

TOE40G-IP reference design by Intel PAC A10GX

Rev1.0 4-Jul-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>

!

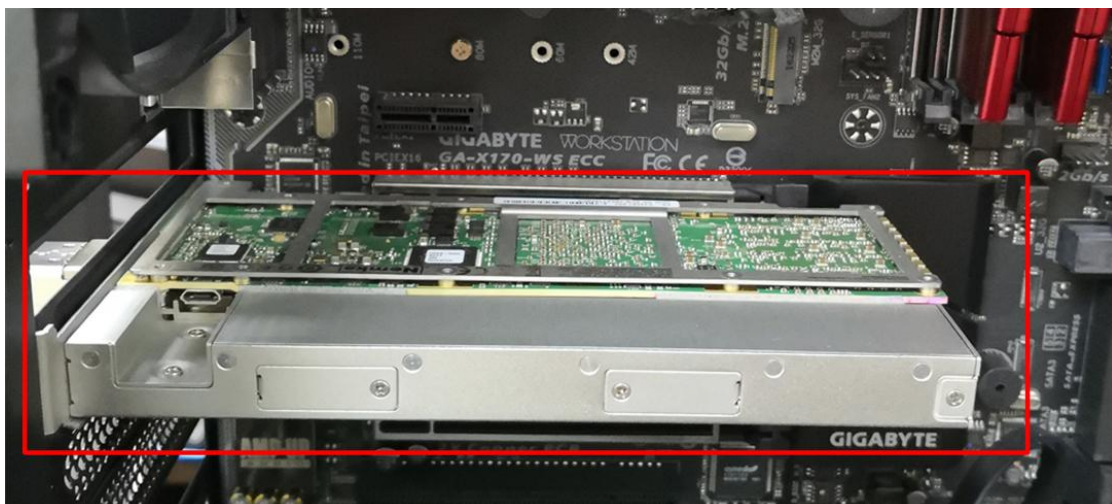


Figure 1-2 Intel PAC installation on Motherboard

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.

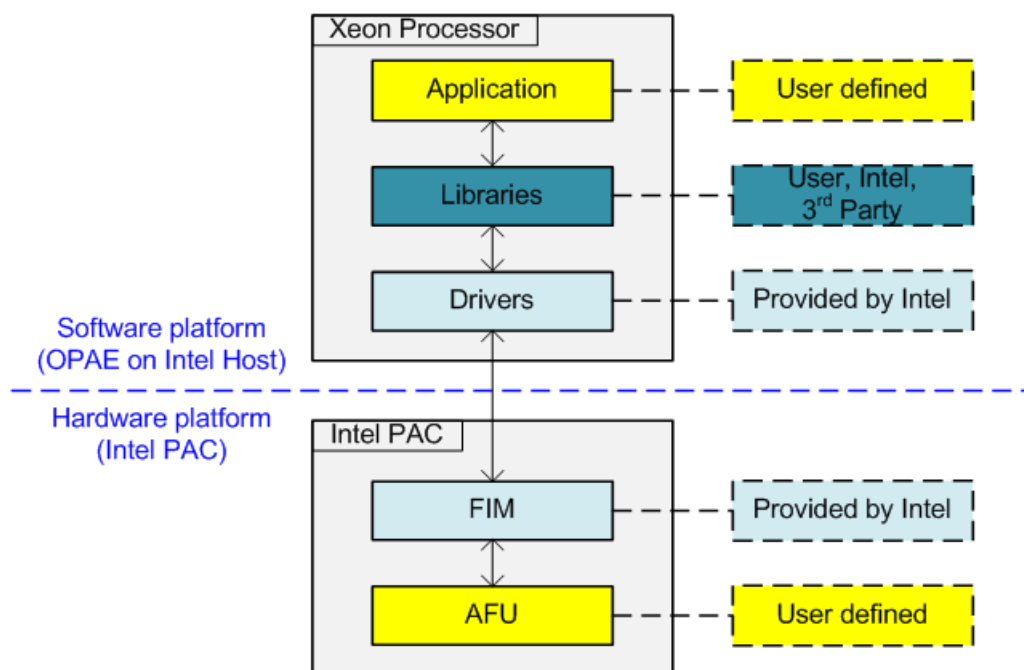


Figure 1-3 Intel PAC hardware and software overview

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 OpenCL.

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.

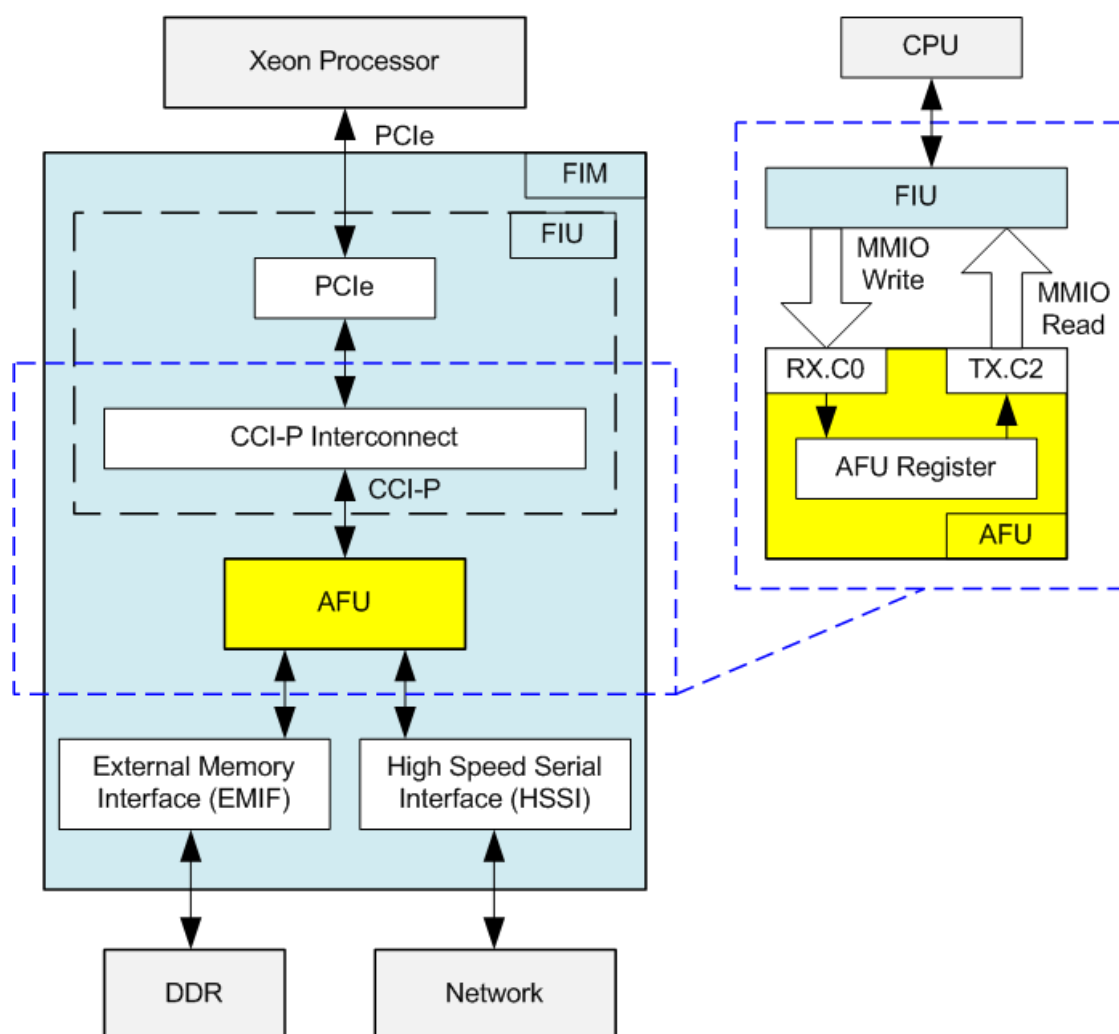


Figure 1-4 FPGA Interface Manager

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>

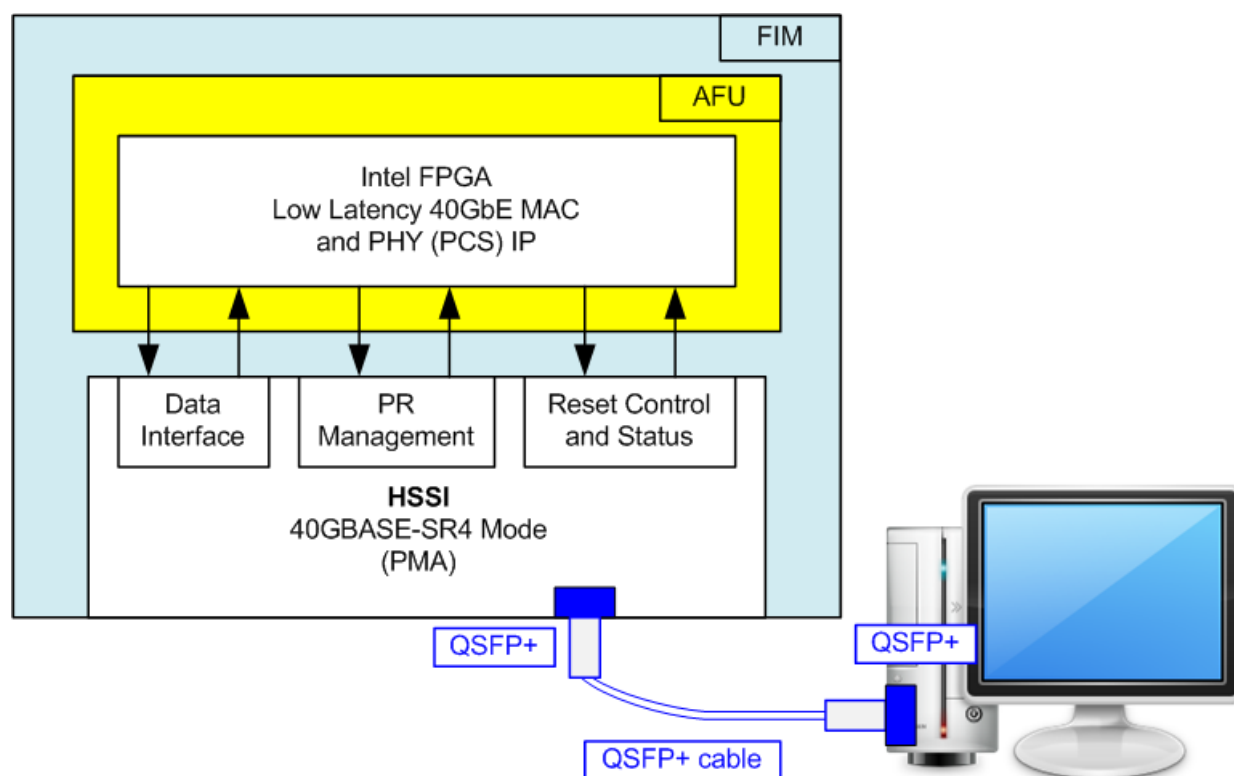


Figure 1-5 High Speed Serial Interface

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 40GBASE-SR4 to transfer 40G Ethernet packet with external network device such as PC. HSSI includes only PMA IP while PCS IP is implemented with 40GbE EMAC IP within AFU. More details of HSSI could be downloaded by following link.

