raNVMe-IP reference design manual

Rev1.0 16-Oct-20

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, 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 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/.

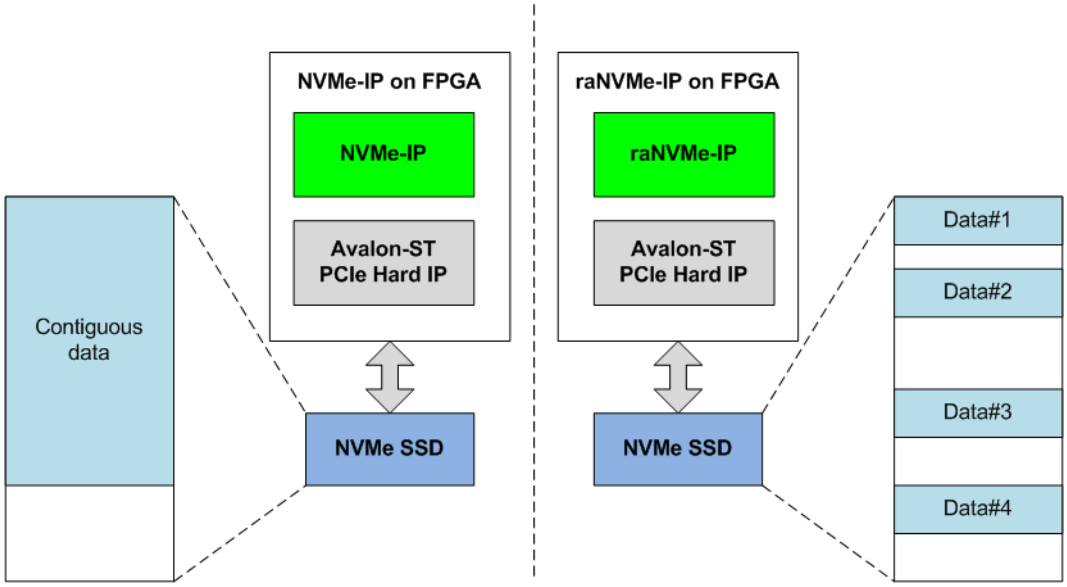

Figure 1-1 NVMe-IP and raNVMe-IP comparison

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 data streaming and the user can send only one command to NVMe-IP at a time. So, 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.

raNVMe-IP is the NVMe-IP which is modified to support up to 32 Write or Read commands with 4 Kbyte data size at the same time. As shown in Figure 1-1, the data stored in the SSD when using raNVMe-IP can be scattered in the SSD while the data in the SSD is contiguous area when using NVMe-IP. Though raNVMe-IP can operate many commands with the different area, the user application can implement raNVMe-IP with assigning the sequential address for comparable with NVMe-IP. The result performance when running sequential addressing by using raNVMe-IP is reduced from NVMe-IP because of the smaller internal buffer size.

2 Hardware overview

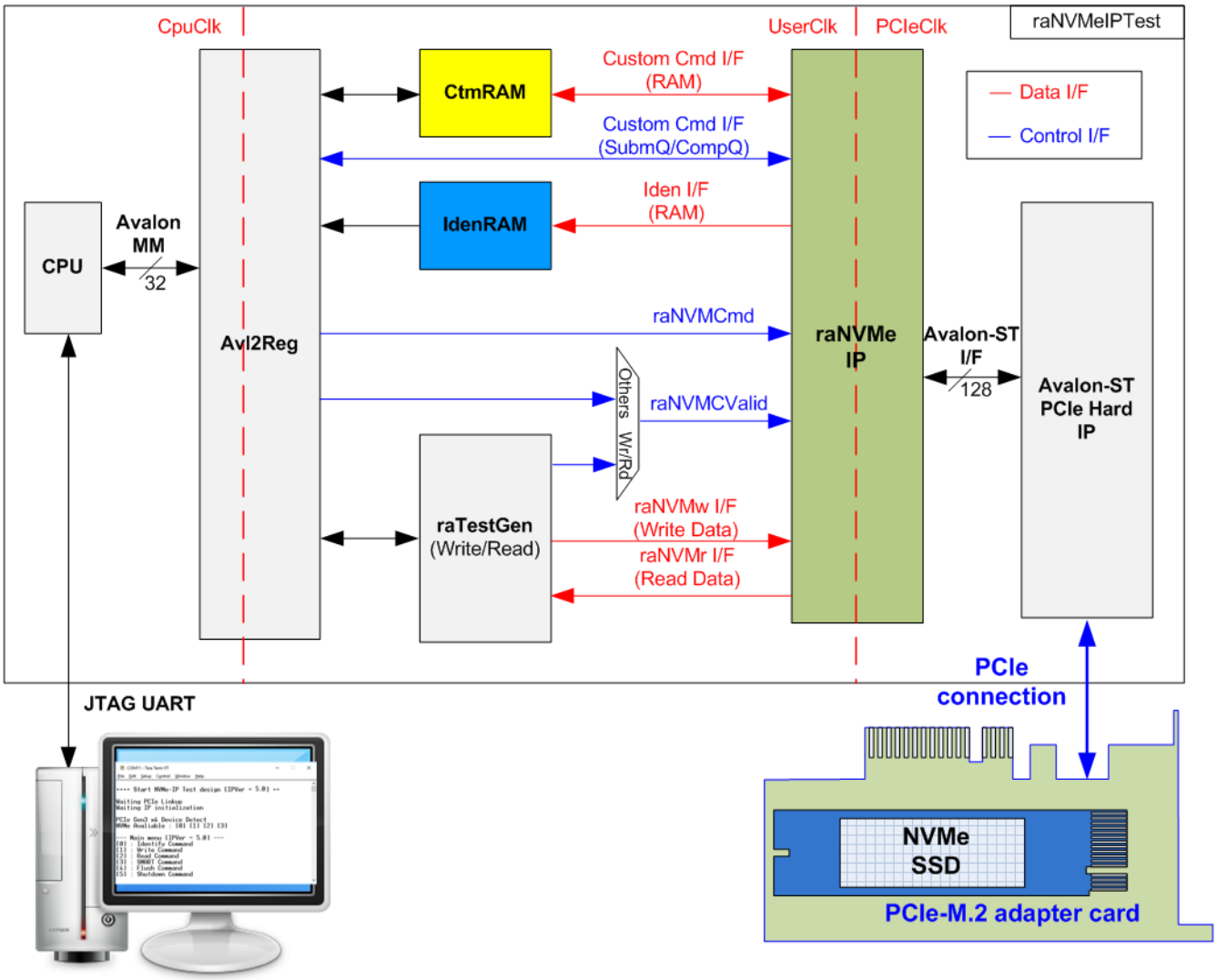

Figure 2-1 raNVMe-IP demo hardware

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 Avl2Reg).

