NVMe-IP reference design manual

Rev3.8 17-Mar-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 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/.

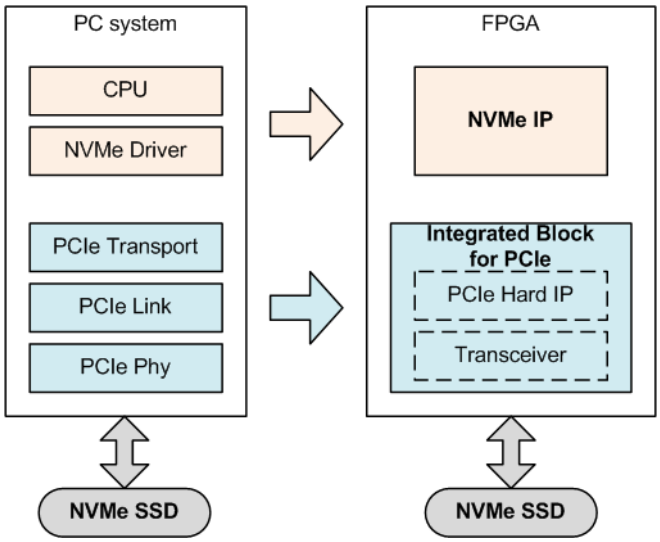

Figure 1-1 NVMe protocol layer

To access NVMe SSD, the general system implements NVMe driver running on the processor, as shown in the left side of Figure 1-1. The physical connection of NVMe standard is PCIe connector which is one-to-one type, so one PCIe host connects to one PCIe device without using switch. NVMe-IP implements NVMe driver to access NVMe SSD by using pure-hardware logic. The user can access NVMe SSD without including any processor and driver but using NVMe IP in FPGA board. Using pure-hardware logic for NVMe host controller reduces the overhead time for software-hardware handshake, so NVMe-IP can show very high performance for writing and reading with NVMe SSD.

2 Hardware overview

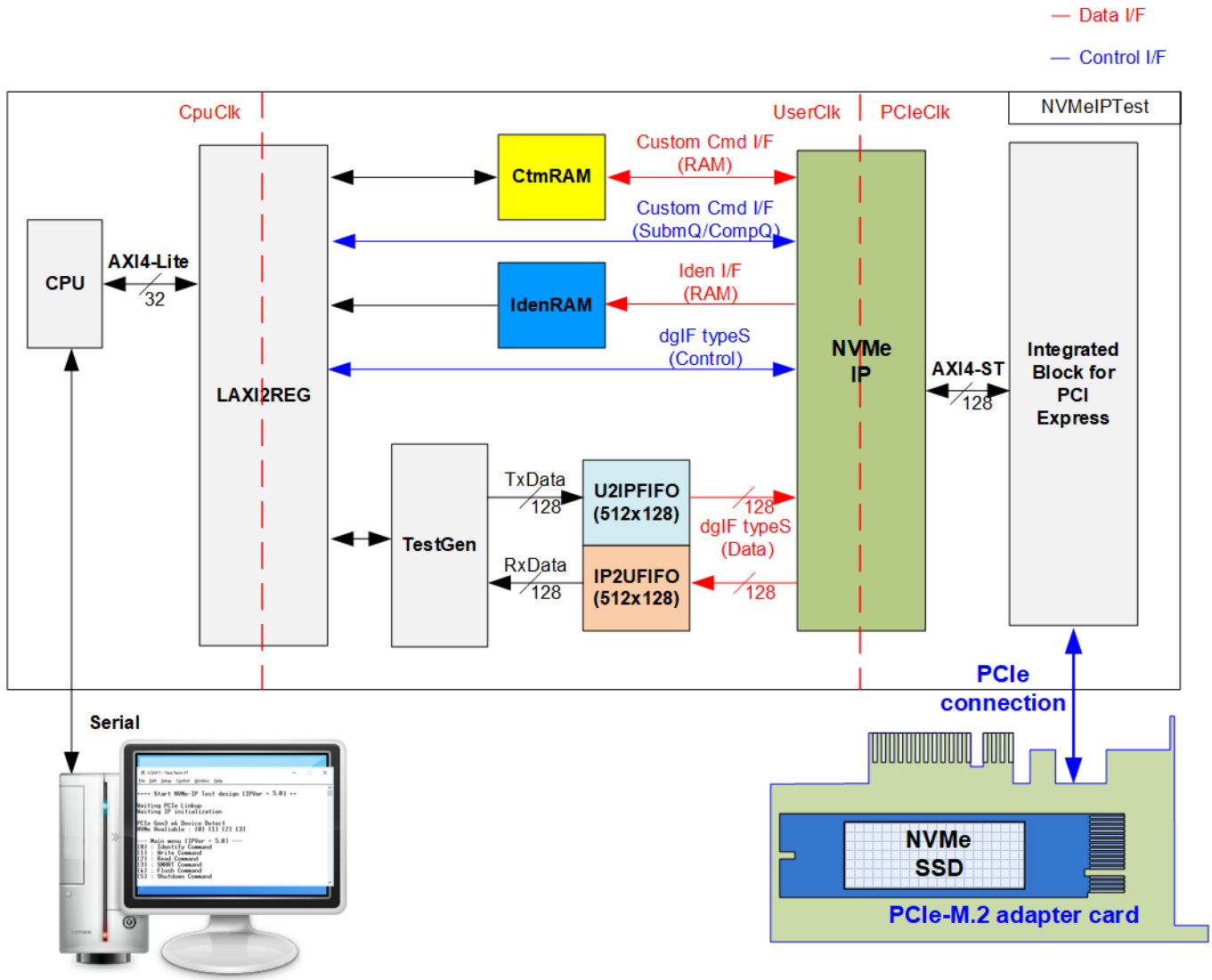

Figure 2-1 NVMe-IP demo hardware

Following the function of each module, all hardware modules inside the test system are divided to three parts, i.e., test function (TestGen), NVMe function (CtmRAM, IdenRAM, U2IPFIFO, IP2UFIFO, NVMe-IP and PCIe block) and CPU system (CPU and LAxi2Reg).

