TOE1G-IP Two-Port Demo Reference Design Manual

Rev1.4 8-Aug-23

1 Introduction

The Two-port reference design is implemented to support multiple TCP sessions through the cooperation of TOE1GIP and the CPU. The TCP sessions are categorized into two types: fast connections and slow connections. Each TOE1G-IP can handle only one fast connection, while the CPU handles the remaining TCP sessions which are slow connections.

This reference design demonstrates the usage of two TCP sessions for transferring two data types. The fast connection is applied for data transfer, while the slow connection is utilized to transfer control and status information.

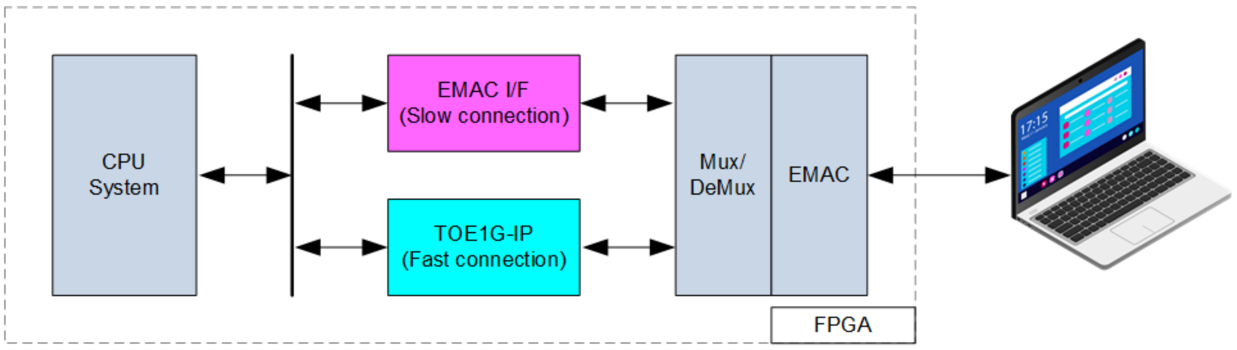


Figure 1-1 Two-port Hardware Structure

By incorporating the CPU system into the design, the Serial console is available to allow the user to set the test parameters and monitor the test status. Design Gateway provides two demo systems that share the same hardware but use different CPU firmware: the Two-port demo and the FTP server demo.

The Two-port demo offers three test menus for basic test operations: Receive data test via fast connection (TOE1G-IP), Send data test via fast connection (TOE1G-IP), and Data transfer test via slow connection (CPU). Design Gateway also provides test applications running on the PC, enabling users to send or receive TCP data with TOE1G-IP when they select the fast connection test. The slow connection can be tested using the Ping command. As a result, the CPU firmware of the Two-port demo implements two types of ICMP protocol: Echo reply and Echo request to support the Ping command.

The FTP server demo is developed using two TCP sessions. The Fast connection handles data transfer while the Slow connection is responsible for transferring FTP Commands and FTP Replies. For more information about the FTP server demo, please visit our website.
https://dgway.com/TOE-IP_X_E.html

The following sections will provide detailed descriptions of the hardware and the CPU firmware implemented in the Two-port demo system.

2 Hardware Structure

To support multiple sessions which are handled by different logics, the user interface of EMAC module is connected to two hardware sets: AXITOE (Fast connection) and AXIEMAC (Slow connection), as shown in Figure 2-1.

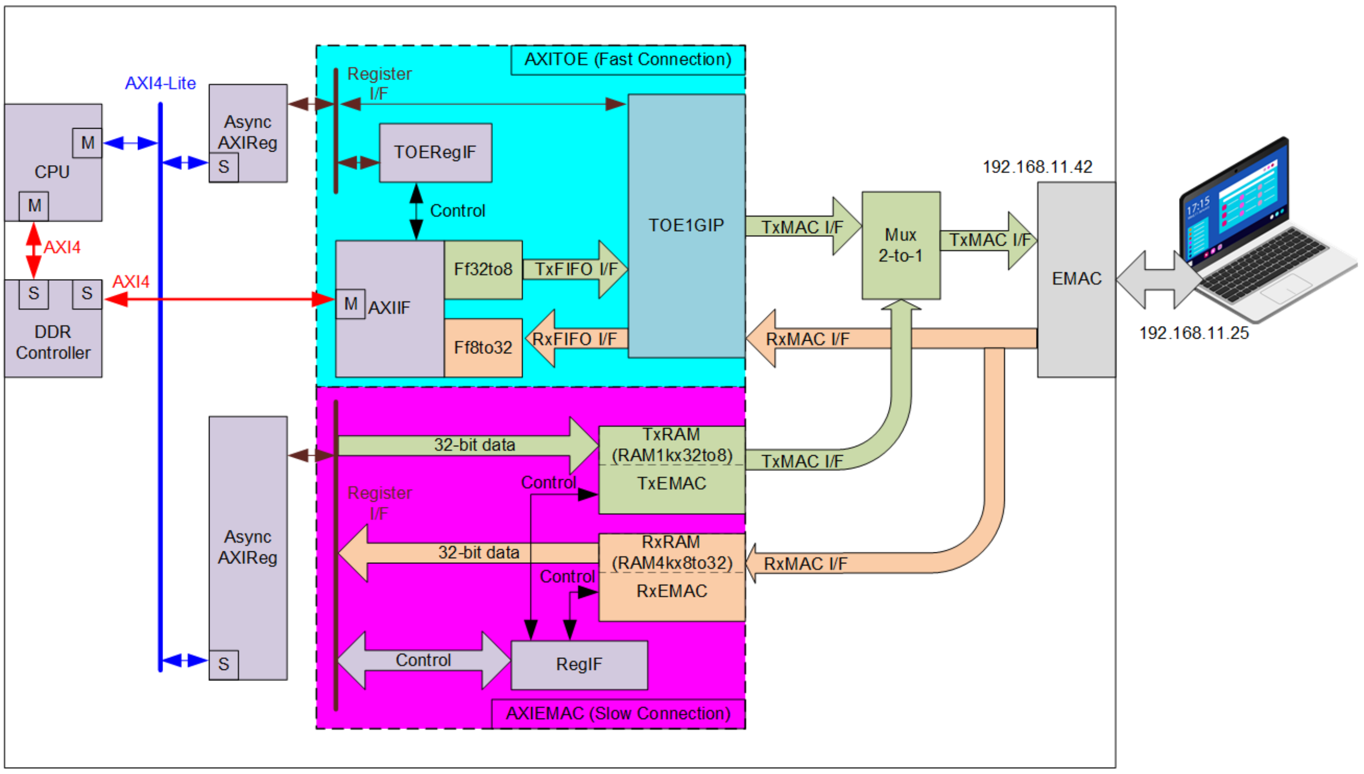


Figure 2-1 Two-port demo hardware block diagram

The CPU system utilizes the AXI4-Lite bus interface to connect additional I/O modules. Within this hardware system, two additional I/O interfaces are exported from the CPU system to connect with AXITOE and AXIEMAC. To facilitate this connection and accommodate different clock domains between the CPU system and the connected hardware module, an AsyncAXIReg module serves as the interface module between the CPU system via the AXI4-Lite bus and the hardware module via the Register interface.

