NVMe IP with PCIe Gen4 Soft IP reference design manual

Rev1.2 19-Jul-21

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). Besides, 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, i.e., 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 (SSD). The comparison between 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 http://www.nvmexpress.org/products/

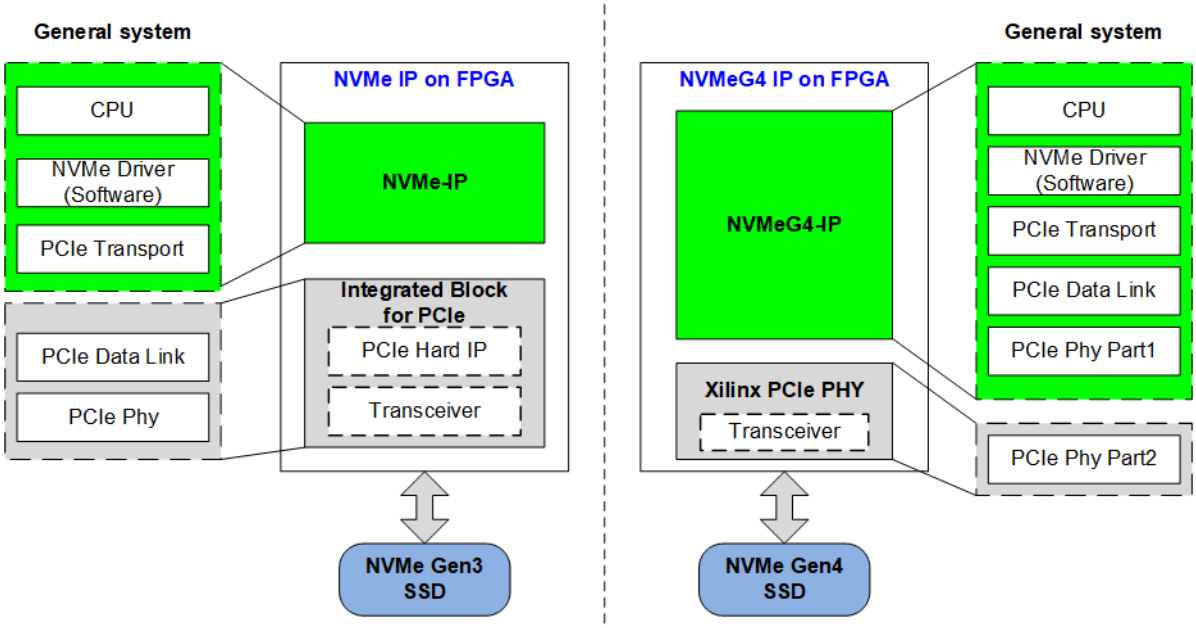

Figure 1-1 NVMe-IP and NVMeG4-IP comparison

DG has the solution by using NVMe IP with Gen3 PCIe hard IP, as shown in the left side of Figure 1-1. Without using CPU and external main memory, the NVMe host controller can be implemented within FPGA with less FPGA resource and achieving ultra-speed write and read performance. However, there are some limitations for the NVMe IP with Gen3 PCIe hard IP.

First, PCIe hard IP in Ultrascale and Ultrascale+ device does not support PCIe Gen4 speed which is higher than Gen3. Second, PCIe hard IP is available in some FPGA models, so some models cannot connect with PCIe SSD by using above solution. Finally, the number of PCIe hard IP in one FPGA is limited which is effect to the maximum number of SSDs connecting to one FPGA.

DG NVMeG4 IP is the latest solution provided by Design Gateway that implements NVMe host IP without using PCIe hard IP to support PCIe Gen 4 speed. The IP includes the lower layer of PCIe protocol, i.e., Data Link Layer and some parts of Physical Layer by using the logic. This feature is known as PCIe soft IP. By integrating PCIe Soft IP and NVMe IP, NVMeG4 IP connecting with Xilinx PCIe PHY is the recent solution for implementing NVMe host in many FPGAs which has the transceivers for running at PCIe Gen4 speed. User interface of NVMeG4 IP is similar to NVMe IP (Gen3) but the data width which are double times from NVMe IP.

