raNVMe-IP reference design manual

Rev1.4 8-Nov-22

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 http://www.nvmexpress.org/products/.

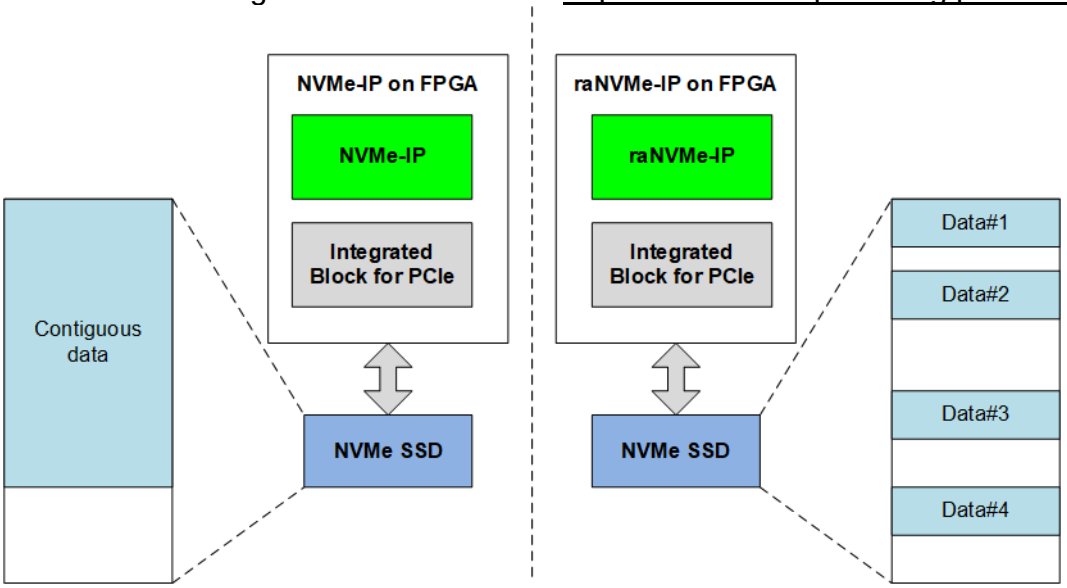

Figure 1-1 NVMe-IP and raNVMe-IP comparison

NVMe-IP is the host controller for accessing one NVMe SSD without integrating CPU or external memory. To achieve the best performance for write and read access, NVMe-IP is designed to store the data in NVMe SSD as contiguous area. Transfer size per command is normally large for storing big data stream and the user send only one command to NVMe-IP. However, NVMe-IP is not proper for some applications which need to store many data types with small size to the different area in the same SSD. The characteristic in this phenomenon looks like random access instead of sequential access.

raNVMe-IP is the IP solution that supports up to 32 Write or Read commands with 4 Kbyte data size. As shown in Figure 1-1, the data stored in the SSD by using raNVMe-IP is random area while the data in the SSD by using NVMe-IP is contiguous area. The random performance when using raNVMe-IP is better than NVMe-IP because of the less overhead time for switching the command. There are many commands in the queue when using raNVMe-IP. However, the result performance in sequential mode by using raNVMe-IP may be less than NVMe-IP because of the smaller internal buffer size (128 Kbytes vs 256 Kbytes).

2 Hardware overview

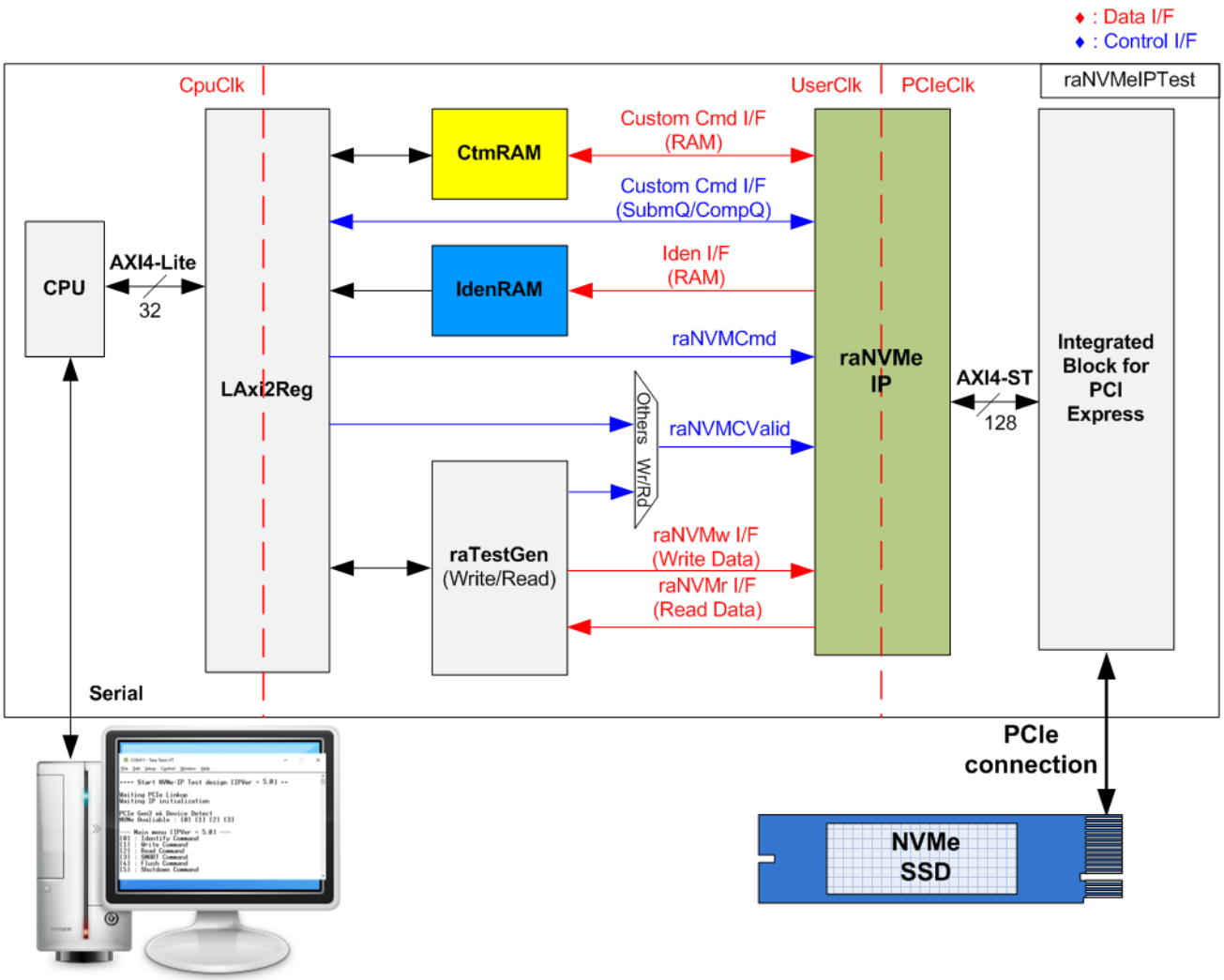

Figure 2-1 raNVMe-IP demo hardware

Following the function of each module, all hardware modules inside the test system are divided to three types, i.e., test function (raTestGen), NVMe function (CtmRAM, IdenRAM, raNVMe-IP, and PCIe block), and CPU system (CPU and LAxi2Reg).

