TOE10GLL IP Core

April 19, 2022 | Product Specification | Rev2.0

Design Gateway Co.,Ltd
E-mail: ip-sales@design-gateway.com
URL: www.design-gateway.com

Core Facts

Provided with Core	
Documentation	Reference Design Manual Demo Instruction Manual
Design File Formats	Encrypted HDL
Instantiation Templates	VHDL
Reference Designs & Application Notes	Vivado Project, See Reference Design Manual
Additional Items	Demo on ZCU102, ZCU106
Support	
Support Provided by Design Gateway Co., Ltd.	

Features

- TCP/IP stack implementation
- Support IPv4 protocol without IP fragmentation
- Ultra-low latency data transmission, measured from start-of-packet to start-of-packet
  - 6.2 ns latency time for transmitting data
  - 46.5 ns latency time for receiving data
- Configured by two data transmission mode: Simple mode and Cut-through mode.
- Support one session per one TOE10GLL IP (using multiple TOE10GLL IPs for multi-sessions)
- Support Server and Client mode (Passive/Active open and close)
- Support ICMP Echo reply (Ping): Up to 118-byte payload data
- Supported payload data size
  - 1-1460 byte per packet for transmitted data (normal frame size, not jumbo frame)
  - 1-16000 byte per packet for received data
- Transmit data buffer size: 4kByte – 64kByte
- User interface: 32-bit data stream interface
- EMAC interface: 32-bit AXI4-stream interface for connecting with DG LL10GEMAC IP or Xilinx 10G/25Gb Ethernet Subsystem
- Individual clock domain for transmit and receive interface at 322.266/312.5 MHz
- Available reference design: 1 and 32-session demo on Xilinx development board (ZCU102/ZCU106)
- Customized service for following features
  - Jumbo frame support
  - Buffer size extension by Window scaling feature
  - Customized user interface

Table 1: Example Implementation Statistics

Family	Example Device	Mode	Fmax (MHz)	CLB Regs	CLB LUTs	CLB	IOB	BRAMTile[2]	Design Tools
Kintex-Ultrascale+	XCKU5PFFVB676-2E	Simple	322.266	3479	3476	677	-	15.5	Vivado2019.1
		Cut-through		3500	3578	706	-	15.5	
Zynq-Ultrascale+	XCZU9EGFFVB1156-2E	Simple	322.266	3479	3488	704	-	15.5	Vivado2019.1
		Cut-through		3500	3575	694	-	15.5	
Virtex-Ultrascale+	XCVU9PFLGA2104-2L-E	Simple	322.266	3479	3485	704	-	15.5	Vivado2019.1
		Cut-through		3500	3579	752	-	15.5	

Notes: 1) Actual logic resource dependent on percentage of unrelated logic
2) Block memory resources are based on 64kByte Tx data buffer size

Applications

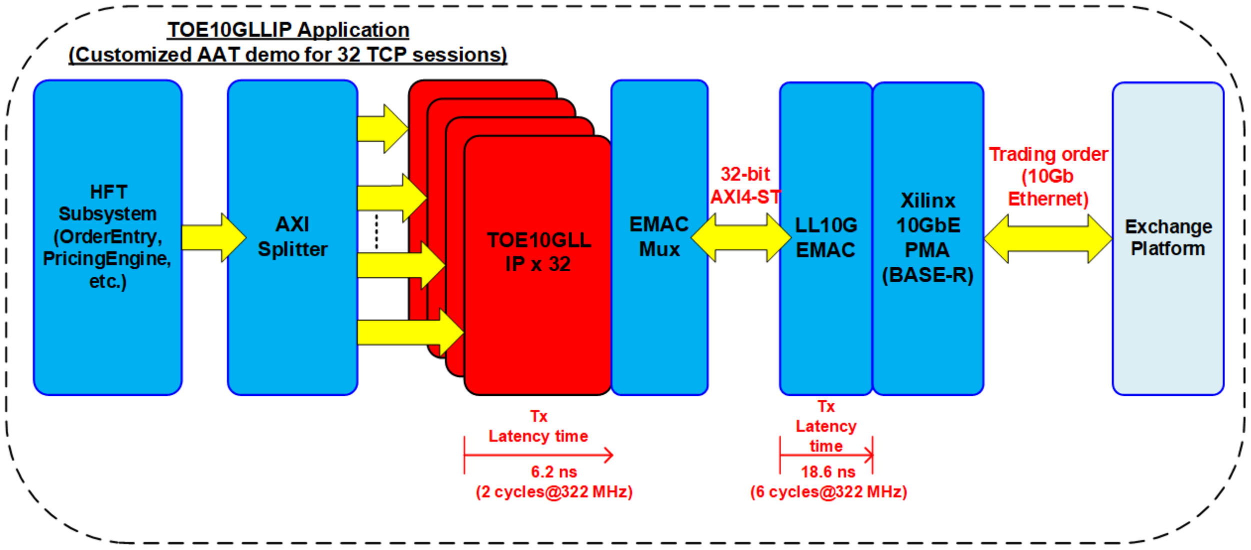

TOE10GLLIP Application
(Customized AAT demo for 32 TCP sessions)
HFT Subsystem (OrderEntry, PricingEngine, etc.)
AXI Splitter
TOE10GLL IP x 32
EMAC Mux
32-bit AXI4-ST
LL10G EMAC
Xilinx 10GbE PMA (BASE-R)
Trading order (10Gb Ethernet)
Exchange Platform
Tx Latency time
6.2 ns (2 cycles@322 MHz)
Tx Latency time
18.6 ns (6 cycles@322 MHz)

Figure 1: TOE10GLL IP Application (Cut-through mode)

Nowadays the network with low latency access is the core for many real-time applications such as High Frequency Trading (HFT), Data center, and Real-time control system in Industrial, Remote surgery, and Automotive products. Using TCP/IP guaratees all data is received by the receiver. TOE10GLL IP is designed to transfer TCP payload data via 10Gb Ethernet (BASE-R) with ultra-low latency time by using TCP/IP protocol.

Figure 1 shows the example application of TOE10GLL IP by integrating the IP to Accelerated Algorithm Trading demo (AAT) for sending sample trading order message via TCP/IP protocol. Two DG low latency IPs are integrated, one LL10GEMAC IP and 32 TOE10GLL IPs. Thirty-two TOE10GLL IPs are applied to support TCP/IP stack for 32 session operating. To plug-in the IPs to AAT system, two modules are additional designed, AXI Splitter and EMACMux. AXI Splitter is designed to transfer 32-session TCP payload data from the application interface (AXI4-ST) to TOE10GLLs. While EMACMux transfers Ethernet packets from 32 TOE10GLL IPs to one LL10GEMAC.

HFT Subsystem includes many blocks for generating sample order message such as OrderEntry and PricingEngine which are designed by HLS. When using DG low latency solution, TCP payload data is forwarded from the application with very low latency time. To transmit Ethernet frame, all low-latency IPs are run at 322.266 MHz. Therefore, the resolution of latency time in Figure 1 is limited by this clock period (3.1 ns).

Test environment

TOE10GLL IP has been launched with a test logic on Xilinx devevelopment board to show latency time and transfer performance. User test logic (DataGen and DataVer) connects to the IP for transferring TCP payload data. Also, Timer is integrated to measure latency time for transmitted packet and received packet. The target device is Test PC which runs the test application for generating and verifying TCP payload data with TOE10GLL IP. There are two configuration modes for TOE10GLL IP, so two test environments are set up to check the operation.

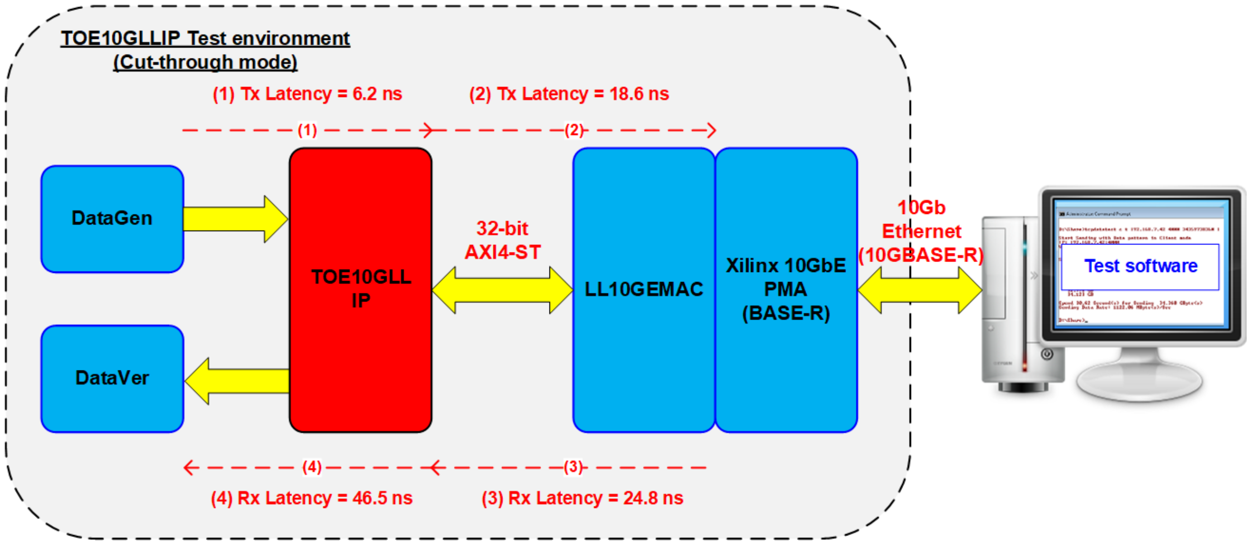


Figure 2: Test system of Cut-through mode

Figure 2 shows test environment of Cut-through mode. The IP connects with LL10GEMAC IP and Xilinx 10GbE PMA (BASE-R) for the lowest latency time solution. The latency time of transmitted path and received path are measured separately. The minimum latency time of transmitted data in TOE10GLL IP and LL10GEMAC is 6.2 ns and 18.6 ns respectively. While the minimum latency time of received data in LL10GEMAC and TOE10GLL IP are 24.8 ns and 46.5 ns respectively.

Note: The minimum latency time is found when there is no pause time from Xilinx 10GbE PMA. The pause time is caused by 64B/66B encoding/decoding logic which de-asserts ready signal for one cycle every 32 cycles.

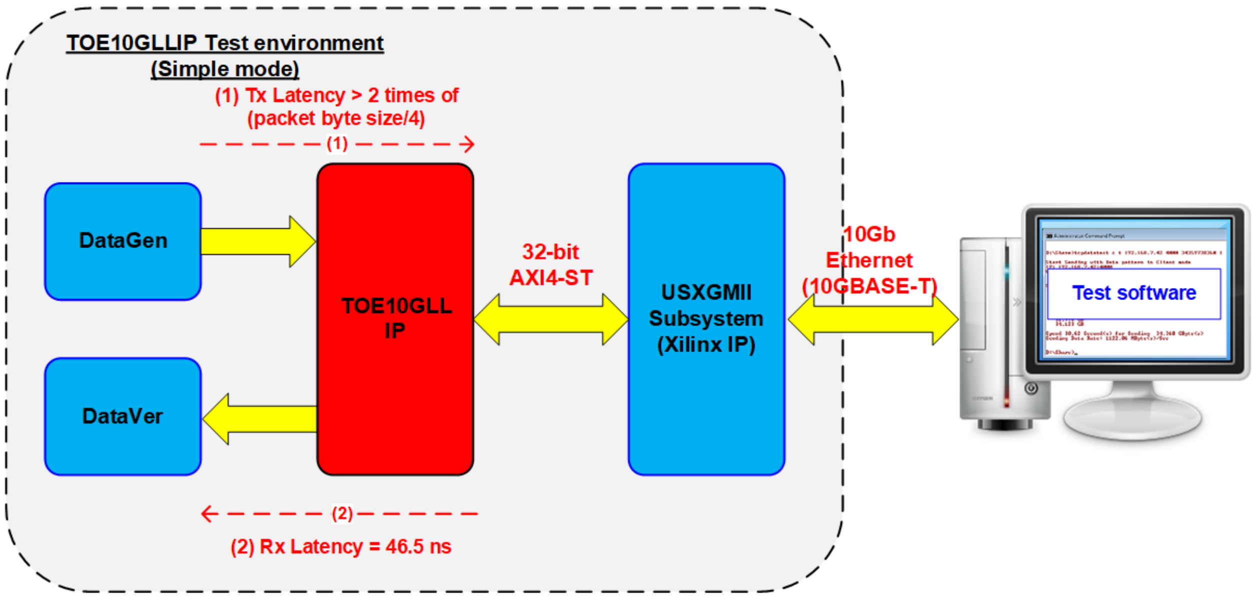

TOE10GLLIP Test environment
(Simple mode)
(1) Tx Latency > 2 times of (packet byte size/4)
(1)
DataGen
DataVer
TOE10GLL IP
32-bit AXI4-ST
USXGMII Subsystem (Xilinx IP)
10Gb Ethernet (10GBASE-T)
Test software
(2)
(2) Rx Latency = 46.5 ns

Figure 3: Test system of Simple mode

Figure 3 shows test environment of Simple mode. EMAC interface of TOE10GLL IP is 32-bit AXI4 stream bus. Therefore, TOE10GLL IP can connect to 10GBASE-R system for low-latency solution or USXGMII Subsystem for 10GBASE-T solution. Comparing to Cut-through mode, Simple mode reduces the number of user interface signals for the simple usage. Therefore, the latency time in transmit path when using Simple mode is increased to be more than two times of (packet byte size/4). While the latency time in receive path is not different.

General Description

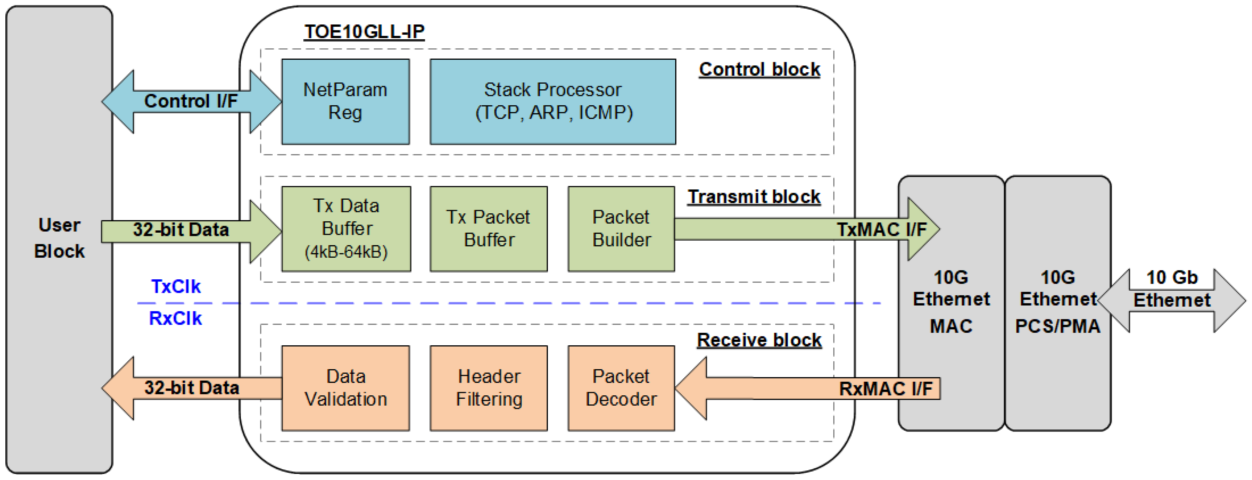


Figure 4: TOE10GLL IP Block diagram

TOE10GLL IP implements Stack processor for operating TCP/IP, ARP, and ICMP by hardwire logic. TOE10GLL IP provides two configuration modes for transmitting the data - Simple mode and Cut-through mode. Simple mode is easy-to-use while Cut-through mode transmits the data at ultra low-latency time. The amount of parameters and the limitation in Simple mode are less than Cut-through mode. In Cut-through mode, user needs to transmit one packet data continuously and the unaligned 32-bit data is allowed for the final data of a packet only. Also, packet size and data checksum are the required parameters to send with the first data of a packet. While Simple mode supports to pause data transmission any time. Also, user can send unaligned 32-bit data at any data in the packet. User does not need to send the packet size and data checksum because they are auto-calculated by TOE10GLL IP.

As shown in Figure 4, there are three user interfaces for TOE10GLL IP, i.e., Control I/F for network parameter assignment, Tx data interface for transmitting the data stream, and Rx data interface for receiving the data stream. The first step to use the IP is to setup the network parameters, assigned by user via Control I/F, for communicating with target device via 10Gb Ethernet. The example of parameters are MAC address, IP address, and Port number. After reset is de-asserted, Control block starts IP initialization to get MAC address of the target device, following the initialization mode. After finishing, the IP is in ready status.

According to TCP/IP standard, the connection must be established, starting by TCP open, before transferring the data. The user can open the port in Active mode (Client) or Passive mode (Server). Similarly, the connection can be terminated by Active mode or Passive mode. User logic can transmit or receive TCP payload data when the connection is still active. TCP payload data is encoded to create Ethernet packet by TOE10GLL IP. On the contrary, Ethernet packet is decoded to extract TCP payload data by TOE10GLL IP.

Besides, TOE10GLL IP implements ARP and ICMP protocol. ARP is applied to translate MAC address from IP address while ICMP is the protocol of Ping command for measuring round-trip time.

To balance the memory resource utilization against transfer performance, Tx data buffer size inside TOE10GLL IP can be configured. Using larger Tx data buffer size, the performance should be better. More details of the hardware inside the IP are described in the next topic.

Functional Description

As shown in Figure 4, TOE10GLL IP consists of three parts, i.e., Control block, Transmit block, and Receive block. Control block and Transmit block are synchronous with TxClk while Receive block is synchronous with RxClk.

Control block

Control block is the interface module with the user to receive the network parameters and the command via Control I/F. Three protocols are handled, TCP/IP, ARP, and ICMP. During transmitting TCP payload data, Control block decodes the amount of data that is successfully transmitted from received packet. After that, the completed size is returned to the user to be the progress of data transmission. Status signals and error signals are also created by Control block when received packet is transferred in the wrong order or the packet includes some errors.

- **NetParam Reg**

This module stores the network parameters set by the user. Most network parameters such as MAC address and IP address are loaded when RstB is de-asserted from '0' to '1' for starting IP initialization. However, the port numbers are loaded when the port is created to allow user to change port number without reset procedure. The network parameters are applied to build and verify the packet header of Ethernet packet.

