

TOE10G-IP reference design by Intel PAC A10GX

Rev1.0 3-Apr-19

1 Intel PAC Overview



Figure 1-1 Intel PAC with Intel Arria10GX FPGA

As described in UG-20166 (Intel Acceleration Quick Start Guide for Intel PAC with A10 GX), the Intel PAC (Intel Programmable Acceleration Card) provides the acceleration platform to free the Intel Xeon processor by offloading computationally intensive tasks. So, Xeon processor is free for running other critical processing tasks. Intel PAC connects to the Intel Xeon processor through the PCIe interface on the motherboard.

UG-20166 could be downloaded by following link.

<https://www.intel.com/content/www/us/en/programmable/documentation/iyu1522005567196.htm>

!

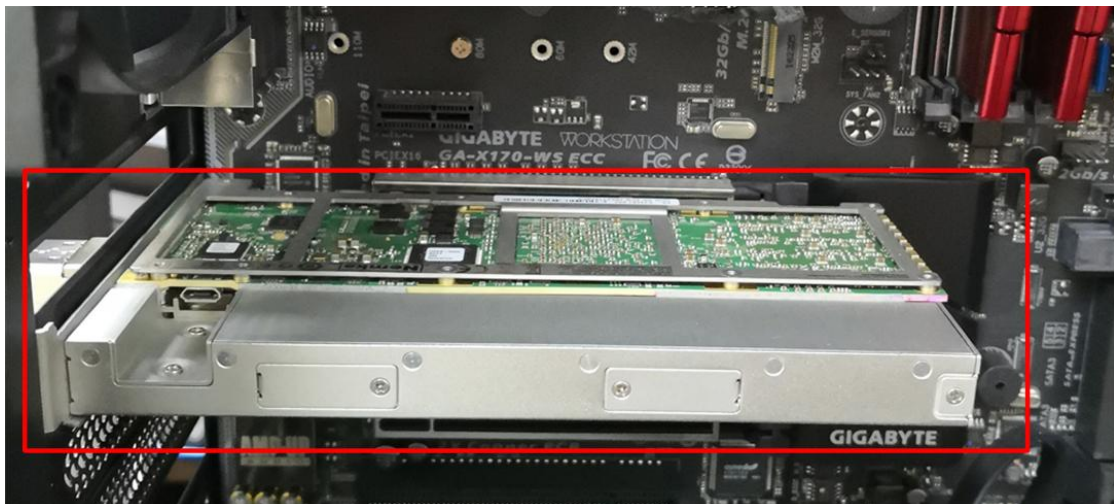


Figure 1-2 Intel PAC installation on Motherboard

Intel PAC is a collection of software, firmware, and tools that allows both software and RTL developers to take advantage of the power of Intel FPGAs. The overview of Intel PAC platform hardware and software is shown in Figure 1-3.

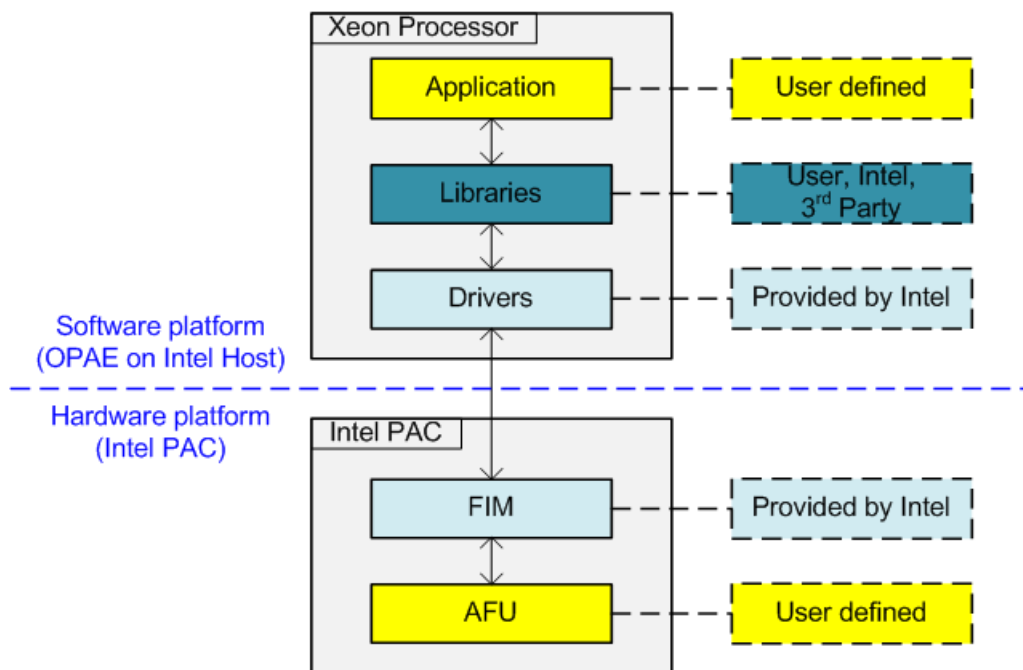


Figure 1-3 Intel PAC hardware and software overview

The hardware on Intel PAC platform has two parts, i.e. static region which is called FIM (FPGA Interface Manager) and partial reconfiguration region which is called AFU (Accelerator Functional Unit).

FIM owns all hard IPs on FPGA such as PLLs, PCIe IP core, DDR memory interfaces, high speed serial interface, and partial reconfiguration (PR) engine to load AFUs. After power up, FPGA configures the FIM only. Next, the software programs AFU images. AFU is FPGA logic designed by user to be CPU offload engine or hardware accelerator. User can design multiple AFUs to swap in and out of PR region. AFU could be designed by RTL or Open CL.

OPAE (Open Programmable Acceleration Engine) software running on the Intel Xeon processor handles all the details of the reconfiguration process. Otherwise, the OPAE provides libraries, drivers, and sample programs useful for AFU development.

Typically, when user implements some offload engines, it needs to design the logic on AFU and develop the software running on Xeon processor. Address mapping for control register is also implemented in AFU. The details of the interface between Xeon processor and FIM are shown in Figure 1-4.

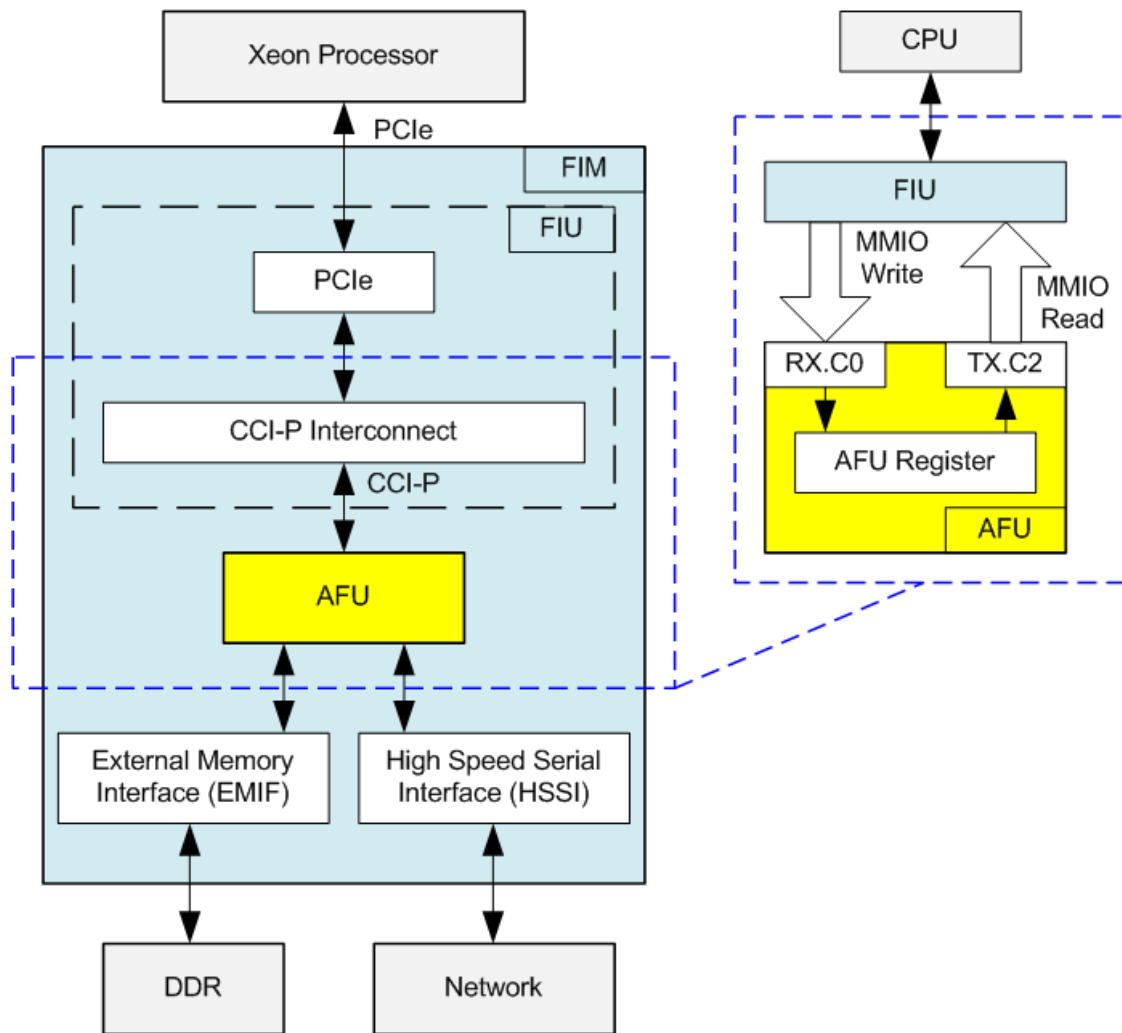


Figure 1-4 FPGA Interface Manager

The FIM consists of FIU (FPGA Interface Unit), EMIF, and HSSI. The AFU communicates with Intel Xeon processor by using CCI-P (Core Cache Interface) standard interface. CCI-P is the host interface which defines the CCI-P protocol and signaling interface and can be implemented on platform interfaces like PCIe.

In this reference design, there is no main memory write or read request from AFU. So, Tx.c0 and Tx.c1 for request main memory are not used. CCI-P is applied to generate MMIO read request and MMIO write requests for accessing the AFU register from the CPU. MMIO Write and Read request are received over Rx.c0 channel. The response to return data uses Tx.C2 channel for MMIO read request. CCI-P drives data of MMIO write request over Rx.c0 channel. More details of CCI-P could be downloaded by following link.

<https://www.intel.com/content/www/us/en/programmable/documentation/buf1506187769663.htm>
!

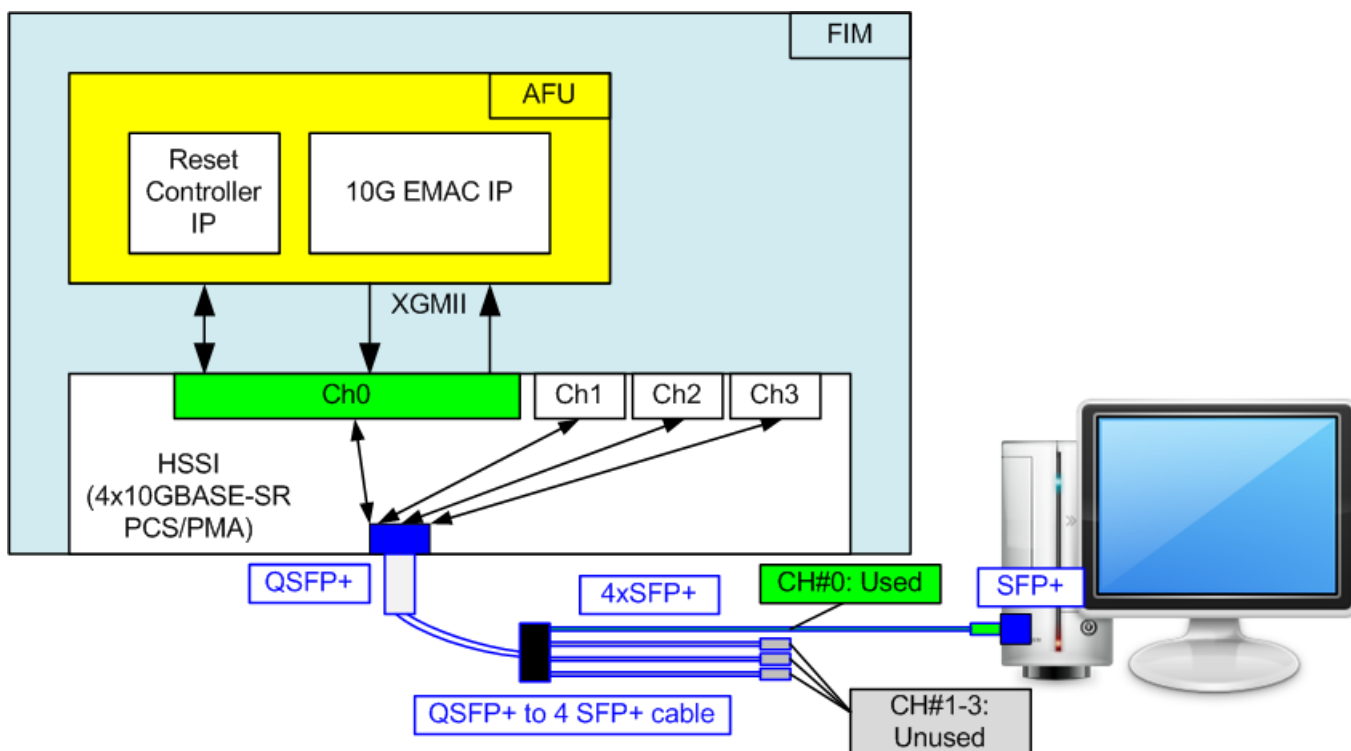


Figure 1-5 High Speed Serial Interface

As described in UG-20188, the Intel PAC with Arria 10 GX features a QSFP+ network port that can be configured for either 4x10GBASE-SR or 40GBASE-SR4 operation. In this reference design, HSSI is configured as 4x10GBASE-SR PCS/PMA to transfer 10G Ethernet packet with external network device such as PC as shown in Figure 1-5.