The command request to raNVMe-IP (raNVMCValid) is created by two modules, depending on the command type. When the command is Single mode (Identify, Shutdown, Flush, or SMART), the command request is generated by CPU firmware via LAxi2Reg. When the command is Multiple mode (Write or Read), the command request is generated by raTestGen module. Also, raTestGen connects to the Data stream interface for transferring the data in Write or Read command. The received data returned from SSD in SMART command and Identify command are stored to CtmRAM and IdenRAM, respectively.

CPU and LAxi2Reg are designed to interface with user via Serial interface. Therefore, Serial console is applied to receive the test parameters from the user and display the current test status to the user.

There are three clock domains displayed in Figure 2-1, i.e., CpuClk, UserClk, and PCIeClk. CpuClk is the clock domain of CPU and its peripherals. This clock must be stable clock which may be independent from the clock of other hardware. UserClk is the user clock domain for the user interface of raNVMe-IP. According to raNVMe-IP datasheet, clock frequency of UserClk must be more than or equal to PCIeClk. The reference design uses 275/280 MHz for UserClk. Finally, PCIeClk is the clock output from PCIe hard IP to synchronous with data stream of 128-bit AXI4 stream bus. When the PCIe hard IP is set to 4-lane PCIe Gen3, PCIeClk frequency is equal to 250 MHz.

More details of the hardware are described as follows.

2.1 raTestGen

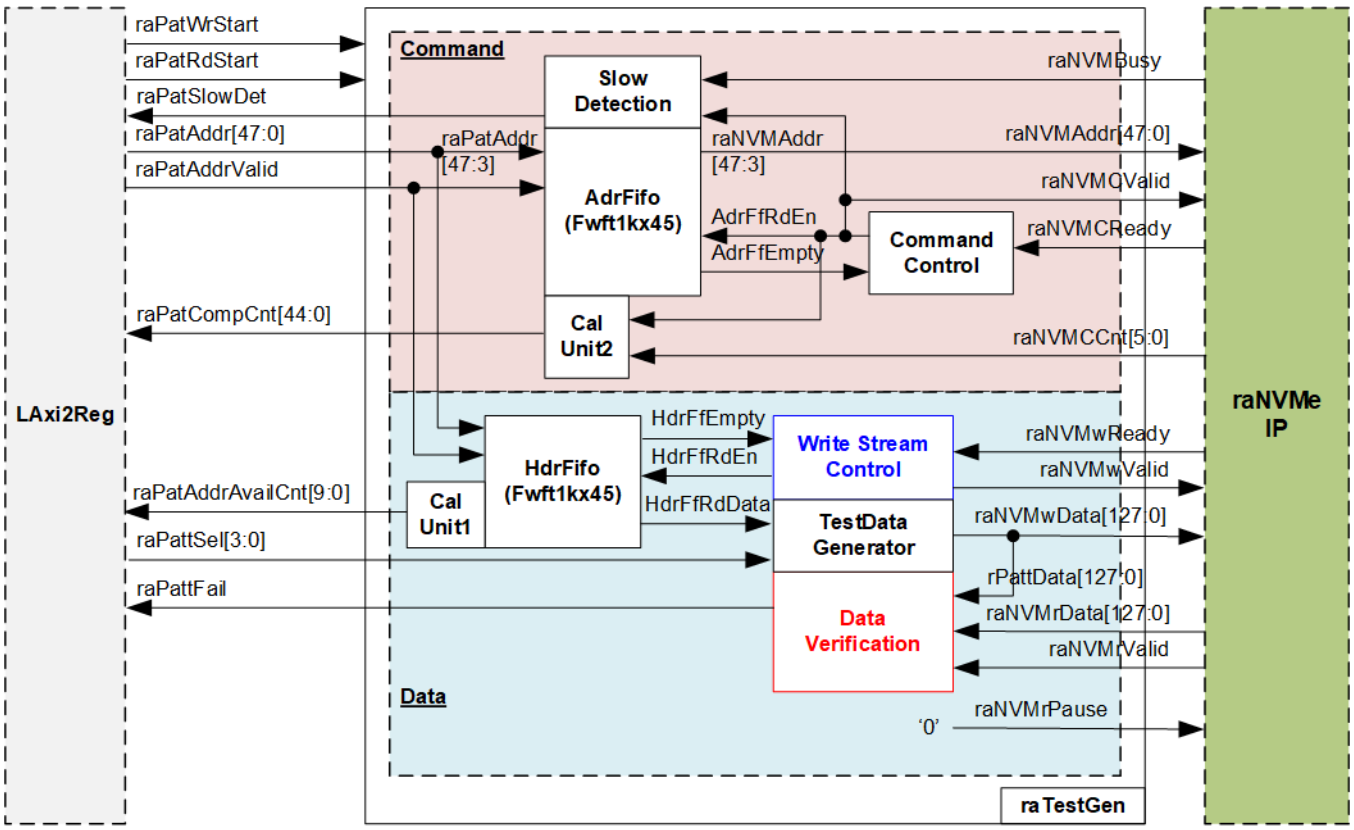

Figure 2-2 raTestGen Interface

raTestGen is the module to generate command request (raNVMCValid) when the user command is Write or Read command. Also, test pattern for sending or verifying in Write or Read command is created in this module. As shown in Figure 2-2, the logic inside raTestGen is divided into two parts - Command and Data.

There are the FIFOs inside Command block and Data block - AdrFifo and HdrFifo to store the address which is created by CPU and set via LAxi2Reg. Though 48-bit address is set, only 45-bit address (bit[47:3]) is stored to both AdrFifo and HdrFifo because bit[2:0] is always equal to 000b to align to 4 KB unit. CPU needs to check the free space of the FIFO before writing the new address, read by raPatAddrAvailCnt. The free space size (raPatAddrAvailCnt) is calculated from the data counter of HdrFifo by CalUnit1. To achieve the best performance, the command request and 4 KB data transmission of each command must be controlled parallelly by Command block and Data block, respectively. Generally, there are many commands that are requested to NVMe SSD via raNVMe-IP before starting transferring 4 KB data. Therefore, HdrFifo is always read for operating data path after AdrFifo which is read for generating command request. When calculating the free space size from the data counter of FIFO, the free space size of HdrFifo is always less than AdrFifo. Therefore, it is safe for CPU to check only the free space size of HdrFifo.

The FIFO depth is set to 1024 to be the buffer for storing many command requests from user, created by CPU. If the amount of command in FIFO is much, CPU has more free time to handle other tasks such as displaying the test progress on Serial console while raNVMe-IP is still processing Write/Read command in the FIFO.

Command

Command Control in Command block reads AdrFifo when the FIFO is not empty (AdrFfEmpty='0') and raNVMe-IP is ready to receive new command (raNVMCReady='1'). AdrFIFO is FWFT type (First Word Fall Through), so the read data (raNVMAddr) is valid at the same time as the read enable (AdrFfRdEn) asserted. Thus, raNVMCValid can be mapped from AdrFfRdEn.

To get the best performance for using raNVMe-IP, the new command must be always ready in AdrFifo. Therefore, the CPU firmware is optimized to support random address generating at high-speed rate. However, raPatSlowDet is designed to indicate that AdrFifo has ever been empty while running the latest Write/Read command. If raPatSlowDet is asserted to '1', it means the FIFO has ever been empty and the Write or Read performance cannot show maximum performance because of CPU resource limitation.

The demo shows the Write/Read performance in IOPs unit on the console every second. Therefore, the logic to show total number of completed command is designed. CalUnit2 creates raPatCompCnt which is equal to the total number of commands sent to raNVMe-IP (counted by using AdrFfRdEn signal) subtracted by the number of incomplete commands of raNVMe-IP (raNVMeCCnt).

