



TCP-UDP-IP STACK 10G MICROBLAZE-ZYNQ EXAMPLE DESIGN

Example design user guide

UG002

Version 1.0

December 04, 2021



Table of Contents

Introduction	3
Example design source code download	3
Firmware generation	4
Firmware synoptic	4
Theory of operations	5
Producer – Consumer model	7
Firmware generation steps	9
Library generation	9
Project generation	9
SDK application project.....	10
Firmware test menu	10
Qt test bench interface.....	12
Test examples.....	13
TCP client test.....	13
Client to Server transfer test.....	13
Server to client transfer test	15
TCP server test.....	17
Server to client transfer test	17
Client to server transfer test	18
UDP transmitter test.....	19
UDP receiver test.....	20
TCP loop test	21
UDP loop test	22

Introduction

The purpose of this example design is to demonstrate the performance of the Microblaze/Zynq Board Support Package (BSP) of the IPCTEK's TCP/UDP/IP stack 10G FPGA IP core. With the BSP, users have a **socket-like C API** of the whole Ethernet 10G TCP/UDP/IP FPGA stack. Within several simple function calls, users can send/receive UDP datagrams or open a TCP session and exchange data with the TCP peer. All hardware-dependent jobs are abstracted from the user's point of view. This is a perfect plug-and-play solution for remote-control applications. Main characteristics of the IP core are given in the table below.

Parameter	Value	Unit
MAC MTU	9000	Bytes
Number of entries in the routing table	64	
Number of TCP server (or client)	1	
TCP TX buffer size	32	KBytes
TCP RX buffer size	32	KBytes
TCP out-of-order Handling Enable	"TRUE"	
Out-of-order reordering buffer size	32	KBytes
Number of UDP TX	1	
Number of UDP RX	1	

Table 1 - IP core's parameters

Example design source code download

In order to compile the example design we need to download the Vivado Hdl firmware, the SDK bare-metal application and the Qt test bench source code. All the design source code and an evaluation netlist can be requested by sending an e-mail to contact@ipctek.net

- **Vivado Hdl firmware:** this repository ([ip_stack_10g/](#)) contains necessary source code and scripts to generate the FPGA firmware Vivado project.
- **SDK bare-metal application project:** the example design is running on a Microblaze-based subsystem. The source code is in **drivers/** folder.
- **Qt-based test bench (optional):** an UDP transceiver and a TCP server/client test bench which can be run on a PC host to interact with the FPGA firmware. The source code is in **Qt/** folder.

Firmware generation

In this section we detail the step-by-step process in order to generate the FPGA firmware used in the example design.

Firmware synoptic

The FPGA firmware synoptic is illustrated in the figure below. In the design, we implement a Ten-Gigabit TCP/UDP/IP stack which will be running on Xilinx evaluation boards. We use the Xilinx's 10G/25G Ethernet Subsystem IP serving as the Ethernet MAC along with the PCS/PMA stack.

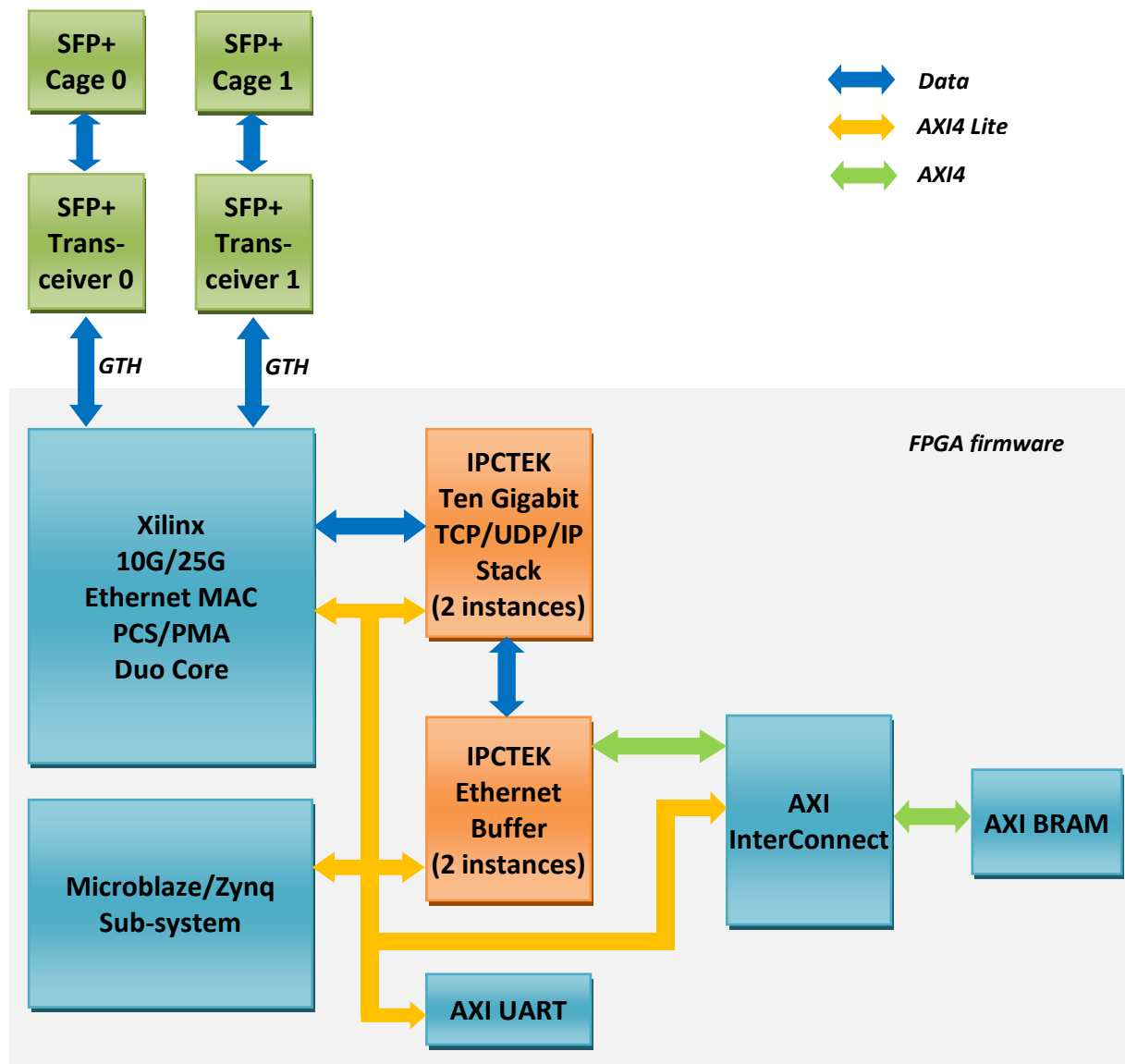


Figure 1 - FPGA firmware synoptic

The FPGA firmware is composed of following principal components.

