It then checks the Ethernet packet for validity and forwards the payload data of valid packets to the IP protocol engine. The following fields are checked for validity and/or are embedded in the receive data stream:

- **Destination MAC Address:**
  If *Cfg_CheckMac* is set to '1', the value of this field must match either the value of *Hdr_MacSrc* or the broadcast MAC address (0xFFFFFFFFFFFF). If *Cfg_CheckMac* is set to '0', the value of this field is not checked. It is embedded in the receive data stream if *EthRxHdrMask_g(0)* = ‘1’.
- **Source MAC Address:**
  This field is not checked for validity. It is embedded in the receive data stream if *EthRxHdrMask_g(0)* = ‘1’.
- **Ethertype:**
  The value of this field must be 0x0800 (IPv4). It is embedded in the receive data stream if *EthRxHdrMask_g(2)* = ‘1’.

Packets identified as invalid are discarded, the payload data is thus not forwarded to the IP protocol engine.

3.2.6 Media Access Controller

The media access controller handles the Ethernet framing for incoming and outgoing traffic. It consists of two fully independent sub-blocks, namely the transmit and the receive path.

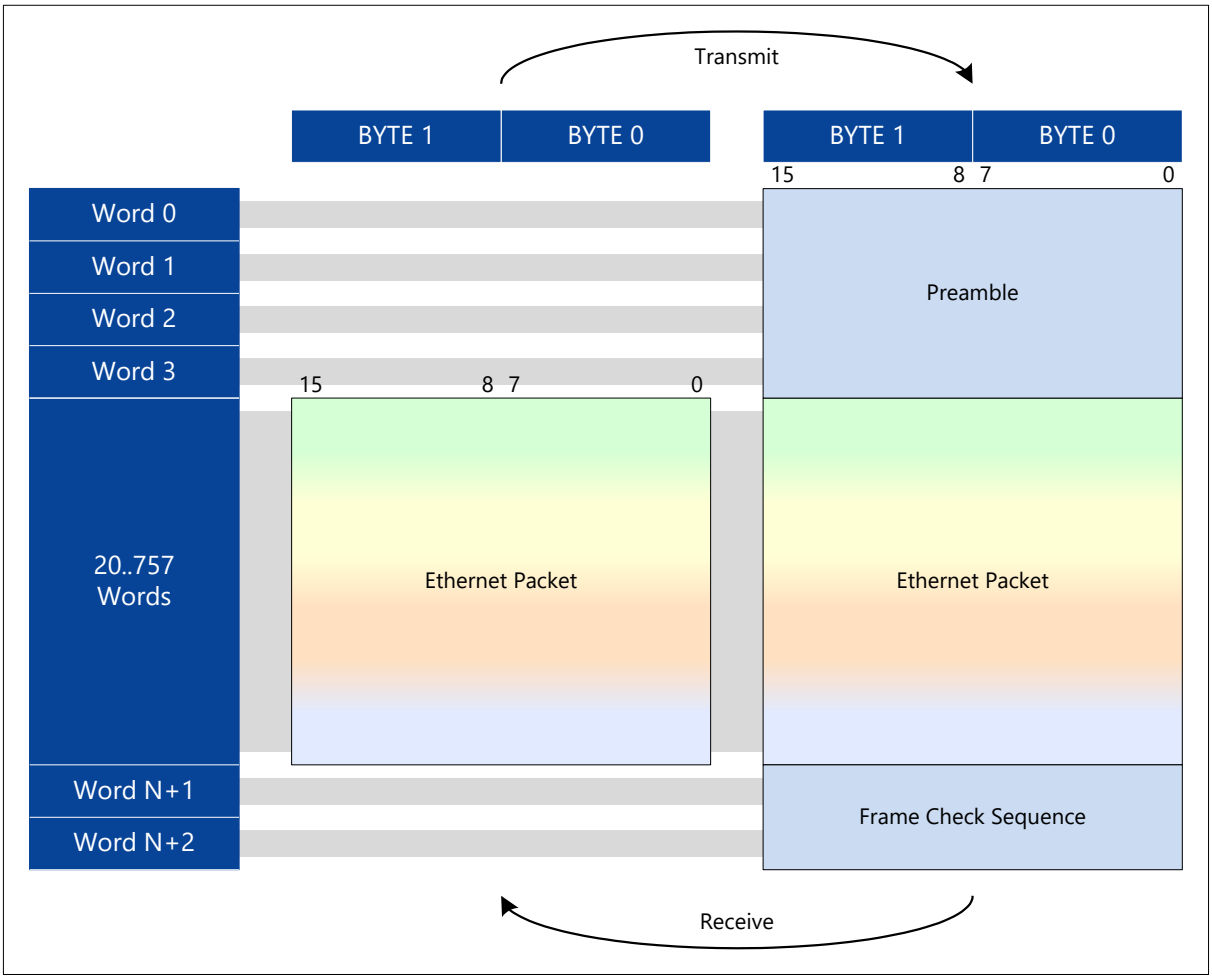


Figure 12: Media Access Controller: Ethernet Framing

Figure 12 shows how the Ethernet preamble and frame check sequence are prepended/appended to the outgoing Ethernet packets (transmit path) and removed from the incoming Ethernet frames (receive path).

3.2.6.1 Transmit Path

The transmit path accepts Ethernet packets from the Ethernet protocol engine. It then prepends the Ethernet preamble, appends the frame checksum and forwards the complete Ethernet frame to the media independent interface. The Ethernet preamble and frame check sequence are populated as follows:

- **Preamble:**
  The value of this field is set to the standard Ethernet preamble value.

- **Frame Check Sequence:**
  The value of this field is calculated internally.

3.2.6.2 Receive Path

The receive path accepts complete Ethernet packets from the media access controller. It then checks the Ethernet packet for validity and forwards the payload data of valid packets to the IP protocol engine. The following fields are checked for validity:

- **Frame Check Sequence:**
  The frame check sequence of the received Ethernet frame is re-calculated, and the re-calculated value must match the value of this field.

Ethernet frames identified as invalid are discarded; the payload data is thus not forwarded to the Ethernet protocol engine.

3.2.7 Media Independent Interface (MII, RMII, GMII, RGMII)

These blocks implement the different media independent interface protocols. Table 16 shows the supported wire speeds for each of the media independent interface protocols.

MII Type	1 Gbit/sec	100 Mbit/sec	10 Mbit/sec
MII	No	Yes (25 MHz clock)	Yes (2.5 MHz clock)
RMII	No	Yes (50 MHz clock)	Yes (50 MHz clock)
GMII	Yes (125 MHz clock)	Yes (25 MHz clock)	Yes (2.5 MHz clock)
RGMII	Yes (125 MHz clock)	Yes (25 MHz clock)	Yes (2.5 MHz clock)

Table 16: Media Independent Interface: Wire Speed Support

Please note that only one of the media independent interface protocols is supported at a time, depending on the selected product options. Only full-duplex operation is supported for all MII types.