Data

Write Stream Control is the state machine for controlling the data transmission to raNVMe-IP in Write command. One data is read by HdrFifo for generating 4 Kbyte data to raNVMe-IP in each Write command request. The write data enable (raNVMwValid) is asserted to '1' to write the test data to raNVMe-IP. Before sending the data, it needs to check if raNVMe-IP is ready to receive the write data. The details of the state machine are described as follows.

(1) stIdle: In Write command, it waits until HdrFifo has the data (HdrFfEmpty='0') for using as 64-bit header data of each 4 Kbyte data. Also, raNVMe-IP must be ready to receive the data by checking raNVMwReady='1'. After that, HdrFfRdEn is asserted to '1' for one clock cycle to read one data and then 4 Kbyte data is transferred in the next step.
(2) stGenWrData: This state is designed to transfer 4 Kbyte data by asserting raNVMwValid to '1' for 256 clock cycles in Write command. 8-bit counter (rDataCnt) counts the amount of Write data or Read data in each command, controlled by raNVMwValid (Write command) or raNVMrValid (Read command). After finishing transferring 256 data (256x128-bit), continue to the next step.
(3) stDelay: This state adds one clock cycle latency time to wait raNVMwReady updated. After that, it returns to stIdle.

TestData Generator creates test data for sending to raNVMe-IP (raNVMwData) in Write command or verifying with the received data from raNVMe-IP (rPattData) in Read command. The test data size of one command is 4 Kbytes which consists of 64-bit header data and the test pattern, selected by raPattSel.

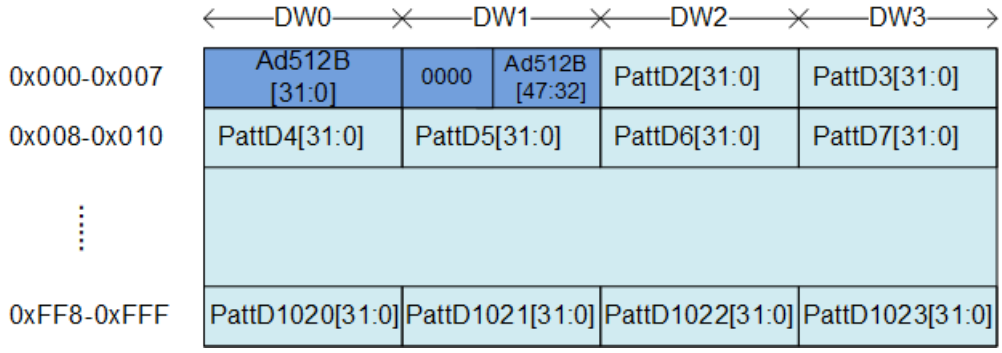

Figure 2-3 Test pattern format of 4096-byte data for Increment/Decrement/LFSR pattern

As shown in Figure 2-3, 64-bit header in DW#0 and DW#1 is created by using 48-bit address, read from HdrFifo. Remaining data (DW#2 – DW#1023) is the test pattern which can be selected by three formats: 32-bit incremental data, 32-bit decremental data, and 32-bit LFSR counter. 32-bit incremental data is designed by using the up-counter. The decremental data can be designed by connecting NOT logic to incremental data. The equation of 32-bit LFSR data is x^31 + x^21 + x + 1. Four 32-bit LFSR data must be generated in the same clock to create 128-bit data, so the logic uses look-ahead style to generate four LFSR data in one clock cycle.

Besides, the user can select test pattern to be all zero or all one data to show the best performance of some SSDs. When the pattern is all zero or all one, there is no 64-bit header inserted to 4-Kbyte data.

The read operation is controlled without the state machine. The logic is always ready to receive the data from raNVMe-IP by de-asserting raNVMrPause to ‘0’ to get the best read performance. Therefore, raNVMrValid is always asserted to ‘1’ for 256 clock cycles continuously for transferring 4 KB data of each Read command request. The first data of each 4 KB data block is detected by checking the rising edge of raNVMrValid. Therefore, HdrFfRdEn is asserted to ‘1’ for reading the address of the Read command request when the logic detects the rising edge of raNVMrValid. TestData Generator is applied to generate the expected data for Read command operation. raPattFail is asserted to ‘1’ when the data verification is failed.

Timing diagram of raTestGen when running Write command and Read command is shown in Figure 2-4 and Figure 2-5, respectively.

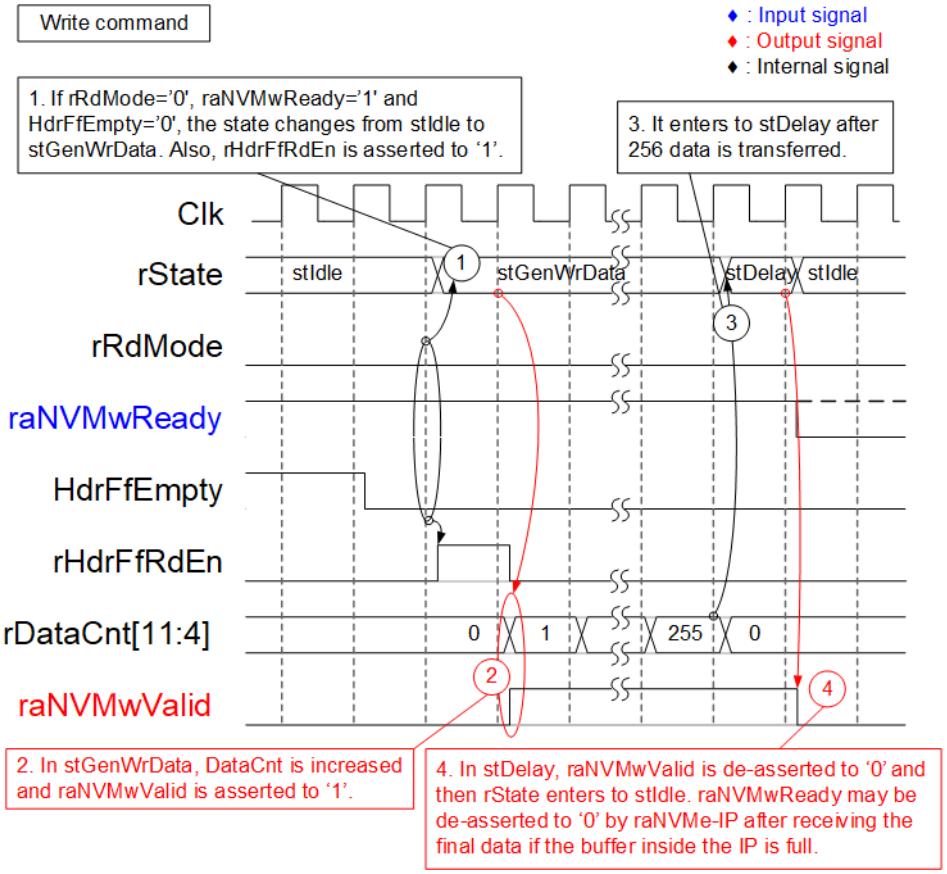

Figure 2-4 Timing diagram of raTestGen when running Write command

(1) When running Write command (RdMode='0'), HdrFfEmpty is applied to check if the new address is requested by CPU firmware. Also, raNVMwReady is applied to check if raNVMe-IP is ready to receive the data. After HdrFfEmpty='0' and raNVMwReady='1', read enable of HdrFifo (HdrFfRdEn) is asserted to '1' to read one address that is requested by CPU for being the header of each 4 KB data block. After that, it enters to the next state, stGenWrData.

