TLS10GC-IP Reference Design

Rev1.02 5-Mar-2024

1 Introduction

This document describes the details of TLS 1.3 Client 10Gbps IP Core (TLS10GC-IP) reference design. In this reference design, TLS10GC-IP is used to establish a secure connection using the Transport Layer Security protocol version 1.3 over TCP by handling TLS1.3 handshake, encrypting and decrypting data transferred between the client and the server. Users can set network parameters for TOE10GLL-IP, download and upload payloads to the server by inputting supported command via the serial console. Further details regarding the hardware design and CPU firmware are provided below.

2 Hardware Overview

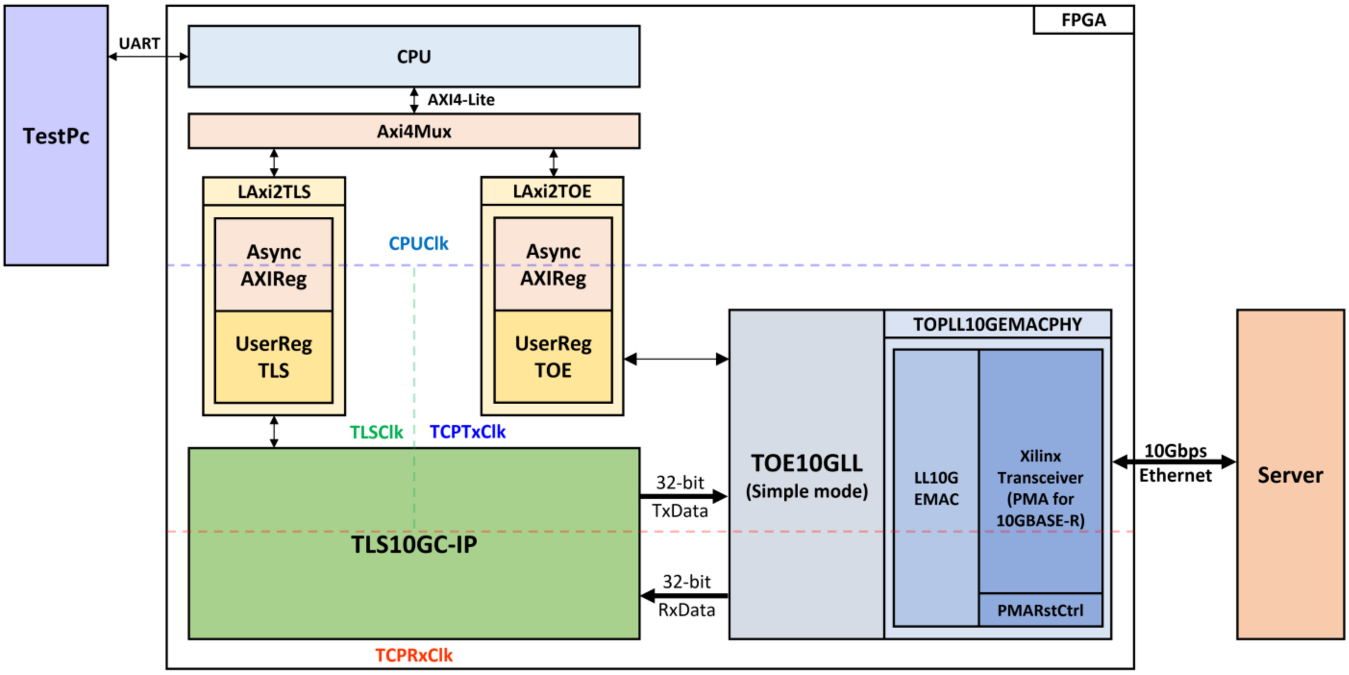

Figure 2-1 TLS10GC-IP reference design block diagram

In this test environment, CPU system is designed to interface with FPGA logic through AXI4 Lite bus and interface with user through serial console in test PC. CPU system communicates with hardware via memory mapping. Axi4Mux is designed to separate the memory mapping for hardware communication into two areas: offset 0x00000-0x000FF for TOE10GLL-IP and 0x00100-0x3FFFF for TLS10GC-IP. To connect the hardware with each memory area of CPU system, AXI4-Lite bus must be implemented by LAxi2TLS for TLS10GC-IP and LAxi2TOE for TOE10GLL-IP, as shown in Figure 2-1.

There are four clock system in this reference design, i.e., CPUClk, TLSClk, TCPTxClk and TCPRxClk. CpuClk is used to interface with CPU through AXI4-Lite bus. TLSClk is the clock domain on which TLS10GC-IP operates and interfaces with users. TCPTxClk is the clock domain which is synchronous to Tx EMAC interface and Tx user data interface. TCPRxClk is the clock domain which is synchronous to Rx EMAC interface and Rx user data interface.

The details of each module are described as follows.

2.1 AsyncAxiReg

This module is designed to convert the signal interface of AXI4-Lite to be register interface. Also, it enables two clock domains to communicate.

To write register, RegWrEn is asserted to ‘1’ with the valid signal of RegAddr (Register address in 32-bit unit), RegWrData (write data of the register), and RegWrByteEn (the byte enable of this access: bit[0] is write enable for RegWrData[7:0], bit[1] is used for RegWrData[15:8], …, and bit[3] is used for RegWrData[31:24]).

To read register, AsyncAxiReg asserts RegRdReq=’1’ with the valid value of RegAddr (the register address in 32-bit unit). After that, the module waits until RegRdValid is asserted to ‘1’ to get the read data through RegRdData signal at the same clock.

