QUIC10GC-IP Reference Design

Table of Contents

QUIC10GC-IP Reference Design

Rev1.00 3-Jul-2024

1 Introduction

This document describes the details of the QUIC Client 10Gbps IP core (QUIC10GC-IP) reference design. In this reference design, the QUIC10GC-IP is used as a medium to transfer data within a secure connection following the QUIC transport protocol version 1 standard (RFC9000). This process involves handling the TLS 1.3 handshake and dealing with data encryption/decryption and flow control. Users can set network parameters, 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

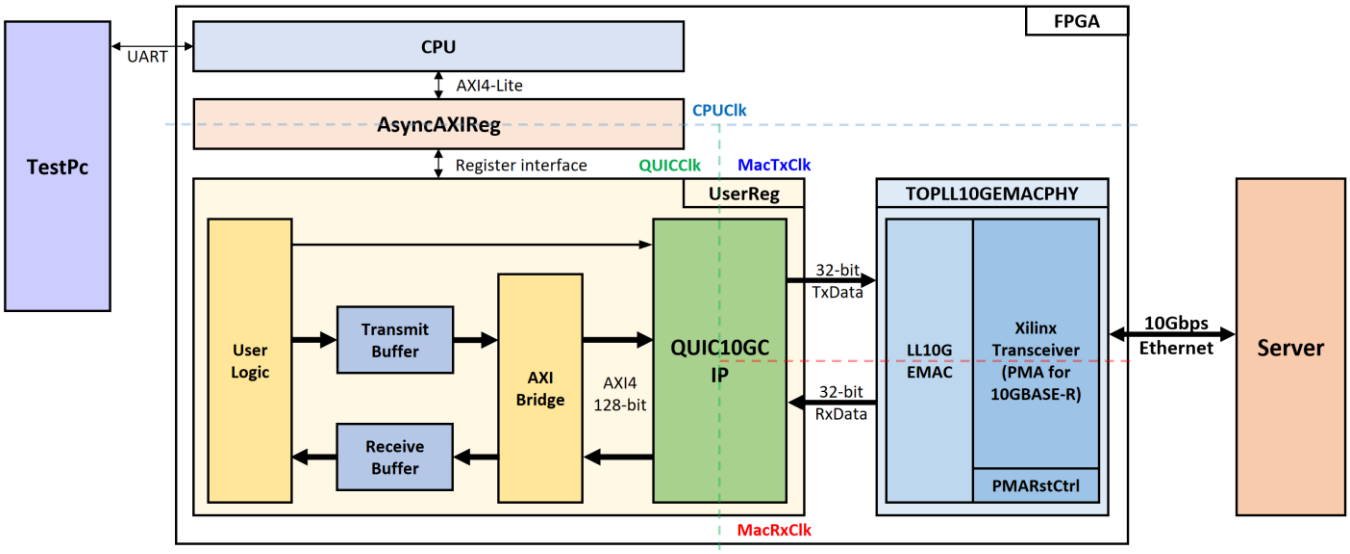

Figure 2-1 QUIC10GC-IP reference design block diagram

In this test environment, two devices are used to transfer data over a 10G Ethernet connection. The FPGA acts as the QUIC Client, while the target device, which can be either a PC or another FPGA, acts as the QUIC Server. As shown in Figure 2-1, the QUIC10GC-IP is integrated within UserReg. UserReg connects to the CPU through AsyncAXIReg using a register interface, and the CPU connects to AsyncAXIReg via an AXI4-Lite interface.

The user interface of the QUIC10GC-IP connects to AXIBridge via an AXI4 interface for reading data from the Transmit Buffer and writing data to the Receive Buffer. The user logic is responsible for generating the sending data, verifying the receiving data, and other user control operations for the QUIC10GC-IP.

There are four system clocks in this reference design, i.e., CPUClk, QUICClk, MacTxClk and MacRxClk. CpuClk is used to interface with CPU through AXI4-Lite bus. QUICClk is the clock domain on which the QUIC10GC-IP operates and interfaces with users. MacTxClk is the clock domain which is synchronous to Tx EMAC interface. MacRxClk is the clock domain which is synchronous to Rx EMAC 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.

The address of Register interface is shared for both write and read transactions, so user cannot write and read the register at the same time. The timing diagram of the Register interface is shown Figure 2-2.