3.2.8 ARP Core

The ARP core handles ARP requests by checking them for correctness and generates ARP replies to the requesting networking partner.

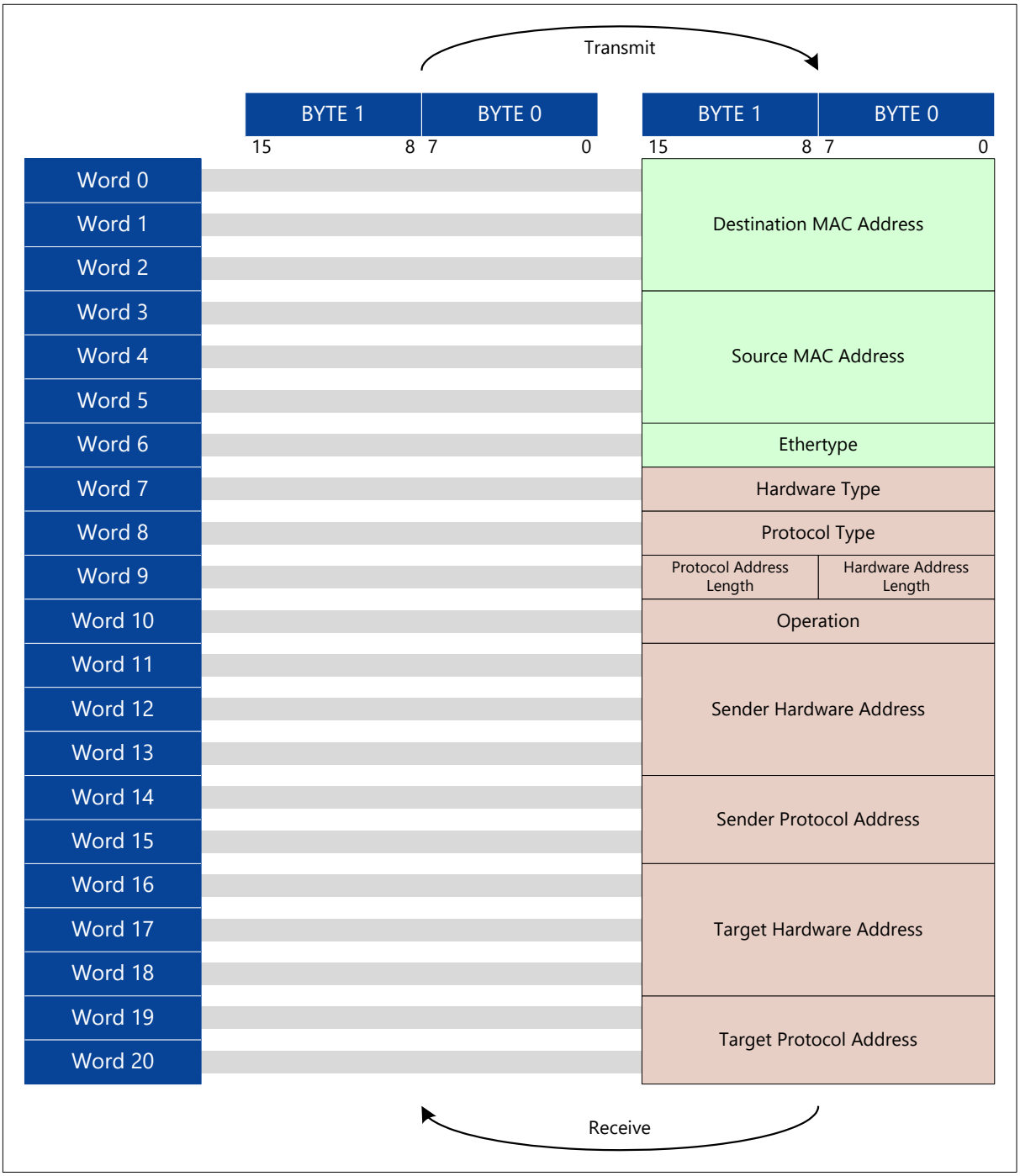


Figure 13: ARP Protocol Engine: ARP Packet Handling

Figure 13 shows how an ARP request packet is received and how, in response to an ARP request, an ARP reply packet is transmitted.

3.2.8.1 ARP Request Processing (Receive)

The ARP protocol engine accepts ARP request packets directly from the media access controller. It then checks the ARP request packet for validity and, if the ARP request packet has been identified as valid, kicks off the generation of an ARP reply packet. The following fields are checked for validity:

- **Ethertype**
  The value of this field must be 0x0806 (ARP).
- **Hardware Type**
  The value of this field must be 0x0001 (Ethernet).
- **Protocol Type**
  The value of this field must be 0x0800 (IPv4).
- **Hardware Address Length**
  The value of this field must be 0x06 (IEEE 802 MAC address length = 6 bytes).
- **Protocol Address Length**
  The value of this field must be 0x04 (IPv4 address length = 4 bytes).
- **Operation**
  The value of this field must be 0x0001 (ARP request).
- **Target Protocol Address**
  The value of this field must match the value of *Hdr_IpSrc*.

3.2.8.2 ARP Reply Generation (Transmit)

In case a valid ARP request packet has been received, the ARP protocol engine generates an according ARP reply packet and forwards it directly to the media access controller. The individual fields of the ARP reply packet are populated as follows:

- **Destination MAC Address:**
  The value of this field is set to the value of the *Sender Hardware Address* field of the received ARP request packet.
- **Source MAC Address:**
  The value of this field is set to the value of *Hdr_MacSrc*.
- **Ethertype:**
  The value of this field is always set to 0x0800 (IPv4).
- **Hardware Type**
  The value of this field is always set to 0x0001 (Ethernet).
- **Protocol Type**
  The value of this field is always set to 0x0800 (IPv4).
- **Hardware Address Length**
  The value of this field is always set to 0x06 (IEEE 802 MAC address length = 6 bytes).
- **Protocol Address Length**
  The value of this field is always set to 0x04 (IPv4 address length = 4 bytes).
- **Operation**
  The value of this field is always set to 0x0002 (ARP reply).
- **Sender Hardware Address**
  The value of this field is set to the value of *Hdr_MacSrc*.

- **Sender Protocol Address**
  The value of this field is set to the value of *Hdr_IpSrc*.
- **Target Hardware Address**
  The value of this field is set to the value of the *Sender Hardware Address* field of the received ARP request packet.
- **Target Protocol Address**
  The value of this field is set to the value of the *Sender Protocol Address* field of the received ARP request packet.

