# UDP/IP/Ethernet

## IP Core Overview

### Purpose

This document provides an overview of the UDP/IP/Ethernet IP Core.

### Summary

In this document, the different operation modes and the different configuration modes of the IP core are described. The document also lists performance and FPGA resource usage numbers for specific use cases.

| **Product Information** | Number | Name |
|---|---|---|
| Product | EN-UDP-IP-ETH | UDP/IP/Ethernet IP Core |
| Base Development Project | | |

| **Document Information** | | | Reference | Version | Date |
|---|---|---|---|---|---|
| Reference | Version | Date | D-0000-106-001 | 1.04 | 20.09.2019 |

| **Approval Information** | Name | Position | Date |
|---|---|---|---|
| Written by | Christoph Glattfelder | Product Manager | 20.09.2019 |
| Verified by | Gian Köppel | Design Engineer | 20.09.2019 |
| Approved by | Patrick Müller | Quality Manager | 20.09.2019 |

## Copyright Reminder



## Document History

| Version | Date | Author | Comment |
|---|---|---|---|
| 1.04 | 20.9.2019 | C. Glattfelder | Minor updates |
| 1.03 | 23.01.2018 | C. Glattfelder | Added SGMII Interface |
| 1.02 | 01.04.2016 | C. Glattfelder | Added resource utilization information |
| 1.01 | 17.02.2016 | C. Glattfelder | Updated block diagram |
| 1.00 | 25.01.2016 | C. Glattfelder | First release |

# Table of Contents

# 1 Conventions

## 1.1 Terms

| Term | Description |
|---|---|
| UDP Interface | The UDP/IP/Ethernet core supports multiple UDP interfaces. Each UDP interface has its own RX and TX buffer and the user can specify a UDP port number for the RX filter as well as source and destination UDP ports for the transmission for each UDP Interface |

*Table 1: Terms*

## 1.2 Abbreviations

| Abbreviation | Description |
|---|---|
| AXI4 | ARM AMBA AXI Protocol V4.0, typically memory mapped |
| AXI4-Lite | Light-weight, single transaction memory mapped interface |
| MAC | Media Access Controller |
| PHY | Ethernet Physical Layer interface chip |
| ETH | Ethernet |
| FPGA | Field programmable gate array |

*Table 2: Abbreviation*

# 2 Introduction

## 2.1 Overview

The UDP/IP/Ethernet IP Core implements a versatile communication solution that allows data transfer via Ethernet using the UDP protocol without the need of a CPU or Ethernet stack.

It supports 10 M, 100 M and Gigabit Ethernet and IPV4. A 10 G version might be available in future, contact us for details.

It provides the following interfaces:

- Multiple UDP interfaces (only limited by the FPGA resources)
- One RAW Ethernet interface (optional)
- Configuration Interface (Mode, MAC and IP address, UDP ports)
- Ethernet PHY Interface (MII, RMII, GMII, RGMII, IGMII, SGMII, UltraScale+ MAC)

An example application of the UDP core is shown in the figure below.