Though four channels are available in HSSI, only channel#0 is connected to AFU for transferring 10 Gb Ethernet data. QSFP+ to 4 SFP+ cable is applied to connect between Intel PAC and external network device to use only channel#0. To connect with 10GBASE-SR PCS/PMA module, two IPs are applied in AFU, i.e. 10G EMAC IP and Reset controller IP.

More details of HSSI could be downloaded by following link.

<https://www.intel.com/content/www/us/en/programmable/documentation/vqj1528674861813.htm>

!

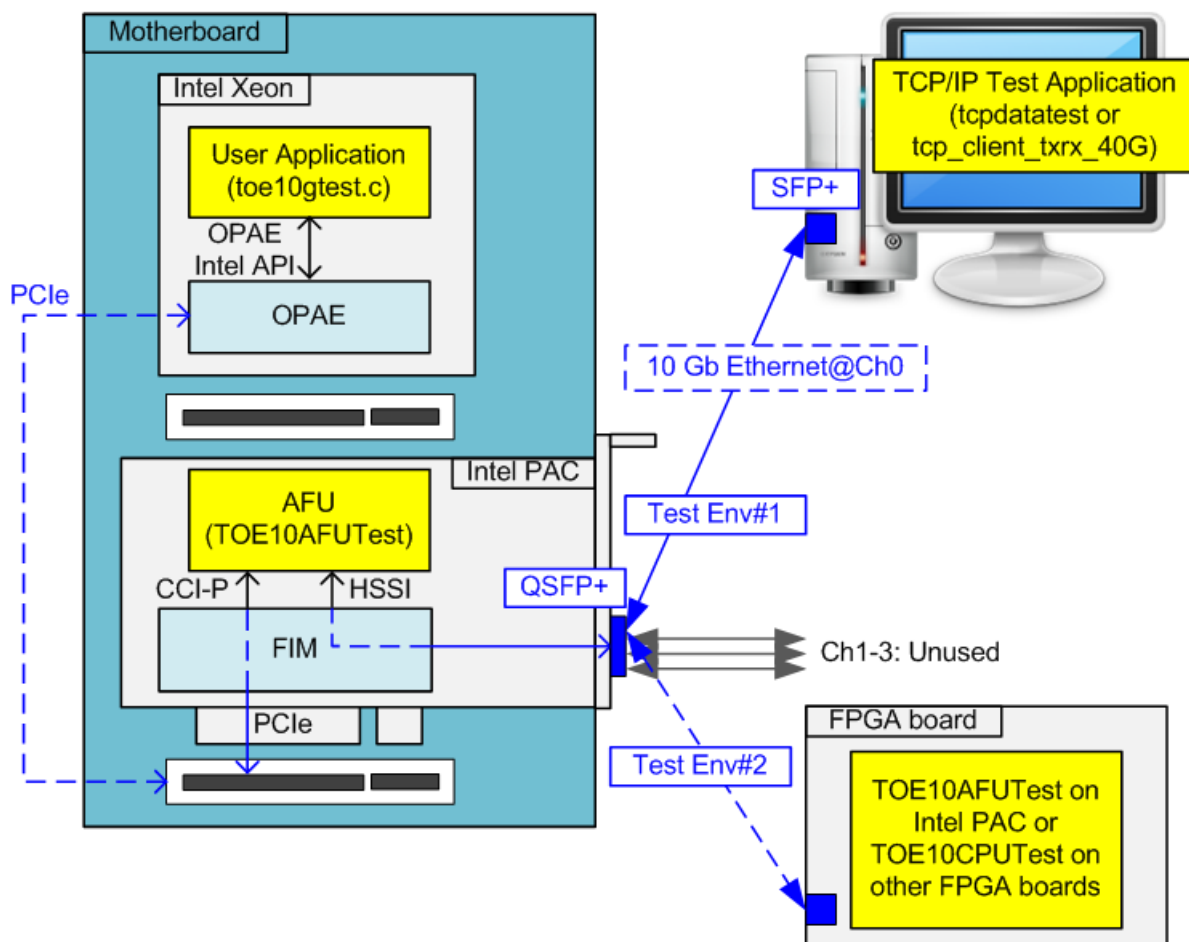


Figure 1-6 TOE10G-IP integrated on the Intel PAC platform

To build 10G Ethernet demo by using TOE10G-IP integrated on Intel PAC, the hardware of AFU and the software application on Intel Xeon must be designed as shown in Figure 1-6. TOE10AFUTest is the AFU logic which includes CCI-P interface for register access by CPU and HSSI interface for 10G Ethernet I/O. The software application to set control register and parameter running on Intel Xeon is toe10gtest.

The destination network device to transfer 10 Gb Ethernet packet by using TCP/IP protocol may be the host PC or the FPGA board including 10 Gb Ethernet connection. For the host PC, TCP/IP test application must be run to send and receive TCP/IP packet with Intel PAC. There are two TCP/IP test applications provided by Design Gateway, i.e. tcpdatatest (half-duplex test) and tcp_client_trrx_40G (full duplex test). For the FPGA board, user can use another Intel PAC installed on Xeon Processor Motherboard or the other FPGA boards configured by TOE10G-IP with CPU reference design. When running with FPGA, the best performance for transferring TCP/IP data could be achieved because the latency is lower. The details to run TOE10G-IP with CPU reference design on the other FPGA boards could be downloaded from following link.
https://dgway.com/products/IP/TOE10G-IP/dg_toe10gip_cpu_refdesign_intel.pdf
https://dgway.com/products/IP/TOE10G-IP/dg_toe10gip_cpu_instruction_intel.pdf

2 AFU Hardware overview

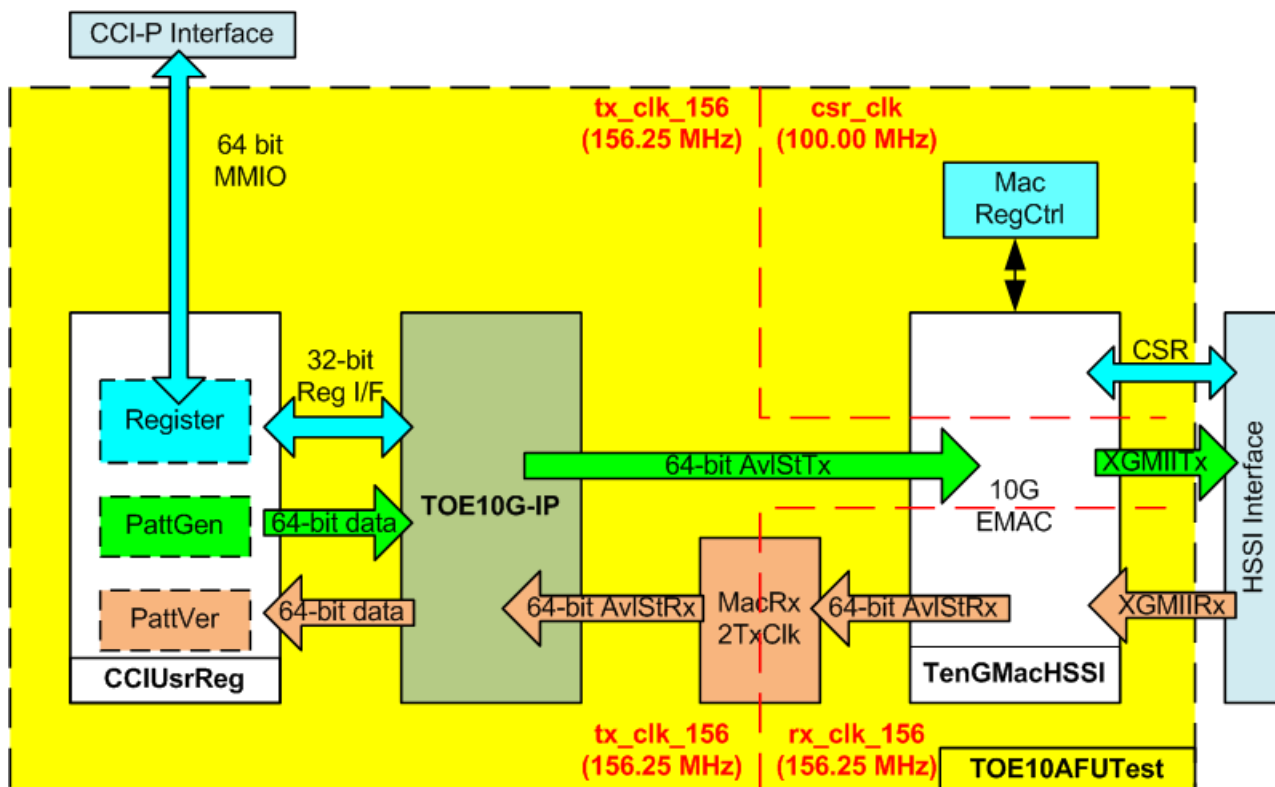


Figure 2-1 AFU block diagram

Figure 2-1 shows the logic details inside TOE10AFUTest module. AFU includes TOE10G-IP for transferring TCP/IP packet. The lower layer (link layer) is implemented by Ethernet MAC IP from Intel FPGA (TenGMACHSSI). User interface of TOE10G-IP connects to CCIUsrReg module for both data interface and register interface. Register files of CCIUsrReg are split into three regions, i.e. AFU CSR region, TOE10G-IP region, and internal test logic region (PattGen and PattVer). Register files of CCIUsrReg are controlled by the software running on Xeon processor (through 64-bit MMIO interface). User can control the operation of TOE10G-IP, PattGen, and PattVer by modifying the software on Xeon processor.

Otherwise, TOE10AFUTest module consists of two small modules, i.e. MacRx2TxClk and MacRegCtrl. MacRegCtrl is designed to setup Ethernet MAC register such as disable and enable flag to send and receive data. This module is run on csr_clk clock domain to synchronous to CSR interface of Ethernet MAC. Tx data stream and Rx data stream of Ethernet MAC run in different clock domain, so MacRx2TxClk is designed to convert Rx data stream of Ethernet MAC clock domain to run on tx_clk_156 instead of rx_clk_156. To convert clock domain, asynchronous buffer and asynchronous register must be designed in MacRx2TxClk. More details of each module inside the AFU are described as follows.

2.1 TenGMachSSI

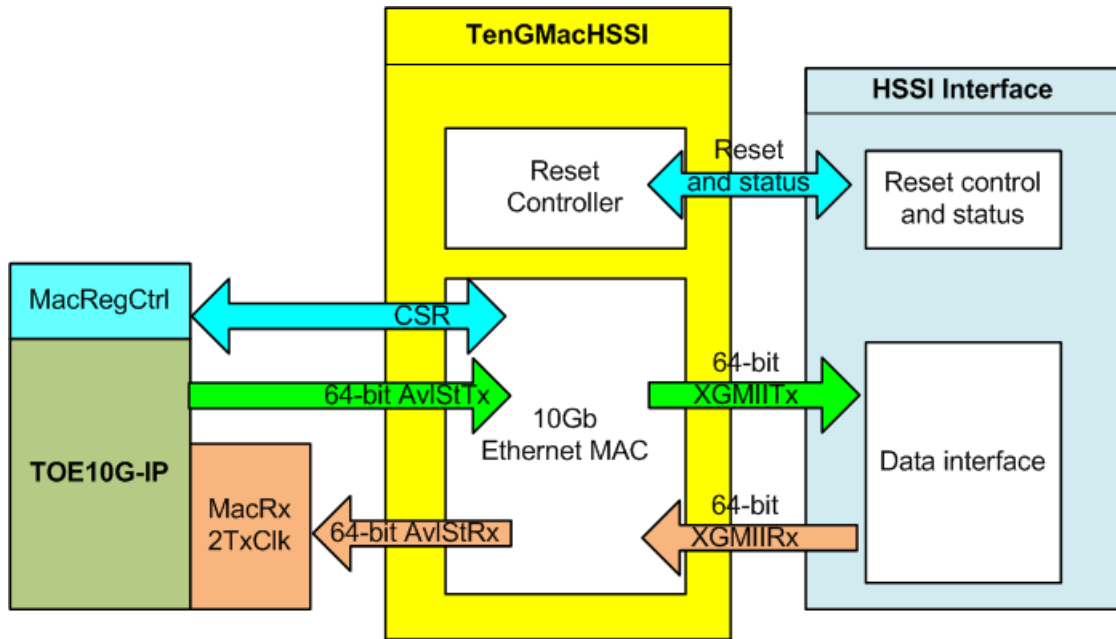


Figure 2-2 TenGMachSSI

TenGMachSSI consists of two submodules, i.e. Reset controller and 10G Ethernet MAC. Both modules are IP core provided by IP catalog of QuartusII. There are three clock domains running inside TenGMachSSI.

