# Xilinx 10Gbps TCP/IP Stack - Datasheet

## Introduction

This IP implements a full TCP/IP stack inside an FPGA. As such it supports session maintenance and control, such as creation and tear-down of TCP sessions, data transmission and reception via UDP and TCP, as well as basic auxiliary functionality including IP, ARP, ICMP and DHCP. The design exposes a simple, socket-like user interface, which allows the user to perform the above-mentioned functionality. It used external DRAM to buffer the segment payload during TCP processing and on-chip BRAM to maintain all the required data structures.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **IP Core Facts** | | | | |
| Device Families | 7-Series | | | |
|  | **IP** | **LUT** | **FF** | **BRAM** |
| Resources Used | TCP Offload Engine | 13602 | 13543 | 508 |
|  | UDP Offload Engine | 1840 | 1773 | 16 |
|  | TCP & UDP Offload System Stack | 92686 | 102012 | 504 |
| Special Features | Gigabit Transceiver, Xilinx 10G Ethernet PCS/PMA & MAC IP, BlockRAM for all configurations. 7-seres MIG, External DRAM for TCP Offload Engine | | | |
| **Provided with Core** | | | | |
| Documentation |  | | | |
| Design Files | C/C++ Vivado HLS source and appropriate C/C++ test-benches.  RTL Wrappers and additional modules as appropriate.  System simulation and test-benches using Mentor Graphics Modelsim.  Python scripts for test vector generation.  Xilinx Vivado projects for Xilinx VC709 and Alpha Data boards. | | | |
| Constraints Files | XDC constraints files | | | |
| **Design Tool Support** | | | | |
| High-Level Synthesis | Vivado HLS 2015.1 | | | |
| HDL Synthesis | Vivado Logic Synthesis 2015.3 | | | |
| Implementation | Vivado 2015.3 | | | |
| Verification Tools | Vivado HLS 2015.3  Mentor Graphics Modelsim 10.1e and above | | | |
| **Support** | | | | |
| Provided by Xilinx Labs. Please contact Kimon Karras ([kimonk@xilinx.com](mailto:kimonk@xilinx.com)) or Michaela Blott (mblott@xilinx.com) | | | | |

## Key Features & Limitations

The key design features are as follows:

* Fully functional TCP/IP end point on VC709/Alpha Data boards
* UDP support
* 10Gbps sustained line-rate performance
* Supports 10k sessions (and can scale further contingent on BRAM availability)
* Sub-2us latency
* Uses external DRAM for payload buffering (130KBs required per session)
* Basic DHCP Client for IP address configuration
* 3rd party compliant
* Flow control/congestion management
* Out of order processing up to 4 segments (programmable)
* ARP Module with support for 10K entries
* Almost exclusively design in C++ using Xilinx’s Vivado HLS

Key limitations of the designs are:

* No IP fragmentation support
* Various IP and TCP options and flags
  + Are ignored if present
* No IPv6 support

## Applications

* Telecommunication
* Data Centers
* Computer Networks
* Network Interface Cards

## Functional Description

The TCP/IP stack is organized into systems which comprise the engines themselves, as well as several supporting components which implement the necessary network functionality to enable TCP and UDP protocol traffic to reach the appropriate engine. These modules comprise:

* + An IP handler, which receives frames from the network interface, verifies that this nodes is indeed the destination node, strips the Ethernet header from the frame and passes it on to the appropriate module depending on the underlying protocol. Supported protocols include:
    - Address Resolution Protocol (ARP)
    - Internet Control Message Protocol (ICMP)
    - Transmission Control Protocol over IP (TCP/IP)
    - User Datagram Protocol over IP (UDP/IP)
  + An ARP Server which answers ARP requests destined for this device and generates ARP requests to resolve MAC addresses of potential destination while caching the results in its own ARP cache.
  + An ICMP server which support a sub-set of all available ICMP messages. It replies to ICMP echo requests with ICMP echo replies and generates Destination Unreachable and TTL expired messages when appropriate.
  + A UDP Offload engine which completely processes UDP packets upon their reception or transmission and passes them on to the application or to the network interface.
  + A TCP Offload engine which performs the equivalent functionality for TCP segments. The TCP Offload engine uses external DRAM to buffer data received or transmitted and a RTL block which is a hash table and performs the Tuple-to-Session ID look up.
  + A DHCP client which can (but doesn’t have to) be used to obtain an IP address for the device from a DHCP server.
  + An Application Multiplexer that allows both a user application and the DHCP client to use the UDP Offload engine at the same time.

Figure 1 illustrates how these modules are interconnected between themselves as well as the external interfaces of the system.