2.2 LAxi2TLS

LAxi2TLS module is connected to CPU through AXI4-Lite bus. The hardware registers for TLS10GC-IP are mapped to CPU memory address, as shown in Table 2-1. The control and status registers for CPU access are designed within LAxi2TLS.

LAxi2TLS consists of AsyncAxiReg and UserRegTLS. AsyncAxiReg is designed to convert the AXI4-Lite signals into a simple register interface with 32-bit data bus size (similar to AXI4-Lite data bus size).

UserRegTLS is connected to TLS10GC-IP to control, monitor and prepare data for TLS10GC-IP operation. UserRegTLS consists of register file, a dual port ram for storing certificate information (CertRam), a data pattern generator (UserDataGen) and a data pattern verification (UserDataVer), as shown in Figure 2-2.

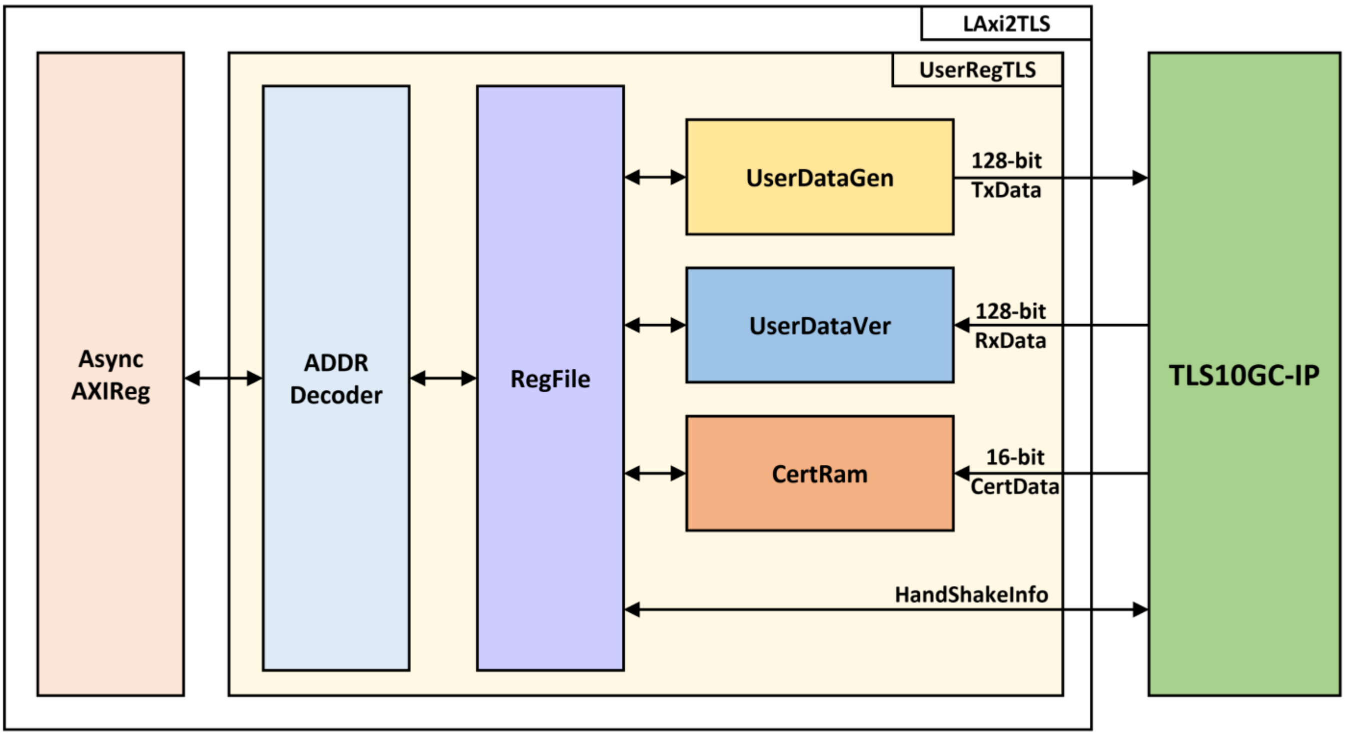

Figure 2-2 UserRegTLS block diagram

Register file

For register file, UserRegTLS is designed to write/read registers, control and check alert of TLS10GC-IP corresponding with write register access or read register request from AsyncAvlReg module. The memory map inside UserRegTLS module is shown in Table 2-1. The timing diagram of register interface is shown in Figure 2-3.