Furthermore, the CPU system includes a DDR controller to enable data transfer with DDR memory. The DDR controller employs the AXI4 bus interface to connect to both the CPU and the hardware module. In this demo system, the DDR memory is utilized to store the data of the fast connection, making it accessible by the CPU or the AXITOE module.

The AXITOE module contains the TOE1G-IP, which processes one TCP session at the peak bandwidth of Gigabit Ethernet. The CPU assigns network parameters, including the port number of TOE1G-IP to handle the fast connection via the Register I/F. The data bus of TOE1G-IP is an 8-bit interface, which requires the user of two FIFOs (Ff32to8 and Ff8to32) to convert the 8-bit data stream at TOE1G-IP to a 32-bit data stream at the AXI4 interface and vice versa. An AXIIF module serves as the DMA engine responsible for data transfer between TOE1G-IP and DDR memory through Ff32to8 and Ff8to32. The CPU system also assigns DMA engine parameters, such as the data transfer size of the AXIIF module, through the TOERegIF module.

The AXIEMAC module includes the TxEMAC and RxEMAC modules, serving as the hardware engine for transferring Ethernet packets using the Ethernet MAC (EMAC). To transmit packets, the CPU prepares the transmitted packet data directly to TxRAM via Register I/F and sends a request to TxEMAC to initiate the transfer of the packet from TxRAM to EMAC. On the other hand, the received packet data, filtered by RxEMAC, is stored in RxRAM. The CPU can directly access RxRAM via the Register I/F to decode the received Ethernet packet. The parameters of both TxEMAC and RxEMAC can be configured by the CPU via the RegIF module.

There are two sources for transmitting Ethernet packet to EMAC via the TxMAC I/F, so a multiplexer (Mux 2-to-1) is employed to select the transmitted packet from one of them. Conversely, the RxMAC I/F for transferring received Ethernet packets from EMAC is directly mapped to both TOE1G-IP and RxEMAC modules. Both the TOE1G-IP and RxEMAC modules include the filtering logic to process only the desired TCP session, which is set by the CPU. The CPU assigns different port numbers to TOE1G-IP and RxEMAC modules to enable them to process Ethernet packets from different TCP sessions.

For detailed information of each hardware module, please refer to the following sections.

2.1 AXITOE

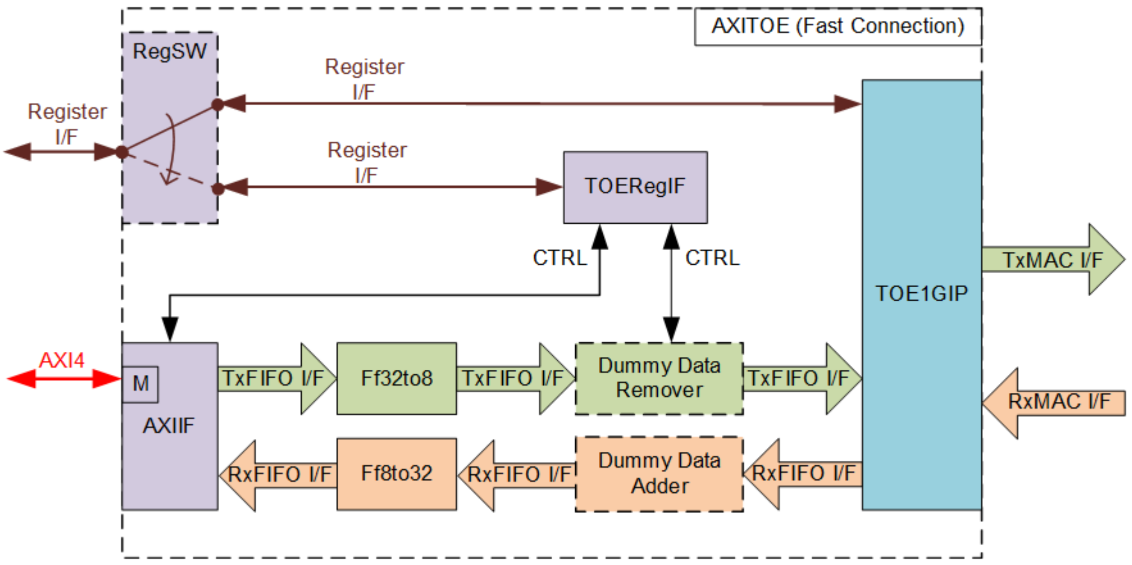


Figure 2-2 AXITOE block diagram

To efficiently handle the TCP/IP packets of the fast connection, additional logics are designed to cooperate with TOE1G-IP and the CPU system. The top module of AXITOE includes small logics with specific functions, including RegSW, Dummy Data Remover, and Dummy Data Adder.

The RegSW logic is responsible for selecting one of the two Register I/Fs (TOE1G-IP or TOERegIF) for operation when the CPU sends requests for write or read access at the AXITOE module. It decodes the address value to select the appropriate destination.

The Dummy Data Remover and Adder modules have been specifically designed to handle the removal and addition of dummy data to the data stream. The AXIIF functions as the DMA engine, and it is designed to fix the burst transfer size to be 512 bytes. When the total transfer size of data is not aligned to 512 bytes, dummy data is included. The Dummy Data Remover effectively removes any dummy data that is transferred from Ff32to8 to TOE1GIP for packet transmission. Conversely, the Dummy Data Adder automatically adds required dummy data that is transferred from TOE1GIP to Ff8to32.

Both Ff32to8 and Ff8to32 serve as asynchronous FIFOs, facilitating data transfer between AXIIF and TOE1G-IP in different clock domains. These asynchronous FIFOs buffer the data stream during the transfer process.

For more detailed information about each submodule within AXITOE, please refer to the following sections.

2.1.1 TOE1G-IP

TOE1G-IP implements TCP/IP stack to be the offload engine for transferring TCP/IP packets with the network device. User interface contains two signal groups: control signals and data signals. The control signal group uses the Register interface which is compatible to the single-port RAM interface while the data signal group uses the FIFO interface. Further information of TOE1G-IP is available on our website.
https://dgway.com/products/IP/TOE1G-IP/dg_toe1gip_data_sheet_xilinx_en/

2.1.2 AXIIF

The AXI4 bus standard enables data transfer in both directions simultaneously. This interface consists of four signal groups: AXIAw for Write address, AXIw for Write data, AXIAr for Read address, and AXIr for Read data.