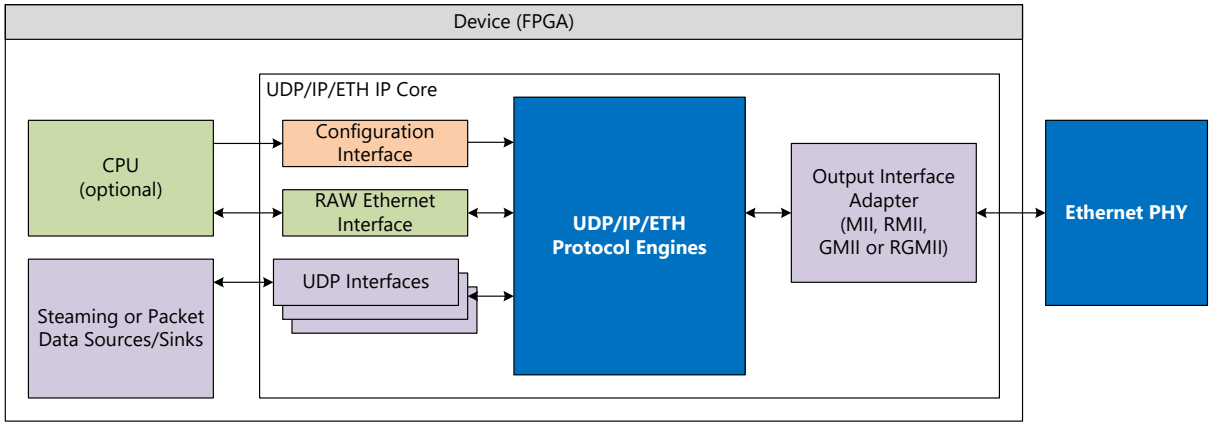


*Figure 1: UDP Core Example Application*

The UDP core supports two basic operation modes. In the streaming mode the core sends packets when a certain amount of data is stored in the transmit buffer or after a timeout. In packet mode the user must provide a packet end signal that triggers the transmission. The receive path is always working in packet mode. For streaming data applications the packet end signal can be ignored.

The UDP core supports multiple UDP interfaces. The user can assign a range of UDP port numbers to each UDP interface. So frames sent to different UDP destination ports can be separated into different receive buffers.

The optional RAW Ethernet interface allows to send and receive all kinds of Ethernet frames. For example a CPU attached to the FPGA can communicate via TCP/IP over the same Ethernet cable or request an IP address via DHCP. Also UDP frames that are not assigned to any UDP Interface are received on the RAW Ethernet interface.

Normally the data for transmitting the UDP, IP and Ethernet headers is taken from the configuration interface and all header data is stripped off in the receive path. For special applications the Header

Pass-Through Mode allows to provide single data fields (e.g. destination IP address or destination UDP port) through the UDP transmit interface. The core can also pass single header fields to a UDP receive interface. So for example the IP address of the sender can be passed to the processing logic.

## 2.2 Licensing System

The UDP/IP/ETH IP core provides a flexible licensing system. The user can granularly select licenses for the features they really require:

- At least one FPGA vendor and architecture
- At least one PHY interface (MII, RMII, GMII or RGMII)

It is possible to extend the license to incorporate additional features at any time by only paying for the cost of the additional features, plus a handling fee. Therefore, the UDP/IP/Ethernet IP core is an optimal long-term investment, which can easily be adapted to new requirements (e.g. new projects or different FPGA vendors) without having to pay for a complete solution again.

With the encrypted project or site license you get encrypted VHDL files. This allows configuring the IP core at compile time with generics.

With a source code license you get the full VHDL source code. This makes the UDP/IP/Ethernet IP core a feasible solution even if you have the requirement of having all source code in house.

# 3 IP Core

## 3.1 Overview

The figure below shows an overview of the IP core architecture.

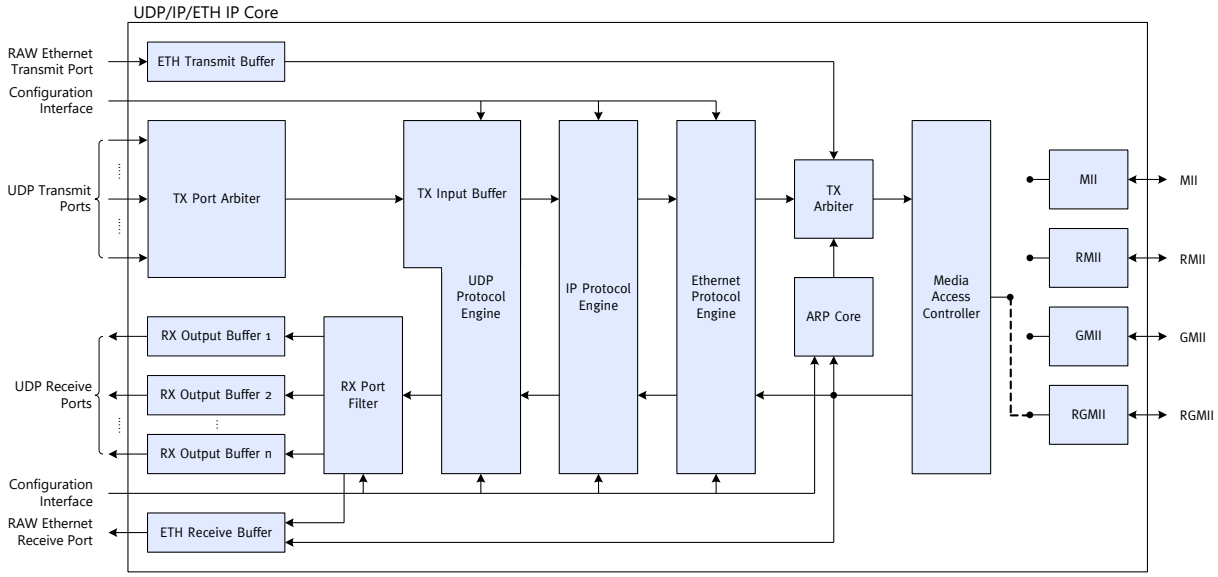


*Figure 2: IP Core Overview*

The IP core consists of following main design parts.