The first clock is CSR clock which is received from HSSI interface. CSR clock is stable clock to use in the logic for initialization process while the other clocks may not be stable. The blocks running CSR clock are Reset controller and MacRegCtrl. Reset controller controls reset and monitors power up sequence of HSSI interface.

The second clock is TxClk which is clock output from HSSI to synchronous to XGMII Tx interface. This clock is main clock domain for AFU logic including TOE10G-IP. Clock domain of CCI-P interface is also fed by TxClk.

The last clock is RxClk, output from HSSI to synchronous to XGMII Rx interface. This clock is fed to Rx data path of 10G Ethernet MAC and MacRx2TxClk module.

More details of Low latency 10G Ethernet MAC are described in following link.
<https://www.intel.com/content/www/us/en/programmable/documentation/bhc1395127830032.html>

2.2 MACRegCtrl

This module is designed to configure parameter of 10G Ethernet MAC and monitors the status through Avalon-MM bus. The logic is simply designed by using state machine and runs only one time after system power up to initialize Ethernet MAC.

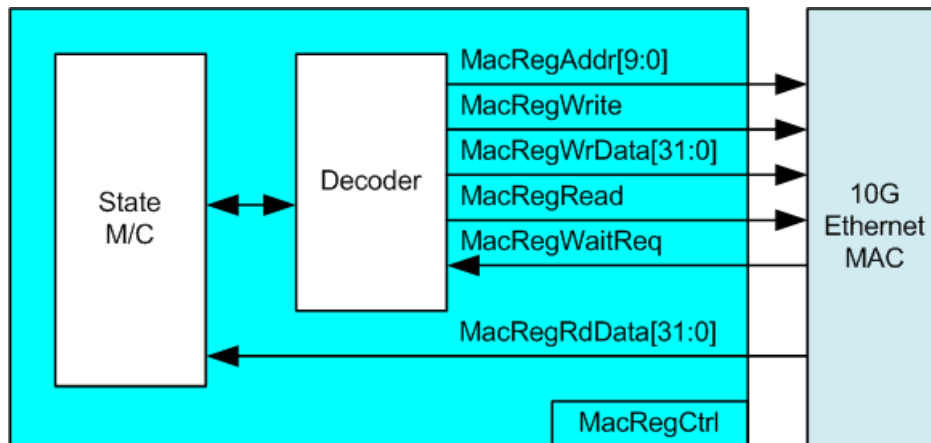


Figure 2-3 MACRegCtrl block diagram

The initialization sequence running on MACRegCtrl is as follows.

- 1) Disable transmit and receive path of EMAC
- 2) Wait until transmit and receive path are idle
- 3) Set receive module to remove CRC and padding
- 4) Disable pause frame transmission
- 5) Enable transmit and receive path of EMAC

2.3 TOE10G-IP

TOE10G-IP implements TCP/IP stack and offload engine. User interface consists of control signals and data signals. Control and status signals are accessed through register interface. Data signals are accessed through FIFO interface. More details are described in datasheet. https://dgway.com/products/IP/TOE10G-IP/dg_toe10gip_data_sheet_altera_en.pdf

2.4 MacRx2TxClk

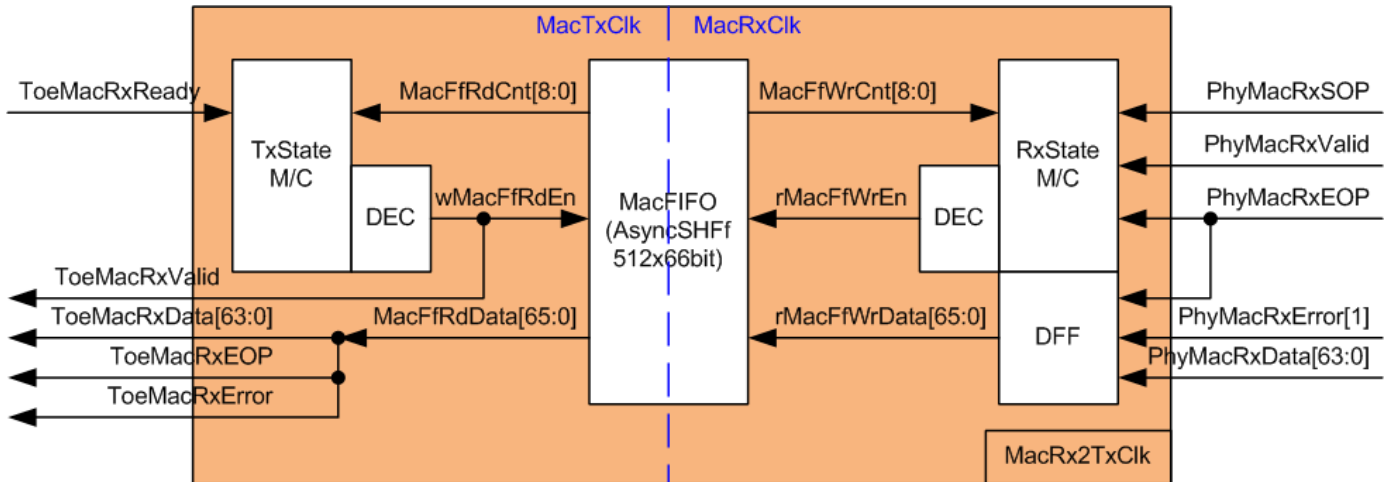


Figure 2-4 MacRx2TxClk module

MacRx2TxClk is designed to convert clock domain of data stream received from 10G EMAC which is synchronous to MacRxClk (rx_clk_156) to MacTxClk (tx_clk_156). Asynchronous FIFO (MacFIFO) must be included to be data buffer. MacFIFO is implemented by using BlockMemory and the size is 512 x 66-bit. FIFO type is show-ahead style (read data is returned at the same clock as read enable='1'). 66 bit data is applied to store 64 bit data, EOP flag, and Error flag. MacRx2TxClk is designed based on following assumption.

- (1) Data stream of one packet (received from 10G EMAC) must be valid continuously during transferring the packet.
- (2) ToeMacRxReady must be asserted to '1' during transferring one packet from MacFIFO. ToeMacRxReady could be deasserted to '0' to pause data transmission after finishing the packet (ToeMacRxEOP='1').
- (3) Clock frequency of MacTxClk and MacRxClk must not be different more than 0.1%. If clock frequency is too much different, Mac FIFO will be overflow or underflow during transferring packet from 10G EMAC to TOE10G-IP.

The logic inside MacRx2TxClk is split into two groups, i.e. the logic for transferring data from 10G EMAC to MAC FIFO which is run on MacRxClk (the right side of Figure 2-4) and the logic for transferring data from MAC FIFO to TOE10G-IP which is run on MacTxClk (the left side of Figure 2-4). More details of the logic in each group are described as follow.

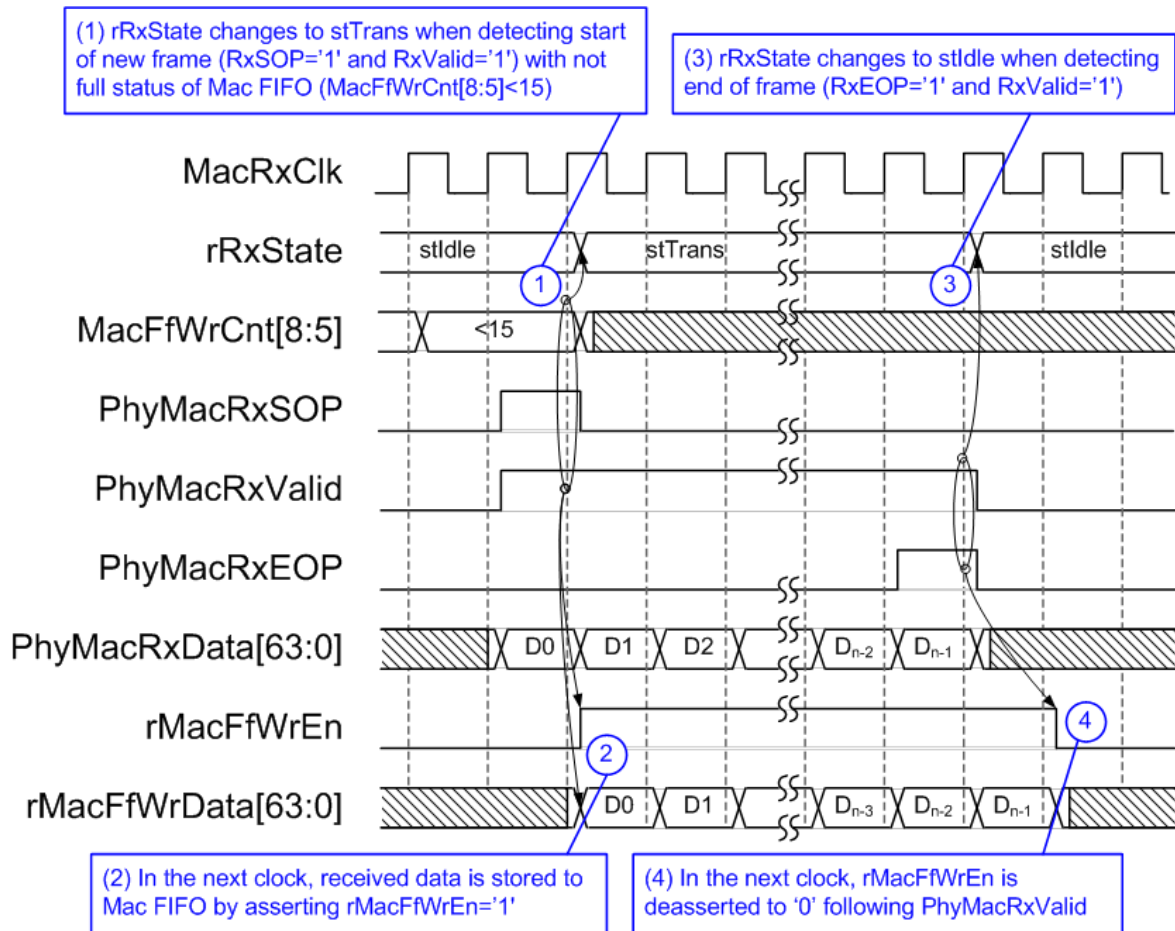


Figure 2-5 Timing diagram to write Mac FIFO

Rx state machine decides to accept or reject the received packet from 10G EMAC. The new packet is accepted when start of frame is found (PhyMacRxSOP='1' and PhyMacRxValid='1') with FIFO not full status (MacFfWrCnt[8:5] is less than 15). After new packet acceptance, state machine changes to stTrans in the next clock. The packet is stored to Mac FIFO by asserting rMacFfWrEn='1' with the valid value of rMacFfWrData. rMacFfWrData is designed by adding one D-FF to PhyMacRxData. During running in stTrans, MacFfWrCnt is not monitored. rMacFfWrEn is always asserted to '1' and rMacFfWrData is bypassed from PhyMacRxData.

When end of packet is detected (PhyMacRxEOP='1' and PhyMacRxValid='1'), Rx state machine changes to stIdle. The last data, EOP flag, and error flag are stored to Mac FIFO as the last data. After that, rMacFfWrEn is deasserted to '0' to complete packet transferring.

If the new packet is received with FIFO full status, Rx state machine will reject the packet by changing to stFlush instead of stTrans. In stFlush, the packet is received but rMacFfWrEn is not asserted to '1'. So, Mac FIFO is not written. After end of frame is found, state machine returns to stIdle for waiting the new packet.