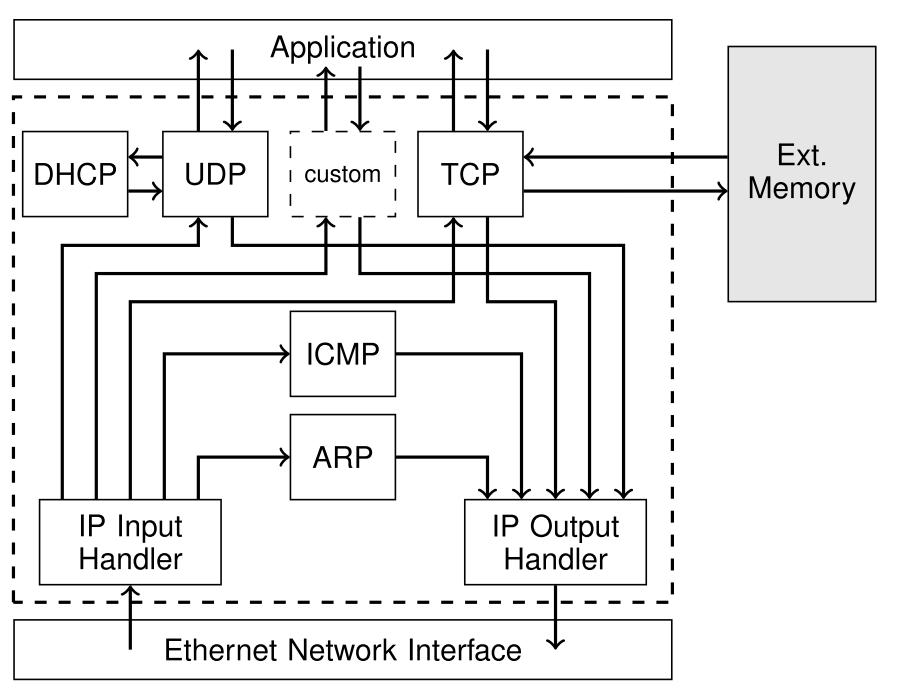


Figure 1 – TCP/IP stack architecture

The system interfaces can be broadly sub-divided into three categories:

* Interfaces to the network interfaces
* Interfaces to the MIG controller
* Interfaces to the user application from the TCP Offload engine
* Interfaces to the user application from the UDP Offload engine

## System Interfaces

This section provides detailed information on the TCP/IP stack’s external interfaces. All of the interfaces are AXI4 streams and as such always consist of data bus and a corresponding *VALID* and *READY* signal. Interfaces marked as *Extended AXI4S* additionally include a *KEEP* and a *LAST* signal. All arithmetic values (e.g. port number & packet lengths) used throughout the interfaces are always little endian.

Table 1 lists general configuration inputs including MAC and IP address. Table 2 includes the two 64 bit AXI4 stream interfaces that the system uses to connect to the Xilinx 10G MAC IP core. While other MAC solutions may be used with the system, their interface must conform to that found in Table 2.

Table 1 – TCP/IP stack configuration signals

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Description** |
| aclk | Input | Wire | Clock input |
| aresetn | Input | Wire | Active low reset |
| myMacAddress[47:0] | Input | Wire | System MAC address input |
| inputIpAddress[31:0] | Input | Wire | System IP address input |
| dhcpEnable | Input | Wire | Enables (1) or disables (0) DHCP address assignment |

Table 2 – Interface Signals to the Network Interface

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Description** |
| AXI\_M\_Stream[63:0] | Output | Extended AXI4S | Output packet data destined for the network interface |
| AXI\_S\_Stream[63:0] | Input | Extended AXI4S | Input packet data from the network interface |

Table 3 lists the interfaces the TCP Offload engine used to communicate with the external DRAM buffer. There are 4 interfaces in total, 2 for accessing the Rx segment data buffer and 2 for accessing the Tx segment data buffer. In each case there’s one interface for writing to the buffer and one interface for reading from it. Each interface consists of two or three AXI4 streams. The first streams transfers the memory command and the second stream transfers the data. The write interfaces include a third stream which contains the response to the memory access from the memory controller.