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

IP Catalog of Quartus tool generates 40GbE EMAC IP subsystem which includes EMAC IP, PCS IP, and PMA IP, but PMA IP has already included in HSSI as shown in Figure 1-5. To implement 40GbE on Intel PAC, the EMAC IP needs to remove PMA IP. The EMAC without PMA IP is available in the 40GbE AFU design example which is located in the same location as the sample AFUs from the OPAE SDK installation. More details are described in Topic “4. Using the Design Example as a Platform for Further Evaluation” of UG-20163: 40Ggbps Ethernet Accelerator Functional Unit (AFU) Design Example User Guide.

<https://www.intel.com/content/www/us/en/programmable/documentation/pee1521131718500.html>

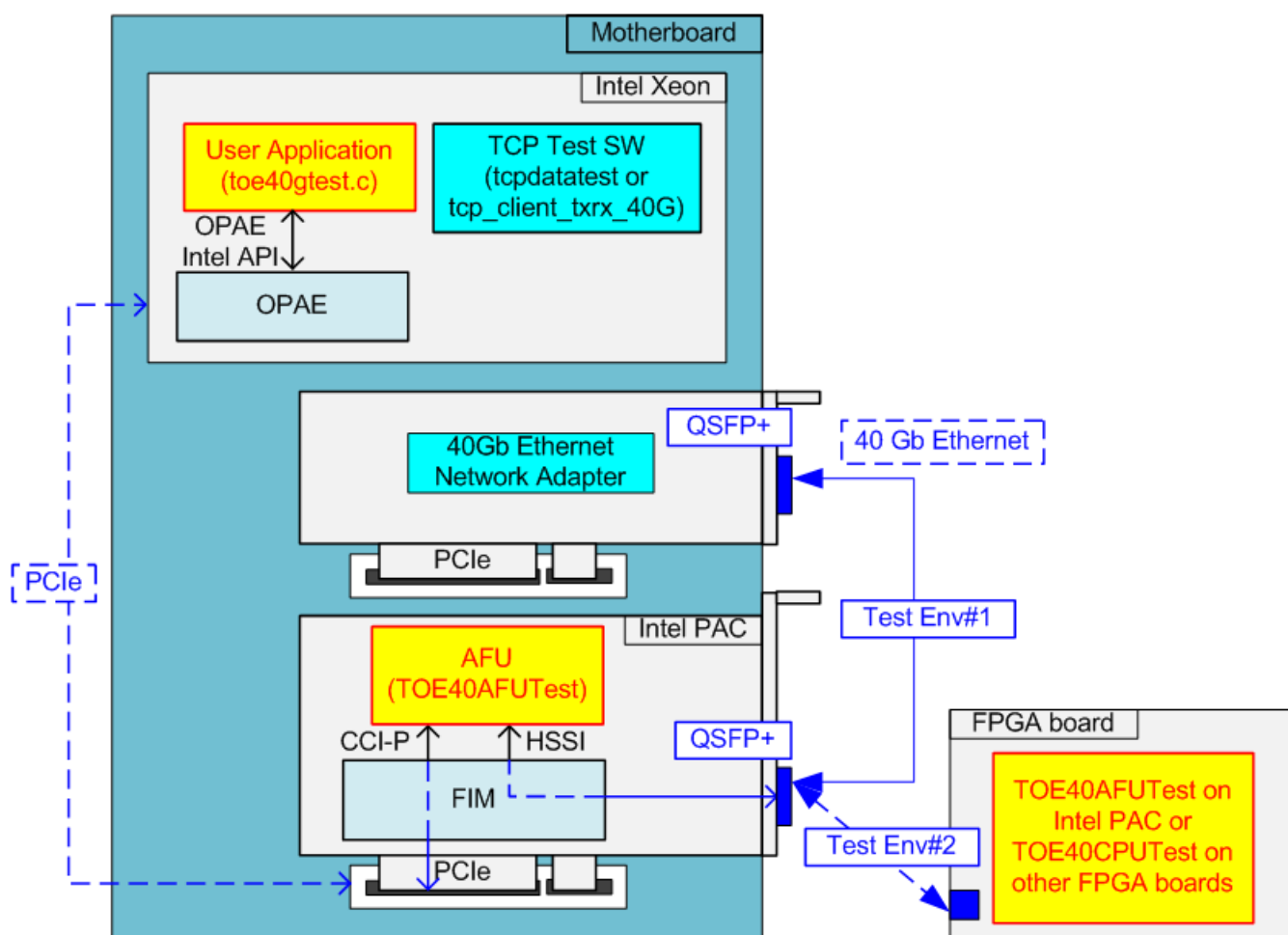


Figure 1-6 TOE40G-IP integrated on the Intel PAC platform

To build 40G Ethernet demo by using TOE40G-IP integrated on Intel PAC, the hardware of AFU and the user application running on OPAAE must be designed as shown in Figure 1-6. TOE40AFUTest is the AFU logic which includes CCI-P interface for register access by CPU and HSSI interface for 40G Ethernet I/O. The user application to set control register and parameter running on Intel Xeon is toe40qtest.

The destination network device to transfer 40 Gb Ethernet packet by using TCP/IP protocol may be 40 GbE Adapter or the FPGA board including 40 Gb Ethernet connection. When connecting 40 GbE Adapter to PCIe connector in the same PC as Intel PAC, the Intel Xeon must run another test software to transfer data following TCP/IP protocol, i.e. `tcpdataest` (half-duplex test) or `tcp_client_txrx` (full duplex test), provided by Design Gateway.

When using TOE40G-IP implemented on another FPGA (Arria10 GX board or Intel PAC), the best performance for transferring TCP/IP data is achieved. The details to setup TOE40G-IP with CPU reference design on another FPGA are downloaded from following link.

https://dgway.com/products/IP/TOE40G-IP/dg_toe40gip_refdesign_intel_en.pdf

https://dqway.com/products/IP/TOE40G-IP/dq_toe40gip_instruction_intel_en.pdf

2 AFU Hardware overview

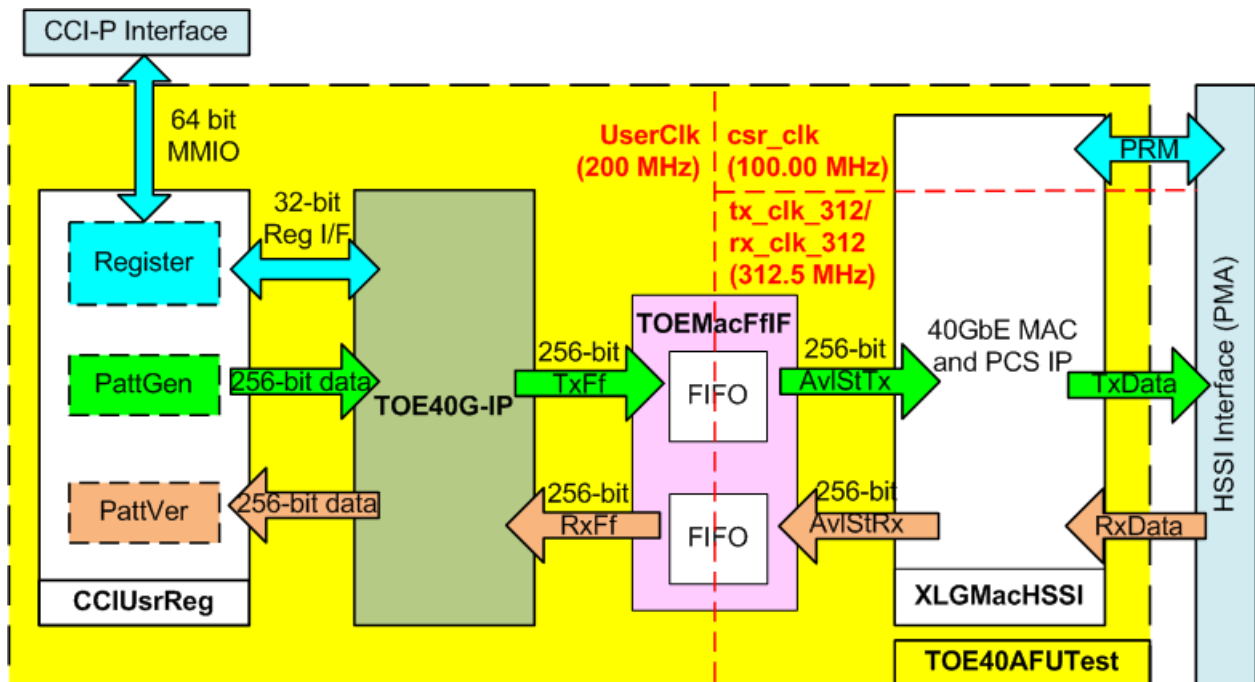


Figure 2-1 AFU block diagram

Figure 2-1 shows the logic details inside TOE40AFUTest module which is the AFU in the reference design. AFU includes TOE40G-IP for transferring TCP/IP packet. To complete all layer implementation, TOE10G-IP must connect to 40GbEMAC, PCS IP, and PMA IP which are provided as Intel FPGA IP. User interface of TOE40G-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, TOE40G-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 TOE40G-IP, PattGen, and PattVer by modifying the software on Xeon processor.

Otherwise, TOE40AFUTest module includes TOEMacFfIF module to be the interface logic between TOE40G-IP and 40 GbE MAC. There are asynchronous FIFO inside TOEMacFfIF to convert the clock domain between UserClk and 40 Gb EMAC clock. Data width of the FIFO is equal to 256 bit. UserClk frequency is equal to 200 MHz which is enough to support 40 Gbps transfer ($200\text{MHz} \times 256\text{-bit} = 51.2\text{ Gbps}$). More details of each module inside the AFU are described as follows.

2.1 XLGMachSSI

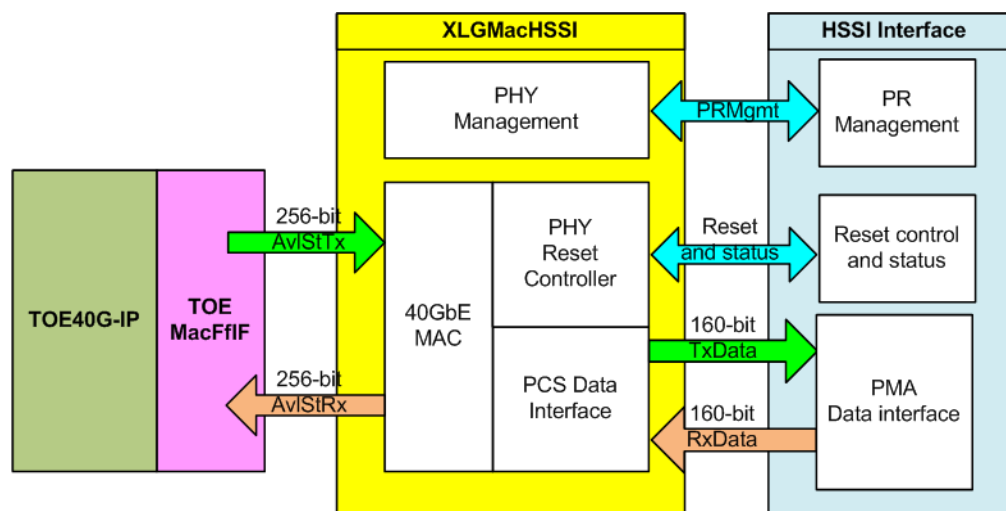


Figure 2-2 XLGMachSSI

XLGMachSSI is the modified 40 GbE MAC IP core to include only EMAC IP and PCS IP. PMA is included in HSSI instead. The interfaces for connecting with HSSI are PR management, Reset and status, and PCS data.