2 Hardware overview

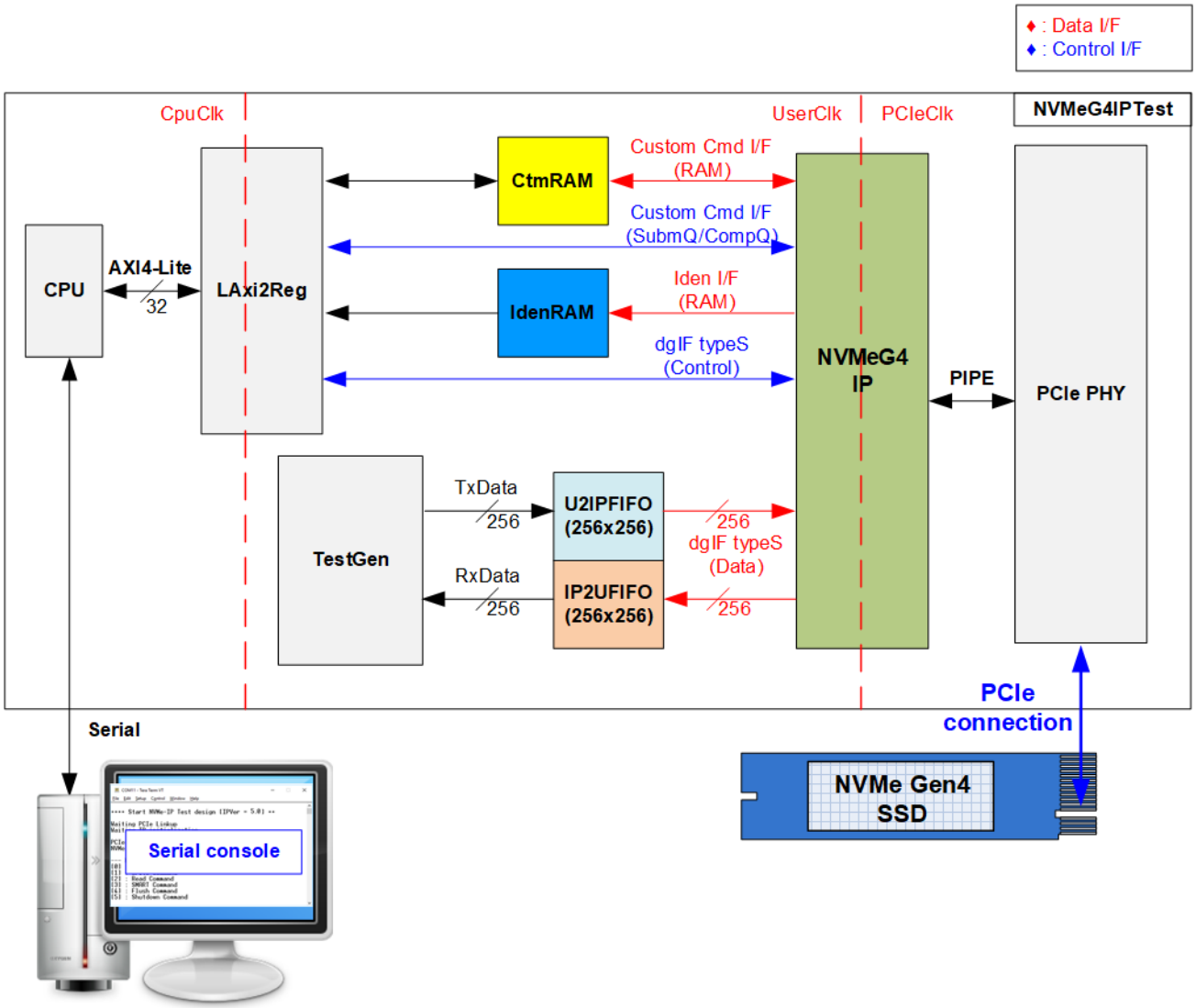

Figure 2-1 NVMeG4 IP demo hardware

Refer to the function, the hardware modules in the test system are divided to three groups, i.e., test function (TestGen), NVMe function (CtmRAM, IdenRAM, U2IPFIFO, IP2UFIFO, NVMeG4-IP, and PCIe PHY), and CPU system (CPU and LAxi2Reg).

TestGen is the example of user module which is designed to generate test data stream for NVMeG4-IP via U2IPFIFO and read/verify data stream output from NVMeG4-IP via IP2UFIFO. NVMe shows the example for connecting one NVMeG4-IP to the PCIe PHY. One NVMe SSD is connected to PCIe PHY directly without PCIe switch. Finally, CPU and LAxi2Reg are the additional design in the demo for user easily setting and monitoring the test system via UART. The console is run on the PC for user setting the command and displaying the test progress and result. Therefore, the CPU firmware implements the example of the control flow for operating each command.

There are four memory blocks in the test system to connect with four data interfaces of NVMeG4-IP - CtmRAM, IdenRAM, U2IPFIFO, and IP2UFIFO. Each memory is applied for running one command. CtmRAM stores returned data from SMART command. IdenRAM stores returned data from Identify command. U2IPFIFO stores transmitted data for Write command. Finally, IP2UFIFO stores the received data for Read command. Another side of U2IPFIFO and IP2UFIFO is connected to TestGen which is always transferred data with both FIFOs to achieve the best write/read performance for accessing NVMe Gen4 SSD.

To emulate the real system that may consists of many clock systems, the demo design uses three clock domains - CpuClk, UserClk, and PCIeClk. CpuClk is the clock domain of CPU system and its peripherals which is the slowest clock frequency. This clock is stable clock and independent from the other hardware interface. UserClk is the example clock domain of user logic for interfacing with NVMeG4-IP. Most logics in the test system are run in this clock domain. According to NVMeG4-IP datasheet, clock frequency of UserClk must be more than or equal to PCIeClk frequency. Therefore, this reference design uses 275 MHz/280 MHz. Finally, PCIeClk is the clock output from PCIe PHY to synchronous with 256-bit data bus on PIPE. When running 4-lane PCIe Gen4, PCIeClk frequency is equal to 250 MHz.

More details of the hardware are described as follows.

2.1 TestGen

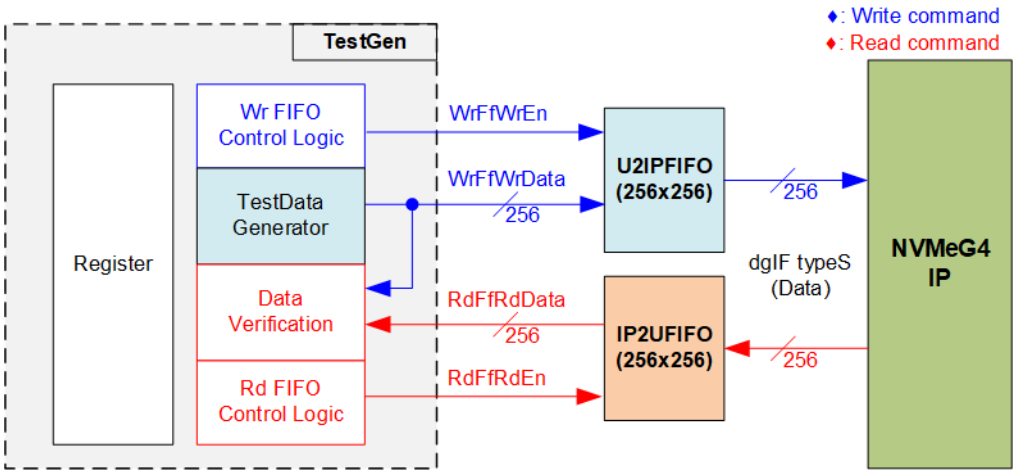

Figure 2-2 TestGen interface

TestGen module handles the data interface of NVMeG4-IP for transferring the data in Write and Read command. In Write command, TestGen sends 256-bit test data to NVMeG4-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.

