TOE1G-IP Multisession Reference design manual

Rev1.0 19-May-17

1. Overview

It is recommended to read “dg_toe1gip_refdesign_xilinx_en.pdf” document which is half duplex demo of TOE1G-IP firstly. It will help the user to understand the basic operation of TOE1G-IP.

Multi-session demo is designed by implementing eight TOE1G-IPs to support data transfer by using eight sessions at the same time. Comparing to half duplex demo, 8-to-1 EMAC multiplexer is additional designed to share transmit channel of one EMAC-IP to all TOE1G-IPs. Asynchronous buffer between receive channel of EMAC-IP and TOE1G-IP is designed to convert data in regional clock domain to be global clock domain for easily routing to eight TOE1G-IPs. The buffer size of each TOE1G-IP in multisession demo is reduced from 64Kbyte size to 8Kbyte size to allow user adding more TOE1G-IP in the design without resource limitation. But reducing buffer size will reduce transfer performance.

To run the demo, TestPC must run “tcpdatatest.exe” application which is provided by Design Gateway to send/receive data with FPGA. More details of the demo are described as follows.

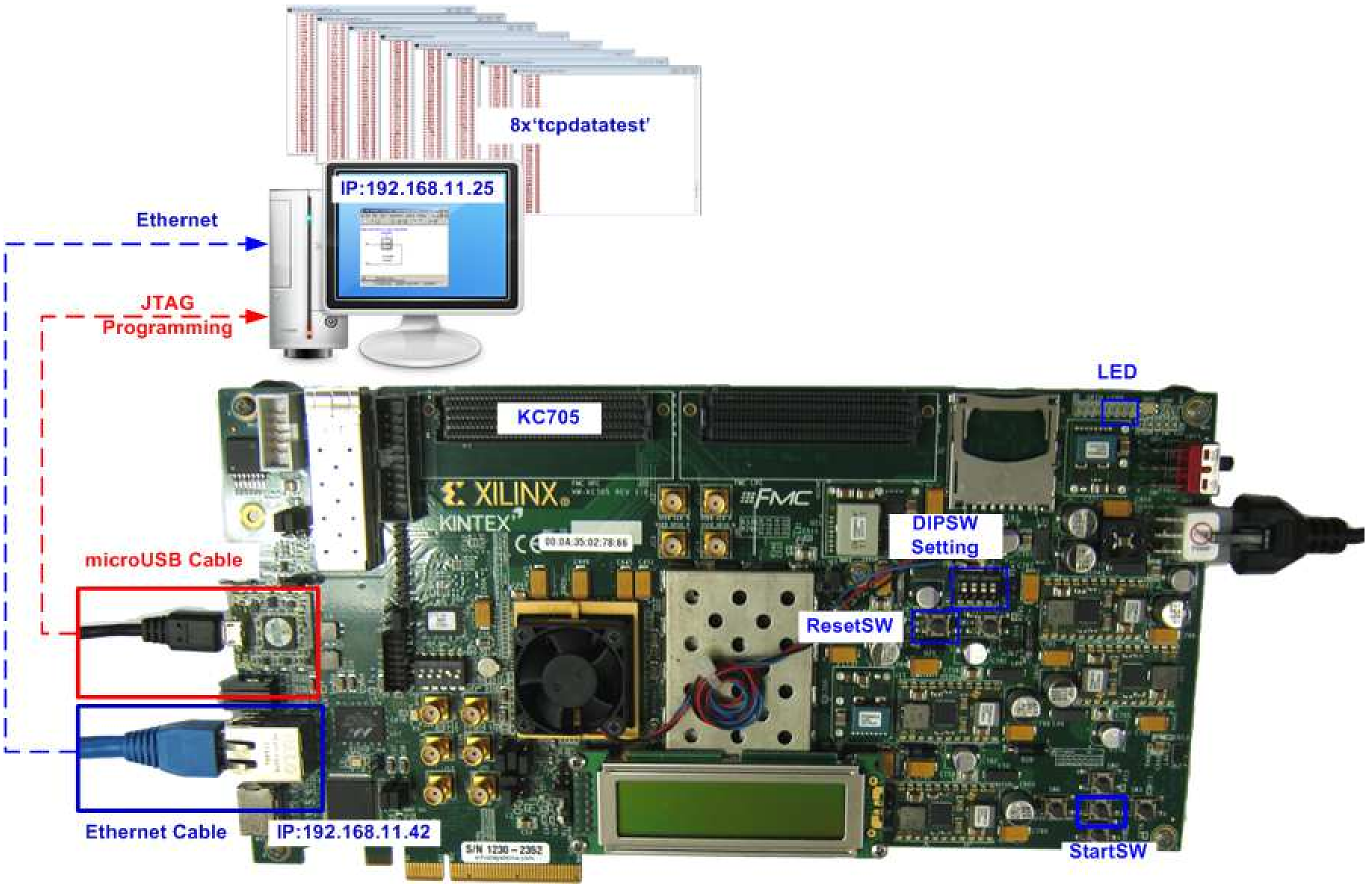


Figure 1 TOE1G-IP Multi-session Demo on FPGA board

2. Hardware description