(2) In stGenWrData state, raNVMwValid is asserted to '1' for sending 256 data to raNVMe-IP in Write command. DataCnt is increased to count the amount of transferred data to raNVMe-IP.

(3) After finishing transferring 256 data, monitored by DataCnt, it changes to stDelay.

(4) raNVMwValid is de-asserted to '0' when the state is stDelay. After that, the state enters to stIdle for processing the next command, as described in step 1). After raNVMe-IP receives the last data of each 4 KB, it will check the internal buffer status. raNVMwReady is de-asserted to '0' of the internal buffer is full. Otherwise, raNVMwReady is still asserted to '1' for receiving the next 4 KB data block.

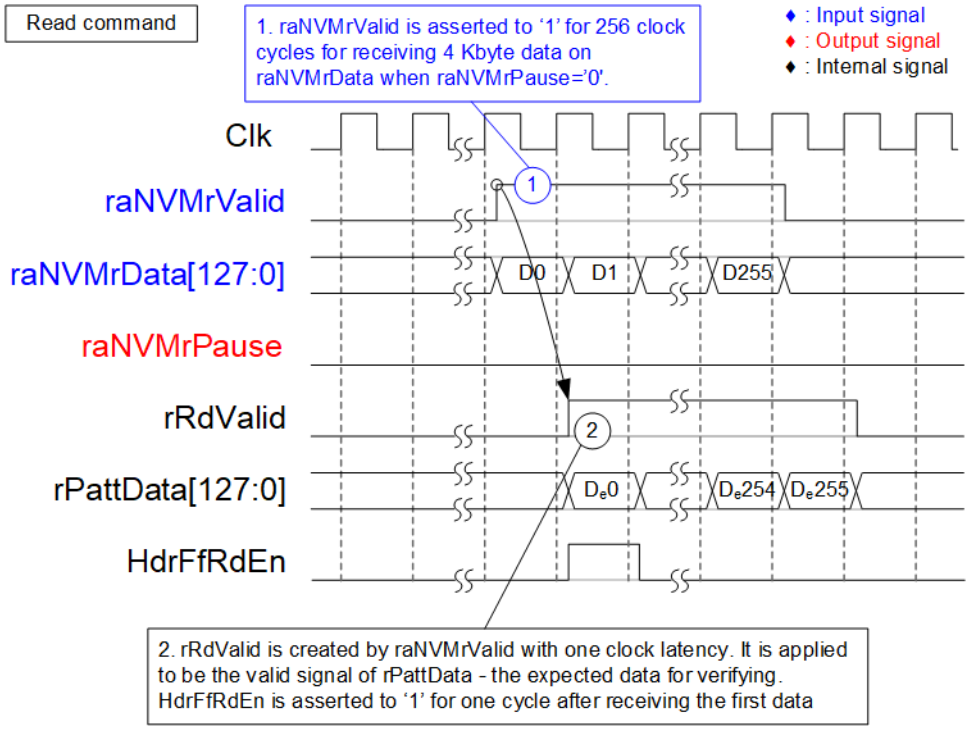

Figure 2-5 Timing diagram of raTestGen when running Read command

(1) To achieve the best read performance, raNVMrPause is always de-asserted to '0'. Therefore, 4Kbyte read data is returned by raNVMe-IP continuously by asserting raNVMrValid to '1' for 256 clock cycles. The received data is valid on raNVMrData when raNVMrValid is asserted to '1'.

(2) When the first data of 4Kbyte data is received, detected by the rising edge of raNVMrValid, HdrFfRdEn is asserted to '1' to read one address from HdrFifo. The address is applied for creating the header data of 4Kbyte data block. The expected data (D_eX) is created by TestData Generator to compare with the received data (raNVMrData). rRdValid, created by raNVMrValid with one clock cycle latency, is applied to be the valid signal of Test pattern (rPattData). Also, it is applied to enable the data verification function. To compare data with rPattData, the received data (raNVMrData) must be fed to one Flip-Flop to add one-clock latency for synchronization with rPattData.

2.2 NVMe

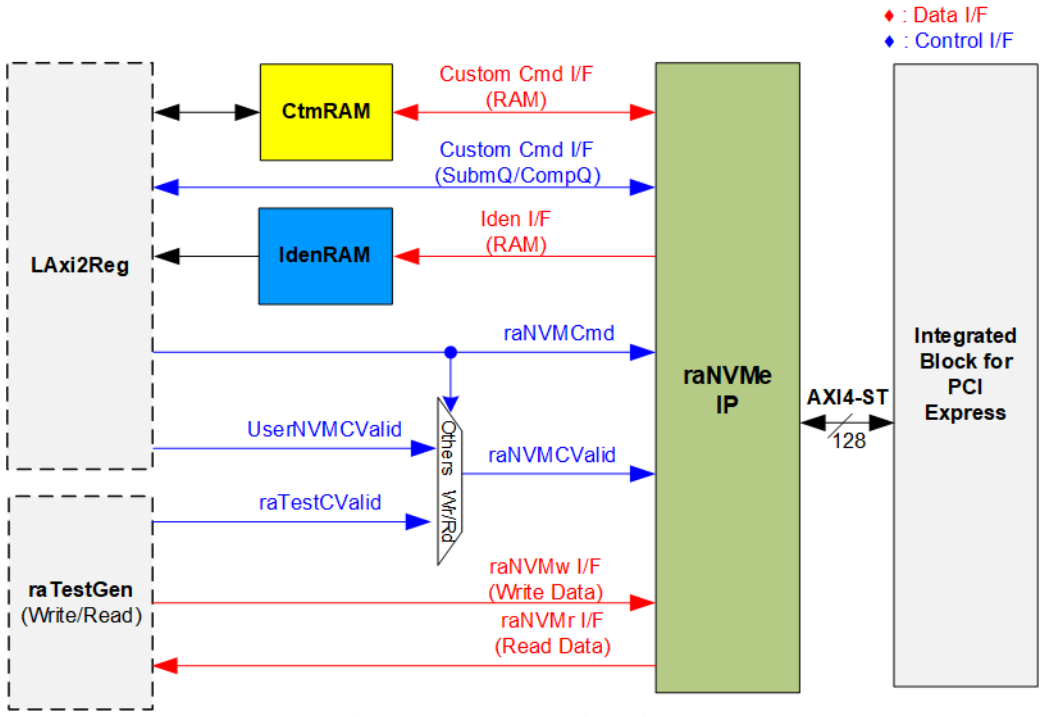

Figure 2-6 NVMe hardware

Figure 2-6 shows the interface of raNVMe-IP in the reference design. The user interface of raNVMe-IP consists of Control interface and Data interface. The Control interface receives command and the parameters from the user while the Data interface transfers the data when the command needs data transferring.

There are two types of the command - Single mode command and Multi-mode command. The command value (raNVMCmd) is set by CPU firmware via LAxi2Reg, but the command request (raNVMCValid) is controlled by two sources - UserNVMCValid and raTestCValid. When the command is Single mode, the command request (UserNVMCValid) is created by CPU firmware. When the command is Multiple mode, the command request (raTestCValid) is created by raTestGen. SMART command and Flush command are the Custom commands that need to set the additional parameters via Custom Cmd I/F. In the test design, these parameters are also set by CPU firmware via LAxi2Reg module.

There are four commands which has data transferring by using its own interface.
- Custom Cmd I/F (RAM): Transfer SMART data to CtmRAM in SMART command.
- Iden I/F (RAM): Transfer Identify data to IdenRAM in Identify command.
- raNVMw I/F: Transfer Write data from raTestGen in Write command.
- raNVMr I/F: Transfer Read data from raNVMe-IP in Read command.