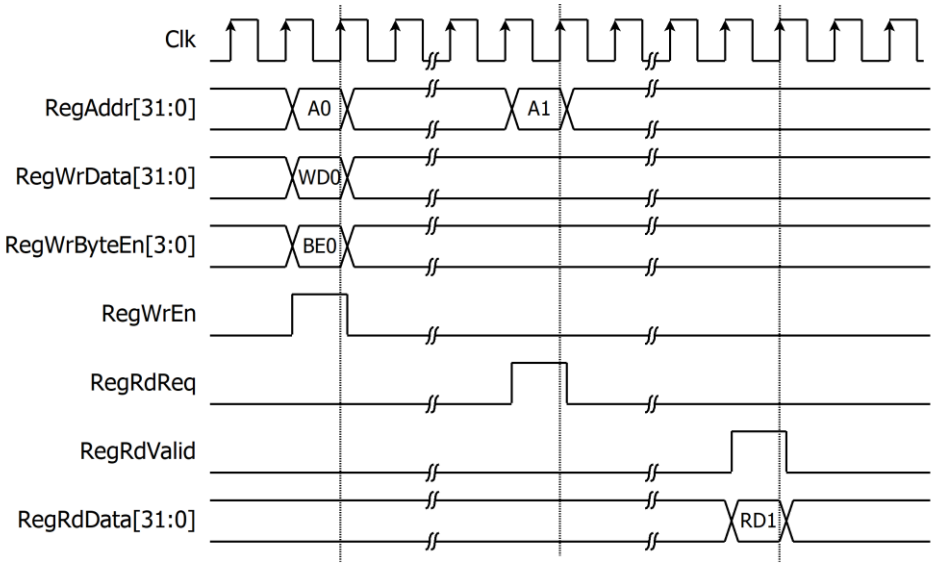

Figure 2-2 Register interface timing diagram

2.2 UserReg

For register file, UserReg is designed to write/read registers, control and check alert of the QUIC10GC-IP corresponding with write register access or read register request from AsyncAvlReg module. The memory map inside UserReg module is shown in Table 2-1.

Table 2-1 Register map Definition of QUIC10GC-IP

Address	Register Name	Description
Ethernet MAC register		
0x0060	EMAC_VER_INTREG	Rd[31:0]: LL10GEMAC-IP Version (MacIPVersion).
0x0064	EMAC_STS_INTREG	Rd[0]: Linkup status of LL10GEMAC-IP (MacLinkup).
QUIC10GC Control register		
0x0100	QUIC_RSTB_REG	Wr/Rd[0]: Reset signal active low (rQUICRstBOut).
0x0104	QUIC_CONN_REG	Wr/Rd[0]: User's Connection status. (rQUICConnOn).
0x0108	QUIC_BUSY_REG	Rd[3]: Data receive operation busy status (QUICRxTrnsBusy). Rd[2]: Data transmit operation busy status (QUICTxTrnsBusy) Rd[1]: Handshake operation busy status (QUICHandshakeBusy) Rd[0]: Connection operation busy status (QUICConnOnBusy).
0x010C	QUIC_ALERT_REG	Rd[15:0]: Normal and alert conditions of the QUIC10GC-IP (QUICAlertCode[15:0]).
QUIC User Data		
0x0120	QUIC_TX_BASE_ADDR_LOW_REG	Wr[31:0]: Lower 32 bits of the base address for the transmit buffer (AppTxBaseAddr[31:0])
0x0124	QUIC_TX_BASE_ADDR_HIGH_REG	Wr[31:0]: Upper 32 bits of the base address for the transmit buffer (AppTxBaseAddr[63:32])
0x0128	QUIC_RX_BASE_ADDR_LOW_REG	Wr[31:0]: Lower 32 bits of the base address for the receive buffer (AppRxBaseAddr[31:0])
0x012C	QUIC_RX_BASE_ADDR_HIGH_REG	Wr[31:0]: Upper 32 bits of the base address for the receive buffer (AppRxBaseAddr[63:32])
0x0140-0x014C	QUIC_TX_USER_PTR_REG	Rd[17:0]: Read pointer of streamID 'X' to indicate the first byte position of TxData that IP will process (AppTxRdAddrX[17:0]). Wr[17:0]: Write pointer of streamID 'X' to indicate the position after the last TxData written (rAppTxWrAddrX[17:0]).
0x0150-0x015C	QUIC_TX_USER_FINAL_REG	Wr[0]: Set the end stream flag of the current Tx write pointer for StreamID 'X' (rAppTxWrFinX)
0x0180-0x018C	QUIC_RX_USER_PTR_REG	Rd[17:0]: Write pointer of streamID 'X' to indicate the position after the last RxData written (AppRxWrAddrX[17:0]). Wr[17:0]: Read pointer of streamID 'X' to indicate the first byte of RxData that user will process (rAppRxRdAddrX[17:0]).
0x0190-0x019C	QUIC_RX_USER_FINAL_REG	Rd[0]: Indicating the end of stream has been received for streamID 'X' (AppRxWrFinX)
0x01C0	QUIC_RX_USER_INFO_READ_REG	Rd[0]: Empty status of QUICRxInfo FIFO, storing QUIC Rx user information (UsrRxInfoFfEmpty). Wr[0]: Set read enable to QUICRxInfo FIFO (UsrRxInfoFfEmpty).
0x01C4	QUIC_RX_USER_COMMON_REG	Rd[7:0]: QUIC Rx user information type (QUICRxInfoType[7:0]). Rd[15:8]: QUIC Rx user information streamID (QUICRxInfoID[7:0]).