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 Avl2Reg. When the command is Multiple mode (Write or Read), the command request is generated by raTestGen module. Also, the data streaming interface for transferring the data in Write or Read command is generated and verified by raTestGen module. The received data returned from SSD in SMART command and Identify command are stored to CtmRAM and IdenRAM respectively.

CPU and Avl2Reg are designed to interface with user via JTAG UART. The user can set command with the parameters on the console. Also, the current status of the test hardware can be displayed on the console for monitoring the test progress and test result.

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 is independent from the other hardware interface. UserClk is the example user clock domain which may be independent clock for running the user interface of raNVMe-IP. According to raNVMe-IP datasheet, clock frequency of UserClk must be more than or equal to PCIeClk. So, this reference design uses 275/280 MHz. PCIeClk is the clock output from PCIe hard IP to synchronous with data stream of 128-bit Avalon-ST interface. 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

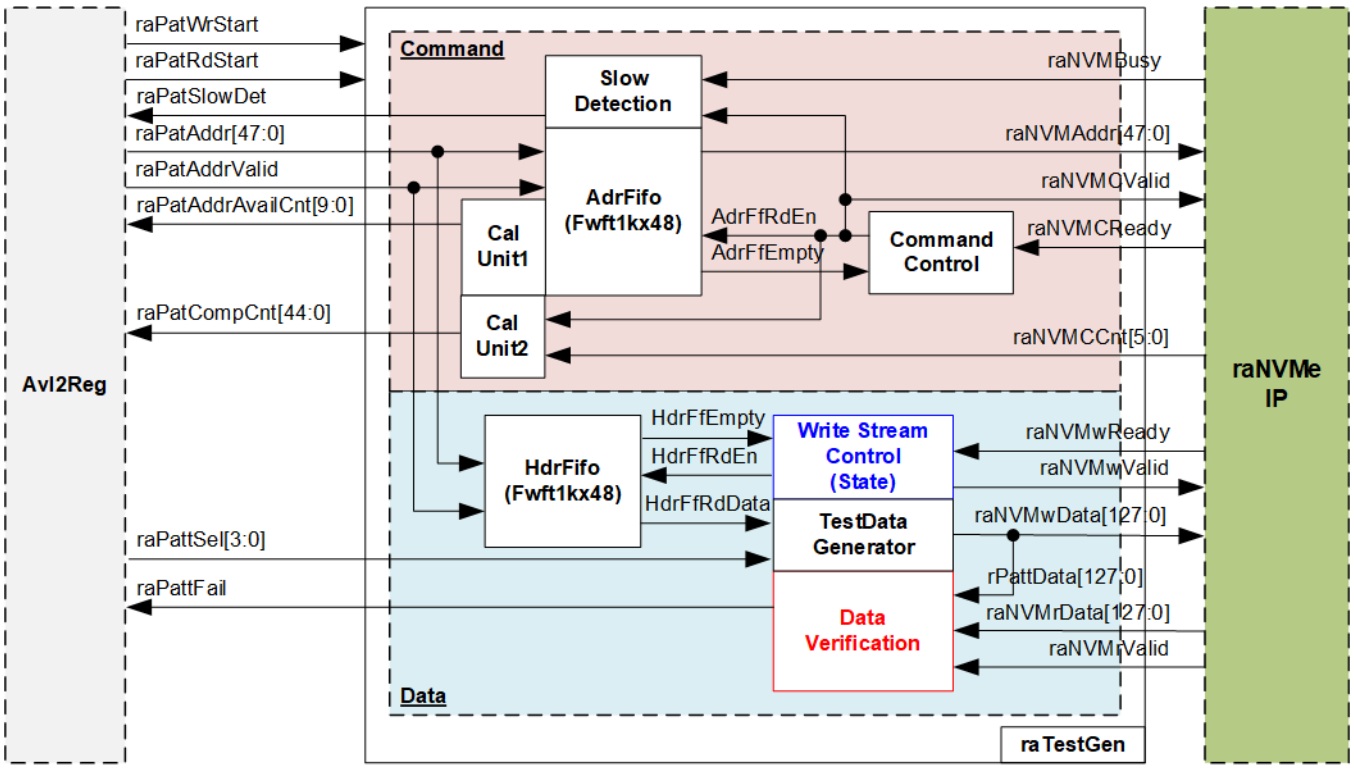

Figure 2-2 raTestGen Interface

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

There are the FIFOs inside Command and Data block, AdrFifo and HdrFifo, for storing 48-bit address (raPatAddr) which is created by CPU firmware via Avl2Reg. The FIFO depth is set to 1024 which is enough for raTestGen creating many requests to raNVMe-IP during CPU running other tasks such as displaying the test progress on the console.

Command

CPU writes the address for Write or Read command to AdrFifo directly. raPatAddrAvailCnt is the signal created by CalUnit1 for CPU checking the free size of AdrFifo before writing the FIFO. The equation of CalUnit1 to create raPatAddrAvailCnt is equal to 1024 (FIFO depth) – data_count of AdrFifo – 32 (the maximum different value between data_count of AdrFifo and HdrFifo). In the real system, HdrFifo is read in slower speed than AdrFifo because HdrFifo is read for creating 4 Kbyte test data in each Write command while AdrFifo is read for creating a request to raNVMe-IP. 4 Kbyte data uses at least 256 clock cycles for transferring while a command request uses only one clock cycle for operating. The maximum command request that can be sent to raNVMe-IP is 32 commands.

In addition, the demo designs to show the performance as IOPs during running Write or Read command on the console every second. So, CPU must read total number of completed commands from the test system, assigned by raPatCompCnt. raPatCompCnt value is calculated by finding the different value of total commands sent to raNVMe-IP (counted by using AdrFfRdEn signal) and total incomplete commands of raNVMe-IP (monitored by raNVMeCCnt signal).

