NVMe-IP for Gen4 reference design manual

Rev2.0 15-Aug-23

NVMe-IP for Gen4 reference design manual

Rev2.0 15-Aug-23

1 NVMe

NVM Express (NVMe) defines the interface for the host controller to access solid state drive (SSD) by PCI Express. NVM Express optimizes the process to issue command and completion by using only two registers - Command issue and Command completion. Also, NVMe supports parallel operation by supporting up to 64K commands within single queue. 64K command entries improve transfer performance for both sequential and random access.

In PCIe SSD market, two standards are used - AHCI and NVMe. AHCI is the older standard to provide the interface for SATA hard disk drive while NVMe is optimized for non-volatile memory like SSD. The comparison between both AHCI and NVMe protocol in more details is described in "A Comparison of NVMe and AHCI" document.
https://sata-io.org/system/files/member-downloads/NVMe%20and%20AHCI_%20_long_.pdf

The example of NVMe storage device is shown in https://nvmexpress.org/compliance/.

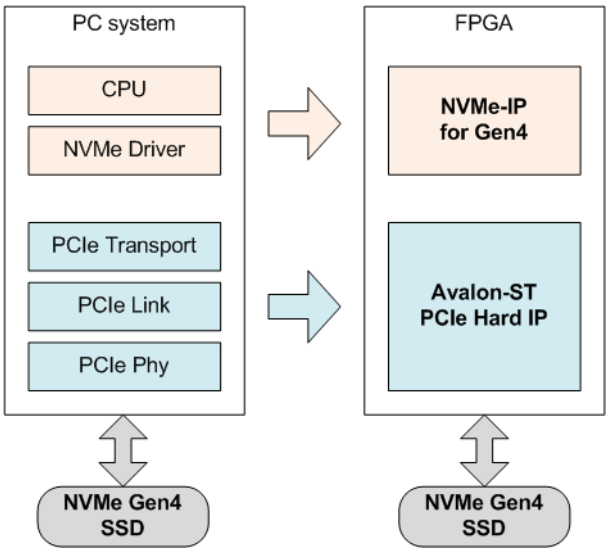

Figure 1-1 NVMe protocol layer

To access NVMe Gen4 SSD, the general system implements NVMe driver running on the processor, as shown in the left side of Figure 1-1. The physical connection of NVMe standard is PCIe connector which is one-to-one type, so one PCIe host connects to one PCIe device when PCIe switch is not integrated. NVMe-IP implements NVMe driver to access NVMe SSD by using pure-hardware logic. The user can access NVMe SSD without including any processor and driver but using NVMe-IP in FPGA board. Using pure-hardware logic for NVMe host controller reduces the overhead time for software-hardware handshake. Therefore, NVMe-IP achieves very high performance for writing and reading with NVMe SSD.

2 Hardware overview

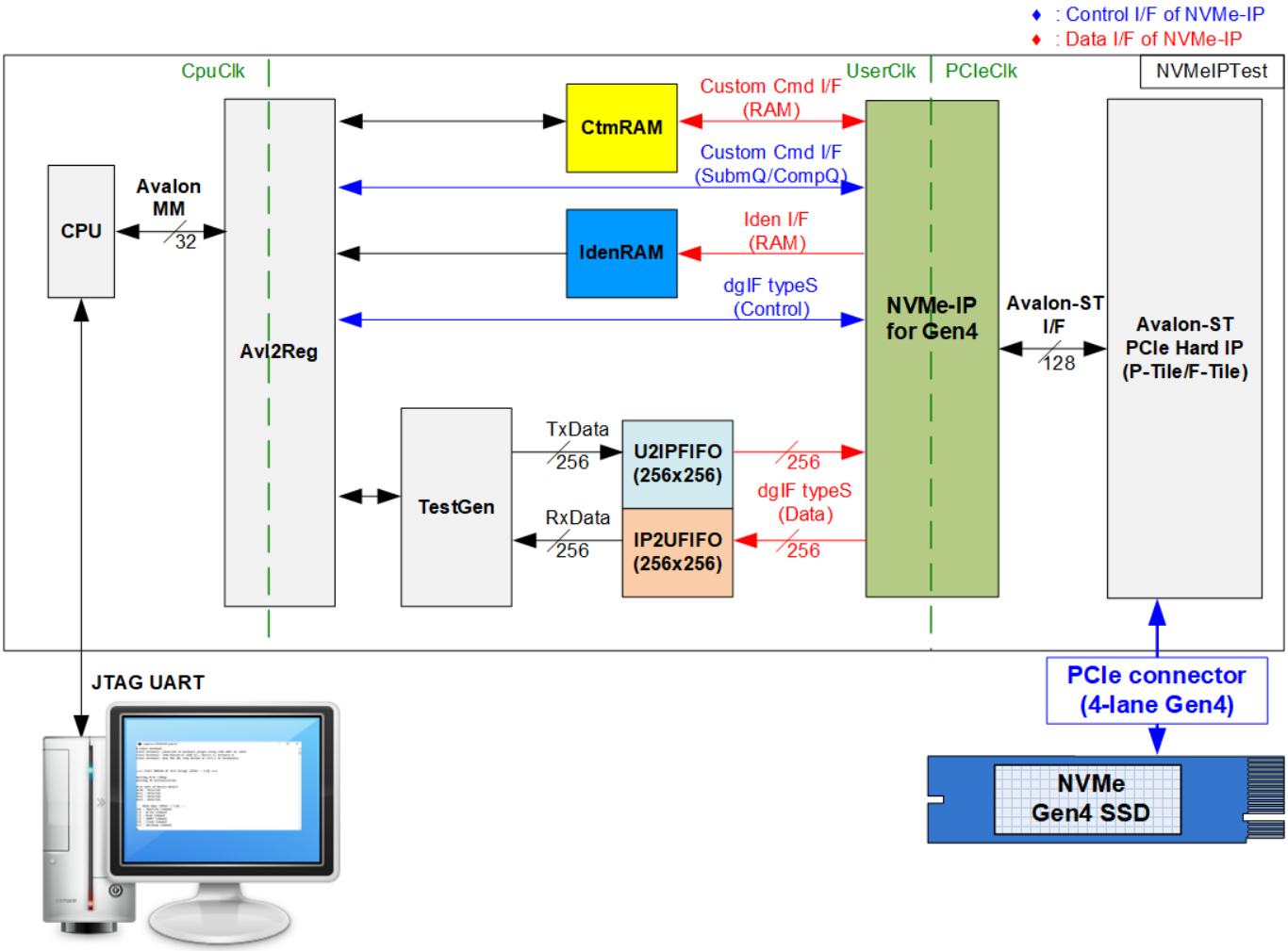

Figure 2-1 NVMe-IP for Gen4 demo hardware

Refer to the function of each module, all hardware modules inside the test system are divided to three groups, i.e., test function (TestGen), NVMe function (CtmRAM, IdenRAM, U2IPFIFO, IP2UFIFO, NVMe-IP for Gen4, and PCIe block), and CPU system (CPU and Avl2Reg).

TestGen is the test logic to generate test data stream for NVMe-IP (NVMe-IP for Gen4) via U2IPFIFO and read data stream output from NVMe-IP via IP2UFIFO for verification. NVMe includes the NVMe-IP and the PCIe hard IP (P-Tile/F-Tile) for accessing NVMe Gen4 SSD directly without PCIe switch. CPU and Avl2Reg are designed to interface with user via JTAG UART. User can set command and the test parameters on the console. Also, the current status of the test hardware is monitored by user on the console. The CPU firmware is implemented to control the flow for operating each command.

The data interface of NVMe-IP connects with four memory blocks, i.e., CtmRAM, IdenRAM, U2IPFIFO, and IP2UFIFO for storing the data from each command in each user. CtmRAM stores returned data from SMART command while IdenRAM stores returned data from Identify command. U2IPFIFO stores data of Write command while IP2UFIFO stores data of Read command. TestGen always writes data with U2IPFIFO or reads data with IP2UFIFO when the FIFO is ready. Thus, the data is always ready for transferring with NVMe-IP to check the best transfer performance.