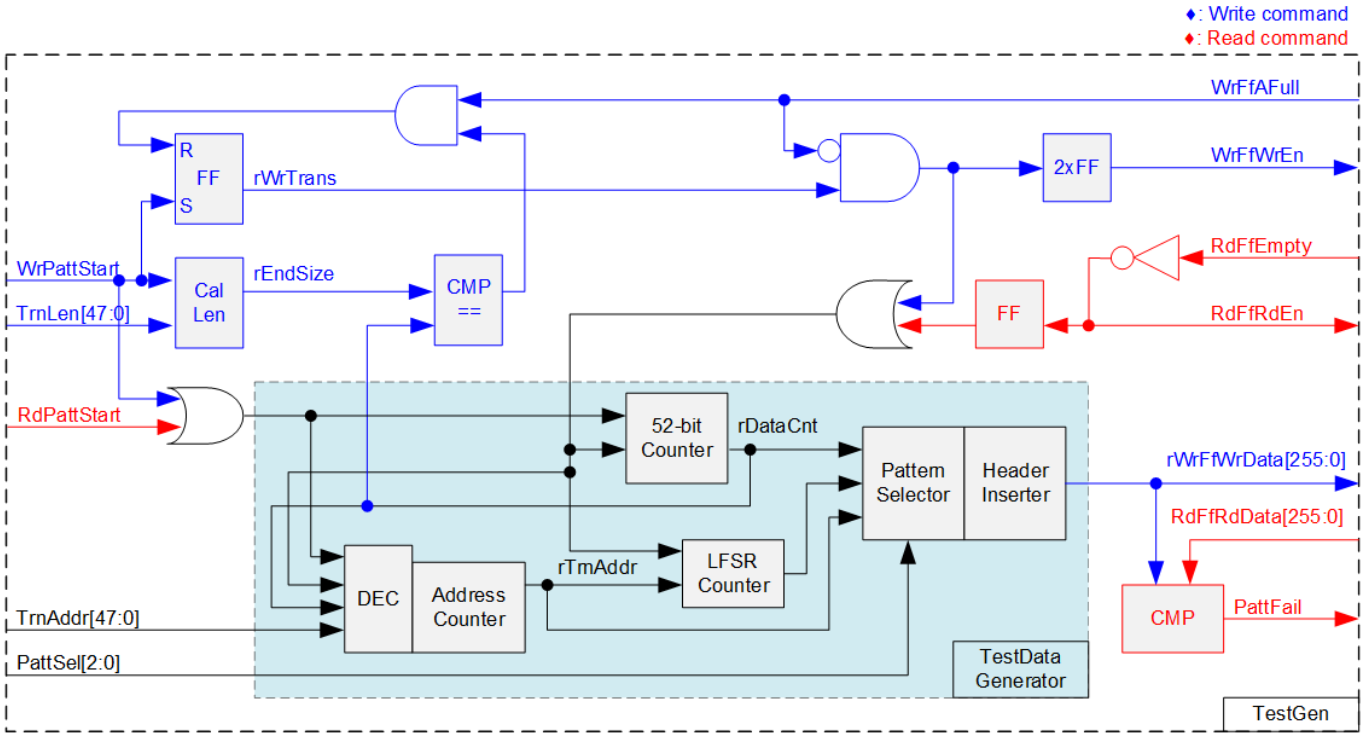

Figure 2-3 TestGen hardware

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 during write operation (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 52-bit counter to count current number of transferred size (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 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.

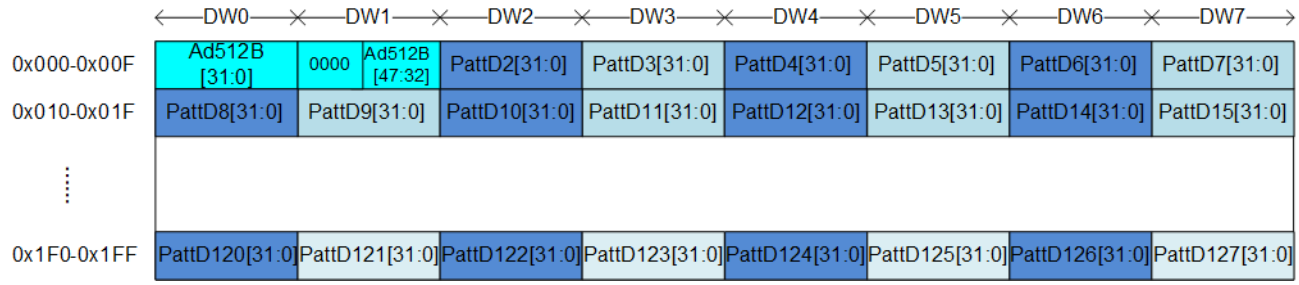

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

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 the value after finishing transferring 512-byte data. 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 to convert the incremental data to be decremental 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).

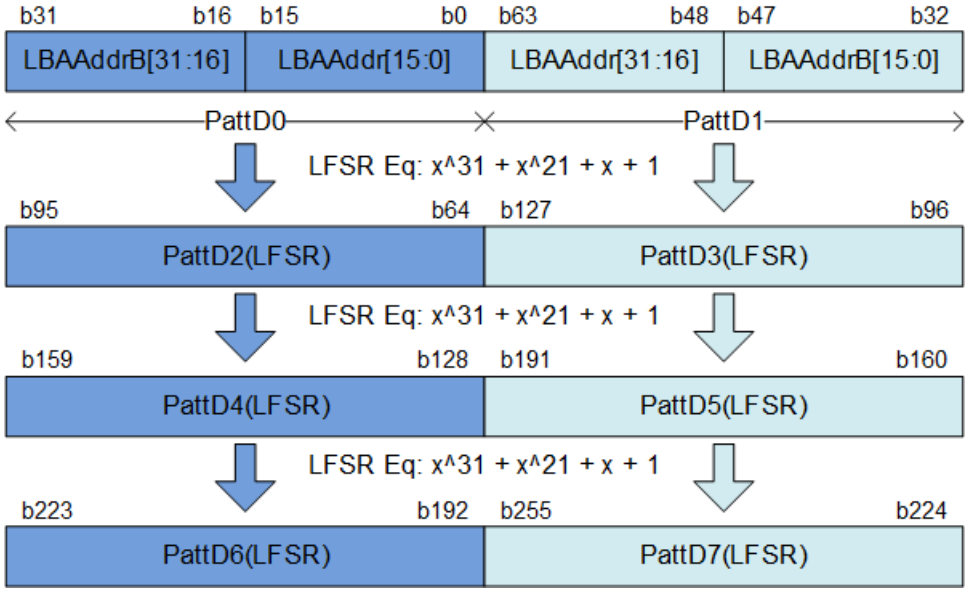

Figure 2-5 256-bit LFSR Pattern in TestGen

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.