3.2.9 ETH Transmit Buffer

The raw Ethernet transmit buffer is used to store packets until they can be transferred by the MAC. On the raw Ethernet port always full Ethernet frames starting with the MAC addresses have to be sent. The CRC32 at the end is calculated by the MAC. If the transmit buffer is bypassed, the whole frame has to be sent at once and EthTx_Vld must be high during the whole frame. EthTx_Rdy will be high as soon as the raw Ethernet port is selected by the arbiter.

3.2.10 ETH Receive Buffer

The raw Ethernet receive buffer is used to store incoming raw Ethernet packets. When the receive buffer is used, faulty frames and are suppressed. Also frames assigned to one of the UDP interfaces are suppressed. When the receive buffer id bypassed these frames are not suppressed but marked with EthRx_Err to be discarded by the next block. If needed the RX filter can also suppress unicast frames addressed to other MAC addresses than specified in the Hdr_RawEthMac input.

3.2.11 TX Arbiter

The transmit arbiter arbitrates between the UDP traffic, the ARP core and the raw Ethernet port. It works in a round robin scheme.

4 Implementation Constraints

This section lists the physical constraints that are required for proper operation of the UDP/IP Ethernet IP Core. Detailed timing constraints can be found in the reference design's constraint files (.sdc, .ucf, .xdc).

4.1 Media Independent Interface (MII)

4.1.1 Location Constraints

4.1.1.1 Clock Inputs

- Mii_RxClk: Clock-capable input pin
- Mii_TxClk: Clock-capable input pin
- Clk: Clock-capable input pin or internal global clock

4.1.2 Timing Constraints

4.1.2.1 Clock Periods

- User clock: Clk period 40 ns
- MII receive clock: Mii_RxClk period 40 ns
- MII transmit clock: Mii_TxClk period 40 ns

4.1.2.2 External Interface Timing

- Clock-to-out
  - From Mii_TxClk to Mii_Tx{D,En}: Min. 0 ns, max 30 ns
- Input setup time
  - Mii_Rx{D,Dv,Er} 10 ns before rising edge of Mii_RxClk
- Input hold time
  - Mii_Rx{D,Dv,Er} 10 ns after rising edge of Mii_RxClk

Please note that the external interface timing listed above is sufficient for most MII interfaces. However, it is the user's responsibility to check the datasheet of the actual PHY for validating external timing constraints.

4.1.2.3 Internal Timing

- Clock domain crossings
  - Maximum delay 40 ns from Clk to Mii_RxClk
  - Maximum delay 40 ns from Mii_TxClk to Clk
  - Maximum delay 40 ns from Clk to Mii_TxClk
  - Maximum delay 40 ns from Mii_TxClk to Clk

4.2 Reduced Gigabit Media Independent Interface (RGMII)

4.2.1 Location Constraints

4.2.1.1 Clock Inputs

- Rgmii_RxClk: Clock-capable input pin
- Clk: Clock-capable input pin or internal global clock

4.2.2 Physical Constraints

4.2.2.1 RGMII Transmit Ports

- Fast slew rate for Rgmii_Tx{Clk,Ctl,D)

4.2.3 Timing Constraints

4.2.3.1 Clock Periods

- User clock: Clk period 8 ns
- MII receive clock: Rgmii_RxClk period 8 ns

4.2.3.2 External Interface Timing

- Clock-to-out
  - Max. skew on Rgmii_Tx{Ctl,D} of ± 500 ps with respect to Rgmii_TxClk
- Input setup time
  - Mii_Rx{D,Dv,Er} 1 ns before rising/falling edge of Rgmii_RxClk
- Input hold time
  - Mii_Rx{D,Dv,Er} 2.6 ns after rising/falling edge of Rgmii_RxClk

Please note that the external interface timing listed above is suited for RGMII 2.0 interfaces with all delays implemented in the PHY. This means that the FPGA transmits Rgmii_TxD/Rgmii_TxCtl edge-aligned to Rgmii_TxClk and expects Rgmii_RxD/Rgmii_RxCtl to arrive center-aligned to Rgmii_RxClk at the FPGA (i.e. Rgmii_RxClk is delayed by 90° at the PHY output). However, it is the user's responsibility to check the datasheet of the actual PHY for validating external timing constraints.

4.2.3.3 Internal Timing

- Clock domain crossings
  - Maximum delay 8 ns from Clk to Rgmii_RxClk
  - Maximum delay 8 ns from Rgmii_RxClk to Clk

5 AXI Wrapper

There is an AXI wrapper for the Vivado IPI and Intel QSYS components. This AXI wrapper provides AXI-Stream ports for the UDP interfaces and an AXI register bank for configuration and the raw Ethernet port. The AXI wrapper supports up to 8 UDP interfaces.

5.1 Address Map

Address	Register ID	Description
0x000	VERSION	Core version
0x004	SYSTEM_CFG	System configuration
0x008	TIMEOUT_CFG	Timeout configuration
0x020	SRCMAC_L	MAC address used for the TX source field and to filter the own MAC address (lower 32 bit)
0x024	SRCMAC_H	MAC address used for the TX source field and to filter the own MAC address (higher 16 bit)
0x028	DESTMAC_0_L	MAC address used for the TX destination field for UDP interface 0 (lower 32 bit)
0x02C	DESTMAC_0_H	MAC address used for the TX destination field for UDP interface 0 (higher 16 bit)
0x030	RAWMAC_L	MAC address used to filter the own MAC address on the raw Ethernet port (lower 32 bit)
0x034	RAWMAC_H	MAC address used to filter the own MAC address on the raw Ethernet port (higher 16 bit)
0x040	IP_SRC	Source IP address (global)
0x044	IP_DEST_0	Destination IP address for UDP interface 0
0x048	IP_CFG	IP Configuration
0x060	UDP_TX_PORT_0	UDP transmit ports for UDP interface 0
0x064	UDP_RX_PORT_0	UDP receive ports for UDP interface 0
0x068	UDP_MTU_0	Maximum transmit unit for UDP interface 0