There are three clock domains displayed in Figure 2-1, i.e., CpuClk, UserClk, and PCIeClk. CpuClk is the clock domain of CPU system and its peripherals. This clock must be stable clock and can be different clock domain from other hardwares. UserClk is the user clock domain for running the user interface of NVMe-IP, RAM, FIFO, and TestGen. According to NVMe-IP datasheet, clock frequency of UserClk must be more than or equal to a half of PCIeClk frequency. This reference design uses 350 MHz for PCIe Gen4 speed. PCIeClk is the clock output from PCIe hard IP to synchronous with data stream of 128-bit AvalonST interface. When the PCIe hard IP is configured to 4-lane PCIe Gen4 at 128-bit data bus, PCIeClk frequency is equal to 500 MHz.

Note: Using the lower frequency for PCIeClk is possible, it limits the maximum bandwidth on the user interface of PCIe hard IP to be lower than PCIe Gen4 performance.

More details of the hardware are described as follows.

2.1 TestGen

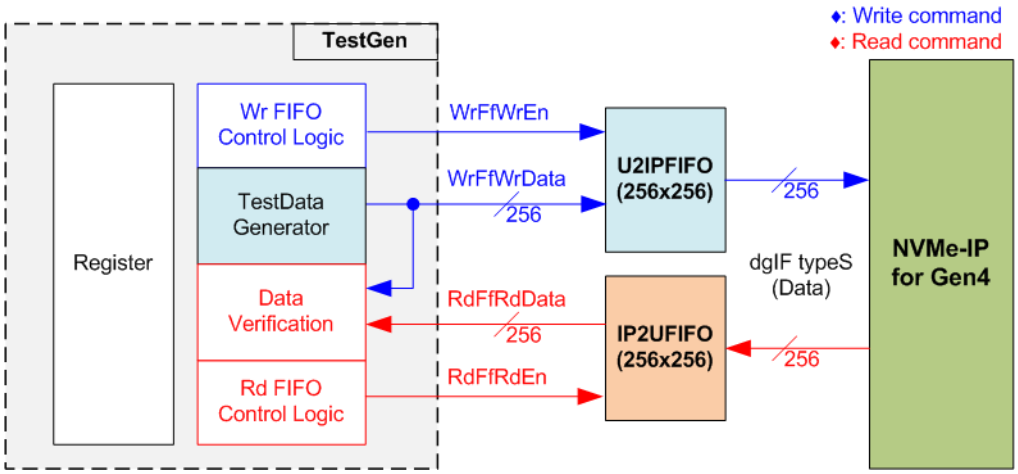

Figure 2-2 TestGen interface

TestGen module handles the data interface of NVMe-IP for transferring the data in Write and Read command. In Write command, TestGen sends 256-bit test data to NVMe-IP via U2IPFIFO. In Read command, the test data is read from IP2UFIFO and then compared with the expected value to verify the data. TestGen is the example of user logic which uses 256-bit data bus size and runs on UserClk domain. To show the best performance, the control logic of Wr FIFO and Rd FIFO are always transferred data with two FIFOs when the FIFOs are ready. Therefore, U2IPFIFO always has the data for Write command and IP2UFIFO always has free area enough for Read command.

For flexible test environment, some test parameters can be set by user to control TestGen module, i.e., total transfer size, transfer direction, verification enable, and test pattern selector. The test parameters are stored in Register block. The details of hardware logic of TestGen are shown in Figure 2-3.

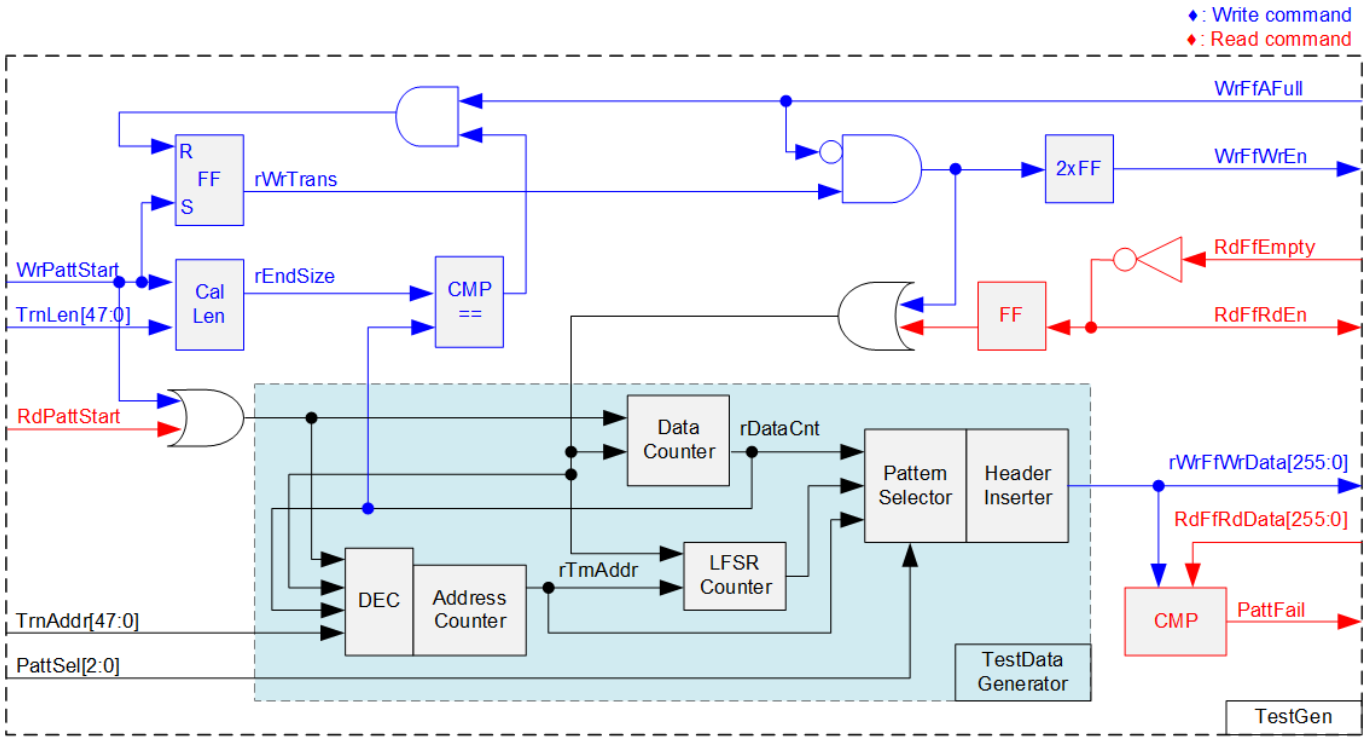

Figure 2-3 TestGen hardware

As shown in the right side of Figure 2-3, flow control signal to Write FIFO in Write command is WrFfAFull while flow control to Read FIFO in Read command is RdFfEmpty. In Write command, when FIFO is almost full (WrFfAFull='1'), WrFfWrEn is de-asserted to '0' to pause data sending to FIFO. In Read command, when FIFO has data (RdFfEmpty='0'), the logic asserts RdFfRdEn to '1' to read the data and then compare with the expected data.

The logics in the left side of Figure 2-3 are the register modules to get total transfer size (TrnLen), start address (TrnAddr), and test pattern selector (PattSel) which are set by user. Inside TestData Generator, there is the data counter to count the amount of transferred data (rDataCnt). When total data count is equal to the end size (rEndSize), write enable or read enable of FIFO is de-asserted to '0'.