Table 2-1 Register map Definition of TLS10GC-IP

Address offset	Register Name	Rd/ Wr	Description
0x00100	TLS_RSTB_REG	Wr	[0]: Reset signal active low (TLSRstB).
0x00104	TLS_BUSY_REG	Rd	[1]: Busy status for handshake operation (TLSHandshakeBusy). [0]: Busy status for data transfer operation (TLSTrnsBusy)
0x00108	TLS_ALERT_REG	Rd	[15:0]: Alert code from TLS10GC-IP (TLSAlertCode[15:0])
0x0010C	TLS_TIMEOUT_REG	Wr	[15:0]: Timeout value for waiting returned packet (TLSTimeOut[15:0])
0x00110	TLS_TX_RDPTR_REG	Rd	[13:0]: Read pointer to indicate the first byte position of TxData that IP will process (TLSTxUserRdPtr).
0x00114	TLS_TX_WRPTR_REG	Wr	[13:0]: Write pointer to indicate the position after the last TxData written (TLSTxUserWrPtr).
0x00118	TLS_RX_RDPTR_REG	Wr	[13:0]: Read pointer to indicate the first byte of RxData that user already to process (TLSRxUserRdPtr).
0x0011C	TLS_RX_WRPTR_REG	Rd	[13:0]: Write pointer to indicate the position after the last RxData written (TLSTxUserWrPtr).
0x00120	USER_TX_PATT_TYPE_REG	Wr	[1:0]: Data Pattern Mode (rPattGenMode) “00”, “01”, “10” and “11” for decreasing binary, increasing binary, decreasing text, and increasing text, respectively.
0x00124	USER_TX_PATT_LEN_REG	Rd	[31:0]: Remaining data pattern length (wPattRemainLen[31:0])
		Wr	[31:0]: Data pattern length (rPattGenLen[31:0])
0x00128	USER_RX_VERIFY_TYPE_REG	Rd	[1]: Status of Data pattern verification (wVerifyBusy) [0]: Validity status (wVerifyInvalid) ‘0’ for indicating that received data is matched with data pattern, ‘1’ for indicating that received data is matched with data pattern
		Wr	[1]: Verify Pattern Mode (rVerifyMode) “00”, “01”, “10” and “11” for decreasing binary, increasing binary, decreasing text, and increasing text, respectively.
0x0012C	USER_RX_VERIFY_LEN_REG	Rd	[31:0]: Remaining verify length (wDataRemainLen[31:0])
		Wr	[31:0]: Verify pattern length (rVerifyDataLen[31:0])
0x00130	HTTPHEADER_LEN_REG	Wr	[13:0]: Number of bytes for skipping HTTP header before verification. (rHttpHeaderLen)

0x00134	HTTPTRAILER_LEN_REG	Wr	[13:0]: Number of bytes for skipping HTTP trailer before verification. (rHttpTrailerLen)
0x00140-0x0014C	USER_RX_ACTUAL_DATA	Rd	[31:0]: Actual RxData (wActualData[127:0])
0x00150-0x0015C	USER_RX_EXP_DATA	Rd	[31:0]: Expected RxData (wExpData[127:0])
0x00200-0x0022C	CTS_REG	Rd	[31:0]: Client Traffic Secret (CTS)
0x00230-0x0025C	STS_REG	Rd	[31:0]: Server Traffic Secret (STS)
0x00260	TLS_KEYVALID_REG	Rd	[0]: Validity status for key material, key and iv (TLSKeyValid)
0x00270-0x0028C	CH_RANDOM_REG	Rd	[31:0]: Random number in ClientHello message. (Random[255:0])
0x00290	CERT_STARTADDR_REG	Wr	[10:2]: Start address to store certificate information. (rUserRamCertAddr[10:2])
0x00294	CERT_READY_REG	Rd	[0]: Ready status for certificate information. (rTLSCertReady). This signal is set to 1 when the last certificate data is written to CertRam (TLSCertLast='1') and is cleared to zero when CERT_STARTADDR_REG is written.
0x003FC	TLS_VER_REG	Rd	[31:0]: Mapped to IP version of TLS10GC-IP (version)
0x04000-0x05FFF	CERTRAM_BASE_ADDR	Rd/Wr	[31:0]: Certificate data in CertRam (wRdCert32)
0x20000-0x23FFF	RXRAM_BASE_ADDR	Rd	[31:0]: Rx data in UserRxBuffer (wRxRdData32)
0x30000-0x33FFF	TXRAM_BASE_ADDR	Wr	[31:0]: Tx data in UserTxBuffer

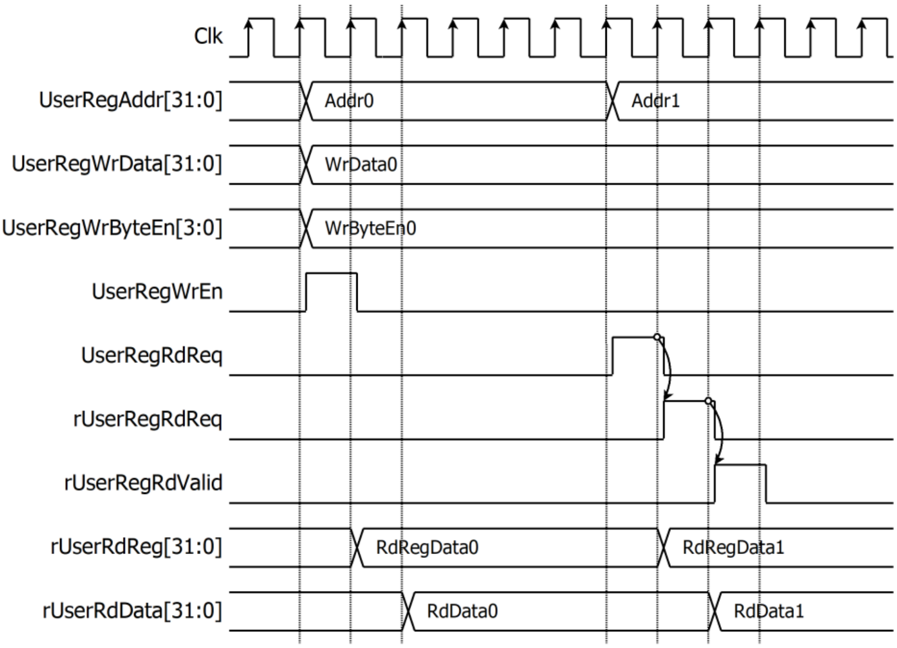

Figure 2-3 Register interface timing diagram

To read register, the multiplexer is designed to select the read data within each address area. UserRegAddr[13:2] is applied in each register area to select the data. Next, the address decoder uses UserRegAddr[17:14] to select the read data from each area to return to the CPU. As shown in Figure 2-3, the read data is valid in next two clock cycles. When UserRegRdReq is active, rUserRegRdReq is asserted to '1'. Then rUserRdValid is active with the valid read value of UserRegAddr.