0x01D0- 0x01D4	QUIC_RX_USER_INFO0_REG	Rd[31:0]: QUIC Rx user information field 0 (QUICRxInfoD0[63:0])
0x01D8- 0x01DC	QUIC_RX_USER_INFO1_REG	Rd[31:0]: QUIC Rx user information field 1 (QUICRxInfoD1[63:0])
0x0200	USER_TX_PATT_ADDR_REG	Rd[19:0]: Current write address for writing Tx data pattern to transmit buffer (rTxUserWrPtr[19:0]). Wr[19:0]: Start Address for writing Tx data pattern.
0x0204	USER_TX_PATT_TYPE_REG	Wr[0]: Data pattern mode (rPattGenMode) “0” for incremental and “1” for decremental 8-bit counter.
0x0208	USER_TX_PATT_LEN_REG	Rd[17:0]: Remaining data pattern length (rPattGenLen[17:0]). Wr[17:0]: Length of data pattern (rPattGenLen[17:0]).
0x0210	USER_RX_VERIFY_ADDR_REG	Rd[19:0]: Read address of the first Rx data that failed verification (rVerifyRxUserRdPtr[19:0]). Wr[19:0]: Start Address for reading Rx data pattern (rRxUserRdPtr[19:0]).
0x0214	USER_RX_VERIFY_TYPE_REG	Rd[1]: Validity status (wVerifyInvalid) ‘0’ for indicating that received data is matched with data pattern, ‘1’ for indicating that received data is NOT matched with data pattern. Rd[0]: Data verification busy status (rVerifyBusy(0)). Wr[0]: Data verification mode (rVerifyMode) “0” for incremental and “1” for decremental 8-bit counter When the data verification mode is set, verification status is reset.
0x0218	USER_RX_VERIFY_LEN_REG	Rd[17:0]: Remaining data verify length (rVerifyLen[17:0]). Wr[17:0]: Length of verification pattern (rVerifyLen[17:0])
0x0220- 0x022C	USER_RX_ACTUAL_DATA	Rd[31:0]: Actual RxData (rVerifyActualData[127:0])
0x0240- 0x024C	USER_RX_EXP_DATA	Rd[31:0]: Expected RxData (rVerifyExpectData[127:0])
0x0280- 0x028C	USER_TX_PATT_DATA_REG	Rd[31:0]: Current data pattern (rPattGenData) Wr[31:0]: Initial data for the data pattern.
0x02A0- 0x02AC	USER_RX_PATT_DATA_REG	Rd[31:0]: Current verification pattern (rVerifyExpData). Wr[31:0]: Initial data for the data verification.
0x02E0- 0x02EC	QUIC_ALPN_DATA_REG0-3	Wr[31:0]: ALPN string value (QUICALPNStr[127:0]).
0x02F0	QUIC_ALPN_LEN_REG	Wr[4:0]: ALPN string length (QUICALPNLen[4:0]).
0x0300- 0x031C	QUIC_CTS_REG	Rd[31:0]: Client Traffic Secret (CTS[255:0])
0x0340- 0x035C	QUIC_STS_REG	Rd[31:0]: Server Traffic Secret (STS[255:0])
0x0380- 0x039C	QUIC_RANDOM_REG	Rd[31:0]: Random number in ClientHello message. (Random[255:0])
0x03C0	QUIC_KEY_VALID_REG	Rd[0]: Validity status for key material, key and iv (QUICKeyValid)
0x0400	QUIC_UDP_SRCMAC_LOW_REG	Wr[31:0]: Lower 32 bits of source MAC address (rSrcMacAddr[31:0]).
0x0404	QUIC_UDP_SRCMAC_HIGH_REG	Wr[15:0]: Upper 16 bits of source MAC address (rSrcMacAddr[47:32]).
0x0408	QUIC_UDP_SRCIP_REG	Wr[31:0]: Source IP address (rSrcIPAddr[31:0])
0x040C	QUIC_UDP_DSTIP_REG	Wr[31:0]: Destination IP address (rDstIPAddr[31:0])
0x0410	QUIC_UDP_SRCPORT_REG	Wr[15:0]: Source port number (rSrcPort[15:0]).
0x0414	QUIC_UDP_DSTPORT_REG	Wr[15:0]: Destination port number (rDstPort[15:0]).
0x0418	QUIC_UDP_GTWIP_REG	Wr[31:0]: Gateway IP address (rGatewayIPAddr[31:0]).
0x041C	QUIC_UDP_IPNETSET_REG	Wr[0]: Set IP network parameters (rNetworkSet).

0x04FC	QUIC_VER_REG	Rd[31:0]: QUIC10GC-IP version (QUICIPVersion[31:0]).
0x4000-0x4FFF	CERTRAM_BASE_ADDR	Rd[31:0]: Certificate data in CertRam (wRamCertRdData[31:0]).
0x5000	CERT_STARTADDR_REG	Wr[11:1]: Start address for CertRam (rUserRamCertAddr[11:1])
0x5004	CERT_READY_REG	Rd[0] : Ready status for certificate information. (rUserCertReady) This signal is set to 1 when the last certificate data is written to CertRam (QUICCertLast='1') and is cleared to zero when CERT_STARTADDR_REG is written.
0x200000-0x2FFFFF	USER_RXRAM_BASE_ADDR	Rd[31:0]: Rx data read from the receive buffer (UserRxRamRdData).
0x300000-0x3FFFFF	USER_TXRAM_BASE_ADDR	Wr[31:0]: Tx data written to the transmit buffer (rUserTxRamWrData).

Storing Certificate information

The QUIC10GC-IP is designed to provide certificates to the user for Certificate Validity Verification. In this reference design, a dual-port RAM is used to store the certificate information. As shown in Figure 2-3, the signals QUICCertValid, QUICCertByteEn[1:0] and QUICCertData[15:0] are used to write certificate information to CertRam. Users can write to the CERT_STARTADDR_REG to set rUserRamCertAddr[11:1] as the start address for storing certificate information. The rUserRamCertAddr is an 11-bit counter that increments by 1 when QUICCertValid is asserted. This address serves as the write address for writing QUICCertData to CertRam. When QUICCertLast is asserted to '1', rUserCertReady is set to '1', indicating that the certificate data is ready.

