muNVMe-IP reference design manual

Rev1.0 17-Jun-22

1 Overview

NVMe-IP has one user interface for sending one command in each time. The next command can be requested after the first command is done. Therefore, user cannot send Write and Read command to access two SSD areas at the same time. To be the solution, muNVMe-IP (multiple-user NVMe-IP) is designed to have two user interfaces for sending up to two commands at the same time.

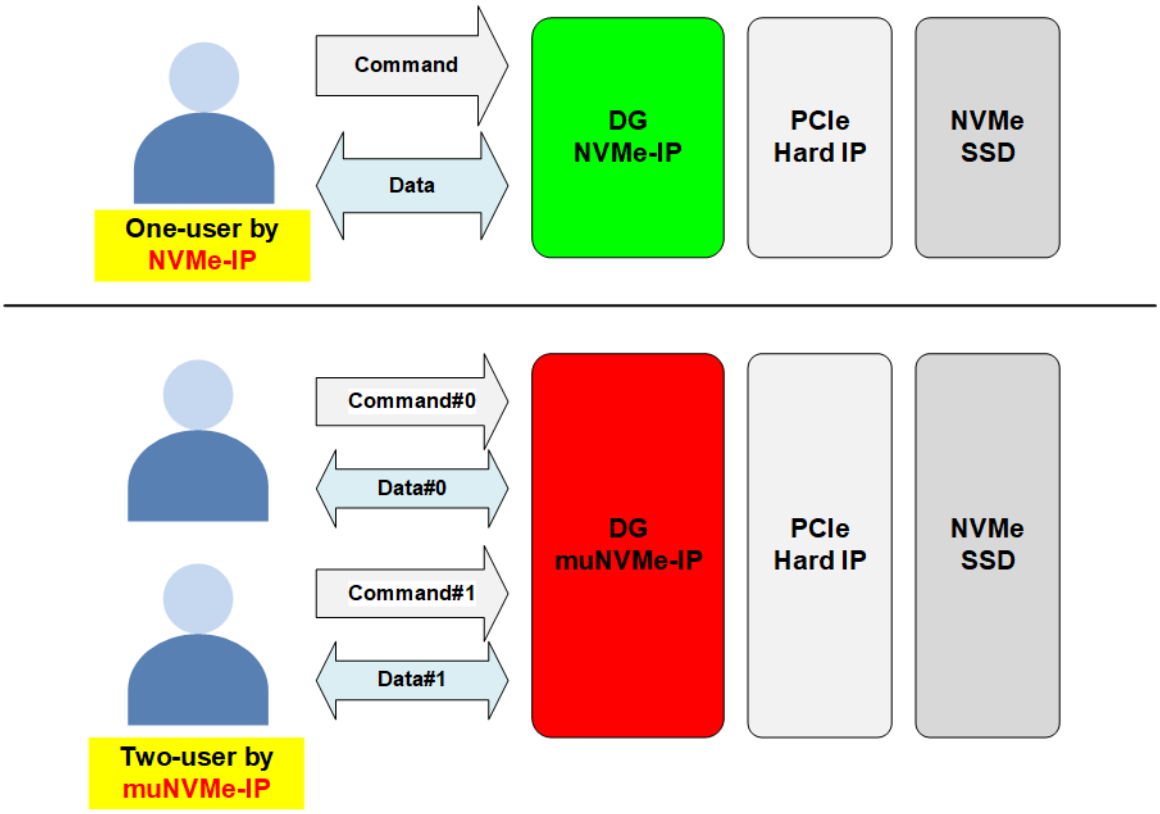

Figure 1-1 NVMe-IP and muNVMe-IP comparison

Using two-user interface of muNVMe-IP, user can send two Write commands, two Read command, or one Write command and one Read command to access the same SSD. When running Write+Write or Read+Read command, the performance of each user is about the performance of one user divided by two. For example, when write performance of one user is equal to 2200 Mbyte/s, the write performance per user of two-user Write test is equal to 1100 Mbyte/s. While running the mixed command (Write with Read command), the performance depends on each SSD characteristic.