- **Stack Processor**

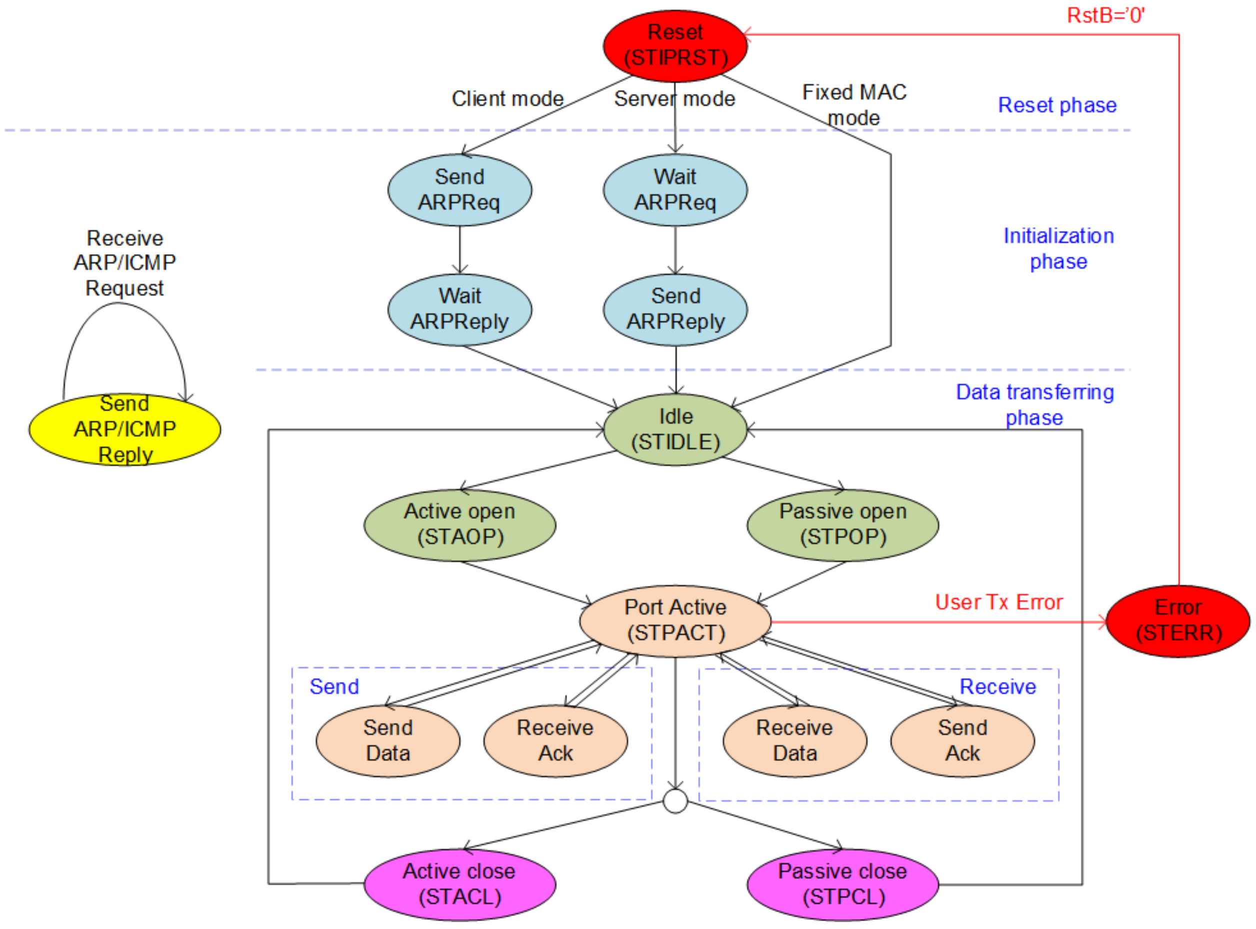


Figure 5: Stack Processor in normal condition

Stack processor is designed to handle TCP/IP packets, ARP request/reply packets, and ICMP Echo request/reply packets. As shown in Figure 5, after finishing reset process, the stack has two operation phases - initialization phase and data transferring phase.

The operation in the initialization phase is to get MAC address of the target device. DstMacMode, set by user, has three values for initializing in Client mode (extracted from ARP reply packet), Server mode (extracted from ARP request packet), or Fixed MAC mode (loaded from the user input). After that, the target MAC address and other network parameters are stored to the internal registers for creating the transmitted packet and validating the received packet. If there is no error detected in this phase, the stack continues to data transferring phase.

The first step before transferring the data is port establishment which can be processed by active mode or passive mode, controlled by TCPCmd from user. After the port is created, Stack processor is in Port Active state (STPACT) which is ready for transferring the data with the target device.

To transfer data, Transmit block and Receive block must be operated for sending the packet and decoding the received packet at the same time. When TOE10GLL IP sends the data, the data packet is created by Transmit block and ACK packet is decoded by Receive block to check the amount of data that is completely received. When TOE10GLL IP receives the data, Receive block extracts the payload data from the valid packet. After that, Transmit block creates ACK packet to confirm the amount of the data that can be completely received.

If the data lost is found, data recovery process is run. During sending data, if the target device detects the lost packet, it returns duplicate ACK to TOE10GLL IP. After that, Stack processsor detects duplicate ACK packet and requests the Transmit block to stop the current data transmission and re-send the lost packet. On the contrary, if Stack processor detects the received packet is lost, the Stack will return duplicate ACK to request the lost data from the target device.

The last step is port termination which can be run in active mode or passive mode. The user sends the command to close the connection by using active close. While the passive close is operated when FIN packet from the target device is received. After the port is terminated, the Stack changes to Idle state (STIDLE) to wait for the new user command.

During sending data, if TOE10GLL IP detects the error packet from user such as wrong packet length, TOE10GLL IP immediately resets the TCP connection and asserts interrupt to user via Control I/F.

After finishing Reset phase, if Receive block detects ARP request or ICMP Echo request, the Stack requests Transmit block to return ARP reply or ICMP Echo reply (when user enables this feature).

Transmit block

There are five packet types which are created by Transmit block, i.e., ARP request packet, ARP reply packet, TCP packet without TCP data, TCP packet with TCP data, and ICMP Echo reply packet. Tx data buffer and Tx packet buffer are applied for creating TCP packet with TCP data. While other packet types are built by Packet builder. Tx data buffer size can be configured by constant value in HDL code. Typically, using bigger buffer size should increase the transmit performance.

- **Tx Data Buffer**

  The depth of Tx data buffer can set by “TxBufBitWidth”, parameter of TOE10GLL IP. The valid value is 10 – 14 for selecting the buffer depth to be 1024 (2^{10}) – 16384 (2^{14}). Tx data buffer is applied to store 32-bit data.

Table 2: TxBufBitWidth parameter

Value	DataBuffer Size
10	1024 x 32-bit (4 kByte)
11	2048 x 32-bit (8 kByte)
12	4096 x 32-bit (16 kByte)
13	8192 x 32-bit (32 kByte)
14	16384 x 32-bit (64 kByte)

According to TCP/IP protocol, the data can be retransmitted if the receiver detects the lost packet and sends the retransmitted request. Therefore, TCP payload data from the user must store in Tx data buffer until the target device accepts the data completely, monitored from received ACK packet. The received ACK packet is decoded to check the amount of data which is completely received. To achieve the best transmit performance, Tx data buffer size must be big to compensate the latency time from processing time of target device and network routing. If the size is too small, all data in the buffer is completely sent by TOE10GLL IP before ACK packet is returned from the target. Therefore, the data transmission must be paused to wait until ACK packet from the target device to clear some data from the buffer. After that, the new payload data from user can be transferred.

- **Tx Packet Buffer**

  This buffer is the temporary transmit buffer to store TCP payload data from Tx data buffer before forwarding to Packet Builder.

- **Packet Builder**

Packet Builder generates the header of every packet which may include payload data. The complete Ethernet packet is created and transmitted to EMAC. More details of each packet type are as follows

TCP packet

Packet Builder loads the network parameters from NetParam Reg module to create the TCP header packet. If the packet includes TCP payload data, then the payload data from user is read. In Cut-through mode, the checksum and the length of payload data are received from user to build the TCP packet with very low latency time. While Simple mode requires the internal logic to calculate TCP checksum and the packet length which needs longer processing time. Therefore, latency time in Simple mode is more than Cut-through mode. Similar to Simple mode, TCP checksum of TCP packet that does not include payload data is calculated by Packet Builder.

ARP packet

In the initialization process, the ARP request is created by using network parameters from NetParam Reg module if the initialization mode is Client mode.

While ARP reply packet is created after receiving ARP request in the initialization phase or data transferring phase. Header data of ARP reply packet consists of the parameters that are extracted from ARP request packet and the parameters loaded from NetParam Reg module.

ICMP packet

Packet Builder creates ICMP Echo reply packet after receiving ICMP Echo request in the initialization phase or data transferring phase. ICMP Echo reply packet consists of the header and the data. Similar to ARP reply, ICMP header loads some parameters from ICMP Echo request and the parameters from NetParam Reg module. ICMP payload data must be extracted from ICMP Echo request to be a part of ICMP Echo reply packet. Small buffer is included to store ICMP payload data. The small bufffer supports up to 118-byte ICMP payload data. Please contact our sales if larger ICMP payload data size is required.

Receive Block

The received packet from EMAC is validated by Receive block. If the network parameters inside the packet header are correct and the packet has TCP payload data, the payload data is extracted and forwarded to the user via Receive interface. If the packet has some errors, the packet will be rejected by the IP except two cases - TCP checksum error or EMAC error. In these two cases, TCP payload data is forwarded to user, but error signal is asserted by Receive block at the end of packet. To achieve the lowest latency time, there is no big buffer inside Receive block.

According to TCP/IP protocol, there is a parameter, called Window size, to show the free space size of the receive buffer to be flow control. The window size of TOE10GLL IP is calculated by 65535 – TCPRxBufWrCnt, the input from the user. The data from the target can be transmitted when the Window size is larger or equal to the amount of transmitted data from the target. Therefore, the best transfer performance in receive path is achieved if TCPRxBufWrCnt value is always equal to 0 which means the user logic is always ready to receive TCP payload data from the target device.

- **Packet Decoder**

  Packet decoder supports four packet types - ARP request, ARP reply, TCP packet, and ICMP Echo request. Other packet types are rejected. Only the supported packet is forwarded to Header filtering.

- **Header Filtering**

  The header filtering verifies the header of the received packet. The packet is valid and forwarded to the next module when the following conditions are met.

  (1) The packet is ARP packet, ICMP packet, or TCP/IPv4 packet without data fragment flag.
  (2) Network parameters are matched to the value in NetParam Reg module, i.e., MAC address, IP address, and Port number (for TCP packet).
  (3) IP header length (IP header length = 20 bytes) and IP checksum are valid for TCP and ICMP packet.
  (4) TCP header length (TCP header length = 20 - 60 bytes) is valid for TCP packet.
  (5) The data pointer decoded by the sequence number is in valid range for TCP packet.
  (6) The acknowledge number is in valid range for TCP packet.

- **Data Validation**

  When the packet is TCP packet with data, TCP payload data is extracted from the packet and forwarded to the user. During forwarding TCP data to the user, Data validation calculates TCP checksum of the packet. If calculated TCP checksum is not equal to the value in the packet header, the error signal is asserted to the user at the end of packet. Therefore, the user logic needs to reject the received data from TOE10GLL IP if the error is found.

  When the packet is ICMP Echo request, ICMP payload data is stored to small buffer for creating the reply packet. Data Validation also calculates ICMP checksum of the packet. If the ICMP checksum is not equal to the value in the packet header, the ICMP Echo request will be rejected. After Data Validation confirms the packet is OK, the ICMP Echo reply will be transmitted by the Transmit block.

10 Gb Ethernet MAC

To achieve the lowest latency, it is recommended to use LL10GEMAC IP from DesignGateway for connecting with TOE10GLL IP. More details of LL10GEMAC IP are described in following website.

https://dgway.com/products/IP/Lowlatency-IP/dg_ll10gemacip_data_sheet_xilinx_en.pdf

Another solution of 10Gb Ethernet MAC is 10G/25G Ethernet Subsystem, provided by Xilinx. The user must select the data bus to be 32-bit interface for the low latency solution. More details of the IP are described in following website.

https://www.xilinx.com/products/intellectual-property/ef-di-25gemac.html

User Logic

Similar to TOE10GLL IP, the user logic must run by using two clock domains – TxClk and RxClk. User logic in the demo system is mainly run on TxClk. Rx data stream interface which is run on RxClk domain must be fed to clock-domain crossing logic for using in TxClk domain.

Control interface is applied to assign the network parameters and the command. They can be designed by using registers or constant value. In the demo system, the simple state machine is designed to assign the sequence to set up each parameters and read the status when running some operation.

Transmit interface uses 32-bit AXI4-stream interface, controlled by valid and ready signal, for sending the data to the target device. However, the details of Transmit interface of TOE10GLL IP for two configuration modes are different.

- Simple mode: User logic transfers 32-bit data without the limitation to pause data transmission or the byte enable assignment of each data. Some logics that are designed by HLS require the feature to send 8-bit, 16-bit, 24-bit, or 32-bit data anytime which can be supported in Simple mode. Please contact our sales for the additional reference design of Simple mode.
- Cut-through mode: User logic may include the small buffer to store one packet data to guarantee that each packet is transmitted continuously. Besides, the small buffer may be applied to store data for calculating data checksum of each packet. The user logic needs to prepare data checksum and packet length to send with the first data of a packet. In Cut-through mode, only byte enable of the final data in each packet can be assigned to be 1-4 byte valid. Other data in each packet must be valid for all 32-bit. TOE10GLL IP provides the example user logic of Cut-through mode in the reference design. Please see more details from following document.

  https://dgway.com/products/IP/Lowlatency-IP/dg_toe10gllip_refdesign_xilinx_en.pdf

To check the progress of data transmission, user logic may include the adder to calculate the sum of the completed transmit length that is returned from TOE10GLL IP.

Receive interface uses 32-bit data stream interface – 32-bit data and valid signal. Ready signal is not applied because user logic must be always ready to receive the data. To control the amount of received data, TCPRxBufWrCnt is applied to show the amount of data that user logic can be accepted. The target device can send the data to the IP when the free space of user logic is enough.

Core I/O Signals

Descriptions of the parameter and I/O signals are provided in Table 3 - Table 5.

Table 3: Core Parameters

Name	Value	Description
TxBufBitWidth	10-14	Setting Tx Data buffer depth which is equal to (2 ** TxBufBitWidth).
IPTrnMode	0-1	Setting the transmission mode of the IP (0: Simple mode, 1: Cut-through mode)

Table 4: User I/O signals

Signal	Dir	Description
General interface		
IPVersion[31:0]	Out	IP version number.
TestPin[31:0]	Out	Reserved to be IP Test point, synchronous in TxClk.
RstB	In	Synchronous reset signal with TxClk. Asserted to '0' to reset the IP and then de-asserted to '1' for starting IP initialization by using the parameters, assigned by the user.
TxClk	In	Clock signal which is synchronous to Tx EMAC interface and Tx user data interface. When running with low-latency 10G EMAC, the frequency is 322.266 MHz.
RxClk	In	Clock signal which is synchronous to Rx EMAC interface and Rx user data interface. When running with low-latency 10G EMAC, the frequency is 322.266 MHz.
Control interface (Synchronous to TxClk)		
Reset phase signals ***The input signals of this group must be valid before RstB is de-asserted from '0' to '1'.***		
ARPICMPEn	In	Enable signal to return ARP reply/ICMP Echo reply after receiving ARP request/ICMP Echo request. '0': Disable, '1': Enable. *Note: This signal is applied when multiple IPs are integrated and shared the same SrcIPAddr. It is recommended to enable ARPICMPEn to one IP and disable to other IPs. Therefore, one ARP reply/ICMP Echo reply packet is generated when ARP/ICMP request is received.*
DstMacMode[1:0]	In	Mode for loading MAC address of the target device "00": Auto-client mode. MAC address is extracted from ARP reply during IP initialization. "01": Auto-server mode. MAC address is extracted from ARP request during IP initialization. "1x": Fixed MAC mode. MAC address is loaded from DstMacAddrIn signal during IP initialization.
InitFinish	Out	IP initialization process is finished. '0': IP is in reset process or initialization process. '1': IP initialization is completed.
SrcMacAddr[47:0]	In	MAC address of TOE10GLL IP.
SrcIPAddr[31:0]	In	IP address of TOE10GLL IP.
DstMacAddrIn[47:0]	In	MAC address of the target device, applied when DstMacMode[1]='1' (Fixed MAC mode)
DstIPAddr[31:0]	In	IP address of the target device.
DstMacAddrOut[47:0]	Out	MAC address of the target device. Valid when InitFinish is asserted to '1'.
TimeOutSet[31:0]	In	Timeout value for waiting the packet returned from the target device before starting recovery process. Valid from 1 – 0xFFFFFFFF. Time unit is equal to TxClk period (3.1 ns for 322.266 MHz).

Signal	Dir	Description
Status monitoring signals		
IPInt	Out	IP timeout interrupt. Assert to high for one cycle when timeout is detected. More details of Interrupt status are monitored from IntStatus[23:0].
IntStatus[31:0]	Out	Interrupt status to show the details of the interrupt. IPInt is asserted to ‘1’ when IntStatus[16:0] is not equal to 0. Other bits are applied for IP monitoring. The signal is valid when IPInt is asserted to ‘1’. [0]-Interrupt when ARP reply packet is not received during IP initialization in Auto-client mode. After that, the IP retries by sending ARP request until ARP reply is received. [1]-Interrupt when SYN|ACK packet is not received during active open operation. After that, the IP retries by sending SYN packet for 16 times. Finally, FIN packet is sent. [2]-Interrupt when ACK packet is not received during passive open operation. After that, the IP retries by sending SYN|ACK packet for 16 times. Finally, FIN packet is sent. [3]-Interrupt when FIN|ACK packet is not received during active close operation. After that, the IP sends RST packet. [4]-Interrupt when ACK packet is not received during passive close operation. After that, the IP retries by sending FIN|ACK packet for 16 times. Finally, RST packet is sent. [5]-Interrupt when ACK packet is not received during sending data operation. After that, the IP retries by retransmitting the previous data packet. [6]-Interrupt when the IP detects data lost during receiving data operation. After that, the IP generates duplicate ACK to request data retransmission. [7]-Interrupt when the IP pauses sending data because the window size of the target device is less than 2Kbyte. After that, the IP transmits reverse data packet to activate the target device retransmitting the ACK packet with the updated windows size. [8]-Interrupt when the window size of the target device during open connection is too less. After that, the IP sends RST packet. [9]-Interrupt when RST packet is received during connection active. After that, the connection is terminated. [10]-Interrupt when the first receiving ACK packet is inapplicable during open connection. After that, the IP sends RST packet. Bit[16:12] – Applied in Cut-through mode only. When these interrupts are asserted, IPState changes to STERR and the IP sends RST packet. [12]-Interrupt when the byte enable of the data which is not the final data of a packet is not equal to 1111b. [13]-Interrupt when the user de-asserts valid signal before the final data of a packet is transmitted. [14]-Interrupt when 16-bit data checksum value of the packet set by the user is incorrect. [15]-Interrupt when the transmit packet length set by the user is more than MSS value on TCPMSS[10:0] port. [16]-Interrupt when the transmit packet length set by the user is not equal to total amount of data of a packet sent via Tx data interface. [21]/[27]-Asserted to ‘1’ when data lost is found during receiving data operation. After that, IPInt and IntStatus[6] are asserted ‘1’. [23]-Interrupt when the IP receives data but TCPRxBufWrCnt shows full status. After that, the IP sends duplicate ACK. *Note: TCPRxBufWrCnt should not be increased more than the amount of received data.* [30]-Asserted to ‘1’ when RST packet is received. [31],[29:28],[26:24]-Internal test status

Signal	Dir	Description
Status monitoring signals		
IPState[4:0]	Out	IP state. “00000” – STIPRST: IP Reset state (RstB is asserted to ‘0’) “00001” – STINIT : Initialization state “00010” – STIDLE : Idle state (No connection) “00011” – STAOP : Active open state “00100” – STPOP : Passive open state “00101” – STPACT : Port Active state (Connection is ON and ready for data transmission) “00110” – STACL : Active close state “00111” – STPCL : Passive close state “01000” – STTRST : TCP Reset state (Send or receive RST packet) “11111” – STERR : Error state (Error from user on Transmit interface during sending operation)
Data transferring phase signals		
TCPCmd[1:0]	In	Command input. “00” – Active close, “01” – Active open, “1X” – Passive open. The signal must be valid when TCPCmdValid is asserted to ‘1’.
TCPCmdValid	In	Command request. Asserted to ‘1’ for one cycle to send the new command while TCPCmdBusy is de-asserted to ‘0’. *Note: The user must check IPState before asserting TCPCmdValid to ‘1’, described as follows.* *(1) Before sending Active or Passive open command, IPState must be equal to STIDLE (00010b).* *If the connection is created completely, IPState will change to STPACT (00101b).* *(2) Before sending Active close command, IPState must be equal to STPACT (00101b).* *If the connection is terminated completely, IPState will change to STIDLE (00010b).*
TCPSrcPort[15:0]	in	Port number of TOE10GLL IP. The signal must be valid when TCPCmdValid is asserted to ‘1’ for Active or Passive open command.
TCPDstPort[15:0]	in	Port number of the target device. The signal must be valid when TCPCmdValid is asserted to ‘1’ for Active open command.
TCPCmdBusy	Out	Command busy. Asserted to ‘1’ when the IP is not ready to receive the new user command request by asserting TCPCmdValid to ‘1’. Busy flag can be asserted by following conditions. *(1) The IP is in reset condition.* *(2) The initialization process is not finished.* *(3) The connection is in terminating.* *(4) The new command is requested by user. TCPCmdBusy is asserted to ‘1’ in the next cycle.* *(5) The IP is transmitting data to the target and the target does not confirm all data acception.*
TCPConnOn	Out	Connection Status. ‘1’: connection is active, ‘0’: no connection.
TCPMSS[10:0]	Out	Maximum segment size in byte unit of this TCP connection. Valid when TCPConnOn is asserted to ‘1’.