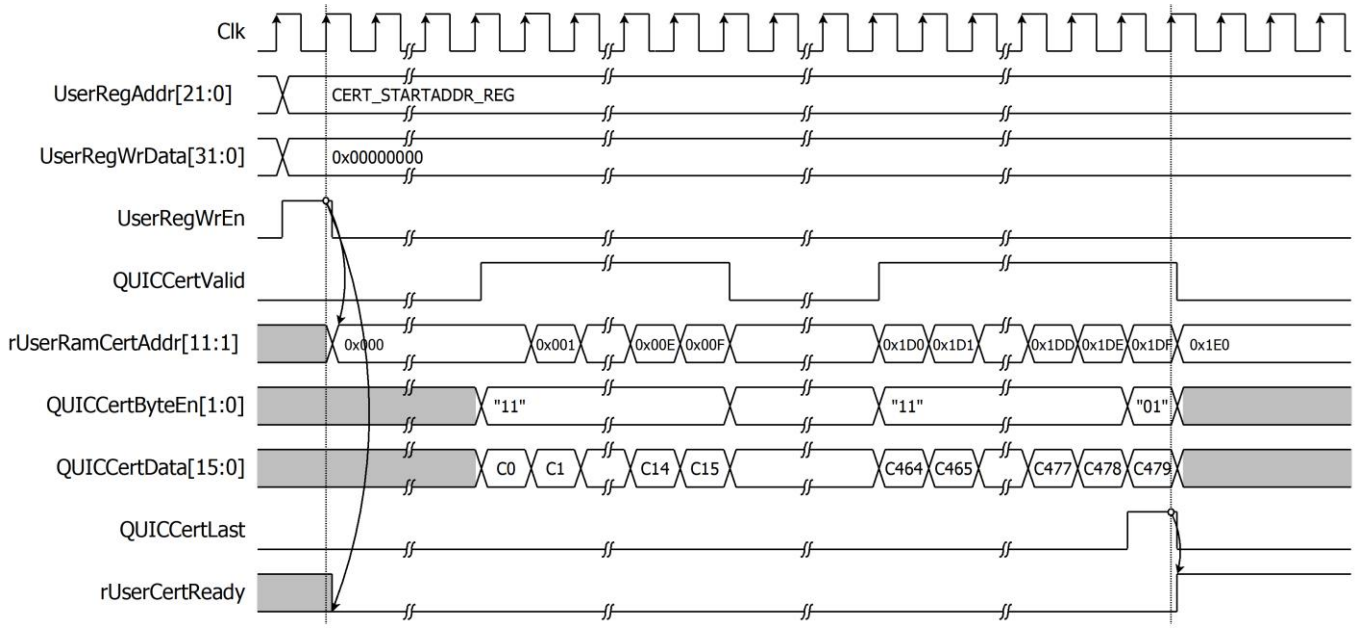

Figure 2-3 Example timing diagram of storing 959-byte certificate information

AXI Bridge

In the reference design, an AXI bridge is used to convert AXI protocol transactions into memory interface operations. The AXI bridge converts AXI write transactions from the QUIC10GC-IP to write receive data into the receive buffer (RxRam) and converts AXI read transactions from the QUIC10GC-IP to read transmit data from the transmit buffer (TxRam).

In the case of sending AXI write requests to the AXI bridge, AxiAwReady will be de-asserted to '0', and AxiwReady will be asserted to '1' in the next clock cycle. The AXI bridge will set Ram0WrAddr[19:0] to AxiAwAddr[19:0], positioning the first address to write data to RxRam. When AxiwReady is '1' and the AXI master is ready to write data, the AXI master will assert AxiwValid to '1'. When the AXI bridge receives AxiwValid as '1' from the AXI master, the AXI bridge will forward information in AxiwValid, AxiwStrb[15:0], and AxiwData[127:0] to Ram0WrEn, Ram0WrByteEn[15:0], and Ram0WrData[127:0], respectively. Ram0WrAddr[19:0] will increment by 16 for each AXI master write data word. To ensure data is written correctly, the AXI master must write data for all bytes in a word except the first or last word. When the AXI master transfers data to the last word, it must assert AxiwLast to '1'. When AxibReady is '1' and AxibValid is '1', it signifies the completion of the write data operation, and the AXI bridge will set AxiAwReady to '1' in the next clock cycle to accept new write requests.