- **Protocol Engines:**
  The protocol engines add the header for each protocol layer in the transmit path and remove the headers in the receive path. The protocol engines also filter the received packet according to the data provided on the configuration interface.

- **TX Port Arbiter:**
  The TX port arbiter arbitrates between the different TX input ports. It can either work in round-robin or in priority mode. Only one port at once can write to the transmit buffer. If packet buffers for each UDP interface are needed these have to be connected externally.

- **Transmit Buffer:**
  The transmit buffer is part of the UDP protocol engine because the UDP header contains the length of the payload as well as a checksum over the whole UDP packet. This information is generated by the UDP protocol engine while the UDP payload is written into the transmit buffer.
  The transmit buffer is implemented as double buffer. This allows already writing the data of the next frame into the UDP core while the last frame is still transmitted.

- **Receive Buffers:**
  The receive buffers are implemented as packet buffers and can store multiple packets. The number of packets per buffer can be adjusted by a generic and has direct influence to the resource utilization.
  A packet must be received completely before it is available on the output interface. Erroneous packets are discarded by the receive buffer.
  The receive buffer can also be bypassed for special applications. Then erroneous frame are indicated by the RX error signal.

- **Receive Filter:**
  The receive filter assigns the UDP packets to the different output buffers according the configured UDP ports. It also removes UDP frames assigned to a UDP interface from the RAW Ethernet interface.

- **Output Interface Adapter:**
  This block converts the 8 bit data from the MAC to according PHY interface. It also takes care of the clock crossing if needed.

- **ARP core:**
  The optional ARP core can be enabled to automatically answer ARP requests.

## 3.2 Architecture

The UDP/IP/ETH core operates on a single system clock. This has to be at least 125 MHz for Gigabit Ethernet and at least 12.5 MHz for 10/100M Ethernet. The clock crossing for the PHY interface is done in the output interface adapter.

The whole UDP/IP/ETH core works internally with 8 bit data width.

## 3.3 Interfaces

### 3.3.1 UDP TX Interface

The UDP TX interfaces are implemented as 8 bit FIFO interfaces that work on the system clock. Further signals to discard packets and to indicate the end of a packet for packet mode are available.

Each TX interface has a built-in double packet buffer. This allows already writing the next frame into the buffer while the last frame is still transferred by the UDP core.

Normally only the UDP payload is transferred via the UDP TX interface. The information for the protocol headers is taken from the configuration interface. For special applications the core supports the header pass-through mode that allows providing single fields of the header (e.g. destination IP address) via the FIFO interface.

### 3.3.2 UDP RX Interface

The UDP RX interfaces are implemented as 8 bit FIFO interfaces that work on the system clock. Each RX interface has a packet buffer on the output. The size of the RX buffers is configurable at compile time. If required the RX buffers can also be bypassed.

The end of each received packet is indicated by the RX last signal. For streaming applications this signal can be ignored.

Normally only the UDP payload is transferred via the UDP RX interface and all header data is stripped off. For special applications the core supports the header pass-through mode that allows receiving single fields of the header (e.g. IP address or UDP port) via the FIFO interface.

### 3.3.3 RAW Ethernet Interface

The RAW Ethernet Interface allows transferring Ethernet packets of any kind.

On the TX interface the user must supply a complete Ethernet frame starting with the MAC addresses and ending with the last data byte. The CRC is added by the UDP/IP/ETH core.

On the receive interface the user gets the whole Ethernet frame from the MAC addresses to the last data byte. The CRC is checked and cut off by the UDP/IP/ETH core. Invalid frames are dropped by the core.

### 3.3.4 Configuration Interface

On the configuration interface the user can enable or disable the RX and the TX path individually.