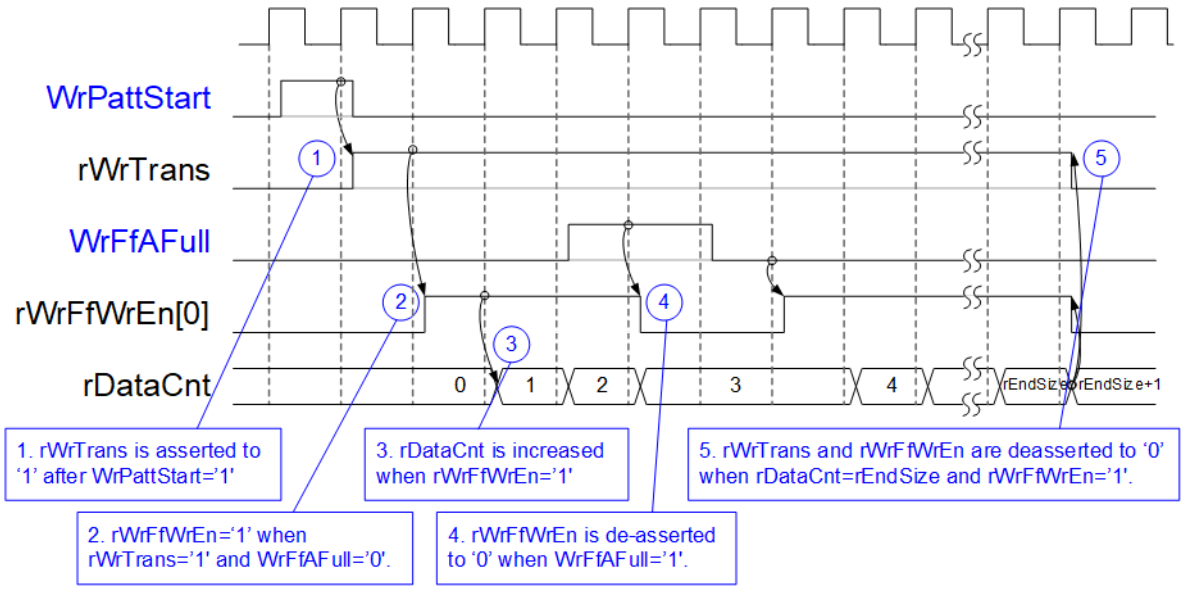

Figure 2-6 Timing diagram of Write command in TestGen

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 generating Write enable and 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 number 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 FIFO must not have the data when 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

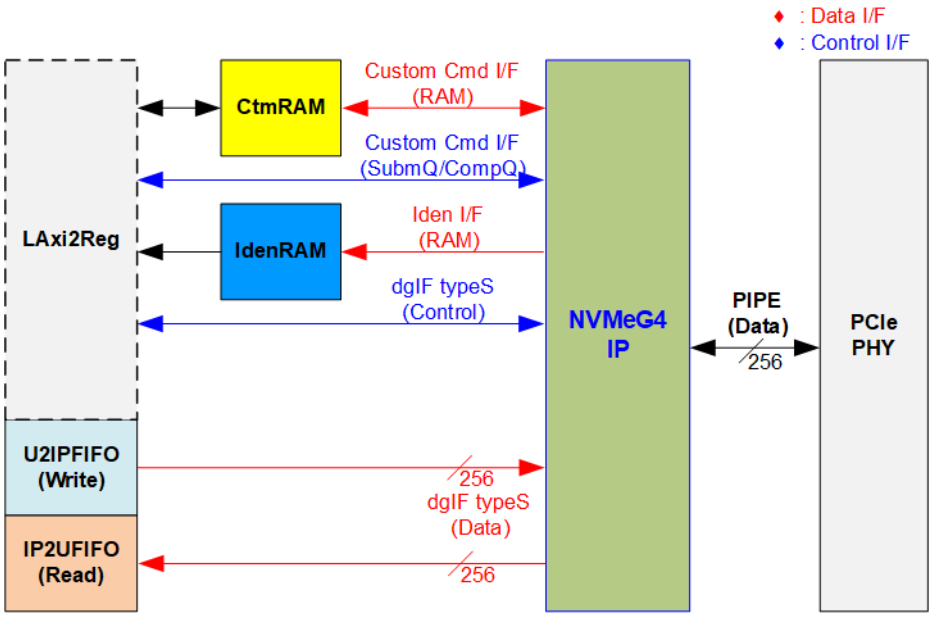

Figure 2-7 NVMe hardware

Figure 2-7 shows the example to interface NVMeG4-IP in the reference design. The user interface of NVMeG4-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. Custom command interface is used when operating SMART command or Flush command.

The data interface of NVMeG4-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 data interface is equal to 256 bits. 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 NVMeG4-IP to LAxi2Reg.

2.2.1 NVMeG4-IP

Similar to NVMe-IP, the NVMeG4-IP implements NVMe protocol of the host side to access one NVMe SSD directly without PCIe switch connection. Besides, PCIe soft IP is included to operate with PCIe PHY without using PCIe hard IP. The NVMeG4-IP supports six commands, i.e., Write, Read, Identify, Shutdown, SMART, and Flush. More details of NVMeG4-IP are described in datasheet.
https://dgway.com/products/IP/NVMe-IP/dg_nvmeg4_ip_data_sheet_xilinx_en.pdf

2.2.2 PCIe PHY

PCIe PHY is provided by Xilinx for designing PCIe MAC by Soft IP. The PCIe PHY includes the transceiver and the user interface is PHY Interface for PCI Express (PIPE). When running with NVMeG4-IP, PCIe PHY is configured to be 4-lane width at 16.0 GT/s. More details of PCIe PHY are described in “PG239: PCI Express PHY” document.

https://www.xilinx.com/support/documentation/ip_documentation/pcie_phy/v1_0/pg239-pcie-phy.pdf

2.2.3 Dual port RAM

Two dual 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. NVMeG4-IP and LAxi2Reg have different data bus size, so IdenRAM sets the different bus size for write port and read port. The data interface of NVMeG4-IP (write port) is 256-bit while the interface of LAxi2Reg (read port) is 32-bit. Besides, NVMeG4-IP has double word enable to write only 32-bit data in some cases. The RAM setting on Xilinx IP tool supports the write byte enable. Therefore, the small logic to convert double word enable to be write byte enable is designed as shown in Figure 2-8.