The AXIIF logics can be categorized into two groups: AXIWr for Write transfer and AXIRd for Read transfer, each operating individually. The details of AXIWr and AXIRd are described below.

AXIRd

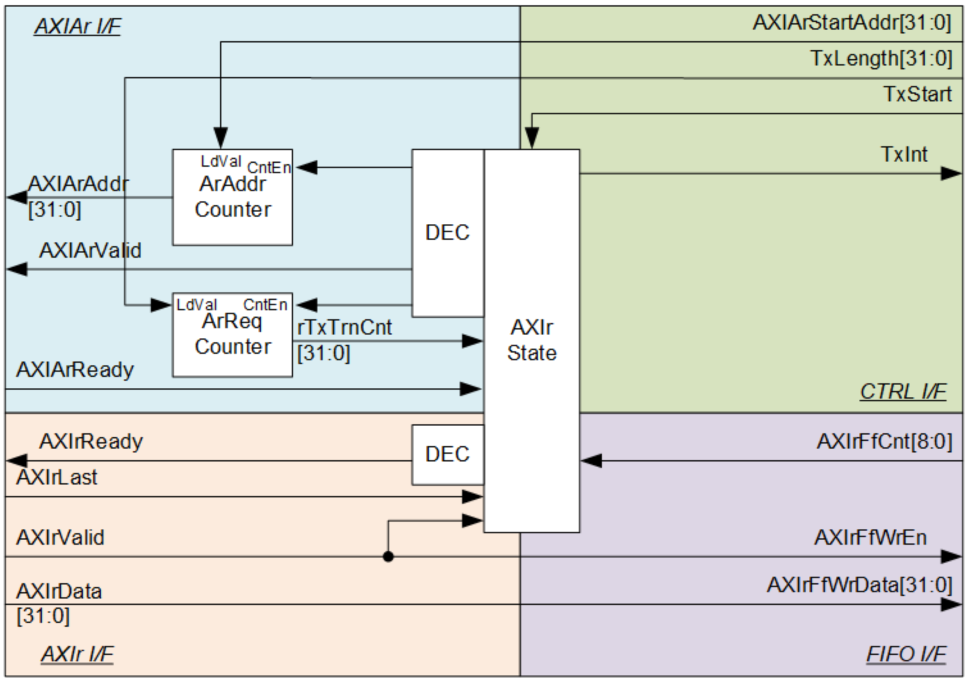


Figure 2-3 AXIRd Transfer Logic diagram

The AXIRd manages the sequence of each signal transmission using a state machine (AXIrState), which operates as follows.

1) Idle (stIdle): In this state, the state waits for a new request to initiate data transfer from the CTRL I/F. When the TxStart signal is asserted, the operation proceeds to the next step.
2) Check FIFO status (stWtDataRdy): In this state, the state reads AXIrFfCnt, which indicates the free space available in the FIFO for transferring 512-byte data. If there is enough free space, the state proceeds to the next step.
3) Address request (stSendReq): In this state, the state generates a read request to the AXI4 interface by asserting AXIArValid along with the address value on AXIArAddr. The request remains asserted until the request is accepted, indicated by AXIArReady being asserted. The ArReq Counter is then decremented to show the remaining transfer size, and the state proceeds to the next step.
4) Data transfer (stRcvData): In this state, the state waits until all 512-byte data is completely transferred from the AXIr I/F to the FIFO I/F. The transfer is completed when AXIrLast is asserted, indicating the last data transfer. The ArAddr Counter is incremented to indicate the next address for reading.
   Note: The address counter updates its value after finishing the read data transfer to indicate the completed read position of the DDR memory.
5) Check remaining length (stChkEnd): In this state, the state reads the remaining transfer size (rTxTrnCnt) to determine if there is any remaining data for the next transfer. If there is remaining data, the state returns to step 2). Otherwise, it returns to the Idle state and asserts the interrupt signal (TxInt) to indicate that the operation is completed.

As shown in Figure 2-3, both the data valid (AXIrValid) and the read data (AXIrData) from the AXIr I/F are directly mapped to the FIFO write enable (AXIrFfWrEn) and the write data (AXIrFfWrData) of the FIFO I/F. The burst size of the AXIr I/F is fixed at 512 bytes.

AXIWr

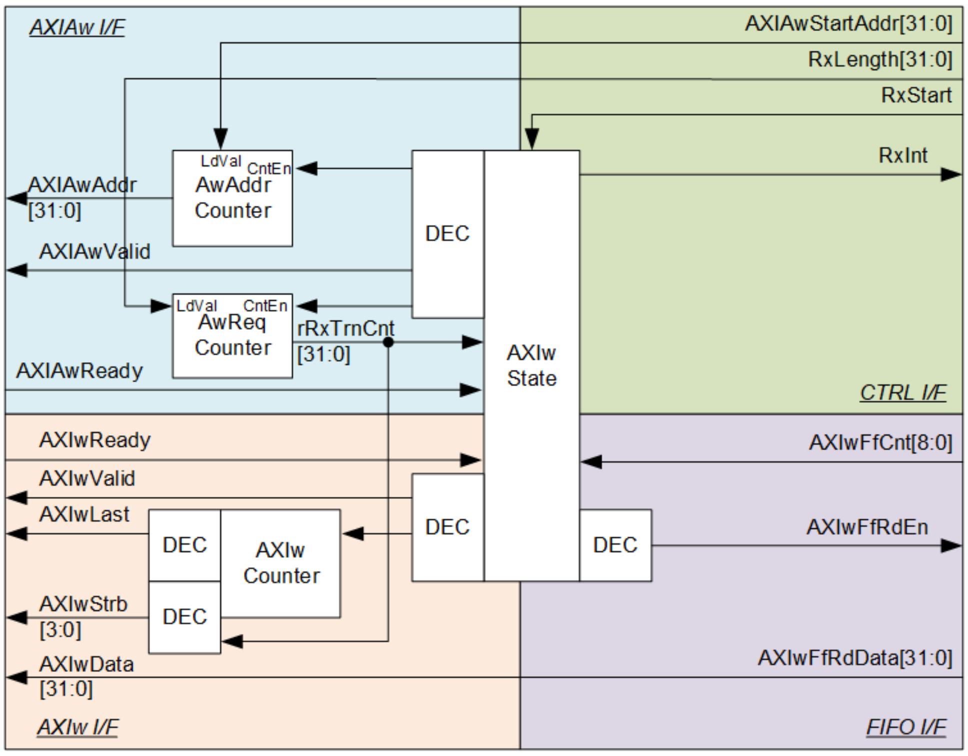


Figure 2-4 AXIWr Transfer Logic diagram

The AXIWr manages the sequence of each signal transmission using a state machine (AXIwState), which operates as follows.