PR Management interface (prmgmt) of HSSI is connected to PHY management which is designed as the HDL code inside “XLGMachSSI.vhd”. “XLGMachSSI.vhd” is modified from “eth_e2e_e40.v” in 40GbE AFU design example to allow the port configuration by the internal logic only, not CPU. The PHY management logic is run under csr_clk which is free running clock at 100 MHz frequency.

The Reset and Status interface is controlled by PHY Reset controller logic which is implemented within 40GbE MAC module.

The data interface between PCS PHY data interface and PMA data interface is the unified data interface transmit and receive data ports. 40GBASE-SR4 mode uses 4 lanes of the transceiver, so the 160-bit data interface is split to four segments of 40-bit PCS PHY data to interface each transceiver lane. 40-bit PCS PHY data is appended by 88 bits zero data, so one lane of the HSSI interface has 128 bit data (40-bit PCS PHY data are the lower bits and 88-bit zero data are the upper bits). The control port of the unified data interface is not utilized. The transmit data port is run under tx_clk_312 while the receive data port is run under rx_clk_312.

More details of Low latency 40G Ethernet IP Core are described in following link.
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_II_40gbe.pdf

2.2 TOE40G-IP

TOE40G-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/TOE40G-IP/dg_toe40gip_data_sheet_intel_en.pdf

2.3 TOEMacFfIF

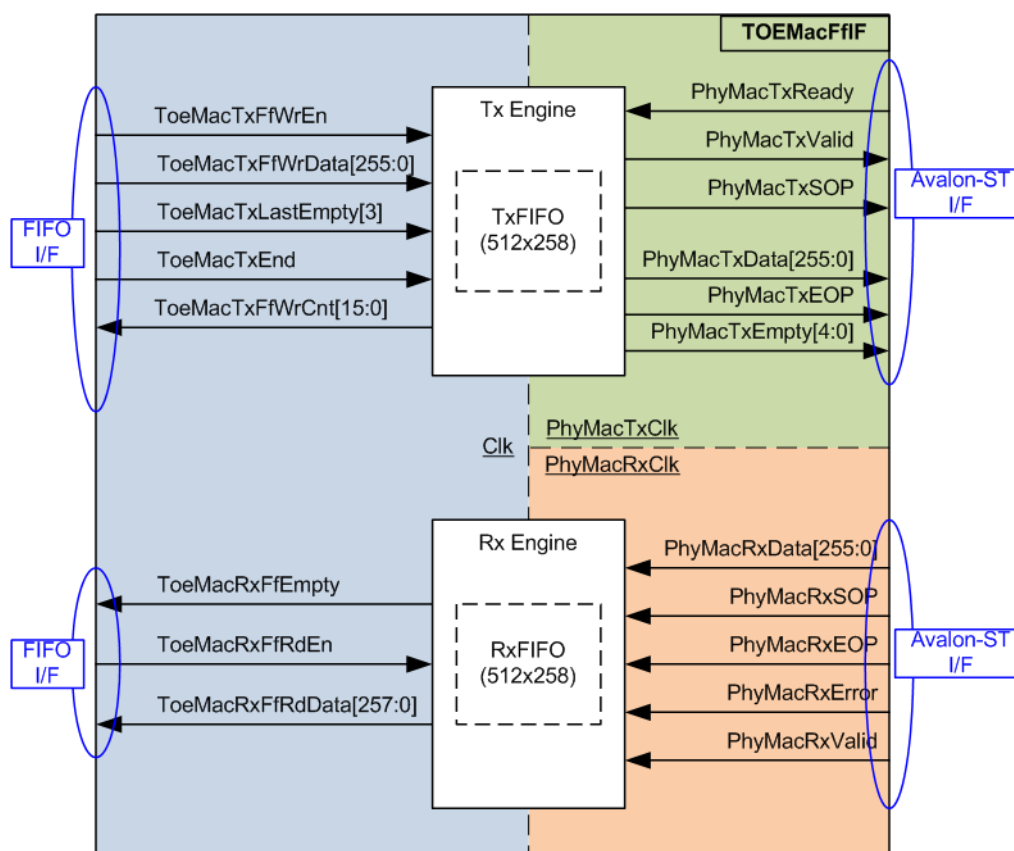


Figure 2-3 TOEMacFfIF block diagram

This module is designed to be the adapter logic connecting between TOE40G-IP and 40G Ethernet MAC. There are three clock domains in this module, i.e. Clk which is synchronous to TOE40G-IP, PhyMacTxClk which is synchronous to Tx interface of 40G EMAC, and PhyMacRxClk which is synchronous to Rx interface of 40G EMAC. Tx and Rx interface of TOE40G-IP is FIFO interface while 40G EMAC interface is Avalon-stream. So, the logic inside TOEMacFfIF is designed to convert interface type and support clock-crossing domain.

The logics to control transmit path and receive path are run independently, i.e. Tx Engine and Rx Engine. Both engines include asynchronous FIFOs (Tx FIFO and Rx FIFO) to convert Tx/Rx packet from one clock domain to another clock domain. FIFO size is 512x258-bit which is much enough to store 9K byte packet (maximum TCP packet size supported on 40G Ethernet card). More details of each engine are described as follows.

2.3.1 Tx Engine

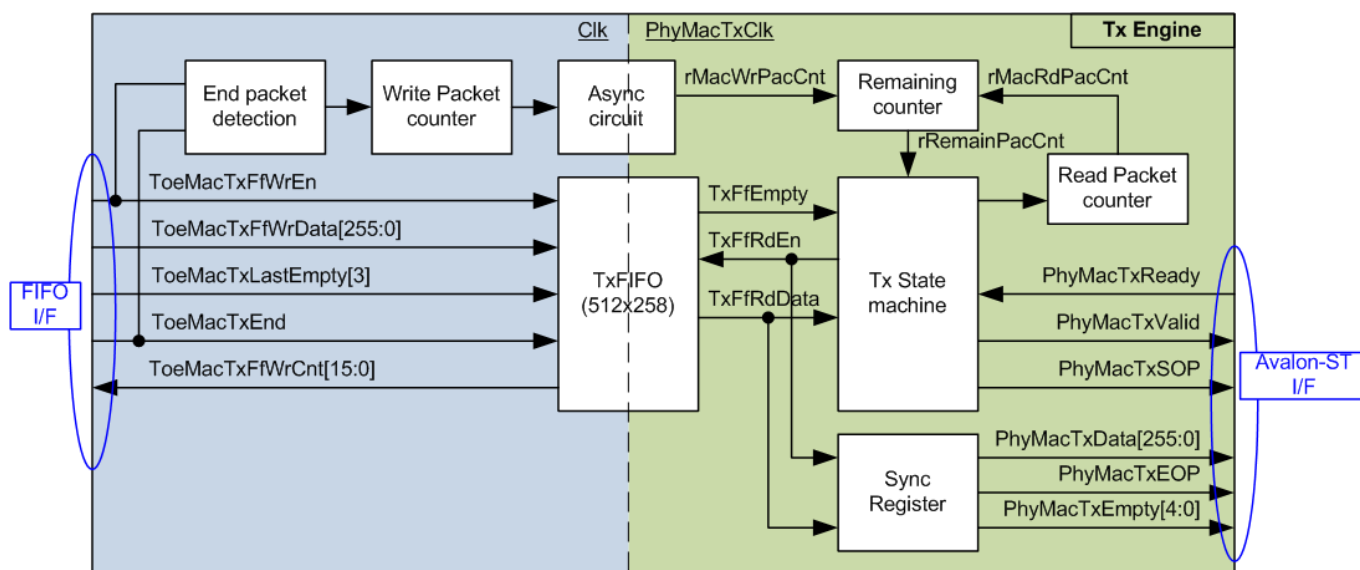


Figure 2-4 Tx Engine logic

On the left side, data of Tx interface from TOE10G-IP is stored to Tx FIFO within Tx Engine. The logic for monitoring Tx data interface to count the packet storing in Tx FIFO is designed as shown in the left side. “End packet detection” generates pulse signal to count the packet after end of packet is detected. “Write Packet counter” is the packet counter. Because TxLastEmpty signal from TOE40G-IP can be equal to two values, i.e. 0x04 and 0x0A, only bit 3 of TxLastEmpty signal is stored to Tx FIFO for optimizing resource.

On the right side, the packet counter value is forwarded from Clk domain to PhyMacTxClk domain. Read interface of Tx FIFO is controlled by Tx state machine. When one packet is stored to Tx FIFO, Tx state machine will forward that packet from Tx FIFO to 40G EMAC following Avalon-stream standard. Similar to the Write side, there is a packet counter designed to count the packet reading from Tx FIFO. The remaining packet (rRemainPacCnt) is calculated by total write packets (rMacWrPacCnt) – total read packets (rMacRdPacCnt). When rRemainPacCnt is more than or equal to 1, Tx State machine starts the packet forwarding operation.

The timing diagram of Tx Engine to read data from Tx FIFO and send the data to Avalon-ST interface is shown in Figure 2-5.