To get the best performance for using raNVMe-IP, AdrFifo must always have the command for raNVMe-IP continuously operating without pausing time. Consequently, CPU firmware must write the data to AdrFifo at faster speed than CPU reading the data from AdrFifo. In the test system, raPatSlowDet is designed to be asserted to ‘1’ when AdrFifo is empty during running the test which means Write or Read performance result may be less than the maximum value from CPU task.

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’). Since AdrFIFO is FWFT type (First Word Fall Through), the read enable signal can be applied to be the valid signal of the read data. So, raNVMCValid is the same signal as the read enable of AdrFIFO (AdrFfRdEn).

Data

Write Stream Control in Data block is designed to assert the write enable (raNVMwValid) for sending 4 Kbyte data per command to raNVMe-IP. Before sending the data, HdrFifo must have the data and raNVMe-IP must be ready to receive the data. State machine is designed in this block to run the following step.

(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 is designed to count the data size of one command, called rDataCnt. In Read command, rDataCnt is increased by raNVMrValid to count total received data size. After finishing transferring 256 data (256x128-bit), continue to the next step.

(3) stDelay: This state is designed to delay the operation for one clock cycle to wait for raNVMwReady updated. After that, the state changes to stIdle to run the next command.

TestData Generator is designed to create 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 of one command is 4-Kbyte size which consists of 64-bit header data and the test pattern, selected by raPattSel.

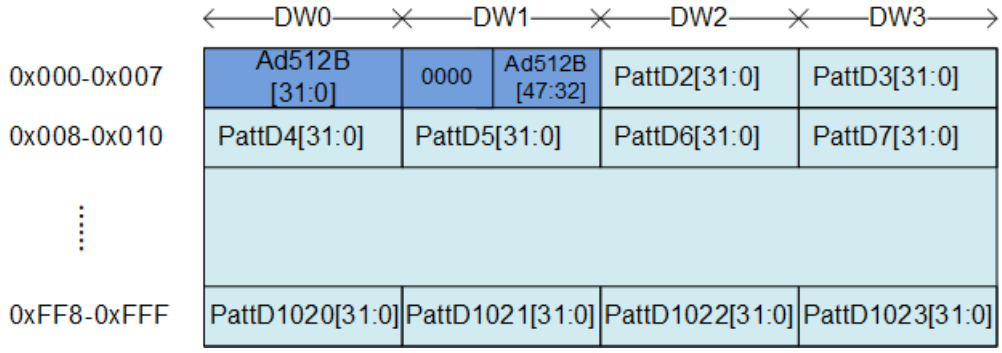

Figure 2-3 Test pattern format of 4096-byte data for Increment/Decrement/LFSR pattern

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.

In addition, the user can select test pattern to be all-zero or all-one data to show the best performance of some SSDs which has data compression algorithm in SSD controller. When the pattern is all-zero or all-one, there is no 64-bit header inserted to 4-Kbyte data.

Data verification is designed to verify the received data (raNVMrData) when raNVMrValid is asserted to ‘1’. The expected data is created by TestData Generator and 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.

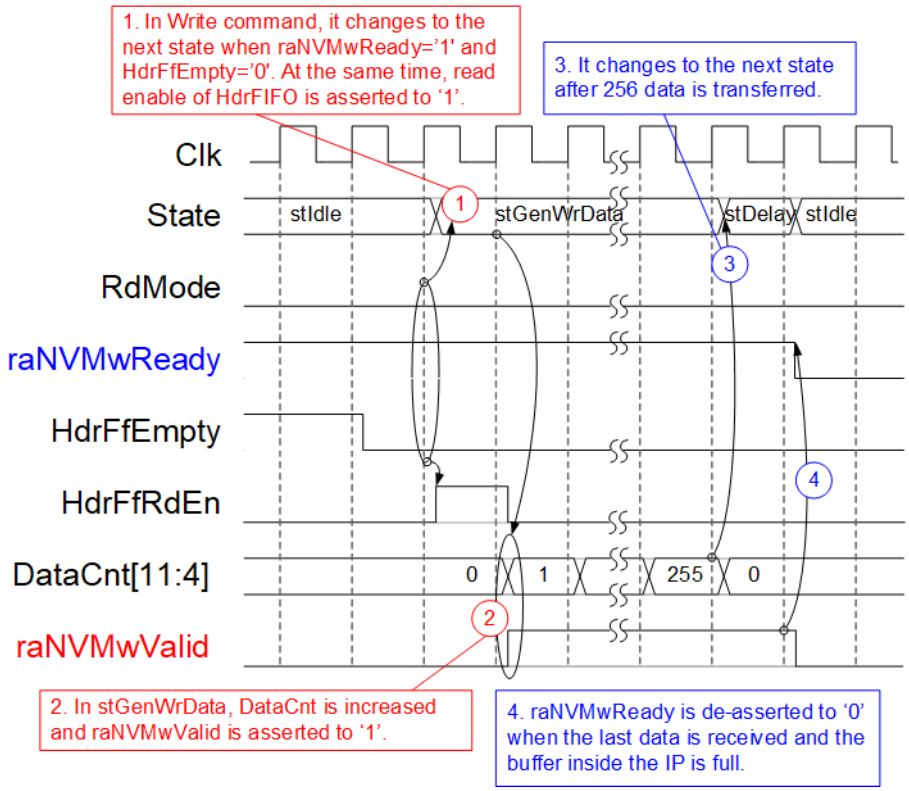

Figure 2-4 Timing diagram of raTestGen when running Write command

(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 the address that CPU sends the request and then changes 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 data size transferred to raNVMe-IP.
(3) After finishing transferring 256 data, monitored by DataCnt, it changes to stDelay.
(4) raNVMwReady is updated after finishing transferring the last data in the command. If the buffer in raNVMe-IP is full, raNVMewReady is de-asserted to '0' to pause data transmission of the next command. After that, go back to step (1) which is the Idle state.