Table 3 – Interface Signals to the DRAM Memory Controller

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Description** |
| m\_axis\_rxread\_cmd[71:0] | Output | AXI4S | Read Command for the Rx Segment Data Buffer |
| m\_axis\_rxwrite\_cmd[71:0] | Output | AXI4S | Write Command for the Rx Segment Data Buffer |
| s\_axis\_rxwrite\_sts[7:0] | Input | AXI4S | Status Signal signalling the success or failure of a write operation to the Rx Buffer |
| s\_axis\_rxread\_data[63:0] | Input | Extended AXI4S | Segment data read out of the Rx Buffer |
| m\_axis\_rxwrite\_data[63:0] | Output | Extended AXI4S | Segment data written into the Rx Buffer |
| m\_axis\_txread\_cmd[71:0] | Output | AXI4S | Read Command for the Tx Segment Data Buffer |
| m\_axis\_txwrite\_cmd[71:0] | Output | AXI4S | Write Command for the Rx Segment Data Buffer |
| s\_axis\_txwrite\_sts[7:0] | Input | AXI4S | Status signal notifying of success or failure of a write operation to the Tx Bufffer |
| s\_axis\_txread\_data[63:0] | Input | Extended AXI4S | Segment data read out of the Tx Buffer |
| m\_axis\_txwrite\_data[63:0] | Output | Extended AXI4S | Segment data written into the Tx Buffer |

Table 4 includes all the signals that the TCP Offload engine uses to interface with the user application. These can broadly be sub-divided into two categories, control signal and data signals, with each path (Rx and Tx) possessing one of each. The control signals are used to open ports for listening and other control-related operation, whereas the data signals are used to read and write segment data from and to the TOE respectively.

Table 4 – Interface Signals from the TCP Offload Engine to the User Application

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Description** |
| s\_axis\_listen\_port[15:0] | Input | AXI4S | Notifies the TOE that this port should be opened for listening. |
| m\_axis\_listen\_port\_status[7:0] | Output | AXI4S | Reply from the TOE indicating success or failure in opening a port. |
| m\_axis\_notifications[87:0] | Output | AXI4S | Notification from the TOE letting the application know that data is available in the Rx buffer. Contains the following fields:   * Session ID [15:0] * Data Length [31:16] * Destination IP Address [63:32] * Destination Port [79:64] * Session Closed Flag [87:80] |
| m\_axis\_rx\_read | Input | AXI4S | Used by the application to request data to be read from the Rx Buffer. It contains the following fields:   * Session IF [15:0] * Data Length [31:0] |
| m\_axis\_rx\_data[63:0] | Output | Extended AXI4S | Data being read out from the Rx buffer and passed over to the user application. |
| m\_axis\_rx\_metadata[15:0] | Output | AXI4S | Response to a Rx buffer data request indicating the length of the data read. |
| s\_axis\_open\_connection[47:0] | Input | AXI4S | Used by the user application to instruct the TOE to actively open a session to a specified remote host. Consists of two fields:   * Remote end IP Address [31:0] * Remote end Port [47:0] |
| s\_axis\_close\_connection[15:0] | Input | AXI4S | Used by the user application to notify the TOE that the session indicated is to be closed. |
| m\_axis\_open\_status[23:0] | Output | AXI4S | Response from the TOE informing the user application of the result of a session open or close attempt. Contains the following two fields:   * Session ID [15:0] * Success Flag [23:0] |
| s\_axis\_tx\_metadata[15:0] | Input | AXI4S | Notifies the TOE that new data is available to send by passing it the session ID for which this data is intended. |
| s\_axis\_tx\_data[63:0] | Input | Extended AXI4S | Used to transfer the packet data to the TOE. |
| s\_axis\_tx\_metadata[15:0] | Input | AXI4S | Response from the TOE indicating success or failure of the data transfer operation. Possible responses are:   * No space in the buffer (-1) * No connection to host (-2) * Length of data written upon success |

The final batch of signals are the ones which enable the UDP Offload engine to communicate with the application, which are listed in Table 5. The interface is similar to that of the TCP Offload engine, with the only difference being that there are no control signals on the Tx path. This means that the user application can send data to any remote host as long as it provides the appropriate IP and port addresses. Successful communication is of course dependant on the appropriate port being open on the remote end.

Table 5 - Interface Signals from the UDP Offload Engine to the User Application

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Direction** | **Type** | **Description** |
| udp\_requestPortOpen[15:0] | Input | AXI4S | Input from the application in order to request that a specific port be opened for listening on the receive side. |
| udp\_portOpenReply[7:0] | Output | AXI4S | Reply from the UDP Offload engine indicating whether the requested port was opened successfully. |
| udp\_rxData[63:0] | Output | Extended AXI4S | Data received by the UDP Offload engine and passed over to the application. |
| udp\_rxMetadata[95:0] | Output | AXI4S | Metadata passed from the UDP Offload engine to the application on a per packet basis. They consist of the following fields:   * Source Port [15:0] * Source IP Address [47:16] * Destination Port [63:48] * Destination IP Address [95:64] |
| udp\_txData[63:0] | Input | Extended AXI4S | Data that the application provides to the UDP for transmission. |
| udp\_txLength[15:0] | Output | AXI4S | Length of the packet data to be transmitted. |
| udp\_txMetadata[95:0] | Input | AXI4S | Metadata passed on to the UDP Offload engine for data transmission. Format is identical to the Rx side metadata. |