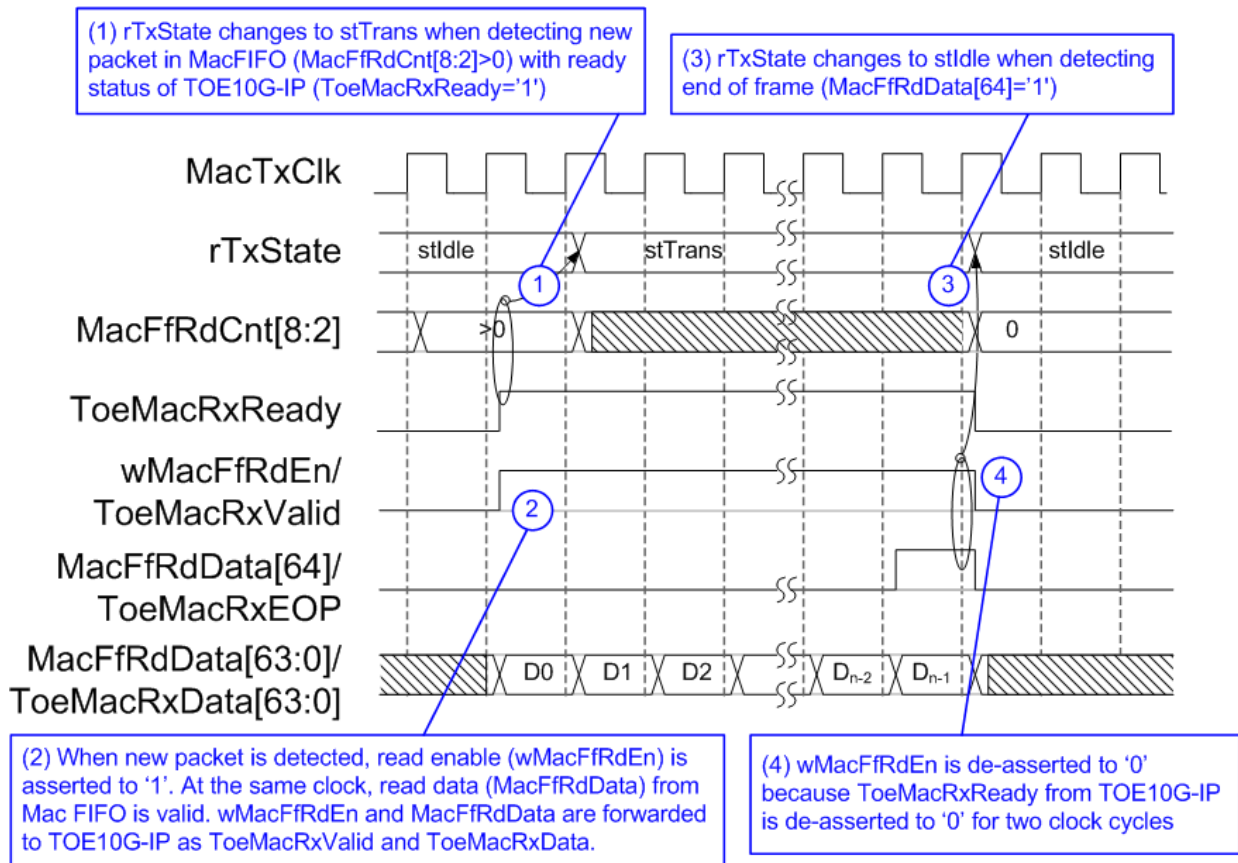


Figure 2-6 Timing diagram to read Mac FIFO

Tx state machine starts to read new data from Mac FIFO when new packet is found with ready status of TOE10G-IP. The new packet is detected by monitoring MacFfRdCnt[8:2] to confirm that at least 4 data are ready in Mac FIFO. 4 data is data buffering to transfer packet to TOE10G-IP continuously when MacTxClk frequency is little more than MacRxClk frequency. Also, the smallest packet size is more than 32 bytes (4 data). If ToeMacRxReady is not asserted to '1', new packet will not be fed from Mac FIFO.

Mac FIFO is show-ahead type, so read data (MacFfRdData) is valid at the same clock as read enable asserted to '1' (MacFfRdEn). MacFfRdEn is applied to be data output valid for TOE10G-IP (ToeMacRxValid). Also, MacFfRdData is applied to be data output for TOE10G-IP (ToeMacRxData).

In stTrans, MacFfRdEn is always asserted to '1' to transfer packet from Mac FIFO to TOE10G-IP continuousl until end of packet. When end of frame flag is detected (MacFfRdData[64]='1'), Tx state machine changes to stIdle. After t hat, ToeMacRxReady output from TOE10G-IP is always de-asserted to '0' for two clock cycles to pause data transmission. During pausing transmission, MacFfRdEn is also de-asserted to '0' to wait TOE10G-IP ready.

2.5 CCI2Reg

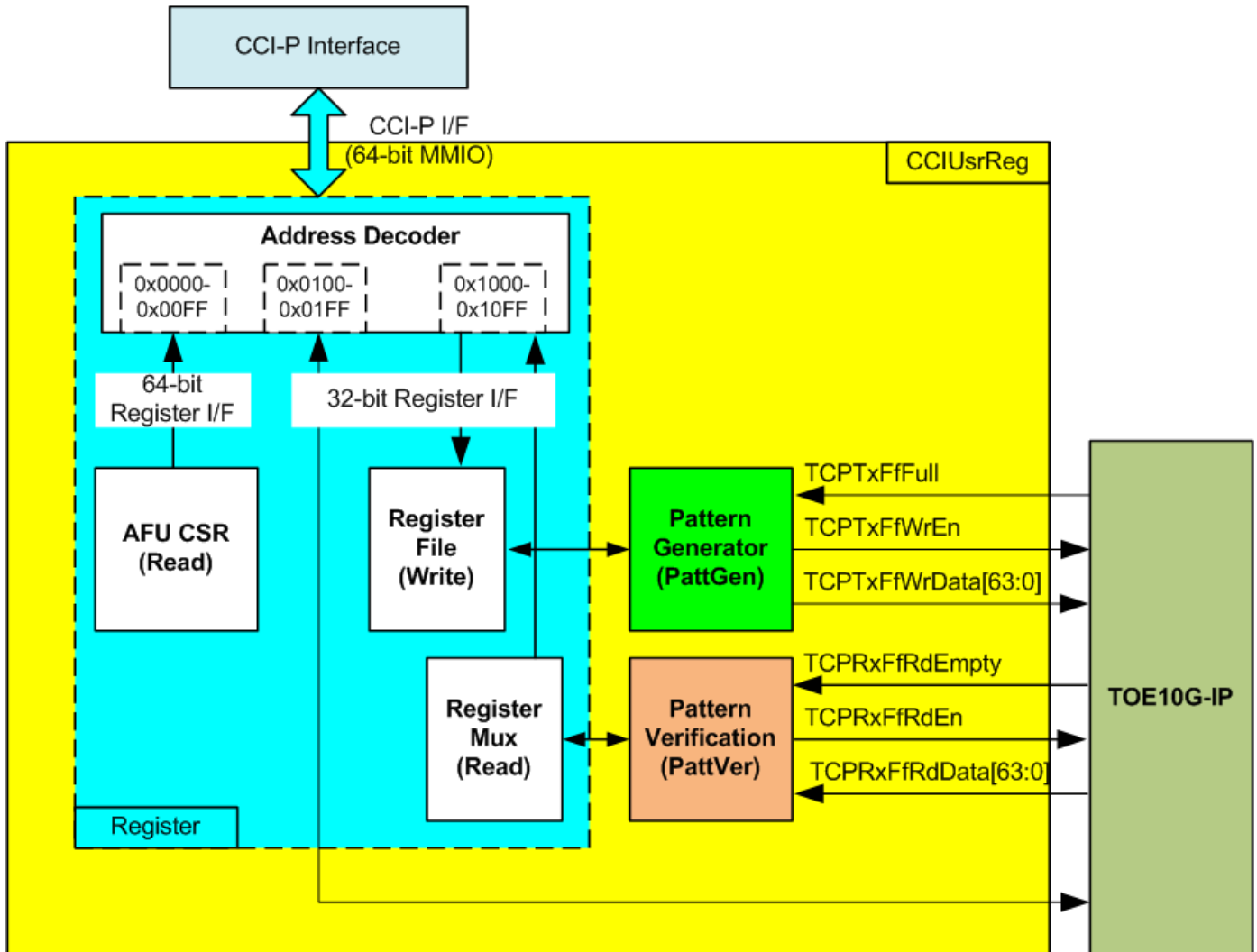


Figure 2-7 CCIUserReg block diagram

The logic inside CCIUserReg could be split into three parts, i.e. Register, Pattern generator (PattGen), and Pattern verification (PattVer). Register block converts MMIO which is 64-bit interface to be 32-bit interface for internal usage and TOE10G-IP register interface. Pattern generator block is designed to send 64-bit test data to TOE10G-IP following FIFO interface standard. Pattern verification block is designed to read and verify 64-bit data from TOE10G-IP following FIFO interface standard. More details of each block are described as follows.

2.5.1 Register

As shown in Figure 2-7, register map of CCI-P interface is split into three areas, i.e. mandatory AFU CSR (0x0000-0x00FF), TOE10G-IP (0x0100-0x01FF), and internal signals (0x1000-0x10FF).

Address signal of CCI-P for MMIO access is based on 32-bit register. But data bus size of MMIO for write and read register is 64-bit register. The address passing from MMIO to TOE10G-IP, PattGen, and PattVer is designed to align 64-bit unit. Since data bus size of TOE10G-IP registers and internal registers for controlling PattGen and PattVer is 32-bit, only lower 32 bits of MMIO write data is applied. The upper 32 bits of MMIO write data is ignored.

AFU CSR is mandatory register which must be implemented in AFU. It is register set for read only. Constant value and AFU ID are returned to MMIO as 64-bit unit, as shown in Figure 2-8. The software uses the AFU_ID to ensure that the correct AFU is matched.

Name	DWORD Address Offset (CCI-P)	Byte Address Offset (Software)	Description
DEV_FEATURE_HDR (DFH)	0x0000	0x0000	—
AFU_ID_L	0x0002	0x0008	Lower 64 bits of the AFU_ID GUID. <ul style="list-style-type: none"> Attribute: Read Only [63:0]; Default: 0x0
AFU_ID_H	0x0004	0x0010	Upper 64 bits of the AFU_ID GUID. <ul style="list-style-type: none"> Attribute: Read Only [63:0]; Default: 0x0
DFH_RSVD0	0x0006	0x0018	[63:0] RSVD
DFH_RSVD1	0x0008	0x0020	[63:0] RSVD

Figure 2-8 Mandatory AFU CSRs (captured from MNL-1092 CCIP Reference manual)

Similar to write access, the read data returned from TOE10G-IP registers and internal registers of PattGen and PattVer is 32-bit unit. So, only lower 32 bits of read data is assigned when MMIO accesses TOE10G-IP area and internal register area.

Signal lists of CCI-P interface for MMIO access are shown in Table 2-1 and timing diagram of CCI-P interface for MMIO access is shown in Figure 2-9.

Table 2-1 CCI-P for MMIO access mapping to CCIUsrReg

CCIUserReg signal name	Direction	CCI-P interface signal name
RegAddress[15:0]	Input	t_if_ccip_Rx.c0.hdr.address[15:0]
RegWrData[31:0]	Input	t_if_ccip_Rx.c0.data[31:0]
RegWrEn	Input	t_if_ccip_Rx.c0.mmioWrValid
RegRdReq	Input	t_if_ccip_Rx.c0.mmioRdValid
RegRdValid	output	t_if_ccip_Tx.c2.mmioRdValid
RegRdData[63:0]	output	t_if_ccip_Tx.c2.data[63:0]

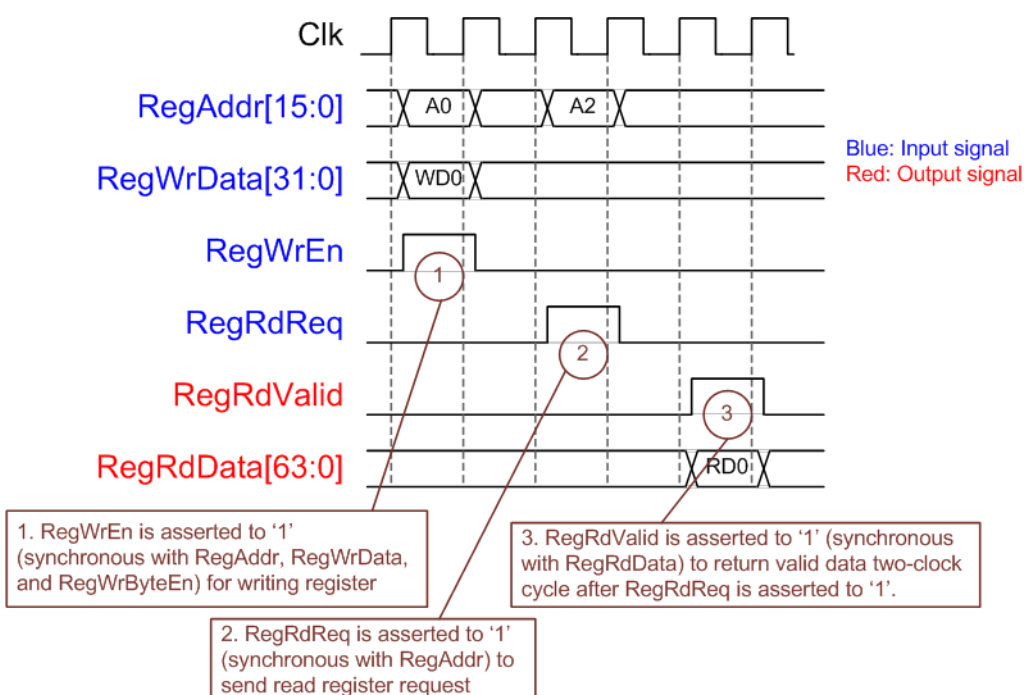


Figure 2-9 CCI-P interface for MMIO access timing diagram

To write register, timing diagram of MMIO access is same as RAM interface. RegAddr and RegWrData must be valid when RegWrEn is asserted to '1'. The upper bit of RegAddr is decoded by Address decoder to assert write enable for TOE10G-IP or internal signals of CCIUsrReg. The lower 32-bit data is forwarded to 32-bit write data for TOE10G-IP and internal signals.