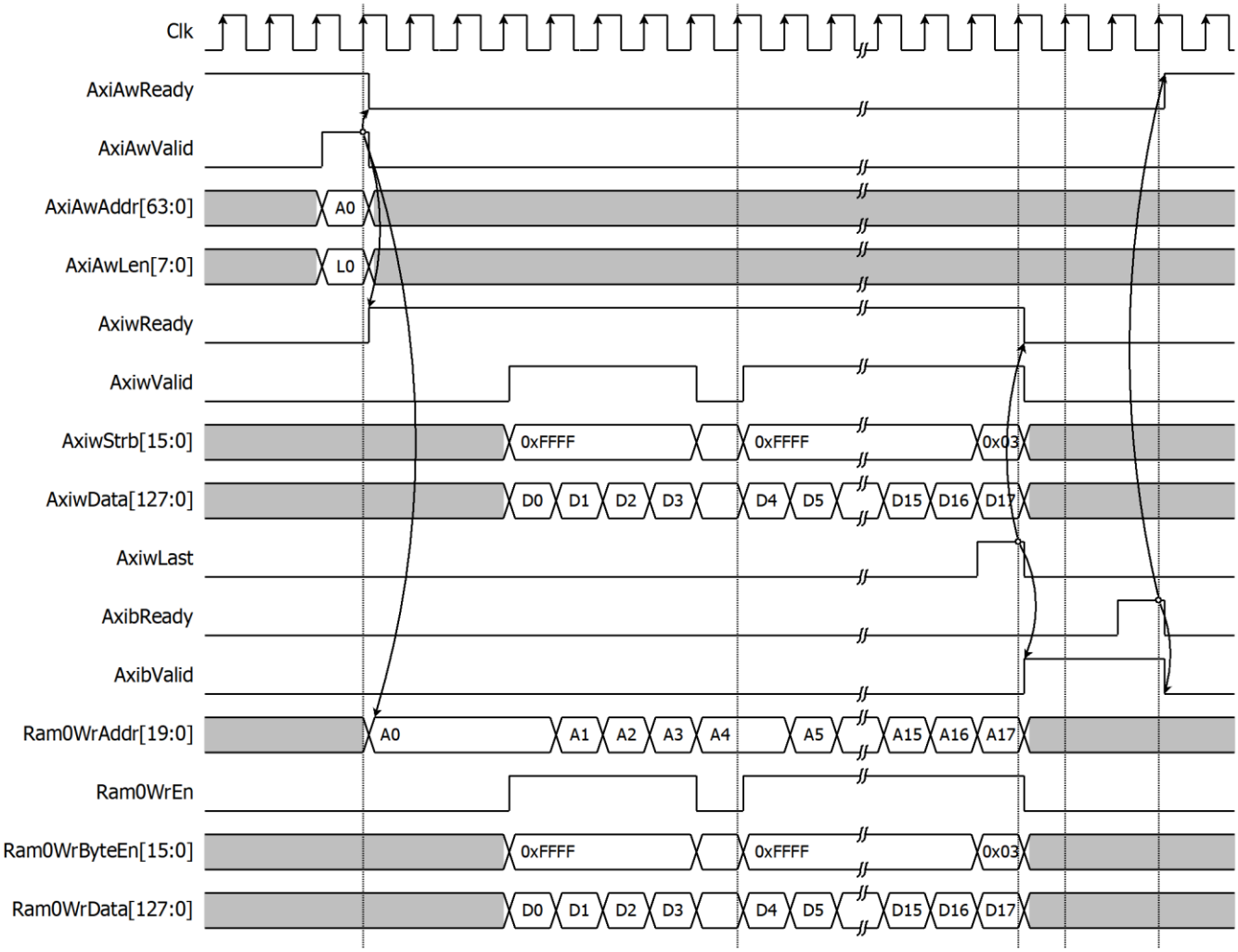

Figure 2-4 Example timing diagram of writing data to RxRam via AXI bridge

In the case of reading AXI read requests to the AXI bridge, AxiArReady will be de-asserted to '0' in the next clock cycle. The AXI bridge will set Ram1RdAddr[19:0] to AxiArAddr[19:0], positioning the first address to read data from TxRam. The AXI bridge will read data and store it in an internal buffer, and Ram1RdAddr[19:0] will increment by 16 until the read operation is finished. When the AXI bridge is ready to transfer data to the AXI master and the AXI master is ready to receive data, the AXI bridge will assert AxirValid to '1'. When the AXI bridge transfers data to the last word, it will assert AxirLast to '1' to specify the last cycle. When the AXI bridge sends AxirLast='1' and the AXI master sends AxiRReady='1', it signifies the completion of the read data operation, and the AXI bridge will set AxiArReady to '1' in the next clock cycle to accept new read requests.