Address	Register ID	Description
0x06C	UDP_TX_PORT_1	UDP transmit ports for UDP interface 1
0x070	UDP_RX_PORT_1	UDP receive ports for UDP interface 1
0x074	UDP_MTU_1	Maximum transmit unit for UDP interface 1
0x078	UDP_TX_PORT_2	UDP transmit ports for UDP interface 2
0x07C	UDP_RX_PORT_2	UDP receive ports for UDP interface 2
0x080	UDP_MTU_2	Maximum transmit unit for UDP interface 2
0x084	UDP_TX_PORT_3	UDP transmit ports for UDP interface 3
0x088	UDP_RX_PORT_3	UDP receive ports for UDP interface 3
0x08C	UDP_MTU_3	Maximum transmit unit for UDP interface 3
0x090	UDP_TX_PORT_4	UDP transmit ports for UDP interface 4
0x094	UDP_RX_PORT_4	UDP receive ports for UDP interface 4
0x098	UDP_MTU_4	Maximum transmit unit for UDP interface 4
0x09C	UDP_TX_PORT_5	UDP transmit ports for UDP interface 5
0x0A0	UDP_RX_PORT_5	UDP receive ports for UDP interface 5
0x0A4	UDP_MTU_5	Maximum transmit unit for UDP interface 5
0x0A8	UDP_TX_PORT_6	UDP transmit ports for UDP interface 6
0x0AC	UDP_RX_PORT_6	UDP receive ports for UDP interface 6
0x0B0	UDP_MTU_6	Maximum transmit unit for UDP interface 6
0x0B4	UDP_TX_PORT_7	UDP transmit ports for UDP interface 7
0x0B8	UDP_RX_PORT_7	UDP receive ports for UDP interface 7
0x0BC	UDP_MTU_7	Maximum transmit unit for UDP interface 7
0x0E0	RAWETH_DATA	Raw Ethernet transmit and receive data
0x0E4	RAWETH_STAT	Raw Ethernet status
0x100	DESTMAC_1_L	Destination MAC address for UDP interface 1
0x104	DESTMAC_1_H	Destination MAC address for UDP interface 1
0x108	IP_DEST_1	Destination IP address for UDP interface 1
0x10C	DESTMAC_2_L	Destination MAC address for UDP interface 2
0x110	DESTMAC_2_H	Destination MAC address for UDP interface 2
0x114	IP_DEST_2	Destination IP address for UDP interface 2

Address	Register ID	Description
0x118	DESTMAC_3_L	Destination MAC address for UDP interface 3
0x11C	DESTMAC_3_H	Destination MAC address for UDP interface 3
0x120	IP_DEST_3	Destination IP address for UDP interface 3
0x124	DESTMAC_4_L	Destination MAC address for UDP interface 4
0x128	DESTMAC_4_H	Destination MAC address for UDP interface 4
0x12C	IP_DEST_4	Destination IP address for UDP interface 4
0x130	DESTMAC_5_L	Destination MAC address for UDP interface 5
0x134	DESTMAC_5_H	Destination MAC address for UDP interface 5
0x138	IP_DEST_5	Destination IP address for UDP interface 5
0x13C	DESTMAC_6_L	Destination MAC address for UDP interface 6
0x140	DESTMAC_6_H	Destination MAC address for UDP interface 6
0x144	IP_DEST_6	Destination IP address for UDP interface 6
0x148	DESTMAC_7_L	Destination MAC address for UDP interface 7
0x14C	DESTMAC_7_H	Destination MAC address for UDP interface 7
0x150	IP_DEST_7	Destination IP address for UDP interface 7

Table 17: Register Bank Address Map

5.2 Register Description

5.2.1 Version

This register contains the build version of the FPGA firmware.

Field	Position	Size	Type	Reset	Description
VERSION	31:0	32	R	Version_c	Build Version

Table 18: Version register

5.2.2 SYSTEM_CFG

Field	Position	Size	Type	Reset	Description
TXEN	0	1	R/W	0	Transmit enable
RXEN	1	1	R/W	0	Receive enable
CHK_RAWMAC	8	1	R/W	0	Filter incoming unicast frames to match the MAC address configured in RAWMAC register
CHK_MAC	9	1	R/W	0	Filter incoming unicast UDP frames to match the MAC address configured in SRCMAC register
CHK_IP	10	1	R/W	0	Filter incoming UDP frames to match the IP address configured in IP_SRC register
CHK_UDP	11	1	R/W	0	Filter incoming UDP frames to match the UDP port configured in UDP_RX_PORT[x] registers
TX_INT_EN	16	1	R/W	0	Enable the raw Ethernet TX interrupt. When enabled a high pulse is generated on IntTx when the TX buffer becomes empty.
RX_INT_EN	17	1	R/W	0	Enable the raw Ethernet RX data interrupt. When enabled IntRx is set high when RX data is available.

Table 19: System configuration register

5.2.3 TIMEOUT_CFG

Configures the timeout value for the streaming mode. The actual width is depending on the generic TimeoutWidth_g.

Field	Position	Size	Type	Reset	Description
TIMEOUT	31:0	32	R/W	0	Timeout value in clock cycles

Table 20: Timeout configuration register

5.2.4 SRCMAC_L

Configures the source address of the Ethernet header in outgoing UDP frames and to filter incoming UDP frames.

Field	Position	Size	Type	Reset	Description
SRCMAC_L	31:0	32	R/W	0	Source MAC address, lower 32 bit

Table 21: SRCMAC_L register

5.2.5 SRCMAC_H

Configures the source address of the Ethernet header in outgoing UDP frames and to filter incoming UDP frames.

Field	Position	Size	Type	Reset	Description
SRCMAC_H	15:0	16	R/W	0	Source MAC address, higher 16 bit

Table 22: SRCMAC_H register

5.2.6 DESTMAC_[x]_L

Configures the destination address of the Ethernet header for the UDP interface x. Only used for transmitting UDP frames.

Field	Position	Size	Type	Reset	Description
DESTMAC_L	31:0	32	R/W	0	Destination MAC address, lower 32 bit

Table 23: DESTMAC_L register

5.2.7 DESTMAC_[x]_H

Configures the destination address of the Ethernet header for the UDP interface x. Only used for transmitting UDP frames.

Field	Position	Size	Type	Reset	Description
DESTMAC_H	15:0	16	R/W	0	Destination MAC address, higher 16 bit

Table 24: DESTMAC_H register

5.2.8 RAWMAC_L

Used to filter incoming raw Ethernet frames.

Field	Position	Size	Type	Reset	Description
RAWMAC_L	31:0	32	R/W	0	MAC address to filter the raw Ethernet port, lower 32 bit

Table 25: RAWMAC_L register

5.2.9 RAWMAC_H

Used to filter incoming raw Ethernet frames.

Field	Position	Size	Type	Reset	Description
RAWMAC_H	15:0	16	R/W	0	MAC address to filter the raw Ethernet port, higher 16 bit

Table 26: RAWMAC_H register

5.2.10 IP_SRC

Configures the source address of the IP header in outgoing UDP frames and to filter incoming UDP frames.

Field	Position	Size	Type	Reset	Description
IP_SRC	31:0	32	R/W	0	Source IP address

Table 27: IP_SRC register

5.2.11 IP_DEST_[x]

Configures the destination address of the IP header for the UDP interface x. Only used for transmitting UDP frames.

Field	Position	Size	Type	Reset	Description
IP_DEST	31:0	32	R/W	0	Destination IP address

Table 28: IP_DEST register

5.2.12 IP_CFG

Configures the fields of the IP header. Only used for transmitting UDP frames.