To read register, CCI-P interface asserts RegRdReq='1' with the valid value on RegAddr. The upper bit of RegAddr is decoded to select read data source which could be fed from AFU CSR, TOE10G-IP, or internal signals. Since there are many registers which are mapped to read access, two pipeline registers are designed in read data path. So, RegRdValid is created by adding two Flip-Flops to RegRdReq. The details of logic design in Register block is shown in Figure 2-10.

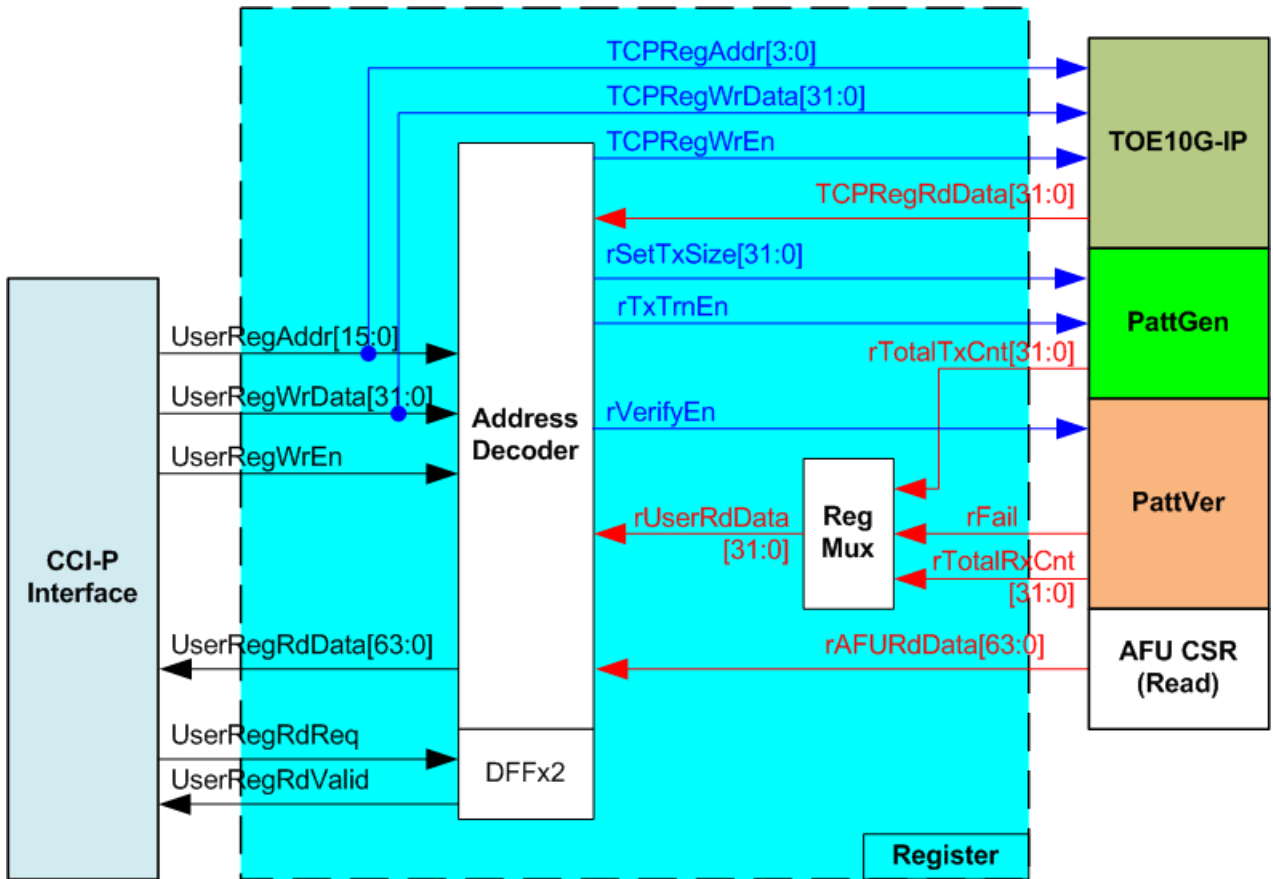


Figure 2-10 Register block

Data bus size of CCI-P is 64 bit, but the address is based on 32 bit. The address and data interface of TOE10G-IP is based on 32 bit. For simple logic design to convert CCI-P register signals to be TOE10G-IP register signals, CCI-P register address map for TOE10G-IP area is aligned to 64 bit or 8 byte. TCPRegAddr[3:0] is fed by UserRegAddr[4:1]. TCPRegWrEn is asserted to '1' when UserRegWrEn='1' and upper bit of UserRegAddr is equal to TOE10G-IP area.

Similar to TOE10G-IP, internal signals for PattGen and PattVer are based on 64 bit address. PattGen parameters, programmed by the software, are rSetTxSize (total transfer size) and rTxTrnEn (enable signal to start PattGen). For PattVer, the software sets rVerifyEn flag to enable or disable the data comparator inside PattVer.

PattGen and PattVer status signals such as rTotalTxCnt (current Tx data size), rTotalRxCnt (current Rx data size), and rFail (verification failed flag) are fed to multiplexer to return read value to CCI-P. Read data bus from TOE10G-IP, PattGen, and PattVer is 32 bit while AFU CSR data bus is 64 bit.

Table 2-2 shows CCIUserReg register map on CCI-P interface by using MMIO access.

Table 2-2 CCIUsrReg register map on MMIO access

Byte Address Wr/Rd	Register Name (Label in the "toe10gtest.c")	Description
BA+0x0000 – BA+0x00FF: Mandatory AFU CSRs Register Area (Refer to MNL-1092 for more details)		
0x0000	DEV_FEATURE_HDR	[63:0] Feature Header CSR
0x0008	AFU_ID_L	[63:0] Lower 64 bits of the AFU_ID GUID
0x0010	AFU_ID_H	[63:0] Upper 64 bits of the AFU_ID GUID
0x0018	DFH_RSVD0	[63:0] Reserved
0x0020	DFH_RSVD1	[63:0] Reserved
BA+0x0100 – BA+0x01FF: TOE10G-IP Register Area More details of each register are described in Table2 of TOE10G-IP datasheet.		
0x0100	TOE10_RST_REG	[31:0] Mapped to RST register within TOE10G-IP
0x0108	TOE10_CMD_REG	[31:0] Mapped to CMD register within TOE10G-IP
0x0110	TOE10_SML_REG	[31:0] Mapped to SML register within TOE10G-IP
0x0118	TOE10_SMH_REG	[31:0] Mapped to SMH register within TOE10G-IP
0x0120	TOE10_DIP_REG	[31:0] Mapped to DIP register within TOE10G-IP
0x0128	TOE10_SIP_REG	[31:0] Mapped to SIP register within TOE10G-IP
0x0130	TOE10_DPN_REG	[31:0] Mapped to DPN register within TOE10G-IP
0x0138	TOE10_SPN_REG	[31:0] Mapped to SPN register within TOE10G-IP
0x0140	TOE10_TDL_REG	[31:0] Mapped to TDL register within TOE10G-IP
0x0148	TOE10_TMO_REG	[31:0] Mapped to TMO register within TOE10G-IP
0x0150	TOE10_PKL_REG	[31:0] Mapped to PKL register within TOE10G-IP
0x0158	TOE10_PSH_REG	[31:0] Mapped to PSH register within TOE10G-IP
0x0160	TOE10_WIN_REG	[31:0] Mapped to WIN register within TOE10G-IP
0x0168	TOE10_ETL_REG	[31:0] Mapped to ETL register within TOE10G-IP
0x0170	TOE10_SRV_REG	[31:0] Mapped to SRV register within TOE10G-IP
0x0178	TOE10_VER_REG	[31:0] Mapped to VER register within TOE10G-IP
BA+0x1000 – BA+0x10FF: Internal signals of PattGen and PattVer		
0x1000 Wr/Rd	Total transmit length (USER_TXLEN_REG)	Wr [31:0] – Set total size of PattGen in Qword unit (64-bit). Valid from 1-0xFFFFFFFF. Rd [31:0] – Completed size of PattGen in Qword unit (64-bit). The value is cleared to 0 when USER_CMD_REG is written by user.
0x1008 Wr/Rd	User Command (USER_CMD_REG)	Wr [0] – Start PattGen. Set '1' to start PattGen operation. This bit is auto-cleared to '0' after end of total transfer. [1] – Enable data verification ('0': Enable data verification, '1': Disable data verification) Rd [0] – Busy flag of PattGen. ('0': Idle, '1': PattGen is busy) [1] – Data verification error ('0': Normal, '1': Error) This bit is auto-cleared when user starts new operation or reset. [2] – Mapped to ConnOn signal of TOE10G-IP
0x1010 Wr/Rd	User Reset (USER_RST_REG)	Wr [0] – Reset signal. Set '1' to reset the logic. This bit is auto-cleared to '0'. [8] – Set '1' to clear TimerInt latch value Rd [8] – Latch value of TimerInt output from IP ('0': Normal, '1': TimerInt='1' is detected). This flag can be cleared by system reset condition or setting USER_RST_REG[8]='1'. [16] – 10G EMAC Linkup ('0': Link is down, '1': Link is up)
0x1018 Rd	FIFO status (USER_FFSTS_REG)	Rd [2:0]: Mapped to TCPRxFfLastRdCnt[2:0] signal of TOE10G-IP [15:3]: Mapped to TCPRxFfRdCnt[12:0] signal of TOE10G-IP [24]: Mapped to TCPTxFfFull signal of TOE10G-IP
0x1020 Rd	Total Receive length (TRN_RXLEN_REG)	Rd [31:0] – Completed size of PattVer in Qword unit (64-bit). The value is cleared to 0 when USER_CMD_REG is written by user.