- **10G Ethernet MAC:** A Xilinx's 10G/25G Ethernet Subsystem is used. The IP is configured with two cores in order to interface with two different SFP+ transceivers.
- **IPCTEK's Ten-Gigabit TCP/UDP/IP stack:** One TCP engine (can be switched between server and client), one UDP transmitter and one UDP receiver are instantiated in the IP core. The MAC MTU is 9000 bytes. The IP core is embedded in an AXI4-Lite wrapper for an easy integration within a Microblaze/Zynq sub-system. Two instances of the IP are implemented in order to have two different TCP/UDP/IP stacks.
- **IPCTEK's Ethernet Buffer IP core:** For the RX path, this IP buffers received TCP and UDP data before writing the data into the BRAM memory. As for the TX path, the IP reads transmitted data from the BRAM before sending them to the TCP/UDP/IP Stack. The Ethernet Buffer IP can be seen as a data memory controller. The Microblaze/Zynq can have access to the IP's register space in order to control the TX and RX data memory address and length.
- **AXI4 BRAM:** This IP serves as a TX/RX data memory for the whole system. This memory buffers RX data (coming from the Ethernet Buffer IP) before they are read by the Microblaze/Zynq. It also buffers TX data (coming from the Microblaze/Zynq) before they are read by the Ethernet Buffer IP. In practice, this memory can also be an external memory such as DDR3 or DDR4. In general, any memory with an AXI4 interface can be used.
- **A Microblaze/Zynq sub-system** to configure the whole system via AXI4-Lite interface. An UART interface is also implemented in order to interact with the user. It also gets access to the BRAM memory via an AXI Interconnect IP.
- The two SFP+ ports must be connected together (or to an Ethernet switch) in case a loopback test is required.

Theory of operations

In this paragraph we describe the organization of the TX/RX data memory, which is the AXI BRAM in this example design, as long as the mechanism of the TX and the RX processes.

The memory organization is illustrated in Figure 2. The memory is divided in to ***M*** zones, where ***M*** is the number of Ethernet devices implemented in the system. In this example design, there are 2 Ethernet devices corresponding to 2 physical SFP+ transceivers.

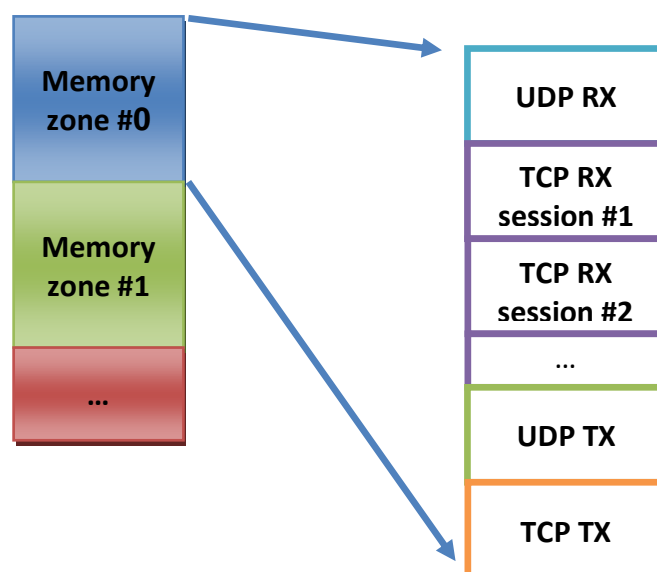


Figure 2 – TX/RX memory organization

Each memory zone described above is divided again in several sections which have the **same size**. These sections are organized from the top to the bottom of the memory.

- One section for received UDP packets.
- **N** section for received TCP sessions, where **N** is the number of TCP sessions implemented in the TCP/UDP/IP Stack.
- One section for transmitted UDP packets.
- One section for transmitted TCP segments.

The memory organization is initialized using the function *ipc_eth_buf_set_default_params()* in the file *ipc_fpga_eth_buf.c*

The section size is defined by the member *eth_buf_mem_section_len* in the structure *eth_dev_init_param_t*

There are several important points that are worth noting during the configuration of the data memory.

- The memory **section size must be larger than the Ethernet MTU**. This is logic because the memory section must be large enough to store at least one UDP packet or one TCP segment.
- The **memory must be large enough** for all the TX and RX memory sections and all the Ethernet devices present in the system.
- The system supports 32-bit memory address. However, **the address 0xFFFFFFFF is not usable**. This limitation is due to the overflow problem when calculating the write pointer low anchor value.

Failing to respect these points described above, the correct functioning of the system is not guaranteed.

Producer – Consumer model

The TX and RX paths function according to a producer – consumer model. The data memory is modeled as a circular buffer into which a producer writes data and from which a consumer reads data. In order to keep track of buffer empty and full states, the producer and the consumer share control signals such as the read pointer, read phase tag, write pointer, write pointer low anchor and the write phase tag (when the read pointer or the write pointer reaches the bottom of the memory, they return to the top of the memory and the corresponding phase tag changes). The meaning of each control signal is as follow.

- **Write pointer:** the memory address up to which the data have been written. This is the address into where a new data will be written by the producer.
- **Write pointer low anchor:** the memory address up to which the data are valid. When the amount of free memory (from the write pointer to the bottom of the memory zone) is not enough for a whole UDP packet (including the IPCTEK's UDP pseudo header) or a whole TCP segment (deduced from the MAC MTU), the producer uses the write pointer low anchor to mark the actual write pointer position before resetting the write pointer to the memory top position and changing the write phase tag. This is because the DMA engine in the FPGA does not support wrapped address and we do not want to fragment an UDP packet or a TCP segment into two DMA transactions in order to maximize the system throughput.
- **Write phase tag:** the phase tag of the write operation. When the producer resets the write pointer to the top of the memory zone, the write pointer low anchor is set and the write phase tag is changed.
- **Read pointer:** the memory address up to which the data has been read. This address is the next data address to be read by the consumer.
- **Read phase tag:** the phase tag of the read operation. When the consumer finishes reading from the top of the memory to the **write pointer low anchor**, it comes back to the top of the memory and changes the read phase tag.

The Figures below illustrate several situations encountered during a normal operation. This is useful to understand the producer-consumer model.