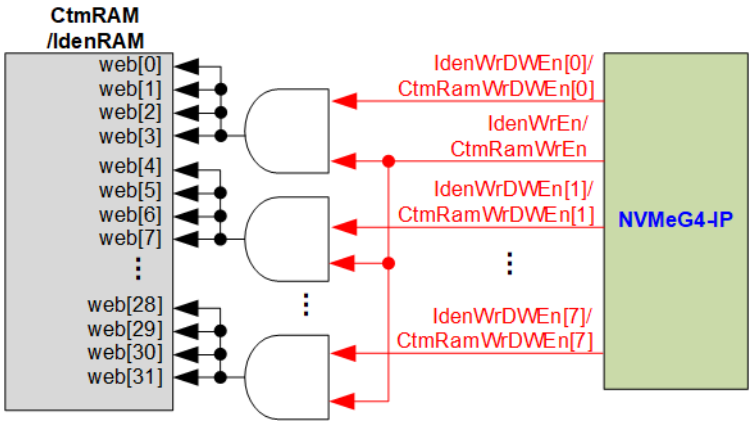

Figure 2-8 Byte write enable conversion logic

Bit[0] of WrDWEn with WrEn signal are the inputs to AND logic. The output of AND logic is fed to bit[7:0] of IdenRAM byte write enable. Bit[1], [2], …, and [7] of WrDWEn are applied to be bit[7:4], [11:8], …, and [31:28] of IdenRAM write byte enable respectively.

Comparing with IdenRAM, CtmRAM is implemented by true dual-port RAM (two read ports and two write ports) with byte write enable. The small logic to convert double word enable of custom interface to be byte write enable must be used, similar to IdenRAM. True dual-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 AXI4-Lite bus is applied to be the bus interface for CPU accessing the peripherals such as Timer and 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 NVMeG4-IP is connected with CPU as a peripheral on 32-bit AXI4-Lite bus for CPU controlling and monitoring.

In the reference design, LAxi2Reg module is designed to connect the CPU system via AXI4-Lite bus standard. CPU specifies the base address and the range to write/read access with LAxi2Reg. More details of LAxi2Reg are shown in Figure 2-9.

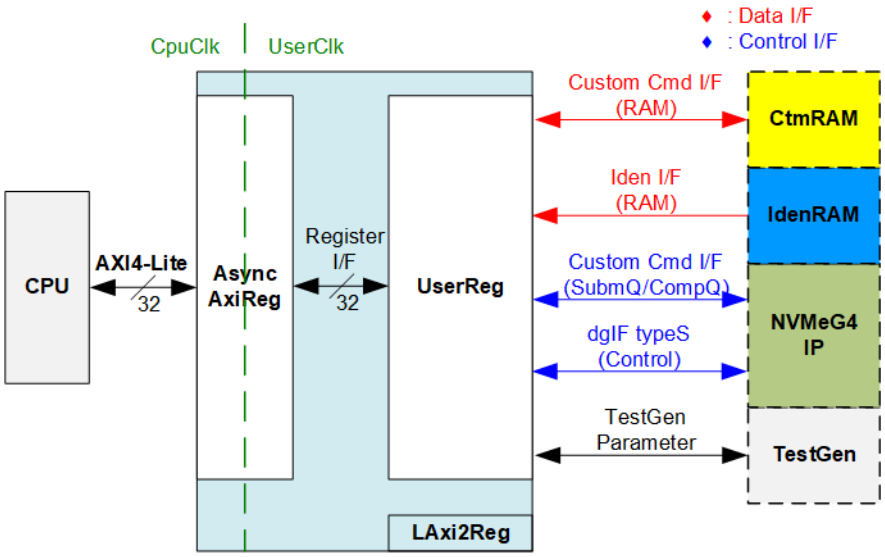

Figure 2-9 CPU and peripherals hardware

LAxi2Reg consists of AsyncAxiReg and UserReg. AsyncAxiReg is designed to convert the AXI4-Lite signals to be the simple register interface which has 32-bit data bus size, similar to AXI4-Lite data bus size. Besides, AsyncAxiReg 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, NVMeG4-IP, and TestGen. More details of AsyncAxiReg and UserReg are described as follows.

2.3.1 AsyncAxiReg

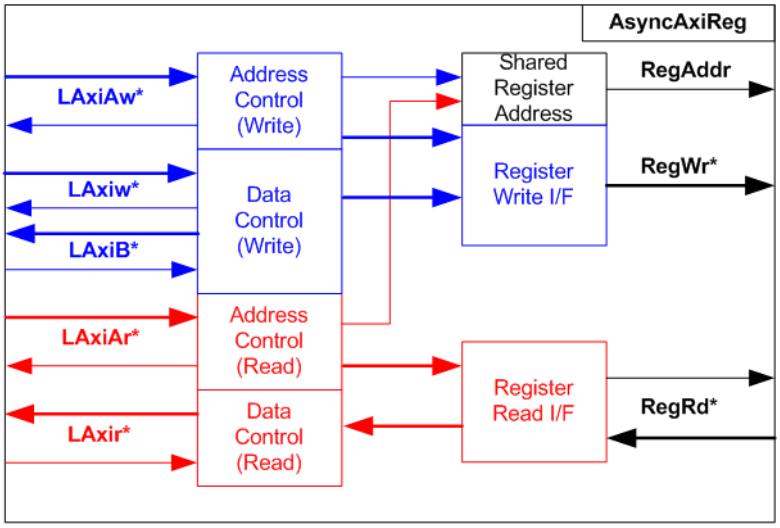

Figure 2-10 AsyncAxiReg Interface

The signal on AXI4-Lite bus interface can be split into five groups, i.e., LAxiAw* (Write address channel), LAxiw* (Write data channel), LAxiB* (Write response channel), LAxiAr* (Read address channel), and LAxir* (Read data channel). More details to build custom logic for AXI4-Lite bus is described in following document.
https://forums.xilinx.com/xlnx/attachments/xlnx/NewUser/34911/1/designing_a_custom_axi_slave_rev1.pdf