2.5.2 Pattern Generator

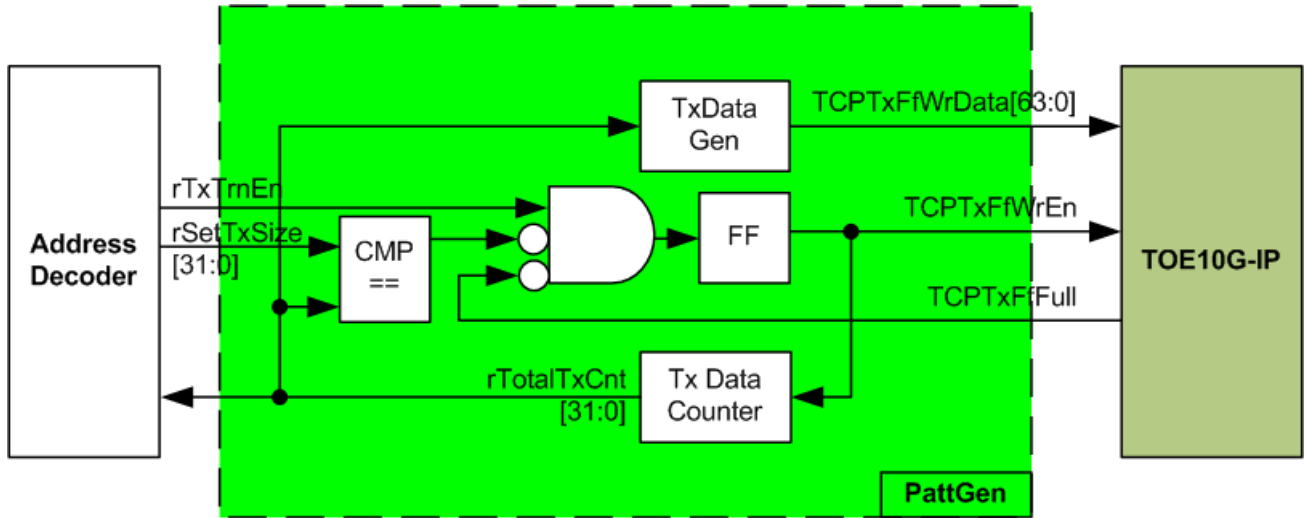


Figure 2-11 PattGen block

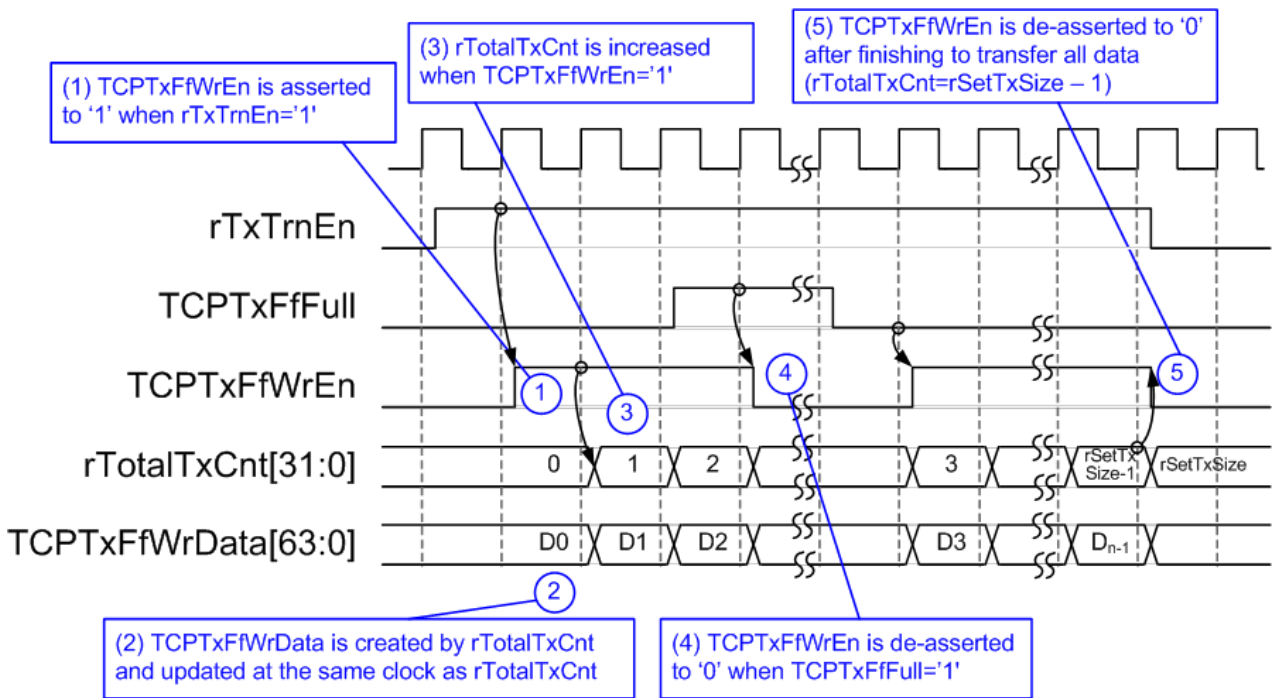


Figure 2-12 PattGen Timing diagram

PattGen is designed to generate test data to TOE10G-IP. rTxTrnEn is asserted to '1' when USER_CMD_REG[0] is set to '1'. When rTxTrnEn is '1', TCPTxFfWrEn is controlled by TCPTxFfFull. TCPTxFfWrEn is de-asserted to '0' when TCPTxFfFull is '1'. rTotalTxCnt is data counter to check total data sending to TOE10G-IP. rTotalTxCnt is also used to generate 32-bit increment data to TCPTxFfWrData signal. rTxTrnEn is de-asserted to '0' when finishing to transfer total data (total data is set by rSetTxSize).

2.5.3 Pattern Verification

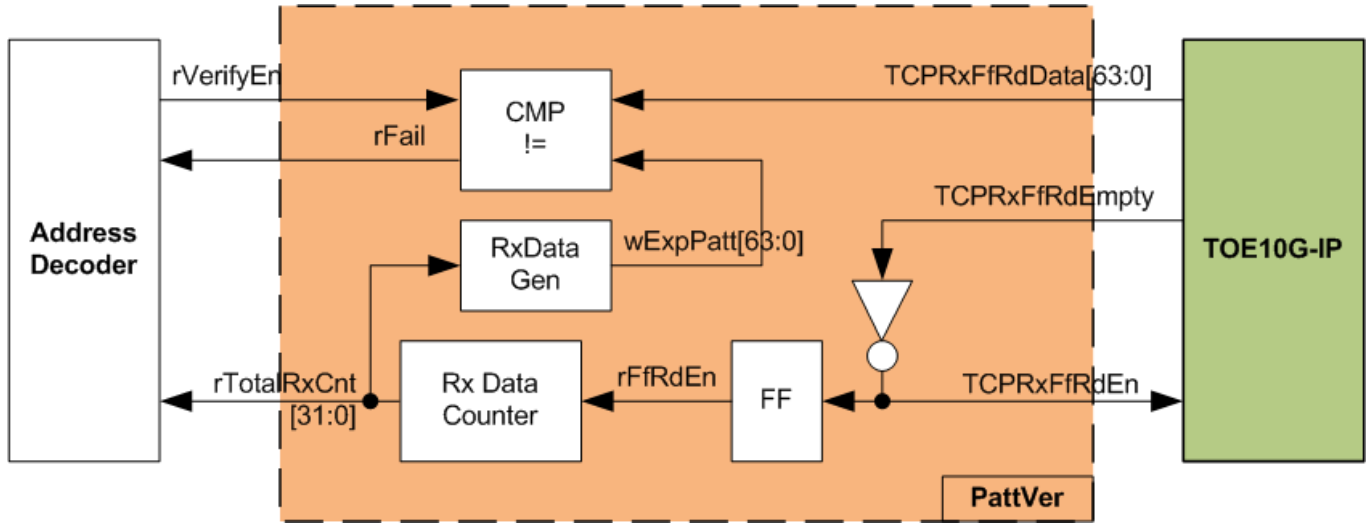


Figure 2-13 PattVer block

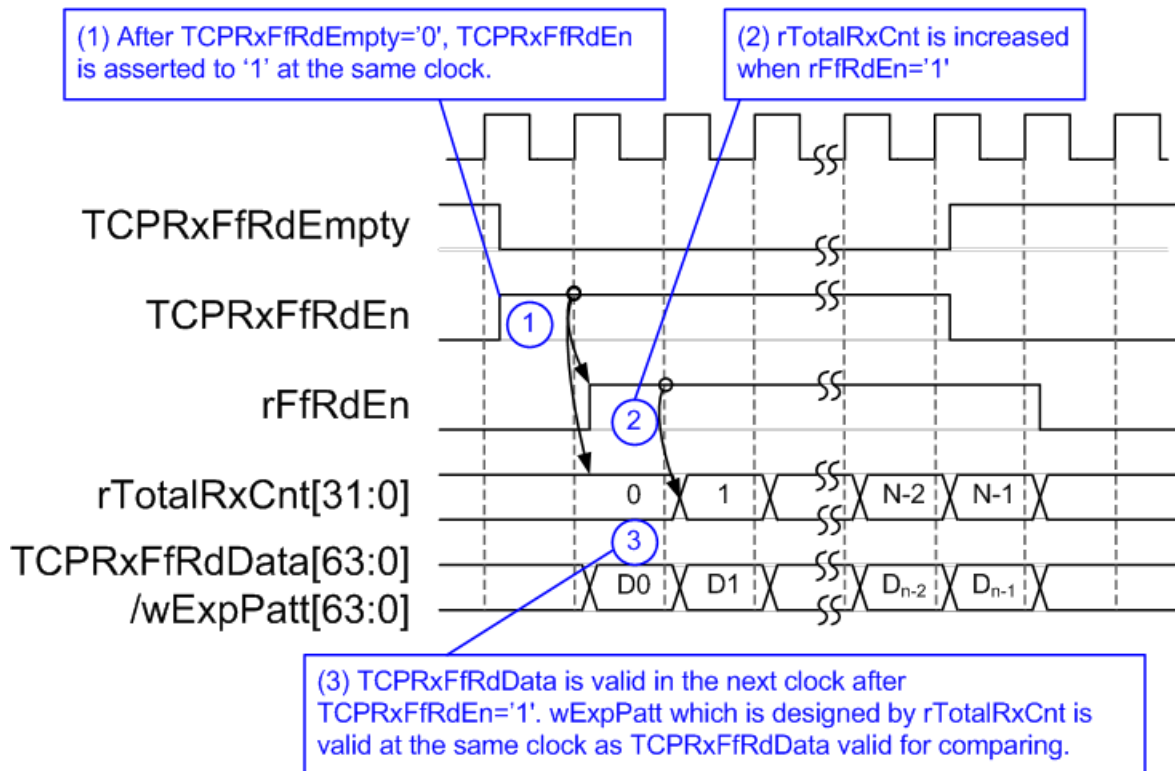


Figure 2-14 PattVer Timing diagram

PattVer is designed to read test data from TOE10G-IP with or without data verification, depending on rVerifyEn flag. When rVerifyEn is set to '1', data comparison is enabled to compare read data (TCPRxFfRdData) to the expected pattern (wExpPatt). If data verification is failed, rFail will be asserted to '1'. TCPRxFfRdEn is designed by using NOT logic of TCPRxFfRdEmpty. TCPRxFfRdData is valid for data comparison in the next clock. rFfRdEn which is one clock latency of TCPRxFfRdEn is applied to be counter enable of rTotalRxCnt to count total transfer size. rTotalRxCnt is used to generate wExpPatt.

3 User Application (Intel PAC)

3.1 Overview

After finishing AFU hardware design, configuration is built by OPAE SDK as shown in following diagram.

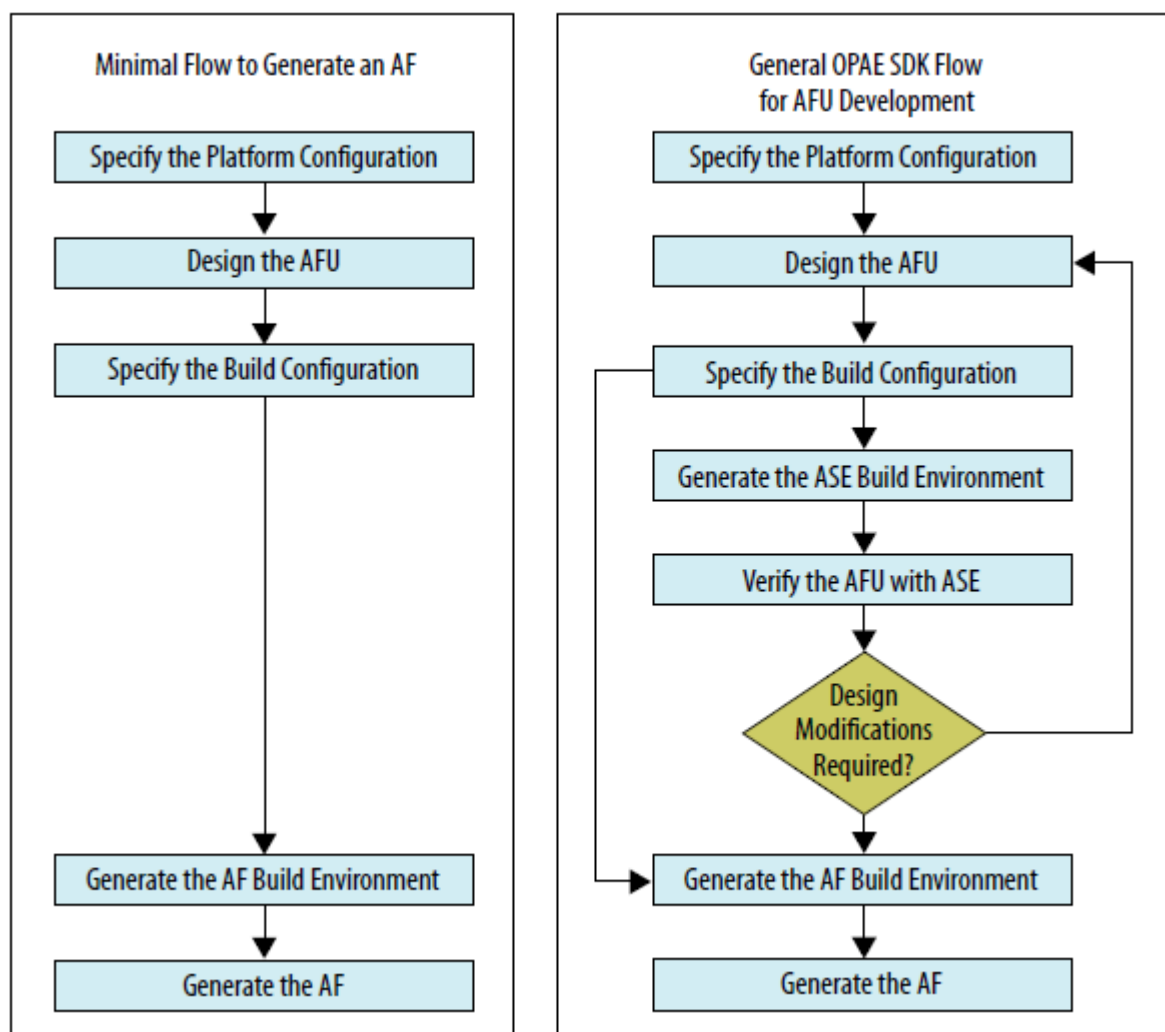


Figure 3-1 OPAE SDK Design Flow for AFU development (captured from UG-20169)

The output file after finishing hardware implementation is gbs file (green bit stream) which is bit stream for running partial configuration by the OPAE software platform. More details of developing AFUs with the OPAE SDK are described in UG-20169.

<https://www.intel.com/content/www/us/en/programmable/documentation/bfr1522087299048.html>

On software platform, the OPAE C library (libopae-c) is provided as a lightweight user-space library. The OPAE C library builds on the driver stack and provides access to the FPGA resources as a set of features for software programs running on the host. These features include the logic preconfigured on the FPGA and functions to reconfigure the FPGA. More details of OPAE are described in following link.