To write register, UserRegWrEn is asserted to '1' with the valid of UserRegAddr. UserRegAddr[17:14] is used to decode whether CPU accesses UserTxBuffer of TLS10GC-IP or the internal register area. The CPU can access UserTxBuffer when UserDataGen is not busy (rPattGenBusy='0'). When the CPU accesses UserTxBuffer (UserRegAddr[17:14]="0111"), UserRegAddr[13:4] is set to TLSTxUserAddr[13:4]. For example, when UserRegAddr[17:0]= 0x1C004 and UserRegWrEn='1', UserRegWrData will be filled into UserTxBuffer in TLS10GC-IP at Address 0x01. Otherwise, UserRegWrData is loaded into the internal register that matches UserRegAddr[13:2]. For example, rTLSRstBOut is loaded with UserRegWrData when UserRegAddr=0x0000.

Storing Certificate information

TLS10GC-IP is designed to provide certificate to the user for Certificate Validity Verification. In this reference design, dual port ram is used to store the certificate information. As shown in Figure 2-4, TLSCertData[15:0], TLSCertValid and TLSCertByteEn[1:0] can be used to write CertRam. Users can write CERT_STARTADDR_REG to set rUserRamCertAddr[12:1] as the start address to store certificate information. rUserRamCertAddr is a 13-bit counter that is incremented by 1 when TLSCertValid is asserted. rUserRamCertAddr is used as the write address for writing TLSCertData to CertRam. When TLSCertLast is asserted to '1', rTLSCertReady is set to be '1' to indicate that certificate data is ready.

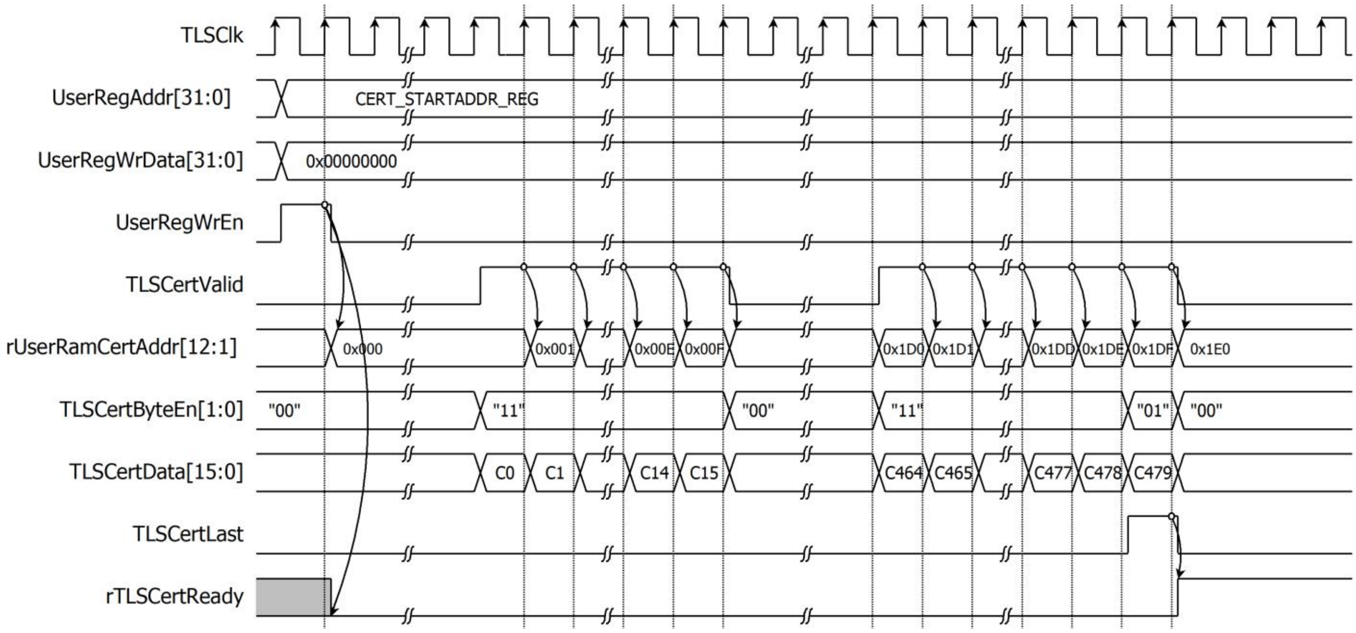

Figure 2-4 Example timing diagram of storing 959-byte certificate information

User Data Generator

UserDataGen is designed to generate a data pattern and write it to UserTxBuffer. There are four types of data patterns: increasing/decreasing binary pattern, increasing/decreasing text pattern. Users can set the type of data by writing to USER_TX_PATT_TYPE_REG, which is mapped to rPattGenMode signal. UserDataGen supports generating unaligned data. After the user sets the data size in byte units to rPattGenLen by writing to USER_TX_PATT_LEN_REG, the data pattern (TLSTxUserDataIn[127:0]) and TLSTxUserByteEn[15:0] are prepared corresponding to the start address.