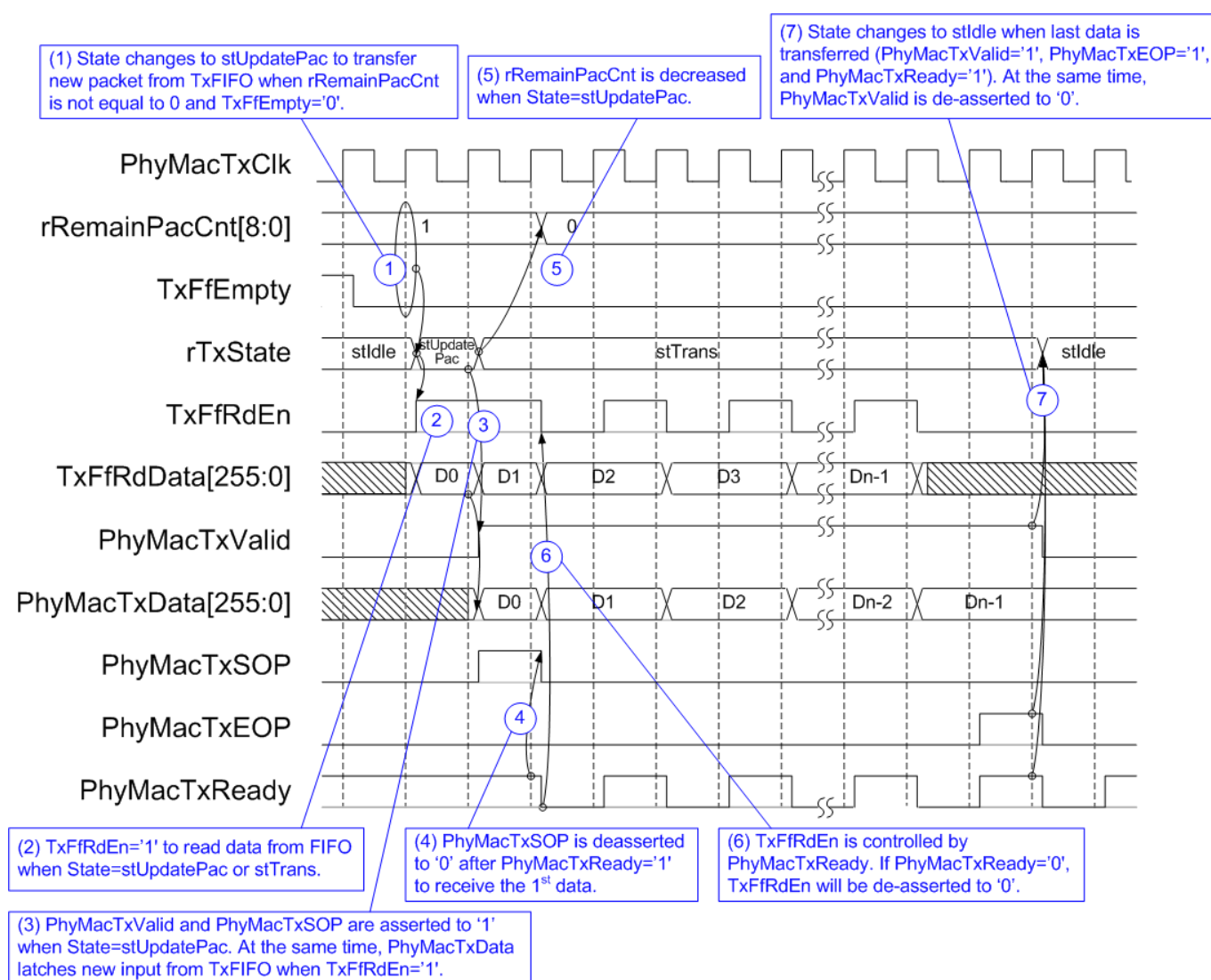


Figure 2-5 Tx Engine timing diagram

- 1) When at least one packet is available in TxFIFO (rRemainPacCnt is not equal to 0) and TxFfEmpty is equal to '0' (Read interface of TxFIFO is ready), rTxState forwards one packet from TxFIFO to 40G EMAC by changing state from stIdle to stUpdatePac.
- 2) stUpdatePac is run only one clock cycle to decrease rRemainPacCnt signal. TxFfRdEn is asserted to '1' at the same time. TxFIFO is Show-ahead type, so TxFfRdData is valid at the same clock as TxFfRdEn asserted to '1'.
- 3) In the next clock, TxState changes to stTrans and the 1st data is transferred to Avalon-ST interface (PhyMacTxSOP='1' and PhyMacTxValid='1'). PhyMacTxData is latched from TxFfRdData when TxFfRdEn is asserted to '1'. PhyMacTxData does not change when TxFfRdEn is de-asserted to '0'.
- 4) After PhyMacTxReady is asserted to '1' to get the 1st data, PhyMacTxSOP is de-asserted to '0'.
- 5) Total remaining data counter is decreased when the start of frame is transmitted.
- 6) In stTrans, TxFfRdEn is asserted to '1' following PhyMacTxReady signal, except the end of frame.
- 7) State changes to stIdle after end-of-frame is transmitted (PhyMacTxEOP='1' and PhyMacTxValid='1'). PhyMacTxValid is de-asserted to '0' to wait the next packet processing.

2.3.2 Rx Engine

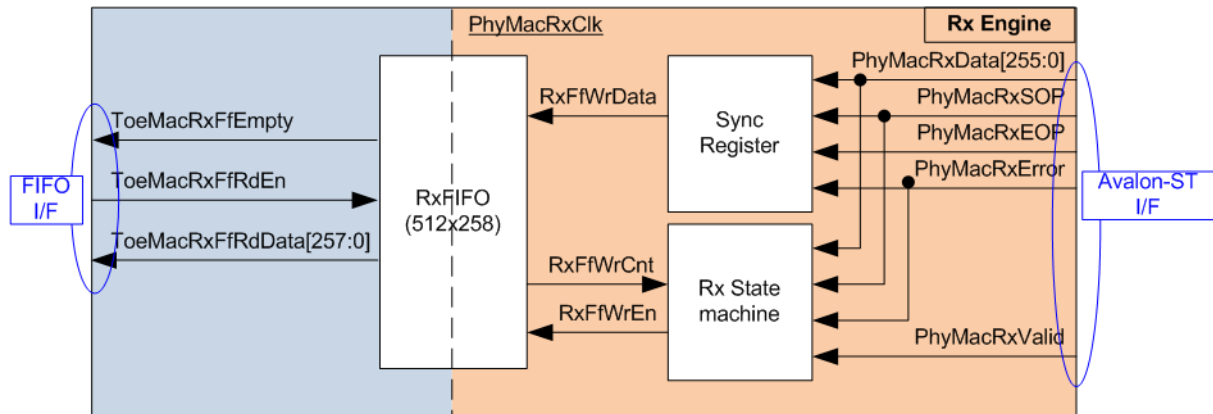


Figure 2-6 Rx Engine logic

Before storing the received packet from Avalon-ST interface to Rx FIFO, Rx state machine must check free space of Rx FIFO by monitoring Rx FfWrCnt firstly. If free space is more than 9 Kbyte, the received packet will be forwarded to Rx FIFO. Otherwise, the received packet is rejected. The timing diagram of Rx Engine is shown in Figure 2-7.

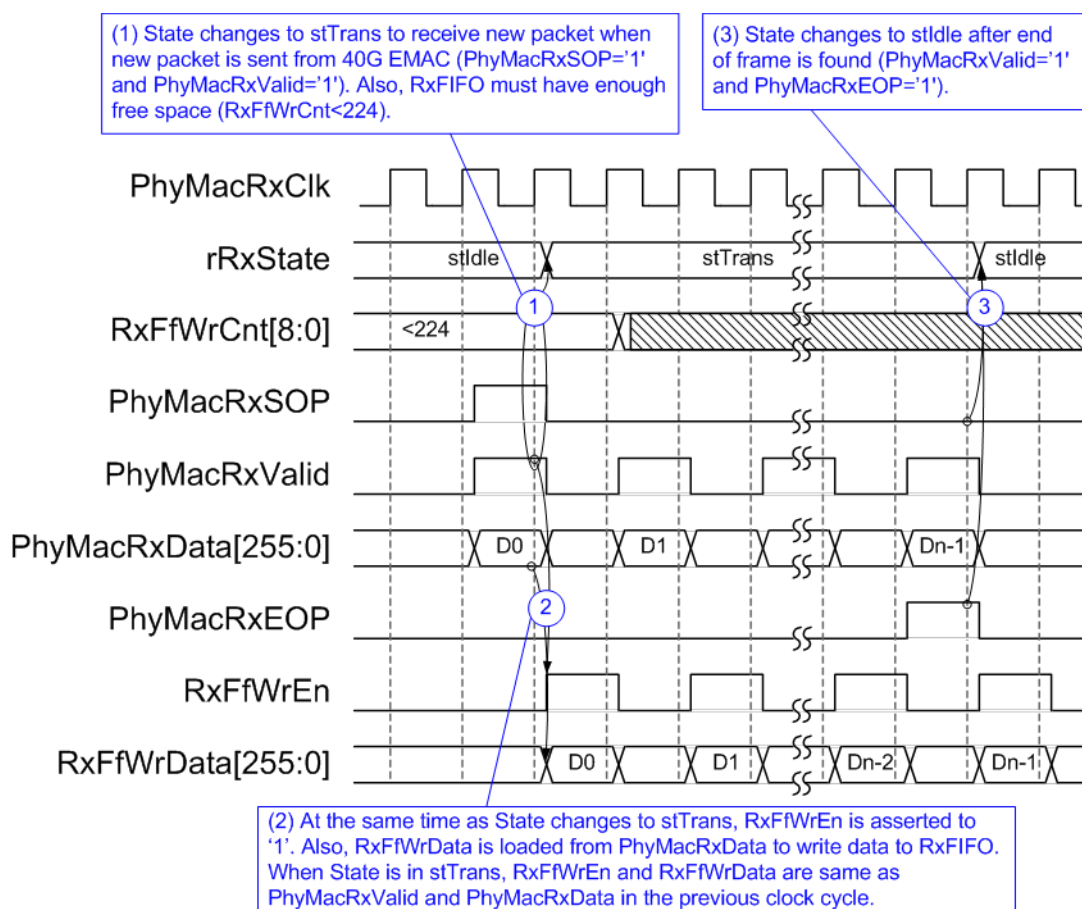


Figure 2-7 Timing diagram when Rx FIFO is ready

- 1) The new packet from Avalon-ST is stored to RxFIFO when at least 9 Kbyte is free in the FIFO, monitored by comparing RxFfWrCnt with 224 ((511 – 224) x 32 = 9184 byte). If RxFfWrCnt is less than 224, Rx state machine will change to stTrans to store the new packet to RxFIFO.
- 2) RxFfWrEn and RxFfWrData are 1-clock cycle delayed from PhyMacRxValid and PhyMacRxData respectively.
- 3) When end of packet is found (PhyMacRxValid='1' and PhyMacRxEOP='1'), Rx state machine will change to stIdle.

The example of timing diagram when the new packet is received but the FIFO is full is shown in Figure 2-8.

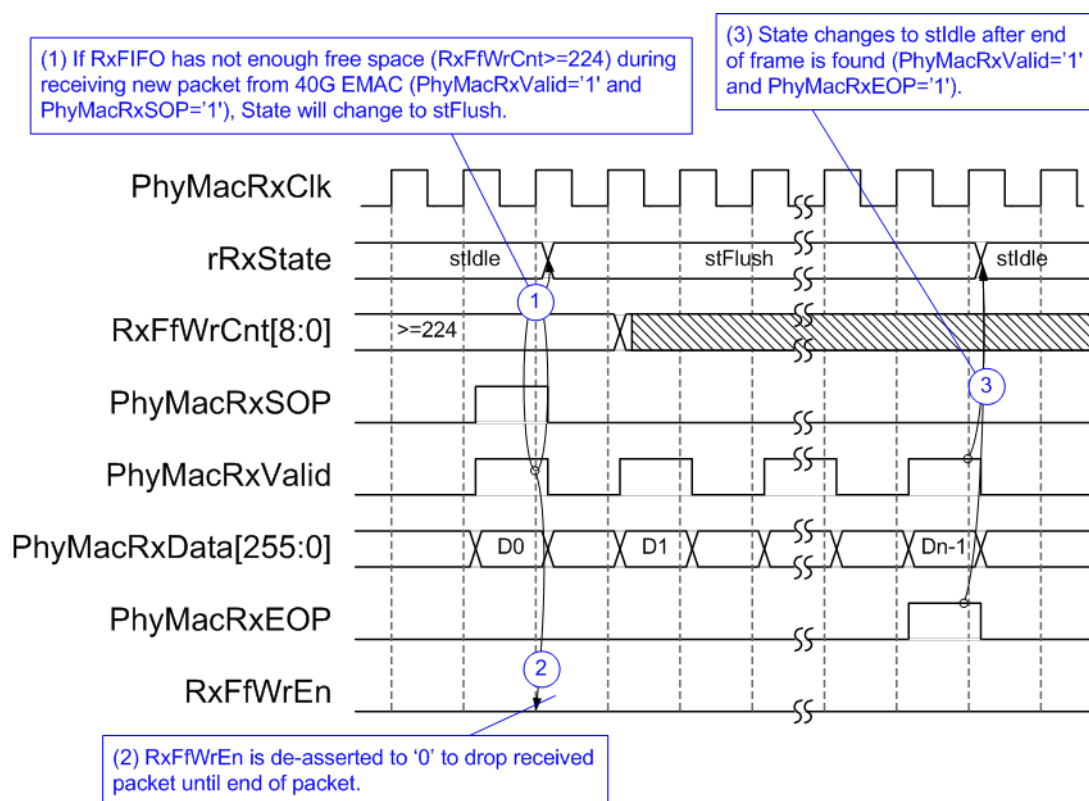


Figure 2-8 Timing diagram when Rx FIFO is not ready

- 1) When the new packet is received but RxFfWrCnt is more than or equal to 224, Rx state machine will change to stFlush.
- 2) In stFlush, RxFfWrEn is de-asserted to '0' to block new data storing to Rx FIFO until end of packet is found (PhyMacRxValid='1' and PhyMacRxEOP='1').
- 3) Rx state machine returns to stIdle to wait the new packet.