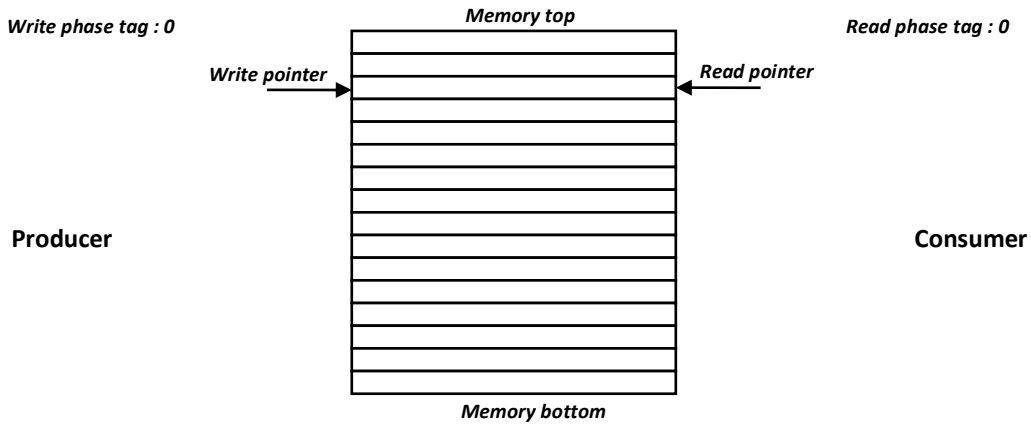


Figure 3 – Memory section buffer empty. Write phase tag = Read phase tag, write pointer = read pointer

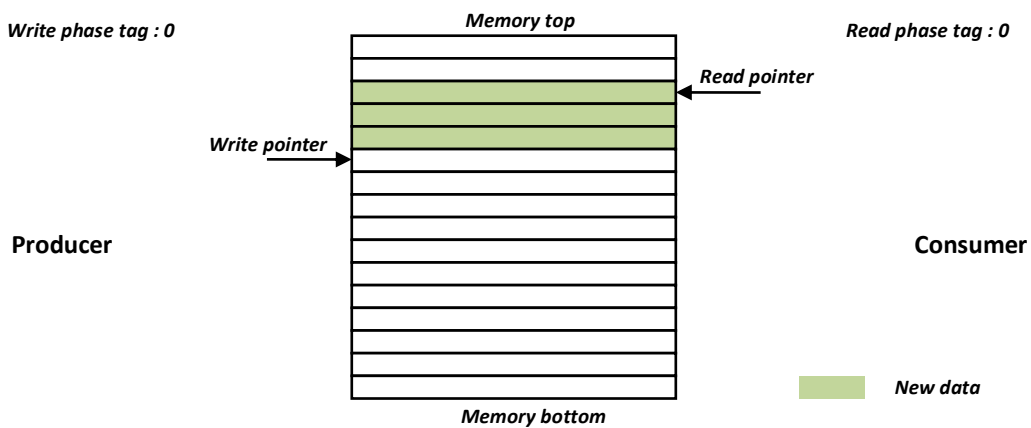


Figure 4 – There are three new entries in the memory section buffer

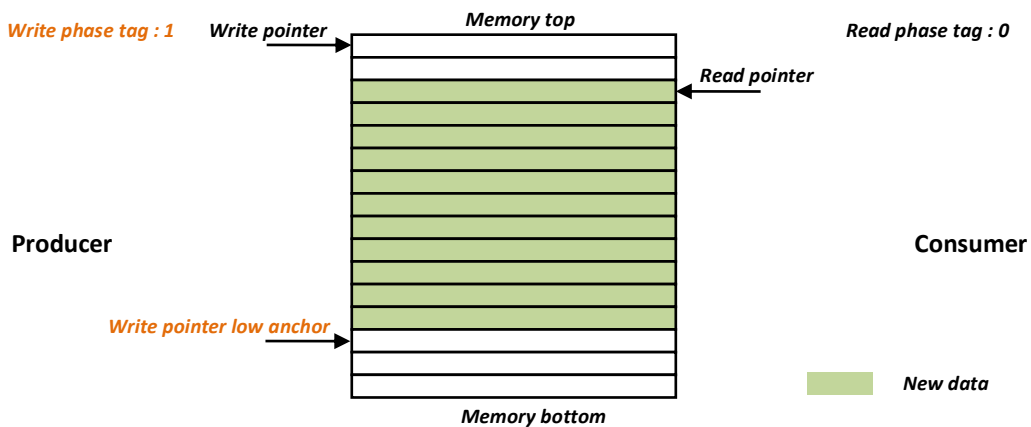


Figure 5 – The write pointer low anchor is set before resetting the write pointer and changing the write phase tag. The number of free slots at the bottom of the memory is not enough for a whole packet

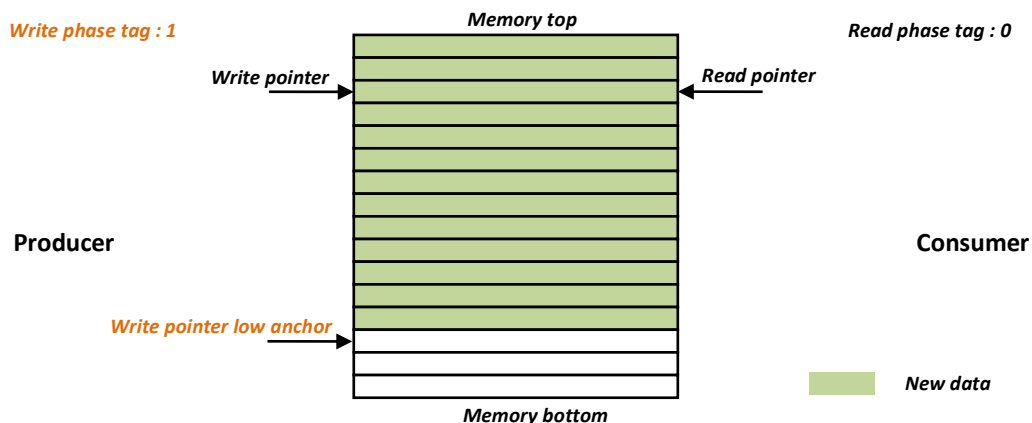


Figure 6 – Memory full. Write phase tag != Read phase tag, write pointer = read pointer

Firmware generation steps

Library generation

Before generating the example project, different necessary IPs need to be compiled.

--- Tip -----

In Windows, open the cmd console then use subst command to create a virtual Disk pointing to the Vivado Hdl root folder in order to avoid Windows's maximum path length limitation error. For example, to create a virtual disk named T, tap the following in the cmd console.

```
subst T: <path_to_the_root_folder>
```

Open Vivado, cd into the /library folder, use the Vivado tcl console

```
cd T:/library
```

Run the script buildLib.tcl to compile the library, use the Vivado tcl console

```
source ./buildLib.tcl
```

When the library compilation is finished, we proceed to generate the example project.