The hardware design in the reference design is shown in Figure 2. Each UserCtrl is applied to control user interface of each TOE1G-IP independently. Transfer direction of UserCtrl is defined from 4-bit user DIPSW. The port number of each TOE1G-IP will be different to allow many sessions transferring data with PC at the same time. The size of all buffers within each IP is fixed to 8 Kbyte instead of 64 Kbyte like half duplex demo. Transmit data from eight TOE1G-IPs is fed to MacTx1GMux to forward data of each session to TEMAC. Typically, Rx interface of TEMAC can be mapped to TOE1G-IP directly. But in this demo, MacRx1GIF is designed to convert clock domain of data stream output from TEMAC to synchronous with MacTxClk. So, TOE1G-IP in this design will run by using one clock domain.

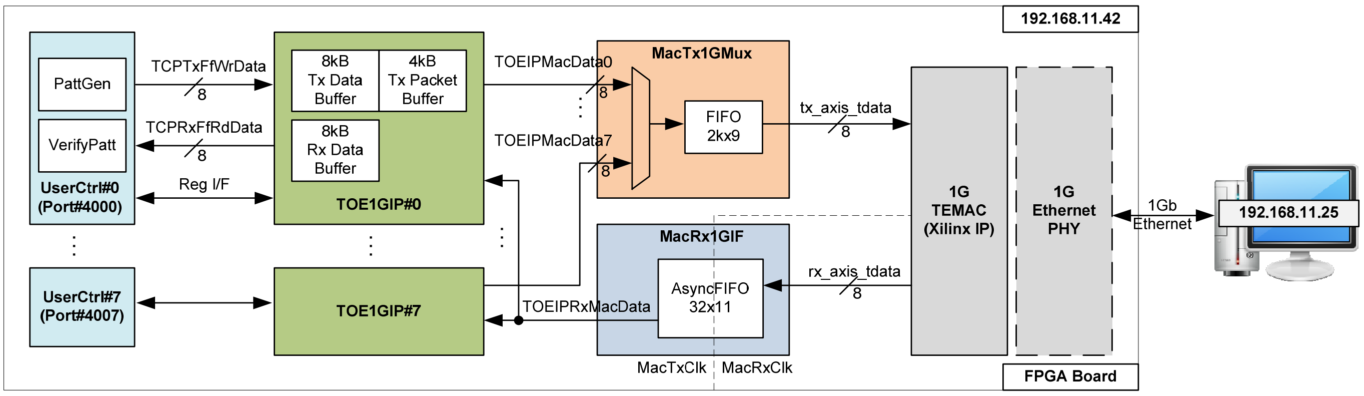


Figure 2 Hardware Architecture in reference design

- 1G TEMAC

This is Tri Mode Ethernet MAC IP core provided by Xilinx. Data rate is fixed to 1Gbps. User can send the request for license evaluation and read more details from following website.
https://www.xilinx.com/products/intellectual-property/temac.html

- MacTx1GMux

This module includes 8-to-1 data multiplexer to select Ethernet data output from eight TOE1G-IPs. FIFO2kx9 is designed to be the buffer to store Ethernet data and last flag. MSB of FIFO counter is monitored by the logic. If remaining size of FIFO is less than 1024, ready flag to TOE1G-IP will be de-asserted.

For read direction, Empty flag is monitored. If data is available in FIFO, the logic will forward all EMAC signals to TEMAC.