TestGen is the test logic to generate test data stream for NVMe-IP via U2IPFIFO and read data stream output from NVMe-IP via IP2UFIFO for verification. NVMe includes the NVMe IP and the PCIe hard IP (Integrated Block for PCI Express) for accessing NVMe SSD directly without PCIe switch. CPU and LAxi2Reg are designed to interface with user via Serial interface. User can set command and the test parameters on Serial console. Also, the current status of the test hardware is monitored by user on Serial console. The CPU firmware must be implemented to control the flow for operating each command.

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

There are three clock domains displayed in Figure 2-1, i.e., CpuClk, UserClk, and PCIeClk. CpuClk is the clock domain of CPU and its peripherals. This clock must be stable clock and can be different clock domain from the other hardwares. UserClk is the user clock domain for running the user interface of NVMe-IP, RAM, FIFO, and TestGen. According to NVMe-IP datasheet, clock frequency of UserClk must be more than or equal to PCIeClk. This reference design uses 275/280 MHz for PCIe Gen3 or 200 MHz for PCIe Gen2. 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. Otherwise, PCIeClk frequency is equal to 125 MHz for PCIe Gen2.

More details of the hardware are described as follows.

2.1 TestGen

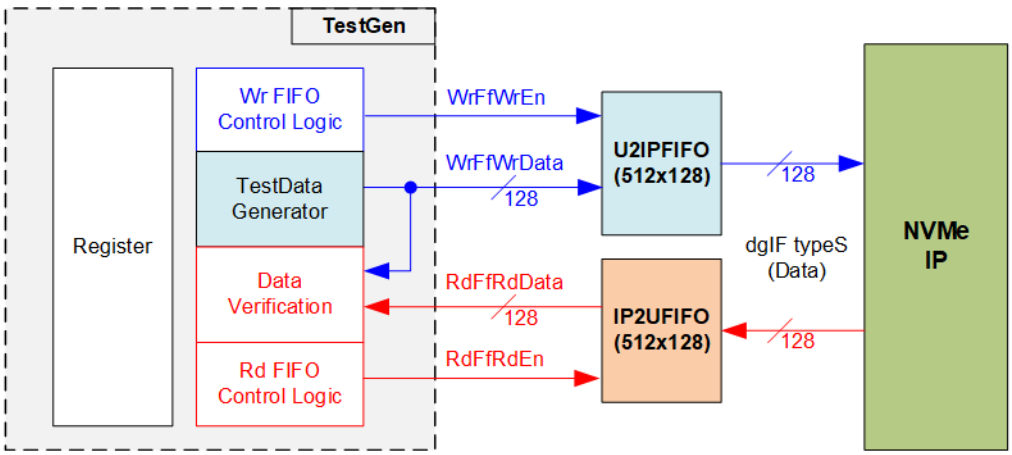

Figure 2-2 TestGen interface

TestGen module handles the data interface of NVMe-IP for transferring the data in Write and Read command. In Write command, TestGen sends 128-bit test data to NVMe-IP via U2IPFIFO. In Read command, the test data is fed from IP2UFIFO to compare with the expected value for data verification. Data bandwidth of TestGen is matched to NVMe-IP by running at the same clock and the same data bus size. Control logic asserts Write enable or Read enable to '1' for writing or reading the FIFO when FIFO is ready. So, NVMe-IP can transfer data with U2IPFIFO and IP2UFIFO without waiting data ready. As a result, the test logic shows the best performance to write and read data with the SSD through NVMe-IP.

Register file in the TestGen receives test parameters from user, i.e., total transfer size, transfer direction, verification enable, and test pattern selector. The internal logic includes the counter to control total transfer size of test data. The details of hardware logic of TestGen are shown in Figure 2-3.