For example, if the start address is 0x1F and user sets UserDataGen to generate 451-byte increasing text pattern, TLSTxUserDataIn[127:120] is set to 0x00 and TLSTxUserByteEn[15:0] is set to 0x8000 at the first clock cycle to write data only the highest byte at TLSTxUserAddr[13:0]= 0x10. TLSTxUserWrPtr is set to the next start address to indicate to TLS10GC-IP that there is available Tx data to transmit. At second clock cycle, every byte of data pattern is written. At the last clock cycle, only the last 2 bytes of the data pattern are written, TLSTxUserDataIn[15:0] is set to 0xC2C1 and TLSTxUserByteEn[15:0] is set to 0x0003, as shown in Figure 2-5.

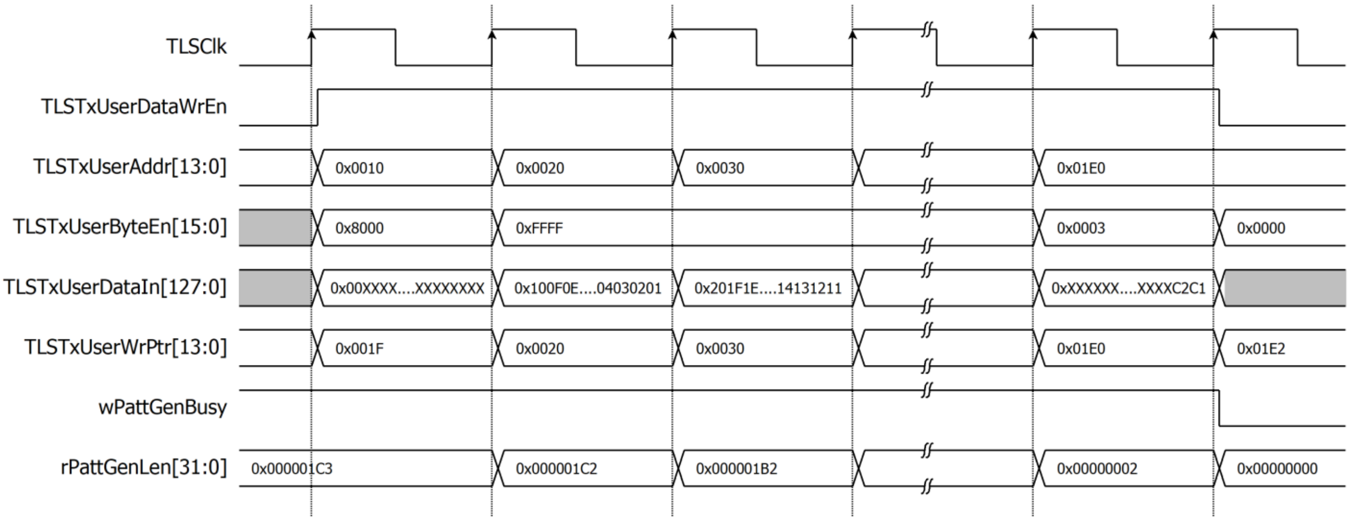

Figure 2-5 Example timing diagram of user data generation process

User Data Verification

UserDataVer reads data via the User Rx interface of TLS10GC-IP when there is available data in UserRxBuffer and verifies the value after starting verification (rVerifyStart is set to ‘1’). There are four types of expected data patterns: increasing/decreasing binary pattern, increasing /decreasing text pattern. Users can set the length of the HTTP header and trailer to skip before starting verification. When there is available data in UserRxBuffer, UserDataVer starts to read received data. UserDataVer supports verifying unaligned data. rExpData[127:0] and rMask[127:0] are prepared corresponding to the start address.

For example, if the start address is 0x1F, and user sets UserDataVer to verify 451-byte increasing text pattern, rExpData[127:120] is set to 0x00 and rMask[127:0] is set to 0xFF000000000000000000000000000000 at the first clock cycle to verify only the highest byte at TLSRxUserAddr[13:0]=0x10. TLSRxUserRdPtr is set to the next start address to indicate to TLS10GC-IP that UserDataVer has already processed RxData. At second clock cycle, every byte of Rx data is verified. At the last clock cycle, only the last 2 bytes of the data pattern are verified, rMask[127:0] is set to 0x0000000000000000000000000000FFFF and rExpData[15:0] is set to 0xC2C1, as shown in Figure 2-6.