Project generation

cd into the project folder /projects/tcp_udp_10g_microblaze_over_[board_name]. E.g. for the KCU105 board use the Vivado tcl console

```
cd T:/projects/tcp_udp_10g_microblaze_over_kcu105
```

Run the script system_project.tcl to generate the project, use the Vivado tcl console

```
source ./system_project.tcl
```

After the script is finished loading the board design, run the synthesis then the implementation and generate the bitstream. These processes take about one hour to finish depending on the host PC.

Export the bitstream then launch the SDK. We proceed to create an SDK application project.

SDK application project

Create an application project based on the hardware that we have just exported from the Vivado project.

The SDK application source code for the example design is located at `/noOs_drivers/ip_stack_10g/tcp_udp_10g_microblaze_over_[board_name]/src` folder. Import this folder into your SDK project.

--- Note-----
Reconfigure the Linker Script to increase the stack size and the heap size to 32 KB for a stable application. In this example design we have to allocate memory for large packets.

Program the board, plug the USB/UART cable into the evaluation board, open a console application (e.g. Tera Term) then run the application. The UART baud rate is 115200 Hz. Remember to check the “*local echo*” option in Tera Term in order to see user’s input commands.

Firmware test menu

Users should find this UART console screen after launching the SDK application.

```
*****
*** Welcome to IPCTEK Ethernet example design ***
***** Microblaze-Zynq-based 10G TCP/UDP/IP *****
***** Embedded software version 2.0.0 *****
*****
Ethernet new device.
Device node index 0
IP stack base address 0x44A20000
Ethernet buffer base address 0x44A30000
Number of TCP session implemented in the hardware 1
Device MTU 9000
Ethernet new device.
Device node index 1
IP stack base address 0x44A40000
Ethernet buffer base address 0x44A50000
Number of TCP session implemented in the hardware 1
Device MTU 9000

Available commands:
udp_send      - Send UDP packets.
udp_receive   - Receive UDP packets. Verify PRBS stream integrity if required.
tcp_send      - Send a TCP stream to peer.
tcp_receive   - Receive a TCP stream from peer. Verify PRBS stream integrity if required.
udp_loop      - Send/Receive UDP packets.
tcp_loop      - Send/Receive TCP stream.
quit          - Quit.
```

Figure 7 - Firmware UART console

Available commands for test are:

- **udp_send** command: use this command to send UDP packets from the evaluation board to a destination. In this test the SFP+ port 0 is used.
- **udp_receive** command: use this command to prepare for receiving UDP packets. If the predefined PRBS sequence is about to be received, the PRBS verification function can be enabled to verify the data integrity. In this test the SFP+ port 0 is used.
- **udp_loop** command: use this command to send UDP packets from one SFP+ port to the other port on the same evaluation board. A PRBS sequence is sent from one TCP/UDP/IP stack, passing through the SFP+ transceivers and is received by the other TCP/UDP/IP stack. A PRBS verification module is also activated in order to verify the data integrity of the UDP packets. In this test both SFP+ port 0 and port 1 are used.
- **tcp_send** command: configure the 1st TCP/IP stack as either a TCP server or a TCP client. After the connection was established, send a data stream to the TCP peer. In this test the SFP+ port 0 is used.
- **tcp_receive** command: configure the 1st TCP/IP stack as either a TCP server or a TCP client. After the connection was established, prepare for receiving a data stream coming from the peer. If the predefined PRBS sequence is to be received, users can activate the PRBS verification module in order to verify the data integrity. Theoretically, as a TCP connection is guaranteed to be error-free, the data integrity check should always pass. If the Qt-based TCP server/client given by the example design is used, this data integrity check function is already implemented. This allows to test the speed performance and to validate the correct functioning of the IP core. In this test the SFP+ port 0 is used.
- **tcp_loop** command: use this command to test the TCP loop configuration. One IP stack is configured as a TCP server, the other is configured as a TCP client. In this test the connection between the TCP server and the TCP client will be established and a PRBS sequence will be sent from the client to the server. At the server side, the PRBS verification component is also activated to verify the data integrity of the TCP data stream. Do not forget to physically connect the two SFP+ ports in your evaluation board during this test. In this test both SFP+ port 0 and port 1 are used.
- **quit** command: use this command to quit the application.

--- Tip -----
*When **udp_loop** or **tcp_loop** commands are used, loopback the SFP+ transceivers in your evaluation board, or connect them to a 10G Ethernet switch.*

Qt test bench interface

The Qt test bench is written with Qt Creator version 4.13.2 and Qt version 5.12.2. In the test bench we implement a TCP Server, a TCP client and an UDP transmit/receive engine in order to interact with the IP stack on the FPGA.

In the test bench we use Windows socket for the TCP server/client and the UDP transmitter. However, we use npcap SDK to implement the UDP Receiver engine because of the poor performance of the native socket.

--- Important Note-----

The npcap version 1.05 is included in the test bench source code. In order to use the library, user must install the npcap on the host PC. By the time this document is written, the nmap version 7.92 has been installed. The nmap installer also handles the npcap installation.

The test bench interface is shown in the figure below.