According to AXI4-Lite standard, the write channel and the read channel are operated independently. Also, the control and data interface of each channel are run separately. So, the logic inside AsyncAxiReg to interface with AXI4-Lite bus is split into four groups, i.e., Write control logic, Write data logic, Read control logic, and Read data logic as shown in the left side of Figure 2-10. Write control I/F and Write data I/F of AXI4-Lite bus are latched and transferred to be Write register interface with clock domain crossing registers. Similarly, Read control I/F of AXI4-Lite bus are latched and transferred to be Read register interface while Read data is returned from Register interface to AXI4-Lite through clock domain crossing registers. In Register interface, RegAddr is shared signal for write and read access, so it loads the value from LAxiAw for write access or LAxiAr for read access.

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.

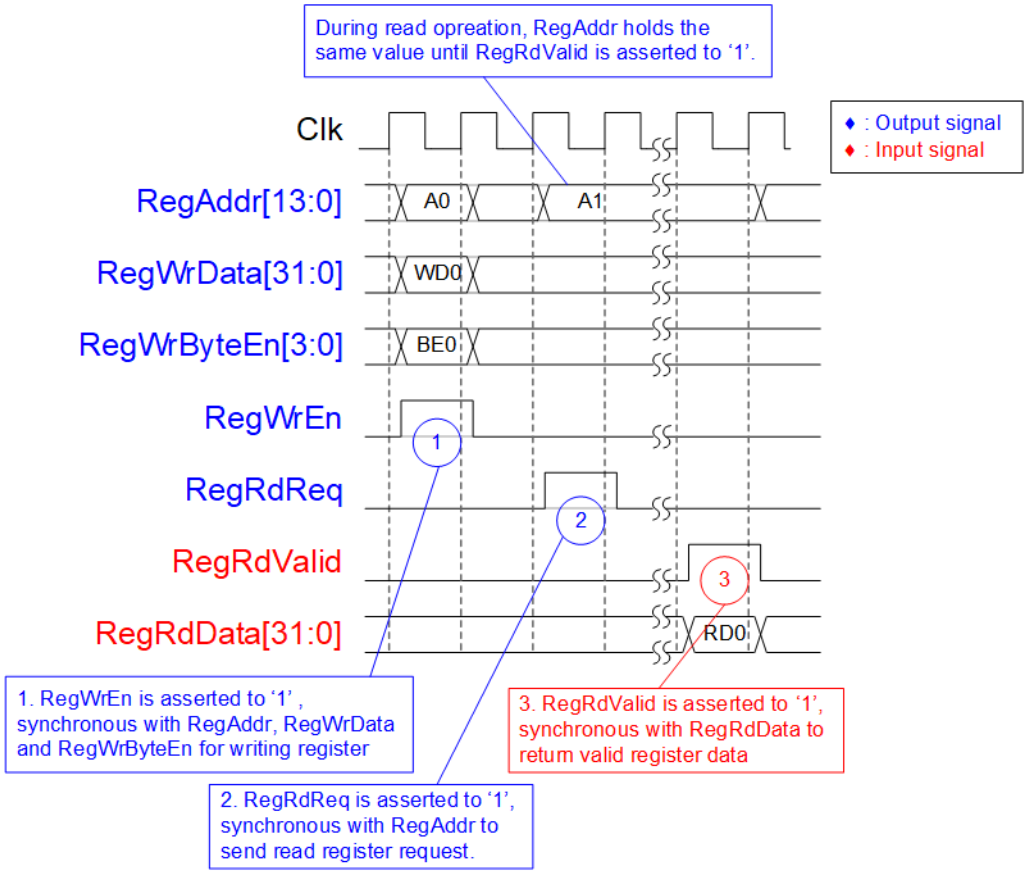

Figure 2-11 Register interface timing diagram

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, AsyncAxiReg asserts RegRdReq to '1' with the valid value of RegAddr. 32-bit data is returned after receiving the read request. The slave detects RegRdReq signal and starts the read transaction. During 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, AsyncAxiReg forwards the read value to LAxir* interface.