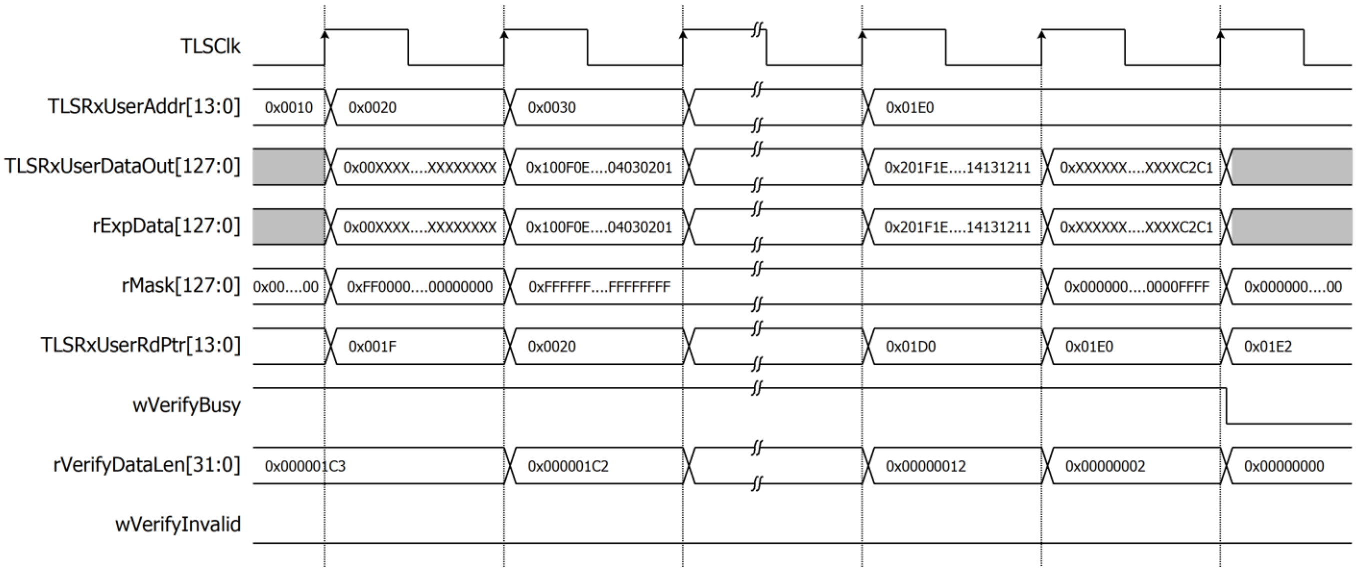

Figure 2-6 Example timing diagram of user data verification process

2.3 TLS10GC-IP

TLS10GC-IP is the IP core provided by Design Gateway to handle TLS1.3 handshake, encrypt and decrypt data as a client. TLS10GC-IP interface is divided into two parts, i.e., User Interface signals and TOE10GLL interface signals. The user interface is connected to LAxi2TLS, allowing users to control, monitor and transfer data with TLS10GC-IP. The TOE10GLL interface is connected to TOE10GLLIP to monitor connection status, send TCPTxData or receive TCPRxData. More details are described in datasheet.

http://www.dgway.com/products/IP/TLS-IP/TLS10GCIP-datasheet-xilinx-en/

2.4 LAxi2TOE

LAxi2TOE module is connected to CPU through AXI4-Lite bus. LAxi2TOE consists of AsyncAxiReg and UserRegTOE. UserRegTOE is designed to write/read registers, control and check status of TOE10GLLIP corresponding with write register access or read register request from AsyncAvlReg module. Memory map inside UserRegTOE module is shown in Table 2-2.

Table 2-2 Register map Definition of TOE10GLLIP

Address offset	Register Name	Rd/ Wr	Description
0x00000	TOE_RST_INTREG	Wr	[0]: Mapped to RstB of TOE10GLL-IP
0x00004	TOE_OPM_INTREG	Wr	[16]: Mapped to ARPICMPEn of TOE10GLL-IP [1:0]: Mapped to DstMacMode of TOE10GLL-IP
0x00008	TOE_SML_INTREG	Wr	[31:0]: Mapped to SrcMacAddr[31:0] of TOE10GLL-IP
0x0000C	TOE_SMH_INTREG	Wr	[15:0]: Mapped to SrcMacAddr[47:32] of TOE10GLL-IP
0x00010	TOE_DMIL_INTREG	Wr	[31:0]: Mapped to DstMacAddr[31:0] of TOE10GLL-IP
0x00014	TOE_DMIH_INTREG	Wr	[15:0]: Mapped to DstMacAddr[47:32] of TOE10GLL-IP
0x00018	TOE_SIP_INTREG	Wr	[31:0]: Mapped to SrcIPAddr of TOE10GLL-IP
0x0001C	TOE_DIP_INTREG	Wr	[31:0]: Mapped to DstIPAddr of TOE10GLL-IP
0x00020	TOE_TMO_INTREG	Wr	[31:0]: Mapped to TimeOutSet of TOE10GLL-IP
0x00024	TOE_TIC_INTREG	Wr	[0]: Set ‘1’ to clear read value of TOE_STS_INTREG[2]
0x00030	TOE_CMD_INTREG	Wr	[1:0]: Mapped to TCPCmd of TOE10GLL-IP.
0x00034	TOE_SPN_INTREG	Wr	[15:0]: Mapped to TCPSrcPort[15:0] of TOE10GLL-IP
0x00038	TOE_DPN_INTREG	Wr	[15:0]: Mapped to TCPDstPort[15:0] of TOE10GLL-IP
0x00040	TOE_VER_INTREG	Rd	[31:0]: Mapped to IP version of TOE10GLL-IP
0x00044	TOE_STS_INTREG	Rd	[20:16]: Mapped to IPState of TOE10GLL-IP [2]: TOE10GLL-IP Interrupt. Asserted to ‘1’ when IPInt is asserted to ‘1’. This flag is cleared by TOE_TIC_INTREG. [1]: Mapped to TCPConnOn of TOE10GLL-IP [0]: Mapped to InitFinish of TOE10GLL-IP
0x00048	TOE_INT_INTREG	Rd	[31:0]: Mapped to IntStatus of TOE10GLL-IP
0x0004C	TOE_DMOL_INTREG	Rd	[31:0]: Mapped to DstMacAddrOut[31:0]
0x00050	TOE_DMOH_INTREG	Rd	[15:0]: Mapped to DstMacAddrOut[47:32]
0x00060	EMAC_VER_INTREG	Rd	[31:0]: Mapped to IP version of DG LL10GEMAC-IP
0x00064	EMAC_STS_INTREG	Rd	[0]: Mapped to Linkup of LL10GEMAC-IP

2.5 TOE10GLL

TOE10GLL-IP is the IP core provided by Design Gateway to implement the TCP/IP stack and offload engine for the low latency solution. User interface has two signal groups, i.e., control signals and data signals. The IP can be configured to run in two modes, i.e., Cut-through mode for low-latency application and Simple mode for simple user interface. This reference design shows the usage in Simple mode. More details are described in datasheet.

https://dgway.com/products/IP/Lowlatency-IP/dg_toe10gllip_data_sheet_xilinx_en/

2.6 LL10GEMAC

The IP core by Design Gateway implements low-latency EMAC and PCS logic for 10Gb Ethernet (BASE-R) standard. The user interface is 32-bit AXI4-stream bus. Please see more details from LL10GEMAC datasheet on our website.