1) Idle (stIdle): In this state, the state waits for a new request to initiate data transfer from the CTRL I/F. When the RxStart signal is asserted, the operation proceeds to the next step.
2) Check FIFO status (stWtDataRdy): In this state, the state reads AXIwFfCnt, which indicates the amount of data in the FIFO for transferring 512-byte data. If there is at least 512-byte data available, the state proceeds to the next step.
3) Address request (stSendReq): In this state, the state generates a write request to the AXI4 interface by asserting AXIAwValid along with the address value on AXIAwAddr. The request remains asserted until the request is accepted, indicated by AXIAwReady being asserted. The AwReq Counter is then decremented to show the remaining transfer size, and the state proceeds to the next step.
4) Data transfer (stSendData): In this state, the state waits until all 512-byte data is completely transferred from the FIFO I/F to the AXIw I/F. The transfer is completed when both AXIwReady and AXIwLast are asserted, indicating the last data transfer.
5) Wait response (stWtResp): In this state, the state waits until the AXI I/F returns the response to confirm the successful write operation. Afterward, the AwAddr Counter is incremented to indicate the next address for reading.
   Note: The address counter updates its value after completing the write data transfer to indicate the completed write position of the DDR memory.
6) Check remaining length (stChkEnd): In this state, the state reads the remaining transfer size (rRxTrnCnt) to determine if there is any remaining data for the next transfer. If there is remaining data, the state returns to step 2). Otherwise, it returns to the Idle state and asserts the interrupt signal (RxInt) to indicate that the operation is completed.

Additionally, Figure 2-4 illustrates the details of the logics that forwards the data stream from FIFO I/F to AXIw I/F. The data bus is directly mapped from the read data of the FIFO I/F (AXIwFfRdData) to the write data of the AXIw I/F (AXIwData). AXIwReady is used as data flow control. When de-asserted, the read enable of the FIFO I/F (AXIwFfRdEn) is also de-asserted to pause data transmission and latch the data during this pause time. The number of cycles to transfer data in each loop is fixed to 128 beats of 32-bit data; however, AXIwStrb is designed to be de-asserted before the end of the transfer if the transfer size is less than 512-bytes.

2.1.3 TOERegIF

The CPU communicates with the AXITOE through AsyncAXIReg, which converts the AXI4-Lite interface to be the Register I/F for simple connection. The memory area of the AXITOE that is mapped to the CPU system is divided into two areas: the TOE1G-IP area and the TOERegIF area. The address map of the Register I/F to each AXITOE's signal is shown in Table 2-1.

The TOERegIF consists of an address decoder and the register files, responsible for storing test parameters and the control signals used within the AXITOE. To accommodate the different clock domains of the TOE1G-IP and AXIIF modules, asynchronous circuits are integrated into the TOERegIF to interface with the signals of the AXIIF module. The Start signal and transfer length for each transfer direction are generated independently in both the TOE1G-IP and AXIIF clock domains. These signals are then utilized by the Dummy Data Remover/Adder and AXIIF modules, respectively.

Table 2-1 Memory Map of TOERegIF within AXITOE module

Address / Rd/Wr	Register Name / Label in the "ftp_demo.c/twoport.c"	Description
(BA1+0x0000) – (BA1+0x00FF): TOE1G-IP Register Area More details of each register are described in TOE1G-IP datasheet.		
BA1+0x0000	TOE_RST_REG	Mapped to RST register within TOE1G-IP
BA1+0x0004	TOE_CMD_REG	Mapped to CMD register within TOE1G-IP
BA1+0x0008	TOE_SML_REG	Mapped to SML register within TOE1G-IP
BA1+0x000C	TOE_SMH_REG	Mapped to SMH register within TOE1G-IP
BA1+0x0010	TOE_DIP_REG	Mapped to DIP register within TOE1G-IP
BA1+0x0014	TOE_SIP_REG	Mapped to SIP register within TOE1G-IP
BA1+0x0018	TOE_DPN_REG	Mapped to DPN register within TOE1G-IP
BA1+0x001C	TOE_SPN_REG	Mapped to SPN register within TOE1G-IP
BA1+0x0020	TOE_TDL_REG	Mapped to TDL register within TOE1G-IP
BA1+0x0024	TOE_TMO_REG	Mapped to TMO register within TOE1G-IP
BA1+0x0028	TOE_PKL_REG	Mapped to PKL register within TOE1G-IP
BA1+0x002C	TOE_PSH_REG	Mapped to PSH register within TOE1G-IP
BA1+0x0030	TOE_WIN_REG	Mapped to WIN register within TOE1G-IP
BA1+0x0038	TOE_SRV_REG	Mapped to SRV register within TOE1G-IP
BA1+0x003C	TOE_VER_REG	Mapped to VER register within TOE1G-IP
(BA1+0x0100) – (BA1+0x011B): Other control/status of AXITOE		
BA1+0x0100 Wr/Rd	Start Transmit DDR Address (TX_DDR_ADDR)	Wr [31:0] – Start DDR Address for transmitting data (DDR to TOE1G-IP) It must be aligned to 32 bits or bit[1:0] must be equal to 00b. Rd [31:0] – Current value of transmitted DDR Address
BA1+0x0104 Wr	Transmit Length (TX_DDR_LEN)	Wr [31:0] – Total amount of transmitted data in byte unit (DDR to TOE1G-IP)
BA1+0x0108 Wr/Rd	Start Receive DDR Address (RX_DDR_ADDR)	Wr [31:0] – Start DDR Address for receiving data (TOE1G-IP to DDR) It must be aligned to 32 bits or bit[1:0] must be equal to 00b. Rd [31:0] – Current value of received DDR Address
BA1+0x010C Wr	Receive Length (RX_DDR_LEN)	Wr [31:0] – Total amount of received data in byte unit (TOE1G-IP to DDR)
BA1+0x0110 Wr/Rd	AXITOE Control (AXITOE_CTRL)	Wr [0] – Asserted to initiate Tx transfer from DDR to TOE1G-IP. [1] – Asserted to initiate Rx transfer from TOE1G-IP to DDR. Bit[1] and bit[0] are automatically cleared to 0b after user asserts them to 1b. Rd [0] – Transmit busy flag. Asserted during data transfer from DDR to TOE1G-IP. [1] – Receive busy flag. Asserted during data transfer from TOE1G-IP to DDR.
BA1+0x0114 Rd	ConnOn of TOE1G-IP CONNON_REG	Rd [0] – Mapped to ConnOn signal output from TOE1G-IP to show connection status. Asserted when the connection has been established.
BA1+0x0118 Rd	FIFO Read Count of TOE1G-IP FFRDCNT_REG	Rd [15:0] – Mapped to TCPRxFfRdCnt signal output from TOE1G-IP to show total number of received data in byte unit