2.4 CCI2Reg

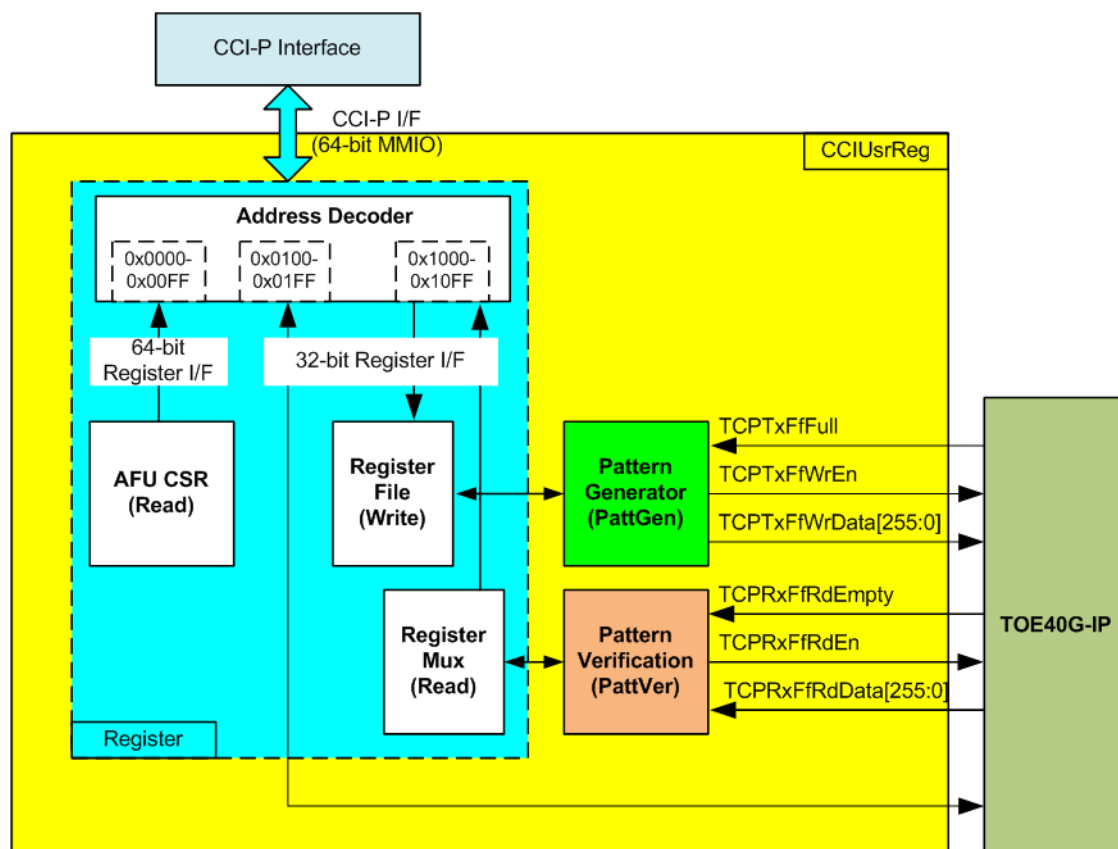


Figure 2-9 CCIUserReg block diagram

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 TOE40G-IP register interface. Pattern generator block is designed to send 256-bit test data to TOE40G-IP following FIFO interface standard. Pattern verification block is designed to read and verify 256-bit data from TOE40G-IP following FIFO interface standard. More details of each block are described as follows.

2.4.1 Register

As shown in Figure 2-9, register map of CCI-P interface is split into three areas, i.e. mandatory AFU CSR (0x0000-0x00FF), TOE40G-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 forwarding from MMIO to TOE40G-IP, PattGen, and PattVer is designed to align 64-bit unit. Since data bus size of TOE40G-IP registers and internal registers for controlling PattGen and PattVer is 32-bit, only the 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 2-10. 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. • Attribute: Read Only • [63:0]; Default: 0x0
AFU_ID_H	0x0004	0x0010	Upper 64 bits of the AFU_ID GUID. • Attribute: Read Only • [63:0]; Default: 0x0
DFH_RSVD0	0x0006	0x0018	[63:0] RSVD
DFH_RSVD1	0x0008	0x0020	[63:0] RSVD

Figure 2-10 Mandatory AFU CSRs (captured from MNL-1092 CCIP Reference manual)

Similar to write access, the read data returned from TOE40G-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 TOE40G-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 2-11.

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]

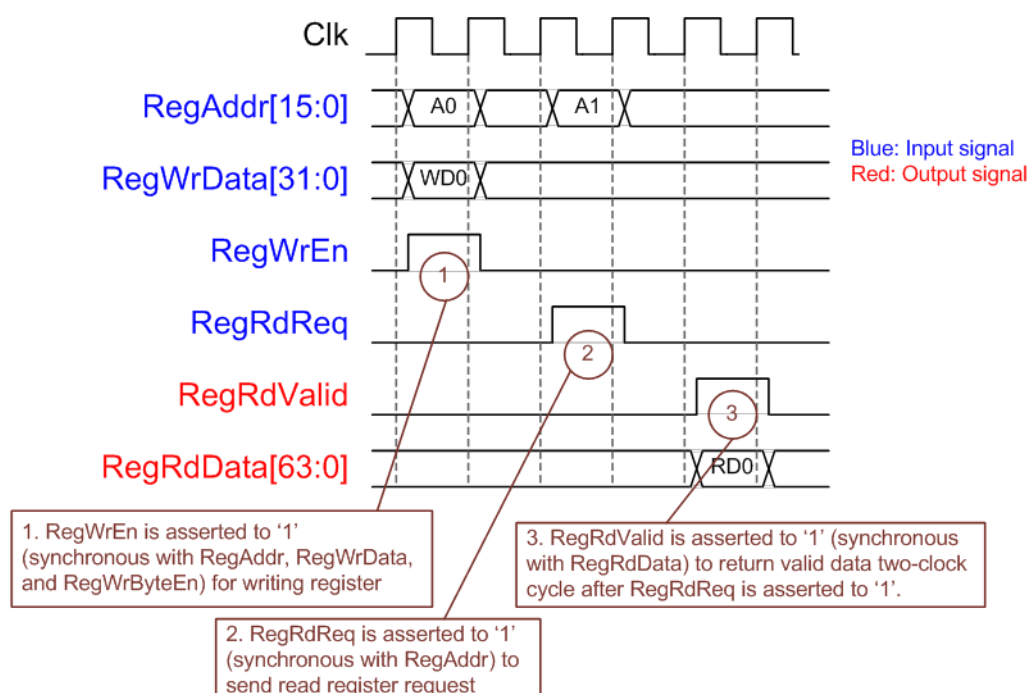


Figure 2-11 CCI-P interface for MMIO access timing diagram

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 TOE40G-IP or the internal signals of CCIUsrReg. The lower 32-bit data is forwarded to 32-bit write data for TOE40G-IP and the 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 from AFU CSR, TOE40G-IP, or the 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 are shown in Figure 2-12.

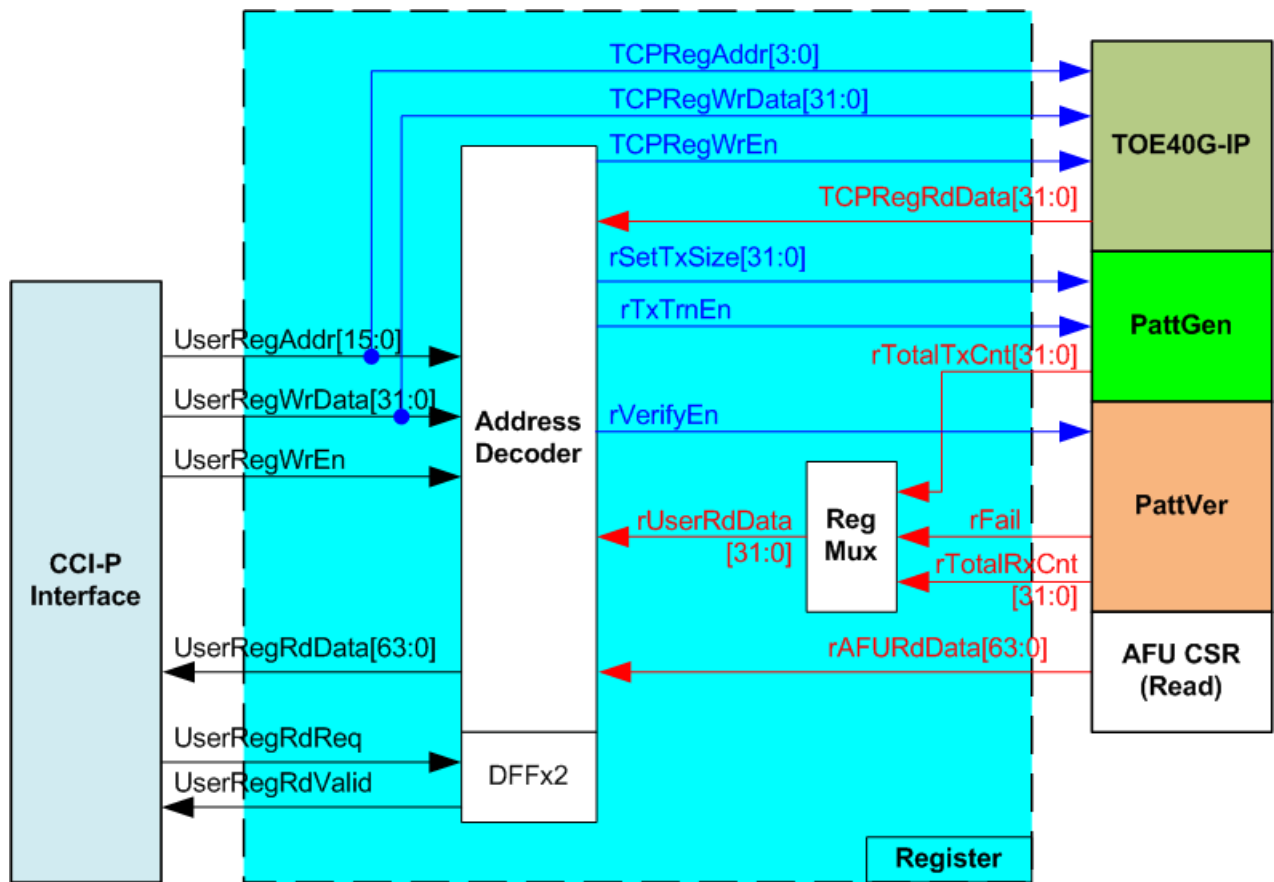


Figure 2-12 Register block

Data bus size of CCI-P is 64 bit, but the address is based on 32 bit. The address and data interface of TOE40G-IP is based on 32 bit. For simple logic design to convert CCI-P register signals to be TOE40G-IP register signals, CCI-P register address map for TOE40G-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 the upper bit of UserRegAddr is equal to TOE40G-IP area.

Similar to TOE40G-IP, the 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 size from TOE40G-IP, PattGen, and PattVer is 32 bit while AFU CSR data bus size is 64 bit.