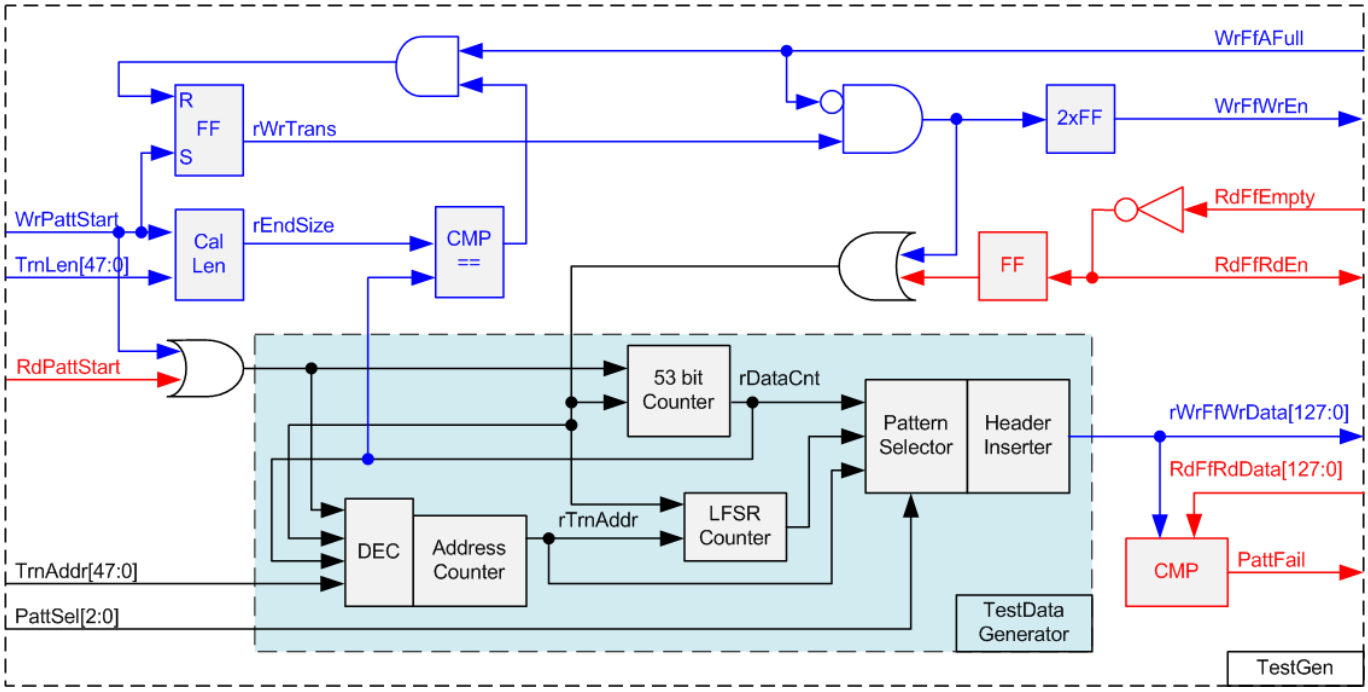

Figure 2-3 TestGen hardware

As shown in the right side of Figure 2-3, flow control signals of FIFO are WrFfAFull and RdFfEmpty. When FIFO is almost full during write operation (WrFfAFull='1'), WrFfWrEn is de-asserted to '0' to pause data sending to FIFO. For read operation, when FIFO has data (RdFfEmpty='0'), the logic reads data from FIFO to compare with the expected data by asserting RdFfRdEn to '1'.

The logic in the left side of Figure 2-3 is designed to count transfer size. When total data count is equal to the end size, set by user, write enable or read enable of FIFO is de-asserted to '0'. The lower side of Figure 2-3 shows the details to generate test data for writing to FIFO or verifying data from FIFO. There are five patterns to generate, i.e., all-zero, all-one, 32-bit incremental data, 32-bit decremental data, and LFSR counter, selected by Pattern Selector. When creating all-zero or all-one pattern, every bit of data is fixed zero or one respectively. While other patterns are designed by separating the data as two parts to create unique test data in every 512-byte data, as shown in Figure 2-4.

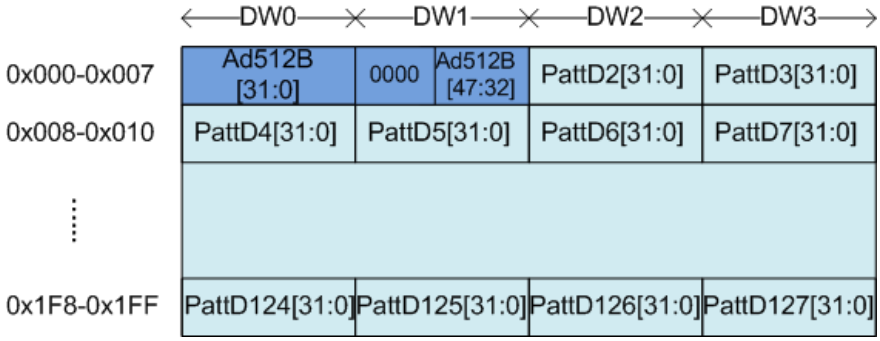

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

512-byte data consists of 64-bit header in Dword#0 and Dword#1 and the test data in remaining words of 512-byte data (Dword#2 – Dword#127). The header is created by using the address in 512-byte unit, Address counter block. The initial value of the address counter is set by user and the value is increased when finishing transferring 512-byte data. Remaining Dwords (DW#2 – DW#127) depends on pattern selector which may be 32-bit incremental data, 32-bit decremental data, or LFSR counter. 32-bit incremental data is designed by using 53-bit counter. The decremental data can be designed by connecting NOT logic to incremental data. The LFSR pattern is designed by using LFSR counter. The equation of LFSR is x^31 + x^21 + x + 1. Data bus size of TestGen is 128-bit, so four 32-bit LFSR data must be generated within one clock by using look-ahead logic.

Test data is fed to be write data to the FIFO or the expected data for comparing with the read data from FIFO. Fail flag is asserted to '1' when data verification is failed. The example of timing diagram to write data to FIFO is shown as follows.