Note: BA1 is base address of AXITOE that is mapped to the CPU system

2.2 AXIEMAC

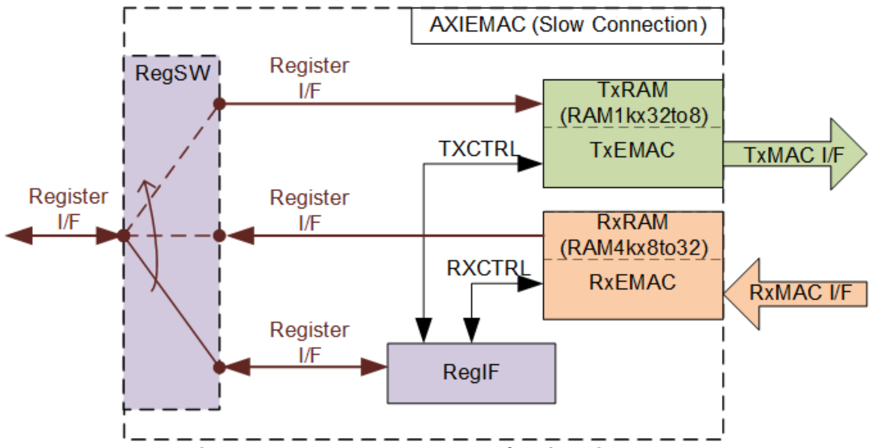


Figure 2-5 AXIEMAC Block Diagram

The AXIEMAC module is designed to facilitate the transfer of Ethernet packets using the Ethernet MAC. The CPU directly manages this module through Register I/F. Within the AXIEMAC, two engines, called TxEMAC and RxEMAC, are the logics responsible for transferring data between the RAM and the Ethernet MAC. There is RAM located within TxEMAC and RxEMAC which are directly mapped to Register I/F. This allows the CPU to have direct control of these RAMs.

Furthermore, RegIF is also connected to the Register I/F to store test parameters, control signals, and status signals of each sub module. The CPU can configure each parameter value to match the specific requirements.

The detailed descriptions of each submodule within the AXIEMAC are provided below.

2.2.1 TxEMAC

TxEMAC is designed to transmit Ethernet packets to the EMAC. It incorporates an asymmetric RAM, known as TxRAM or RAM1kx32to8, which serves as the Tx buffer for the Ethernet packet transmission. Also, TxRAM is responsible for converting data width from 32 bits to 8 bits. The Write interface of TxRAM is connected to the CPU system via Register I/F, while specific logics are designed to interface with the Read interface of TxRAM.

The functions of the logics at the Read interface of TxRAM are outlined below.

1) Address counter (rTxRamRdAddr): This counter generates the read address of TxRAM. It loads the initial value from the RegIF and increments to read the next data from TxRAM after the EMAC confirms the successful reception of each data by asserting MacTxReady to 1b to.
2) Length counter (rTxLengthCnt): This counter controls the number of transmitted data in each packet. The transfer size of each packet is initially loaded from the RegIF and is decremented after each data is successfully sent to the EMAC.
3) Data stream controller: The transmitted data to the EMAC (MacTxData) is directly fed by the read data of TxRAM (TxRamRdData). Additionally, specific logics are designed to generate the data control signals, including valid (MacTxValid), start-of-packet (MacTxSOP), and end-of-packet (MacTxEOP).

The transmit operation is initiated when the CPU sets the start flag to 1b. The CPU can monitor the progress of the operation by reading the remaining length from the length counter. The remaining length will become zero once all data has been successfully transferred.

2.2.2 RxEMAC

The RxEMAC module is responsible for processing received packets from the EMAC and storing them in the RxRAM, which is an asymmetric RAM module integrated within RxEMAC. The read interface of RxRAM is directly connected to the CPU, which utilizes a 32-bit data bus. On the other hand, the write interface of RxRAM is connected to the EMAC which uses 8-bit data bus.

Upon receiving each packet from the EMAC, the internal logics of RxEMAC store the new packet to RxRAM. The write address is incremented after receiving each data until the end of the packet is reached. During the write process, the packet header is verified by the filtering logic to validate its contents. The CPU configures all values used for comparison with the packet header. These values include 48-bit Destination MAC address, 32-bit Destination IP address, 8-bit Protocol value, and 16-bit Destination port number.

The CPU has the option to enable or disable the verification of the Protocol value and Destination port number, assigned by three-bit signals, while the verification of other parameters is always operated. Additionally, the IP header must be IPv4 type and the IP header length must be equal to 20 bytes.

For instance, to support the Ping command, the CPU configures the parameters for header verification using the following values.

- Protocol	= ICMP (RxPatt23=01h, RxPattEn[0]=1b)
- Destination port number	= Disable (RxPattEn[2:1]=00b)

If the packet header is valid, the RxEMAC module stores the latest write address of RxRAM after receiving the end of packet into a latched register (rRxRamWrEndAddrLat). The CPU monitors the value of this latched register to indicate the reception of a new valid packet. After that, the CPU initiates the process to decode the new packet.

However, if the packet is determined to be invalid during the header verification, the write address of RxRAM will be reverted to the previous value, which corresponds to the start position before receiving the latest packet. Subsequently, the next received packet will replace the invalid one in RxRAM.

2.2.3 RegIF

The CPU communicates with the AXIEMAC through AsyncAXIReg, which converts the AXI4-Lite interface to be the Register I/F for simple connection. The memory area of the AXIEMAC that is mapped to the CPU system is divided into three areas: the TxRAM area, the RxRAM area, and the RegIF area. The address map of the RegIF is shown in Table 2-2.

The RegIF consists of an address decoder and the register files, responsible for storing test parameters and the control signals used within the TxEMAC and RxEMAC.