Field	Position	Size	Type	Reset	Description
IP_ID	15:0	16	R/W	0	ID filed of the IP header
IP_DSCP	23:16	8	R/W	0	DSCP filed of the IP header
IP_TTL	31:24	8	R/W	0	TTL filed of the IP header

Table 29: IP_CFG register

5.2.13 UDP_TX_PORT_[x]

Configures the UDP ports of the UDP header for transmission. There is a UDP_TX_PORT for each UDP interface.

Field	Position	Size	Type	Reset	Description
SRC	15:0	16	R/W	0	Source UDP port
DEST	31:16	16	R/W	0	Destination UDP port

Table 30: UDP_TX_PORT register

5.2.14 UDP_RX_PORT_[x]

Configures the UDP ports reception of UDP frames. There is a UDP_RX_PORT for each UDP interface.

Field	Position	Size	Type	Reset	Description
RX_PORT	15:0	16	R/W	0	receive UDP port
RX_MASK	31:16	16	R/W	0	receive UDP port mask

Table 31: UDP_RX_PORT register

5.2.15 UDP_MTU_[x]

Configures the maximum transfer unit used for the transmission of UDP frames in streaming mode. There is a UDP_MTU for each UDP interface.

Field	Position	Size	Type	Reset	Description
MTU	13:0	14	R/W	0	Maximum transfer unit

Table 32: UDP_MTU register

5.2.16 RAWETH_DATA

Field	Position	Size	Type	Reset	Description
TX_DATA	23:0	24	W	-	TX data (3 bytes), the byte in the lowest bits is transferred first.
RX_DATA	23:0	24	R	-	RX data (3 bytes), the first received byte is stored in the lowest bits.
TX_CNT	25:24	2	W	-	TX byte count: indicates the number of valid bytes written to the TX data field
RX_CNT	25:24	2	R	0	RX byte count: indicates the number of valid bytes in the RX data field
TX_LAST	29:27	3	W	-	TX last mask. A ‘1’ indicates that the according byte is the last byte of a frame.
RX_LAST	29:27	3	R	0	RX last mask. A ‘1’ indicates that the according byte is the last byte of a frame.
TX_ERR	30	1	W	-	TX error flag. Writing a ‘1’ discards the current frame. The error flag must always be used together with a “last” flag.
RX_ERR	30	1	R	0	RX error flag. A ‘1’ indicated that a receive error occurred in the current frame.

Table 33: RAWETH_DATA register

5.2.17 RAWETH_STAT

Field	Position	Size	Type	Reset	Description
RX_VLD	0	1	R	-	Indicates that RX data is available on the raw Ethernet port.
TX_RDY	1	1	R	-	Indicates that a free TX buffer is available on the raw Ethernet port.

Table 34: RAWETH status register

6 Reference Design

6.1 Interface Description

6.1.1 Interface Generics

Name	Type	Range	Description
Simulation_g	boolean	True / false	Simulation flag True: Puts the reference design in simulation mode (traffic led and blinking led counters count to 2^4 and 2^6, respectively). False: Puts the reference design in synthesis mode (traffic led and blinking led counters count to 2^{22} and 2^{24}, respectively).

Table 35: Reference Design Interface Generics

6.1.2 Interface Ports

6.1.2.1 Ethernet Physical Interface Ports

Please see section 0 for a detailed description of the Ethernet physical interface ports.

6.1.2.2 Local Interface Ports

Name	Width	Direction	Description
Clk	1	Input	Global clock 125 MHz for 1 Gbit/sec, 25 MHz for 100 Mbit/sec and 2.5 MHz for 10 Mbit/sec operation.
RstIn_N	1	Input	Global reset Active low, asynchronous
Eth_Rst_N	1	Output	Ethernet PHY reset Active low, asynchronous

Name	Width	Direction	Description
Led_N	2	Output	Status LEDs, active low Led_N(0): Traffic LED Pulses for ~1/8 sec when a valid UDP packet is received Led_N(1): Clock activity LED: Blinks with ~1/2 sec period if the global clock is running and the global reset is deasserted

Table 36: Reference Design Local Interface Ports

6.2 Functional Description

The reference design loops back UDP frames sent to IP address 16.0.0.1 and UDP port 50100. A small state machine exchanges source and destination MAC addresses, IP addresses and UDP ports. LED_0 will blink when UDP frames are received on the configured port.

The raw Ethernet port is also enabled in the reference design. The RX port is directly connected to the TX port, therefore all packets received on the raw Ethernet port are sent back without any change. LED_1 will blink when frames are received on the raw Ethernet port.

The UDP loopback can be tested with the example host software, see section 6.4 for details. The raw Ethernet loopback can be tested with an Ethernet sniffer software like Wireshark. All packets not routed to the UDP port will appear twice with a very short delay.

6.2.1 Network Setup

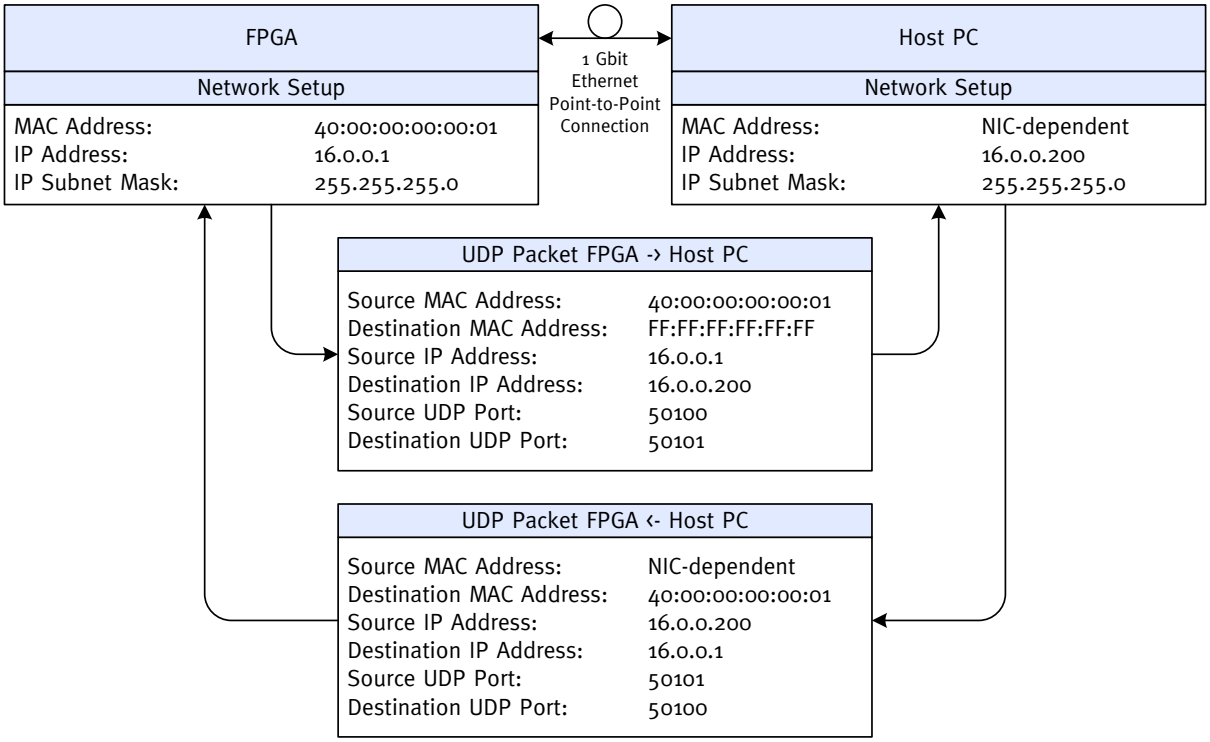


Figure 14: Reference Design Network Setup

Figure 14 shows the network setup and the addressing schemes that have to be used for UDP communication with the reference design.