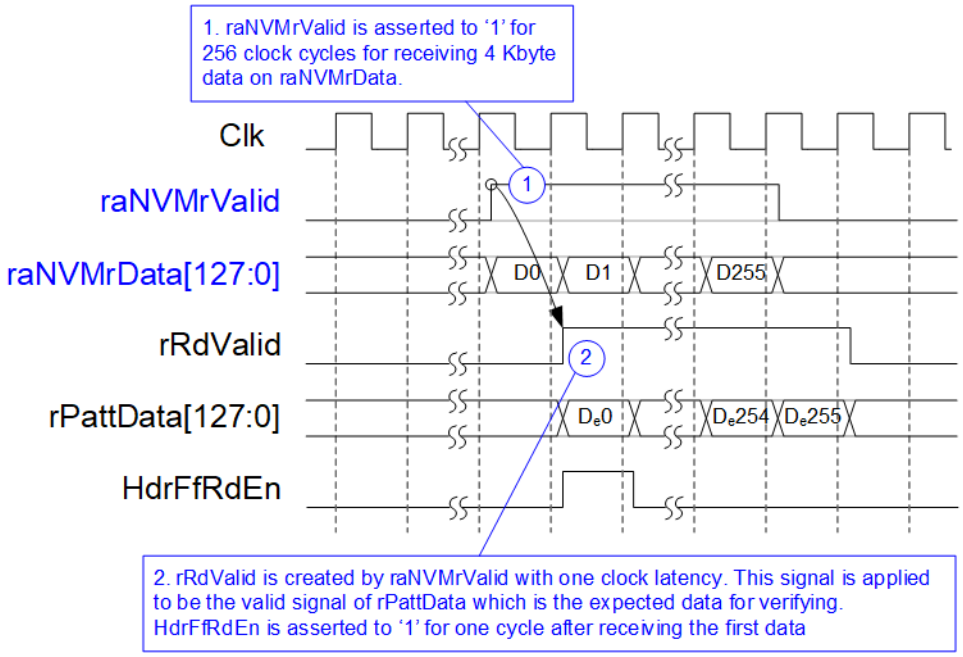

Figure 2-5 Timing diagram of raTestGen when running Read command

(1) During running Read command, raNVMe-IP returns 4Kbyte data continuously by asserting raNVMrValid to '1' for 256 clock cycles. The received data is available 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 for creating the header data of 4Kbyte data. 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) and applied to be data verification enable. Received data (raNVMrData) must be fed to one Flip-Flop to add one-clock latency for synchronizing with rPattData.

2.2 NVMe

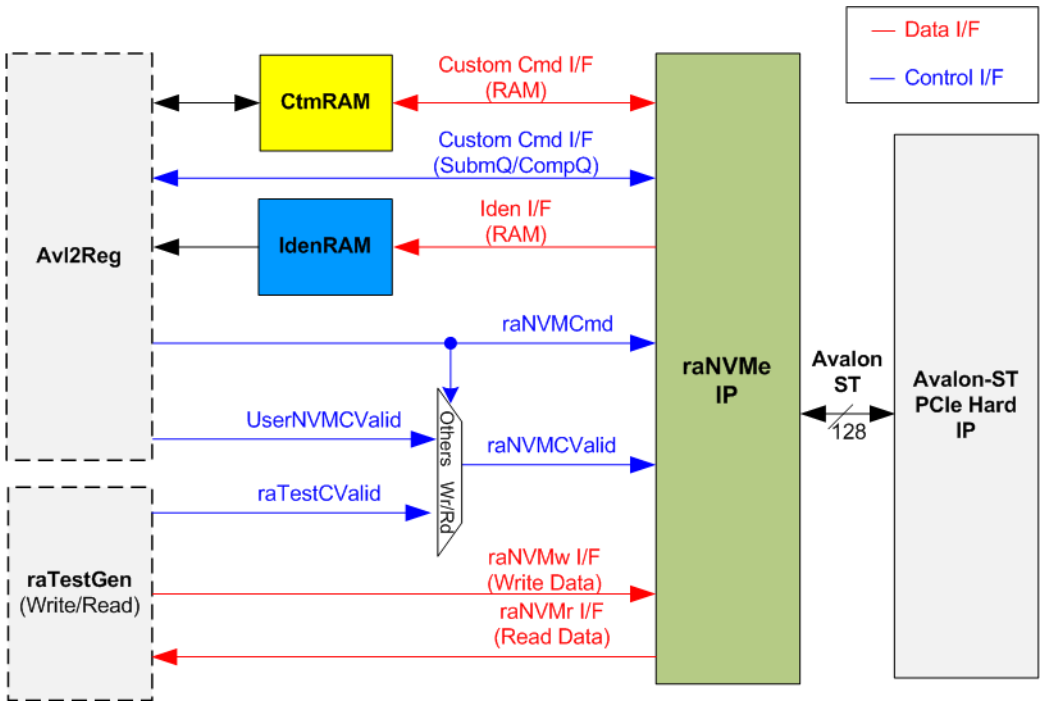

Figure 2-6 NVMe hardware

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 data interface transfers the data when the command needs data transferring.

There are two types of the command, i.e., Single command and Multiple command. The command (raNVMCmd) is set by CPU firmware via Avl2Reg, but the command request (raNVMCValid) is controlled by two sources, i.e., UserNVMCValid and raTestCValid. When the command is Single mode, the command request, called UserNVMCValid, is created by CPU firmware. When the command is Multiple mode, the command request is created by raTestGen, called raTestCValid. SMART command and Flush command are the custom command which needs to set additional parameters via Custom Cmd I/F. In the test design, these parameters are set by CPU firmware via Avl2Reg module.

There are four commands which has data transferring by using its own interface.
1) Custom Cmd I/F (RAM) sends SMART data to CtmRAM when running SMART command.
2) Iden I/F (RAM) sends Identify data to IdenRAM when running Identify command.
3) raNVMw I/F sends Write data from raTestGen when running Write command.
4) raNVMr I/F sends Read data from raNVMe-IP when running 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. 32 Write or Read commands with random addressing can be sent to raNVMe-IP. Six commands are supported in the standard 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_ranvmeip_datasheet_intel.pdf

2.2.2 Avalon-ST PCIe Hard IP

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

ArriaV Avalon-ST Interface for PCIe Solutions User Guide
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_a5_pcie_avst.pdf
Stratix V Avalon-ST Interface for PCIe Solutions User Guide
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_s5_pcie_avst.pdf
Intel Arria10 and Intel Cyclone 10 GX Avalon-ST Interface for PCIe Solutions User Guide
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_a10_pcie_avst.pdf

2.2.3 Two-port RAM

Two of 2-Port RAMs, CtmRAM and IdenRAM, store the returned data from Identify command and SMART command respectively.