Table 2-2 Memory Map of RegIF within AXIEMAC module

Address Rd/Wr	Register Name (Label in the "ftp_demo.c/twoport.c")	Description
BA0+0x00000-BA0+0x00FFF Wr	TxRAM in TxEMAC (TXRAM)	4Kbyte RAM to store the transmitted Ethernet packet to the EMAC. It is applied for operating the Slow connection, controlled by the CPU.
BA0+0x10000-BA0+0x10FFF Rd	RxRAM in RxEMAC (RXRAM)	4Kbyte RAM to store the received Ethernet packet from the EMAC. It is applied for operating the Slow connection, controlled by the CPU.
BA0+0x20000 Wr	Destination MAC address [31:0] (DSTMACL_REG)	[31:0] - 32 lower bits of destination MAC address, assigned to filtering packet inside RxEMAC.
BA0+0x20004 Wr	Destination MAC address [47:32] (DSTMACH_REG)	[15:0] - 16 upper bits of destination MAC address, assigned to filtering packet inside RxEMAC.
BA0+0x20008 Wr	Destination IP address (DSTIP_REG)	[31:0] - 32-bit destination IP address, assigned to filtering packet inside RxEMAC.
BA0+0x20010 Wr	Rx Pattern enable (RXPATTEN_REG)	Enable flag to verify certain bytes inside the header of the received packet (1b: enable, 0b: disable). [0] – Asserted to compare byte#23 of received packet with RXPATT23_REG[7:0] [1] – Asserted to compare byte#36 of received packet with RXPATT32_REG[7:0] [2] – Asserted to compare byte#37 of received packet with RXPATT32_REG[15:8]
BA0+0x20014 Wr	Rx Pattern Byte 23 (RXPATT23_REG)	[7:0]: Expected value for comparing with byte#23 of received packet
BA0+0x20018 Wr	Rx Pattern Byte 34 (RXPATT36_REG)	[7:0] - Expected value for comparing with byte#36 of received packet [15:8] - Expected value for comparing with byte#37 of received packet
BA0+0x20100 Wr	Start TXRAM address (TXADDR_REG)	[11:0] - Start address of TxRAM to transmit Ethernet packet to EMAC
BA0+0x20104 Wr/Rd	Transmit length (TXLEN_REG)	Wr [11:0] - Transmitted Ethernet packet size in byte unit. Valid form 2-4095. The TXEMAC initiates data transfer after this register is set to value other than 0 or 1. Rd [11:0] – The remaining transmitted size in byte unit. The CPU detects the end of packet transmission by polling this register until the read value becomes 0.
BA0+0x20200 Rd	Received RXRAM address (RXADDR_REG)	The latest write address of RxRAM to store the received packet. The CPU reads this register until its value changes, indicating the arrival of a new packet. Additionally, this value is used to calculate the total data size by determining the difference between the current value and the previous value stored in the register.

Note: BA0 is base address of AXIEMAC that is mapped to the CPU system.

2.3 AsyncAXIReg

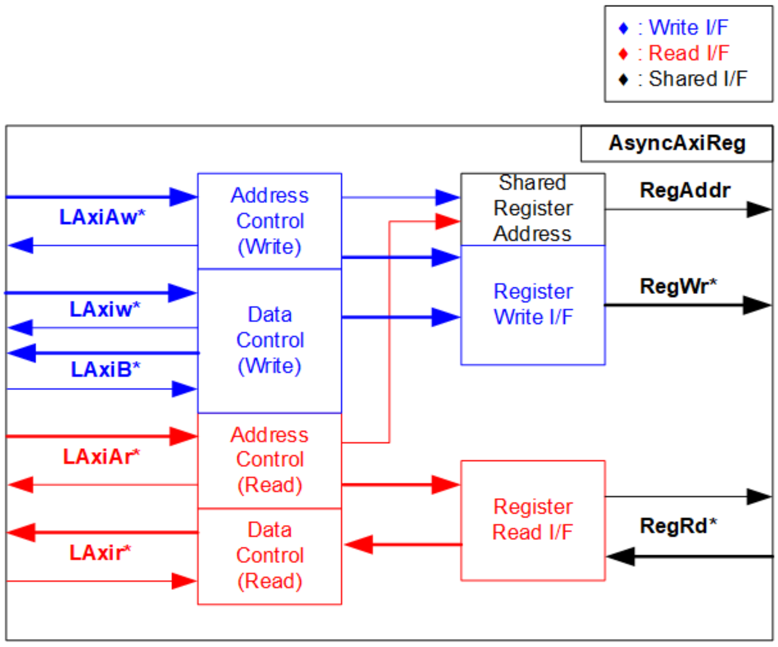


Figure 2-6 AsyncAxiReg interface

The AXI4-Lite bus interface consists of five groups: LAxiAw* (Write address channel), LAxiw* (Write data channel), LAxiB* (Write response channel), LAxiAr* (Read address channel), and LAxir* (Read data channel). The details to build custom logic for AXI4-Lite bus can be found in the following document.
https://github.com/Architech-Silica/Designing-a-Custom-AXI-Slave-Peripheral/blob/master/designing_a_custom_axi_slave_rev1.pdf

According to the AXI4-Lite standard, the write and read channels operate independently for both control and data interfaces. Therefore, the logic within AsyncAxiReg, which interfaces with the AXI4-Lite bus, is divided into four groups: Write control logic, Write data logic, Read control logic, and Read data logic, as illustrated on the left side of Figure 2-6.

The Write control I/F and Write data I/F of the AXI4-Lite bus are latched and transferred to become the Write register interface with the use of clock domain crossing registers. Similarly, the Read control I/F of the AXI4-Lite bus is latched and transferred to the Read register interface, while Read data is returned from Register interface to AXI4-Lite via clock domain crossing registers. In the Register interface, RegAddr serves as a shared signal for write and read access, loading the value from LAxiAw for write access or LAxiAr for read access.

The Register interface is compatible with a single-port RAM interface for write transaction. However, the read transaction of the Register interface has been slightly modified from the RAM interface by adding the RdReq and RdValid signals to control read latency time. Since the address of Register interface is shared for both write and read transactions, user cannot write and read register simultaneously. The timing diagram of the Register interface is shown in Figure 2-7.