Signal	Dir	Description
Tx data interface (Synchronous to TxClk)		
Simple mode/Cut-through mode		
TCPTxPSH	In	PSH flag asserted in TCP header of this packet. In Simple mode, this signal is read in every data (similar to TCPTxByteEn). If this signal is asserted with some valid data, PSH flag of the packet which includes that data will be asserted. In Cut-through mode, this signal is loaded when the first data of a packet is transmitted, similar to TCPTxPkLen and TCPTxCsum.
TCPTxData[31:0]	In	Transmitted TCP data. Valid when TCPTxValid='1'. In Cut-through mode, while transferring the final data of the packet, the unused data that is not enabled by TCPTxByteEn must be filled with zero.
TCPTxValid	In	Asserted to '1' when TCPTxData is valid. In Simple mode, this signal can be de-asserted during the packet transmission. In Cut-through mode, this signal must be asserted to '1' until sending end-of-packet.
TCPTxByteEn[3:0]	In	Byte enable of TCPTxData. Valid when TCPTxValid='1'. In Simple mode, this signal can be equal to four values - 0001b, 0011b, 0111b or 1111b when TCPTxData[7:0], [15:0], [23:0] or [31:0] is valid respectively for any data in the packet. In Cut-through mode, this signal must be equal to 1111b for every data except the final data of a packet. Similar to simplem mode, four value can be set to enable only some bytes for the final data.
TCPTxEOP	In	Asserted to '1' with the final data of the packet on TCPTxData. Valid when TCPTxValid is asserted to '1'. *Note: While total amount of transmitted user data in one packet for Cut-through mode is limited by TCPMSS value, total amount of transmitted user data in one packet for Simple mode must be less than or equal to 1460 bytes.*
TCPTxReady	Out	IP ready signal.'0'-IP is not ready to receive data, '1'-IP is ready to receive data. When TCPTxReady is de-assserted to '0' and TCPTxValid is asserted to '1', TCPTx signals (inputs from user) must hold the same value.
TCPTxCplLen[15:0]	Out	Show the completed data size which the target is completely received in byte unit. Valid when TCPTxCplValid='1'.
TCPTxCplValid	Out	Asserted to '1' for one cycle when TCPTxCplLen is valid.
Cut-through mode		
TCPTxPkLen[10:0]	In	Transmit packet size in byte unit. Valid when the first data of the packet is transmitted. This value must be less than or equal to TCPMSS.
TCPTxCSum[15:0]	In	16-bit data checksum value of the packet. Valid when the first data of the packet is transmitted.

Signal	Dir	Description
Rx data interface (Synchronous to RxClk)		
TCPRxData[31:0]	Out	Received TCP data. Valid when TCPRxValid='1'.
TCPRxValid	Out	Asserted to '1' when TCPRxData is valid.
TCPRxByteEn[3:0]	Out	Byte enable of TCPRxData. Valid when TCPRxValid='1'. During packet transmission, this signal is equal to 1111b for enabling all 32-bit data except the final data of the packet (TCPRxEOP='1') which can be equal to four values - 0001b, 0011b, 0111b, or 1111b when TCPRxData[7:0], [15:0], [23:0], or [31:0] is valid respectively.
TCPRxSOP	Out	Start of packet. Asserted to '1' when sending the first data of the packet. Valid when TCPRxValid='1'.
TCPRxEOP	Out	End of packet. Asserted to '1' when sending the final data of the packet. Valid when TCPRxValid='1'.
TCPRxError[7:0]	Out	Error status of the received packet. Valid at the end of packet (TCPRxEOP='1' and TCPRxValid='1'). [0]-TCP checksum of the packet is error. [1]-Error detected by EMAC. [2]-Error when user sends "Active close" command while the received data is still forwarded to the user. [7-3]-Reserved
TCPRxBufWrCnt[15:0]	In	Data counter of Rx buffer in user logic to calculate free space size which is equal to (65535 – TCPRxBufWrCnt). The free size is applied to be Window size returned to the target device. *Note: After finishing receiving each data packet, TCPRxBufWrCnt must not be increased more than the amount of received data.*

Table 5: EMAC I/O signals

Signal	Dir	Description
Tx MAC interface (Synchronous to TxClk)		
MacTxData[31:0]	Out	Transmitted data. Valid when MacTxValid='1'.
MacTxValid	Out	Asserted to '1' when MacTxData is valid.
MacTxByteEn[3:0]	Out	Byte enable of MacTxData. Four bits are applied for 32-bit data bus. Similar to TCPRxByteEn in Cut-through mode, this signal is equal to 1111b for enabling all 32-bit data in one packet except the final data of the packet (MacTxEOP='1') which can be equal to four values - 0001b, 0011b, 0111b or 1111b when MacTxData[7:0], [15:0], [23:0], or [31:0] is valid respectively.
MacTxSOP	Out	Asserted to '1' when transmitting the first data of the packet. Valid when MacTxValid='1'.
MacTxEOP	Out	Asserted to '1' when transmitting the final data of the packet. Valid when MacTxValid='1'.
MacTxError	Out	Asserted to '1' to cancel the current transmitted packet. Valid when MacTxValid='1'.
MacTxReady	In	Asserted to '1' by EMAC IP when the transmitted data is received correctly. If MacTxReady='0', transmitted data and control signals (MacTxData, MacTxValid, MacTxByteEn, MacTxSOP, and MacTxEOP) must hold the same value until MacTxReady is re-asserted to '1'.
Rx MAC interface (Synchronous to RxClk)		
MacRxData[31:0]	In	Received data. Valid when MacRxValid='1'.
MacRxValid	In	Asserted to '1' when MacRxData is valid.
MacRxEOP	In	Asserted to '1' when receiving the final data of the packet
MacRxError	In	Asserted to '1' at the end of packet when the packet has error. Valid when MacRxValid='1' and MacRxEOP='1'.

Timing Diagram

IP Reset

When the user asserts RstB to '0', all logics inside TOE10GLL IP are in reset condition and IPState shows STIPRST. The IP initialization begins after RstB is de-asserted to '1', as shown in Figure 6.

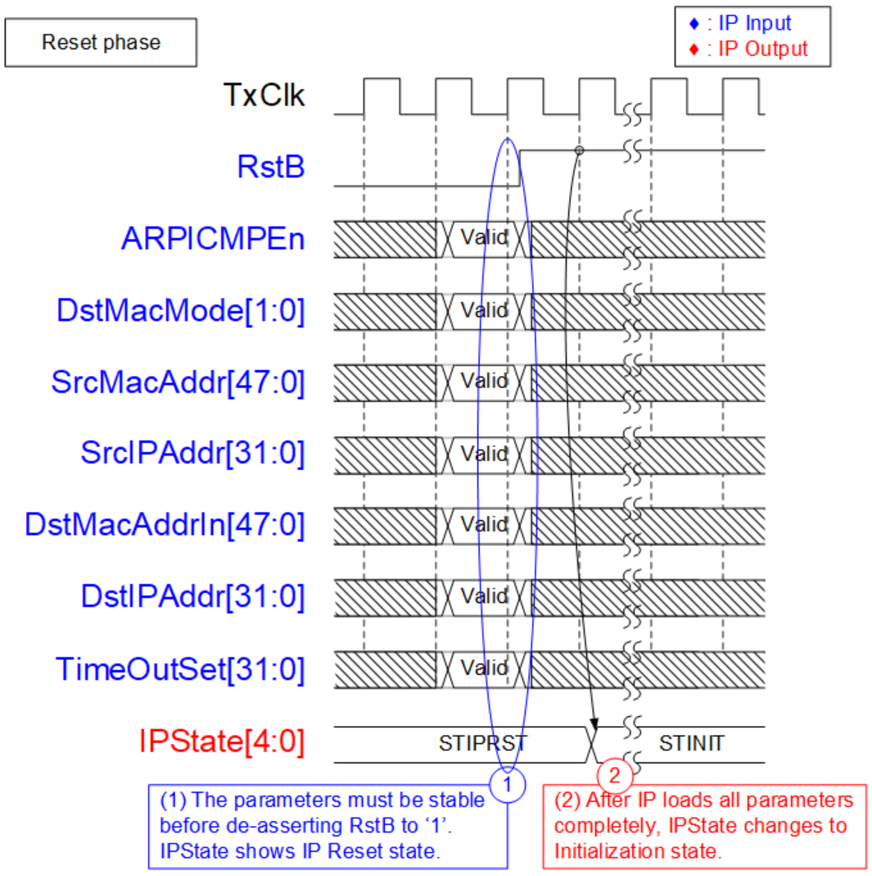

Reset phase
♦ : IP Input
♦ : IP Output
TxClk
RstB
ARPICMPEn
DstMacMode[1:0]
SrcMacAddr[47:0]
SrcIPAddr[31:0]
DstMacAddrIn[47:0]
DstIPAddr[31:0]
TimeOutSet[31:0]
IPState[4:0]
Valid
STIPRST
STINIT
1
2
(1) The parameters must be stable before de-asserting RstB to '1'. IPState shows IP Reset state.
(2) After IP loads all parameters completely, IPState changes to Initialization state.

Figure 6: IP Reset process

(1) Before the user de-asserts RstB to '1', the user must set the valid value of ARP/ICMP enable (ARPICMPEn), initialization mode (DstMacMode), Timeout value (TimeOutSet), and network parameters, i.e., SrcMacAddr, SrcIPAddr, DstMacAddrIn (when using Fixed MAC mode), and DstIPAddr.

(2) After RstB changes from '0' to '1', TOE10GLL loads all input parameters. IPState changes to STINIT when running IP initialization.

The details of each initialization mode are described in more details as follows.

IP Initialization

There are three modes to initailze TOE10GLL IP, set by DstMacMode, i.e., 00b-Client mode, 01b-Server mode, and 10b/11b-Fixed MAC mode. Figure 7 shows timing diagram when initializing in Client mode.

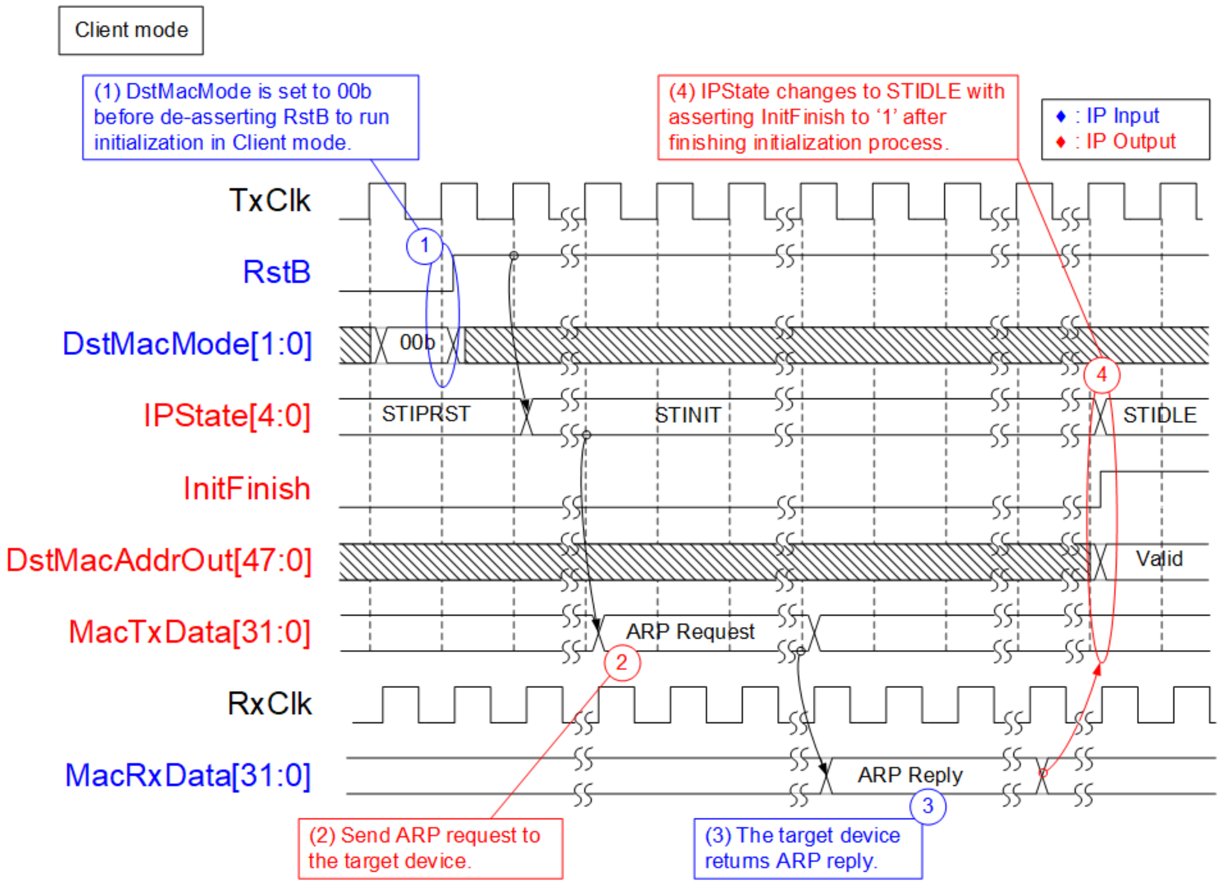

Client mode
(1) DstMacMode is set to 00b before de-asserting RstB to run initialization in Client mode.
(4) IPState changes to STIDLE with asserting InitFinish to '1' after finishing initialization process.
♦ : IP Input
♦ : IP Output
TxClk
RstB
DstMacMode[1:0]
00b
IPState[4:0]
STIPRST
STINIT
STIDLE
InitFinish
DstMacAddrOut[47:0]
Valid
MacTxData[31:0]
ARP Request
RxClk
MacRxData[31:0]
ARP Reply
1
2
3
4
(2) Send ARP request to the target device.
(3) The target device returns ARP reply.

Figure 7: IP Initialization in Client mode

(1) User sets DstMacMode to 00b and then de-asserts RstB to '1' to start IP initialization in Client mode.
(2) IPState changes status from STIPRST to STINIT. After that, TOE10GLL IP sends ARP request packet to the target device.
(3) After the target device detects ARP request, it generates ARP reply to return MAC address.
(4) TOE10GLL IP receives the ARP reply packet and then extracts MAC address of the target device. After finishing the initialization process, IPState changes to STIDLE and InitFinish is asserted to '1. DstMacAddrOut is also valid to show the MAC address of the target which is extracted from ARP reply packet.

Next, timing diagram of TOE10GLL IP when running the initialization in Server mode is displayed in Figure 8.

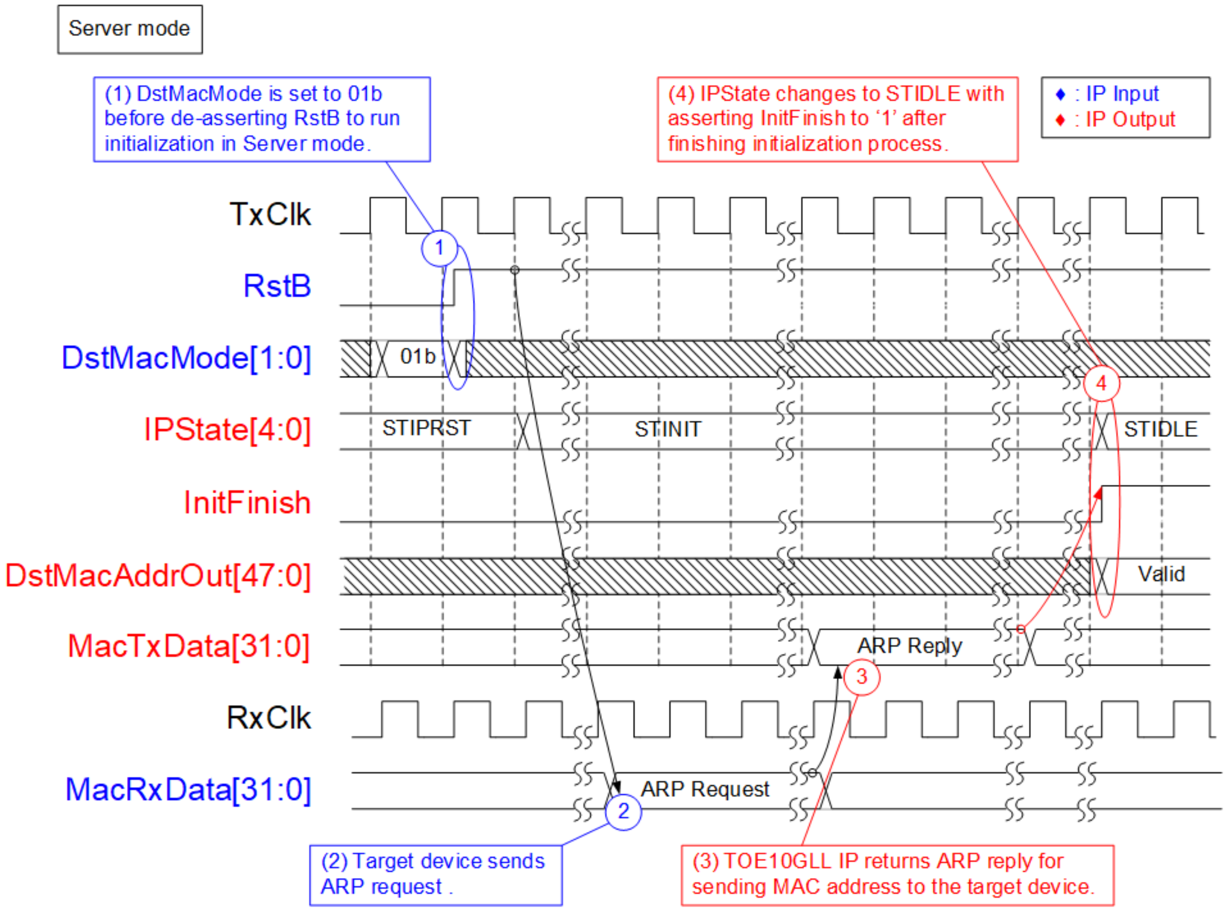

Server mode
(1) DstMacMode is set to 01b before de-asserting RstB to run initialization in Server mode.
(4) IPState changes to STIDLE with asserting InitFinish to '1' after finishing initialization process.
♦ : IP Input
♦ : IP Output
TxClk
RstB
DstMacMode[1:0]
01b
IPState[4:0]
STIPRST
STINIT
STIDLE
InitFinish
DstMacAddrOut[47:0]
Valid
MacTxData[31:0]
ARP Reply
RxClk
MacRxData[31:0]
ARP Request
1
2
3
4
(2) Target device sends ARP request .
(3) TOE10GLL IP returns ARP reply for sending MAC address to the target device.

Figure 8: IP Initialization in Server mode

(1) User sets DstMacMode to 01b and then de-asserts RstB to '1' to start IP initialization in Server mode.
(2) IPState changes status from STIPRST to STINIT. After that, TOE10GLL IP waits until ARP request packet is transmitted from the target device.
(3) TOE10GLL IP extracts MAC address of the target device from ARP request and then returns ARP reply.
(4) After that, the IP finishes the initialization process. IPState changes to STIDLE and InitFinish is asserted to '1. DstMacAddrOut is also valid to show the MAC address of the target which is extracted from ARP request packet.