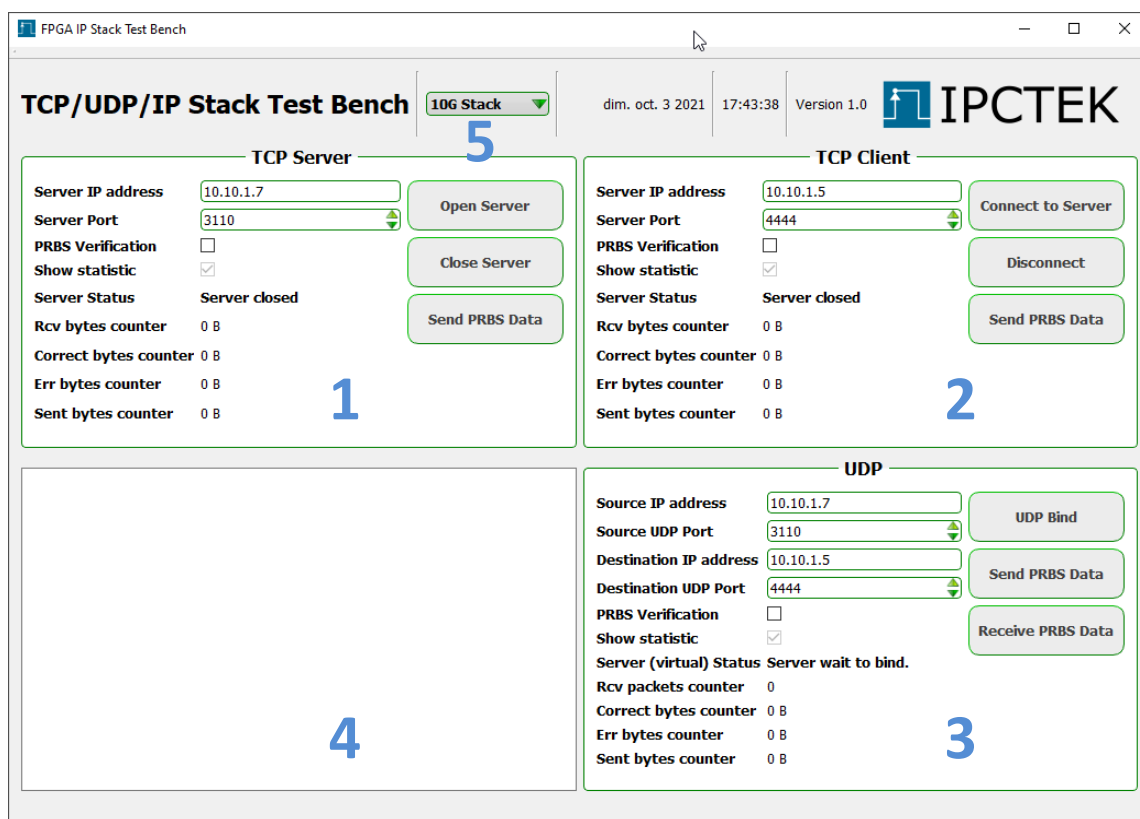


Figure 8 - Qt test bench interface

1. TCP server interface, used to test the FPGA TCP client mode.
2. TCP client interface, used to test the FPGA TCP server mode.
3. UDP transceiver interface, used to test the FPGA UDP Tx and Rx engines.

4. Test bench log, used to display useful messages during the test.
5. Select the speed. Select **10G Stack** for this example design. This is used to correctly configure the PRBS engine that matches with the one used in the Microblaze code.

Test examples

In this test examples, the SFP+ **port 0** of the evaluation board is directly connected to an Intel(R) 82599 10 Gigabit Network Connection PCIe card. The Qt test bench is running on a Windows 10 host PC.

Configure the host PC Ethernet interface to static mode with the following information

- IP address: 10.10.1.7
- Netmask: 255.255.255.0
- Gateway: 10.10.1.1

TCP client test

In this test we use the Qt TCP server interface and the FPGA TCP/IP stack is configured to the client mode.

Open the server by clicking to the **Open Server** button.

Wait until the "**Server is listening.**" text is displayed on the Server Status.

--- Note-----

The Windows firewall may ask for permission for the program to have access to the Ethernet interface. In this case click Yes to give the permission.

The TCP server is now listening for a connection request. We proceed to prepare for a TCP client on the FPGA.

Client to Server transfer test

In this example we will transfer 1000 segments of size 8960 bytes from the Microblaze client to the Qt server. While receiving the data stream, the server also verifies the data integrity of the stream.

Check the **PRBS Verification** check box to enable the verification option.

In the FPGA UART console, use the **tcp_send** command to send data to the server.

When asked for the server/client mode, input 0 to select the client mode.

When asked for the server address, input 10.10.1.7, which is the address of the Qt TCP server.

When asked for the server port, input the port number configured in the Qt TCP Server interface, by default it is 3110.

When asked for the client's port number, input a number, e.g. 4444.

Upon success, the **"Connected"** text should be displayed at the Qt TCP server interface. Several pieces of information concerning the server are also displayed on the UART screen.

When asked for the data mode, input 1 to select the PRBS mode.

When asked for the segment length, input 8960. It is noted that this is the maximum segment length value, given that the MAC MTU is equal to 9000 bytes.

When asked for the number of segments, input 1000.

The screenshots of the UART console and the test bench interface when the transmission is finished are shown in figures below.

```
Please input server/client mode (0 for client mode, 1 for server mode)
Please input TCP server ip address to connect to (i.e. 10.10.1.7)
Please input TCP server's port to connect to (i.e. 3110)
Please input our TCP port (i.e. 4444)
Retry to connect in 1 second.
*** Connected server's information ***
*** MAC address : 80:61:5F: 7:9E: 4
*** IP address : 10.10.1.7
*** TCP port: 3110
Please input data mode (0 for IPCTEK greeting message, 1 for PRBS sequence)
Please input segment length (i.e. 8960 (8960 max))
Please input number of segments to be sent (i.e. 1000000)
11% sent
22% sent
33% sent
44% sent
56% sent
67% sent
78% sent
89% sent
Transmission done.
```

Figure 9 - FPGA TCP Client. Client to server transfer, UART screen

--- Important Note-----
Make sure to configure your network card to support the jumbo frame (or the extended mode with the packet length larger than 9000 bytes).
If the segment length is configured not to be a multiple of 8, it is possible that the PRBS verification at the server side is not reliable. This is because in the PRBS generation engine an 8-byte LFSR is used. At the server side, it is possible that the network interface card pushes segments whose size is different from that of the client side. Hence, it is better to configure a multiple-of-8 segment size at the transmitter.

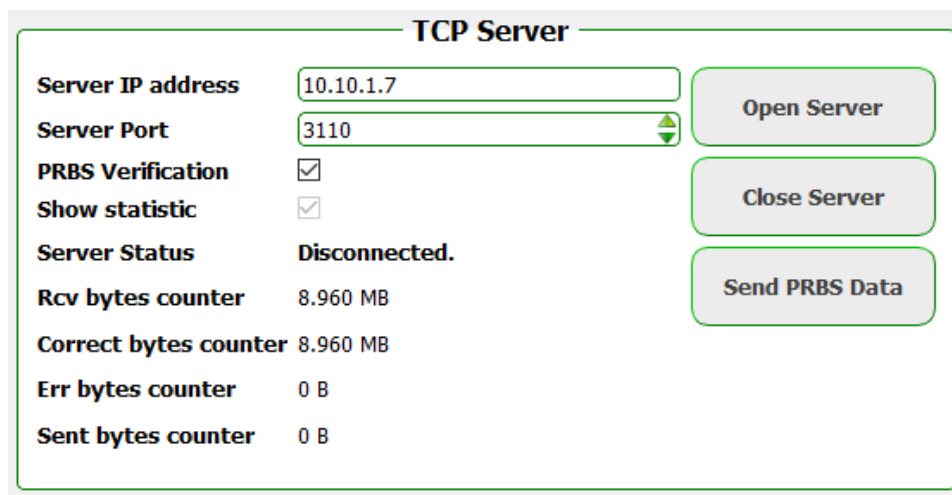


Figure 10 - FPGA TCP Client. Client to server transfer, test bench screen

--- Note-----

After each transmission using the **tcp_send** command, the PRBS generator of the Microblaze is resetted. Before restarting another transmission, the PRBS generator in the test bench should also be resetted to the initial value. Toggle the PRBS Verification checkbox to reset the PRBS engine.