Though each command uses the different interface for transferring the data, every data interface has the same data bus size, 128-bit data.

2.2.1 raNVMe-IP

The raNVMe-IP implements NVMe protocol of the host side to access one NVMe SSD. Up to 32 Write commands or Read commands with random addressing can be sent to raNVMe-IP at the same time. Six commands are supported by the IP, i.e., Write, Read, Identify, Shutdown, SMART, and Flush. raNVMe-IP can connect with the PCIe hard IP directly. More details of raNVMe-IP are described in datasheet.
https://dgway.com/products/IP/NVMe-IP/dg_ranvme_ip_data_sheet_xilinx.pdf

2.2.2 Integrated Block for PCIe

This block is the hard IP in Xilinx device which implements Physical, Data Link, and Transaction Layers of PCIe specification. More details are described in Xilinx document.
PG156: UltraScale Devices Gen3 Integrated Block for PCI Express
PG213: UltraScale+ Devices Integrated Block for PCI Express

2.2.3 Dual port RAM

Two dual port RAMs, CtmRAM and IdenRAM, store the returned data from Identify command and SMART command, respectively. IdenRAM has 8 Kbyte size to store 8 Kbyte data, output from Identify command. raNVMe-IP and LAxi2Reg have the different data bus size, 128-bit on raNVMe-IP but 32-bit on LAxi2Reg, so IdenRAM is asymmetric RAM that has the different bus size for Write interface and Read interface. Also, raNVMe-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, so the small logic to convert double word enable to be write byte enable is designed, as shown in Figure 2-7.

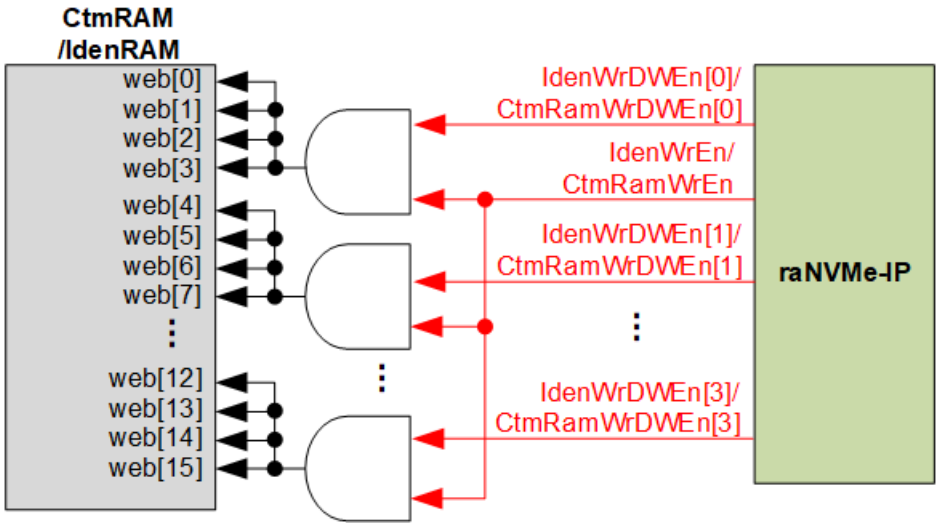

Figure 2-7 Byte write enable conversion logic

Bit[0] of WrDWEn with WrEn signal are the inputs to AND logic. The output of AND logic is fed to bit[3:0] of IdenRAM byte write enable. Bit[1], [2], and [3] of WrDWEn are applied to be bit[7:4], [11:8], and [15:12] of IdenRAM write byte enable, respectively.

Comparing with IdenRAM, CtmRAM is implemented by true dual-port RAM 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. The data size returned from SMART command is 512 bytes.

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 system integrates an additional peripheral to access raNVMe-IP test logic by assigning a unique base address and the address range. Therefore, the hardware logic must be designed to support AXI4-Lite bus standard for CPU writing and reading. LAxi2Reg module is applied to connect with the CPU system via AXI4-Lite bus standard, as shown in Figure 2-8.

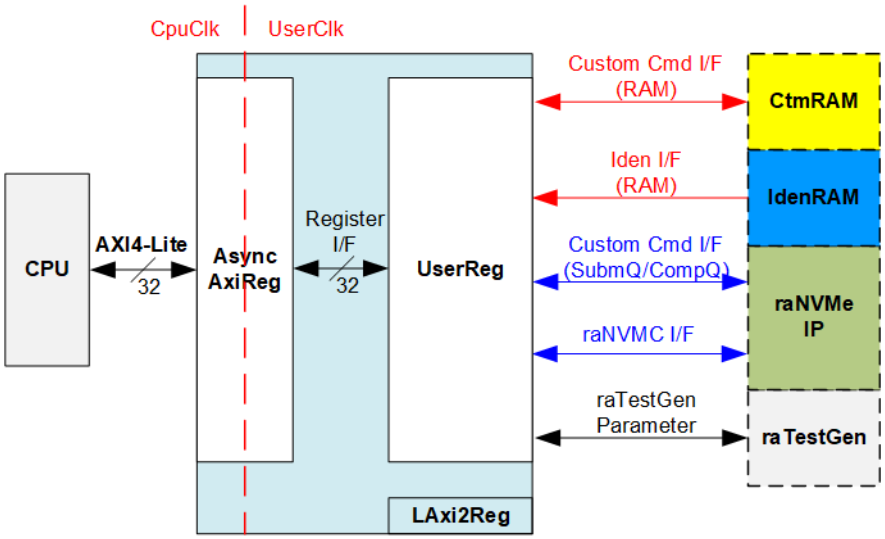

Figure 2-8 CPU and peripherals hardware

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 and UserClk.

UserReg includes the register file of the parameters and the status signals of the modules in raNVMe-IP Test system, i.e., CtmRAM, IdenRAM, raNVMe-IP, and raTestGen. More details of AsyncAxiReg and UserReg are described as follows.

2.3.1 AsyncAxiReg

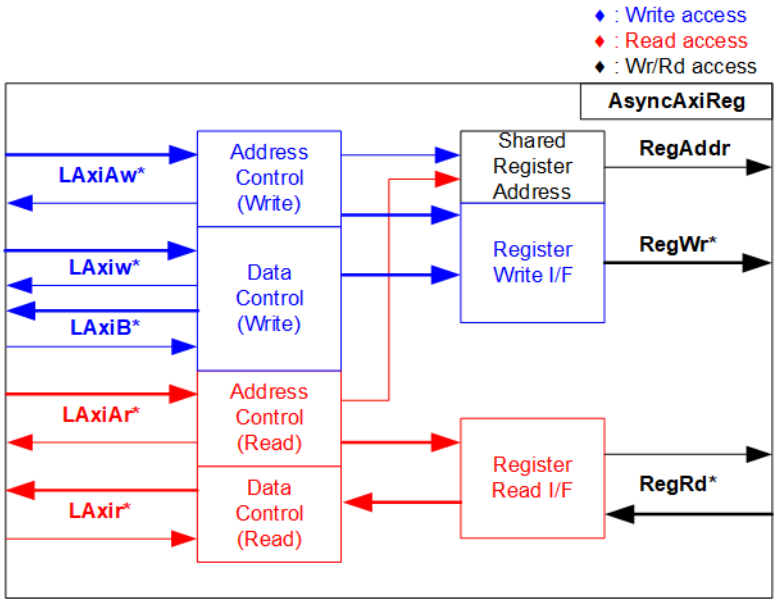

Figure 2-9 AsyncAxiReg Interface

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 2-9. 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 the 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 2-10.