Table 2-2 shows CCIUsrReq 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 "toe40gtest.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: TOE40G-IP Register Area More details of each register are described in Table2 of TOE40G-IP datasheet.		
0x0100	TOE40_RST_REG	[31:0] Mapped to RST register within TOE40G-IP
0x0108	TOE40_CMD_REG	[31:0] Mapped to CMD register within TOE40G-IP
0x0110	TOE40_SML_REG	[31:0] Mapped to SML register within TOE40G-IP
0x0118	TOE40_SMH_REG	[31:0] Mapped to SMH register within TOE40G-IP
0x0120	TOE40_DIP_REG	[31:0] Mapped to DIP register within TOE40G-IP
0x0128	TOE40_SIP_REG	[31:0] Mapped to SIP register within TOE40G-IP
0x0130	TOE40_DPN_REG	[31:0] Mapped to DPN register within TOE40G-IP
0x0138	TOE40_SPN_REG	[31:0] Mapped to SPN register within TOE40G-IP
0x0140	TOE40_TDL_REG	[31:0] Mapped to TDL register within TOE40G-IP
0x0148	TOE40_TMO_REG	[31:0] Mapped to TMO register within TOE40G-IP
0x0150	TOE40_PKL_REG	[31:0] Mapped to PKL register within TOE40G-IP
0x0158	TOE40_PSH_REG	[31:0] Mapped to PSH register within TOE40G-IP
0x0160	TOE40_WIN_REG	[31:0] Mapped to WIN register within TOE40G-IP
0x0168	TOE40_ETL_REG	[31:0] Mapped to ETL register within TOE40G-IP
0x0170	TOE40_SRV_REG	[31:0] Mapped to SRV register within TOE40G-IP
0x0178	TOE40_VER_REG	[31:0] Mapped to VER register within TOE40G-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 256-bit unit. Valid from 1-0xFFFFFFFF. Rd [31:0] – Completed size of PattGen in 256-bit unit. 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 TOE40G-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] – 40G EMAC Linkup ('0': Link is down, '1': Link is up)
0x1018 Rd	FIFO status (USER_FFSTS_REG)	Rd [4:0]: Mapped to TCPRxFfLastRdCnt[4:0] signal of TOE40G-IP [15:5]: Mapped to TCPRxFfRdCnt[10:0] signal of TOE40G-IP [24]: Mapped to TCPTxFfFull signal of TOE40G-IP
0x1020 Rd	Total Receive length (TRN_RXLEN_REG)	Rd [31:0] – Completed size of PattVer in 256-bit unit. The value is cleared to 0 when USER_CMD_REG is written by user.

2.4.2 Pattern Generator

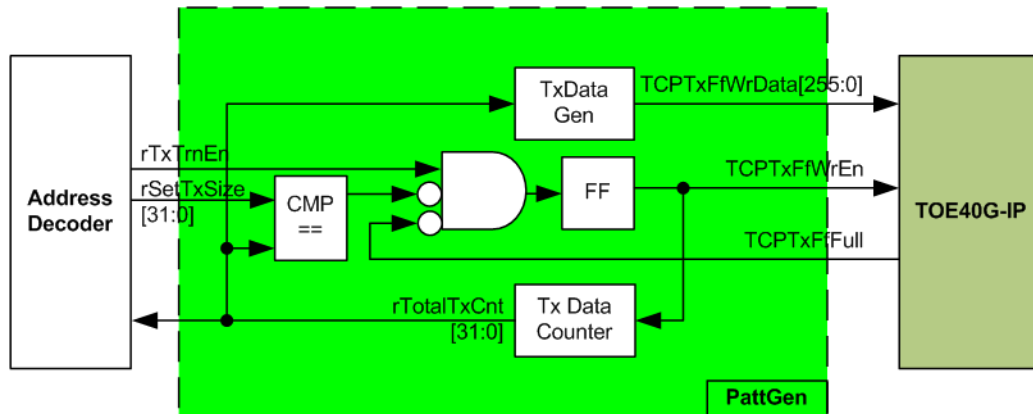


Figure 2-13 PattGen block

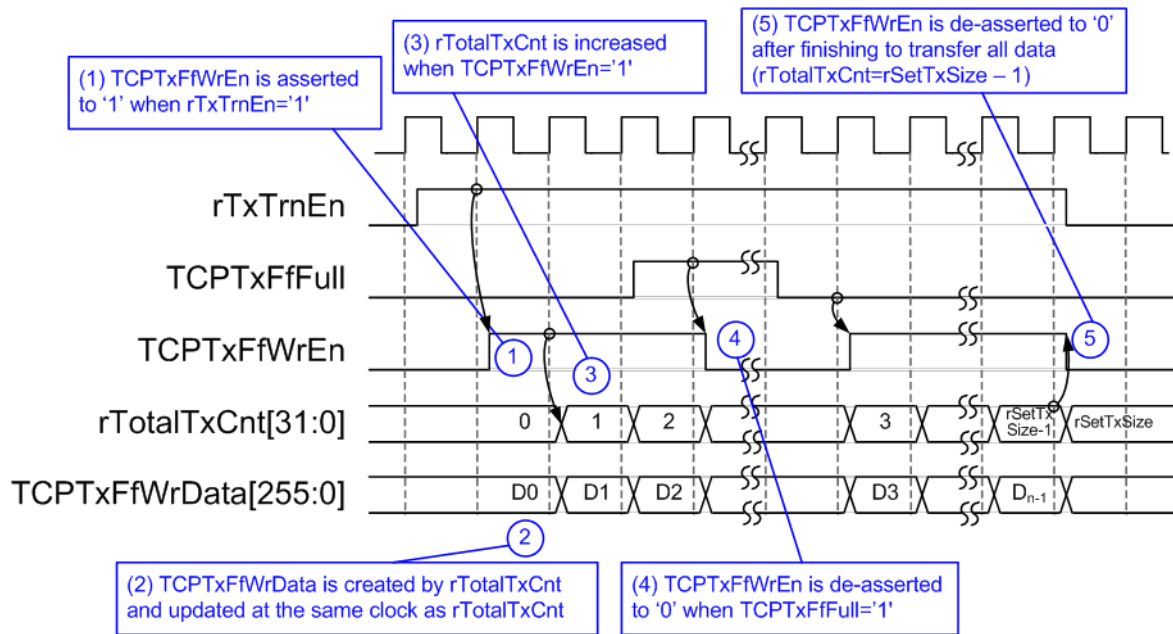


Figure 2-14 PattGen Timing diagram

PattGen is designed to generate test data to TOE40G-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 the data counter to check total data sending to TOE40G-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).

2.4.3 Pattern Verification

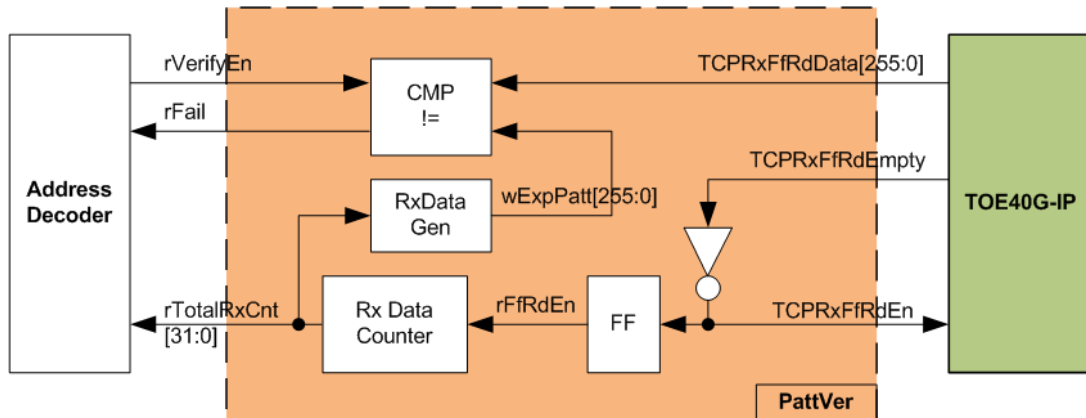


Figure 2-15 PattVer block

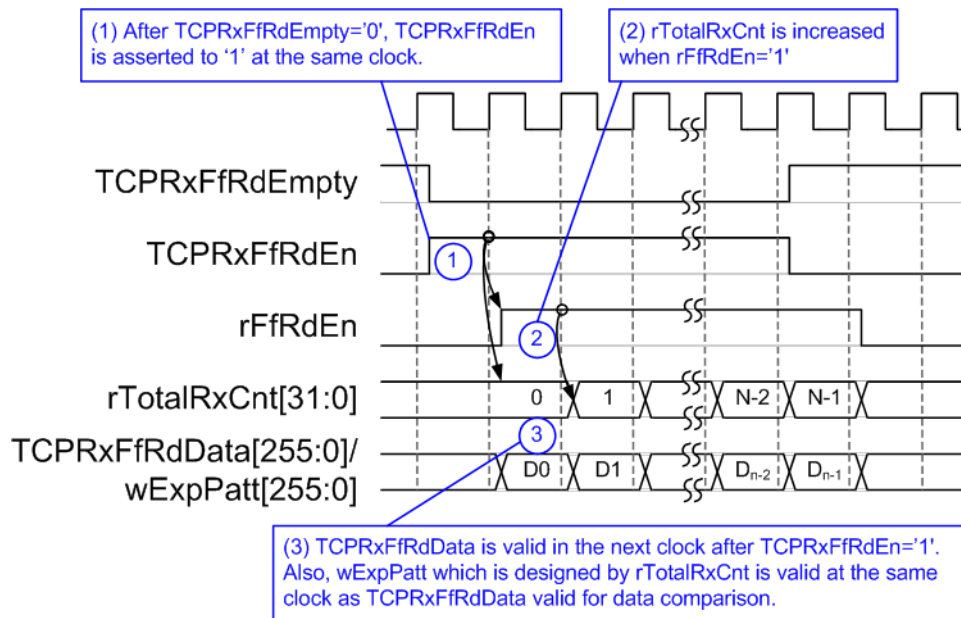


Figure 2-16 PattVer Timing diagram

PattVer is designed to read test data from TOE40G-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. After TCPRxFfRdEn is asserted to '1', 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 also used to generate wExpPatt (expected data to compare with TCPRxFfRdData).