Ref: NVM-IP reference design document
https://dgway.com/products/IP/NVMe-IP/dg_nvmeip_refdesign_en.pdf

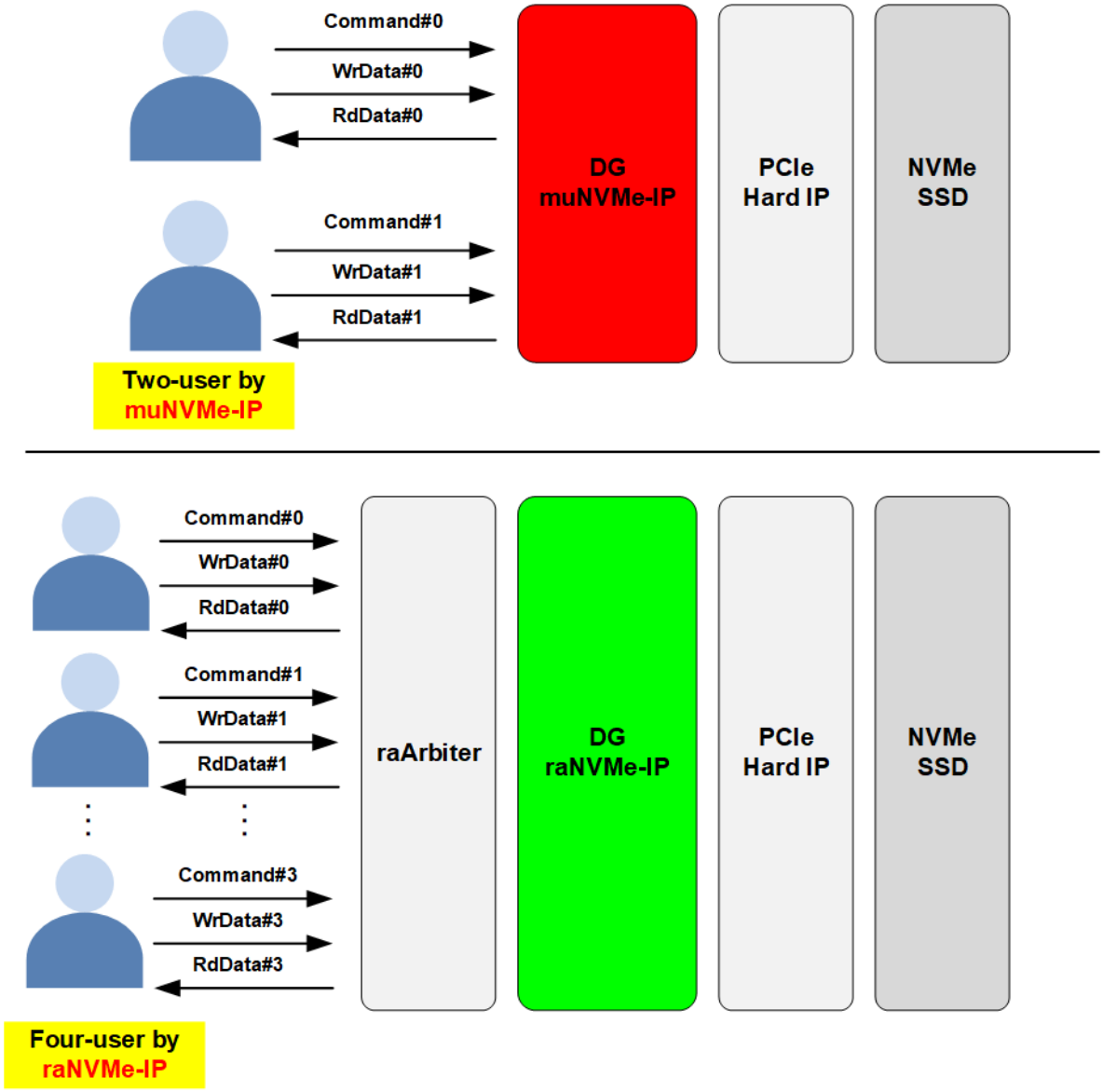

Figure 1-2 muNVMe-IP and raNVMe-IP comparison

Design Gateway has another solution for supporting multiple-user system – raNVMe-IP with raArbiter. In multiple-user reference design by raNVMe-IP, four user interfaces are implemented. According to raNVMe-IP specification, up to 32 Write commands or Read commands can be requested at the same time by 4 KB data per command. However, all 32 commands must be the same command (Write or Read command). Mixed Write-Read command is not supported without customization. While muNVMe-IP supports mixed Write-Read command with 128 KB per command. Therefore, when running with some SSDs that write/read performance of 128 KB size is better than 4 KB size, muNVMe-IP achieves better performance than raNVMe-IP with multiple-user design.