IdenRAM has 8Kbyte size to store 8Kbyte data, output from Identify command. raNVMe-IP and Avl2Reg have the different data bus size, 128-bit on raNVMe-IP but 32-bit on Avl2Reg, so IdenRAM has the different bus size for connecting with two modules. Also, raNVMe-IP has double word enable to write only 32-bit data in some cases. The RAM setting on IP catalog of QuartusII supports the byte write enable, so one of double word enable is extended to be 4-bit write byte enable as shown in Figure 2-7.

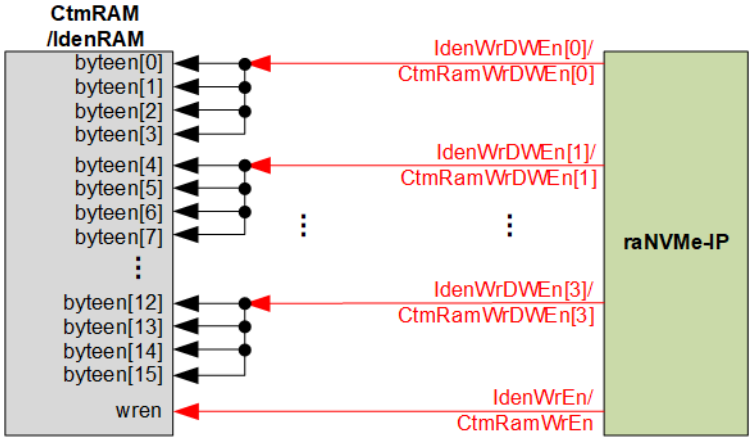

Figure 2-7 Byte write enable conversion logic

Bit[0], [1], [2] and [3] of WrDWEn are fed to be bit[3:0], bit[7:4], bit[11:8] and bit[15:12] 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 data width of both interfaces generated by IP catalog is 128-bit. Double-word enable is exteneded to be 4-bit write byte enable, 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 (one read port and one write port) is enough. The data size returned from SMART command is 512 bytes.

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. The test system of raNVMe-IP is connected with CPU as a peripheral on 32-bit Avalon-MM bus for CPU controlling and monitoring. CPU assigns the different base address and the address range to each peripheral for accessing one peripheral at a time.

In the reference design, the CPU system is built with one additional peripheral to access the test logic. The base address and the range for accessing the test logic are defined in the CPU system. So, the hardware logic must be designed to support Avalon-MM bus standard for CPU writing and reading. Avl2Reg module is designed to connect with the CPU system as shown in Figure 2-8.

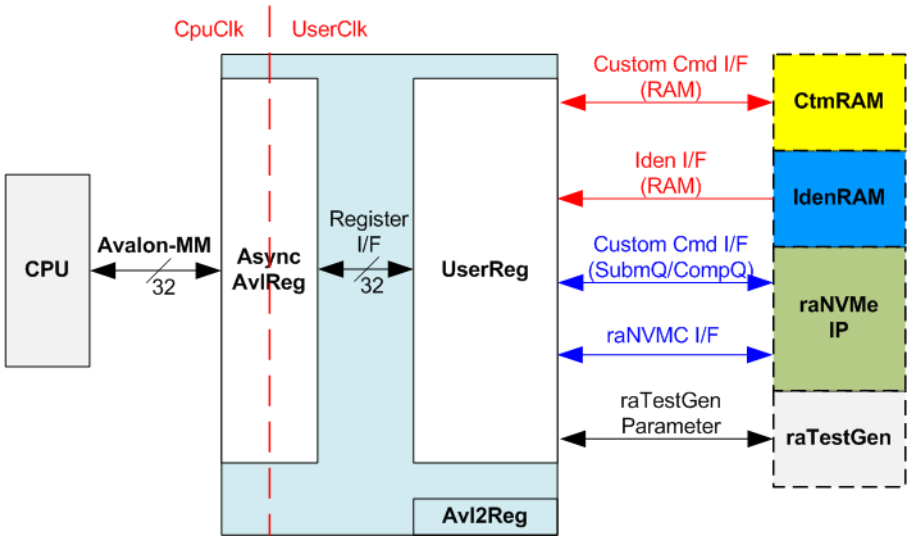

Figure 2-8 CPU and peripherals hardware

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 crossing between CpuClk domain and UserClk domain

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

2.3.1 AsyncAvlReg

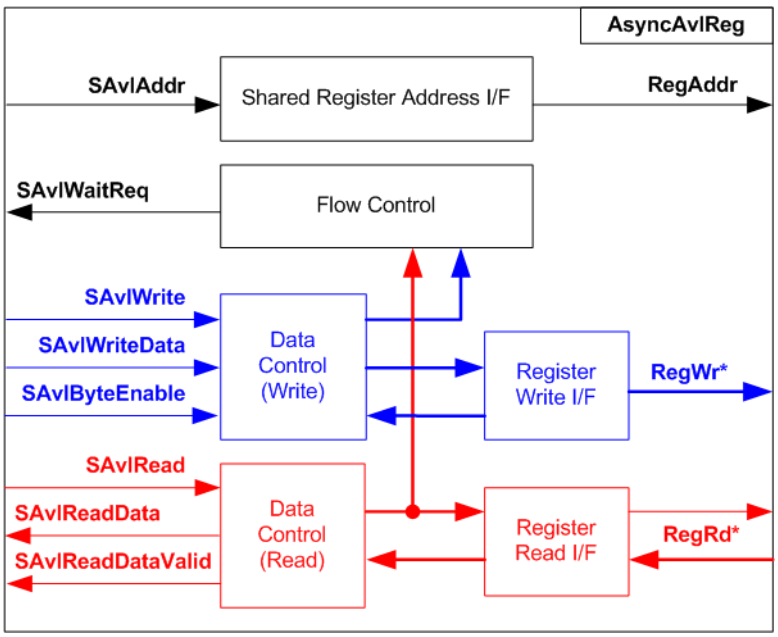

Figure 2-9 AsyncAvlReg Interface

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 is 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-crossing registers. In the same way, Read control I/F are latched and transferred to be Read register interface with clock-crossing registers. After that, the returned data from Register Read I/F is transferred to Avalon-MM bus by using clock-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 2-10.