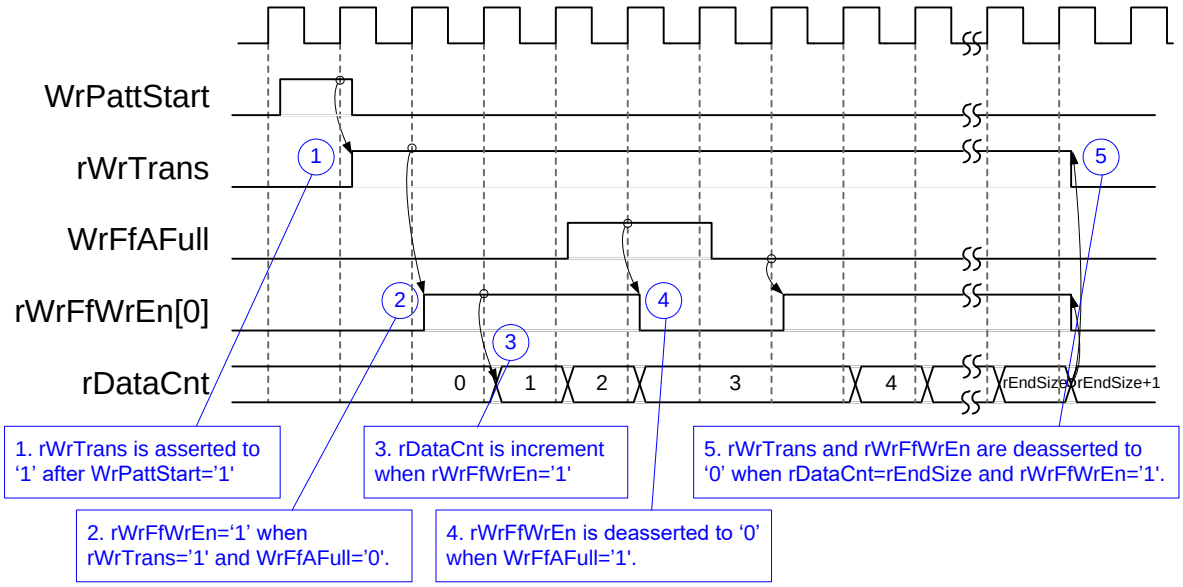

Figure 2-5 Timing diagram of Write operation in TestGen

1) WrPattStart is asserted to '1' for one clock cycle when user sets the register to start write operation. In the next clock, rWrTrans is asserted to '1' to enable the control logic for generating write enable to FIFO.
2) Write enable to FIFO (rWrFfWrEn) is asserted to '1' when two conditions are met. First, rWrTrans must be asserted to '1' during the write operation being active. Second, the FIFO must not be full by monitoring WrFfAFull='0'.
3) The write enable is fed back to be counter enable to count total amount of data 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 set value, rWrTrans is de-asserted to '0'. At the same time, rWrFfWrEn is also de-asserted to '0' to finish generating data.

For read timing diagram, read enable of FIFO is controlled by empty flag of FIFO. Comparing to write enable, the read enable signal is not stopped by total data count and not started by start flag. When the read enable is asserted to '1', the data counter and the address counter are increased for counting total amount of data and generating the header of expect value respectively.

2.2 NVMe

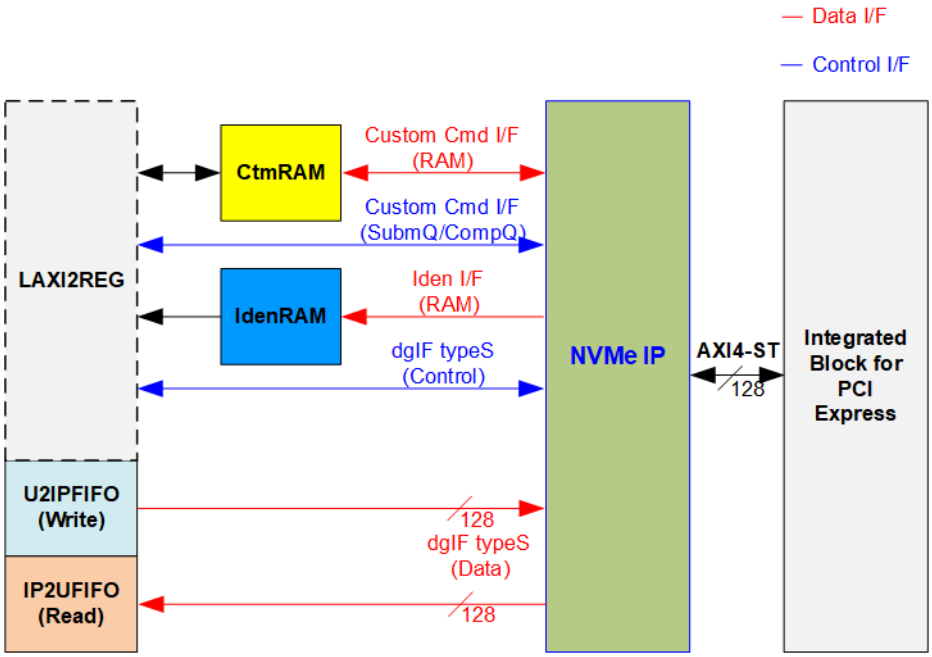

Figure 2-6 NVMe hardware

Figure 2-6 shows the example to interface NVMe-IP in the reference design. The user interface of NVMe-IP consists of control interface and data interface. The control interface receives the command and the parameters from custom command interface or dgIF typeS, depending on the command value. Custom command interface is used when operating SMART command or Flush command.

The data interface of NVMe-IP has four interfaces, i.e., custom command RAM interface, Identify interface, FIFO input interface (dgIF typeS), and FIFO output interface (dgIF typeS). Data bus width of all data interface is 128-bit. The custom command RAM interface is bi-directional interface while the other interfaces are one directional interface. In the reference design, the custom command RAM interface is used for transferring one direction only when NVMe-IP sends SMART data to LAxi2Reg.

2.2.1 NVMe-IP

The NVMe-IP implements NVMe protocol of the host side to access one NVMe SSD directly without PCIe switch connection. The NVMe-IP supports six commands, i.e., Write, Read, Identify, Shutdown, SMART, and Flush. NVMe-IP can connect to the PCIe hard IP directly. More details of NVMe-IP are described in datasheet.
https://dgway.com/products/IP/NVMe-IP/dg_nvme_ip_data_sheet_en.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.
PG054: 7 Series FPGAs Integrated Block for PCI Express
PG023: Virtex-7 FPGA Gen3 Integrated Block for PCI Express
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 data from Identify command and SMART command respectively. IdenRAM has 8-Kbyte size to store 8-Kbyte data, output from Identify command. NVMe-IP and LAxi2Reg have the different data bus size, 128-bit on NVMe-IP but 32-bit on LAxi2Reg. Therefore, IdenRAM has the different bus size on Write interface and Read interface for connecting with two modules. Also, NVMe-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. The small logic to convert double word enable to be write byte enable is designed as shown in Figure 2-7.