Ref: raNVMe-IP with multiple-user reference design document
https://dgway.com/products/IP/NVMe-IP/dg_ranvmemult_refdesign_xilinx.pdf

2 Hardware overview

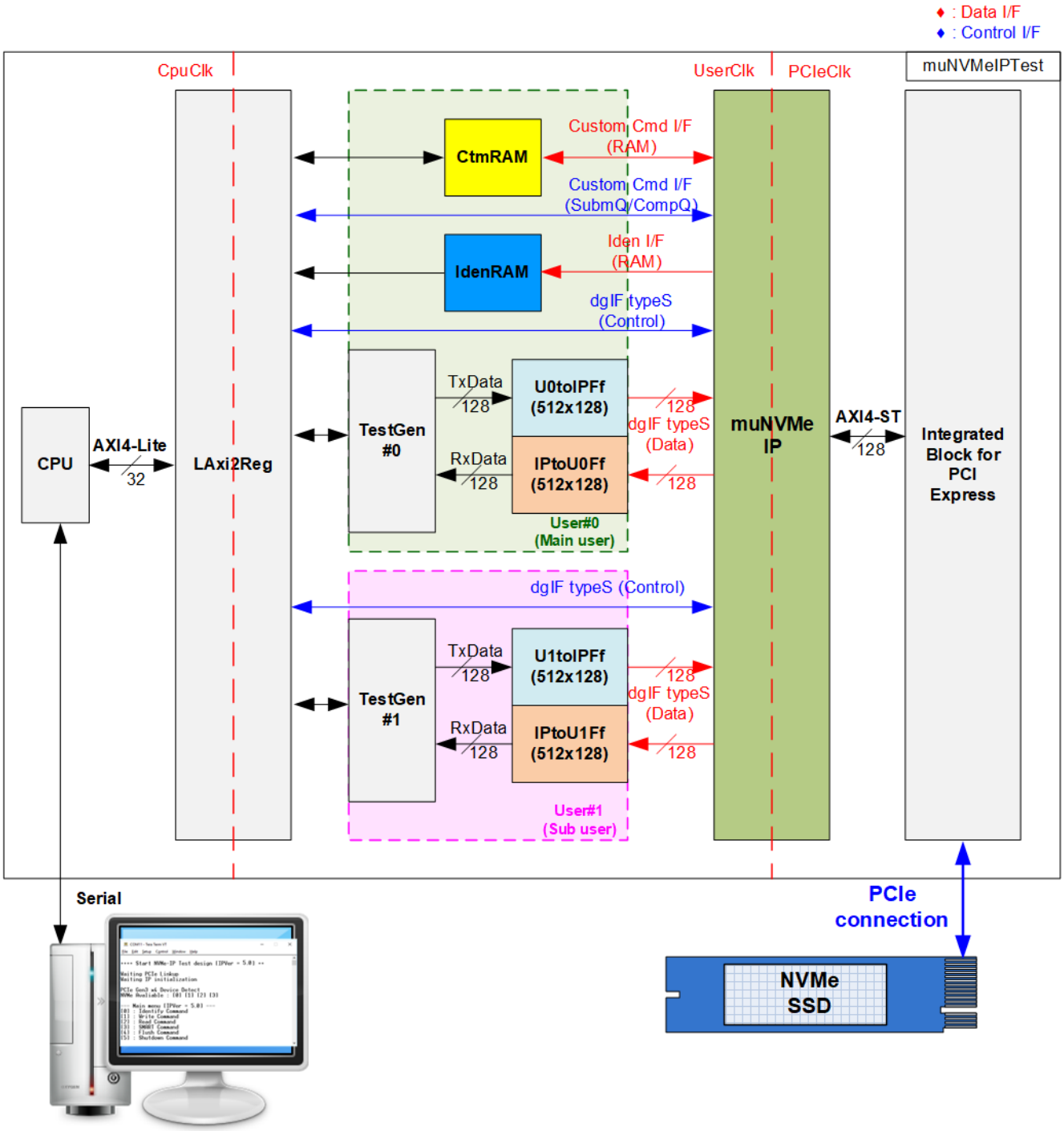

Figure 2-1 muNVMe-IP demo hardware

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, UxtoIPFf, IPtoUxFf, muNVMe-IP, and PCIe block), and CPU system (CPU and LAxi2Reg).