6.2.2 Architecture

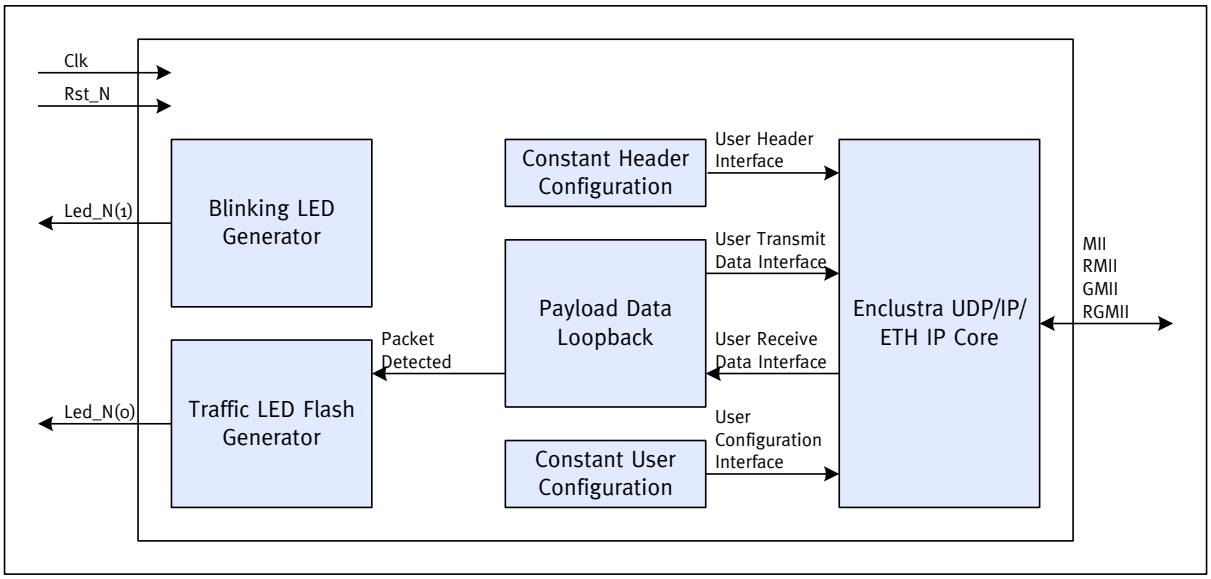


Figure 15: Reference Design Architecture

Figure 15 shows a simplified block diagram of the reference design's internal architecture. Please note that only one type of MII interface is comprised, depending on the selected product options.

6.2.3 Functional Blocks Description

6.2.3.1 Enclustra UDP/IP/Ethernet IP Core

This is an instance of Enclustra's UDP/IP/Ethernet IP core.

6.2.3.2 Constant Header Configuration

This block is a placeholder for the constant header configuration values used in the VHDL code.

6.2.3.3 Constant User Configuration

This block is a placeholder for the constant user configuration values used in the VHDL code.

6.2.3.4 Payload Data Loopback

This block loops back received header pass-through and payload data to the user transmit data interface. Additionally, it generates a pulse to the traffic LED flash generator for each received packet.

6.2.3.5 Blinking LED Generator

This block generates a 50% duty cycle on/off signal with a period of ~1/2 sec.

6.2.3.6 Traffic LED Generator

This block generates pulse with ~1/8 sec duration for each packet detected pulse from the payload data loopback block.

6.3 Synthesis, Place & Route

6.3.1 Intel® Quartus®

1. Start the Intel® Quartus® software
2. Add the license file ip\hdl\lic\en_udp_ip_eth_<mii>_alt_<arch>_<lic>.dat to the Quartus® license file search path (optional, depending on product options)
3. Open the Quartus® project file \rd\en_udp_ip_eth_<mii>_alt_<arch>\quartus\en_udp_ip_eth_<mii>_alt_<arch>_rd.qpf
4. Click the "Start Compilation" button

6.3.2 Xilinx® Vivado™

1. Start the Xilinx® Vivado™ software
2. Change to the Vivado™ project directory by typing
   cd <path to the Vivado™ project directory> (e.g. \rd\en_udp_ip_eth_<mii>_xil_<arch>\vivado into the TCL command window

3. Create the Vivado™ project “project_1” by typing
   source create_project_1.tcl
   in the TCL command window
4. Click the "Run Implementation" button

6.4 Running on Hardware

1. Download the bitstream generate in the previous section to the FPGA.
2. Connect an Ethernet cable between a Windows PC and the FPGA board. Make sure the DIP switches are correctly for 3.3V IO voltage and Gigabit Ethernet. Refer to the according user manual for details.
3. Configure the IP address of your PC to 16.0.0.200 or change the IP address in the reference design and recompile the design. Enable the “Jumbo frame” option of your network adapter if you want to test frames bigger as 1500 bytes.
4. Start the example host application from the folder below:
   \rd\host_sw\en_udp_reference_app\bin\Release
   Alternatively any other Ethernet packet generator can be used to send UDP packets to IP address 16.0.0.1 and UDP port 50100.
5. Change the FPGA IP-Address if you modified the reference design and click on “Open”.
6. Configure Packet Size and Packet Count and start the test with the “Execute Loopback Test” button. Please note that sometimes the PC’s are too slow to generate and receive all packets, especially with packet sizes around 1000 bytes. Increasing the packet size to 4000-8000 bytes or enabling the “Check packet content” option often solves this problem.
7. While UDP frames are received LED_0 will be switched on, when other Ethernet Frames are received LED_1. The traffic can also be monitored using a tool like Wireshark.