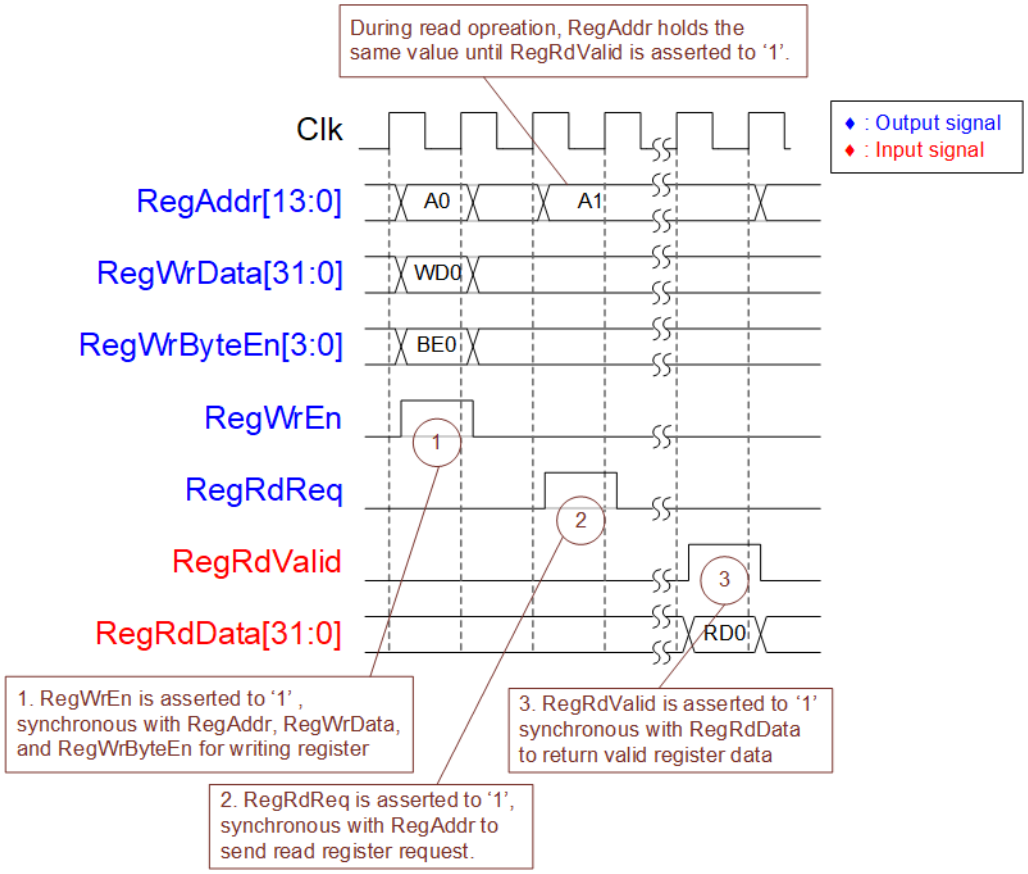

Figure 2-10 Register interface timing diagram

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, 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 signal to start the read transaction. During read operation, the address value (RegAddr) does not change the value until RegRdValid is asserted to '1'. So, 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

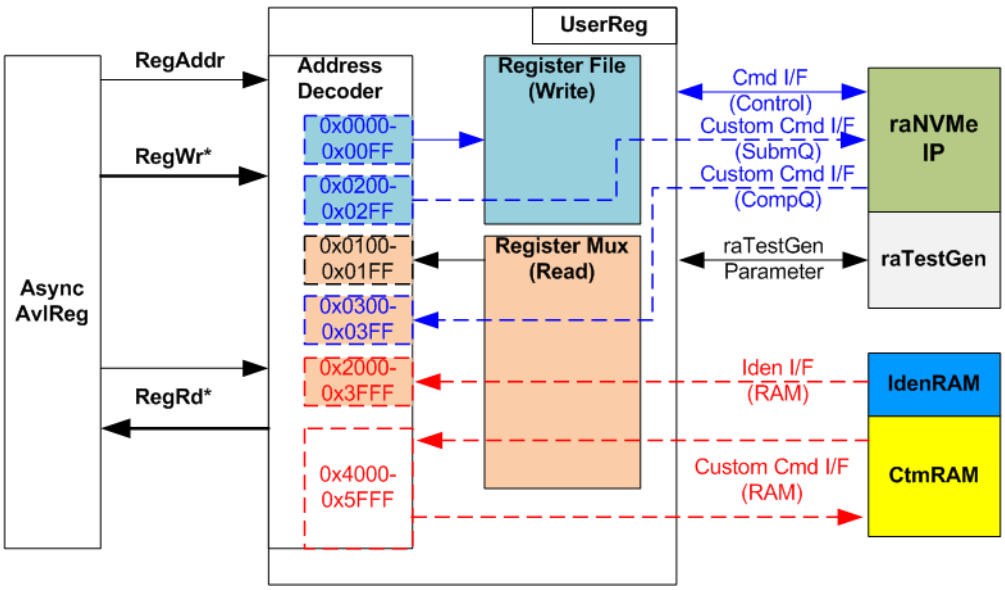

Figure 2-11 UserReg Interface

The address range to map to 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 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. The register file inside UserReg is 32-bit bus size, so write byte enable (RegWrByteEn) is not used. To write hardware registers, the CPU must use 32-bit pointer to place 32-bit valid value on the write data bus.