## TCP Offload Engine Description

The most sizeable part of the TCP/IP Stack is arguably the TCP Offload Engine, which offloads all TCP communication from the user application or host processor. It consists of two parallel processing paths, one for data reception and one for data transmission, joined together by a group of tables which maintain the state of the TCP sessions. Since TCP mandates that data is buffered upon reception or before transmission and due to the size of the data that has to be stored, these buffers are implemented in external DRAM. Two separate DRAM connections are used one for the Rx segment data buffer and one for the Tx. The interfaces have been thoroughly described in Table 3. This section will describe the flow of a segment within the TCP Offload Engine in more detail. The reception of a normal segment will be used as an example here, though it must be kept in mind that depending on the type of the segment different events and processing are triggered.

Upon reception of new segment data, the Rx engine first verifies the TCP checksum of this packet and parses its header to extract the metadata from it. If the checksum is verified then the metadata is processed and depending on the current state the appropriate action is performed. Assuming the state of the session for which this packet is destined is open, then this packet is received, its segment data is written into the Rx Buffer and an event is sent to the Tx Engine to generate an acknowledgment for this segment. Additionally, a notification is sent to the user application that there’s new data available in the Rx buffer. The application can then read the data out over the Rx Application Interface at its own convenience. The Tx Engine receives the acknowledgment event and creates and sends an appropriate segment to the sender of the original data thus verifying their reception.

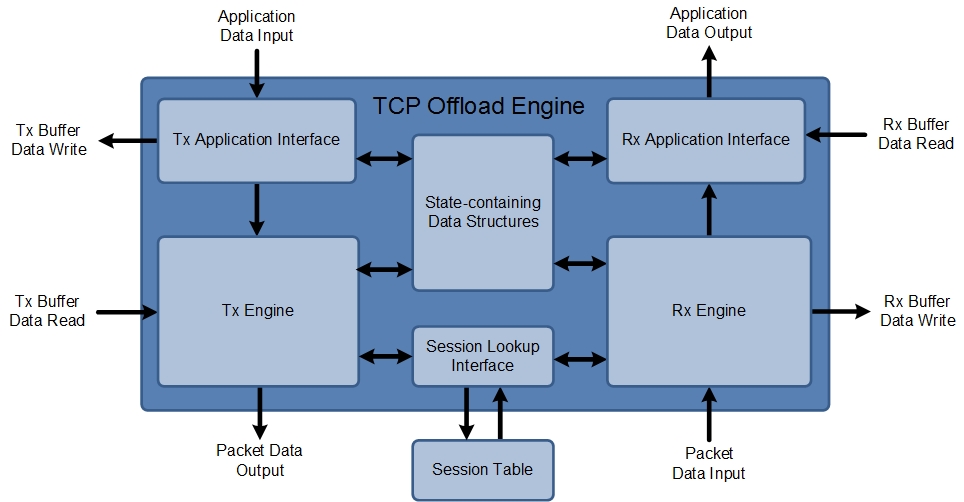


Figure 2 – Block Diagram of the TCP Offload Engine

Most of the input and output interfaces of the TCP Offload engine overlap with the external interfaces of the full TCP/IP Stack. Thus the descriptions of Table 3 and Table 4 apply to the TCP Offload Engine.

## Revision History

The following table shows the revision history for this document.

|  |  |  |
| --- | --- | --- |
| **Date** | **Version** | **Revision** |
| 15/12/2014 | 1.0 | Initial release |
| 24/02/2015 | 1.1 | Title, some wording and explanations |

## Notice of Disclaimer

Xilinx is providing this product documentation, hereinafter “Information,” to you “AS IS” with no warranty of any kind, express or implied. Xilinx makes no representation that the Information, or any particular implementation thereof, is free from any claims of infringement. You are responsible for obtaining any rights you may require for any implementation based on the Information. All specifications are subject to change without notice. XILINX EXPRESSLY DISCLAIMS ANY WARRANTY WHATSOEVER WITH RESPECT TO THE ADEQUACY OF THE INFORMATION OR ANY IMPLEMENTATION BASED THEREON, INCLUDING BUT NOT LIMITED TO ANY WARRANTIES OR REPRESENTATIONS THAT THIS IMPLEMENTATION IS FREE FROM CLAIMS OF INFRINGEMENT AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Except as stated herein, none of the Information may be copied, reproduced, distributed, republished, downloaded, displayed, posted, or transmitted in any form or by any means including, but not limited to, electronic, mechanical, photocopying, recording, or otherwise, without the prior written consent of Xilinx.