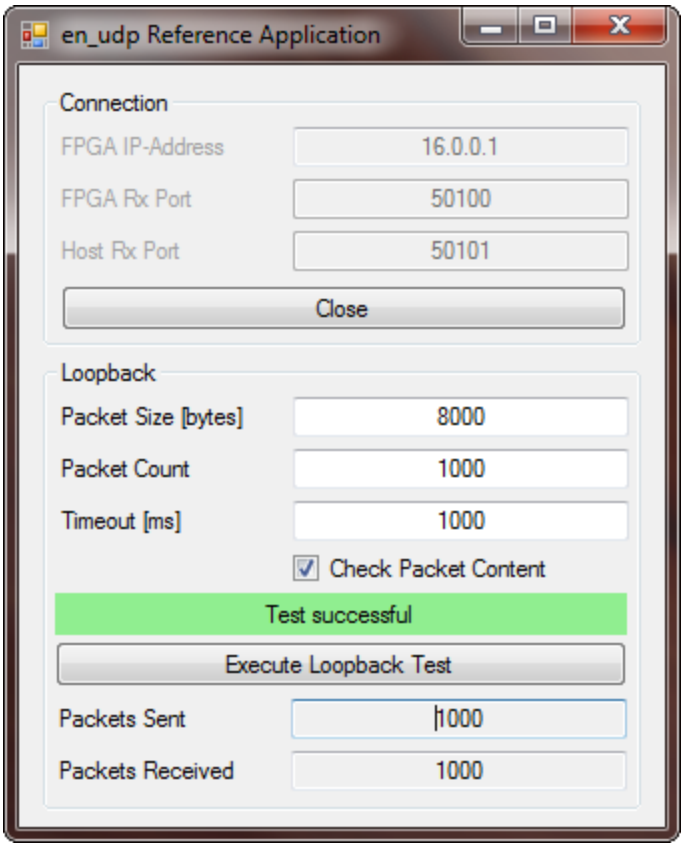
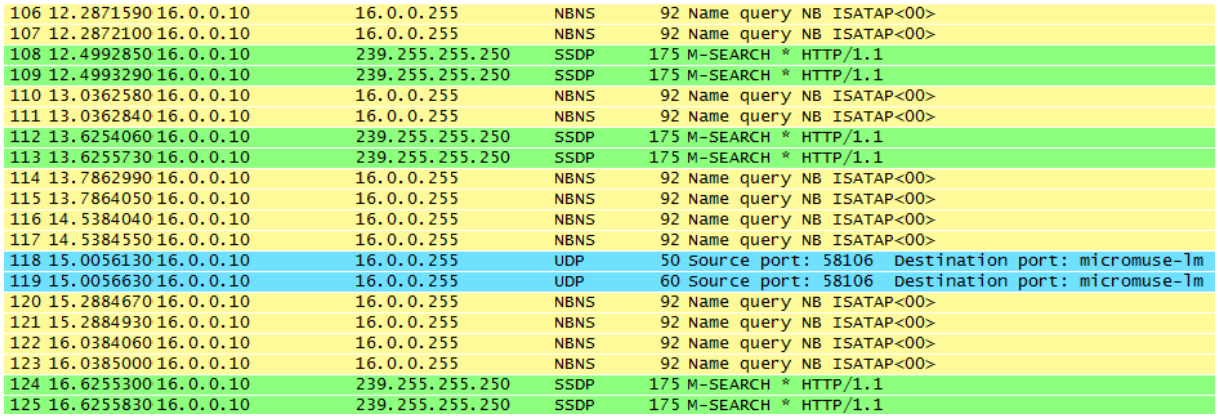


Figure 16: Host application

5	4.68468200	16.0.0.10	16.0.0.1	UDP	1042 Source port: 50101 Destination port: 50100
6	4.68474800	16.0.0.1	16.0.0.10	UDP	1042 Source port: 50100 Destination port: 50101
7	4.68640500	16.0.0.10	16.0.0.1	UDP	1042 Source port: 50101 Destination port: 50100
8	4.68646600	16.0.0.1	16.0.0.10	UDP	1042 Source port: 50100 Destination port: 50101
9	4.68651600	16.0.0.10	16.0.0.1	UDP	1042 Source port: 50101 Destination port: 50100
10	4.68658300	16.0.0.1	16.0.0.10	UDP	1042 Source port: 50100 Destination port: 50101
11	4.68664900	16.0.0.10	16.0.0.1	UDP	1042 Source port: 50101 Destination port: 50100
12	4.68671700	16.0.0.1	16.0.0.10	UDP	1042 Source port: 50100 Destination port: 50101
13	4.68675800	16.0.0.10	16.0.0.1	UDP	1042 Source port: 50101 Destination port: 50100
14	4.68682700	16.0.0.1	16.0.0.10	UDP	1042 Source port: 50100 Destination port: 50101
15	4.68688800	16.0.0.10	16.0.0.1	UDP	1042 Source port: 50101 Destination port: 50100
16	4.68695400	16.0.0.1	16.0.0.10	UDP	1042 Source port: 50100 Destination port: 50101
17	4.68700200	16.0.0.10	16.0.0.1	UDP	1042 Source port: 50101 Destination port: 50100
18	4.68707200	16.0.0.1	16.0.0.10	UDP	1042 Source port: 50100 Destination port: 50101
19	4.68713900	16.0.0.10	16.0.0.1	UDP	1042 Source port: 50101 Destination port: 50100
20	4.68721000	16.0.0.1	16.0.0.10	UDP	1042 Source port: 50100 Destination port: 50101
21	4.68724300	16.0.0.10	16.0.0.1	UDP	1042 Source port: 50101 Destination port: 50100
22	4.68731100	16.0.0.1	16.0.0.10	UDP	1042 Source port: 50100 Destination port: 50101

Figure 17: Wireshark capture of UDP packets



106	12.2871590	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
107	12.2872100	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
108	12.4992850	16.0.0.10	239.255.255.250	SSDP	175 M-SEARCH * HTTP/1.1
109	12.4993290	16.0.0.10	239.255.255.250	SSDP	175 M-SEARCH * HTTP/1.1
110	13.0362580	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
111	13.0362840	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
112	13.6254060	16.0.0.10	239.255.255.250	SSDP	175 M-SEARCH * HTTP/1.1
113	13.6255730	16.0.0.10	239.255.255.250	SSDP	175 M-SEARCH * HTTP/1.1
114	13.7862990	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
115	13.7864050	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
116	14.5384040	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
117	14.5384550	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
118	15.0056130	16.0.0.10	16.0.0.255	UDP	50 Source port: 58106 Destination port: micromuse-lm
119	15.0056630	16.0.0.10	16.0.0.255	UDP	60 Source port: 58106 Destination port: micromuse-lm
120	15.2884670	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
121	15.2884930	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
122	16.0384060	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
123	16.0385000	16.0.0.10	16.0.0.255	NBNS	92 Name query NB ISATAP<00>
124	16.6255300	16.0.0.10	239.255.255.250	SSDP	175 M-SEARCH * HTTP/1.1
125	16.6255830	16.0.0.10	239.255.255.250	SSDP	175 M-SEARCH * HTTP/1.1

Figure 18: Wireshark capture of duplicated non-UDP packets

6.5 Functional Simulation

6.5.1 Test Bench

The very simple test bench instantiates the following entities:

- Reference design top-level as device under test (DUT)
- UDP/IP/Ethernet IP core top-level as host PC emulator
- PHY models to emulate the physical data transmission

The stimuli application process uses the UDP/IP/Ethernet IP core instance to send a short UDP packet to the reference design instance. The reference design instance then sends back the received payload data as another UDP packet. The received payload data can visually be checked for correctness at the UDP/IP/Ethernet IP core instance's user receive data interface.