Server to client transfer test

In this example the Qt test bench interface server will transfer 1000 segments of size 8960 bytes to the Microblaze/Zynq TCP client. While receiving the data stream, the client also verifies the data integrity of the stream.

In the FPGA UART console, use the **tcp_receive** command to prepare for receiving data from the server.

When asked for the server/client mode, input 0 to select the client mode.

When asked for the server address, input 10.10.1.7, which is the address of the Qt TCP server.

When asked for the server port, input the port number configured in the Qt TCP Server interface, by default it is 3110.

When asked for the client's port number, input a number, e.g. 4444.

When asked for the segment length (in number of bytes), input 8960.

When asked for the number of segments expected to receive, input 1000.

When asked for the verification option, input 1 to enable the PRBS verification function.

In the Qt test bench interface, click on the button **Send PRBS Data** to begin sending data to the client. Configure the corresponding numbers of segments and the segment length as shown in the figure below.

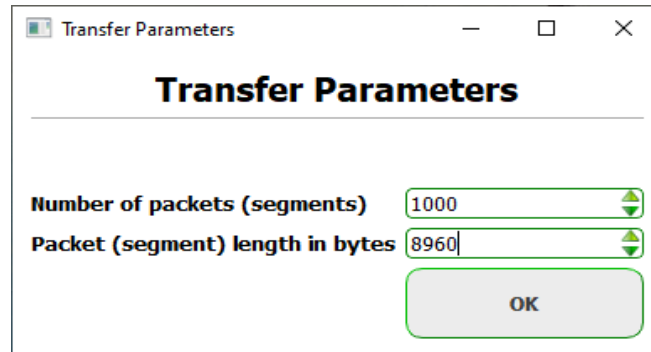


Figure 11 - FPGA TCP Client. Server to client transfer parameters

The screenshots of the UART console and the Qt test bench interface when the transmission is finished are shown in figures below.

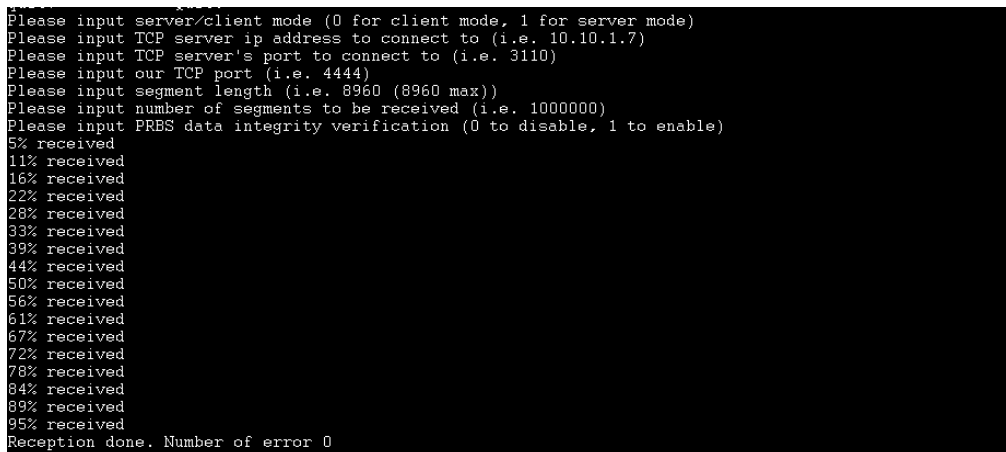


Figure 12 - FPGA TCP Client. Server to client transfer, UART screen

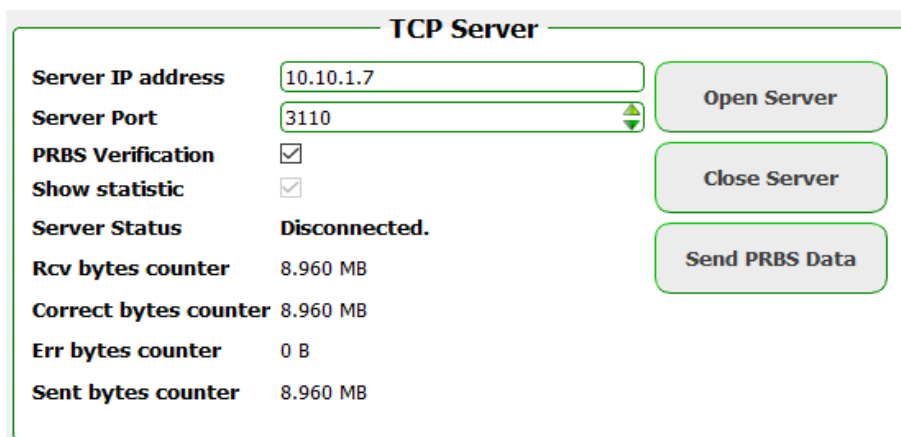


Figure 13 - FPGA TCP Client. Server to client transfer, test bench screen

TCP server test

In this test the Microblaze/Zynq will configure a TCP server. We use the Qt test bench TCP client to interact with this server.

Server to client transfer test

In this example test we will transfer 10000 segments of size 8960 bytes from the Microblaze/Zynq server to the Qt test bench client.

Check the **PRBS Verification** checkbox in order to enable the data integrity check.

Use the **tcp_send** command to send a data stream to the Qt test bench.

When asked for the server/client mode, input 1 to select the server mode.

When asked for our port, input 4444 to match the "**Server Port**" valued configured in the Qt test bench.

The Microblaze/Zynq server is waiting for a connection request, Click on the **Connect to Server** button.