Remaining module inside TestData Generator is the logic for generating test data to FIFO or verifying data from FIFO. There are five test patterns - all zero, all one, 32-bit incremental data, 32-bit decremental data, and LFSR. All zero pattern and all one pattern are designed by using constant value. While other patterns are designed by separating the data into two parts – 64-bit header and 504-byte test data for creating unique test data in every 512-byte data, as shown in Figure 2-4.

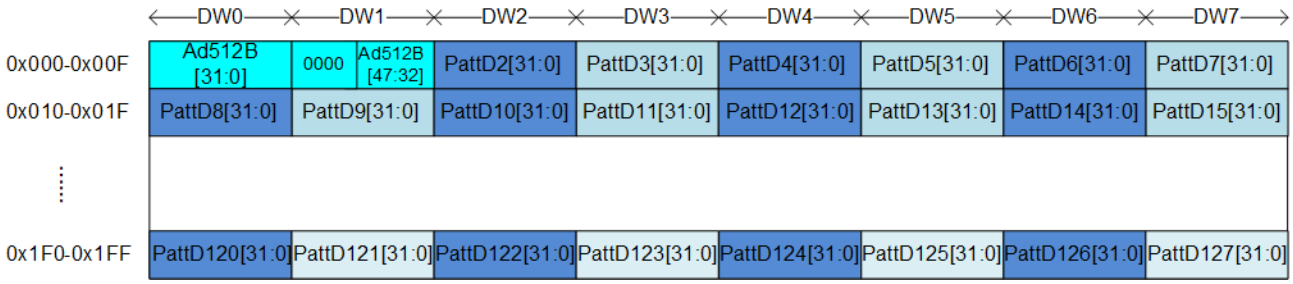

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

64-bit header is assigned in Dword#0 and Dword#1 while the test data is assigned in remaining words (Dword#2 – Dword#127). The header is created by using the address in 512-byte unit (rTrnAddr), output from the Address counter. The address counter loads the start value from user (TrnAddr) and then increases its value after finishing 512-byte data transferring. While three different test patterns are designed by using different counter. 32-bit incremental data is designed by using a part of 52-bit counter. The decremental data uses NOT logic of the incremental data. The LFSR pattern uses LFSR counter which has the equation: x^31 + x^21 + x + 1.

To implement 256-bit LFSR pattern, the data is split to be two sets of 128-bit data with assigning different initial value. Each 128-bit data uses look-ahead technique to calculate four 32-bit LFSR data in one clock cycle. As shown Figure 2-5, the initial value of LFSR is designed by mixing a part of 32-bit LBA address (LBAAddr) with a part of NOT logic of 32-bit LBA address (LBAAddrB).

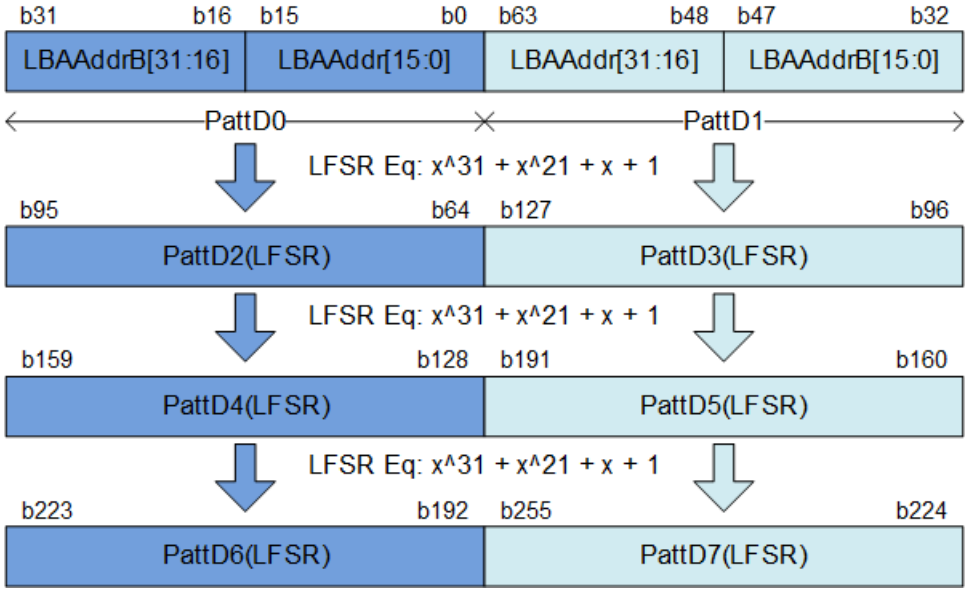

Figure 2-5 256-bit LFSR Pattern in TestGen

Test data is fed to be write data to the FIFO (rWrFfWrData) in Write command. Also, it is applied to be the expected data for verifying with the read data from FIFO (RdFfRdData). If verification is failed, fail flag (PattFail) is asserted to '1'.

The timing diagram when running Write command is shown as follows.

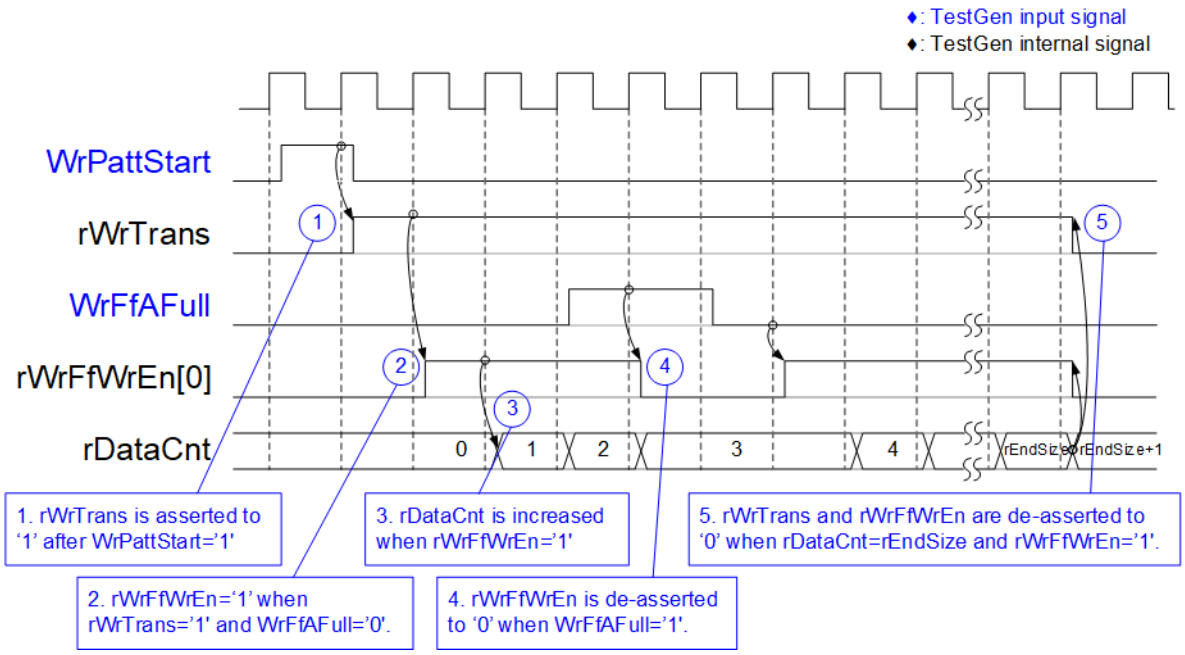

Figure 2-6 Timing diagram of Write command in TestGen

1) WrPattStart is asserted to '1' for one clock cycle when user starts Write command. In the next clock, rWrTrans is asserted to '1' to enable the control logic for asserting Write enable and generating Write data to FIFO.
2) Write enable to FIFO (rWrFfWrEn) is asserted to '1' when two conditions are met. First, rWrTrans must be asserted to '1' (the write operation is active). Second, the FIFO must not be full by monitoring WrFfAFull='0'.
3) The write enable is applied to enable signal for counting total amount of data (rDataCnt) in the write operation.
4) If FIFO is almost full (WrFfAFull='1'), the write process is paused by de-asserting rWrFfWrEn to '0'.
5) When total data count is equal to the total transfer size (rEndSize), rWrTrans is de-asserted to '0'. At the same time, rWrFfWrEn is also de-asserted to '0' to finish data generating.