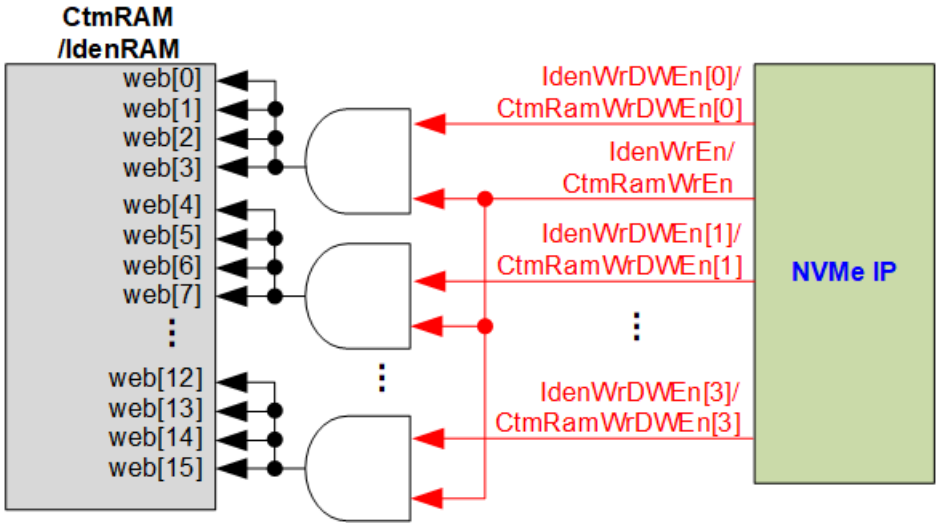

Figure 2-7 Byte write enable conversion logic

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. The test system of NVMe-IP is connected with CPU as a peripheral on 32-bit AXI4-Lite 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. So, the hardware logic must be designed to support AXI4-Lite bus standard for CPU writing and reading. LAxi2Reg module is designed to connect with the CPU system as shown in Figure 2-8.

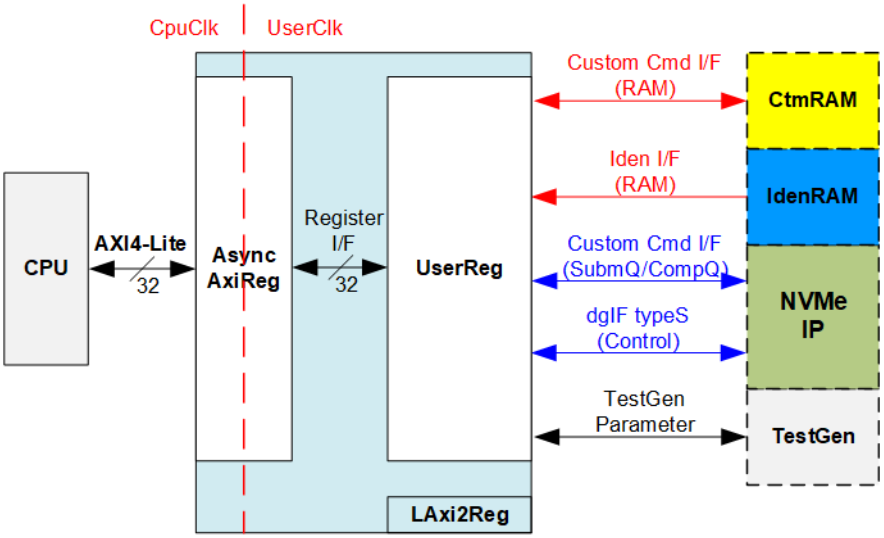

Figure 2-8 CPU and peripherals hardware

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. Additionally, 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 other modules in the test system, i.e., CtmRAM, IdenRAM, NVMe-IP, and TestGen. More details of AsyncAxiReg and UserReg are described as follows.

2.3.1 AsyncAxiReg

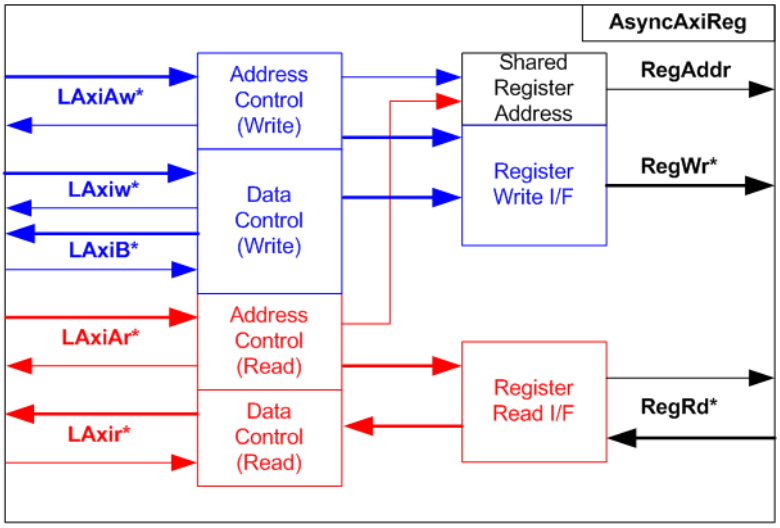

Figure 2-9 AsyncAxiReg Interface

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 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.