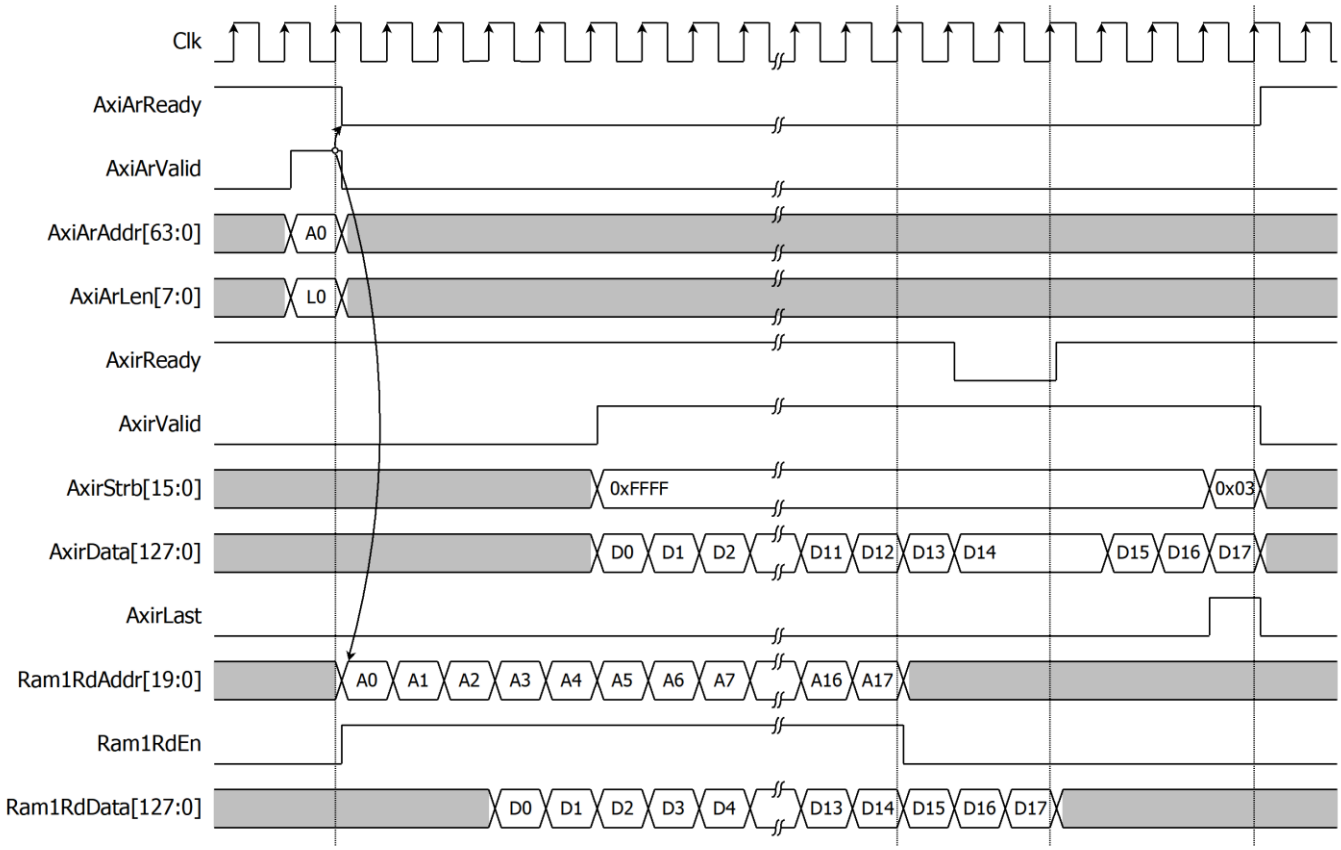

Figure 2-5 Example timing diagram of reading data to TxRam via AXI bridge

User Data Generator

In the reference design, a data pattern is generated and written to TxRam. There are two types of data patterns available: increasing and decreasing binary patterns. The user can set the type of data by writing to USER_TX_PATT_TYPE_REG, which is mapped to the rPattGenMode signal, supporting the generation of unaligned data. After setting the data pattern size in byte units to rPattGenLen[17:0] by writing to USER_TX_PATT_LEN_REG, the data pattern (rUserTxRamWrData[127:0]) and rUserTxRamWrByteEn[15:0] are prepared corresponding to the start address.

For example, if the user want to generate a data pattern for transmitting data in streamID0, user can set the start address to 0x1F and set rPattGenLen[17:0] to generate a 451-byte increasing binary pattern. rUserTxRamWrData[127:120] is set to 0x00 and rUserTxRamWrByteEn[15:0] is set to 0x8000 at the first clock cycle to write data only to the highest byte at rUserTxRamWrAddr[19:4]=0x0001. At the second clock cycle, every byte of the data pattern is written. At the last clock cycle, only the last 2 bytes of the data pattern are written: rUserTxRamWrData[15:0] is set to 0xC2C1 and rUserTxRamWrByteEn[15:0] is set to 0x0003, as shown in Figure 2-6.

The user can check if the data pattern write to TxRam is complete by verifying that rPattGenLen[17:0]=0, which can be read from USER_TX_PATT_LEN_REG. Once the data pattern generation is complete, the user can update the write pointer (rAppTxWrAddr0[17:0]) by writing to QUIC_TX_USER_PTR_REG0, indicating to QUIC10GC-IP that there is available Tx data to transmit. When the user wants to determine that the end of the data in the stream has been reached, they can assert rAppTxWrFin0 to '1' by writing to QUIC_TX_USER_FINAL_REG0.