In Read command, the logic is simpler. Read enable of FIFO is controlled by empty flag of FIFO. Data is read when FIFO has the data. The read operation does not have start/stop for controlling the operating time. Therefore, the empty flag of FIFO must not de-asserted to '0' while running other commands. When the read enable is asserted to '1', the data counter and the address counter are increased for counting total amount of data.

2.2 NVMe

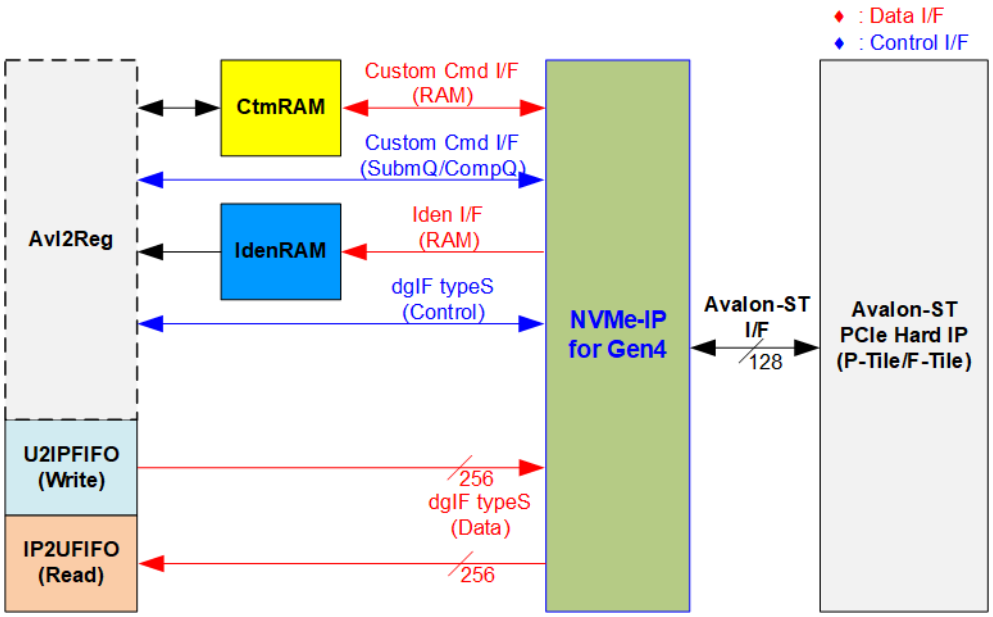

Figure 2-7 NVMe hardware

Figure 2-7 shows the example to interface NVMe-IP in the reference design. The user interface of NVMe-IP consists of control interface and data interface. The control interface receives the command and the parameters from Custom command interface or dgIF typeS, depending on the command value. Custom command interface is used when operating SMART command or Flush command.

The data interface of NVMe-IP has four interfaces, i.e., Custom command RAM interface, Identify interface, FIFO input interface (dgIF typeS), and FIFO output interface (dgIF typeS). Data bus width of all interfaces is 256-bit. The Custom command RAM interface is bi-directional interface while the other interfaces are unidirectional interface. In the reference design, the Custom command RAM interface is used for transferring one direction only to store SMART data transferred from NVMe-IP to Avl2Reg.

2.2.1 NVMe-IP for Gen4

NVMe-IP implements NVMe protocol of the host side to access one NVMe SSD directly without PCIe switch connection. The NVMe-IP supports six commands, i.e., Write, Read, Identify, Shutdown, SMART, and Flush. NVMe-IP can connect to the PCIe Hard IP (P-Tile/F-Tile) directly. More details of NVMe-IP are described in datasheet.
https://dgway.com/products/IP/NVMe-IP/dg_nvme_datasheet_g4_intel.pdf

2.2.2 PCIe Hard IP (P-Tile/F-Tile Avalon-ST Intel FPGA for PCIe)

This block is the hard IP in Intel FPGA device which implements Physical, Data Link, and Transaction Layers of PCIe protocol. More details are described in Intel FPGA document.

P-Tile Avalon-ST Intel FPGA for PCIe
https://www.intel.com/content/www/us/en/docs/programmable/683059/
F-Tile Avalon-ST Intel FPGA for PCIe
https://www.intel.com/content/www/us/en/docs/programmable/683140/

2.2.3 Two-port RAM

Two of two-Port RAMs, CtmRAM and IdenRAM, store data from Identify command and SMART command, respectively. IdenRAM is simple dual-port RAM which has one read port and one write port. The data size of Identify command is 8 Kbytes, so IdenRAM size is 8Kbyte. NVMe-IP and Avl2Reg have different data bus size, 256-bit on NVMe-IP but 32-bit on Avl2Reg. Therefore, IdenRAM has the different bus size on Write interface and Read interface. Besides, NVMe-IP has double word enable to write only 32-bit data in some cases. The RAM setting on IP catalog of Quartus supports the write byte enable. Therefore, one bit of double word enable is extended to be 4-bit write byte enable as shown in Figure 2-8.

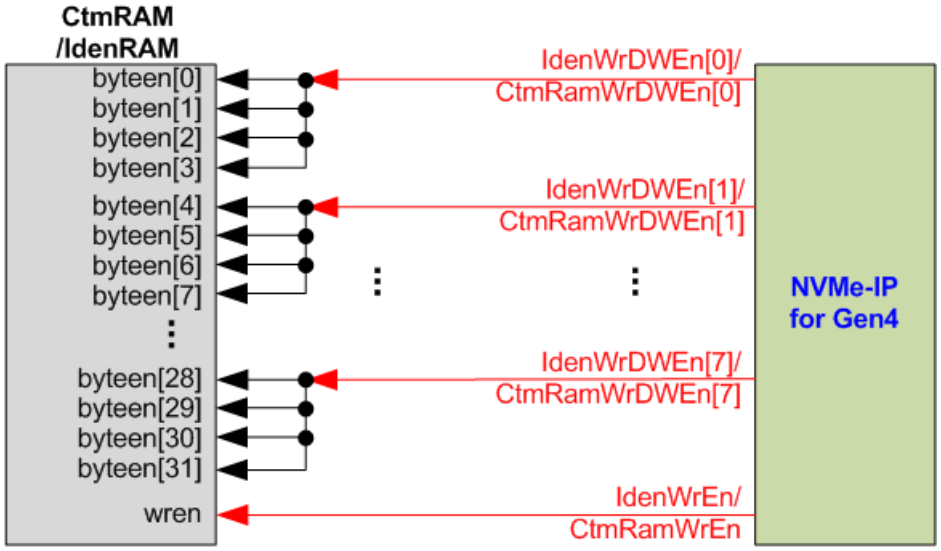

Figure 2-8 Word enable to be byte write enable connection

Bit[0], [1], …, [7] of WrDWEn is fed to bit[3:0], [7:4], …, [31:28] of IdenRAM byte write enable, respectively.

Comparing with IdenRAM, CtmRAM is implemented by two-Port RAM (two read ports and two write ports) with byte write enable. The connection to convert from word enable of NVMe-IP to byte enable of CtmRAM is similar to IdenRAM. Two-Port RAM is used to support the additional features when the customized custom command needs the data input. To support SMART command, using simple dual port RAM is enough. Though the data size returned from SMART command is 512 bytes, CtmRAM is implemented by 8Kbyte RAM for customized custom command.