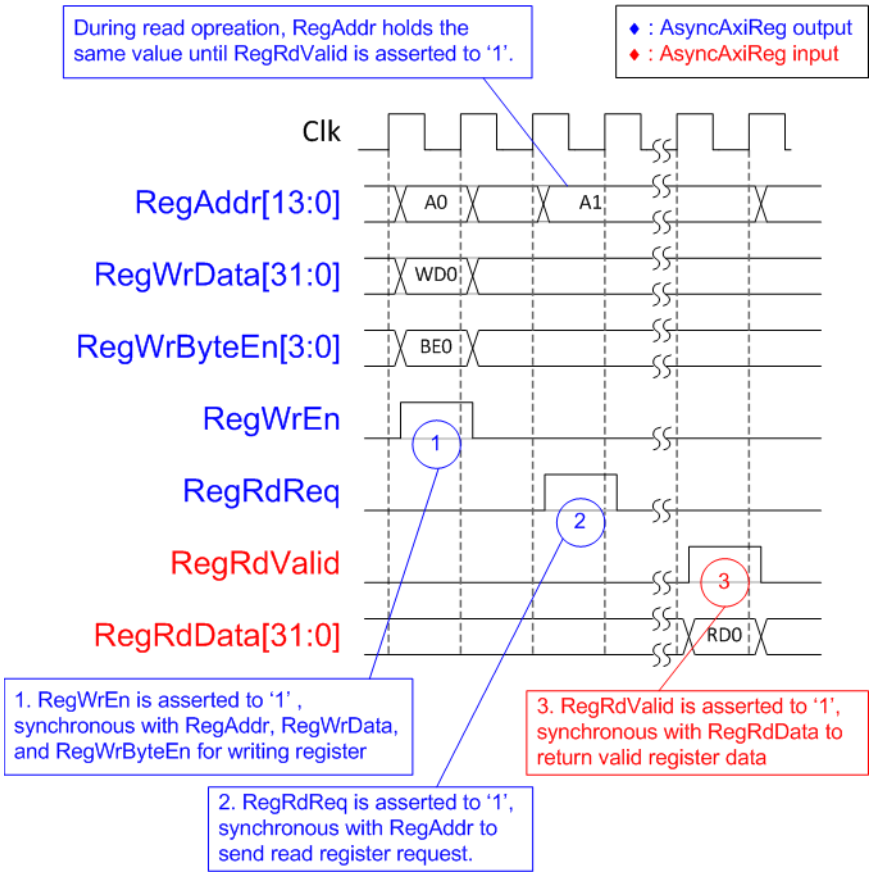

Figure 2-10 Register interface timing diagram

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 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, AsyncAxiReg forwards the read value to LAxir* interface.