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 data. Next the address decoder uses the upper bit to select the read data from each area for returning to CPU. Totally, the latency of read data is equal to two clock cycles, so 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_REG)	[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_REG)	[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_REG)	[2:0]: Input to be user command (UserCmd of raNVMe-IP) "000": Identify, "001": Shutdown, "010": Write SSD, "011": Read SSD, "100": SMART, "110": Flush, "101"/"111": Reserved When this register is written with Single command (not Write/Read), the new command request (raNVMCValid) is asserted to raNVMe-IP. Otherwise, start flag for Write or Read command is asserted to raTestGen. After that, the command request (raNVMCValid) for Multiple command is asserted.
BA+0x0014	Test Pattern Reg (PATTSEL_REG)	[2:0]: Select test pattern. "000"-Increment, "001"-Decrement, "010"-All 0, "011"-All 1, "100"-LFSR. [3]: Verification enable. '0' -No verification, '1'-Enable verification.
BA+0x0020	NVMe Timeout Reg (NVMTIMEOUT_REG)	[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_REG)	[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. [12:8]: Mapped to raNVMCId of raNVMe-IP to show current command ID. [20:16]: Mapped to raNVMDId of raNVMe-IP to show command ID which currently transfers data on Data stream interface. [29:24]: 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_REG)	[31:0]: Mapped to LBASize[31:0] of raNVMe-IP
BA+0x0108	Total disk size (High) Reg (LBASIZEH_REG)	[15:0]: Mapped to LBASize[47:32] of raNVMe-IP
BA+0x010C	User Error Type Reg (USRERRTYPE_REG)	[31:0]: Mapped to UserErrorType[31:0] of raNVMe-IP to show error status
BA+0x0110	PCIe Status Reg (PCISTS_REG)	[0]: PCIe linkup status from PCIe hard IP ('0': No linkup, '1': linkup) [3:2]: PCIe link speed from PCIe hard IP ("00": Not linkup, "01": PCIe Gen1, "10": PCIe Gen2, "11": PCIe Gen3) [7:4]: PCIe link width status from PCIe hard IP ("0001": 1-lane, "0010": 2-lane, "0100": 4-lane, "1000": 8-lane) [12:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in Avalon-ST PCIe Hard IP datasheet
BA+0x0114	NVMe CAP Reg (NVMCAP_REG)	[31:0]: Mapped to NVMeCAPReg[31:0] of raNVMe-IP
BA+0x0118	Admin Completion Status Reg (ADMCOMPSTS_REG)	[15:0]: Mapped to AdmCompStatus[15:0] of raNVMe-IP to show status of Admin completion
BA+0x011C	IO Completion Status Reg (IOCOMPSTS_REG)	[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_REG))	[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	Adr FIFO Remain Cnt Reg (AFIFOREMCNT_REG)	[9:0]: Remaining size of AdrFIFO in raTestGen (raPatAddrAvailCnt of raTestGen)
BA+0x0128	Slow Transfer detect Reg (SLOWDET_REG)	[0]: Slow transfer is found 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 dropped from CPU task. This flag is auto-cleared when CPU sets USRCMD_REG=Write or Read command.
BA+0x0130	Expected value Word0 Reg (EXPPATW0_REG)	[31:0]: Bit[31:0] of the expected data at the 1st failure data in Read command
BA+0x0134	Expected value Word1 Reg (EXPPATW1_REG)	[31:0]: Bit[63:32] of the expected data at the 1st failure data in Read command
BA+0x0138	Expected value Word2 Reg (EXPPATW2_REG)	[31:0]: Bit[95:64] of the expected data at the 1st failure data in Read command
BA+0x013C	Expected value Word3 Reg (EXPPATW3_REG)	[31:0]: Bit[127:96] of the expected data at the 1st failure data in Read command
BA+0x0140	Read value Word0 Reg (RDPATW0_REG)	[31:0]: Bit[31:0] of the read data at the 1st failure data in Read command
BA+0x0144	Read value Word1 Reg (RDPATW1_REG)	[31:0]: Bit[63:32] of the read data at the 1st failure data in Read command
BA+0x0148	Read value Word2 Reg (RDPATW2_REG)	[31:0]: Bit[95:64] of the read data at the 1st failure data in Read command
BA+0x014C	Read value Word3 Reg (RDPATW3_REG)	[31:0]: Bit[127:96] of the read data at the 1st failure data in Read command
BA+0x0150	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+0x0154	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+0x0158	Completed Count (Low) Reg (CMDCMPCNTL_REG)	[31:0]: Bit[31:0] of the completed command count in raTestGen (raPatCompCnt of raTestGen)
BA+0x015C	Completed Count (High) Reg (CMDCMPCNTH_REG)	[12:0]: Bit[44:32] of the completed command count in raTestGen (raPatCompCnt of raTestGen)
Other interfaces (Custom command of raNVMe-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 raNVMe-IP. 0x200: DW0, 0x204: DW1, …, 0x23C: DW15
BA+0x0300 – BA+0x030F Rd	Custom Completion Queue Reg (CTMCOMPQ_REG)	[31:0]: CtmCompDW0-DW3 output from raNVMe-IP. 0x300: DW0, 0x304: DW1, …, 0x30C: DW3
BA+0x0800 Rd	IP Version Reg (IPVERSION_REG)	[31:0]: Mapped to IPVersion[31:0] of raNVMe-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 (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 (PCISTS_REG[0]='1').
3) CPU waits until raNVMe-IP completes initialization process (USRSTS_REG[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 PCISTS_REG[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_REG="000". Next, Test logic generates command and asserts command request to raNVMe-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' after finishing operating the command. After the command is completed, 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_REG[31:0]. Finally, the process is stopped.

3) After busy flag (USRSTS_REG[0]) is de-asserted to '0', CPU displays the information decoded from IdenRAM (IDENCTRL_REG) such as SSD model name and the information from raNVMe-IP output such as SSD capacity (LBASIZEH/L_REG). Also, CPU sets the polynomial for creating LFSR test pattern following SSD capacity.

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 the 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_REG.
3) Set USRCMD_REG[2:0]=”010” for Write command or “011” for Read command.
4) Calculate the number of the address request by reading the remaining buffer size to store the address request from AFIFOREMCNT_REG[9:0] and comparing with the remaining total transfer length. Use the less value for sending the request to the hardware in the current loop.
5) Repeat this step to send the address to the hardware by setting USRADRL/H_REG[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_REG[2:1]. Display the error message when some bits are asserted to ‘1’.

   Bit[1] is asserted when IP error is detected. The process is hanged up when this error is found.
   Bit[2] is asserted when data verification is enabled in Read command and data failure is found. The verification error message is displayed. In this condition, CPU is still running until the operation is done or user inputs any keys to cancel operation
7) The current transfer size which read from CMDCMPCNTL/H_REG is displayed as percentage every second.
8) Go to the next step if remaining total transfer size is equal to 0. Otherwise, go to step 4) to calculate the transfer size for the next loop.
9) CPU waits until raNVMe-IP finishes the operation (USRSTS_REG[0]=’0’). Display the error message if some errors are found.
10) Read SLOWDET_REG and print the message that the test performance is limited by CPU firmware when the flag is asserted. After that, display the test results on the console: 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_REG) to be SMART command value.
2) Set USRCMD_REG[2:0]=”100”. Next, Test logic generates command and asserts the request to raNVMe-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’ after finishing operating the command. If the command is completed, 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_REG[31:0]. Finally, the process is stopped.