2.3 CPU and Peripherals

32-bit Avalon-MM bus is applied to be the bus interface for CPU accessing the peripherals such as Timer and JTAG UART. CPU assigns the different base address and the address range to each peripheral for accessing one peripheral at a time. The test system of NVMe-IP is connected with CPU as a peripheral on 32-bit Avalon-MM bus for CPU controlling and monitoring.

In the reference design, Avl2Reg module is designed to connect the CPU system via Avalon-MM bus standard. CPU specifies the base address and the range to write/read access with Avl2Reg. More details of Avl2Reg are shown in Figure 2-9.

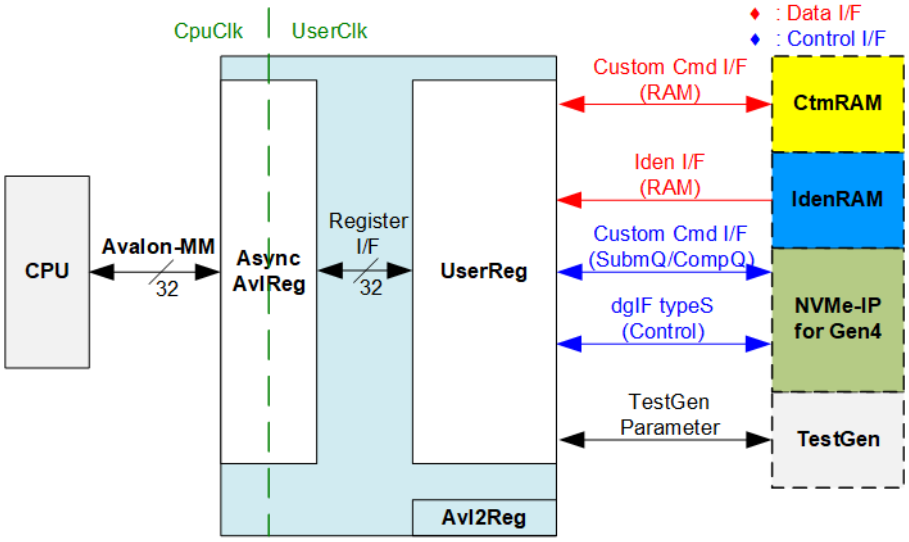

Figure 2-9 CPU and peripherals hardware

Avl2Reg consists of AsyncAvlReg and UserReg. AsyncAvlReg is designed to convert the Avalon-MM signals to be the simple register interface which has 32-bit data bus size, similar to Avalon-MM data bus size. In addition, AsyncAvlReg includes asynchronous logic to support clock domain crossing between CpuClk domain and UserClk domain.

UserReg includes the register file for setting the parameters and storing the status signals of other modules in the test system, i.e., CtmRAM, IdenRAM, NVMe-IP, and TestGen. More details of AsyncAvlReg and UserReg are described as follows.

2.3.1 AsyncAvlReg

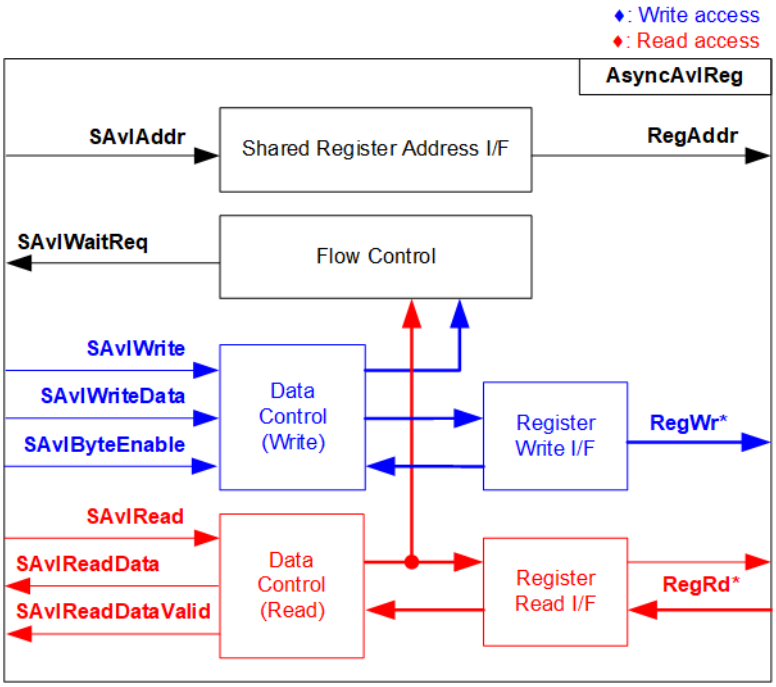

Figure 2-10 AsyncAvlReg Interface

The signal on Avalon-MM bus interface can be split into three groups, i.e., Write channel (blue color), Read channel (red color), and Shared control channel (black color). More details of Avalon-MM interface specification are described in following document.
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf

According to Avalon-MM specification, one command (write or read) can be operated at a time. The logics inside AsyncAvlReg are split into three groups, i.e., Write control logic, Read control logic, and Flow control logic. Flow control logic controls SAvlWaitReq to hold the next request from Avalon-MM interface if the current request does not finish. Write control and Write data I/F of Avalon-MM bus are latched and transferred to be Write register interface with clock domain crossing registers. Similarly, Read control I/F are latched and transferred to be Read register interface while the Read data is returned from Register Read I/F to Avalon-MM bus by using clock domain crossing registers. Address I/F of Avalon-MM is latched and transferred to Address register interface as well.

The simple register interface is compatible with single-port RAM interface for write transaction. The read transaction of the register interface is slightly modified from RAM interface by adding RdReq and RdValid signals for controlling read latency time. The address of register interface is shared for write and read transaction, so user cannot write and read the register at the same time. The timing diagram of the register interface is shown in Figure 2-11.