2.3.2 UserReg

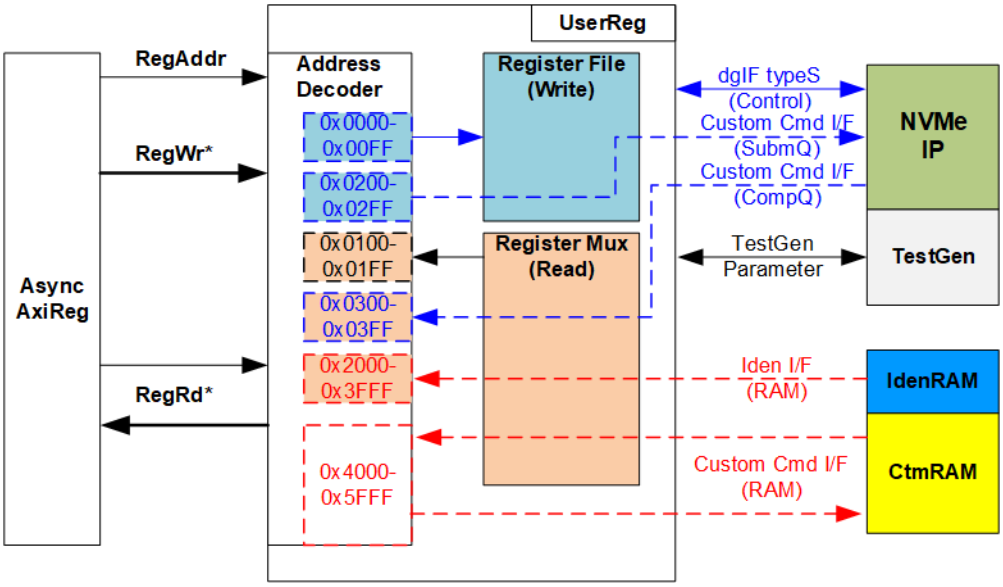

Figure 2-11 UserReg Interface

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 NVMe-IP and TestGen. This area is write-access only.
2) 0x0200 – 0x02FF: mapped to set the parameters for custom command interface of NVMe-IP. This area is write-access only.
3) 0x0100 – 0x01FF: mapped to read the status signals of NVMe-IP and TestGen. This area is read-access only.
4) 0x0300 – 0x03FF: mapped to read the status of custom command interface (NVMe-IP). This area is read-access only.
5) 0x2000 – 0x3FFF: mapped to read data from IdenRAM. This area is read-access only.
6) 0x4000 – 0x5FFF: mapped to write or read data with custom command RAM interface. This area supports both 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, 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 Rd/Wr	Register Name (Label in the “nvmeiptest.c”)	Description
0x0000 – 0x00FF: Control signals of NVMe-IP and TestGen (Write access only)		
BA+0x0000	User Address (Low) Reg (USRADRL_INTREG)	[31:0]: Input to be bit[31:0] of start address as 512-byte unit (UserAddr[31:0] of dgIF typeS)
BA+0x0004	User Address (High) Reg (USRADRH_INTREG)	[15:0]: Input to be bit[47:32] of start address as 512-byte unit (UserAddr[47:32] of dgIF typeS)
BA+0x0008	User Length (Low) Reg (USRLENL_INTREG)	[31:0]: Input to be bit[31:0] of transfer length as 512-byte unit (UserLen[31:0] of dgIF typeS)
BA+0x000C	User Length (High) Reg (USRLENH_INTREG)	[15:0]: Input to be bit[47:32] of transfer length as 512-byte unit (UserLen[47:32] of dgIF typeS)
BA+0x0010	User Command Reg (USRCMD_INTREG)	[2:0]: Input to be user command (UserCmd of dgIF typeS for NVMe-IP) “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 NVMe-IP to start the operation.
BA+0x0014	Test Pattern Reg (PATTSEL_INTREG)	[2:0]: Select test pattern “000”-Increment, “001”-Decrement, “010”-All 0, “011”-All 1, “100”-LFSR
BA+0x0020	NVMe Timeout Reg (NVMTIMEOUT_INTREG)	[31:0]: Mapped to TimeOutSet[31:0] of NVMe-IP
0x0100 – 0x01FF: Status signals of NVMe-IP and TestGen (Read access only)		
BA+0x0100	User Status Reg (USRSTS_INTREG)	[0]: UserBusy of dgIF typeS (‘0’: Idle, ‘1’: Busy) [1]: UserError of dgIF typeS (‘0’: Normal, ‘1’: Error) [2]: Data verification fail (‘0’: Normal, ‘1’: Error)
BA+0x0104	Total disk size (Low) Reg (LBASIZEL_INTREG)	[31:0]: Mapped to LBASize[31:0] of NVMe-IP
BA+0x0108	Total disk size (High) Reg (LBASIZEH_INTREG)	[15:0]: Mapped to LBASize[47:32] of NVMe-IP [31]: Mapped to LBAMode of NVMe-IP
BA+0x010C	User Error Type Reg (USRERRTYPE_INTREG)	[31:0]: Mapped to UserErrorType[31:0] of NVMe-IP to show error status
BA+0x0110	PCIe Status Reg (PCISTS_INTREG)	[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) [13:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in Integrated Block for PCIe datasheet
BA+0x0114	Completion Status Reg (COMPSTS_INTREG)	[15:0]: Mapped to AdmCompStatus[15:0] of NVMe-IP [31:16]: Mapped to IOCompStatus[15:0] of NVMe-IP
BA+0x0118	NVMe CAP Reg (NVMCAP_INTREG)	[31:0]: Mapped to NVMeCAPReg[31:0] of NVMe-IP
BA+0x011C	NVMe IP Test pin Reg (NVMTESTPIN_INTREG)	[31:0]: Mapped to TestPin[31:0] of NVMe-IP

Address Rd/Wr	Register Name (Label in the “nvmeiptest.c”)	Description
0x0100 – 0x01FF: Status signals of NVMe-IP and TestGen (Read access only)		
BA+0x0130 - BA+0x013C	Expected value Word0-3 Reg (EXPPATW0-W3_INTREG)	128-bit of the expected data at the 1st failure data in Read command 0x0130: Bit[31:0], 0x0134[31:0]: Bit[63:32], …, 0x013C[31:0]: Bit[127:96]
BA+0x0150 - BA+0x015C	Read value Word0-3 Reg (RDPATW0-W3_INTREG)	128-bit of the read data at the 1st failure data in Read command 0x0150: Bit[31:0], 0x0154[31:0]: Bit[63:32], …, 0x015C[31:0]: Bit[127:96]
BA+0x0170	Data Failure Address(Low) Reg (RDFAILNOL_INTREG)	[31:0]: Bit[31:0] of the byte address of the 1st failure data in Read command
BA+0x0174	Data Failure Address(High) Reg (RDFAILNOH_INTREG)	[24:0]: Bit[56:32] of the byte address of the 1st failure data in Read command
BA+0x0178	Current test byte (Low) Reg (CURTESTSIZEL_INTREG)	[31:0]: Bit[31:0] of the current test data size in TestGen module
BA+0x017C	Current test byte (High) Reg (CURTESTSIZEH_INTREG)	[24:0]: Bit[56:32] of the current test data size of TestGen module
Other interfaces (Custom command of NVMe-IP, IdenRAM, and Custom RAM)		
BA+0x0200- BA+0x023F Wr	Custom Submission Queue Reg (CTMSUBMQ_STRUCT)	[31:0]: Submission queue entry of SMART and Flush command. Input to be CtmSubmDW0-DW15 of NVMe-IP. 0x200: DW0, 0x204: DW1, …, 0x23C: DW15
BA+0x0300– BA+0x030F Rd	Custom Completion Queue Reg (CTMCOMPQ_STRUCT)	[31:0]: CtmCompDW0-DW3 output from NVMe-IP. 0x300: DW0, 0x304: DW1, …, 0x30C: DW3
BA+0x0800 Rd	IP Version Reg (IPVERSION_INTREG)	[31:0]: Mapped to IPVersion[31:0] of NVMe-IP
BA+0x2000- BA+0x2FFF Rd	Identify Controller Data (IDENCTRL_CHARREG)	4Kbyte Identify Controller Data Structure
BA+0x3000– BA+0x3FFF Rd	Identify Namespace Data (IDENNAME_CHARREG)	4Kbyte Identify Namespace Data Structure
BA+0x4000– BA+0x5FFF Wr/Rd	Custom command Ram (CTMRAM_CHARREG)	Connect to 8K byte CtmRAM interface. Used to store 512-byte data output from SMART Command.

3 CPU Firmware

3.1 Test firmware (nvmeiptest.c)

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

1) CPU initializes UART and Timer parameters.
2) CPU waits until PCIe connection links up (PCISTS_INTREG[0]='1').
3) CPU waits until NVMe-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 PCISTS_INTREG[7:2].
5) CPU displays the main menu. There are six menus for running six commands of NVMe-IP, i.e., Identify, Write, Read, SMART, Flush, and Shutdown.

More details of the sequence in 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="000". Next, Test logic generates command and asserts command request to NVMe-IP. After that, Busy flag (USRSTS_INTREG[0]) changes from '0' to '1'.
2) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

   Bit[0] is de-asserted to '0' after finishing operating the command. After that, the data from Identify command of NVMe-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 NVMe-IP output such as SSD capacity and LBA unit size (LBASIZE_INTREG).