<https://opae.github.io/1.3.0/index.html>

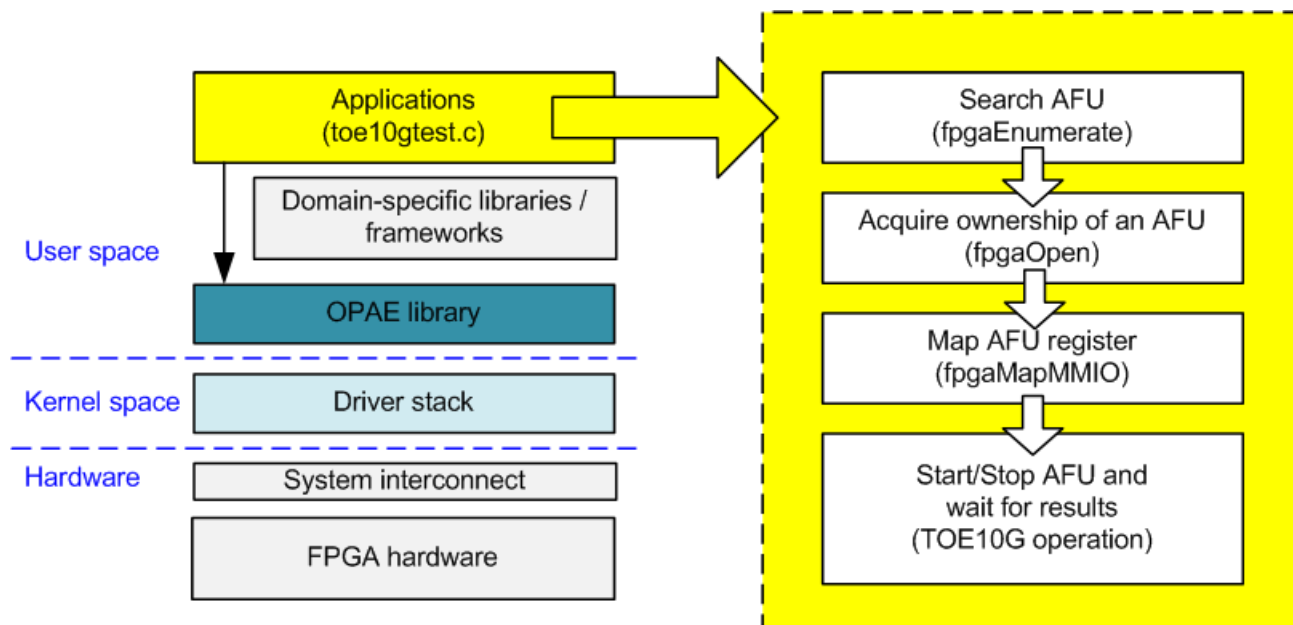


Figure 3-2 Software on Intel PAC PC

The basic application flow is shown in the right side of Figure 3-2. Before starting AFU, AFU initialization is run by calling following function.

- 1) Call fpgaEnumerate function to search AFU.
- 2) Call fpgaOpen function to acquire ownership of an AFU. A token is returned from fpgaEnumerate in the previous step.
- 3) Call fpgaMapMMIO to map the register file of AFU into the process's virtual memory space.

After that, start signal is sent to AFU for starting acceleration function. The 1st step of TOE10G-IP AFU is monitoring Ethernet linkup status. More details of TOE10G-IP software are described in the next topic.

3.2 TOE10G-IP software

The software sequence of TOE10G-IP running on Intel PAC is same as TOE10G-IP reference design on the other FPGA boards. The different point is the function to display message and receive parameter from user. After finishing MMIO mapping, 10G Ethernet link up status (USER_RST_REG[16]) is polling. The processor waits until link up is found. Next, welcome message is displayed and user selects operation mode of TOE10G-IP to be client or server mode.

To initialize as client mode, TOE10G-IP sends ARP request to get the MAC address from the destination device. For server mode, TOE10G-IP waits ARP request to decode MAC address and returns ARP reply to complete initialization process.

If test environment is setup by using two FPGA boards, the operation mode of TOE10G-IP must be different (one is client and another is server). To run with Test PC, it is recommended to set Intel PAC as client mode. When Test PC receives ARP request, Test PC always returns ARP reply. It is not simple to force PC sending ARP request to Intel PAC.

The software has two default parameters for each operation mode. Figure 3-3 shows the example of the initialization sequence after system boot-up.

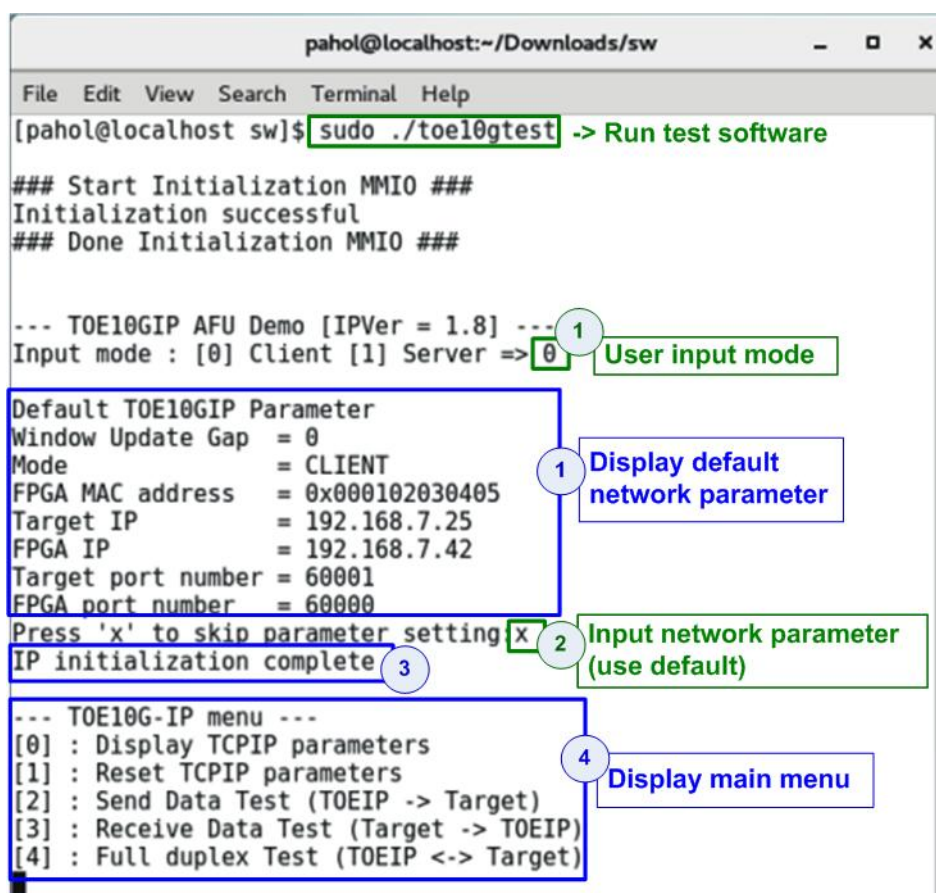


Figure 3-3 Example of initialization sequence in client mode on Linux terminal

There are four steps to complete initialization sequence as follows.

- 1) The software receives the operation mode from user and displays default parameters of selected mode on the Linux terminal.
- 2) User inputs 'x' to complete initialization sequence by using default parameters or inputs other keys to change some parameters. To change some parameters, please follow the step of Reset IP menu which is described in topic 3.2.2.
- 3) The Intel PAC host waits until TOE10G-IP finishing initialization sequence (monitoring TOE10_CMD_REG[0]='0').
- 4) Main menu is displayed. There are five test operations for user selection. More details of each menu are described as follows.

3.2.1 Show parameters

This menu is used to show current parameters of TOE10G-IP such as operation mode, source MAC address, destination IP address, source IP address, destination port, and source port. The sequence to display the parameters is as follows.

- 1) Read network parameter from each variable in the software.
- 2) Print out each variable.

3.2.2 Reset IP

This menu is used to change TOE10G-IP parameters such as IP address and source port number. After setting updated parameter to TOE10G-IP register, the Intel PAC host resets the IP to re-initialize by using new parameters. Finally, the Intel PAC host monitors busy flag to wait until the initialization is completed. The sequence to reset IP is as follows.

- 1) Display current parameter value to the Linux terminal.
- 2) Receive new input parameters from user and check input value whether valid or not. If the input is invalid, the old value will be used instead.
- 3) Force reset to IP by setting TOE10_RST_REG[0]='1'.
- 4) Set all parameters to TOE10G-IP register such as TOE10_SML_REG, TOE10_DIP_REG.
- 5) De-assert IP reset by setting TOE10_RESET_REG[0]='0'.
- 6) Clear PattGen and PattVer logic by sending reset to user logic (USER_RST_REG[0]='1').
- 7) Monitor IP busy flag (TOE10_CMD_REG[0]) until initialization sequence is completed (busy flag is de-asserted to '0').

3.2.3 Send data test

Three user inputs are required to set total transmit length, packet size, and connection mode (active open for client operation or passive open for server operation). The operation will be cancelled if the input is invalid. During the test, 32-bit increment data is generated from the logic and sent to Test PC or FPGA. Data is verified by Test application on Test PC or verification module on FPGA. The operation is finished when total data are transferred from Intel PAC to Test PC or FPGA. The sequence of this test is as follows.

- 1) Receive transfer size, packet size, and connection mode from user and verify that the value is valid.
- 2) Set CCIUsrReg registers, i.e. transfer size (USER_TXLEN_REG), reset flag to clear initial value of test pattern (USER_RST_REG[0]='1'), and command register to start PattGen (USER_CMD_REG=0). After that, PattGen in CCIUsrReg transmits data to TOE10G-IP.
- 3) Display recommended parameter of test application running on Test PC by reading current parameters in the system.
- 4) Open connection following connection mode.
 - a. For active open, the Intel PAC host sets TOE10_CMD_REG=2 and monitors ConnOn status (USER_CMD_REG[2]) until it is equal to '1'.
 - b. For passive open, the Intel PAC host waits until connection is opened by Test PC or FPGA. ConnOn status (USER_CMD_REG[2]) is monitored until it is equal to '1'.
- 5) Set packet size to TOE10_PKL_REG and calculate total loops from total transfer size. Maximum transfer size of each loop is 4 GB. The operation of each loop is as follows.
 - a. Set transfer size of this loop to TOE10_TDL_REG. The set value is equal to remaining transfer size for the last loop or equal to 4 GB for the other loops.
 - b. Set send command (0x0) to TOE10_CMD_REG.
 - c. Wait until operation is completed by monitoring busy flag (TOE10_CMD_REG[0]). The operation is finished when busy flag changes to '0'. During monitoring busy flag, the Intel PAC host reads current transfer size from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays the results on the Linux terminal every second.
- 6) Set close connection command (0x3) to TOE10_CMD_REG.
- 7) Calculate performance and show test result on the Linux terminal.

3.2.4 Receive data test

User sets total received size, data verification mode (enable or disable), and connection mode (active open for client operation or passive open for server operation). The operation will be cancelled if the input is invalid. During the test, 32-bit increment data is generated to verify the received data from Test PC or FPGA when data verification mode is enabled. The sequence of this test is as follows.

- 1) Receive total transfer size, data verification mode, and connection mode from user input. Verify that all inputs are valid.
- 2) Set CCIUsrReg registers, i.e. reset flag to clear initial value of test pattern (USER_RST_REG[0]='1') and data verification mode (USER_CMD_REG[1]='0' or '1').
- 3) Display recommended parameter (same as Step 3 of Send data test).
- 4) Open connection following connection mode (same as Step 4 of Send data test).
- 5) Wait until connection is closed by Test PC or FPGA. ConnOn status (USER_CMD_REG[2]) is monitored until it is equal to '0'. During monitoring ConnOn, the Intel PAC host reads current transfer size from user logic (USER_TXLEN_REG and USER_RXLEN_REG) and displays the results on the Linux terminal every second.
- 6) Read total received length of user logic (USER_RXLEN_REG) and wait until read value is equal to total size set from user. After total data is received, verification result is read to check failure flag (USER_CMD_REG[1]='0' for normal condition). If the error is detected, error message will be displayed.
- 7) Calculate performance and show test result on the Linux terminal.

3.2.5 Full duplex test

This menu is designed to run full duplex test by transferring data between Intel PAC and Test PC or FPGA in both directions by using same port number at the same time. Four inputs are received from user, i.e. total size for both directions, packet size for PattGen, data verification mode for PattVer, and connection mode (active open/close for client operation or passive open/close for server operation).

When running the test by using Test PC and Intel PAC, transfer size setting on Intel PAC must be matched to the size setting on test application (tcp_client_txrx_40G). Connection mode on Intel PAC when running with Test PC must be set to server (passive operation).

The test runs in forever loop until user cancels operation. User inputs Ctrl+C to cancel the operation when running with Test PC. If running with FPGA, user inputs some keys to the terminal. The sequence of this test is as follows.