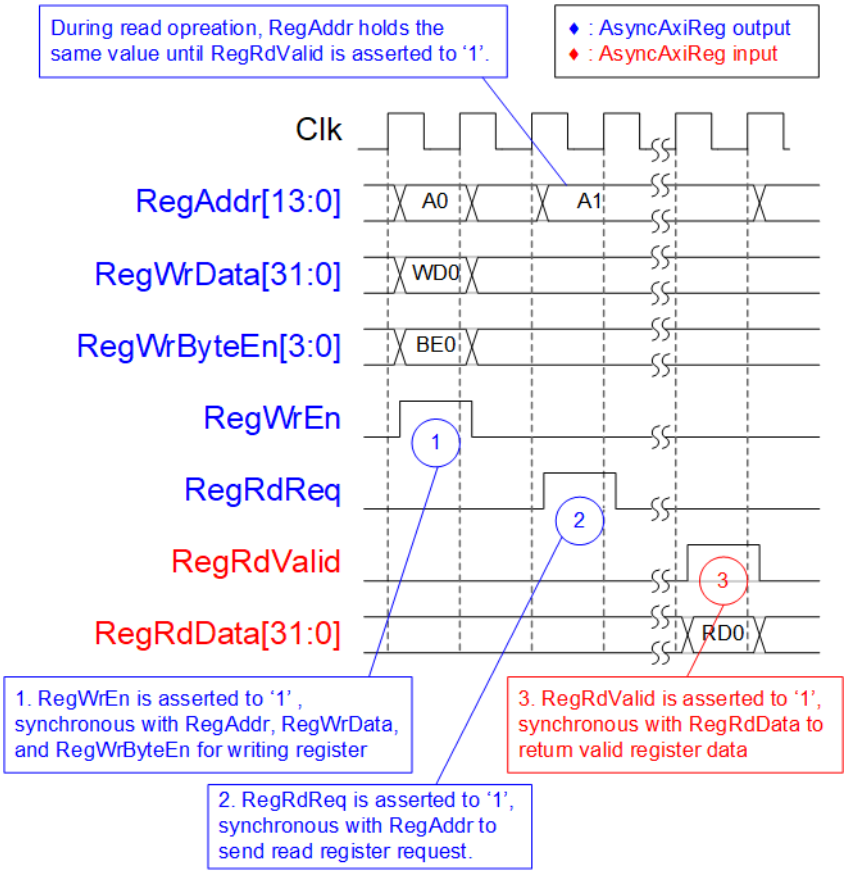

Figure 2-11 Register interface timing diagram

1) To write register, the timing diagram is similar to single-port RAM interface. RegWrEn is asserted to '1' with the valid signal of RegAddr (Register address in 32-bit unit), RegWrData (write data of the register), and RegWrByteEn (the write byte enable). Byte enable has four bits to be the byte data valid. Bit[0], [1], [2], and [3] are equal to '1' when RegWrData[7:0], [15:8], [23:16], and [31:24] are valid respectively.
2) To read register, AsyncAvlReg asserts RegRdReq to '1' with the valid value of RegAddr. 32-bit data must be returned after receiving the read request. The slave must monitor RegRdReq to start the read transaction. In read operation, the address value (RegAddr) does not change until RegRdValid is asserted to '1'. Therefore, the address can be used for selecting the returned data by using multiple layers of multiplexer.
3) The read data is returned on RegRdData bus by the slave with asserting RegRdValid to '1'. After that, AsyncAvlReg forwards the read value to SAvlRead interface.

2.3.2 UserReg

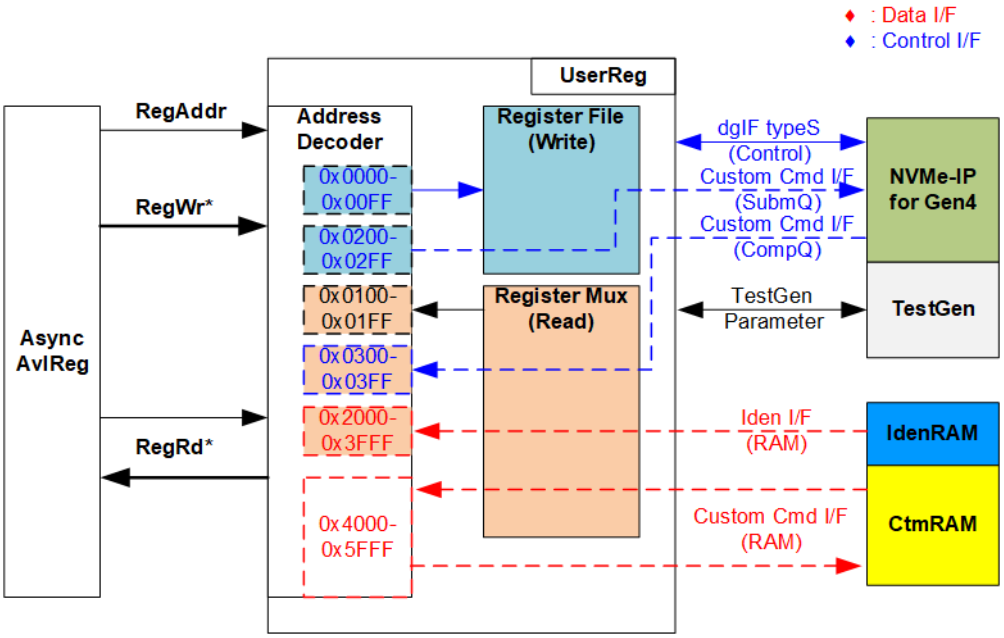

Figure 2-12 UserReg Interface

The address range to map to UserReg is split into six areas, as shown in Figure 2-12.

1) 0x0000 – 0x00FF: mapped to set the command with the parameters of NVMe-IP and TestGen. This area is write-access only.
2) 0x0200 – 0x02FF: mapped to set the parameters for Custom command interface of NVMe-IP. This area is write-access only.
3) 0x0100 – 0x01FF: mapped to read the status signals of NVMe-IP and TestGen. This area is read-access only.
4) 0x0300 – 0x03FF: mapped to read the status of Custom command interface (NVMe-IP). This area is read-access only.
5) 0x2000 – 0x3FFF: mapped to read data from IdenRAM. This area is read-access only.
6) 0x4000 – 0x5FFF: mapped to write or read data with Custom command RAM interface. This area supports write-access and read-access. The demo shows only read-access by running SMART command.

Address decoder decodes the upper bit of RegAddr for selecting the active hardware. The register file inside UserReg is 32-bit bus size. Therefore, write byte enable (RegWrByteEn) is not applied in the test system and the CPU uses 32-bit pointer to set the hardware register.

To read register, two-step multiplexer selects the data to return to CPU by using the address. The lower bit of RegAddr is fed to the submodule to select the active data from each submodule. While the upper bit is applied in UserReg to select the returned data from each submodule. Totally, the latency time of read data is equal to two clock cycles. Therefore, RegRdValid is created by RegRdReq with asserting two D Flip-flops. More details of the address mapping within UserReg module are shown in Table 2-1.

Table 2-1 Register Map

Address Wr/Rd	Register Name (Label in the “nvmeiptest.c”)	Description
0x0000 – 0x00FF: Control signals of NVMe-IP and TestGen (Write access only)		
BA+0x0000	User Address (Low) Reg (USRADRL_INTREG)	[31:0]: Input to be bit[31:0] of start address as 512-byte unit (UserAddr[31:0] of dgIF typeS)
BA+0x0004	User Address (High) Reg (USRADRH_INTREG)	[15:0]: Input to be bit[47:32] of start address as 512-byte unit (UserAddr[47:32] of dgIF typeS)
BA+0x0008	User Length (Low) Reg (USRLENL_INTREG)	[31:0]: Input to be bit[31:0] of transfer length as 512-byte unit (UserLen[31:0] of dgIF typeS)
BA+0x000C	User Length (High) Reg (USRLENH_INTREG)	[15:0]: Input to be bit[47:32] of transfer length as 512-byte unit (UserLen[47:32] of dgIF typeS)
BA+0x0010	User Command Reg (USRCMD_INTREG)	[2:0]: Input to be user command (UserCmd of dgIF typeS for NVMe-IP) 000b: Identify, 001b: Shutdown, 010b: Write SSD, 011b: Read SSD, 100b: SMART, 110b: Flush, 101b/111b: Reserved When this register is written, the command request is sent to NVMe-IP. After that, the IP starts operating the command.
BA+0x0014	Test Pattern Reg (PATTSEL_INTREG)	[2:0]: Select test pattern 000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1, 100b-LFSR
BA+0x0020	NVMe Timeout Reg (NVMTIMEOUT_INTREG)	[31:0]: Mapped to TimeOutSet[31:0] of NVMe-IP
0x0100 – 0x01FF: Status signals of NVMe-IP and TestGen (Read access only)		
BA+0x0100	User Status Reg (USRSTS_INTREG)	[0]: UserBusy of dgIF typeS (‘0’: Idle, ‘1’: Busy) [1]: UserError of dgIF typeS (‘0’: Normal, ‘1’: Error) [2]: Data verification fail (‘0’: Normal, ‘1’: Error)
BA+0x0104	Total disk size (Low) Reg (LBASIZEL_INTREG)	[31:0]: Mapped to LBASize[31:0] of NVMe-IP
BA+0x0108	Total disk size (High) Reg (LBASIZEH_INTREG)	[15:0]: Mapped to LBASize[47:32] of NVMe-IP [31]: Mapped to LBAMode of NVMe-IP
BA+0x010C	User Error Type Reg (USRERRTYPE_INTREG)	[31:0]: Mapped to UserErrorType[31:0] of NVMe-IP to show error status
BA+0x0110	PCIe Status Reg (PCIESTS_INTREG)	[0]: PCIe linkup status from PCIe hard IP (‘0’: No linkup, ’1’: linkup) [3:2]: Two lower bits to show PCIe link speed of PCIe hard IP. MSB is bit[16]. (000b: Not linkup, 001b: PCIe Gen1, 010b: PCIe Gen2, 011b: PCIe Gen3, 111b”: PCIe Gen4) [6:4]: PCIe link width status from PCIe hard IP (001b: 1-lane, 010b: 2-lane, 100b: 4-lane) [13:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in Avalon-ST PCIe Hard IP datasheet [16]: The upper bit to show PCIe link speed of PCIe hard IP. Two lower bits are bit[3:2].
BA+0x0114	Completion Status Reg (COMPSTS_INTREG)	[15:0]: Mapped to AdmCompStatus[15:0] of NVMe-IP [31:16]: Mapped to IOCompStatus[15:0] of NVMe-IP
BA+0x0118	NVMe CAP Reg (NVMCAP_INTREG)	[31:0]: Mapped to NVMeCAPReg[31:0] of NVMe-IP
BA+0x011C	NVMe IP Test pin Reg (NVMTESTPIN_INTREG)	[31:0]: Mapped to TestPin[31:0] of NVMe-IP