Arbiter logic is designed to control the active channel to pass Ethernet packet to TEMAC. The active channel will be switched after complete to transfer current packet and data request in the other channels are asserted. If eight channels are transferred at the same time, the active channel sequence will be CH0 -> CH1 -> CH2 -> … CH6 -> CH7 -> CH0 …

- MacRx1GIF

MacRxClk input to TOE1G-IP is used to synchronous with received data and used to be clock input to Rx data buffer inside the IP. To support multiple TOE1G-IP, MacRxClk should be routed by global clock. RxClk output from TEMAC is regional clock, so MacRx1GIF must be designed to convert Ethernet data stream to synchronous with global clock instead of regional clock. AsyncFIFO32x11 is applied to convert clock domain of data stream.

11-bit data bus is used to store signal output from TEMAC, i.e. 8-bit receive data, 1-bit data valid, 1-bit data last, and 1-bit user signal. FIFO will store 11-bit signal during packet transmission, starting from Start-of-frame (1st data valid in the packet) to End-of-frame (data last is asserted).

For read side, Empty flag is monitored to check that new packet is received. After empty is de-asserted to '0' about 4 clock, the logic inside MacRx1GIF will start to read data from FIFO and then forward to all TOE1G-IP. 4 clocks are used to allow 4-5 data are available in FIFO before forwarding packet. After complete to forward the packet, the data valid/last/user output to TOE1G-IP will be fixed to '0'.

- TOE1G-IP

RAM size within the IP in the reference design is set to be 8 Kbyte. In Test result, test performance of one channel is about 26-36 Mbyte/sec.

- UserCtrl

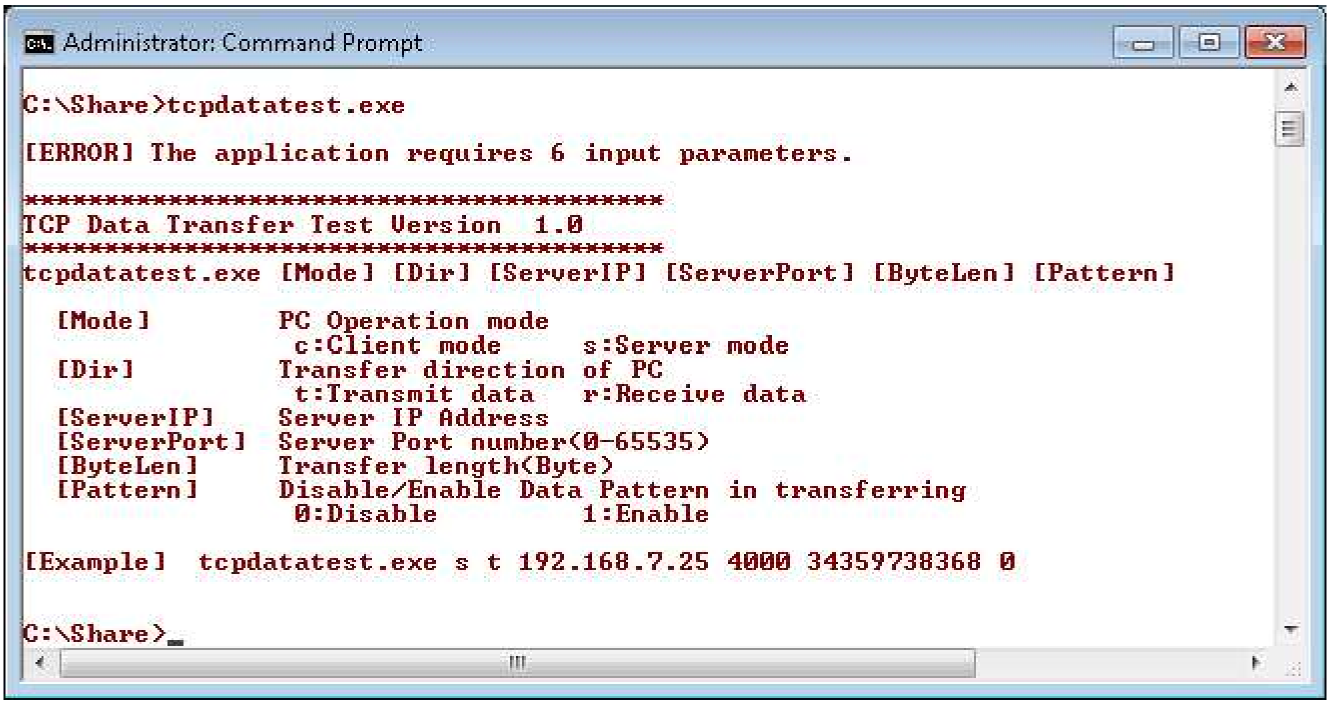
HDL code of this module is almost same as the design of half duplex demo. But Test pattern generator is increased to next value in every clock to operate with "tcpdatatest" application, not every packet like half duplex demo. The details of UserCtrl in half duplex demo are described in http://www.dgway.com/products/IP/TOE1G-IP/dg_toe1gip_refdesign_xilinx_en.pdf.