The last initialization mode is Fixed MAC mode. The target MAC address is defined by DstMacAddrIn signal. Timing diagram when running this mode is displayed in Figure 9.

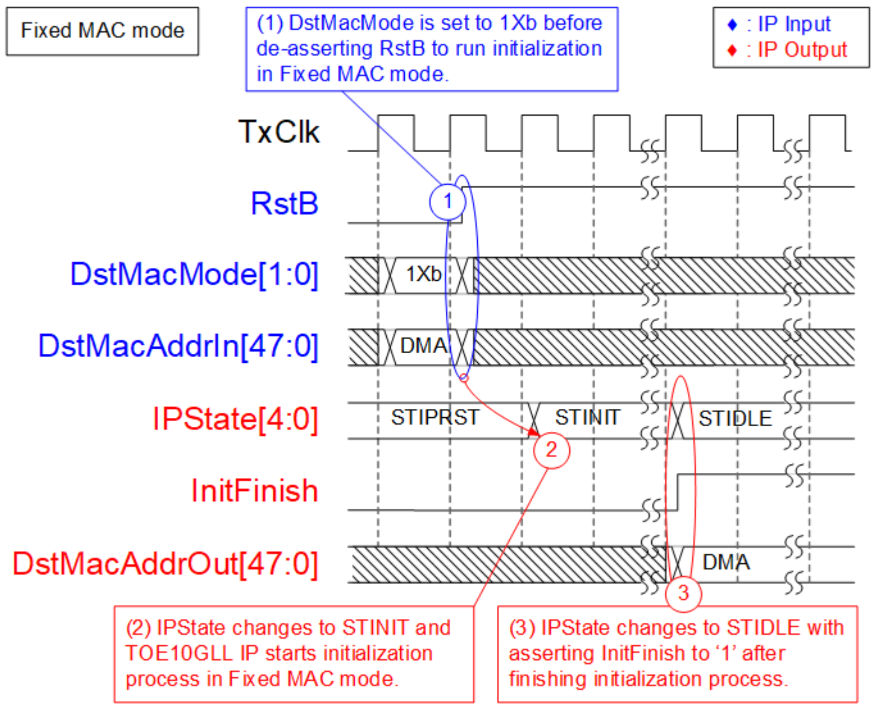

Fixed MAC mode
(1) DstMacMode is set to 1Xb before de-asserting RstB to run initialization in Fixed MAC mode.
♦ : IP Input
♦ : IP Output
TxClk
RstB
DstMacMode[1:0]
1Xb
DstMacAddrIn[47:0]
DMA
IPState[4:0]
STIPRST
STINIT
STIDLE
InitFinish
DstMacAddrOut[47:0]
DMA
1
2
3
(2) IPState changes to STINIT and TOE10GLL IP starts initialization process in Fixed MAC mode.
(3) IPState changes to STIDLE with asserting InitFinish to '1' after finishing initialization process.

Figure 9: IP Initialization in Fixed MAC mode

(1) User sets DstMacMode to 1Xb and then de-asserts RstB to '1' to start IP initialization in Fixed MAC mode.
(2) IPState changes status from STIPRST to STINIT. After that, TOE10GLL IP starts the initialization process.
(3) After finishing the initialization process, IPState changes to STIDLE and InitFinish is asserted to '1. DstMacAddrOut is equal to DstMacAddrIn.

ARP/ICMP Enable

ARPICMPEn is the input by user which is loaded to TOE10GLL IP in Reset phase. This flag is designed for the application that integrates many TOE10GLL IPs that share the same source IP address. Therefore, it should return one ARP reply or ICMP Echo reply when ARP request or ICMP Echo request is received. To generate one reply packet, user enable this flag to one IP and disable this flag to other IPs. Figure 10 shows timing diagram when ARPCMPEn is enabled.

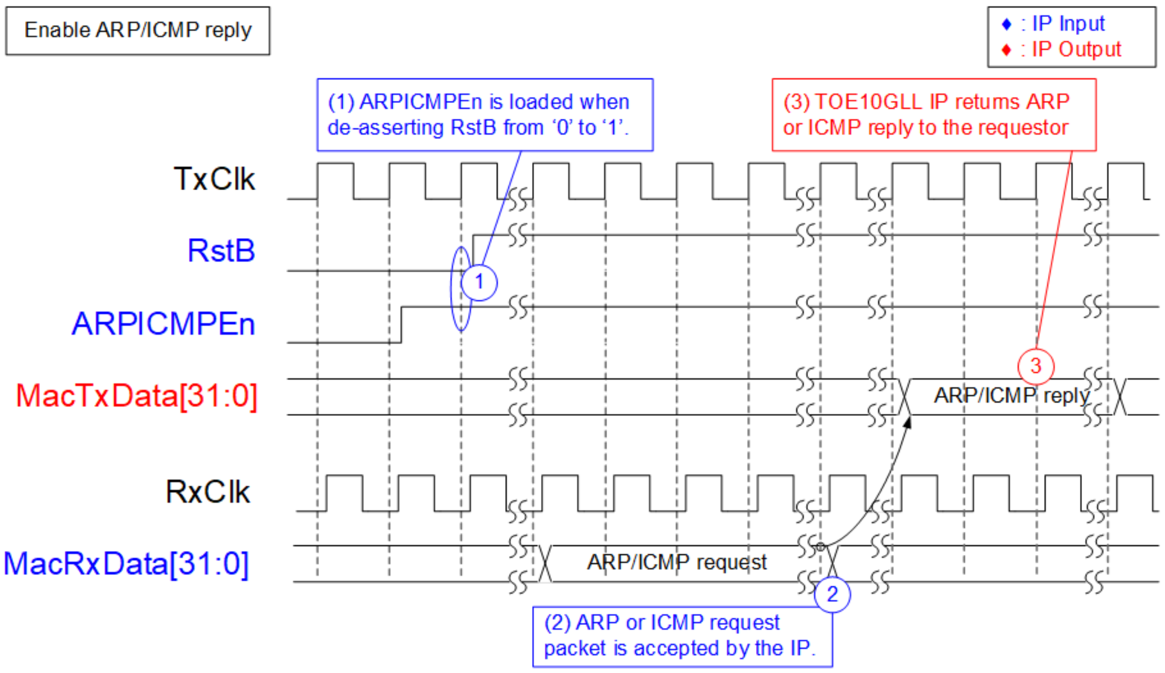

Enable ARP/ICMP reply
♦ : IP Input
♦ : IP Output
(1) ARPICMPEn is loaded when de-asserting RstB from '0' to '1'.
(3) TOE10GLL IP returns ARP or ICMP reply to the requestor
TxClk
RstB
ARPICMPEn
MacTxData[31:0]
ARP/ICMP reply
RxClk
MacRxData[31:0]
ARP/ICMP request
1
2
3
(2) ARP or ICMP request packet is accepted by the IP.

Figure 10: Enable ARP/ICMP reply feature

(1) Before RstB changes from '0' to '1', the IP can be configured to enable the ARP/ICMP reply process by asserting ARPICMPEn to '1'.
(2) If ARPICMPEn='1' when RstB is de-asserted, the received ARP request or ICMP Echo request packet will be accepted. Otherwise, the request packet will be ignored.
(3) The IP creates and returns the ARP reply or ICMP Echo reply packet after accepting the request packet.

Open Connection

According to TCP/IP standard, the connection must be opened as the first step before starting data transmission. There are two modes to open the port by TOE10GLL IP, i.e., Active open (TCPCmd=01b) and Passive open (TCPCmd=1Xb). Before sending open connection command, IPState must be equal to STIDLE and TOE10GLL IP must be ready to accept the new command by checking TCPCmdBusy is equal to '0'. More details of open connection in Active mode and Passive mode are described as follows.

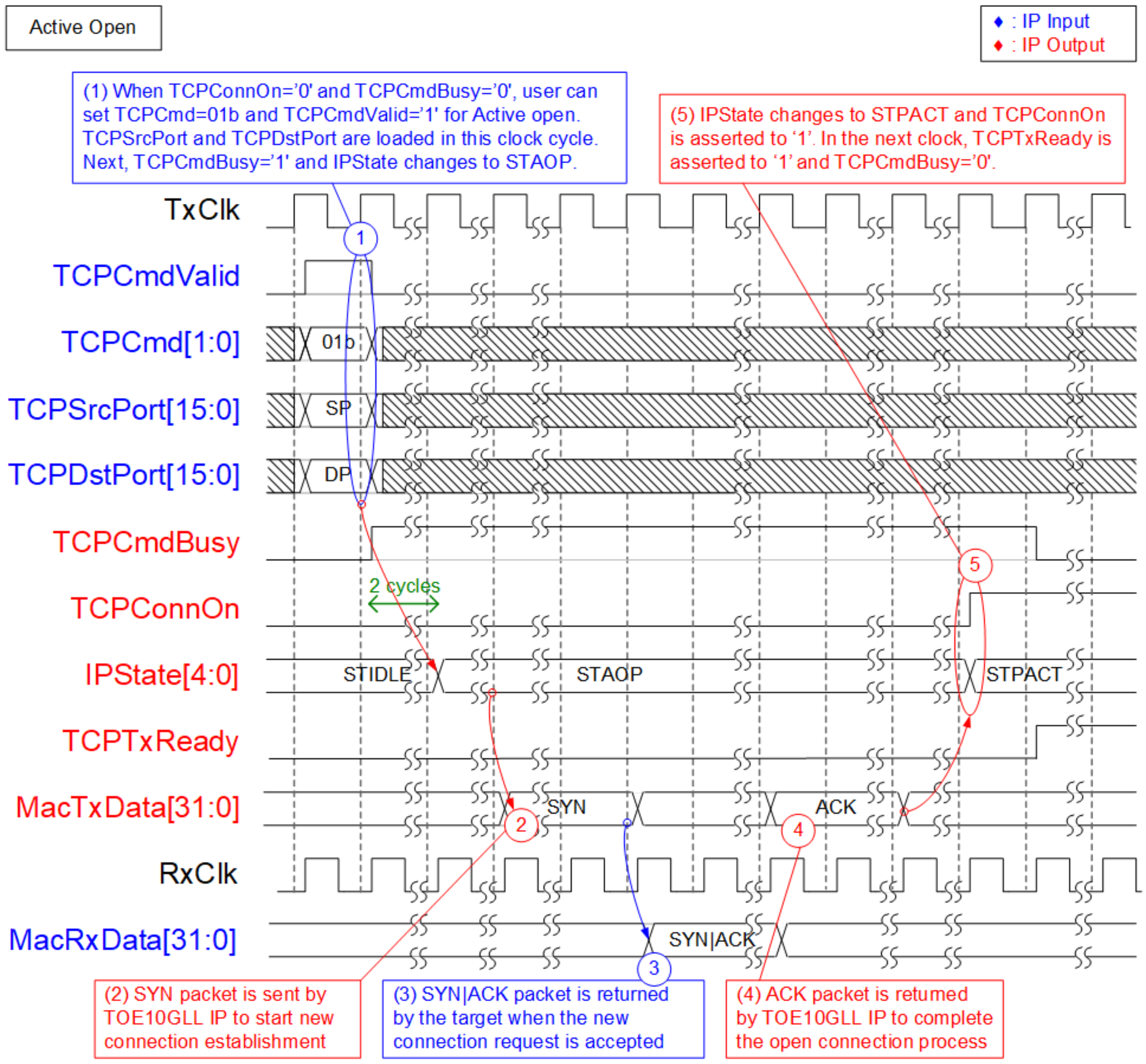

Active Open
♦ : IP Input
♦ : IP Output
(1) When TCPConnOn='0' and TCPCmdBusy='0', user can set TCPCmd=01b and TCPCmdValid='1' for Active open. TCPSrcPort and TCPDstPort are loaded in this clock cycle. Next, TCPCmdBusy='1' and IPState changes to STAOP.
(5) IPState changes to STPACT and TCPConnOn is asserted to '1'. In the next clock, TCPTxReady is asserted to '1' and TCPCmdBusy='0'.
TxClk
TCPCmdValid
TCPCmd[1:0]
01b
TCPSrcPort[15:0]
SP
TCPDstPort[15:0]
DP
TCPCmdBusy
TCPConnOn
2 cycles
IPState[4:0]
STIDLE
STAOP
STPACT
TCPTxReady
MacTxData[31:0]
SYN
ACK
RxClk
MacRxData[31:0]
SYN|ACK
1
2
3
4
5
(2) SYN packet is sent by TOE10GLL IP to start new connection establishment
(3) SYN|ACK packet is returned by the target when the new connection request is accepted
(4) ACK packet is returned by TOE10GLL IP to complete the open connection process

Figure 11: Active open connection

(1) User confirms the IP is ready to receive the new connection establishment command by checking TCPCmdBusy='0' and TCPConnOn='0'. After that, user sets TCPCmd=01b with the valid TCPSrcPort (TOE10GLL Port) and TCPDstPort (Target device Port). Also, user asserts TCPCmdValid to '1' for one cycle to send Active open command.
(2) Next, TCPCmdBusy is assserted to '1' to accept the request. IPState changes from STIDLE to STAOP after TCPCmdBusy is asserted for two clock cycles. Next, TOE10GLL IP sends SYN packet to the target device for creating the new connection.
(3) The target device accepts the new connection request and then returns SYN|ACK packet.
(4) TOE10GLL IP returns ACK packet to complete the new connection establishment.
(5) After finishing port establishment, IPState changes to STPACT and TCPConnOn is asserted to '1'. Next, TCPTxReady is asserted to '1' and TCPCmdBusy is de-asserted to '0'. Finally, it is ready to transfer data in both directions and the new command can be requested.

<u>Note</u>*: If the target device does not return SYN|ACK packet until timeout, TOE10GLL IP re-transmits SYN packet with asserting IPInt signal for 16 times. After the last retry time, the IP sends FIN packet to cancel the operation and IPState changes to STIDLE.*

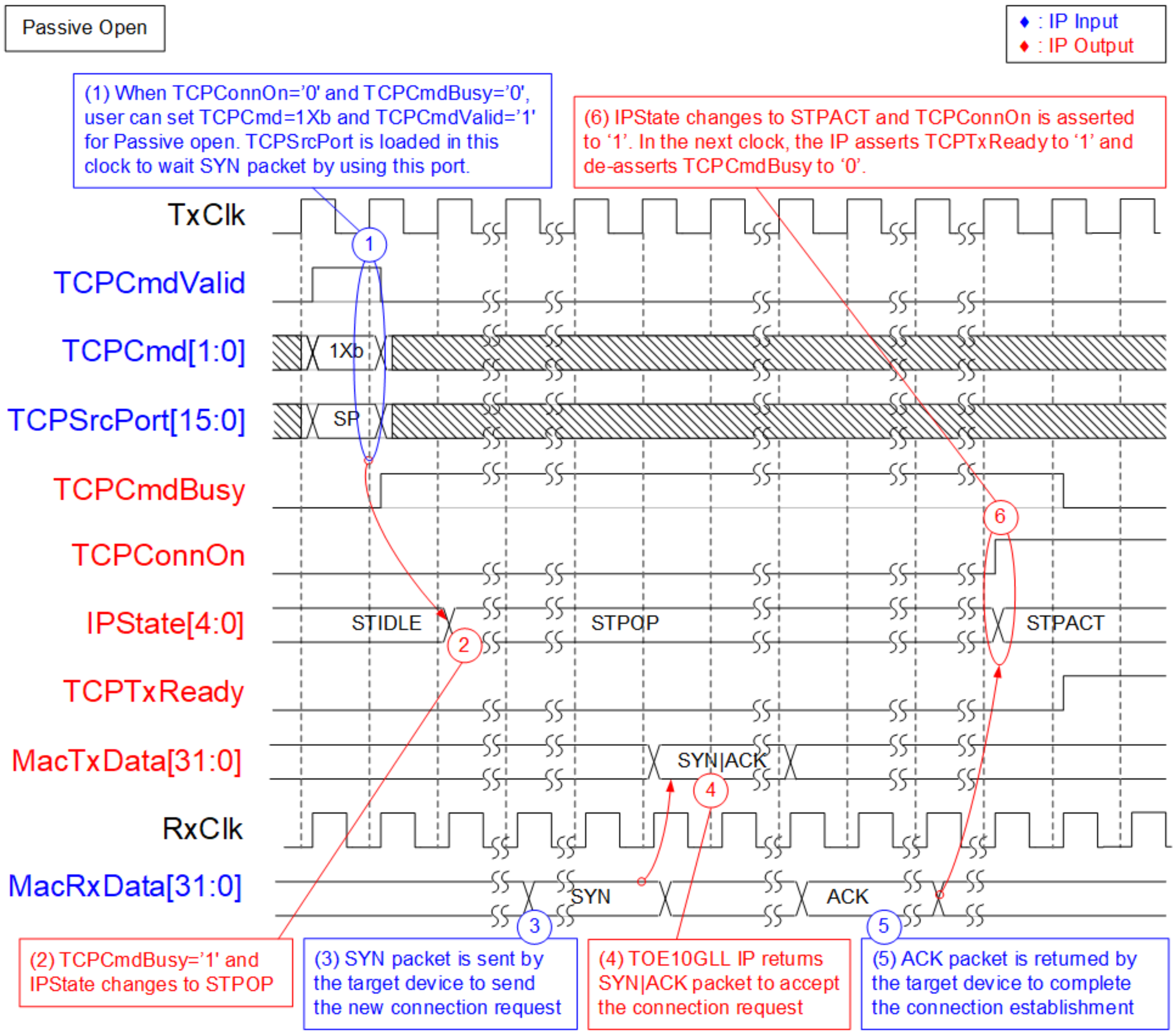

Passive Open
♦ : IP Input
♦ : IP Output
(1) When TCPConnOn='0' and TCPCmdBusy='0', user can set TCPCmd=1Xb and TCPCmdValid='1' for Passive open. TCPSrcPort is loaded in this clock to wait SYN packet by using this port.
(6) IPState changes to STPACT and TCPConnOn is asserted to '1'. In the next clock, the IP asserts TCPTxReady to '1' and de-asserts TCPCmdBusy to '0'.
TxClk
TCPCmdValid
TCPCmd[1:0]
1Xb
TCPSrcPort[15:0]
SP
TCPCmdBusy
TCPConnOn
IPState[4:0]
STIDLE
STPOP
STPACT
TCPTxReady
MacTxData[31:0]
SYN|ACK
RxClk
MacRxData[31:0]
SYN
ACK
1
2
3
4
5
6
(2) TCPCmdBusy='1' and IPState changes to STPOP
(3) SYN packet is sent by the target device to send the new connection request
(4) TOE10GLL IP returns SYN|ACK packet to accept the connection request
(5) ACK packet is returned by the target device to complete the connection establishment

Figure 12: Passive open connection

(1) User confirms the IP is ready to receive the new connection establishment command by checking TCPCmdBusy='0' and TCPConnOn='0'. After that, user sets TCPCmd=1Xb with the valid TCPSrcPort (TOE10GLL Port). Also, user asserts TCPCmdValid to '1' for one cycle to send Passive open command.

(2) Next, TCPCmdBusy is assserted to '1' to accept the request. IPState changes from STIDLE to STPOP in the next clock. After that, TOE10GLL IP waits until SYN packet is sent to request the connection establishement on TCPSrcPort number.

(3) SYN packet sent by the target is received. The IP continues the next step if the parameters in the packet is matched. Otherwise, the request is ignored.

(4) TOE10GLL IP sends SYN|ACK packet to accept the new connection request.

(5) ACK packet is received to complete the connection establishment process.

(6) IPState changes to STPACT and TCPConnOn is asserted to '1'. In the next clock, TCPTxReady is asserted to '1' to allow data transmission with the user and TCPCmdBusy is de-asserted to '0' for receiving the new command. The connection now is ready for data transmission in both directions.