6.5.2 Running the Simulation

1. Start the Mentor Graphics® ModelSim® software
2. Change the working directory to the simulation folder (e.g. \rd\en_udp_ip_eth_<mii>_<vnd>_<arch>\modelsim)
3. Map or generate required vendor-specific libraries (e.g. unisim, Intel_mf, etc.), if required
4. Start the simulation by typing "do simulate.do" into the TCL command window
5. Modelsim will now show the simulation in the wave window and some status messages in the transcript window.

6.6 Target Hardware Platforms

MII Type	Target FPGA Vendor	Target FPGA Architecture	Target Hardware Platform
RMII	Xilinx®	Spartan-6 LX®	Mars MX1
RGMII	Intel®	Cyclone IV®	Mercury CA1
	Xilinx®	Spartan-6 LXT®	Mars MX2
		Artix-7T™	Mars AX3
		Kintex-7T™	Mercury KX1
		ZYNQ 7000™	Mars ZX3
		UltraScale+	Mercury XU5
IMAC	Xilinx®	UltraScale+	Mercury XU1
SGMII	Xilinx®	Virtex-7™	Xilinx VC707

Table 37: Reference Design Target Hardware Platforms

Table 37 lists the MII type, target FPGA vendor and target FPGA architecture combinations for which Enclustra provides reference design target hardware platforms.

6.7 Required Development Tools

Action	Intel®	Xilinx®
Synthesis, Place & Route	Intel® Quartus® 18.1	Xilinx® Vivado™ 2019.1
Functional Simulation	Mentor Graphics® ModelSim PE 2019.3	

Table 38: Required Development Tools

Table 38 lists the required development tools and the recommended tool versions.

7 Using the IP-Core in projects

7.1 Plain HDL Projects

For a plain VHDL or Verilog projects the files must be compiled into the following libraries. Then the IP core can be instantiated from the library lib_en_udp_ip_eth_<core version>. “Core version” can be for example “rgmii_xil_x7”.

Folder	Library
\ip\hdl\vhdl	lib_en_udp_ip_eth_<core version>
\ip\hdl\lib\en_udp_ip_eth_<core version>_cl	en_udp_ip_eth_<core version>_cl
\ip\hdl\lib\en_udp_ip_eth_<core version>_cn	en_udp_ip_eth_<core version>_cn
\ip\hdl\lib\en_udp_ip_eth_<core version>_cs	en_udp_ip_eth_<core version>_cs

8 Ordering, Support and License Types

8.1 Product Ordering

8.1.1 Product Ordering Codes

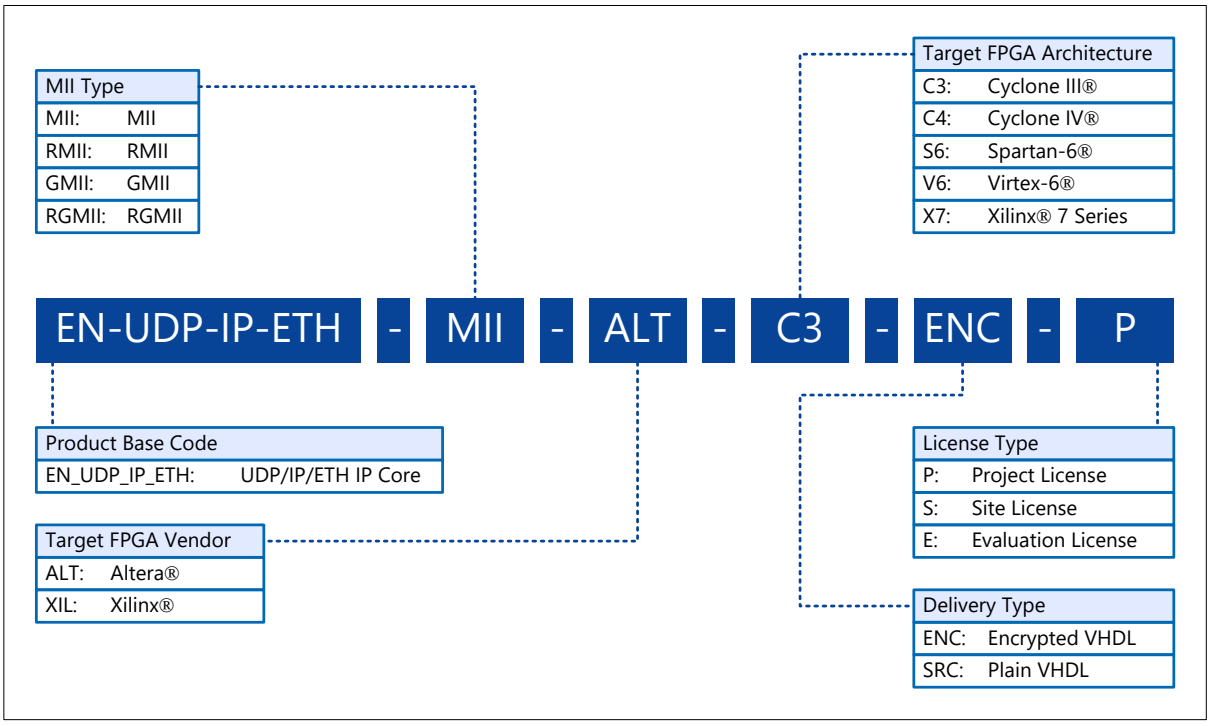


Figure 19: Product Ordering Codes

Please note that the plain VHDL option (SRC) is only available together with the site license option (S) and the evaluation option (E) is only available with the encrypted VHDL option (ENC).

8.1.2 Product Ordering Contact

Please use Enclustra's online request/order form for ordering or requesting information:

http://www.enclustra.com/order

8.2 Product Support

Please follow the instructions on Enclustra's online support site:

http://www.enclustra.com/support

8.3 Product License Types

8.3.1 SignOnce Product License

The license is granted for the use in a specific product or project, respectively. The full SignOnce product license agreement can be found here:

http://www.enclustra.com/signonce-project.

8.3.2 SignOnce Site License

The license is granted for the use at a specific site for the use in an unlimited number of products or projects, respectively. The full SignOnce product license agreement can be found here:

http://www.enclustra.com/signonce-site.

Figures

Tables

References