4) After busy flag (USRSTS_REG[0]) is de-asserted to ‘0’, CPU displays some information decoded from CtmRAM (CTMRAM_REG) such as 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_REG) to be Flush command value.
2) Set USRCMD_REG[2:0]=”110”. Next, Test logic generates command and asserts the request to raNVMe-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’ after finishing operating the command. If the command is completed, the CPU goes back 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_REG[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_REG[2:0]=”001”. Next, Test logic generates command and asserts the request to raNVMe-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’ 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_REG[31:0]. Finally, the process is stopped.

3) After Shutdown command, the SSD and raNVMe-IP change to inactive status. So, 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
Return value	nextaddr
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 flush_ctmnvm(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the raNVMe-IP
Description	Set Flush command to CTMSUBMQ_REG and call exec_ctm function to operate 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).

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.

void show_pciestat(void)	
Parameters	None
Return value	None
Description	Read PCISTS_REG 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 CMDCMPCNT_REG 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_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 a limit (500 PB), the overflow message is displayed instead.

void show_vererr(void)	
Parameters	None
Return value	None
Description	Read RDFAILNOL/H_REG (error byte address), EXPPATW0-W3_REG (expected value) and RDPATW0-W3_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 smart_ctmadm(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the raNVMe-IP
Description	Set SMART command to CTMSUBMQ_REG and call exec_ctm function to operate SMART command. Finally, decode and display SMART information on the console.

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 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 sequential mode

4 Example Test Result

The example test results when running demo system by using 512 GB Samsung 970 Pro and A10GX board (PCIe Gen3) in Random access and Sequential access are shown in Figure 4-1.

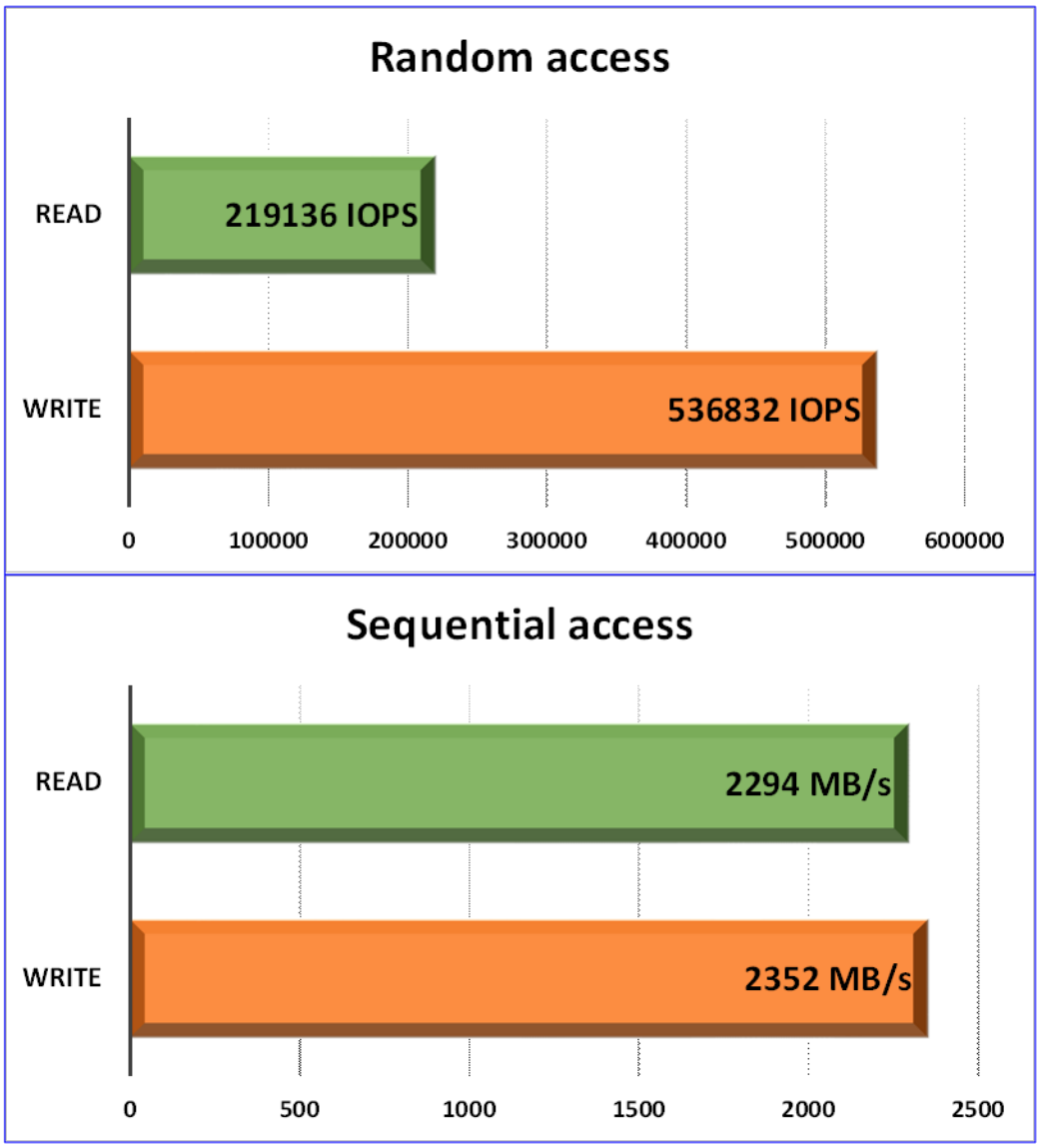

Figure 4-1 Test Performance of raNVMe-IP demo

In Random access by 4 KB size, Write performance is about 536,000 IOPS and Read performance is about 219,000 IOPS. In Sequential access, Write performance is about 2,300 MB/s and Read performance is about 2,200 MB/s. Write performance in Random and Sequential mode is not much different (536832 x 4096 = 2200 MB/s). Comparing raNVMe-IP performance with NVMe-IP, write performance of raNVMe-IP is slightly reduced which is caused from smaller buffer size.

Read performance in Random mode is much dropped from Sequential mode, matched with SSD specification. Also, Read performance in Sequential mode from raNVMe-IP is reduced from NVMe-IP because the buffer size in raNVMe-IP is a half of the buffer size in NVMe-IP.

5 Revision History

Revision	Date	Description
1.0	16-Oct-20	Initial Release