TestGen is the test logic to generate test data stream for muNVMe-IP via UxtoIPFf and read data stream output from muNVMe-IP via IPtoUxFf for verification. NVMe includes the muNVMe-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 is implemented to control the flow for operating each command.

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

There are three clock domains displayed in Figure 2-1, i.e., CpuClk, UserClk, and PCIeClk. CpuClk is the clock domain of CPU and its peripherals. This clock must be stable clock and can be different clock domain from other hardwares. UserClk is the user clock domain for running the user interface of muNVMe-IP, RAM, FIFO, and TestGen. According to muNVMe-IP datasheet, clock frequency of UserClk must be more than or equal to PCIeClk. This reference design uses 275/280 MHz for PCIe Gen3. PCIeClk is the clock output from PCIe hard IP to synchronous with data stream of 128-bit AXI4 stream bus that is equal to 250 MHz when configured to 4-lane PCIe Gen3.

More details of the hardware are described as follows.

2.1 TestGen

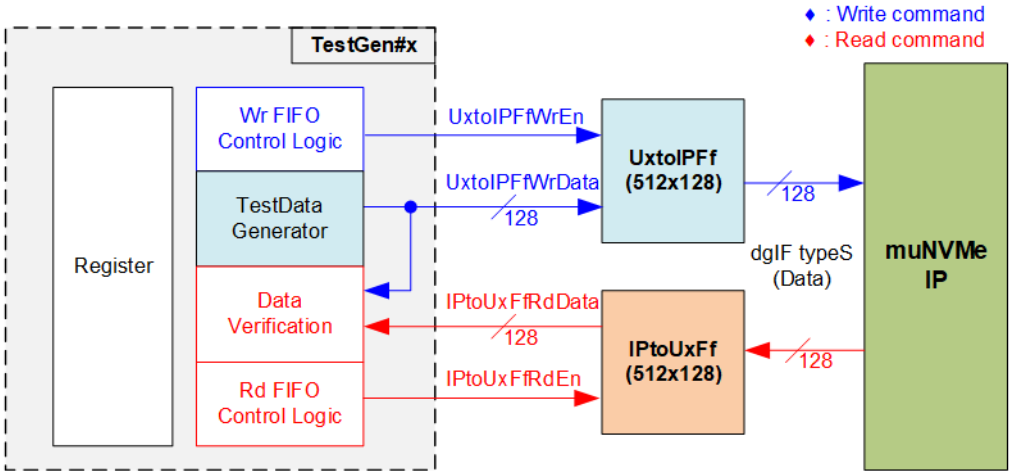

Figure 2-2 TestGen interface

TestGen module handles the data interface of muNVMe-IP for transferring the data in Write and Read command. In Write command, TestGen sends 128-bit test data to muNVMe-IP via UxtoIPFf. In Read command, the test data is fed from IPtoUxFf to compare with the expected value for data verification. Data bandwidth of TestGen is matched to muNVMe-IP by running at the same clock and the same data bus size. Control logic always asserts Write enable or Read enable to '1' for writing or reading the FIFO when FIFO is ready. Thus, UxtoIPFf and IPtoUxFf are available for transferring data with muNVMe-IP and then the test logic shows the best performance to write and read data with the SSD through muNVMe-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 details of hardware logic of TestGen are shown in Figure 2-3.