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

2.7 Xilinx Transceiver (PMA for 10GBASE-R)

PMA IP core for 10Gb Ethernet (BASE-R) can be generated by using Vivado IP catalog. In FPGA Transceivers Wizard, the user uses the following settings.

• Transceiver configuration preset	: GT-10GBASE-R
• Encoding/Decoding	: Raw
• Transmitter Buffer	: Bypass
• Receiver Buffer	: Bypass
• User/Internal data width	: 32

The example of Transceiver wizard in Ultrascale model is described in the following link.

https://www.xilinx.com/products/intellectual-property/ultrascale_transceivers_wizard.html

2.8 PMARstCtrl

When the buffer inside Xilinx Transceiver is bypassed, the user logic must control reset signal of Tx and Rx buffer. The module is designed by state machine to run following step.

(1) Assert Tx reset of the transceiver to ‘1’ for one cycle.

(2) Wait until Tx reset done, output from the transceiver, is asserted to ‘1’.

(3) Finish Tx reset sequence and de-assert Tx reset to allow the user logic beginning Tx operation.

(4) Assert Rx reset to the transceiver.

(5) Wait until Rx reset done is asserted to ‘1’.

(6) Finish Rx reset sequence and de-assert Rx reset to allow the user logic beginning Rx operation.

3 CPU Firmware

After system boot-up, CPU initializes its peripherals such as UART and Timer. Then the supported command usage is displayed. The main function runs in an infinite loop to receive line command input from the user. Users can set the IP address, port number and MAC address of the FPGA board, show key materials, show certificate information, download and upload data using the supported commands. More details of the sequence in each command are described as follows.

3.1 Set FPGA's IP Address

```
command> setip ddd.ddd.ddd.ddd
```

All network parameters are set to TOE10GLL-IP. Users can set IP address for TOE10GLL-IP by inputing setip followed by desired IP address in dotted-decimal format. The setip function is called to change the IP address value in src_ip_set array. This array will be written to the register mapped to SrcIPAddr to set the FPGA's IP address. Subsequently, TOE10GLL-IP is initialized with the current network parameter setting. The default FPGA's IP address is 192.168.7.42. The setip function is described in Table 3-1.

Table 3-1 setip function

int setip(unsigned char * ipStr, unsigned char *ip_set)	
Parameter	ipStr: ip address as string input from user ip_set: array stored IP address
Return value	0: Valid input, -1: Invalid input
Description	This function receives IP Address as string input and set value of ip_set array.

3.2 Set FPGA’s Port Number

command> setport ddddd

Users can set port number to TOE10GLL-IP by inputting setport followed by the static port number of the FPGA in decimal format or “dynamic”, “d” or “-d” to set the port number to be dynamic. The setport function is called to change the port number value in src_port_set array. This array will be written to the register mapped to TCPSrcPort to set the FPGA’s port number. Dynamic ports are in the range 49152 to 65535. If port number is set to be dynamic, the port number will be automatically increased by 1 before establishing a new connection. If the port number is set as a static port number and the user does not set the new port number value, the FPGA’s port number will not be changed. The setport function is described in Table 3-2.

Table 3-2 setport function

int setport(unsigned char *portstr, unsigned char *port_set)	
Parameter	portstr: port number as string input from user port_set: array stored port number
Return value	0: Valid input, -1: Invalid input
Description	This function receives port number as string input and set value of port_set array.

3.3 Set FPGA’s MAC address

command> setmac hh-hh-hh-hh-hh-hh

Users can set MAC address to TOE10GLLIP by inputing setmac followed by the FPGA’s MAC address in hexadecimal format. The setmac function is called to change the MAC address value in mac_set array. This array will be written to the register mapped to SrcMacAddr to set the FPGA’s MAC address. The default FPGA’s MAC address is 00-01-02-03-04-05. The setmac function is described in Table 3-3.

Table 3-3 setmac function

int setmac(unsigned char *macstr)	
Parameter	macstr: MAC address as string input from user
Return value	0: Valid input, -1: Invalid input
Description	This function receives MAC Address as string input and set value of mac_set array.

3.4 Show key materials

command> showkey <1: enable, 0: disable>

To change showkey mode, users can input showkey <1: enable, 0: disable> to modify a global variable, showTrafficSecret. If showTrafficSecret is set to ‘1’, traffic tickets will be displayed on the serial console after the handshake process is completed. Users can use the TLS traffic ticket as a (Pre)-Master-Secret log file for Wireshark* to decrypt transferred data over the current connection.

*Wireshark, a network packet analyzer tool used for network troubleshooting, analysis, and security purposes.

3.5 Show certificate information

command> showcert <1: enable, 0: disable>

To change showcert mode, users can input showcert <1: enable, 0: disable> to modify a global variable, showCertificate. If showCertificate is set to ‘1’, certificate information will be displayed on the serial console after the certificate is ready during the handshake phase. Users can use certificate information for further certificate validity verification.