3 User Application (Intel PAC)

3.1 Overview

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

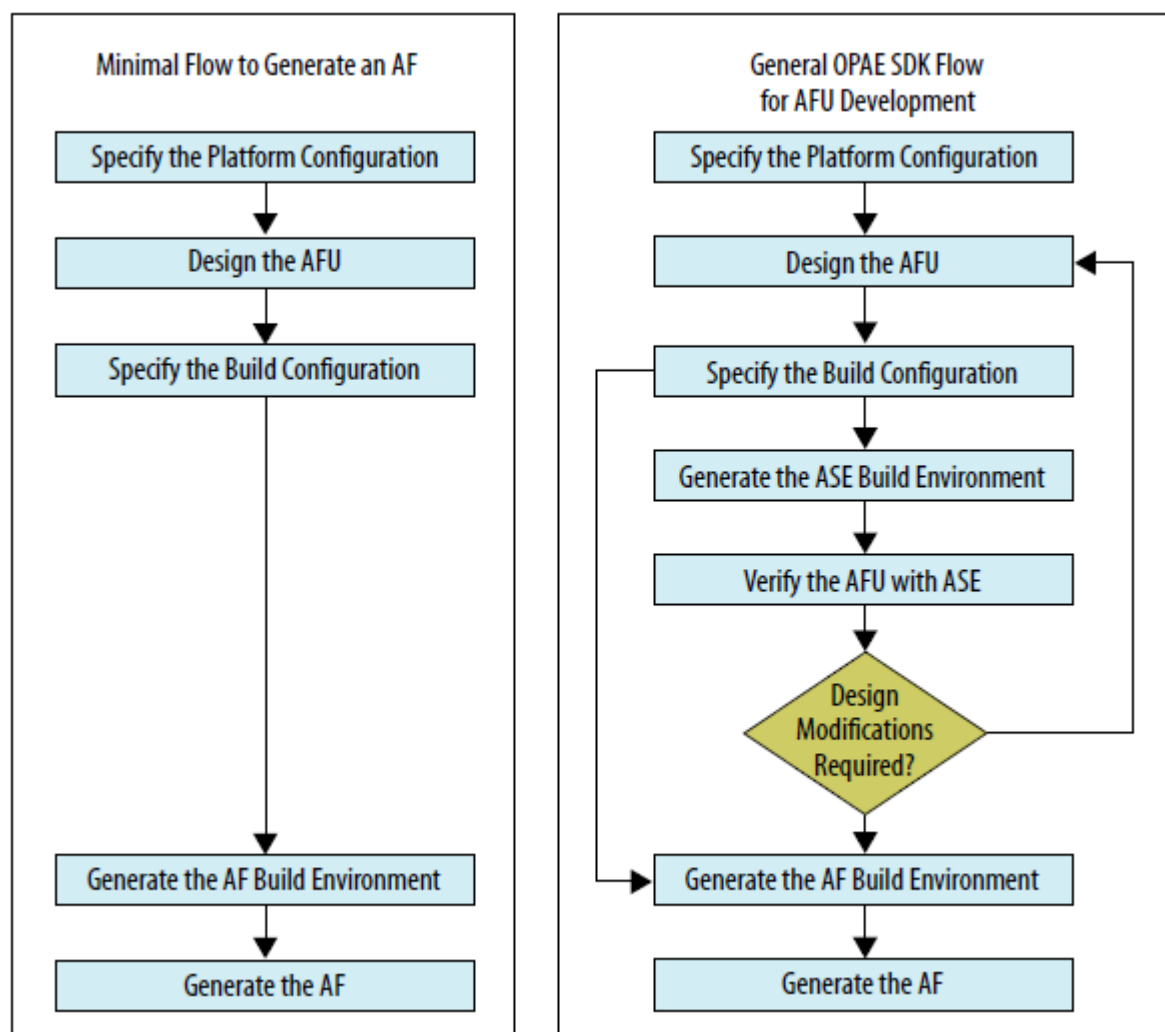


Figure 3-1 OPAE SDK Design Flow for AFU development (captured from UG-20169)

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 to develop 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 is built 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>

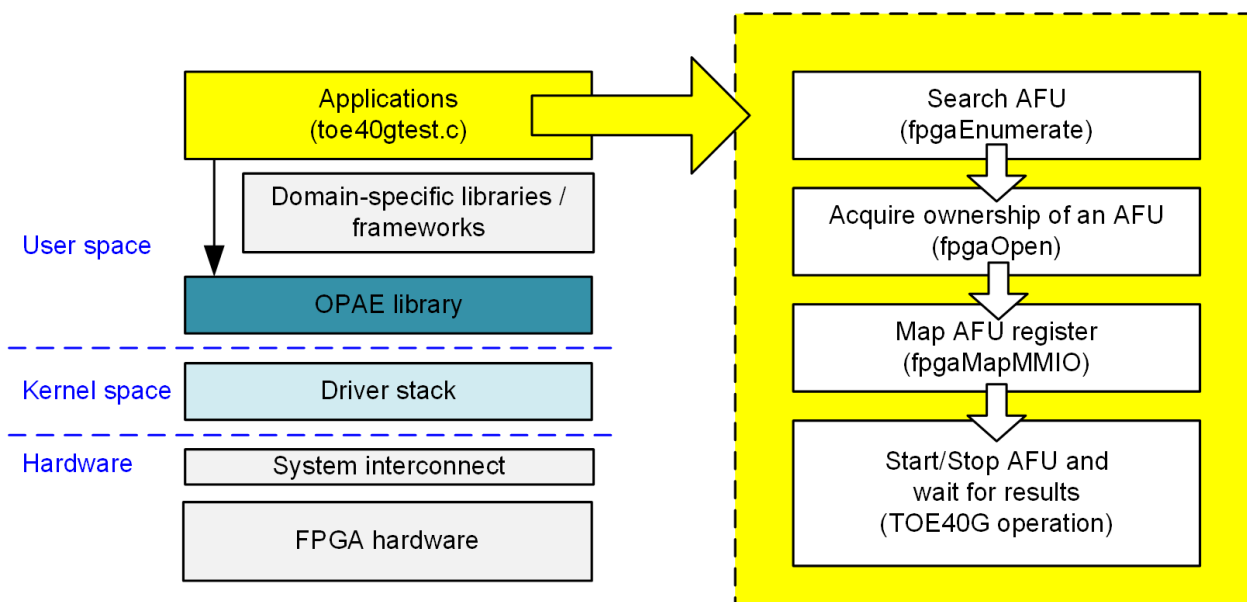


Figure 3-2 Software on Intel PAC PC

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 TOE40G-IP AFU is monitoring Ethernet linkup status. More details of TOE40G-IP software are described in the next topic.

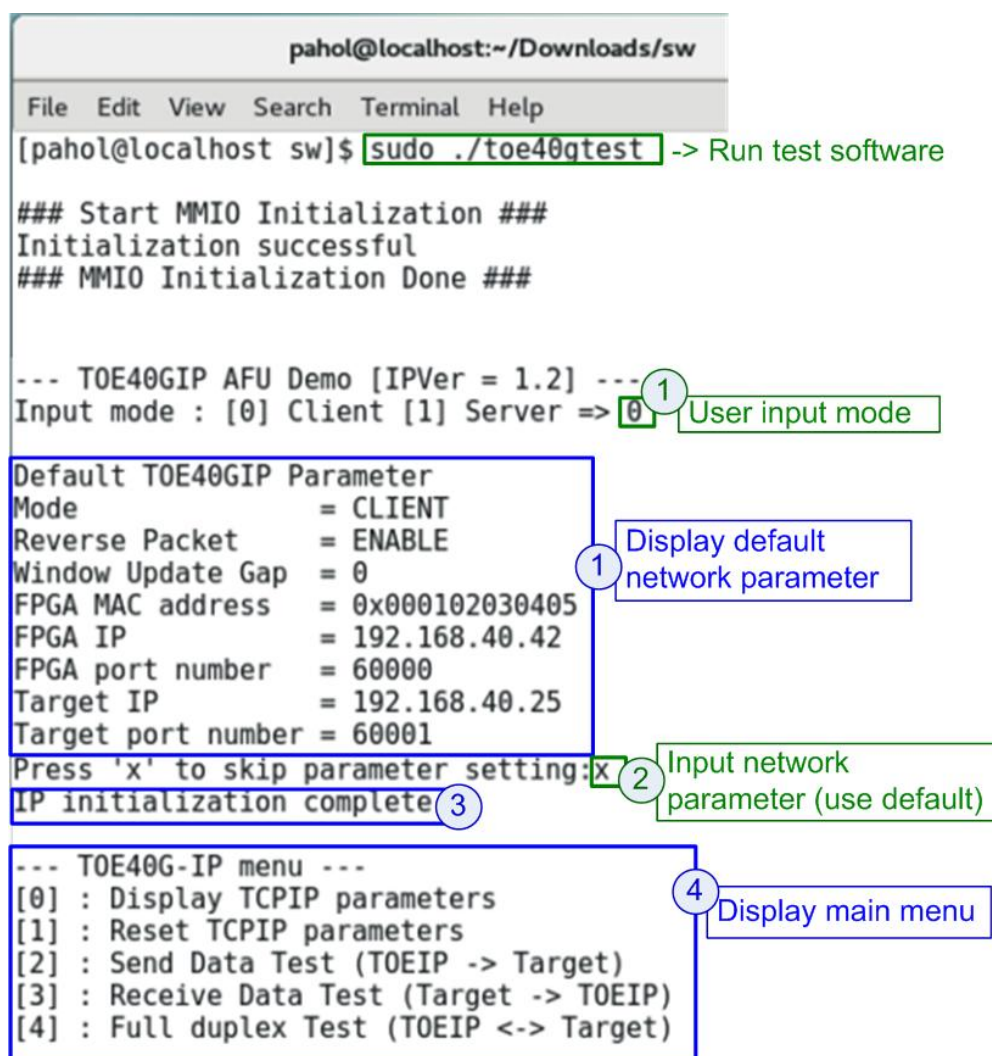
3.2 TOE40G-IP Test software

The software sequence of TOE40G-IP running on Intel PAC is same as TOE40G-IP reference design on other FPGA boards. The different point is the function to display message and receive parameter from user. After finishing MMIO mapping, 40G 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 TOE40G-IP to be client or server mode.

To initialize as client mode, TOE40G-IP sends ARP request to get the MAC address from the destination device. For server mode, TOE40G-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 TOE40G-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.



```

pahol@localhost:~/Downloads/sw
File Edit View Search Terminal Help
[pahol@localhost sw]$ sudo ./toe40gtest -> Run test software

### Start MMIO Initialization ###
Initialization successful
### MMIO Initialization Done ###

--- TOE40GIP AFU Demo [IPVer = 1.2] ---
Input mode : [0] Client [1] Server => 0

Default TOE40GIP Parameter
Mode           = CLIENT
Reverse Packet = ENABLE
Window Update Gap = 0
FPGA MAC address = 0x000102030405
FPGA IP        = 192.168.40.42
FPGA port number = 60000
Target IP      = 192.168.40.25
Target port number = 60001
Press 'x' to skip parameter setting: x
IP initialization complete

--- TOE40G-IP menu ---
[0] : Display TCPIP parameters
[1] : Reset TCPIP parameters
[2] : Send Data Test (TOEIP -> Target)
[3] : Receive Data Test (Target -> TOEIP)
[4] : Full duplex Test (TOEIP <-> Target)
  
```

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 TOE40G-IP finishing initialization sequence (monitoring TOE40_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 Display IP parameters

This menu is used to show current value of TOE40G-IP parameters 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 parameters

This menu is used to change TOE40G-IP parameters such as IP address and source port number. After setting the updated parameter to TOE40G-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 the current value of parameters on the Linux terminal.
- 2) Receive the new input parameters from user and check the input value whether valid or not. If which input is invalid, the value of that input will not change.
- 3) Force reset to IP by setting TOE40_RST_REG[0]='1'.
- 4) Set all parameters to TOE40G-IP register such as TOE40_SML_REG and TOE40_DIP_REG.
- 5) De-assert IP reset by setting TOE40_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 (TOE40_CMD_REG[0]) until the initialization sequence is finished (busy flag is de-asserted to '0').

3.2.3 Send data test

Three user inputs are required, i.e. total transmit length, packet size, and connection mode. The connection mode means 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 external device which may be Test PC or FPGA. Data is verified by the external device (Test application on Test PC or verification module on FPGA). The operation is finished when total data are transferred from Intel PAC to the external device. 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 TOE40G-IP.
- 3) Display the recommended parameter of test application running on Test PC.
- 4) Open connection following connection mode.
 - a. For active open, the Intel PAC host sets TOE40_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 TOE40_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 TOE40_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 TOE40_CMD_REG.
 - c. Wait until operation is completed by monitoring busy flag (TOE40_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 TOE40_CMD_REG.
- 7) Calculate performance and show test result on the Linux terminal.