3.1.2 Write/Read Command

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

1) Receive start address, transfer length, and test pattern from Serial 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_INTREG, USRLENL/H_INTREG, and PATTSEL_INTREG.
3) Set USRCMD_INTREG[2:0]=”010” for Write command or “011” for Read command. After that, NVMe-IP receives the new command request and busy flag (USRSTS_INTREG[0]) changes from ‘0’ to ‘1’.
4) CPU waits until the operation is completed or some errors (except verification error) are found by monitoring USRSTS_INTREG[2:0].

   Bit[0] is de-asserted to ‘0’ when command is completed.
   Bit[1] is asserted when 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 failed. Then, the verification error message is displayed. CPU is still running until the operation is done or user inputs any key to cancel operation.

   During running command, current transfer size read from CURTESTSIZE_INTREG is displayed every second.

5) After busy flag (USRSTS_INTREG[0]) is de-asserted to ‘0’, CPU displays the test result 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]=”100”. Next, Test logic generates command and asserts the request to NVMe-IP. After that, Busy flag (USRSTS_INTREG[0]) changes from ‘0’ to ‘1’.
3) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

   Bit[0] is de-asserted to ‘0’ when command is completed. After that, the data from SMART command of NVMe 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 some information decoded from CtmRAM (CTMRAM_CHARREG) 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_STRUCT) to be Flush command value.
2) Set USRCMD_INTREG[2:0]=”110”. Next, Test logic generates command and asserts the request to NVMe-IP. After that, Busy flag (USRSTS_INTREG[0]) changes from ‘0’ to ‘1’.
3) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

   Bit[0] is de-asserted to ‘0’ when command is completed. If the command is completed, 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_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]=”001”. Next, Test logic generates command and asserts the request to NVMe-IP. After that, Busy flag (USRSTS_INTREG[0]) changes from ‘0’ to ‘1’.
2) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

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

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

3.2 Function list in Test firmware

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

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

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

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

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

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

4 Example Test Result

The example test result when running demo system by using 512 GB Samsung 970 Pro is shown in Figure 4-1.

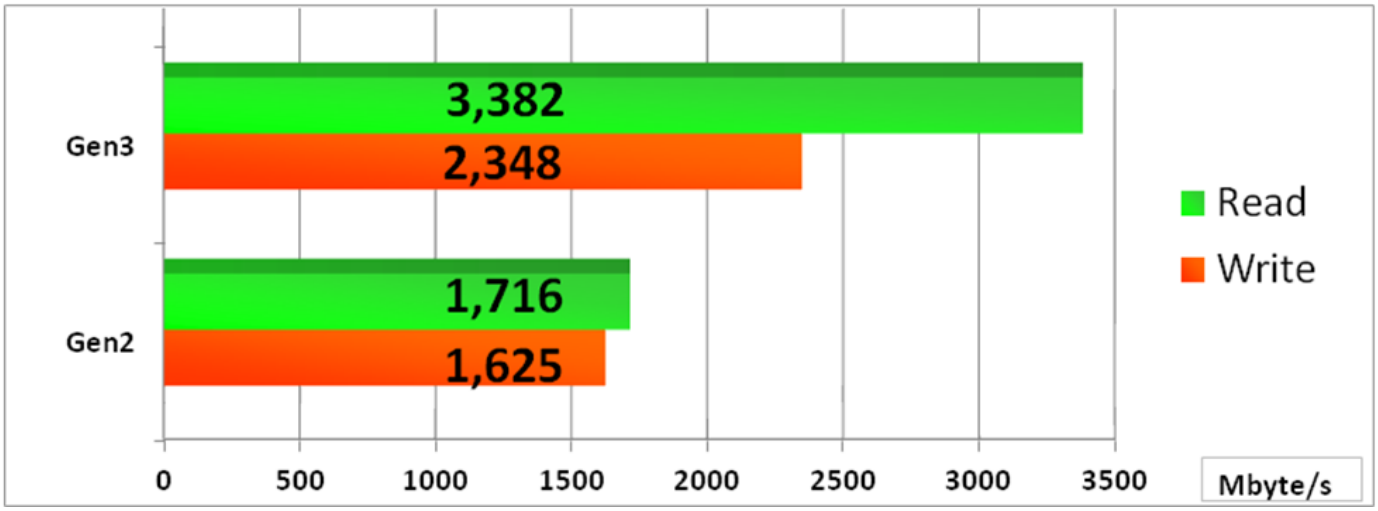

Figure 4-1 Test Performance of NVMe-IP demo by using Samsung 970 Pro SSD

By using PCIe Gen3 on KCU105 board, write performance is about 2300 Mbyte/sec and read performance is about 3300 Mbyte/sec. Performance by using PCIe Gen2 on ZC706 board is slower than Gen3. Write and read performance on Gen2 are about 1600-1700 Mbyte/sec.

5 Revision History

Revision	Date	Description
1.0	1-Jun-16	Initial Release
1.1	5-Sep-16	Add CURTESTSIZE register
1.2	6-Dec-16	Change buffer from DDR to BRAM
2.0	12-Jun-17	New NVMe-IP version
2.1	27-Jul-17	Add LFSR pattern
3.0	19-Jul-18	Support Shutdown, SMART and Flush command
3.1	23-Nov-18	Add dword enable for RAM interface
3.2	3-May-19	Update SSD performance and system information
3.3	28-Jan-20	Remove header of all zero and all one patterns
3.4	21-Feb-20	Add Function list in Test firmware
3.5	27-Aug-20	Correct the information
3.6	5-Mar-21	Update Function name
3.7	7-Jun-21	Update Register map
3.8	17-Mar-22	Update firmware and register type