Address Wr/Rd	Register Name (Label in the “nvmeiptest.c”)	Description
0x0100 – 0x01FF: Status signals of NVMe-IP and TestGen (Read access only)		
BA+0x0130 – BA+0x014F	Expected value Word0-7 Reg (EXPPATW0-W7_INTREG)	256-bit of the expected data at the 1st failure data in Read command 0x0130: Bit[31:0], 0x0134[31:0]: Bit[63:32], …, 0x014C[31:0]: Bit[255:224]
BA+0x0150 – BA+0x016F	Read value Word0-7 Reg (RDPATW0-W7_INTREG)	256-bit of the read data at the 1st failure data in Read command 0x0150: Bit[31:0], 0x0154[31:0]: Bit[63:32], …, 0x016C[31:0]: Bit[255:224]
BA+0x0170	Data Failure Address(Low) Reg (RDFAILNOL_INTREG)	[31:0]: Bit[31:0] of the byte address of the 1st failure data in Read command
BA+0x0174	Data Failure Address(High) Reg (RDFAILNOH_INTREG)	[24:0]: Bit[56:32] of the byte address of the 1st failure data in Read command
BA+0x0178	Current test byte (Low) Reg (CURTESTSIZEL_INTREG)	[31:0]: Bit[31:0] of the current test data size in TestGen module
BA+0x017C	Current test byte (High) Reg (CURTESTSIZEH_INTREG)	[24:0]: Bit[56:32] of the current test data size of TestGen module
Other interfaces (Custom command of NVMe-IP, IdenRAM, and Custom RAM)		
BA+0x0200 – BA+0x023F Wr	Custom Submission Queue Reg (CTMSUBMQ_STRUCT)	[31:0]: Submission queue entry of SMART and Flush command. Input to be CtmSubmDW0-DW15 of NVMe-IP. 0x200: DW0, 0x204: DW1, …, 0x23C: DW15
BA+0x0300 – BA+0x030F Rd	Custom Completion Queue Reg (CTMCOMPQ_STRUCT)	[31:0]: CtmCompDW0-DW3 output from NVMe-IP. 0x300: DW0, 0x304: DW1, …, 0x30C: DW3
BA+0x0800 Rd	IP Version Reg (IPVERSION_INTREG)	[31:0]: Mapped to IPVersion[31:0] of NVMe-IP
BA+0x2000 – BA+0x2FFF Rd	Identify Controller Data (IDENCTRL_CHARREG)	4Kbyte Identify controller data structure
BA+0x3000 – BA+0x3FFF Rd	Identify Namespace Data (IDENNAME_CHARREG)	4Kbyte Identify Namespace Data Structure
BA+0x4000 – BA+0x5FFF Wr/Rd	Custom command Ram (CTMRAM_CHARREG)	Connect to 8K byte CtmRAM interface. Used to store 512-byte data output from SMART command.

3 CPU Firmware

3.1 Test firmware (nvmeiptest.c)

After system boot-up, CPU runs following steps to finish the initialization process.

1) CPU initializes JTAG UART and Timer parameters.
2) CPU waits until PCIe connection links up (PCIESTS_INTREG[0]='1').
3) CPU waits until NVMe-IP completes initialization process (USRSTS_INTREG[0]='0'). The error message is displayed and the process stops when some errors are found.
4) CPU displays PCIe link status (the number of PCIe lanes and the PCIe speed) by reading PCIESTS_INTREG[16:2].
5) CPU displays the main menu. There are six menus for running six commands of NVMe-IP, i.e., Identify, Write, Read, SMART, Flush, and Shutdown.

More details of the operation flow in each command are described as follows.

3.1.1 Identify Command

The step to operate Identify command is described as follows.

1) Set USRCMD_INTREG[2:0]=000b to send Identify command request to NVMe-IP. After that, Busy flag (USRSTS_INTREG[0]) changes from '0' to '1'.
2) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

   Bit[0] is de-asserted to '0' when command is completed. After that, the data from Identify command is stored to IdenRAM.
   Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE_INTREG[31:0]. Finally, the process is stopped.

3) After busy flag (USRSTS_INTREG[0]) is de-asserted to '0', CPU displays some information decoded from IdenRAM (IDENCTRL_CHARREG) such as SSD model name and the information from NVMe-IP output such as SSD capacity and LBA unit size (LBASIZEH/L_INTREG).

3.1.2 Write/Read Command

The step to operate Write/Read command is described as follows.

1) Receive start address, transfer length, and test pattern from JTAG UART. If some inputs are invalid, the operation is cancelled.
   Note: If LBA unit size = 4 Kbyte, start address and transfer length must be aligned to 8.
2) Get all inputs and set to USRADRL/H_INTREG, USRLENL/H_INTREG, and PATTSEL_INTREG.
3) Set USRCMD_INTREG[2:0]=010b for Write command or 011b for Read command. After that, the new command request is sent to NVMe-IP for running Write command or Read command. Busy flag (USRSTS_INTREG[0]) changes from '0' to '1'.
4) CPU waits until the operation is completed or some errors (except verification error) are found by monitoring USRSTS_INTREG[2:0].

   Bit[0] is de-asserted to '0' when command is completed.
   Bit[1] is asserted to '1' when some errors are detected. After that, the error message is displayed on the console to show the error details, decoded from USRERRTYPE_INTREG[31:0]. Finally, the process is stopped.
   Bit[2] is asserted to '1' when data verification is failed. The verification error message is displayed on the console to show the error details. In this condition, CPU is still running until the operation is done or user presses any key(s) to cancel operation.

   When the operation does not finish, current transfer size read from CURTESTSIZE_INTREG is displayed every second.

5) After busy flag (USRSTS_INTREG[0]) is de-asserted to '0', CPU calculates and displays the test result on the console, i.e., total time usage, total transfer size, and transfer speed.

3.1.3 SMART Command,

The step to operate SMART command is described as follows.

1) Set 16-Dword of Submission queue entry (CTMSUBMQ_STRUCT) to be SMART command value.
2) Set USRCMD_INTREG[2:0]=100b. Next, Test logic generates command and asserts the request to NVMe-IP. After that, Busy flag (USRSTS_INTREG[0]) changes from '0' to '1'.
3) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

   Bit[0] is de-asserted to '0' when command is completed. After that, the data returned from SMART command is stored to CtmRAM.
   Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE_INTREG[31:0]. Finally, the process is stopped.

4) After busy flag (USRSTS_INTREG[0]) is de-asserted to '0', CPU decodes SMART command information from CtmRAM (CTMRAM_CHARREG), i.e., Remaining Life, Percentage Used, Temperature, Total Data Read, Total Data Written, Power On Cycles, Power On Hours, and Number of Unsafe Shutdown.

   More details of SMART log are described in NVM Express Specification.
   https://nvmexpress.org/resources/specifications/

3.1.4 Flush Command

The step to operate Flush command is described as follows.