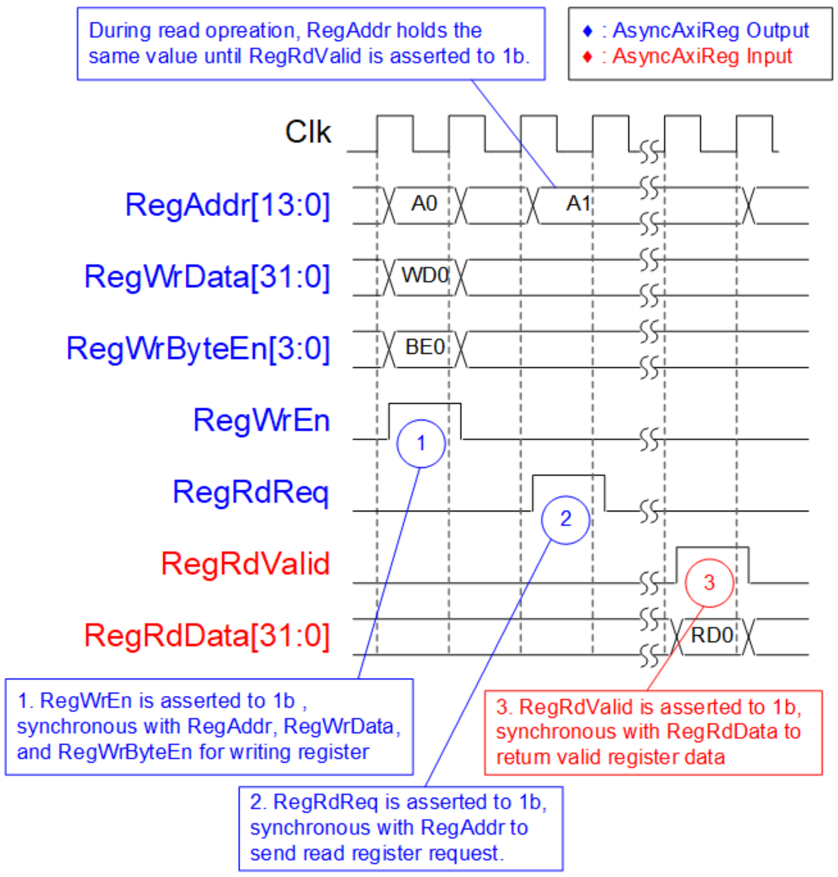

During read opreation, RegAddr holds the same value until RegRdValid is asserted to 1b.
♦ : AsyncAxiReg Output
♦ : AsyncAxiReg Input
Clk
RegAddr[13:0]
A0
A1
RegWrData[31:0]
WD0
RegWrByteEn[3:0]
BE0
RegWrEn
RegRdReq
RegRdValid
RegRdData[31:0]
RD0
1
2
3
1. RegWrEn is asserted to 1b , synchronous with RegAddr, RegWrData, and RegWrByteEn for writing register
3. RegRdValid is asserted to 1b, synchronous with RegRdData to return valid register data
2. RegRdReq is asserted to 1b, synchronous with RegAddr to send read register request.

Figure 2-7 Register interface timing diagram

1) The timing diagram for writing to the register is similar to that of a single-port RAM. When initiating a write operation, the RegWrEn signal is set to 1b, along with valid RegAddr (Register address in 32-bit units), RegWrData (write data for the register), and RegWrByteEn (write byte enable). The byte enable consists of four bits that indicate the validity of the byte data. For instance, bit[0], [1], [2], and [3] are set to 1b when RegWrData[7:0], [15:8], [23:16], and [31:24] are valid, respectively.
2) To read register, AsyncAxiReg sets the RegRdReq signal to 1b with a valid value for RegAddr. The 32-bit data is then returned after the read request is received. The slave detects the RegRdReq signal being set to start the read transaction. In the read operation, the address value (RegAddr) remains unchanged until RegRdValid is set to 1b. The address can then be used to select the returned data using multiple layers of multiplexers.
3) The slave returns the read data on RegRdData bus by setting the RegRdValid signal to 1b. After that, AsyncAxiReg forwards the read value to the LAxir* interface.

3 CPU Firmware

The CPU in this reference design operates using a bare-metal OS, which facilitates hardware handling. After the test system is booted, the initialization of the system follows these steps.

1) The CPU sets the initial values of the network parameters of TOE1G-IP such as MAC address, IP address, and port number. The initial values can be updated in the CPU firmware.
2) The CPU sets the initial values of the network parameters and expected values of the received packet header for ICMP packet to the AXIEMAC module.
3) The CPU de-asserts the reset signal of TOE1G-IP to initiate the IP initialization process and then checks the busy status to determine when the initialization process is completed.
4) The test menu is displayed on the console, offering three test options.
   a) Receive data test (Target -> TOE1G-IP) using the fast connection
   b) Send data test (TOE1G-IP -> Target) using the fast connection
   c) Ping Test using the slow connection

When executing the Receive data test or Send data test using the fast connection, half of the DDR memory size is utilized as the data buffer. For instance, the KCU105 board has 2 GB DDR memory, only 1 GB DDR is allocated as the data buffer for data transfer over the fast connection.

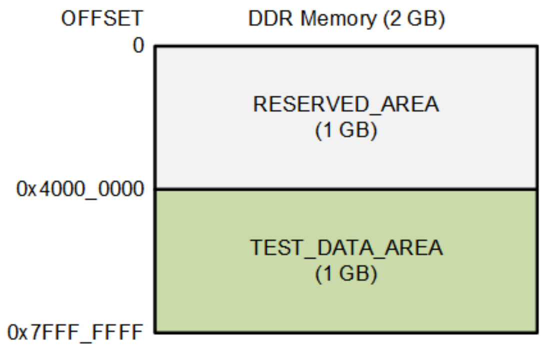


Figure 3-1 DDR Memory Map

3.1 Receive Data Test

This menu is selected to execute the receive operation of TOE1G-IP. In our test environment, the target device (the PC) runs the "send_tcp_client" application to transmit test data through a specific TCP port (port 4000: the port number assigned to TOE1G-IP). If the total transfer size exceeds the allocated DDR area, the data verification function within the CPU firmware will be disabled. Conversely, if the data size is within the memory capacity, the console asks the user to proceed with the data verification process or finish the test operation. The firmware steps are outlined as follows.

1) The CPU monitors CONNON_REG to wait for the establishment of a new connection by the target device (PC). When the PC runs the "send_tcp_client" application and the connection is successfully established, the read value of CONNON_REG becomes equal to 1. If user enters any key during this period, the operation is cancelled.
2) The CPU reads FFRDCNT_REG to determine the amount of received data from TOE1G-IP and also checks CONNON_REG to decide the next action following these steps.
   i) If the connection has been closed (CONNON_REG=0) and there is no remaining received data in TOE1G-IP (FFRDCNT_REG=0), the process proceeds to step 3).
   ii) The CPU calculates the data size to be transferred from TOE1G-IP to DDR memory using the following conditions.
      - If the amount of received data exceeds the free space of DDR, the data size is set to be equal to the available free space of DDR.
      - If the free space of DDR is sufficient and this is last transfer (the connection has been closed), the data size is set to be equal to the total amount of received data (FFRDCNT_REG).
      - Otherwise, the data size is determined by the amount of received data, which is aligned to 512-byte size.
   iii) The result is set to RX_DDR_LEN register, the start DDR address value is updated to RX_DDR_ADDR, and the start flag of Rx transfer (AXITOE_CTRL[1]=1b) is asserted. Subsequently, the AXIIF initiates data transfer from TOE1G-IP to DDR.
   iv) The CPU waits until the data transfer is completed, indicated by AXITOE_CTRL[1]=0, and then returns to step i) for the next data transfer.
3) The total transfer size is displayed on the console. If the total transfer size does not exceed the allocated DDR area, the console offers the option to start data verification on the DDR. Otherwise, the operation is finished.
4) If the user chooses to initiate data verification, the CPU reads and verifies data at DDR using 32-bit incremental pattern. In case any mismatched data is found, an error message will be displayed. The operation is finished once all the last data has been verified.