2.3.2 UserReg

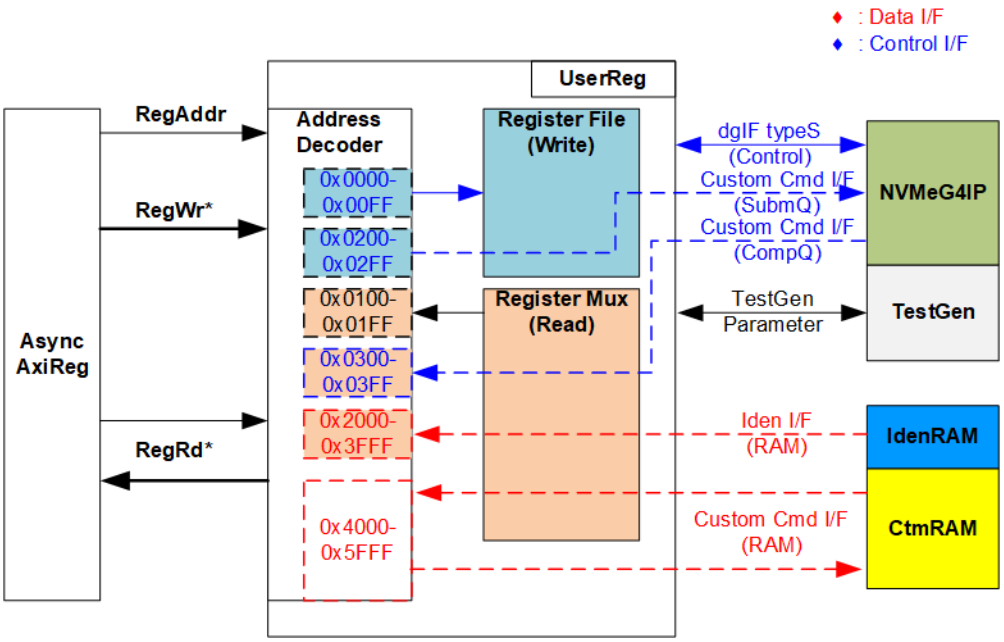

Figure 2-12 UserReg Interface

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 NVMeG4-IP and TestGen. This area is write-access only.
2) 0x0100 – 0x01FF: mapped to read the status signals of NVMeG4-IP and TestGen. This area is read-access only.
3) 0x0200 – 0x02FF: mapped to set the parameters for custom command interface of NVMeG4-IP. This area is write-access only.
4) 0x0300 – 0x03FF: mapped to read the status of custom command interface (NVMeG4-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 custom command RAM interface (NVMeG4-IP). 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 is designed to select the read data within each address area. The lower bit of RegAddr is applied in each Register area to select the active data. Next, the address decoder uses the upper bit to select the read data from active area and returns to CPU. Totally, the latency 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 Rd/Wr	Register Name (Label in the “nvmeiptest.c”)	Description
0x0000 – 0x00FF: Control signals of NVMeG4-IP and TestGen (Write access only)		
BA+0x0000	User Address (Low) Reg (USRADRL_REG)	[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_REG)	[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_REG)	[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_REG)	[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_REG)	[2:0]: Input to be user command (UserCmd of dgIF typeS for NVMeG4-IP) “000”: Identify, “001”: Shutdown, “010”: Write SSD, “011”: Read SSD, “100”: SMART, “110”: Flush, “101”/”111”: Reserved When this register is written, the command request is sent to NVMeG4-IP. After that, the IP starts operating the command.
BA+0x0014	Test Pattern Reg (PATTSEL_REG)	[2:0]: Select test pattern “000”-Increment, “001”-Decrement, “010”-All 0, “011”-All 1, “100”-LFSR
BA+0x0020	NVMe Timeout Reg (NVMTIMEOUT_REG)	[31:0]: Mapped to TimeOutSet[31:0] of NVMeG4-IP
0x0100 – 0x01FF: Status signals of NVMeG4-IP and TestGen (Read access only)		
BA+0x0100	User Status Reg (USRSTS_REG)	[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_REG)	[31:0]: Mapped to LBASize[31:0] of NVMeG4-IP
BA+0x0108	Total disk size (High) Reg (LBASIZEH_REG)	[15:0]: Mapped to LBASize[47:32] of NVMeG4-IP [31]: Mapped to LBAMode of NVMeG4-IP
BA+0x010C	User Error Type Reg (USRERRTYPE_REG)	[31:0]: Mapped to UserErrorType[31:0] of NVMeG4-IP to show error status
BA+0x0110	PCIe Status Reg (PCIESTS_REG)	[7:0]: Unused for NVMeG4-IP [15:8]: MACStatus output from NVMeG4-IP
BA+0x0114	Completion Status Reg (COMPSTS_REG)	[15:0]: Mapped to AdmCompStatus[15:0] of NVMeG4-IP [31:16]: Mapped to IOCompStatus[15:0] of NVMeG4-IP
BA+0x0118	NVMe CAP Reg (NVMCAP_REG)	[31:0]: Mapped to NVMeCAPReg[31:0] of NVMeG4-IP
BA+0x011C	NVMe Test pin Reg (NVMTESTPIN_REG)	[31:0]: Mapped to TestPin[31:0] of NVMeG4-IP
BA+0x0120	MAC Test pin (Low) Reg (MACTESTPINL_REG)	[31:0]: Mapped to MACTestPin [31:0] of NVMeG4-IP
BA+0x0124	MAC Test pin (Mid) Reg (MACTESTPINM_REG)	[31:0]: Mapped to MACTestPin [63:32] of NVMeG4-IP
BA+0x0128	MAC Test pin (High) Reg (MACTESTPINH_REG)	[15:0]: Mapped to MACTestPin [79:64] of NVMeG4-IP

Address Rd/Wr	Register Name (Label in the “nvmeipg4test.c”)	Description
0x0100 – 0x01FF: Status signals of NVMeG4-IP and TestGen (Read access only)		
BA+0x0130 – BA+0x014C	Expected value Word0-7 Reg (EXPPATW0-W7_REG)	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+0x016C	Read value Word0-7 Reg (RDPATW0-W7_REG)	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_REG)	[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_REG)	[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_REG)	[31:0]: Bit[31:0] of the current test data size in TestGen module
BA+0x017C	Current test byte (High) Reg (CURTESTSIZEH_REG)	[24:0]: Bit[56:32] of the current test data size of TestGen module
Other interfaces (Custom command of NVMeG4-IP, IdenRAM, and Custom RAM)		
BA+0x0200 – BA+0x023F Wr	Custom Submission Queue Reg (CTMSUBMQ_REG)	[31:0]: Submission queue entry of SMART and Flush command. Input to be CtmSubmDW0-DW15 of NVMeG4-IP. 0x200: DW0, 0x204: DW1, …, 0x23C: DW15
BA+0x0300 – BA+0x030F Rd	Custom Completion Queue Reg (CTMCOMPQ_REG)	[31:0]: Submission queue entry of SMART and Flush command. Input to be CtmSubmDW0-DW15 of NVMeG4-IP. 0x300: DW0, 0x304: DW1, …, 0x30C: DW3
BA+0x0800 Rd	IP Version Reg (IPVERSION_REG)	[31:0]: Mapped to IPVersion[31:0] of NVMeG4-IP
BA+0x2000 – BA+0x2FFF Rd	Identify Controller Data (IDENCTRL_REG)	4Kbyte Identify controller data structure
BA+0x3000 – BA+0x3FFF Rd	Identify Namespace Data (IDENNAME_REG)	4Kbyte Identify Namespace Data Structure
BA+0x4000 – BA+0x5FFF Wr/Rd	Custom command Ram (CTMRAM_REG)	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 UART and Timer parameters.
2) CPU waits until IP completes PCIe and NVMe initialization process by monitoring IP busy flag (USRSTS_REG[0]=’0’). When some errors are found, the process stops and displays the error message.
3) CPU displays the main menu. There are six menus for running six commands of NVMeG4-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_REG[2:0]=”000”. Next, Test logic generates command and asserts command request to NVMeG4-IP. After that, Busy flag (USRSTS_REG[0]) changes from ‘0’ to ‘1’.
2) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_REG[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_REG[31:0]. Finally, the process is stopped.