To support multi-session, the port number of each IP will be different. First port value is 4000 and increased by 1 for the next port (IP#0 uses port#4000, IP#1 uses port#4001, …, and IP#7 uses port#4007).

WIN register of TOE1G-IP is set to 1Kbyte size to allow the IP sending the windows update packet after user logic completes to verify data every 1Kbyte data. Windows update packet is used to show the available size of received buffer inside the IP. So, test application can continue to send the new packet to the IP when available size is enough for new packet.

The transfer direction of UserCtrl is setting from DIPSW. Only four bits of DIPSW are used in the demo to select data direction of eight channels. So, one bit is used to set transfer direction for two sessions.

3. Test Software description


Administrator: Command Prompt

```
C:\Share>tcpdatatest.exe

[ERROR] The application requires 6 input parameters.

****************************************
TCP Data Transfer Test Version  1.0
****************************************
tcpdatatest.exe [Mode] [Dir] [ServerIP] [ServerPort] [ByteLen] [Pattern]

  [Mode]        PC Operation mode
                 c:Client mode      s:Server mode
  [Dir]         Transfer direction of PC
                 t:Transmit data    r:Receive data
  [ServerIP]    Server IP Address
  [ServerPort]  Server Port number(0-65535)
  [ByteLen]     Transfer length(Byte)
  [Pattern]     Disable/Enable Data Pattern in transferring
                 0:Disable          1:Enable

[Example]  tcpdatatest.exe s t 192.168.7.25 4000 34359738368 0


C:\Share>_
```

Figure 3 Test application usage

"tcpdatatest" is designed to run on PC for sending/receiving TCP data through Ethernet for both Server and Client mode. User can input parameter to select transfer direction and the mode. Six parameters are required, i.e.

1) Mode: c – when PC runs in Client mode and FPGA runs in Server mode
2) Dir: t – when PC sends data to FPGA
   r – when PC receives data from FPGA
3) ServerIP: IP address of FPGA when PC runs in client mode. Valid value for hardware in the reference design is 192.168.11.42.
4) ServerPort: Port number of FPGA when PC runs in client mode. Valid value for hardware in the reference design is 4000 – 4007.
5) ByteLen: Total transfer size in byte unit. This input is used in transmit mode only and is ignored in receive mode. In receive mode, application will not know received data size and end operation when connection is destroyed from sender.
6) Pattern:
   0 – Generate dummy data in transmit mode/Disable data verification in receive mode.
   1 – Generate increment data in transmit mode/Enable data verification in receive mode.

4.1 Transmit data mode

Following is the sequence when test application runs in transmit mode.

1) Allocate 1 MB memory to be send buffer.
2) Create socket and set properties of send buffer.
3) Create new connection
   a) For Client mode, create new connection to server by using IP address and port number from user.
4) Generate increment test pattern to send buffer when test pattern is enabled. Skip this step if dummy pattern is selected.
5) Send data out and decrease remaining transfer size.
6) Print total transfer size every second.
7) Run step 4) – 6) in the loop until remaining transfer size is 0.
8) Close socket and print total size and performance.

4.2 Receive data mode

Following is the sequence when test application runs in receive mode.

1) Allocate 1 MB memory to be received buffer.
2) Create socket and set properties of received buffer.
3) Same step as step3) in Transmit data mode.
4) Read data from received buffer and increase total received data size.
5) If verification is enabled, data will be verified with increment pattern and error message will be printed out when data is not correct. Skip this step if data verification is disabled.
6) Print total transfer size every second.
7) Run step 4) – 6) in the loop until connection status is closed.
8) Close socket and print total size and performance.

4. Revision History

Revision	Date	Description
1.0	18-May-17	Initial Release