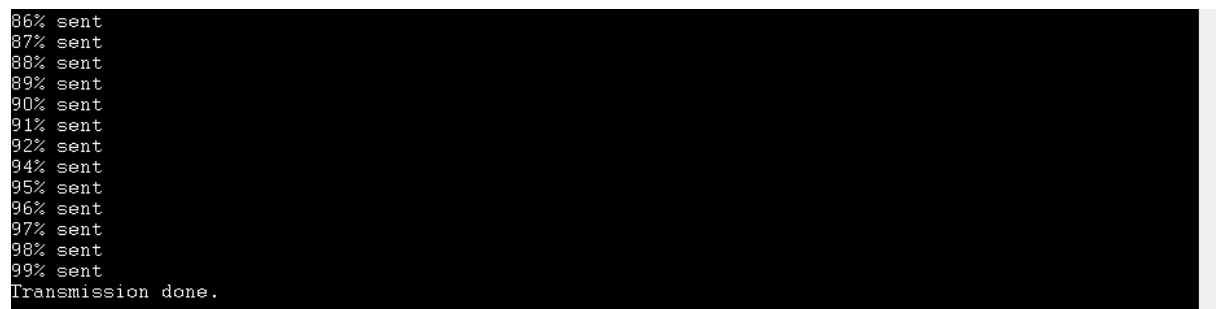
Wait until the "**Connected.**" text is displayed in the Server Status label

When asked for the data mode, input 1 to configure PRBS data.

When asked for the segment length, input 8960.

When asked for the number of segments to be sent, input 10000.

The screenshots of the UART console and the test bench interface when the transmission is finished are shown in figures below.



```
86% sent
87% sent
88% sent
89% sent
90% sent
91% sent
92% sent
94% sent
95% sent
96% sent
97% sent
98% sent
99% sent
Transmission done.
```

Figure 14 - FPGA TCP Server. Server to client transfer, UART screen

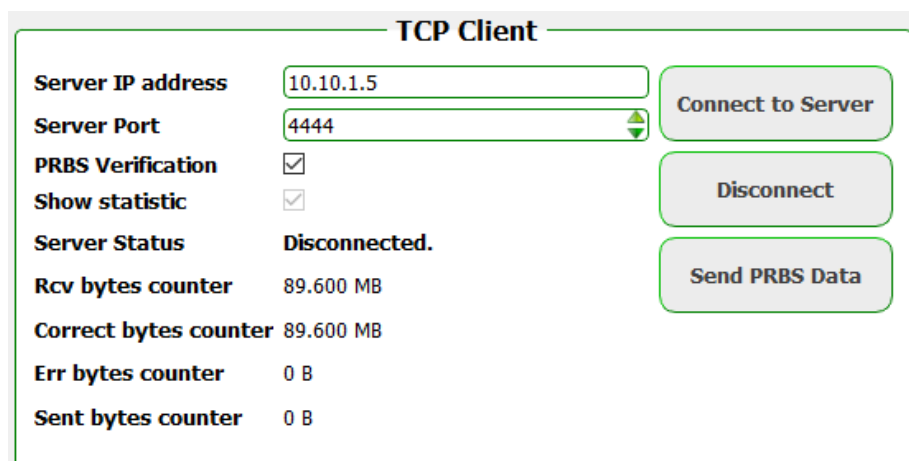


Figure 15 - FPGA TCP Server. Server to client transfer, test bench screen

Client to server transfer test

In this example test we prepare for sending 1000 segments of 8960 bytes from the Qt test bench Client to the Microblaze/Zynq server. The PRBS Verification engine in the Microblaze/Zynq is also enabled to verify the data integrity.

Use the **tcp_receive** command to prepare for the reception of the data stream.

When asked for the server/client mode, input 1 to select the server mode.

When asked for our port, input 4444 to match the "**Server Port**" valued configured in the Qt test bench.

The Microblaze/Zynq server is waiting for a connection request, Click on the **Connect to Server** button.

Wait until the "**Connected.**" text is displayed in the Server Status label.

When asked for the segment length, input 8960.

When asked for the expected number of segments, input 1000.

When asked for the PRBS verification option, input 1 to enable this functionality.

Click on the button **Send PRBS Data** (on the TCP Client group) to send data to the Microblaze/Zynq server.

The screenshots of the UART console and the test bench interface when the transmission is finished are shown in figures below.

```

Please input server/client mode (0 for client mode, 1 for server mode)
Please input our TCP port (i.e. 4444)
TCP connection has been established. Now proceed to receive data.
Please input segment length (i.e. 8960 (8960 max))
Please input number of segments to be received (i.e. 1000000)
Please input PRBS data integrity verification (0 to disable, 1 to enable)
5% received
11% received
16% received
22% received
28% received
33% received
39% received
44% received
50% received
56% received
61% received
67% received
72% received
78% received
84% received
89% received
95% received
Reception done. Number of error 0

```

Figure 16 - FPGA TCP Server. Client to server transfer, UART screen

TCP Client

Server IP address	<input type="text" value="10.10.1.5"/>	Connect to Server
Server Port	<input type="text" value="4444"/>	
PRBS Verification	<input checked="" type="checkbox"/>	Disconnect
Show statistic	<input checked="" type="checkbox"/>	
Server Status	Disconnected.	
Rcv bytes counter	89.600 MB	Send PRBS Data
Correct bytes counter	89.600 MB	
Err bytes counter	0 B	
Sent bytes counter	8.960 MB	

Figure 17 - FPGA TCP Server. Client to server transfer, test bench screen

UDP transmitter test

In this example test we use the FPGA UDP Tx engine to send 100000 packets of size 8960 bytes to the Qt test bench UDP receiver. The PRBS verification option is also enabled to verify the data integrity of the received data.

Click on the button **UDP Bind** to bind the UDP socket to the corresponding address and ports. The "**Binding success.**" text should appear in the **Server (virtual) status** label.

Enable the PRBS verification option by checking the **PRBS Verification** checkbox.

Click on the **Receive PRBS data** button to prepare for receiving the data. Refer to the system log for useful information. At this stage, the npcap is called to sniff for 100,000 UDP packets whose destination port is equal to 3110.

Use the **udp_send** command to send data to the test bench virtual server.

When asked for the destination IP address, input 10.10.1.7.

When asked for the UDP source port, input for example 4444.

When asked for the UDP destination port, the user must input 3110 to match that of the npcap packet filter.

When asked for the data mode, input 1 for PRBS data.

When asked for the packet length, input 8960.

When asked for the number of packets, input 100000.