3) After busy flag (USRSTS_REG[0]) is de-asserted to ‘0’, CPU displays some information decoded from IdenRAM (IDENCTRL_REG) such as SSD model name and the information from NVMeG4-IP output such as SSD capacity and LBA unit size (LBASIZEH/L_REG).

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 the console. 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_REG, USRLENL/H_REG, and PATTSEL_REG.
3) Set USRCMD_REG[2:0]=”010” for Write command or “011” for Read command. After that, the new command request is sent to NVMeG4-IP for running Write or Read command. Busy flag (USRSTS_REG[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_REG[2:0].

   Bit[0] is de-asserted to ‘0’ when command is completed.
   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_REG[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 which is read from CURTESTSIZEL/H_REG is displayed every second.

5) After busy flag (USRSTS_REG[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_REG) to be SMART command value.
2) Set USRCMD_REG[2:0]=”100”. Next, Test logic generates command and asserts the request to NVMeG4-IP. After that, Busy flag (USRSTS_REG[0]) changes from ‘0’ to ‘1’.
3) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_REG[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_REG[31:0]. Finally, the process is stopped.

4) After busy flag (USRSTS_REG[0]) is de-asserted to ‘0’, CPU decodes SMART command information from CtmRAM (CTMRAM_REG), 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_REG) to be Flush command value.
2) Set USRCMD_REG[2:0]=”110”. Next, Test logic generates command and asserts the request to NVMeG4-IP. After that, Busy flag (USRSTS_REG[0]) changes from ‘0’ to ‘1’.
3) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_REG[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_REG[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_REG[2:0]=”001”. Next, Test logic generates command and asserts the request to NVMeG4-IP. After that, Busy flag (USRSTS_REG[0]) changes from ‘0’ to ‘1’.
2) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_REG[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_REG[31:0]. Finally, the process is stopped.

3) After busy flag (USRSTS_REG[0]) is de-asserted to ‘0’, the SSD and NVMeG4-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_REG
Description	Read CURTESTSIZEH/L_REG 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 set_ctmflush(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the NVMe-IP
Description	Set Flush command to CTMSUBMQ_REG 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_REG 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_REG, 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_REG until the read value from two read times is stable. After that, display the read value on the console. Also, debug signals are read from NVMTESTPIN_REG and MACTESTPINL/M/H_REG to display on the console.

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_hex(unsigned char *char_ptr16B)	
Parameters	*char_ptr16B
Return value	None
Description	Display SMART data as hexadecimal unit.

void show_smart_raw(unsigned char *char_ptr16B)	
Parameters	*char_ptr16B
Return value	None
Description	Display SMART data as decimal unit when the input value is less than 4 MB. Otherwise, display overflow message.

void show_smart_unit(unsigned char *char_ptr16B)	
Parameters	*char_ptr16B
Return value	None
Description	Display SMART data as GB or TB unit. When the input value is more than limit (500 PB), overflow message is displayed instead.

void show_vererr(void)	
Parameters	None
Return value	None
Description	Read RDFAILNOL/H_REG (error byte address), EXPPATW0-7_REG (expected value), and RDPATW0-7_REG (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.

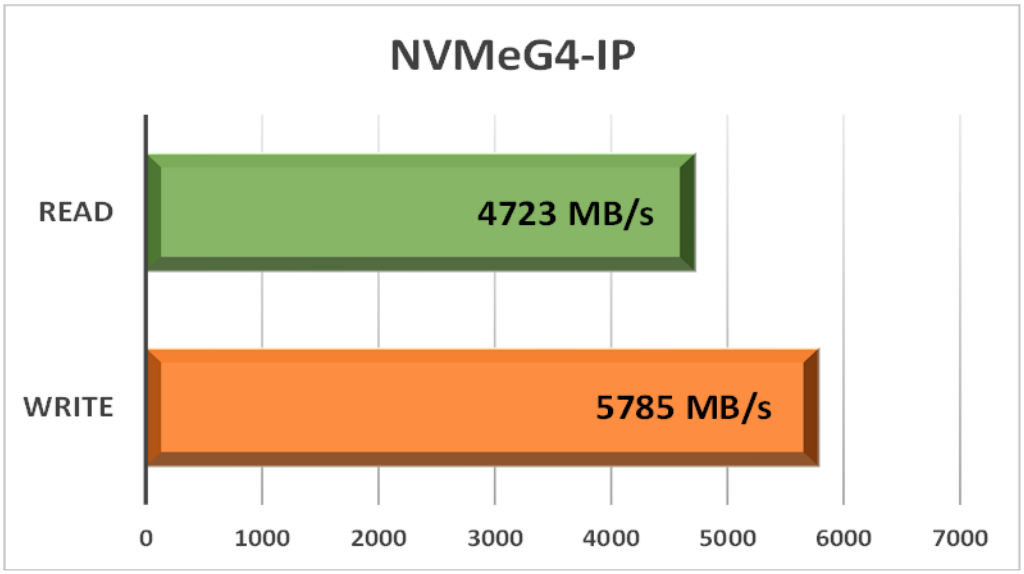

Figure 4-1 Test Performance of NVMeG4 IP demo by using 2TB addlink S95 SSD

By running NVMeG4-IP on ZCU106 board, write performance is about 5785 Mbyte/sec and read performance is about 4723 Mbyte/sec. The test pattern is all zero value and transfer size are 32 Gbytes.

Note: By customizing the IP to increase the buffer size to be 1 Mbyte instead of 256 Kbyte, the read performance is increased. By using 2TB addlink S95 SSD, the read performance is equal to 6646 Mbyte/s.

5 Revision History

Revision	Date	Description
1.0	29-Jan-20	Initial Release
1.1	21-Dec-20	Update performance result
1.2	19-Jul-21	Add details, update register map, and performance result