<u>Note</u>*: If the target device does not return ACK packet in step (5) until timeout, TOE10GLL IP re-transmits SYN|ACK packet with asserting IPInt signal for 16 times. After the last retry time, the IP sends FIN packet to cancel the operation and IPState changes to STIDLE.*

Close Connection

When there is no data transmission between two devices, the connection could be terminated. Similar to Open connection process, there are two modes for closing the port by TOE10GLL IP, i.e., Active close by setting TCPCmd=11b and Passive close which is requested by the target device. The details of close connection in each mode are described as follows.

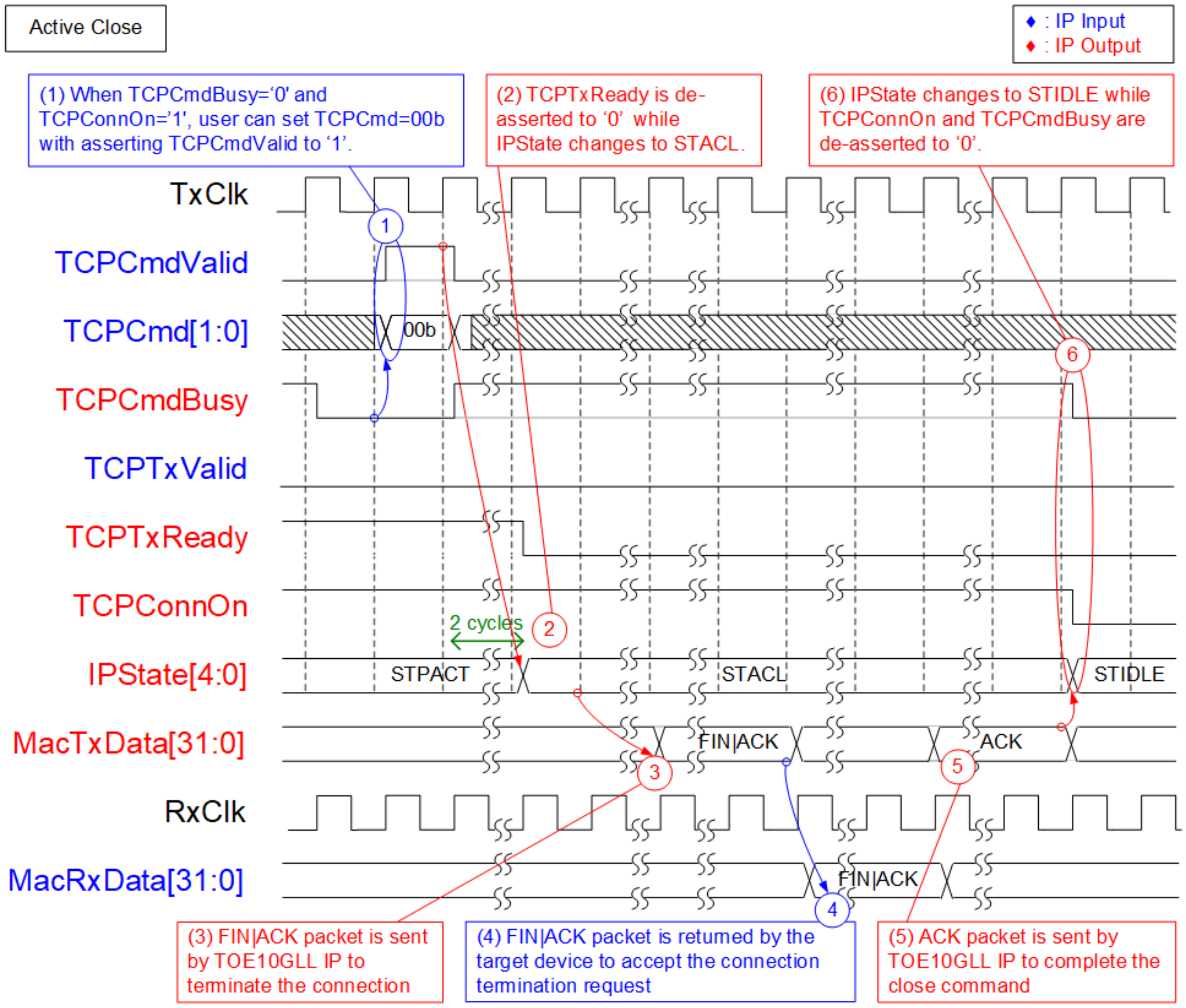

Active Close
♦ : IP Input
♦ : IP Output
(1) When TCPCmdBusy='0' and TCPConnOn='1', user can set TCPCmd=00b with asserting TCPCmdValid to '1'.
(2) TCPTxReady is de-asserted to '0' while IPState changes to STACL.
(6) IPState changes to STIDLE while TCPConnOn and TCPCmdBusy are de-asserted to '0'.
TxClk
TCPCmdValid
TCPCmd[1:0]
00b
TCPCmdBusy
TCPTxValid
TCPTxReady
TCPConnOn
2 cycles
IPState[4:0]
STPACT
STACL
STIDLE
MacTxData[31:0]
FIN|ACK
ACK
RxClk
MacRxData[31:0]
FIN|ACK
1
2
3
4
5
6
(3) FIN|ACK packet is sent by TOE10GLL IP to terminate the connection
(4) FIN|ACK packet is returned by the target device to accept the connection termination request
(5) ACK packet is sent by TOE10GLL IP to complete the close command

Figure 13: Active close connection

(1) Before sending Active close command, user needs to confirm that TOE10GLL IP is ready to receive close command by checking if TCPCmdBusy='0' and TCPConnOn='1'. After that, user asserts TCPCmdValid to '1' and TCPCmd=00b to send Active close command.
Note: When Active close command is requested, user must not send more data to TOE10GLL IP. TCPTxValid must be de-asserted to '0' because the connection for transferring data is in terminating.
(2) Two clock cycles after receiving the user request, IPState changes from STPACT to STACL and TCPTxReady is de-asserted to '0' to stop data transmission from the user.
(3) TOE10GLL IP sends FIN|ACK packet to the target device for terminating the connection.
(4) The target device accepts to terminate the connection and returns FIN|ACK packet.
(5) TOE10GLL IP returns ACK packet to complete the connection termination.
(6) IPState changes to STIDLE while TCPConnOn and TCPCmdBusy are de-asserted to '0'. After that, user can open the new connection by using the different port number, defined by TCPSrcPort and TCPDstPort.

Note: If the target device does not return FIN|ACK packet in step (4) until timeout, TOE10GLL IP sends RST packet to close the connection with asserting IPInt signal once. After that, IPState changes to STIDLE.

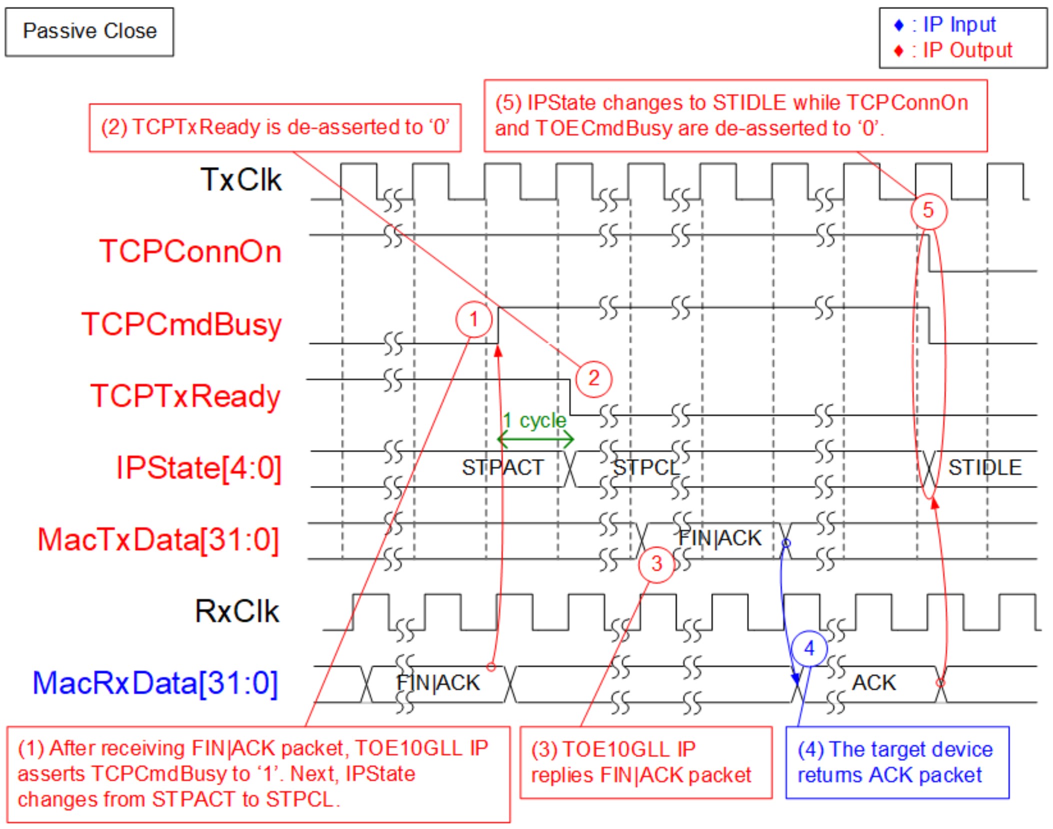

Passive Close
♦ : IP Input
♦ : IP Output
(2) TCPTxReady is de-asserted to '0'
(5) IPState changes to STIDLE while TCPConnOn and TOECmdBusy are de-asserted to '0'.
TxClk
TCPConnOn
TCPCmdBusy
TCPTxReady
IPState[4:0]
MacTxData[31:0]
RxClk
MacRxData[31:0]
1 cycle
STPACT
STPCL
STIDLE
FIN|ACK
FIN|ACK
ACK
1
2
3
4
5
(1) After receiving FIN|ACK packet, TOE10GLL IP asserts TCPCmdBusy to '1'. Next, IPState changes from STPACT to STPCL.
(3) TOE10GLL IP replies FIN|ACK packet
(4) The target device returns ACK packet

Figure 14: Passive close connection

(1) The passive close operation begins when TOE10GLL IP receives FIN|ACK packet from the target device in STPACT state. After that, TCPCmdBusy is asserted to '1' for blocking the command. After TCPCmdBusy is asserted for one clock cycle, IPState changes to STPCL to start passive close operation.

(2) TCPTxReady is de-asserted to '0' to stop data transmission from the user while IPState changes from STPACT to STPCL.
<u>Note</u>: The transmitted data from user should not be remained in TOE10GLL IP when the port termination is requested. The target can not accept more data when the port is terminated. If there is unsent data in TOE10GLL IP, the data will be flushed.

(3) TOE10GLL IP sends FIN|ACK packet to accept the request for terminating the connection.

(4) The target device returns ACK packet to complete the connection termination.

(5) After receiving ACK packet, IPState changes to STIDLE. At the same time, TCPConnOn and TCPCmdBusy are de-asserted to '0'. After that, user can open the new connection by using the different port number, defined by TCPSrcPort and TCPDstPort.

<u>Note</u>: If the target device does not return ACK packet in step (4) until timeout, TOE10GLL IP re-transmits FIN|ACK packet with asserting IPInt signal for 16 times. After the last retry time, the IP sends RST packet to close the connection and IPState changes to STIDLE.

When the data transmission by TCP/IP protocol has critical problem and the recovery process cannot solve the problem, RST packet may be sent to terminate the connection. If TOE10GLL IP receives RST packet and IPState is equal to STPACT, the connection will be closed as shown in Figure 15.

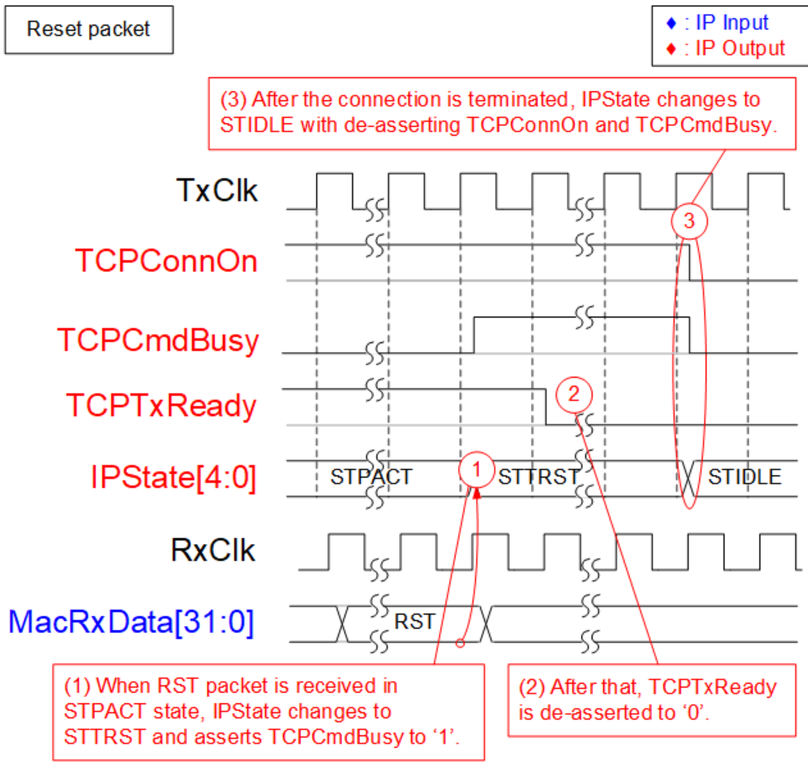

Reset packet
♦ : IP Input
♦ : IP Output
(3) After the connection is terminated, IPState changes to STIDLE with de-asserting TCPConnOn and TCPCmdBusy.
TxClk
TCPConnOn
TCPCmdBusy
TCPTxReady
IPState[4:0]
STPACT
STTRST
STIDLE
RxClk
MacRxData[31:0]
RST
1
2
3
(1) When RST packet is received in STPACT state, IPState changes to STTRST and asserts TCPCmdBusy to '1'.
(2) After that, TCPTxReady is de-asserted to '0'.

Figure 15: Connection closed by RST packet

(1) When RST packet is received from the target device and IPState is equal to STPACT, TOE10GLL IP starts the reset connection process which is similar to closing the connection process. IPState changes to STTRST and TCPCmdBusy is asserted to '1'.
(2) In the next clock cycle, TCPTxReady is de-asserted to '0' to stop data transmission from the user.
(3) There is no packet transmitted by TOE10GLL IP during running reset connection. After finishing, all signals change to no connection status. IPState changes to STIDLE and TCPConnOn is de-asserted to '0'.

Tx data interface

Simple mode and Cut-through mode have its own Transmit data interface requirement. The details of Tx data interface in each transmit modes are described in the following topic.

Tx data interface – Simple mode

Figure 16 shows the example when user sends the data to TOE10GLL-IP in Simple mode for normal condition. User sends two packets, 8-byte packet and 16-byte packet to TOE10GLL-IP. In Simple mode, the packet size is auto-calculated by TOE10GLL-IP to find the maximum size which should be transmitted at that time. After finishing processing, TOE10GLL-IP transmits Ethernet packet which includes user data. When the target device accepts the packet, ACK packet is returned to confirm the amount of received data. TOE10GLL-IP decodes ACK packet to check the amount of completed data and then returns the completed size to user via Cpl interface.

<u>Note</u>: The amount of completed data depends on ACK packet which is returned by the target device. It may not be equal to the payload size of the transmitted packet from TOE10GLL IP. However, the sum value of completed size on Cpl interface is equal to total amount of transmitted data from user.

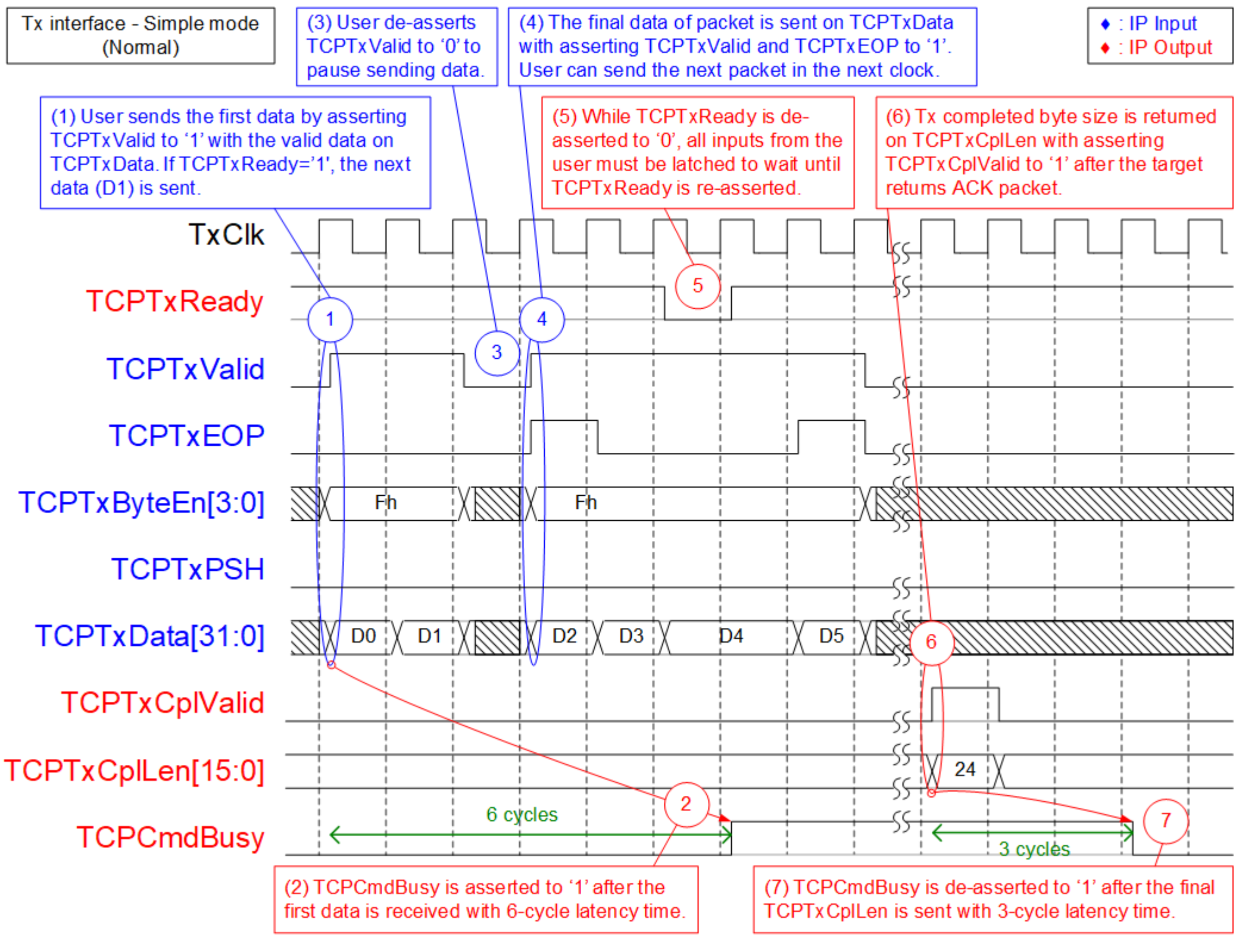

Tx interface - Simple mode (Normal)
(3) User de-asserts TCPTxValid to '0' to pause sending data.
(4) The final data of packet is sent on TCPTxData with asserting TCPTxValid and TCPTxEOP to '1'. User can send the next packet in the next clock.
♦ : IP Input
♦ : IP Output
(1) User sends the first data by asserting TCPTxValid to '1' with the valid data on TCPTxData. If TCPTxReady='1', the next data (D1) is sent.
(5) While TCPTxReady is de-asserted to '0', all inputs from the user must be latched to wait until TCPTxReady is re-asserted.
(6) Tx completed byte size is returned on TCPTxCplLen with asserting TCPTxCplValid to '1' after the target returns ACK packet.
TxClk
TCPTxReady
TCPTxValid
TCPTxEOP
TCPTxByteEn[3:0]
TCPTxPSH
TCPTxData[31:0]
TCPTxCplValid
TCPTxCplLen[15:0]
TCPCmdBusy
Fh
Fh
D0
D1
D2
D3
D4
D5
24
6 cycles
3 cycles
1
2
3
4
5
6
7
(2) TCPCmdBusy is asserted to '1' after the first data is received with 6-cycle latency time.
(7) TCPCmdBusy is de-asserted to '1' after the final TCPTxCplLen is sent with 3-cycle latency time.