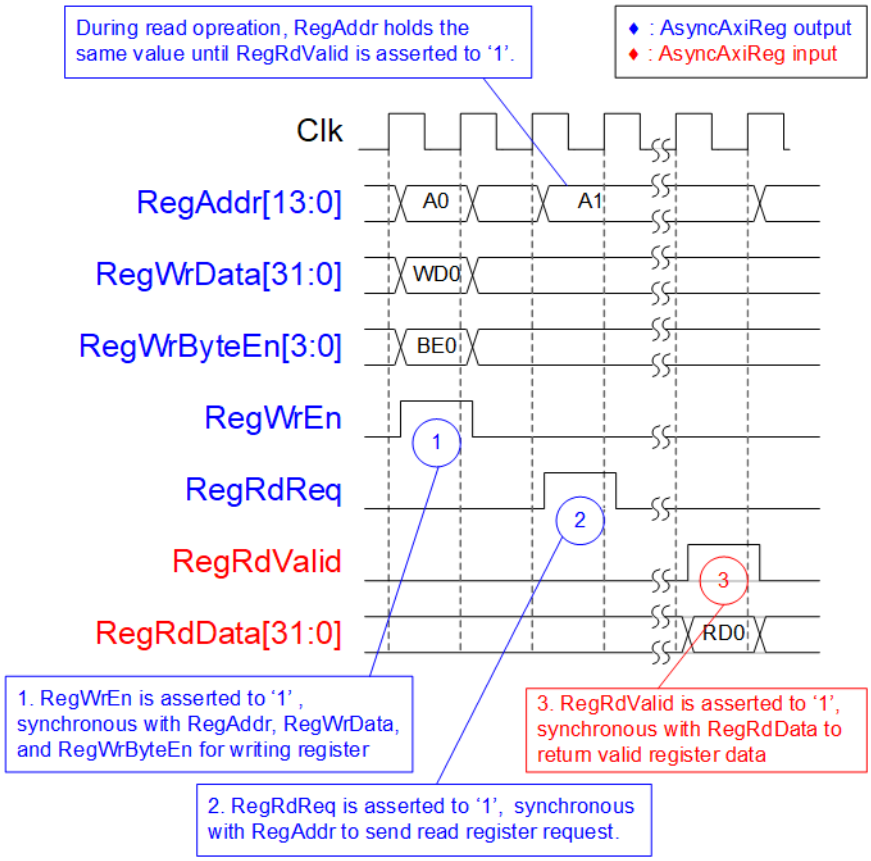

Figure 2-10 Register interface timing diagram

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 asserted 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 levels 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

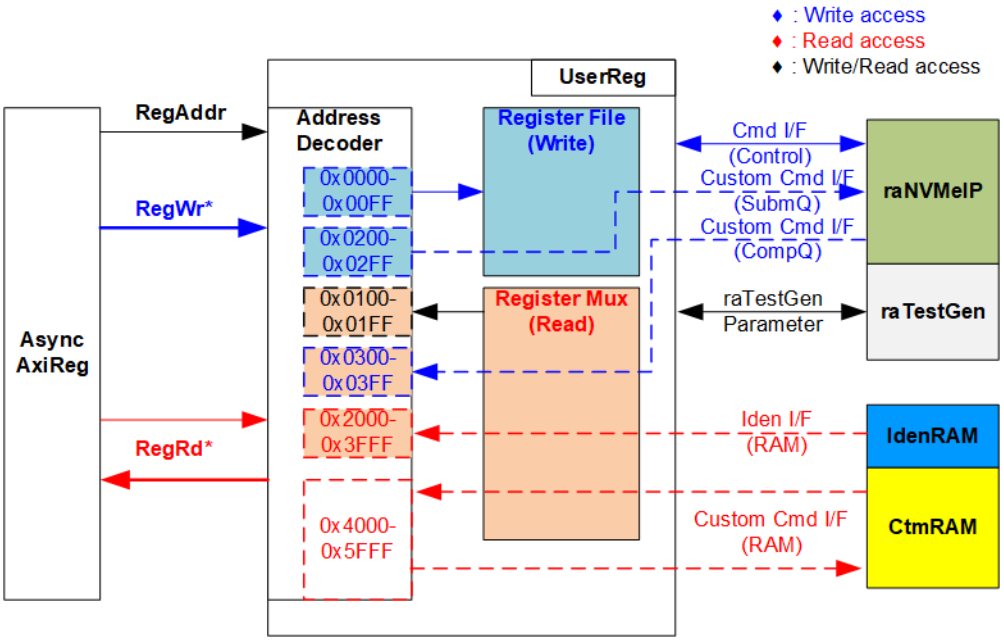

Figure 2-11 UserReg Interface

The logic inside UserReg consists of Address decoder, RegFile, and RegMux. The address decoder decodes the address which is requested from AsyncAxiReg and then selects the active register for write or read transaction. The address range assigned in UserReg is split into six areas, as shown in Figure 2-11.

1) 0x0000 – 0x00FF: mapped to set the command with the parameters of raNVMe-IP and raTestGen. This area is write-access only.
2) 0x0200 – 0x02FF: mapped to set the parameters for Custom command interface of raNVMe-IP. This area is write-access only.
3) 0x0100 – 0x01FF: mapped to read the status signals of raNVMe-IP and raTestGen. This area is read-access only.
4) 0x0300 – 0x03FF: mapped to read the status of Custom command interface (raNVMe-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 data or read data with Custom command RAM interface. This area supports write and read access. The demo shows only read access for running SMART command.

Address decoder decodes the upper bit of RegAddr for selecting the active hardware that is raNVMe-IP, raTestGen, IdenRAM, or CtmRAM. 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 use 32-bit pointer to set the hardware register.