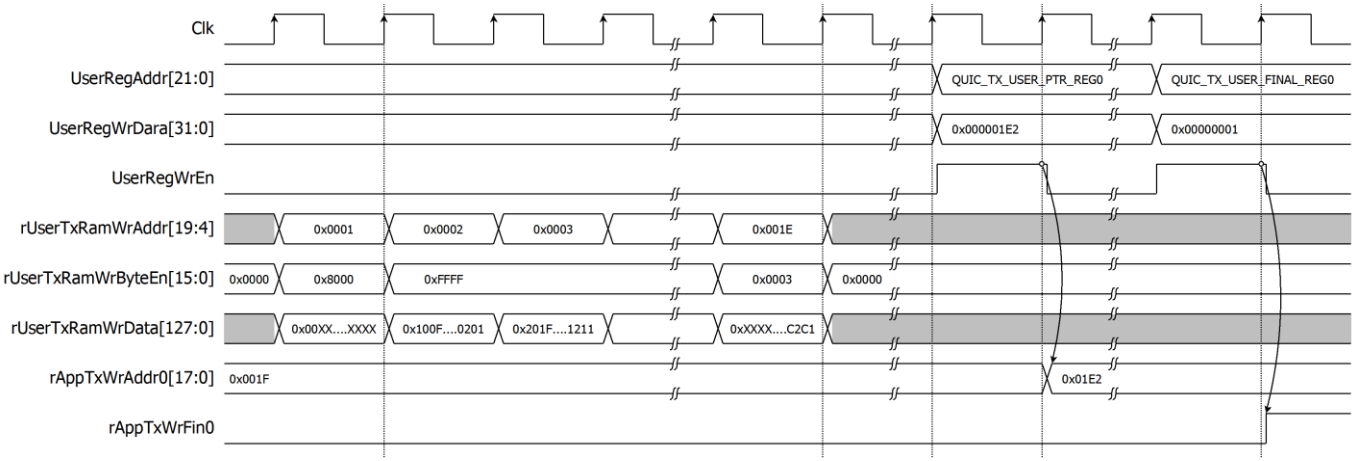

Figure 2-6 Example timing diagram of user data generation process

User Data Verification

In the reference design, a data verify pattern is used to verify data from RxRam. There are two types of expected data patterns: increasing and decreasing binary patterns. The user can set the type of data by writing to USER_RX_VERIFY_TYPE_REG, which is mapped to the rVerifyMode signal. This supports verifying unaligned data by reading data from RxRam via the UserRdIF. After setting the data verify size in byte units to rVerifyLen[17:0] by writing to USER_RX_VERIFY_LEN_REG, UserRdIF will read data from RxRam into VerifyRdData[127:0]. wVerifyExpData[127:0] is the expected data used for comparison, and wVerifyByteEn[15:0] is used to enable verification for each byte. wVerifyInvalid will be asserted to '1' when the verification is valid but not all bytes match.

For example, if the user wants to verify a data pattern from received data in streamID0, the user can set the start address to 0x1F and set rVerifyLen[17:0] to verify a 451-byte increasing binary pattern. UserRdIF will read data from RxRam into VerifyRdData[127:0] and compare it with wVerifyExpData[127:0] when wVerifyByteEn is active. At the last clock cycle, only the last 3 bytes of the data pattern are verified, wVerifyByteEn[15:0] is set to 0x0007, and wVerifyExpData[23:0] is set to 0xC2C1C0, as shown in Figure 2-7.

The user can check if the data pattern verification from RxRam is complete by verifying that rVerifyLen[17:0]=0, which can be read from USER_RX_VERIFY_LEN_REG. Once the data pattern verification is complete, the user can update the read pointer (rAppRxRdAddr0[17:0]) by writing to QUIC_RX_USER_PTR_REG0, indicating to QUIC10GC-IP that the data has been processed. If the other endpoint requests to close streamID0, the QUIC10GC-IP will set AppRxWrFin0 to '1', which can be read from QUIC_RX_USER_FINAL_REG0.

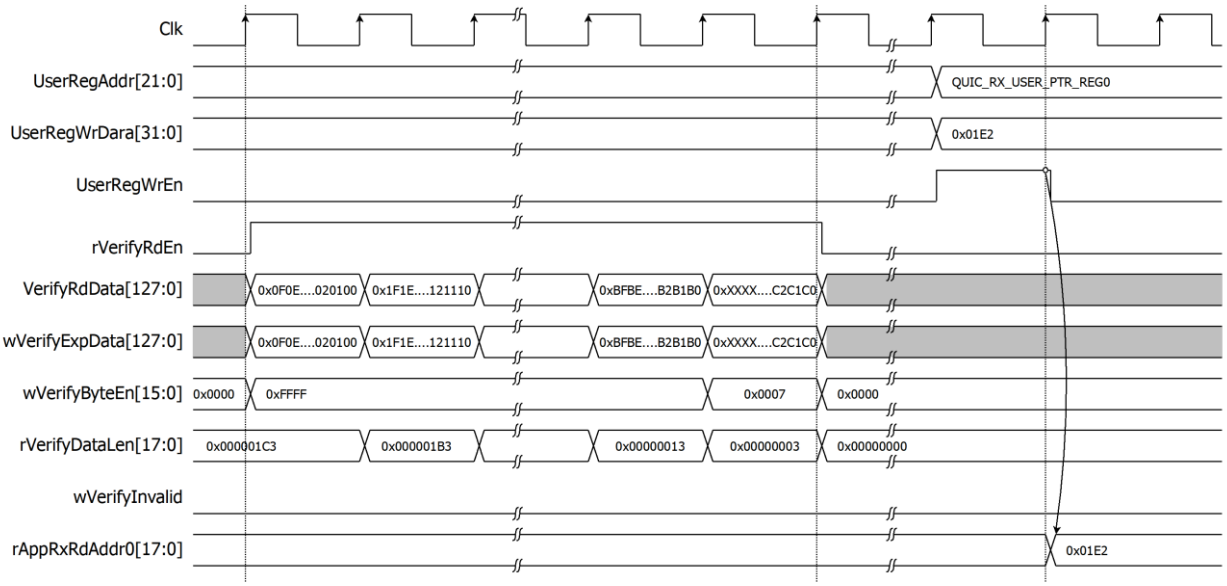

Figure 2-7 Example timing diagram of user data verification process

2.3 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/

2.4 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

2.5 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 network and connection parameter, display key materials and certificate information, download/upload data and test performance using the supported commands. More details of the sequence in each command are described as follows.

3.1 Set Gateway IP Address

command> setgatewayip ddd.ddd.ddd.ddd

Users can set a Gateway IP address for the QUIC10GC-IP by inputing setgatewayip followed by the desired Gateway IP address in dotted-decimal format. The setip function is called to change the Gateway IP address value in netparam variable. This variable will be written to the register mapped to GatewayIPAddr to set the FPGA's IP address. Subsequently, the QUIC10GC-IP is initialized with the current network parameter setting. The default Gateway IP address is 0.0.0.0. The setip function is described in Table 3-1.