Figure 16: Tx data interface of Simple mode (normal condition)

(1) The first data (D0) of the packet is sent on TCPTxData with asserting TCPTxValid to ‘1’. At the same time, TCPTxByteEn and TCPTxPSH must be valid to show the number of valid bytes and PSH flag of each data. Figure 16 shows the basic example in Simple mode. All bytes are valid (TCPTxByteEn=Fh) and PSH is de-asserted to ‘0’ for every transmitted data. If TOE10GLL IP accepts the data, TCPTxReady will be asserted to ‘1’. After that, the next data (D1) can be transmitted.

(2) After the first data is received (TCPTxValid=’1’ and TCPTxReady=’1’) for 6 clock cycles, the IP asserts busy flag (TCPCmdBusy) to ‘1’ to block the new command request from user.

(3) In Simple mode, the user is allowed to pause data transmission by de-asserting TCPTxValid to ‘0’ before the end of packet is transmitted (TCPTxValid=’1’ and TCPTxEOP=’1’).

(4) The final data of the packet is sent on TCPTxData with asserting TCPTxEOP and TCPTxValid to ‘1’. If the final data is accepted by TOE10GLL IP (TCPTxReady=’1’), the user can send the first data of the next packet in the next clock cycle.

(5) When TOE10GLL IP is not ready to receive data by de-asserting TCPTxReady to ‘0’, user needs to hold all input signals (TCPTxValid, TCPTxByteEn, TCPTxPSH, TCPTxData, and TCPTxEOP). The next data can be transmitted after the IP re-asserts TCPTxReady to ‘1’.

(6) TOE10GLL IP creates Ethernet packet that includes TCP payload data to the target. The size of payload data in each Ethernet packet is determined by TOE10GLL IP. It depends on several factors such as MSS (Maximum segment size) and the amount of data from user. After the target returns ACK packet to confirm the amount of received data, TOE10GLL IP decodes the completed data size and then returns the value to user on TCPTxCplLen with asserting TCPTxCplValid to ‘1’. Figure 16 shows the target sends one ACK packet to confirm that all data (D0-D2: 12-byte packet and D3-D5: 12-byte packet) are received. Therefore, TCPTxCplLen is transmitted for one time to show 24-byte data is accepted. However, it is possible that the target generates many ACK packets instead such as two ACKs to confirm 12-byte and 12-byte or 8-byte and 16-byte respectively. TCPTxCplValid is asserted to ‘1’ when each ACK is received. However, total amount of received data returned on TCPTxCplLen for every case (24 bytes, 12 + 12 bytes, or 8 + 16 bytes) is equal to total amount of transmitted data from user (12 + 12 bytes).

(7) After the last amount of received data is transmitted to user (Last TCPTxCplLen) for 3 cycles, the busy flag (TCPCmdBusy) is de-asserted to ‘0’. After that, user can send the next data or Active close command if there is no more data for transmission.

Simple mode supports to receive unaligned 32-bit data from user any time. Therefore, user logic can send 8-bit, 16-bit, 24-bit, or 32-bit data on TCPTxData by controlling TCPTxByteEn to be 0001b, 0011b, 0111b, or 1111b respectively. Also, TCPTxPSH flag is allowed to be asserted to '1' for any valid data. TOE10GLL IP asserts PSH flag of TCP/IP packet when the packet includes user data that must asserts PSH flag. Figure 17 shows timing diagram to send unaligned data and PSH flag in Simple mode.

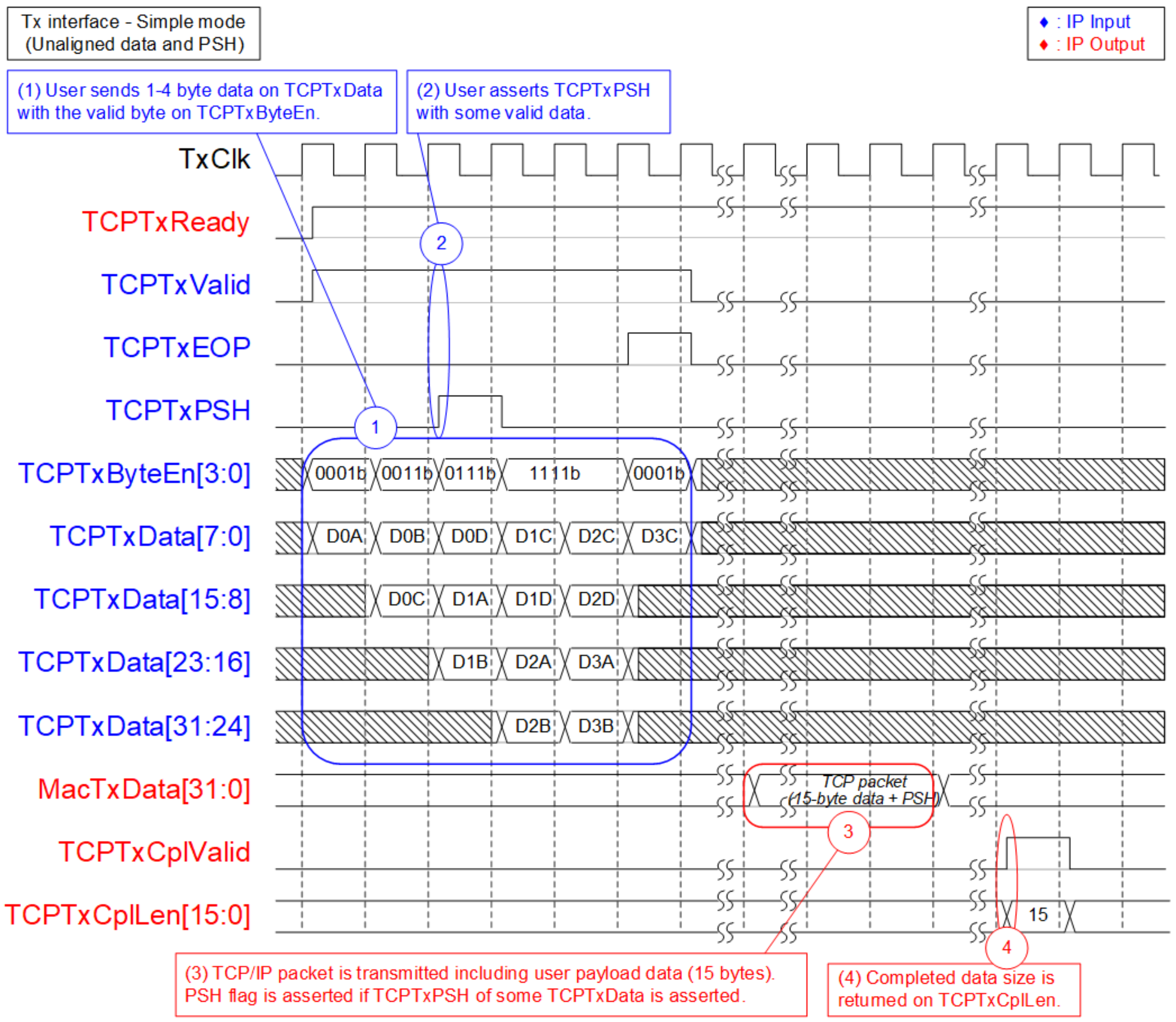

Tx interface - Simple mode (Unaligned data and PSH)
♦ : IP Input
♦ : IP Output
(1) User sends 1-4 byte data on TCPTxData with the valid byte on TCPTxByteEn.
(2) User asserts TCPTxPSH with some valid data.
TxClk
TCPTxReady
TCPTxValid
TCPTxEOP
TCPTxPSH
TCPTxByteEn[3:0]
TCPTxData[7:0]
TCPTxData[15:8]
TCPTxData[23:16]
TCPTxData[31:24]
MacTxData[31:0]
TCPTxCplValid
TCPTxCplLen[15:0]
1
2
3
4
0001b
0011b
0111b
1111b
0001b
D0A
D0B
D0D
D1C
D2C
D3C
D0C
D1A
D1D
D2D
D1B
D2A
D3A
D2B
D3B
TCP packet (15-byte data + PSH)
15
(3) TCP/IP packet is transmitted including user payload data (15 bytes). PSH flag is asserted if TCPTxPSH of some TCPTxData is asserted.
(4) Completed data size is returned on TCPTxCplLen.

Figure 17: Tx data interface of Simple mode (unaligned data and PSH flag)

(1) User sends 8-bit data, 16-bit data, 24-bit data, or 32-bit data on TCPTxData and sets TCPTxByteEn to show how many bytes are valid on each data.
(2) User asserts TCPTxPSH to '1' at the third cycle, so Ethernet packet that includes D0D, D1A, or D1B asserts PSH flag in TCP header.
(3) Assume that TOE10GLL IP determines to create one Ethernet packet for all 15-byte payload data. PSH flag in this packet is asserted to '1'.
(4) After target device receives TCP/IP packet and returns ACK packet to confirm that all data are received. TCPTxCplValid is asserted to '1' and TCPTxCplLen shows total amount of received data that can be transferred completely. Similar to the normal condition, busy flag (TCPCmdBusy) is de-asserted to '0' if all data are accepted completely.

Tx data interface – Cut-through mode

Comparing to Simple mode, two parameters are additional inputs from user in Cut-through mode for ultra low-latency result - the packet size and 16-bit data checksum. Data flow control is similar to Simple mode by using the valid and ready signal. However, TCPTxValid and TCPTxReady are always asserted to '1' during a packet transmission. Figure 18 shows the timing diagram when sending two data packets on Tx data interface in Cut-through mode, 12-byte data packet and 14-byte data packet respectively.

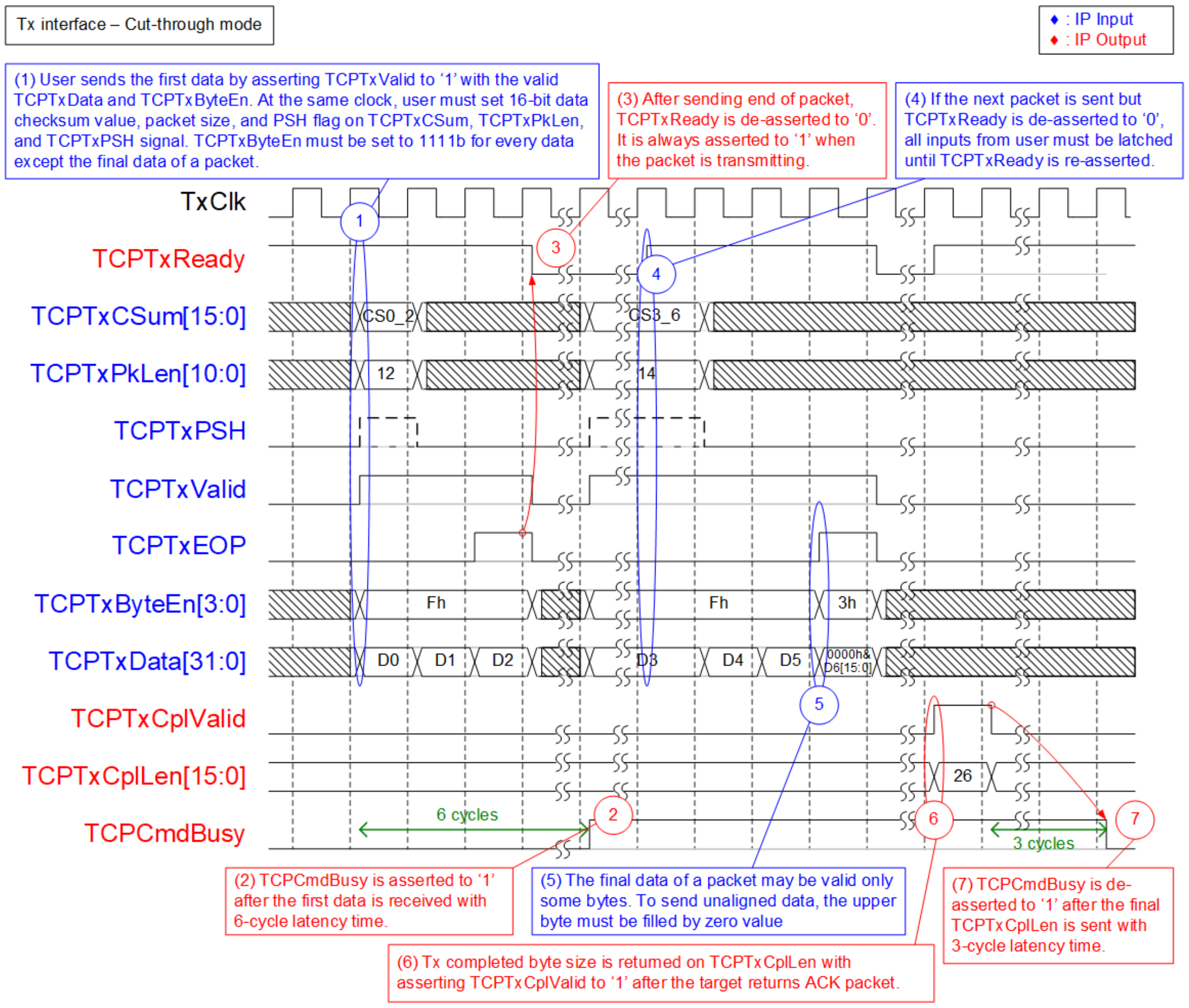

Tx interface – Cut-through mode
♦ : IP Input
♦ : IP Output
(1) User sends the first data by asserting TCPTxValid to '1' with the valid TCPTxData and TCPTxByteEn. At the same clock, user must set 16-bit data checksum value, packet size, and PSH flag on TCPTxCSum, TCPTxPkLen, and TCPTxPSH signal. TCPTxByteEn must be set to 1111b for every data except the final data of a packet.
(3) After sending end of packet, TCPTxReady is de-asserted to '0'. It is always asserted to '1' when the packet is transmitting.
(4) If the next packet is sent but TCPTxReady is de-asserted to '0', all inputs from user must be latched until TCPTxReady is re-asserted.
TxClk
TCPTxReady
TCPTxCSum[15:0]
TCPTxPkLen[10:0]
TCPTxPSH
TCPTxValid
TCPTxEOP
TCPTxByteEn[3:0]
TCPTxData[31:0]
TCPTxCplValid
TCPTxCplLen[15:0]
TCPCmdBusy
CS0_2
CS3_6
12
14
Fh
Fh
3h
D0
D1
D2
D3
D4
D5
0000h&D6[15:0]
26
6 cycles
3 cycles
1
2
3
4
5
6
7
(2) TCPCmdBusy is asserted to '1' after the first data is received with 6-cycle latency time.
(5) The final data of a packet may be valid only some bytes. To send unaligned data, the upper byte must be filled by zero value
(7) TCPCmdBusy is de-asserted to '1' after the final TCPTxCplLen is sent with 3-cycle latency time.
(6) Tx completed byte size is returned on TCPTxCplLen with asserting TCPTxCplValid to '1' after the target returns ACK packet.

Figure 18: Tx data interface of Cut-through mode

(1) The first data (D0) of the packet is sent on TCPTxData with asserting TCPTxValid to '1'. TCPTxByteEn is set to 1111b for transmitting 32-bit data. This signal can be assigned to other value when sending the final data of the packet only (TCPTxEOP='1'). At the same time, user must set the packet size on TCPTxPkLen, 16-bit data checksum on TCPTxCSum, and PSH flag on TCPTxPSH. User must also hold these value until TOE10GLL asserts TCPTxReady to '1' to accept the first data. After that, TCPTxValid and TCPTxReady are asserted to '1' until the end of packet for sending a packet data continuously.

(2) After the first data is accepted by TOE10GLL IP for 6 clock cycles, busy flag (TCPCmdBusy) is asserted to '1' to block the new command request from user.

(3) The final data of the packet is sent on TCPTxData with asserting TCPTxValid and TCPTxEOP to '1'. Total numbers of data must be matched to the value set on TCPTxPkLen at the start of packet. For example, when packet length is equal to 12 bytes, three 32-bit data (D0-D2) are transferred on TCPTxData. After finishing the packet transmission, TCPTxReady is de-asserted to '0' to run packet post-processing.

(4) If user sends the new packet but IP is not ready (TCPTxReady is de-asserted to '0'), all inputs on Tx data interface (TCPTxPkLen, TCPTxCSum, TCPTxPSH, TCPTxValid, TCPTxEOP, TCPTxByteEn, and TCPTxData) must hold the same value until TCPTxReady is re-asserted to '1' to accept the next data packet.

(5) While sending the final data of a packet (TCPTxEOP and TCPTxValid are asserted to '1'), user can send 8-bit, 16-bit, 24-bit, or 32-bit data by assigning TCPTxByteEn to 0001b, 0011b, 0111b, or 1111b respectively. The unused data at the upper byte must be filled by zero value. For example, when packet length is 14 bytes, the 1st data (D3) – the 3rd data (D5) are valid for all 32-bit. The last data (D6) is valid only bit[15:0] because TCPTxByteEn is equal to 0011b. While bit[31:16] of the last data must be filled by 0000h.

(6) Similar to Simple mode, the amount of completely transmitted data is returned on TCPTxCplLen with asserting TCPTxCplValid. The amount of data depends on ACK packet that is returned from the target device. It may return one ACK packet with 26-byte data confirmation or two ACK packets with 12-byte data and 14-byte data respectively. Therefore, TCPTxCplValid may be asserted for one time with 26 bytes or two times with 12 bytes and 14 bytes. Finally, the sum of TCPTxCplLen that is returned to user is equal to the sum of packet length that is transmitted by user.

(7) After the last TCPTxCplLen is sent to user for 3 clock cycles, TCPCmdBusy is de-asserted to '0'. User can send the next data or Active close command.

<u>Note</u>: In Cut-through mode, the packet size generated by TOE10GLL IP is controlled by user inputs, not automatically calculated like Simple mode. Similarly, PSH flag of each TCP/IP packet is controlled by user. Error is returned by IP if 16-bit checksum or packet length from user does not match with the data stream transmitted on TCPTxData.

16-bit data checksum

To transmit data in Cut-through mode, 16-bit data checksum of each data packet must be calculated by user logic. The examples to calculate data checksum are described as follows.

Example1: 12-byte data – 1111_1111h, 2222_2222h, and 3333_3333h.

The 16-bit data checksum = 1111h + 1111h + 2222h + 2222h + 3333h + 3333h = CCCCh.

Example2: 14-byte data – AAAA_AAAAh, BBBB_BBBBh, CCCC_CCCCh, DDDDh.

The 16-bit data checksum = AAAAh + AAAAh + BBBBh + BBBBh + CCCCh + CCCCh + DDDDh

= 5_443Fh = 443Fh + 5h = 4444h

Tx data latency time

The latency time of transmitted data is measured from the first data of packet transmitted by user to the first data of packet transmitted to EMAC. The minimum Tx data latency time is equal to two clock cycles when TOE10GLL IP is configured in Cut-through mode and EMAC and TOE10GLL IP are ready (MacTxReady='1' and TCPTxReady='1'). When TxClk frequency is equal to 322.265625 MHz, the minimum Tx latency time is 6.2 ns, as shown in Figure 19.

Note: Latency time in Simple mode is more than Cut-through mode because it needs to calculate data checksum and re-align payload data. Tx data latency time in Simple mode is more than two times of (payload data size of transmitted packet in byte unit/4). Transmitted packet size in Simple mode is auto-calculated by TOE10GLL IP, so user cannot estimate the latency time in Simple mode.