To read register, multi-level multiplexers (mux) select 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	Register Name (Label in the "ranvmeiptest.c")	Description
0x0000 – 0x00FF: Control signals of raNVMe-IP and TestGen (Write access only)		
BA+0x0000	User Address (Low) Reg (USRADRL_INTREG)	[31:0]: Input to be start address as 512-byte unit (UserAddr[31:0] of raNVMe-IP for Write or Read command)
BA+0x0004	User Address (High) Reg (USRADRH_INTREG)	[15:0]: Input to be start address as 512-byte unit (UserAddr[47:32] of raNVMe-IP for Write or Read command) When writing this register, 48-bit UserAddr is stored to FIFO within raTestGen.
BA+0x0010	User Command Reg (USRCMD_INTREG)	[2:0]: Input to be user command (UserCmd of raNVMe-IP) 000b: Identify, 001b: Shutdown, 010b: Write SSD, 011b: Read SSD, 100b: SMART, 110b: Flush, 101b/111b: Reserved When Single mode command (not Write/Read) is written to this register, the new command request (raNVMCValid) is asserted to raNVMe-IP. Otherwise, start flag for Write or Read command is asserted to raTestGen. Multi-mode command request (raNVMCValid) is asserted by raTestGen.
BA+0x0014	Test Pattern Reg (PATTSEL_INTREG)	[2:0]: Select test pattern. 000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1, 100b-LFSR. [3]: Verification enable. '0' -No verification, '1'-Enable verification.
BA+0x0020	NVMe Timeout Reg (NVMTIMEOUT_INTREG)	[31:0]: Timeout value of raNVMe-IP (TimeOutSet[31:0] of raNVMe-IP)
0x0100 – 0x01FF: Status signals of raNVMe-IP and TestGen (Read access only)		
BA+0x0100	User Status Reg (USRSTS_INTREG)	[0]: Mapped to raNVMBusy of raNVMe-IP. '0': IP is Idle, '1': IP is busy. [1]: Mapped to raNVMError of raNVMe-IP. '0': No error, '1': Error is found. [2]: Data verification fail. '0': Normal, '1': Error. [8:4]: Mapped to raNVMCId of raNVMe-IP to show current command ID. [16:12]: Mapped to raNVMDId of raNVMe-IP to show command ID which currently transfers data on Data stream interface. [25:20]: Mapped to raNVMCCnt of raNVMe-IP to show remaining command count stored in raNVMe-IP when running Write or Read command. [31]: Mapped to raNVMCReady of raNVMe-IP to show command ready.
BA+0x0104	Total disk size (Low) Reg (LBASIZEL_INTREG)	[31:0]: Mapped to LBASize[31:0] of raNVMe-IP
BA+0x0108	Total disk size (High) Reg (LBASIZEH_INTREG)	[15:0]: Mapped to LBASize[47:32] of raNVMe-IP
BA+0x010C	User Error Type Reg (USRERRTYPE_INTREG)	[31:0]: Mapped to UserErrorType[31:0] of raNVMe-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]: PCIe link speed from PCIe hard IP (00b: Not linkup, 01b: PCIe Gen1, 10b: PCIe Gen2, 11b: PCIe Gen3) [7:4]: PCIe link width status from PCIe hard IP (0001b: 1-lane, 0010b: 2-lane, 0100b: 4-lane, 1000b: 8-lane) [13:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in Integrated Block for PCIe datasheet
BA+0x0114	NVMe CAP Reg (NVMCAP_INTREG)	[31:0]: Mapped to NVMeCAPReg[31:0] of raNVMe-IP
BA+0x0118	Admin Completion Status Reg (ADMCOMPSTS_INTREG)	[15:0]: Mapped to AdmCompStatus[15:0] of raNVMe-IP to show status of Admin completion
BA+0x011C	IO Completion Status Reg (IOCOMPSTS_INTREG)	[31:0]: Mapped to IOCompStatus[15:0] of raNVMe-IP to show status of I/O completion.
BA+0x0120	NVMe IP Test pin Reg (NVMTESTPIN_INTREG))	[31:0]: Mapped to TestPin[31:0] of raNVMe-IP

Address Rd/Wr	Register Name (Label in the “ranvmeiptest.c”)	Description
0x0100 – 0x01FF: Status signals of raNVMe-IP and TestGen (Read access only)		
BA+0x0124	Test FIFO Remaining Count Reg (TESTFFREMCNT_INTREG)	[9:0]: Remaining size of FIFO in raTestGen (raPatAddrAvailCnt of raTestGen)
BA+0x0128	Slow Transfer detect Reg (SLOWDET_INTREG)	[0]: ‘1’-Slow transfer is found, ‘0’-Not detect. It is asserted when CPU firmware sends the address for Write or Read command at slower rate than raNVMe-IP processing. If this flag is asserted to ‘1’, SSD performance result is limited by CPU task. This flag is auto-cleared when CPU sets USRCMD_INTREG=Write or Read command.
BA+0x0140	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+0x0144	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+0x0148	Completed Count (Low) Reg (CMDCMPCNTL_INTREG)	[31:0]: Bit[31:0] of the number of completed commands in raTestGen (raPatCompCnt of raTestGen)
BA+0x014C	Completed Count (High) Reg (CMDCMPCNTH_INTREG)	[12:0]: Bit[44:32] of the number of completed commands in raTestGen (raPatCompCnt of raTestGen)
BA+0x0180 - BA+0x018F	Expected value Word0-3 Reg (EXPPATW0-W3_INTREG)	128-bit of the expected data at the 1st failure data in Read command 0x0180: Bit[31:0], 0x0184: Bit[63:32], …, 0x018C: Bit[127:96]
BA+0x01C0 - BA+0x01CF	Read value Word0-3 Reg (RDPATW0-W3_INTREG)	128-bit of the read data at the 1st failure data in Read command 0x01C0: Bit[31:0], 0x01C4: Bit[63:32], …, 0x01CC: Bit[127:96]
Other interfaces (Custom command of raNVMe-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 raNVMe-IP. 0x200: DW0, 0x204: DW1, …, 0x23C: DW15
BA+0x0300 – BA+0x030F Rd	Custom Completion Queue Reg (CTMCOMPQ_STRUCT)	[31:0]: CtmCompDW0-DW3 output from raNVMe-IP. 0x300: DW0, 0x304: DW1, …, 0x30C: DW3
BA+0x0800 Rd	IP Version Reg (IPVERSION_INTREG)	[31:0]: Mapped to IPVersion[31:0] of raNVMe-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 (ranvmeiptest.c)

After system boot-up, CPU starts the initialization sequence as follows.

1) CPU initializes UART and Timer parameters.
2) CPU waits until PCIe connection links up (PCIESTS_INTREG[0]='1').
3) CPU waits until raNVMe-IP completes initialization process (USRSTS_INTREG[0]='0'). If some errors are found, the process stops with displaying the error message.
4) CPU displays PCIe link status (the number of PCIe lanes and the PCIe speed) by reading PCIESTS_INTREG[7:2].
5) CPU displays the main menu. There are six menus for running six commands of raNVMe-IP, i.e., Identify, Write, Read, SMART, Flush, and Shutdown.

More details for operating each command in CPU firmware are described as follows.

3.1.1 Identify Command

The sequence of the firmware when user selects Identify command is below.

1) Set USRCMD_INTREG=000b to send Identify command request to raNVMe-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' after finishing operating the command. After that, the data from Identify command of raNVMe-IP is stored in 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 raNVMe-IP output such as SSD capacity (LBASIZEH/L_INTREG). Finally, CPU prepares the polynomial for creating LFSR test pattern following SSD capacity. LFSR test pattern is applied to generate random address for Write command test or Read command test.

3.1.2 Write/Read Command

The sequence of the firmware when user selects Write/Read command is below.

1) Receive transfer mode, data verification (for Read command), start address, transfer length, and test pattern from Serial console. If some inputs are invalid, the operation is cancelled.
   Note: Start address and transfer length must be aligned to 8.
2) Get all inputs and set test pattern to PATTSEL_INTREG.
3) Set USRCMD_INTREG[2:0]= 010b for Write command or 011b for Read command.
4) Calculate the maximum address request that can be created. Compare the remaining buffer size to store the address request in the Test logic (TESTFFREMCNT_INTREG[9:0]) with the remaining user request length. The firmware selects the less value.
5) Repeat this step to send the address to the hardware by setting USRADRL/H_INTREG[31:0] until finishing sending total request. The address value is calculated by using the following rules.
   a. If this is the first value of the test, use the Start address received from the user.
   b. If transfer mode is Sequential mode, the next address is increased to be the next 4 Kbyte address.
   c. If transfer mode is Random mode, calculate the next address by using LFSR equation until the address is not more than SSD capacity.
6) CPU reads error status by reading USRSTS_INTREG[2:1]. Display the error message when some bits are asserted to '1'.

   Bit[1] is asserted when IP error is detected. After that, error message is displayed on the console to show the error details. Finally, the process is hanged up.
   Bit[2] is asserted when data verification is enabled in Read command and data failure is found. After that, the verification error message is displayed. However, CPU is still running until the operation is done or user inputs any keys to cancel operation.