3.2 Set FPGA's IP Address

command> setip ddd.ddd.ddd.ddd

Users can set an IP address for the QUIC10GC-IP by inputing setip followed by the desired IP address in dotted-decimal format. The setip function is called to change the IP address value in netparam variable. This variable will be written to the register mapped to SrcIPAddr to set the FPGA's IP address. Subsequently, the QUIC10GC-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(uint8_t *string, uint32_t *ip_set)	
Parameter	string: 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.

3.3 Set FPGA's MAC address

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

Users can set a MAC address to the QUIC10GC-IP 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 netparam variable. 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 80-01-02-03-04-05. The setmac function is described in Table 3-2.

Table 3-2 setmac function

int setmac(uint8_t *string, uint64_t *mac_set)	
Parameter	string: MAC address as string input from user mac_set: array stored mac address
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 Load network parameters

command> loadnetworkparameters

This command configures network parameters and must be run before connecting to the network. When executed, it sets the current Gateway IP address, FPGA's IP address, and FPGA's MAC address to the QUIC10GC-IP while the NetworkSet signal is asserted to '1'.

3.5 Set FPGA's Port Number

command> setport ddddd

Users can set a port number to the QUIC10GC-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 netparam variable. This variable will be written to the register mapped to SrcPort to set the FPGA's port number. Dynamic ports are in the range 49152 to 65535. If the 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 3-3.

Table 3-3 setport function

int setport(uint8_t *string, uint16_t *port_set)	
Parameter	string: 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.

3.6 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, bshowTrafficSecret. If bshowTrafficSecret is set to true, 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.

3.7 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, bshowCertificate. If bshowCertificate is set to true, certificate information will be displayed on the serial console after the certificate is ready during the handshake phase. Users can use the certificate information for further certificate validity verification.

3.8 Download data pattern with HTTP GET command

command> myGET https://ip:port/size

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

This command simulates the GET method of HTTP to download data from the server. The myGET function extracts the server's IP address and port number, sets the registers to open connections, and monitors the 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 an HTTP GET command from the URL.
3) Open connection and wait for the handshake process to finish.
4) Write the GET HTTP command to TxRam and set rAppTxWrAddr0[17:0] by writing QUIC_TX_USER_PTR_REG.
5) Monitor HTTP header response and validate HTTP data length.
6) Compute and display transfer speed on the serial console until the reception of data is complete. If the received data length is less than 4 kB, the received data will also be shown on the serial console.

Table 3-4 myGET function

int myGET(uint8_t *string)	
Parameter	string: 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. Monitor and display receiving result.

3.9 Upload data pattern with HTTP POST command

command> myECHO https://ip:port/echo size

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

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

1) Split the URL input and set network parameters corresponding to the URL.
2) Construct an HTTP POST command from the URL and request size.
3) Open connection and wait for the handshake process to finish.
4) Write the POST HTTP command to TxRam, enable the user data generator to write data to TxRam and set rAppTxWrAddr0[17:0] by writing QUIC_TX_USER_PTR_REG, and monitor upload status.
5) Monitor HTTP header response, validate HTTP data length and verification status.
6) Compute and display transfer speed on the serial console until the reception of data is complete. If the received data length is less than 4 kB, the received data will also be shown on the serial console.

Table 3-7 myPOST function

int myECHO(uint8_t *string, uint8_t *reqSize)	
Parameter	urlStr: URL as string input from user reqSize: Upload length as string input from user
Return value	0: Valid input, -1: Invalid input
Description	This function receives a URL string input from the user, validates it, constructs an HTTP POST command, and sends it before transferring data to the server. It monitors the number of transferred data bytes to show the transfer speed

3.10 Upload and Download data pattern like secnetperf

command> myPREF ip:port uploadlength downloadlength

Where	ip	represents server's ip address in dot-decimal notation
	port	represents server's port number
	uploadlength	represents upload data length in byte
	downloadlength	represents download data length in byte

This command uses for performance testing using the unique application protocol with MsQuic. myPREF function is called to extract the server's IP address and the server's port number, set registers to start data generator and verify data pattern, monitor status. The sequence of the myPREF function is as follows.

1) Split the URL input and set network parameters corresponding to the URL.
2) Open connection and wait for finishing handshake process.
3) Write the downloadlength to TxRam for request download data from MsQuic
4) If the uploadlength is greater than zero, this command will repeat to set USER_TX_PATT_LEN_REG for generating pattern data to TxRam, and move rAppTxWrAddr0[17:0] by writing QUIC_TX_USER_PTR_REG, until all pattern data is filled in TxRam. Then monitor the upload status.
5) If downloadlength is greater than zero, monitor verification status and remaining verify data length.
6) Compute and display transfer speed on the serial console until the reception of data is complete. If the received data length is less than 4 kB, the received data will also be shown on the serial console.

Table 3-7 myPERF function

int myPERF(uint8_t * netAddress, uint8_t * upSize, uint8_t * downSize)	
Parameter	netAddress: URL as string input from user upSize: Upload length as string input from user downSize: Download length as string input from user
Return value	0: Valid input, -1: Invalid input
Description	This function receives parameter string input by the user, validates the parameters before transferring data to the server, and monitors the number of transferred data to show the transfer speed.

4 Revision History

Revision	Date	Description
1.00	3-Jul-24	Initial version release