Further the user can choose if the MAC and IP address and the UDP port of incoming packets shall be checked. If the check is disabled for one layer, all incoming packets are received independent of the according address or port. If the checks are enabled only packets matching the own MAC or IP address or UDP port are processed. Multicast and broadcast packets are always processed.

Also the TX trigger level and the timeout for the streaming mode are configured via the configuration interface.

All parameters of the configuration interface are signals and can be changed at runtime. In the plain VHDL version these signals must be connected to a register bank or constants by the user.

In Q2 2016 there will also be components for Xilinx Vivado and Intel Qsys. These components will have a built in AXI register bank for the configuration interface and access to the RAW Ethernet interface.

### 3.3.5 PHY Interfaces

The common PHY interfaces (MII, RMII, GMII and RGMII) are supported on all architectures. Some interfaces that require FPGA specific blocks or additional IP cores (like SGMII) are only available for some architectures. For special use cases there is also an IGMII interface available that allows the user to connect the UDP/IP/Ethernet core to an own user logic or third-party IP cores instead of an Ethernet PHY.

Further the UDP/IP/Ethernet core can also be connected to the built in MAC of the Xilinx ZYNQ UltraScale+ SoC devices.

## 3.4 Operating Modes

The operating mode is configured at compile time and affects all UDP interfaces.

### 3.4.1 Streaming Mode

In streaming mode the transmission of a frame is triggered when a certain amount of data has been written to the TX buffer or after a timeout. The trigger level and timeout are configurable for each port at runtime. The UDP payloads of the received packets can be read from the FIFO-like UDP RX interface as a data stream.

### 3.4.2 Packet Mode

In packet mode the user must trigger the transmission of each UDP packet via the TX interface. The end of each received packet is indicated on the RX interface.

### 3.4.3 Header Pass-Through Mode

Because the static configuration of the MAC and IP destination address and other data via the configuration interface is not very flexible, UDP core offers the header pass-through mode.

In this mode the user can supply the data for single header fields (e.g. the destination IP address) via the UDP TX interface. This information must be sent as a header in front of each packet. The data for the other fields is taken from the configuration interface.

The user can also configure the core to provide single header fields in front of each received packet on the UDP RX interface.

Which fields are processed by the header pass-through mode can be defined at compile time and applies to all UDP interfaces. The header pass-through mode is only available in packet mode.

Figure 3 shows the basic operation of the header pass-through mode.

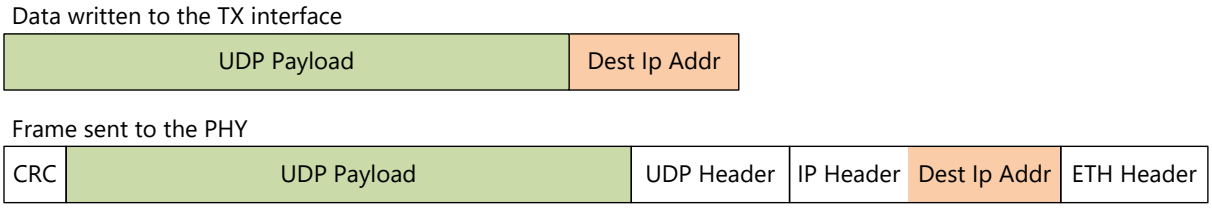


*Figure 3: Header pass-through mode*

# 4 FPGA Resource Utilization

This section lists the FPGA resource utilization for typical uses cases. The number of used BRAM is strongly dependent of the buffer size and number of streaming interfaces.

| Configuration | Slice FF | Slice LUT | BRAM |
|---|---|---|---|
| FPGA: Artix-7<br>UdpPortCount_g = 1<br>MaxPayloadSize_g = 8000<br>RxBufferCount_g = 4<br>EnableRawEth_g = true<br>RawEthRxBufferCnt_g = 2<br>RawEthTxBufferCnt_g = 2<br>RawEthPayloadSize_g = 2048<br>ArpReply_g = true | 2550 | 2100 | 16 (18k) |
| FPGA: Cyclone IV<br>UdpPortCount_g = 1<br>MaxPayloadSize_g = 2000<br>RxBufferCount_g = 2<br>EnableRawEth_g = false<br>ArpReply_g = true | 2700 | 2100 | 11 (9k) |

*Table 3: Resource utilization*

## Figures



## Tables