   While the command is running, the current transfer size (CMDCMPCNTL/H_INTREG) is read and displayed as percentage every second. Go to the next step (step 7) if remaining total transfer size is equal to 0. Otherwise, return to step 4) to calculate the transfer size for the next loop.
7) CPU waits until raNVMe-IP finishes the operation (USRSTS_INTREG[0]='0'). Display the error message if some errors are found.
8) Read SLOWDET_INTREG. If the flag is asserted, print the warning message about the performance limitation by CPU firmware. Finally, display the test results on the console, i.e., total time usage, total transfer size, and transfer speed.

3.1.3 SMART Command

The sequence of the firmware when user selects SMART command is below.

1) Set 16-Dword of Submission queue entry (CTMSUBMQ_STRUCT) to be SMART command value.
2) Set USRCMD_INTREG[2:0]=100b to send SMART command request to raNVMe-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' after finishing operating the command. Next, the data from SMART command of raNVMe-IP is stored in CtmRAM.
   Bit[1] is asserted 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 displays the information which is decoded from CtmRAM (CTMRAM_CHARREG) such as 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 sequence of the firmware when user selects Flush command is below.

1) Set 16-Dword of Submission queue entry (CTMSUBMQ_ STRUCT) to be Flush command value.
2) Set USRCMD_INTREG[2:0]=110b to send Flush command request of raNVMe-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' after finishing operating the command. Next, the CPU returns to the main menu.
   Bit[1] is asserted 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 sequence of the firmware when user selects Shutdown command is below.

1) Set USRCMD_INTREG[2:0]=001b to send Shutdown command request of raNVMe-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' after finishing operating the command. After that, the CPU goes to the next step.
   Bit[1] is asserted 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 Shutdown command is done, the SSD and raNVMe-IP change to inactive status. The CPU cannot receive the new command from user and the user must power off the test system.

3.2 Function list in Test firmware

unsigned int cal_lfsr(unsigned int curaddr)	
Parameters	curaddr: The latest value of 32-bit address
Return value	The next value of 32-bit address after finishing LFSR calcalation
Description	Calculate LFSR value in Write or Read command when running random access mode.

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 raNVMe-IP
Description	Run SMART command or Flush command, following in topic 3.1.3 (SMART Command) and 3.1.4 (Flush Command).

int get_param(unsigned int user_cmd, userin_struct* userin)	
Parameters	user_cmd: 2-Write command, 3-Read command 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 raNVMe-IP
Description	Set Flush command to CTMSUBMQ_STRUCT and call exec_ctm function to operate Flush command.

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

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.

void show_result(void)	
Parameters	None
Return value	None
Description	Print total size by reading CMDCMPCNTL/H_INTREG and then calling 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 on MB/s unit and IOPS 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-W3_INTREG (expected value), and RDPATW0-W3_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). Call wrrd_seq() function when running in sequential mode or wrrd_rand() function when running in random mode.

void wrrd_rand(unsigned int user_cmd, userin_struct* userin)	
Parameters	user_cmd: 2-Write command, 3-Read command userin: Three inputs from user, i.e., start address, total length in 512-byte unit, and test pattern
Return value	None
Description	Run Write command or Read command with 32-bit address calculation in 4 KB unit by using random mode.

void wrrd_seq(unsigned int user_cmd, userin_struct* userin)	
Parameters	user_cmd: 2-Write command, 3-Read command userin: Three inputs from user, i.e., start address, total length in 512-byte unit, and test pattern
Return value	None
Description	Run Write command or Read command with 48-bit address calculation in 512-byte unit by using sequential mode

4 Example Test Result

The test results when running demo system by using 800 GB Intel Optane P5800X and KCU116 board (PCIe Gen3) in Random access and Sequential access are shown in Figure 4-1.

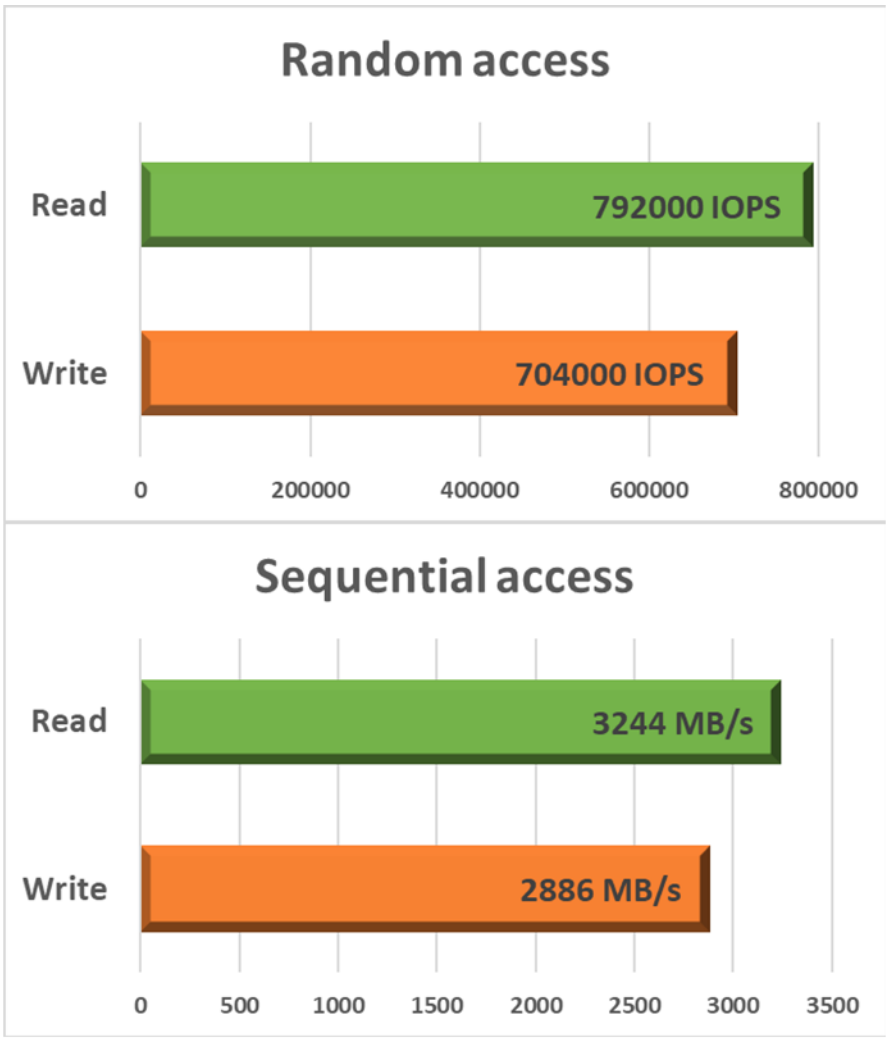

Figure 4-1 Test Performance of raNVMe-IP demo

The random access uses 4 KB size per command. The result shows 704,000 IOPS (2,883 MB/s) for Write command and 792,000 IOPS (3,244 MB/s) for Read command. While sequential mode shows almost same performance as random access – 2,886 MB/s for Write command and 3,244 MB/s for Read command. This SSD shows the good performance for both random access and sequential access. However, many SSDs show the less performance when running Random access, comparing to sequential access.

Note: Some SSDs require the large buffer to achieve better performance. Please contact to our sales for extending the internal buffer size of raNVMe-IP.

5 Revision History

Revision	Date	Description
1.4	8-Nov-22	Update raTestGen, Register map, and firmware
1.3	16-Jun-22	Update firmware and register type
1.2	12-Jan-21	Update Test result
1.1	16-Oct-20	Correct information
1.0	18-Aug-20	Initial Release