3.2.4 Receive data test

User sets three parameters, i.e. 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 the external device (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 the recommended parameter of the software (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 the external device (Test PC or FPGA). The close status is monitored by reading ConnOn status (USER_CMD_REG[2]) which must be 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, the failure flag to show verification result is read (USER_CMD_REG[1]='0' when error is not found). 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). 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.
- 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 TOE40G-IP registers, i.e. packet size (TOE40_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 TOE40_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 TOE40_CMD_REG.
 - c. Wait until send command is completed by monitoring busy flag (TOE40_CMD_REG[0]). When the 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 the 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 the external device (Test PC or FPGA). ConnOn (USER_CMD_REG[2]) is monitored until it is equal to '0'.
- 7) Check received result and error status (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 TOE40G-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 TOE40_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 TOE40G-IP register from global parameters. After reset is de-asserted, it waits until TOE40G-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 TOE40G-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 toe40g_rcv_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 toe40g_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

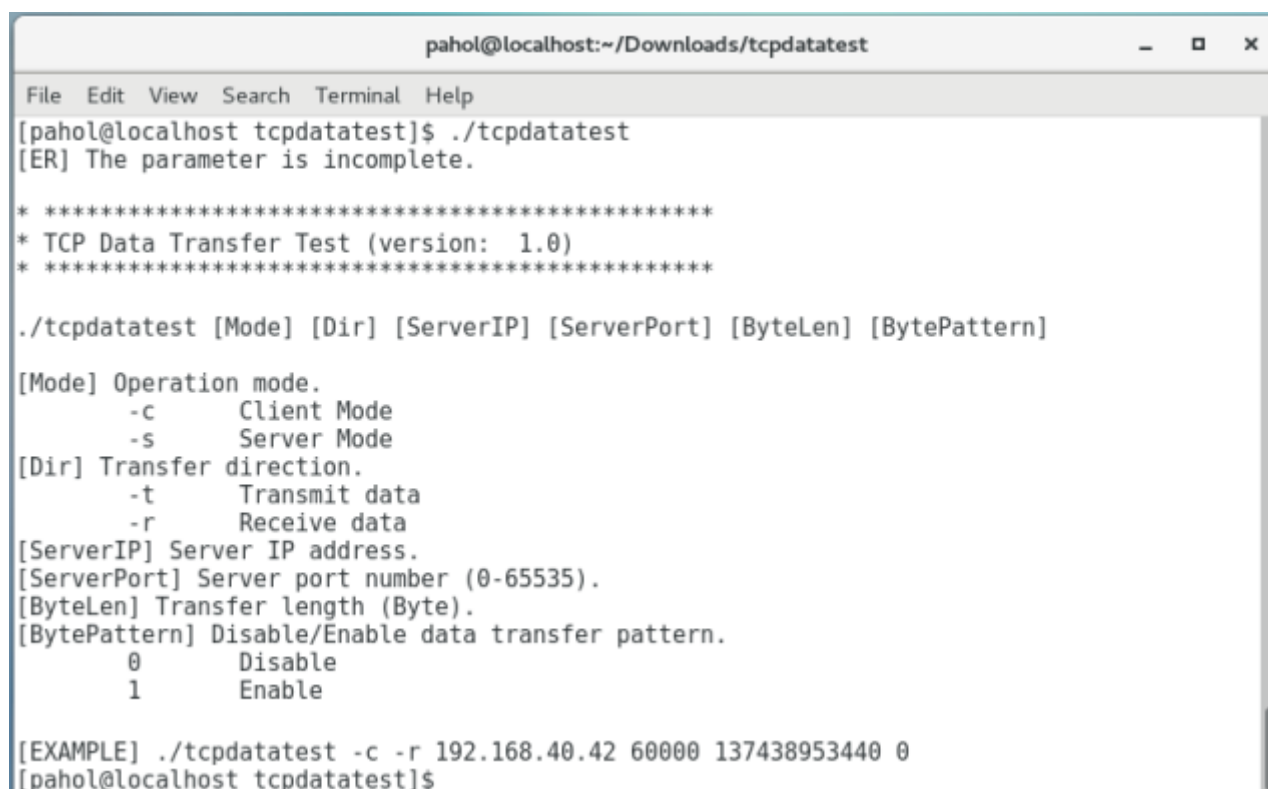
int toe40g_txrx_test(void)	
Parameters	None
Return value	0: Operation is successful -1: Receive invalid input or error is found
Description	Run Full duplex test following described in topic 3.2.5

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

4 TCP Test Software

Two test applications are designed to transfer TCP packets with Intel PAC, i.e. “tcpdatatest” for half-duplex test and “tcp_client_txrx” for full-duplex test. The test application is run on Linux OS. In the reference design, we run the application by using the same PC which Intel PAC installed. More details of the test software are described as follows.

4.1 “tcpdatatest” for half duplex test



```
pahol@localhost:~/Downloads/tcpdatatest
File Edit View Search Terminal Help
[pahol@localhost tcpdatatest]$ ./tcpdatatest
[ER] The parameter is incomplete.

* *****
* TCP Data Transfer Test (version: 1.0)
* *****

./tcpdatatest [Mode] [Dir] [ServerIP] [ServerPort] [ByteLen] [BytePattern]

[Mode] Operation mode.
      -c      Client Mode
      -s      Server Mode
[Dir] Transfer direction.
      -t      Transmit data
      -r      Receive data
[ServerIP] Server IP address.
[ServerPort] Server port number (0-65535).
[ByteLen] Transfer length (Byte).
[BytePattern] Disable/Enable data transfer pattern.
           0      Disable
           1      Enable

[EXAMPLE] ./tcpdatatest -c -r 192.168.40.42 60000 137438953440 0
[pahol@localhost tcpdatatest]$
```

Figure 4-1 “tcpdatatest” application usage

“tcpdatatest” is designed to send or receive TCP data through Ethernet in server or client mode. In this demo, only client mode is applied. Six parameters are necessary as follows.

- 1) Mode : -c – The software runs in client mode to communicate with Intel PAC which is run in server mode
- 2) Dir : -t – transmit mode (software sends data to Intel PAC)
-r – receive mode (software receives data from Intel PAC)
- 3) ServerIP: IP address of Intel PAC which runs as server mode (default is 192.168.7.42)
- 4) ServerPort: Port number of Intel PAC which runs as server mode (default is 4000)
- 5) ByteLen: Total transfer size in byte unit when running in transmit mode. The parameter is ignored when running in receive mode.
- 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.

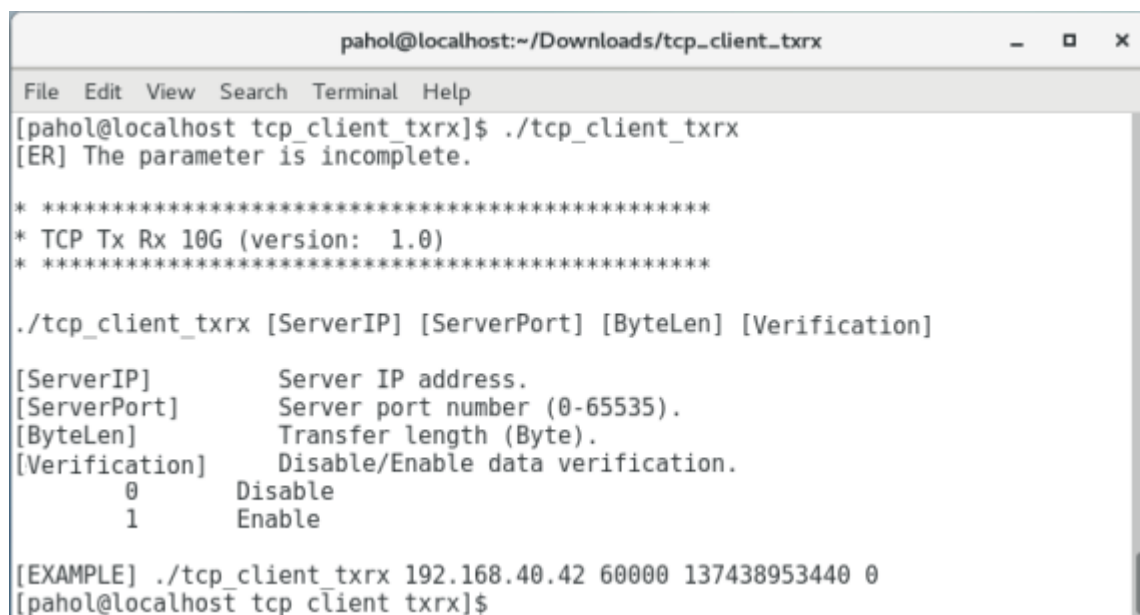
- 1) Get parameters from the user and verify the input that is in the valid range.
- 2) Create socket and set socket options.
- 3) Create the new connection by using server IP address and server port number.
- 4) Allocate 128 Kbyte memory to be send buffer.
- 5) Generate increment test pattern to send buffer when test pattern is enabled. Skip this step if dummy pattern is selected.
- 6) Send data out and read total sent data from the function.
- 7) Calculate remaining transfer size.
- 8) Print total transfer size every second.
- 9) Run step 5) – 7) in the loop until the remaining transfer size is 0.
- 10) Calculate total performance and print the result on the console.
- 11) Close socket.

Receive data mode

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

- 1) Follow the same step as Transmit data mode in step 1) – 3).
- 2) Allocate 128 Kbyte memory to be received buffer.
- 3) Read data from received buffer and increase total received size.
- 4) 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.
- 5) Print total transfer size every second.
- 6) Run step 3) – 5) in the loop until the connection is closed.
- 7) Calculate total performance and print the result on the console.
- 8) Close socket.

4.2 “tcp_client_txrx” for full duplex test



```
pahol@localhost:~/Downloads/tcp_client_txrx
File Edit View Search Terminal Help
[pahol@localhost tcp_client_txrx]$ ./tcp_client_txrx
[ER] The parameter is incomplete.

* ****
* TCP Tx Rx 10G (version: 1.0)
* ****

./tcp_client_txrx [ServerIP] [ServerPort] [ByteLen] [Verification]

[ServerIP]      Server IP address.
[ServerPort]    Server port number (0-65535).
[ByteLen]       Transfer length (Byte).
[Verification]  Disable/Enable data verification.
                0      Disable
                1      Enable

[EXAMPLE] ./tcp_client_txrx 192.168.40.42 60000 137438953440 0
[pahol@localhost tcp_client_txrx]$
```

Figure 4-2 “tcp_client_txrx” application usage

“tcp_client_txrx” is designed to send and receive TCP data at the same time through Ethernet by using same port number. The application is run in client mode and the user needs to input the network parameters of the server (TOE40G-IP) when running the application. 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) Get parameters from the user and verify the input that is in the valid range.
- (2) Create socket and set socket options.
- (3) Create the new connection by using server IP address and server port number.
- (4) Allocate 98 KB memory for send and receive buffer.
- (5) Create thread to write and read data and wait until both threads finishing the operation.
 - a. In the thread to send data, it generates the increment test pattern when verification is enabled or skips to fill the pattern when verification is disabled. After that, the thread sends data out.
 - b. In the thread to read/verify data, the buffer is read and then the remaining data is calculated. The read data is verified by the increment pattern when verification is enabled. The error message is printed out when data is not correct. Otherwise, the verification is skipped.
- (6) Print total transfer size of both directions every second.
- (7) Run step 5) – 6) until total sending data and total receiving data are equal to ByteLen (input from user).
- (8) Print total size and performance and close socket.
- (9) Sleep for 1 second to wait the hardware complete the current test loop.
- (10) Run step 3) – 10) in forever loop. If verification is failed, the application will quit.

5 Revision History

Revision	Date	Description
1.0	4-Jul-19	Initial version release