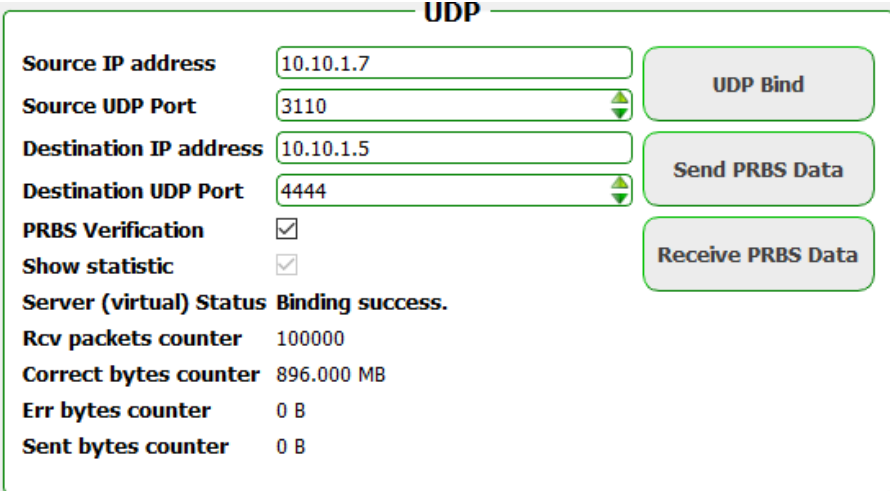
The screenshots of the UART console and the Qt test bench interface when the transmission is finished are shown in figures below.

```

99300 of total 100000 packets sent
99400 of total 100000 packets sent
99500 of total 100000 packets sent
99600 of total 100000 packets sent
99700 of total 100000 packets sent
99800 of total 100000 packets sent
99900 of total 100000 packets sent
100000 of total 100000 packets sent
Transmission done.

```

Figure 18 - UDP TX test, UART screen



The screenshot shows a Qt test bench interface titled "UDP". It contains the following fields and buttons:

- Source IP address:** 10.10.1.7
- Source UDP Port:** 3110
- Destination IP address:** 10.10.1.5
- Destination UDP Port:** 4444
- PRBS Verification:**
- Show statistic:**
- Server (virtual) Status:** Binding success.
- Rcv packets counter:** 100000
- Correct bytes counter:** 896.000 MB
- Err bytes counter:** 0 B
- Sent bytes counter:** 0 B

Buttons on the right include "UDP Bind", "Send PRBS Data", and "Receive PRBS Data".

Figure 19 - UDP Tx test, test bench screen

UDP receiver test

In this example test we use the test bench interface to send 10 UDP packets of size 1024 bytes to the Microblaze/Zynq UDP receiver. The PRBS verification module in the Microblaze/Zynq is also enabled in order to verify the data integrity.

Use the **udp_receive** command to prepare for receiving UDP packets.

When asked for the source IP address, input 10.10.1.7, which the IP address of the Qt test bench UDP virtual server.

When asked for the UDP source port, input 3110.

When asked for the UDP destination port, input 4444.

When asked for the PRBS verification option, input 1 to enable the module.

When asked for the number of packets, input 10.

On the Qt test bench interface, click on the **Send PRBS Data** button to begin sending. Configure to send 10 packets of size 1024 bytes then click on the **OK** button.

The screenshot of the UART console when the transmission is finished is shown in the figure below.

```
Please input filtered source ip address (i.e. 10.10.1.7)
Please input filtered UDP source port (i.e. 3110)
Please input filtered UDP destination port (i.e. 4444)
Please input PRBS data integrity verification (0 to disable, 1 to enable)
Please input number of packets to be received (i.e. 1000)
10 of total 10 packets received
Received 10 packets. Number of error bytes 0
Number of packets dropped by the interface: 1273
```

Figure 20 - UDP Rx test, UART screen

--- Remark-----
Why in this test do we send only 10 packets of 1024 bytes? This is because the Microblaze memory access is extremely slow when compared to a 10G Ethernet full speed. If we send many packets to the Microblaze UDP receiver, there would be a lot of dropped packets. And the PRBS verification at the receiver side is not reliable any more. The Microblaze BSP includes also functions to retrieve statistic information about the TX and RX paths. At the end of the UDP RX test, the number of UDP packets dropped by the interface (more precisely it is the Ethernet Buffer IP which discards the packets in case the packets' consumer (Microblaze in this case) is not fast enough) is also displayed.

It is noted that this example design is for remote control applications where we control or supervise our system from distance via the Microblaze. If high-speed data transfer or broadcasting applications are required, RTL designs (UG001) would be more suitable.

TCP loop test

In this test the two SFP+ modules on the evaluation board should be connected together. In this test one Ten-Gigabit TCP/UDP/IP instance serves as the TCP server, the other instance serves as the TCP client.

After the connection is established, we send 1000 segments of size 8960 bytes from the client to the server.

Use the **tcp_loop** command to begin the test.

When asked for the segment length, input 8960.

When asked for the number of segments to be sent, input 1000.

When asked for the PRBS verification option, input 1 to enable the module.

The screenshot of the UART console when the transmission is finished is shown in the figure below.

```
Please input segment length (i.e. 8960 (8960 max))
Please input number of segments to be sent (i.e. 100000)
Please input PRBS data integrity verification (0 to disable, 1 to enable)
Cannot connect to sever. Retry in 1 second.
*** Connected peer's information ***
*** MAC address : 77:88:99:AA:BB:CC
*** IP address : 10.10.1.6
*** TCP port: 1025
TCP connection has been established. Now send and receive data.
100/1000 segments sent.
200/1000 segments sent.
300/1000 segments sent.
400/1000 segments sent.
500/1000 segments sent.
600/1000 segments sent.
700/1000 segments sent.
800/1000 segments sent.
900/1000 segments sent.
1000/1000 segments sent.
TCP loop done. Number of bytes received 8960000, number of bytes error 0
```

Figure 21 - FPGA TCP Loop. Client to server transfer, UART screen

--- Note ---

The design does not support the hot-plug functionality. After connecting the two SFP+ ports together, the firmware needs to be restarted.

UDP loop test

In this test we use one Ten-Gigabit TCP/UDP/IP instance to send 1000 UDP packets of size 8960 bytes to the other instance.

Use the **udp_loop** command to begin the test.

When asked for the packet length, input 8960.

When asked for the number of packets to be sent, input 1000.

When asked for the PRBS verification option, input 1 to enable the module.

The screenshot of the UART console is shown in the following figure.

```
Please input packet length (i.e. 8960 (8972 max))
Please input number of packets to be sent (i.e. 100000)
Please input PRBS data integrity verification (0 to disable, 1 to enable)
100 of total 1000 packets sent
200 of total 1000 packets sent
300 of total 1000 packets sent
400 of total 1000 packets sent
500 of total 1000 packets sent
600 of total 1000 packets sent
700 of total 1000 packets sent
800 of total 1000 packets sent
900 of total 1000 packets sent
1000 of total 1000 packets sent
UDP loop done. Number of packets received 1000, number of bytes error 0
```

Figure 22 - UDP Loop test, UART screen

End Of Document.