3.6 Download data pattern

command> myGET https://ip:port/download/pattern/length

Where	ip	represent server's ip address in dot-decimal notation
	port	represent server's port number
	pattern	represent data pattern
	length	represent data length in byte

This command simulates GET method of HTTP to download data from the server. myGET function is called to extract the server's IP address and the server's port number, initialize network parameters, set registers to start data verification and monitor status. The sequence of the myGET function is as follows.

1) Split the URL input and set network parameters corresponding to the URL.

2) Construct a HTTP GET command from the URL

3) Open connection and wait for finishing handshake process.

4) Write the GET HTTP command to UserTxRam and set TLSTxUserWrPtr by writing TLS_TX_WRPTR_REG.

5) If the user requests data pattern, set HTTP header length to HTTPHEADER_LEN_REG, and trailer length to HTTPTRAILER_LEN_REG, set the data pattern type to USER_RX_VERIFY_TYPE_REG and data length to USER_RX_VERIFY_LEN_REG to verify data pattern. If user requests HTML page, verification mode is disable.

6) Monitor verification status (TLS_VERIFY_INVALID) and remaining verify pattern length (USER_RX_VERIFY_LEN_REG).

7) Until the reception of data is complete, the transfer speed is computed and displayed on the serial console. If the received data length is less than 16 kB, the received data also be shown on the serial console.

Table 3-4 myGET function

int myGET(unsigned char *urlStr)	
Parameter	urlStr: URL as string input from user
Return value	0: Valid input, -1: Invalid input
Description	This function receives URL as string input and validate URL. If URL is valid, HTTP GET command will be sent and display receiving result.

3.7 Download data in log folder

command> myGET https://ip:port/download/log/filename

Where	ip	represent server’s ip address in dot-decimal notation
	port	represent server’s port number
	filename	represent filename in log folder

This reference design simulates GET method of HTTP to download existing files in the log folder. myGET function is called and operates following the same sequence as described in 3.6 Download data pattern, but verification mode is disabled.

3.8 Upload data pattern

command> myPOST https://ip:port/upload/pattern/length

Where	ip	represent server's ip address in dot-decimal notation
	port	represent server's port number
	pattern	represent data pattern
	length	represent data length in byte

This command simulates POST method of HTTP to upload data pattern to the server. myPOST function is called to extract the server's IP address and the server's port number, initialize network parameters, set registers to start data generator and monitor status. The sequence of the myPOST function is as follows.

1) Split the URL input and set network parameters corresponding to the URL.
2) Construct a HTTP POST command from the URL
3) Open connection and wait for finishing handshake process.
4) Write the POST HTTP command to UserTxRam and set TLSTxUserWrPtr by writing TLS_TX_WRPTR_REG.
5) Set the data pattern type to USER_TX_PATT_TYPE_REG and the data length to USER_TX_PATT_LEN_REG.
6) Monitor remaining data pattern length (USER_TX_PATT_LEN_REG).
7) Until the remaining data pattern length is zero, indicating that the user logic has completely prepared the desired amount of data for TLS10GC-IP, or the connection is closed. The transfer speed is computed and displayed on the serial console.

Table 3-7 myPOST function

int myPOST(unsigned char *urlStr)	
Parameter	urlStr: URL as string input from user
Return value	0: Valid input, -1: Invalid input
Description	This function receives URL string input by user, validate and construct HTTP POST command to send before transferring data to server. Monitor UserDataGen status and number of transferred data to show transfer speed.

3.9 Full duplex test

command> myFullduplex https://ip:port/fullduplex/pattern/length

Where	ip	represent server's ip address in dot-decimal notation
	port	represent server's port number
	pattern	represent data pattern
	length	represent data length in byte

This command is used to transfer data between the client and the server in full duplex mode. It simulates POST method of HTTP with the fullduplex URL that requests a data pattern from the server and also uploads the data pattern to the server. myFullduplex function is called to extract the server's IP address and the server's port number, initialize network parameters, set registers to start data generator/verification and monitor status. The sequence of the myFullduplex function is as follows.

1) Split the URL input and set network parameters corresponding to the URL.

2) Construct a HTTP POST command from the URL

3) Open connection and wait for finishing handshake process.

4) Write the POST HTTP command to UserTxRam and set TLSTxUserWrPtr by writing TLS_TX_WRPTR_REG.

5) Set the data pattern type to USER_TX_PATT_TYPE_REG and the data length to USER_TX_PATT_LEN_REG for Tx operation and set the HTTP header length to HTTPHEADER_LEN_REG, and the trailer length to HTTPTRAILER_LEN_REG, set the data pattern type to USER_RX_VERIFY_TYPE_REG and the data length to USER_RX_VERIFY_LEN_REG to verify data pattern for Rx operation.

6) Monitor Tx operation status by considering the remaining data pattern length (USER_TX_PATT_LEN_REG) and TOE10GLL busy status (TOE_STS_INTREG) and Rx operation status by considering verification status (TLS_VERIFY_INVALID) and remaining verify pattern length (USER_RX_VERIFY_LEN_REG).

7) Until the transmission and reception of data are complete, or the connection is closed. The transfer speed is computed and displayed on the serial console.

Table 3-7 myFullduplex function

int myFullduplex(unsigned char *urlStr)	
Parameter	urlStr: URL as string input from user
Return value	0: Valid input, -1: Invalid input
Description	This function receives URL string input by user, validate and construct HTTP POST command to send data pattern and request the same data from server. Monitor Tx/Rx operation status and number of data and show transfer speed.

4 Revision History

Revision	Date	Description
1.02	5-Mar-2024	Correct some descriptions.
1.01	3-Jan-2024	Add full duplex test
1.00	8-Sep-2023	Initial version release