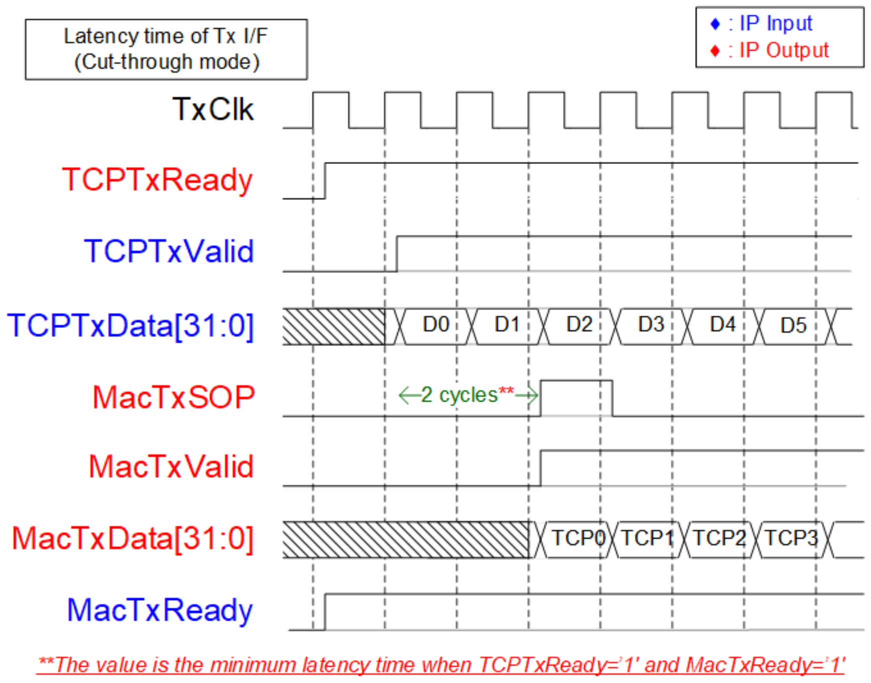
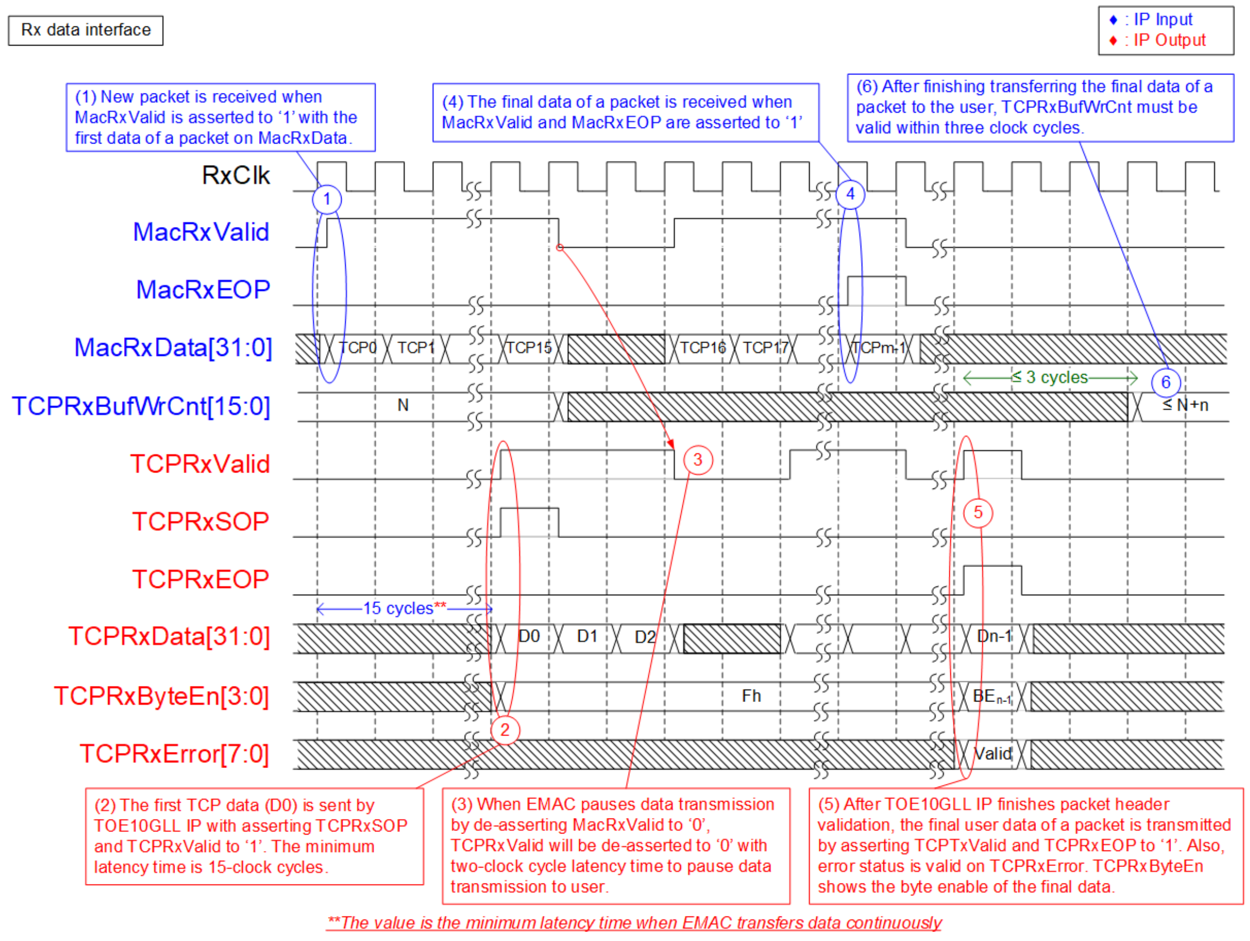


Figure 19: Tx latency time in Cut-through mode

Rx data interface

When the packet is received from EMAC, TOE10GLL IP compares the network parameters of a packet with the set value by the user. If the parameters are correct, only TCP payload data of a packet is extracted and forwarded to the user logic. The minimum latency time of Rx data interface, measured from the first data of EMAC I/F to the first data of User I/F, is equal to 15 clock cycles or 46.5 ns @ 322.265625 MHz. Latency time is increased if EMAC pauses data transmission by de-asserting MacRxValid to ‘0’. Figure 20 shows timing diagram of Rx data interface when receiving the packet from EMAC.



(1) New packet is received when MacRxValid is asserted to ‘1’ with the first data of a packet on MacRxData.

(4) The final data of a packet is received when MacRxValid and MacRxEOP are asserted to ‘1’

(6) After finishing transferring the final data of a packet to the user, TCPRxBufWrCnt must be valid within three clock cycles.

(2) The first TCP data (D0) is sent by TOE10GLL IP with asserting TCPRxSOP and TCPRxValid to ‘1’. The minimum latency time is 15-clock cycles.

(3) When EMAC pauses data transmission by de-asserting MacRxValid to ‘0’, TCPRxValid will be de-asserted to ‘0’ with two-clock cycle latency time to pause data transmission to user.

(5) After TOE10GLL IP finishes packet header validation, the final user data of a packet is transmitted by asserting TCPTxValid and TCPRxEOP to ‘1’. Also, error status is valid on TCPRxError. TCPRxByteEn shows the byte enable of the final data.

***The value is the minimum latency time when EMAC transfers data continuously*

Figure 20: Rx data interface

(1) EMAC sends the first data of a packet (TCP0) to TOE10GLL by asserting MacRxValid to ‘1’ with the valid data on MacRxData.

(2) TOE10GLL IP verifies the network parameters in the header which must be matched to the set value from the user. If the packet is valid, TCP payload data is extracted and forwarded to the user with 15-clock cycle latency time as minimum value.
<u>Note</u>*: 15-clock cycle is the minimum latency time when EMAC sends data continuously by asserting MacRxValid to ‘1’. Latency time is increased if EMAC de-asserts MacRxValid to ‘0’ to pause data transmission.*

(3) When EMAC pauses data transmission by de-asserting MacRxValid to ‘0’, TOE10GLL IP also de-asserts TCPRxValid to ‘0’ to pause data transmission to the user logic with two-clock cycle latency time.

(4) EMAC sends the final data of a packet by asserting MacRxValid and MacRxEOP to ‘1’.

(5) After finishing verifying the header and data in the packet, TOE10GLL IP sends the final data to user by asserting TCPRxValid and TCPRxEOP to ‘1’. At the same time, Error status is valid on TCPRxError and TCPRxByteEn shows the amount of valid byte for the final data which may be not equal to all 1111b. If some errors are detected (TCPRxError is not equal to 00h), the user logic must ignore this packet. If the error can be recovered, the next packet will be the retransmitted packet from data recovery process.

(6) According to TCP/IP standard, the sender must check the free space size of the receiver’s buffer before sending the data. TCPRxBufWrCnt, input from the user, is applied to calculate Window size value returned to the target device for data flow control. Window size is the free spae size which is equal to 65565 – TCPRxBufWrCnt. TCPRxBufWrCnt is not applied when the received packet is transmitting to user. After the final data of the packet is transmitted to user, TCPRxBufWrCnt must be valid within three clock cycles, as shown in Figure 20.
<u>Note</u>*:*
- *Please be caution that the value of TCPRxBufWrCnt after finishing receiving a packet that includes n-byte TCP payload data must not be increased more than n bytes. For example, TCPRxBufWrCnt before receiving a packet is equal to N. After receiving TCP packet which includes n-byte TCP payload data, TCPRxBufWrCnt value must not be more than N (old value) + n (payload size).*
- *If the user logic is always ready to receive data, user can set zero value to TCPRxBufWrCnt by using constant value.*

Error

The examples of error situation when running the IP core are shown in this topic. The error can be found in transmit operation when user sends incorrect inputs to TOE10GLL IP that is configured in Cut-through mode. Also, the error can be found in receive operation when the received packet is corrupt. More details of each error are described as follows.

Tx Error (Cut-through mode only)

When user sends the inputs that are not in agreement to the IP such as wrong packet size or incorrect 16-bit data checksum, IPInt is asserted to '1' and the interrupt status is shown on IntStatus[16:12]. After that, IPState changes to STERR to send RST packet to the target device for disconnecting. Finally, IP state returns to STIDLE, as shown in Figure 21.

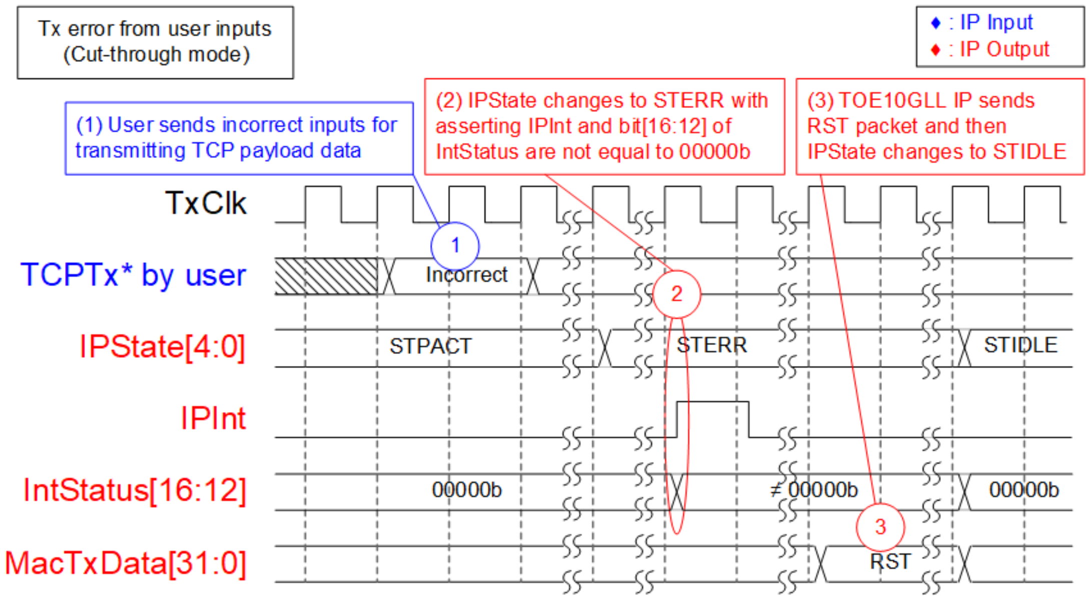

Tx error from user inputs (Cut-through mode)
♦ : IP Input
♦ : IP Output
(1) User sends incorrect inputs for transmitting TCP payload data
(2) IPState changes to STERR with asserting IPInt and bit[16:12] of IntStatus are not equal to 00000b
(3) TOE10GLL IP sends RST packet and then IPState changes to STIDLE
TxClk
TCPTx* by user
IPState[4:0]
IPInt
IntStatus[16:12]
MacTxData[31:0]
1
Incorrect
2
STPACT
STERR
STIDLE
00000b
≠ 00000b
00000b
3
RST

Figure 21: Tx error from user inputs in Cut-through mode

The details of each Tx error are shown as follows.

Figure 22 shows timing diagram of error when transmit packet size is set to 20 bytes, but the user sends only 16-byte data. After that, TCPTxReady is de-asserted to ‘0’ to block transmitted data from user. Finally, IPInt is asserted to ‘1’ by IntStatus[16].

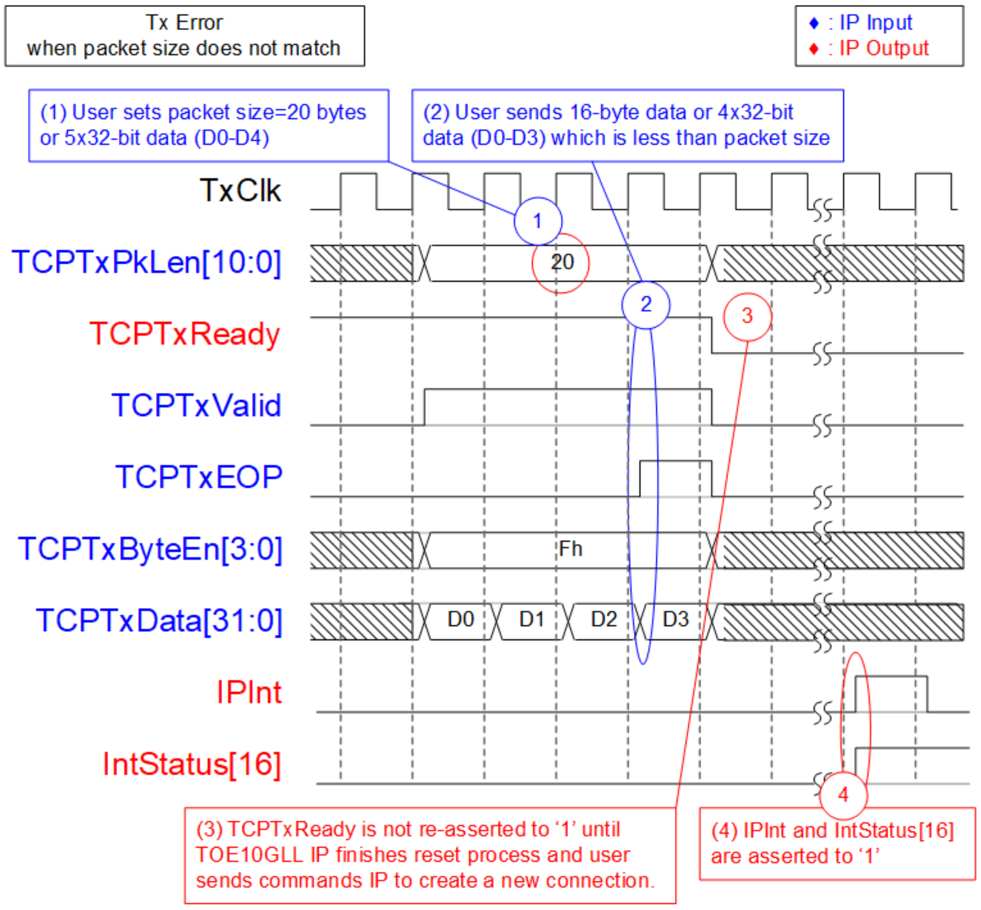

Tx Error
when packet size does not match
♦ : IP Input
♦ : IP Output
(1) User sets packet size=20 bytes or 5x32-bit data (D0-D4)
(2) User sends 16-byte data or 4x32-bit data (D0-D3) which is less than packet size
TxClk
TCPTxPkLen[10:0]
20
TCPTxReady
TCPTxValid
TCPTxEOP
TCPTxByteEn[3:0]
Fh
TCPTxData[31:0]
D0
D1
D2
D3
IPInt
IntStatus[16]
1
2
3
4
(3) TCPTxReady is not re-asserted to ‘1’ until TOE10GLL IP finishes reset process and user sends commands IP to create a new connection.
(4) IPInt and IntStatus[16] are asserted to ‘1’

Figure 22: Tx error when packet size does not match to total amount of transmitted data

Figure 23 shows the timing diagram of Tx Error from setting packet size higher than maximum segment size (TCPMSS) which is sent by user logic. The corresponding interrupt is IntStatus[15].

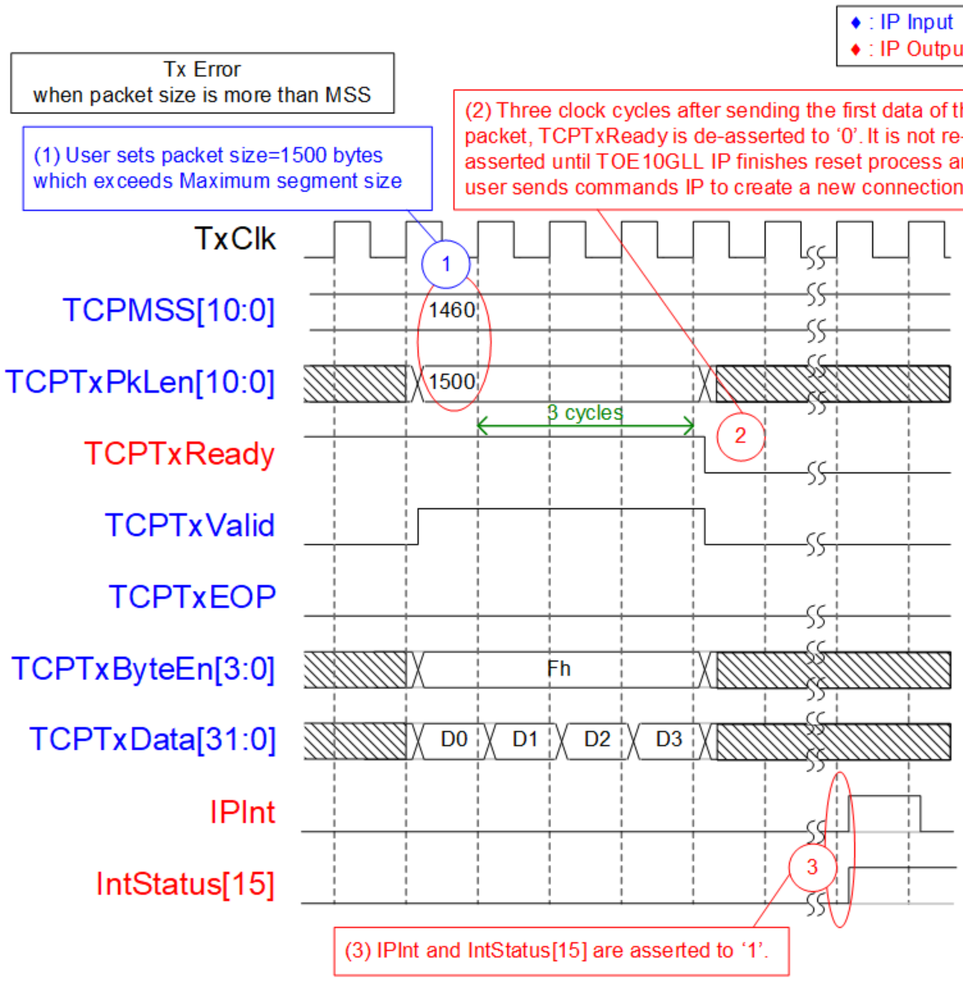

♦ : IP Input
♦ : IP Output
Tx Error when packet size is more than MSS
(1) User sets packet size=1500 bytes which exceeds Maximum segment size
(2) Three clock cycles after sending the first data of the packet, TCPTxReady is de-asserted to '0'. It is not re-asserted until TOE10GLL IP finishes reset process and user sends commands IP to create a new connection.
TxClk
TCPMSS[10:0]
TCPTxPkLen[10:0]
TCPTxReady
TCPTxValid
TCPTxEOP
TCPTxByteEn[3:0]
TCPTxData[31:0]
IPInt
IntStatus[15]
1460
1500
3 cycles
Fh
D0
D1
D2
D3
1
2
3
(3) IPInt and IntStatus[15] are asserted to '1'.