1) Set 16-Dword of Submission queue entry (CTMSUBMQ_STRUCT) to be Flush command value.
2) Set USRCMD_INTREG[2:0]=110b. Next, Test logic generates command and asserts the request to NVMe-IP. After that, Busy flag (USRSTS_INTREG[0]) changes from '0' to '1'.
3) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

   Bit[0] is de-asserted to '0' when command is completed. After that, CPU goes back to the main menu.
   Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE_INTREG[31:0]. Finally, the process is stopped.

3.1.5 Shutdown Command

The step to operate Shutdown command is described as follows.

1) Set USRCMD_INTREG[2:0]=001b. Next, Test logic generates command and asserts the request to NVMe-IP. After that, Busy flag (USRSTS_INTREG[0]) changes from '0' to '1'.
2) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

   Bit[0] is de-asserted to '0' when command is completed. After that, the CPU goes to the next step.
   Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE_INTREG[31:0]. Finally, the process is stopped.

3) After busy flag (USRSTS_INTREG[0]) is de-asserted to '0', the SSD and NVMe-IP change to inactive status. The CPU cannot receive the new command from user. The user must power off the test system.

3.2 Function list in Test firmware

int exec_ctm(unsigned int user_cmd)	
Parameters	user_cmd: 4-SMART command, 6-Flush command
Return value	0: No error, -1: Some errors are found in the NVMe-IP
Description	Run SMART command following topic 3.1.3 (SMART Command,) or Flush command following topic 3.1.4 (Flush Command).

unsigned long long get_cursize(void)	
Parameters	None
Return value	Read value of CURTESTSIZEH/L_INTREG
Description	Read CURTESTSIZEH/L_INTREG and return read value as function result.

int get_param(userin_struct* userin)	
Parameters	userin: Three inputs from user, i.e., start address, total length in 512-byte unit, and test pattern
Return value	0: Valid input, -1: Invalid input
Description	Receive the input parameters from the user and verify the value. When the input is invalid, the function returns -1. Otherwise, all inputs are updated to userin parameter.

void iden_dev(void)	
Parameters	None
Return value	None
Description	Run Identify command, following in topic 3.1.1. (Identify Command).

int setctm_flush(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the NVMe-IP
Description	Set Flush command to CTMSUBMQ_STRUCT and call exec_ctm function to run Flush command.

int setctm_smart(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the NVMe-IP
Description	Set SMART command to CTMSUBMQ_STRUCT and call exec_ctm function to run SMART command. Finally, decode and display SMART information on the console

void show_error(void)	
Parameters	None
Return value	None
Description	Read USRERRTYPE_INTREG, decode the error flag, and display error message following the error flag. Also, call show_pciestat function to check the debug signal in the hardware.

void show_pciestat(void)	
Parameters	None
Return value	None
Description	Read PCIESTS_INTREG until the read value from two read times is stable. After that, display the read value on the console. Also, debug signal is read by NVMTESTPIN_INTREG.

void show_result(void)	
Parameters	None
Return value	None
Description	Print total transfer size by calling get_cursize and show_size function. After that, calculate total time usage from global parameters (timer_val and timer_upper_val) and display in usec, msec, or sec unit. Finally, transfer performance is calculated and displayed in MB/s unit.

void show_size(unsigned long long size_input)	
Parameters	size_input: transfer size to display on the console
Return value	None
Description	Calculate and display the input value in MByte, GByte, or TByte unit

void show_smart_hex16byte(volatile unsigned char *char_ptr)	
Parameters	*char_ptr: Pointer of 16-byte SMART data
Return value	None
Description	Display 16-byte SMART data as hexadecimal unit.

void show_smart_int8byte(volatile unsigned char *char_ptr)	
Parameters	*char_ptr: Pointer of 8-byte SMART data
Return value	None
Description	When the input value is less than 4 billion (32-bit), display 8-byte SMART data as decimal unit. Otherwise, display overflow message.

void show_smart_size8byte(volatile unsigned char *char_ptr)	
Parameters	*char_ptr: Pointer of 8-byte SMART data
Return value	None
Description	Display 8-byte SMART data as GB or TB unit. When the input value is more than limit (500 PB), the overflow message is displayed instead.

void show_vererr(void)	
Parameters	None
Return value	None
Description	Read RDFAILNOL/H_INTREG (error byte address), EXPPATW0-W7_INTREG (expected value), and RDPATW0-W7_INTREG (read value) to display verification error details on the console.

void shutdown_dev(void)	
Parameters	None
Return value	None
Description	Run Shutdown command, following in topic 3.1.5. (Shutdown Command)

int wrrd_dev(unsigned int user_cmd)	
Parameters	user_cmd: 2-Write command, 3-Read command
Return value	0: No error, -1: Receive invalid input or some errors are found.
Description	Run Write command or Read command, following in topic 3.1.2. (Write/Read Command). Show_result function is called to calculate and display transfer performance in Write/Read command.

4 Example Test Result

The example test result when running demo system by using 2 TB Addlink S95 is shown in Figure 4-1.

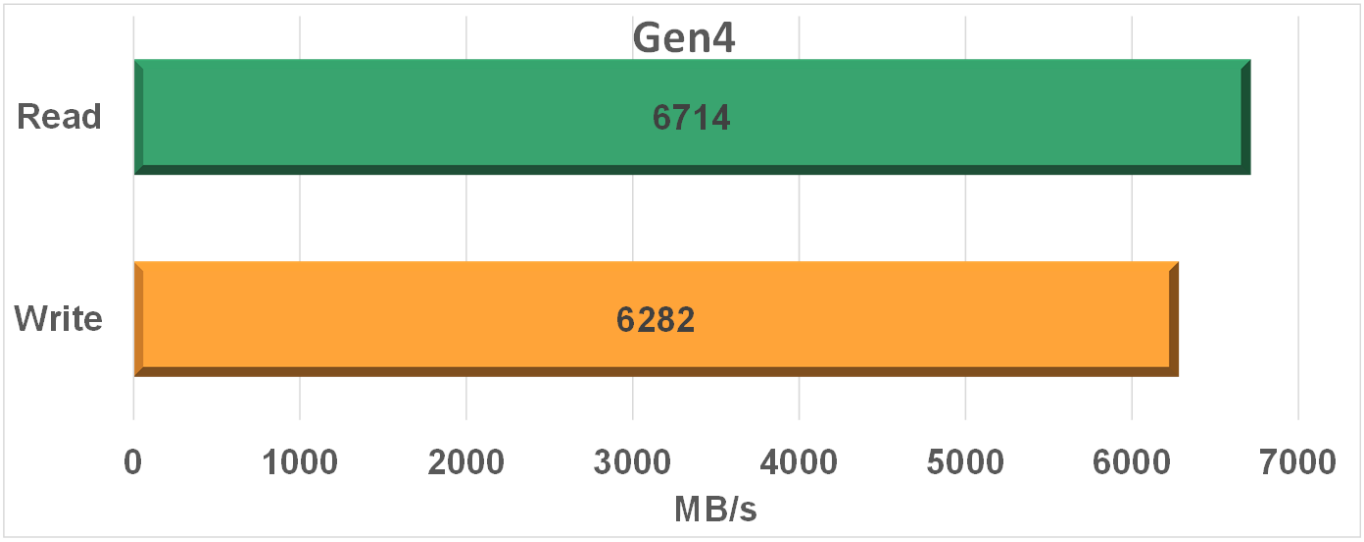

Figure 4-1 Test Performance of NVMe-IP demo by using Addlink S95 SSD

By using PCIe Gen4 on Agilex F-Series board, write performance is about 6200 Mbyte/sec and read performance is about 6700 Mbyte/sec. The test pattern is all zero value and the transfer size is 32 GB.

5 Revision History

Revision	Date	Description
2.0	29-Sep-22	Update IP and firmware
1.0	1-Jun-21	Initial Release