- 1) Receive total data size, packet size, data verification mode, and connection mode from user and verify that the value is valid.
- 2) Display recommended parameter of test application running on Test PC by reading current parameters in the system.
- 3) Set CCIUsrReg registers, i.e. transfer size (USER_TXLEN_REG), reset flag to clear initial value of test pattern (USER_RST_REG[0]='1'), and command register to start PattGen with data verification mode of PattVer (USER_CMD_REG=1 or 3).
- 4) Open connection following connection mode (same as Step 4 of Send data test).
- 5) Set TOE10G-IP registers, i.e. packet size (TOE10_PKL_REG=user input) and calculate total transfer size in each loop. Maximum size of one loop is 4 GB. The operation of each loop is as follows.
 - a. Set transfer size of this loop to TOE10_TDL_REG. Except the last loop, transfer size in each loop is set to maximum size (4GB) which is also aligned to packet size. For the last loop, transfer size is equal to the remaining size.
 - b. Set send command (0x0) to TOE10_CMD_REG.
 - c. Wait until send command is completed by monitoring busy flag (TOE10_CMD_REG[0]). When operation is finished, busy flag changes to '0'. During monitoring busy flag, the Intel PAC host reads current transfer size from USER_TXLEN_REG and USER_RXLEN_REG and displays the results on the terminal every second.
- 6) Close connection following connection mode.
 - a. For active close, the Intel PAC host waits until transfer size is equal to set value. Then, set USER_CMD_REG=3 to close connection. Next, the Intel PAC host waits until connection is closed by monitoring ConnOn (USER_CMD_REG[2])='0'.
 - b. For passive close, the Intel PAC host waits until connection is closed by Test PC or FPGA. ConnOn (USER_CMD_REG[2]) is monitored until it is equal to '0'.
- 7) Check received result and error (same as Step 6 of Receive data test).
- 8) Calculate performance and show test result on the Linux terminal. Go back to step 3 to run the test in forever loop.

3.3 Function list in User application

This topic describes the function list to run TOE10G-IP operation. The initialization sequence before starting AFU is not described in this topic.

void exec_port(unsigned int port_ctl, unsigned int mode_active)	
Parameters	port_ctl: 1-Open port, 0-Close port mode_active: 1-Active open/close, 0-Passive open/close
Return value	None
Description	For active mode, write TOE10_CMD_REG to open or close connection. After that, call read_conon function to monitor connection status until it changes from ON to OFF or OFF to ON, depending on port_ctl mode.

void init_param(void)	
Parameters	None
Return value	None
Description	Set network parameters to TOE10G-IP register from global parameters. After reset is de-asserted, it waits until TOE10G-IP busy flag is de-asserted to '0'.

int input_param(void)	
Parameters	None
Return value	0: Valid input, -1: Invalid input
Description	Receive network parameters from user, i.e. mode, window threshold, FPGA MAC address, FPGA IP address, FPGA port number, Target IP address, and Target port number. If the input is valid, the parameters will be updated. Otherwise, same values are used. After receiving all parameters, the current value of each parameter is displayed.

Unsigned int read_conon(void)	
Parameters	None
Return value	0: Connection is OFF, 1: Connection is ON.
Description	Read value from USER_CMD_CONNON register and return only bit2 value to show connection status.

void show_cursize(void)	
Parameters	None
Return value	None
Description	Read USER_TXLEN_REG and USER_RXLEN_REG, and then display in Byte, KByte, or MByte unit

void show_param(void)	
Parameters	None
Return value	None
Description	Display current value of network parameters setting to TOE10G-IP such as IP address, MAC address, and port number.

void show_result(void)	
Parameters	None
Return value	None
Description	Read USER_TXLEN_REG and USER_RXLEN_REG to display total size. Read global parameters (timer_val_end and timer_val_start) and calculate total time usage to display in usec, msec, or sec unit. Finally, transfer performance is calculated and displayed on MB/s unit.

int toe10g_recv_test(void)	
Parameters	None
Return value	0: Operation is successful -1: Receive invalid input or error is found
Description	Run Receive data test following described in topic 3.2.4

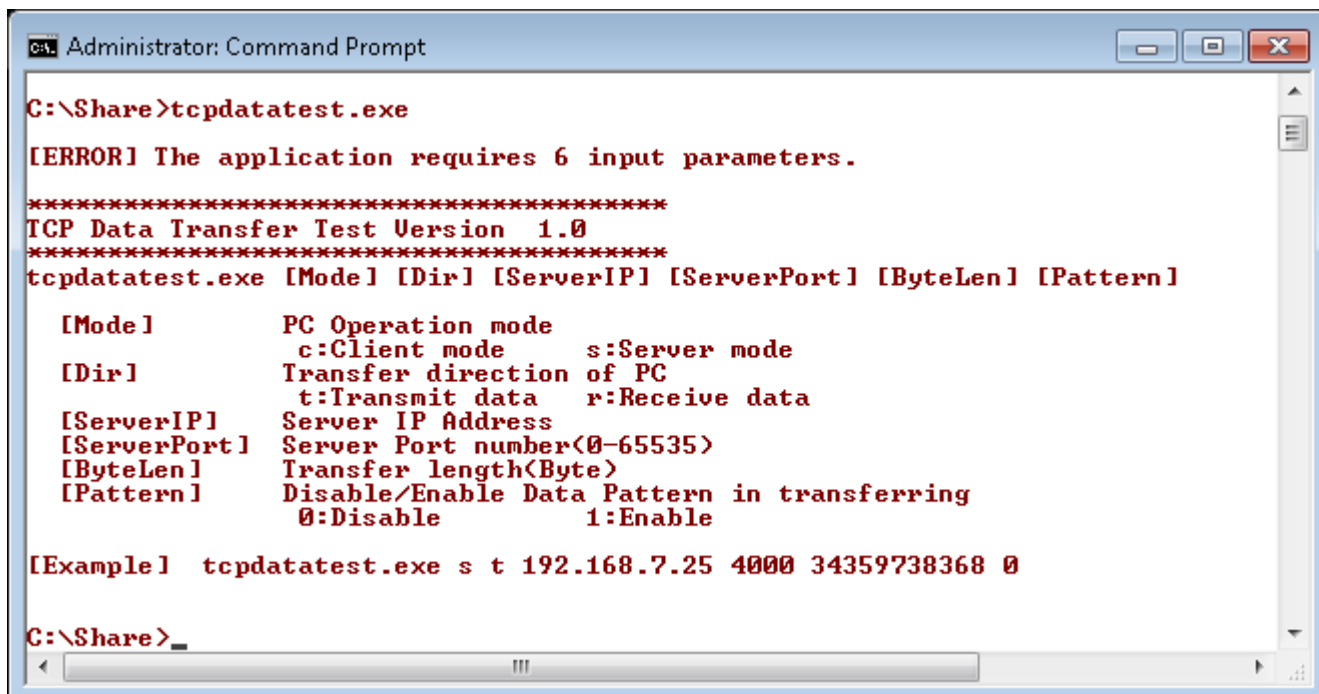
int toe10g_send_test(void)	
Parameters	None
Return value	0: Operation is successful -1: Receive invalid input or error is found
Description	Run Send data test following described in topic 3.2.3

int toe10g_txrx_test(void)	
Parameters	None
Return value	0: Operation is successful -1: Receive invalid input or error is found
Description	Run Full duplex test following described in topic 3.2.5

void wait_ethlink(void)	
Parameters	None
Return value	None
Description	Read USER_RST_REG[16] and wait until linkup status is found

4 Test Software on Test PC

4.1 “tcpdatatest” for half duplex test



```

Administrator: Command Prompt

C:\Share>tcpdatatest.exe

[ERROR] The application requires 6 input parameters.

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

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

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

C:\Share>_
  
```

Figure 4-1 “tcpdatatest” application usage

“tcpdatatest” is designed to run on Test PC for sending or receiving TCP data through Ethernet in both server or client mode. Test PC of this demo should run in client mode. User inputs parameter to select transfer direction and the mode. Six parameters are required as follows.

- 1) Mode : c – Test PC runs in client mode and Intel PAC runs in server mode
- 2) Dir : t – transmit mode (Test PC sends data to Intel PAC)
r – receive mode (Test PC receives data from Intel PAC)
- 3) ServerIP: IP address of Intel PAC when Test PC runs in client mode (default is 192.168.7.42)
- 4) ServerPort: Port number of Intel PAC when Test PC runs in client mode (default is 4000)
- 5) ByteLen: Total transfer size in byte unit. This input is used in transmit mode only and ignored in receive mode. In receive mode, the application is closed when the connection is destroyed. ByteLen in transmit mode must be equal to the total transfer size on Intel PAC, set in received data test menu.
- 6) Pattern :
0 – Generate dummy data in transmit mode or disable data verification in receive mode.
1 – Generate increment data in transmit mode or enable data verification in receive mode.

Transmit data mode

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

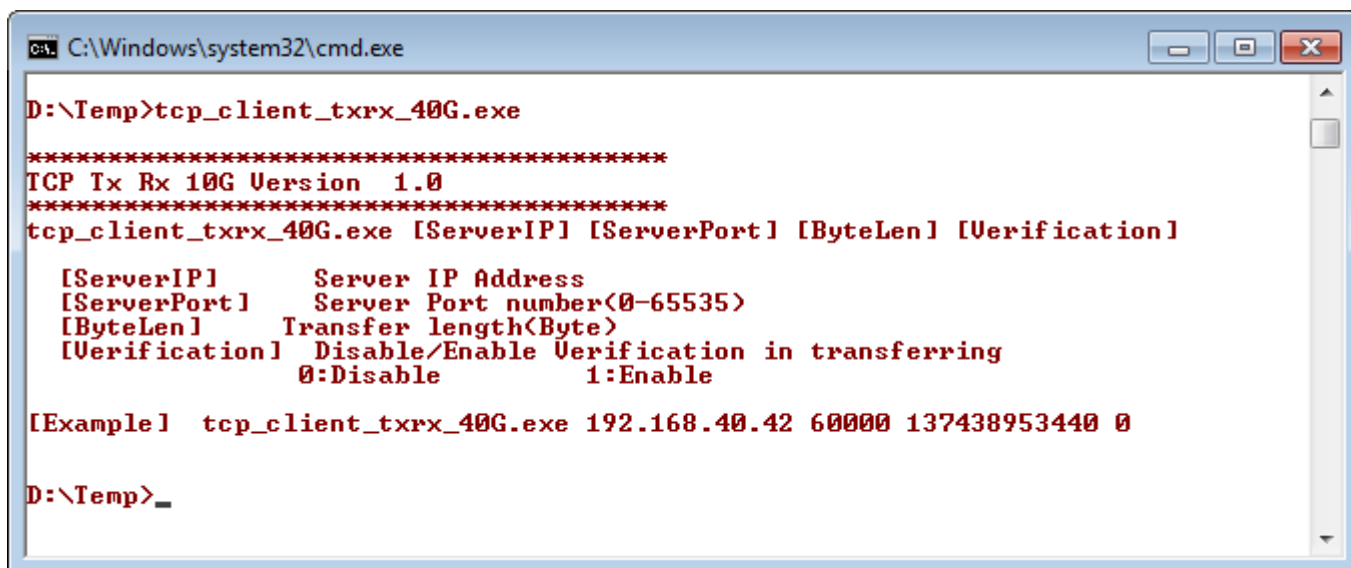
- 1) Allocate 1 MB memory to be send buffer.
- 2) Create socket and set properties of send buffer.
- 3) Create new connection to server by using IP address and port number from user.
- 4) Generate increment test pattern to send buffer when test pattern is enabled. Skip this step if dummy pattern is selected.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

Receive data mode

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

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

4.2 “tcp_client_txrx_40G” for full duplex test



```

C:\Windows\system32\cmd.exe
D:\Temp>tcp_client_txrx_40G.exe
*****
TCP Tx Rx 10G Version 1.0
*****
tcp_client_txrx_40G.exe [ServerIP] [ServerPort] [ByteLen] [Verification]

[ServerIP]      Server IP Address
[ServerPort]    Server Port number(0-65535)
[ByteLen]       Transfer length(Byte)
[Verification] Disable/Enable Verification in transferring
                0:Disable          1:Enable

[Example] tcp_client_txrx_40G.exe 192.168.40.42 60000 137438953440 0

D:\Temp>_

```

Figure 4-2 “tcp_client_txrx_40G” application usage

“tcp_client_txrx_40G” application is designed to run on Test PC for sending and receiving TCP data through Ethernet by using same port number at the same time. The application is run in client mode, so user needs to input the server parameters (network parameters of TOE10G-IP). As shown in Figure 4-2, there are four parameters to run the application, i.e.

- 1) ServerIP: IP address of Intel PAC
- 2) ServerPort: Port number of Intel PAC
- 3) ByteLen: Total transfer size in byte unit. This is total size to transmit and receive data.
- 4) Verification:
 - 0 – Generate dummy data for sending function and disable data verification for receiving function. This mode is used to check the best performance of full-duplex transfer.
 - 1 – Generate increment data for sending function and enable data verification for receiving function.

The sequence of test application is as follows.

- (1) Allocate 60 KB memory for send and receive buffer.
- (2) Create socket and set properties.
- (3) Create new connection by using IP address and port number from user.
- (4) Generate increment test pattern to send buffer when test pattern is enabled. Skip this step if dummy pattern is selected.
- (5) Send data out and decrease remaining transfer size.
- (6) Read data from received buffer and increase total received size.
- (7) If verification is enabled, data will be verified by increment pattern. Error message is printed out when data is not correct. Skip this step if data verification is disabled.
- (8) Print total transfer size every second.
- (9) Run step 5) – 8) until total sending data and total receiving data are equal to ByteLen (input from user).
- (10) Print total size and performance and close socket.
- (11) Sleep for 1 second to wait the hardware complete the current test loop.
- (12) Run step 3) – 11) in forever loop. If verification is failed, the application will quit.

5 Revision History

Revision	Date	Description
1.0	3-Apr-19	Initial version release