Figure 23: Tx error when packet size is more than TCPMSS size

Figure 24 shows the timing diagram of Tx Error from incorrect value of 16-bit data checksum which is sent by user logic. The corresponding interrput is IntStatus[14].

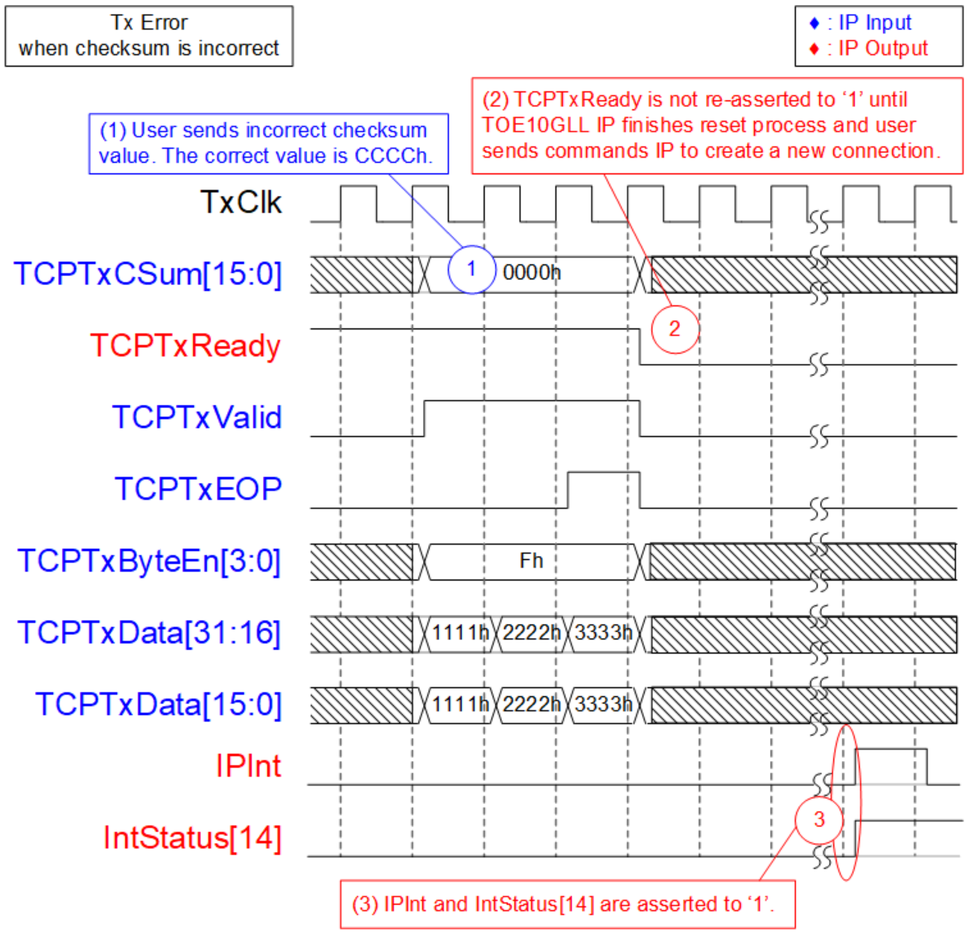

Tx Error
when checksum is incorrect
♦ : IP Input
♦ : IP Output
(1) User sends incorrect checksum value. The correct value is CCCCh.
(2) TCPTxReady is not re-asserted to '1' until TOE10GLL IP finishes reset process and user sends commands IP to create a new connection.
TxClk
TCPTxCSum[15:0]
0000h
TCPTxReady
TCPTxValid
TCPTxEOP
TCPTxByteEn[3:0]
Fh
TCPTxData[31:16]
1111h
2222h
3333h
TCPTxData[15:0]
1111h
2222h
3333h
IPInt
IntStatus[14]
1
2
3
(3) IPInt and IntStatus[14] are asserted to '1'.

Figure 24: Tx error when incorrect 16-bit data checksum value is transmitted

Figure 25 shows the timing diagram of Tx Error from pausing the data transmission by de-asserting TCPTxValid to ‘0’ after IP receiving the first data of the packet and before the final data of the packet is transmitted. The corresponding interrput is IntStatus[13].

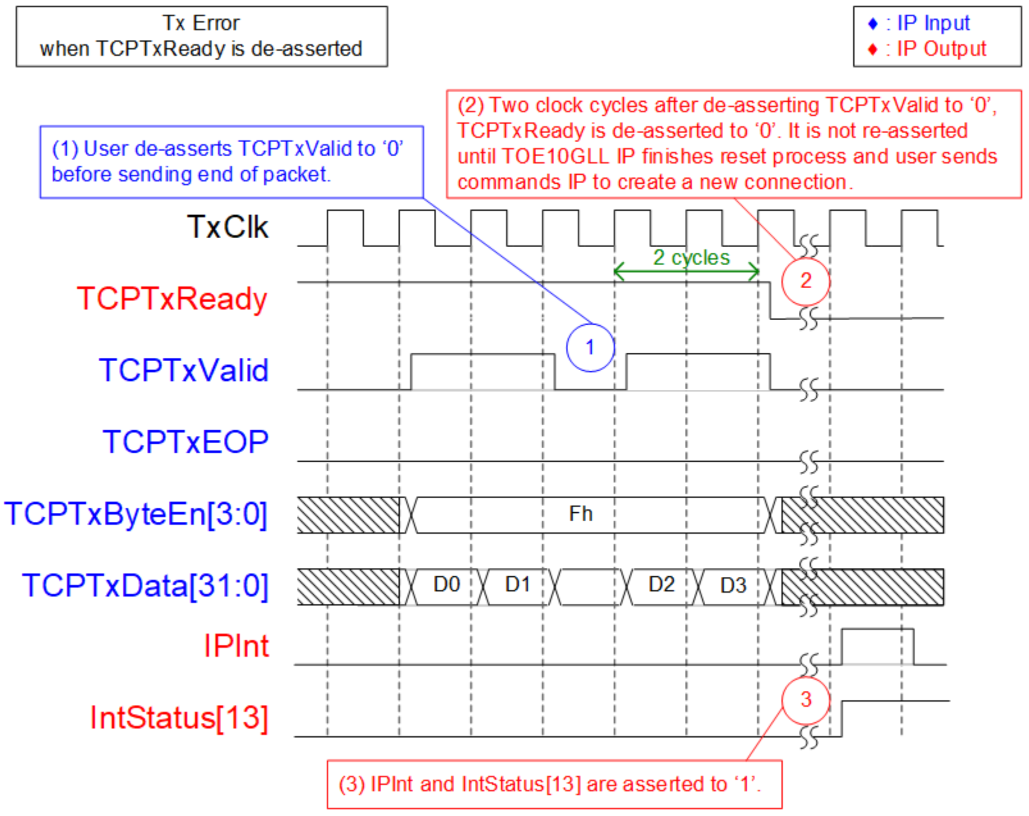

Tx Error
when TCPTxReady is de-asserted
♦ : IP Input
♦ : IP Output
(1) User de-asserts TCPTxValid to ‘0’ before sending end of packet.
(2) Two clock cycles after de-asserting TCPTxValid to ‘0’, TCPTxReady is de-asserted to ‘0’. It is not re-asserted until TOE10GLL IP finishes reset process and user sends commands IP to create a new connection.
TxClk
TCPTxReady
TCPTxValid
TCPTxEOP
TCPTxByteEn[3:0]
TCPTxData[31:0]
IPInt
IntStatus[13]
2 cycles
1
2
3
Fh
D0
D1
D2
D3
(3) IPInt and IntStatus[13] are asserted to ‘1’.

Figure 25: Tx error when the user de-asserts TCPTxValid to ‘0’ before end of packet

Figure 26 shows the timing diagram of Tx Error from sending the unaligned (with 4-byte) data which is not the final data of the packet. The corresponding interrput is IntStatus[12].

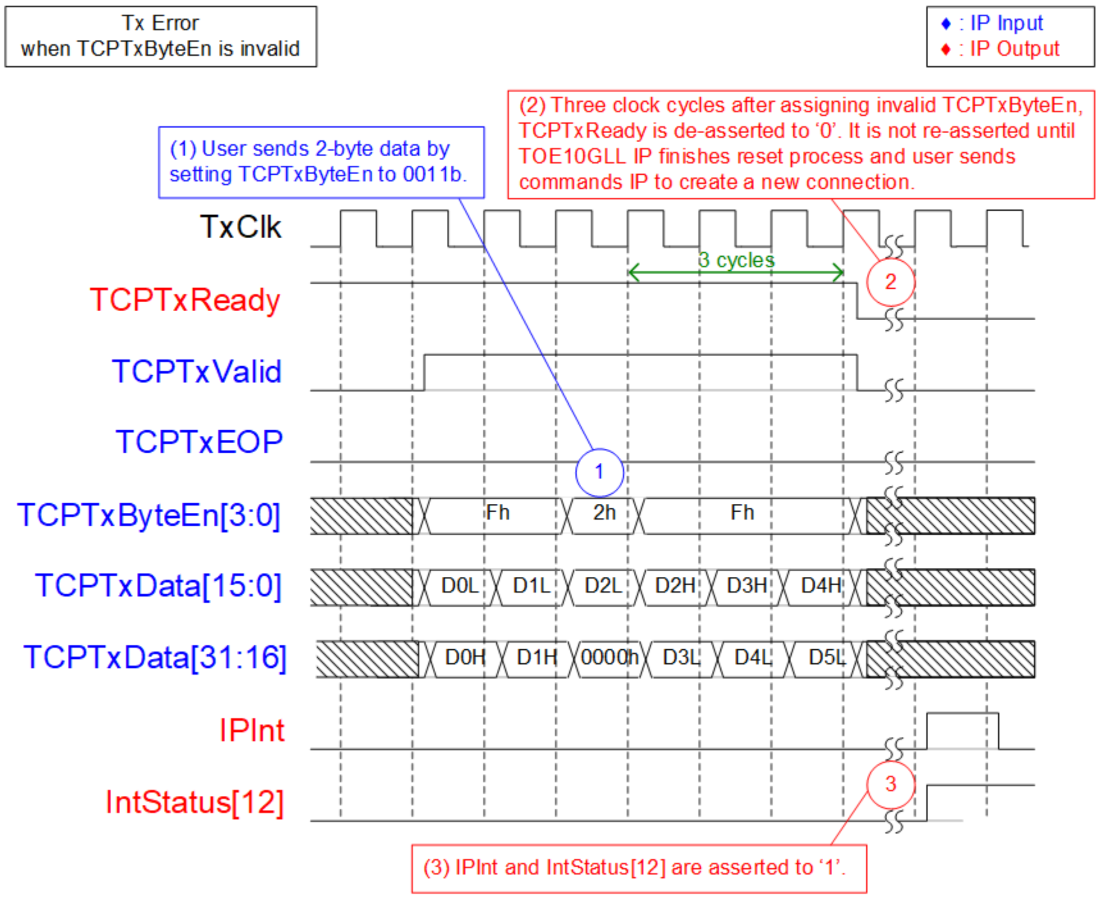

Tx Error
when TCPTxByteEn is invalid
♦ : IP Input
♦ : IP Output
(1) User sends 2-byte data by setting TCPTxByteEn to 0011b.
(2) Three clock cycles after assigning invalid TCPTxByteEn, TCPTxReady is de-asserted to '0'. It is not re-asserted until TOE10GLL IP finishes reset process and user sends commands IP to create a new connection.
TxClk
TCPTxReady
TCPTxValid
TCPTxEOP
TCPTxByteEn[3:0]
TCPTxData[15:0]
TCPTxData[31:16]
IPInt
IntStatus[12]
3 cycles
Fh
2h
Fh
D0L
D1L
D2L
D2H
D3H
D4H
D0H
D1H
0000h
D3L
D4L
D5L
1
2
3
(3) IPInt and IntStatus[12] are asserted to '1'.

Figure 26: Tx error when when the user sends the unaligned data before end of packet

Rx Error

When TCP checksum is error or EMAC returns error in received packet, TCPRxError[0] or [1] (output from TOE10GLL IP) is asserted to '1'. These errors are caused by ungood signal quality in the hardware. After that, TOE10GLL IP sends the packet re-transmission request to the target. The same received packet will be re-sent by the target. Therefore, the user needs to ignore the first received packet which has TCPRxError[0]/[1] asserted and then waits until the new packet is re-transmitted by TOE10GLL IP as shown in Figure 27.

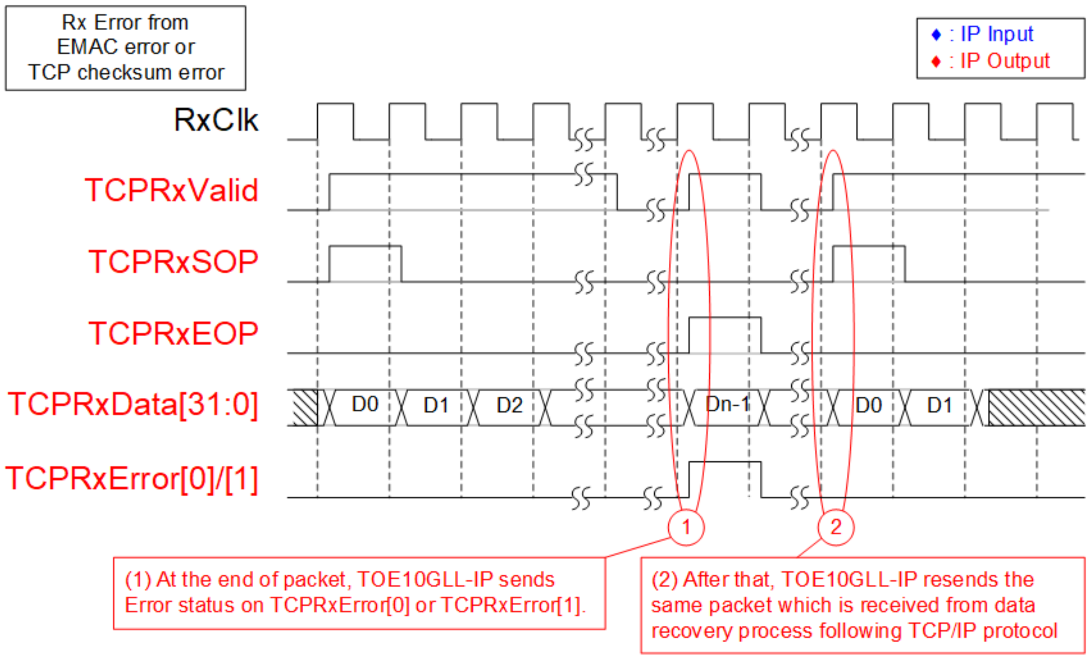

Rx Error from EMAC error or TCP checksum error
♦ : IP Input
♦ : IP Output
RxClk
TCPRxValid
TCPRxSOP
TCPRxEOP
TCPRxData[31:0]
TCPRxError[0]/[1]
D0
D1
D2
Dn-1
D0
D1
1
2
(1) At the end of packet, TOE10GLL-IP sends Error status on TCPRxError[0] or TCPRxError[1].
(2) After that, TOE10GLL-IP resends the same packet which is received from data recovery process following TCP/IP protocol

Figure 27: Packet retransmission in Receive interface when TCPRxError[0]/[1]='1'

(1) If EMAC asserts receive error or TCP checksum in the received packet is error, TCPRxError[1:0] will not be equal to 0 at the end of packet (TCPRxValid='1' and TCPRxEOP='1').
Note:
TCPRxError[0] is asserted to '1' when TCP checksum is error.
TCPRxError[1] is asserted to '1' when EMAC detects CRC error in the packet.
The user logic must ignore the error received packet and wait for the next packet re-transmitted.

(2) After that, TOE10GLL IP starts the error packet recovery process. Finally, TOE10GLL IP resends the same packet that is retransmitted by the target device to the user.

If TOE10GLL IP is still transmitting the received packet to the user but the user sends Active close command, TOE10GLL IP assigns the priority to Active close command. Therefore, the receiving packet will be cancelled by asserting TCPRxError[2] to '1' at the end of packet as shown in Figure 28.

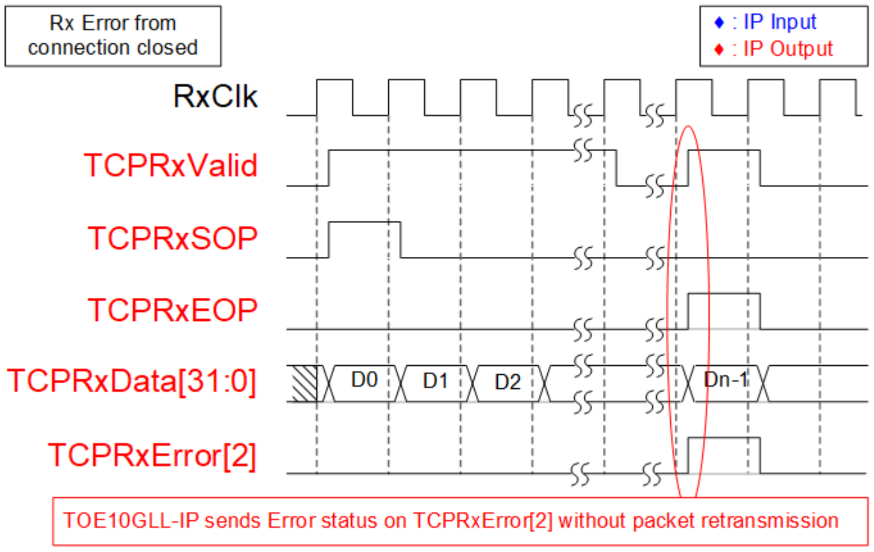

Rx Error from connection closed
♦ : IP Input
♦ : IP Output
RxClk
TCPRxValid
TCPRxSOP
TCPRxEOP
TCPRxData[31:0]
D0
D1
D2
Dn-1
TCPRxError[2]
TOE10GLL-IP sends Error status on TCPRxError[2] without packet retransmission

Figure 28: Error in Receive interface when TCPRxError[2]='1'

Comparing to TCPRxError[0]/[1], there is no retransmitted packet after TCPRxError[2] is asserted. TOE10GLL IP starts the connection termination, not packet re-transmission. Finally, TCPConnOn is de-asserted to '0' and no more packet is transferred until the connection is re-established.

Verification Methods

The TOE10GLL IP Core functionality was verified by simulation and also proved on real board design by using ZCU102 and ZCU106 evaluation board.

Recommended Design Experience

User must be familiar with HDL design methodology to integrate this IP into the design.

Ordering Information

This product is available directly from Design Gateway Co., Ltd. Please contact Design Gateway Co., Ltd. for pricing and additional information about this product using the contact information on the front page of this datasheet.

Revision History

Revision	Date	Description
1.0	3-Nov-20	New release
1.1	30-Apr-21	Update IP information
2.0	19-Apr-22	Update IP to have two transmission mode and support ICMP