3.2 Send Data Test

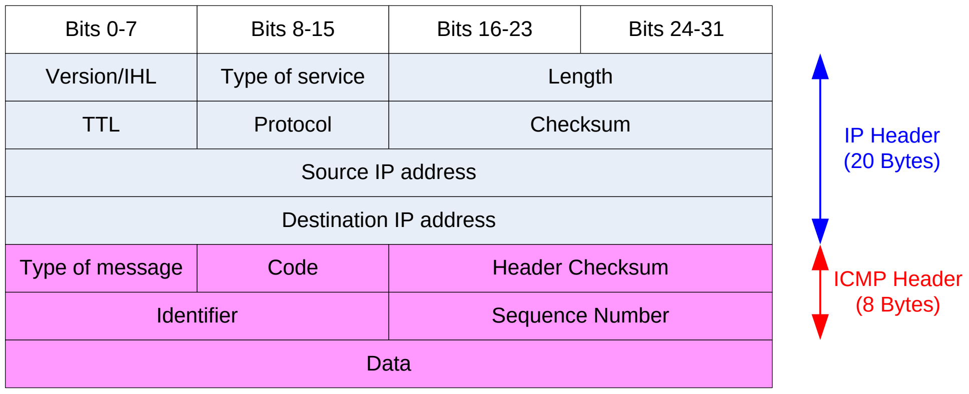
This menu is selected to execute the transmit operation of TOE1G-IP. In our test environment, the target device (the PC) runs the “recv_tcp_client_single” application to receive test data through a specific TCP port (port 4000: the port number assigned to TOE1G-IP). The user has the option to enable or disable data verification for the “recv_tcp_client_single” application.

On the FPGA console, the user can specify the total transfer size for the test. If the total transfer size does not exceed the allocated DDR area, the user has the option to prepare test data in the DDR or proceed without preparing the data. However, if the transfer size exceeds the allocated DDR area, the operation begins using dummy data. The firmware steps to execute the Send data test are outlined as follows.

1) Receive the packet size and transfer length from the user. If the inputs are valid, assign them to TOE_PKL_REG and TOE_TDL_REG.
2) If the transfer length does not exceed the allocated DDR area, the console displays the option to prepare the test data (using a 32-bit incremental pattern) to the DDR. If the user chooses data preparation, the CPU initiates a loop to fill the test data into DDR until the process is completed.
3) Calculate the number of test loops required for transferring data from DDR to TOE1G-IP. If total transfer size exceeds the allocated DDR area, multiple test loops are necessary.
4) The CPU monitors CONNON_REG to wait for the establishment of a new connection by the targe device (PC). Once the PC runs the “recv_tcp_client_single” application and the connection is successfully established, the read value of CONNON_REG becomes equal to 1. If user enters any key during this period, the operation is cancelled.
5) Set TOE_CMD_REG to initiate the transmit operation of TOE1G-IP.
6) The CPU controls data transfer from DDR to TOE1G-IP register by following steps. The number of loops to operate these steps is calculated in step 3).
   i) Verify that the connection is still active (CONNON_REG=1b) and the AXIIF is not busy. If the AXIIF is still busy (AXITOE_CTRL[0]=1b), the CPU waits until it becomes de-asserted.
   ii) Set the start address of DDR and the transfer size of this loop to TX_DDR_ADDR and TX_DDR_LEN. The value of TX_DDR_LEN is equal to the allocated DDR area in every loop, except the last loop, which is set by the remaining transfer length.
   iii) Set AXITOE_CTRL[0] to 1b to initiate the data transfer from DDR to TOE1G-IP using the AXIIF module. Afterward, the CPU proceeds to step 7) if this is the last run loop. Otherwise, it returns to step i) to prepare the next loop transfer.
7) The CPU waits until TOE1G-IP completes the transmit operation, indicated by the busy flag (TOE_CMD_REG[0]) being de-asserted.
8) The CPU sets TOE_CMD_REG to initiate the close connection operation.
9) The CPU waits until the connection status becomes OFF (CONNON_REG=0), and then the operation is completed.

3.3 Ping Command Test

The Ping command is used to demonstrate data transfer via the slow connection. The CPU firmware implements a simple method to decode and encode the Ethernet packet following the ICMP protocol. The IP Datagram of the ICMP protocol utilized to support the Ping command is illustrated in Figure 3-2.



Bits 0-7	Bits 8-15	Bits 16-23	Bits 24-31
Version/IHL	Type of service	Length	
TTL	Protocol	Checksum	
Source IP address			
Destination IP address			
Type of message	Code	Header Checksum	
Identifier		Sequence Number	
Data			

Figure 3-2 IP Datagram for ICMP

In response to receiving an Echo request packet (Type of message=8) from the target device (PC), the CPU firmware implements a function to generate an Echo reply packet (Type of message=0). The Echo reply packet replicates all data within the Echo request packet. Further information about Pin command can be found from the following site.
http://en.wikipedia.org/wiki/Ping_(networking_utility)

The firmware steps for processing Ping command are outlined as follows.

1) The CPU waits until the read value of RXADDR_REG is updated, indicating the reception of a new packet.
2) Copy the received packet from RxRAM to a temporal buffer for further processing. The firmware allocates a 256-byte area as the temporal buffer for the receive operation, defined by “TMPBUF_SIZE” parameter.
3) Validate the IP checksum and ICMP checksum of the received packet. If the checksum is incorrect, an error message is displayed and the packet processing is terminated.
4) Create the Echo reply packet in the temporal buffer for the transmit operation. The network parameters, most header data, and the ICMP data of the created packet are loaded from the values inside the received packet.
5) Read the value of the created packet to calculate the IP checksum and ICMP checksum. Fill the result to the temporal buffer for transmit operation.
6) Copy the created packet from the temporal buffer for transmit operation to TxRAM.
7) The CPU sets the start address of TxRAM and the transfer size to TXADDR_REG and TXLEN_REG registers. The TxEMAC initiates data transfer after TXLEN_REG is set. As the processing time from step 1) to 5) is expected to be longer than the processing time to transmit each packet by TxEMAC, the CPU firmware ignores to wait for the completion of the TxEMAC operation. Instead, it returns to step 1) for the next packet processing. The user can exit the Ping command test menu by entering any key into the console.

4 Necessary consideration

For simple test menu, the fast connection and slow connection tests are not designed to operate simultaneously. However, user can modify the firmware to enable the operation of fast and slow connections concurrently if needed. An alternative example using two-port demo hardware, known as the FTP Server demo, is available on our website to demonstrate how fast and slow connections can be operated simultaneously.

5 Revision History

Revision	Date	Description
1.4	26-Jul-23	Add the details of the hardware structure and CPU firmware
1.3	2-Sep-16	IP core product renamed from TOE2-IP to TOE1G-IP
1.2	29-Dec-15	Add asynchronous clock support
1.1	28-Oct-15	Correct Table1 description
1.0	9-Jan-15	Initial version release