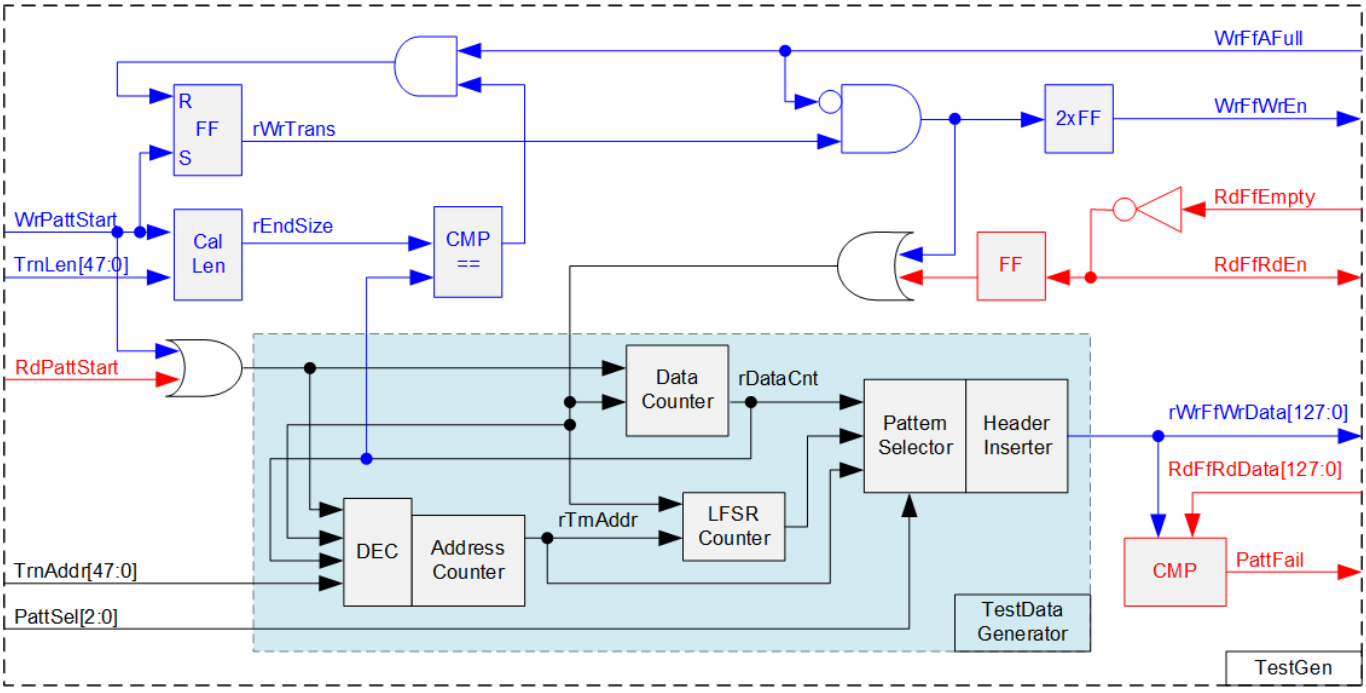

Figure 2-3 TestGen hardware

As shown in the right side of Figure 2-3, flow control signals of FIFO are WrFfAFull and RdFfEmpty. When FIFO is almost full in write operation (WrFfAFull='1'), WrFfWrEn is de-asserted to '0' to pause sending data 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'.

Two counters – Data counter and Address counter are designed in TestGen. Data counter (rDataCnt) counts the amount of transferred data in Write command and Read command. When total amount of transferred data is equal to the end size, set by user, write enable or read enable of FIFO is de-asserted to '0'. Also, the Data counter (rDataCnt) are fed to be the write data for Write command or the expected data for Read command. TestGen supports to generate five patterns of test data, 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 consist of two data parts to create unique test data in every 512-byte data, as shown in Figure 2-4.

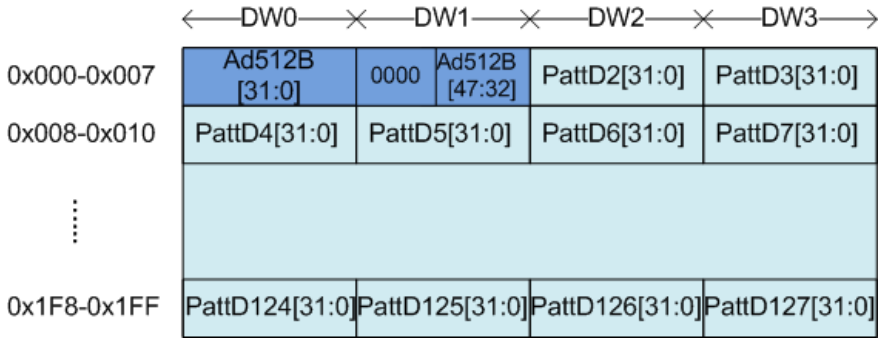

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

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 the address counter (rTrnAddr) which shows the address in 512-byte unit. The initial value of rTrnAddr is set by user and it 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 implemented by using rDataCnt. The decremental data can be designed by connecting NOT logic to rDataCnt. 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.

When data verification detects incorrect received data in Read command, Fail flag (PattFail) is asserted to '1'. The timing diagram to write data to FIFO is shown as follows.

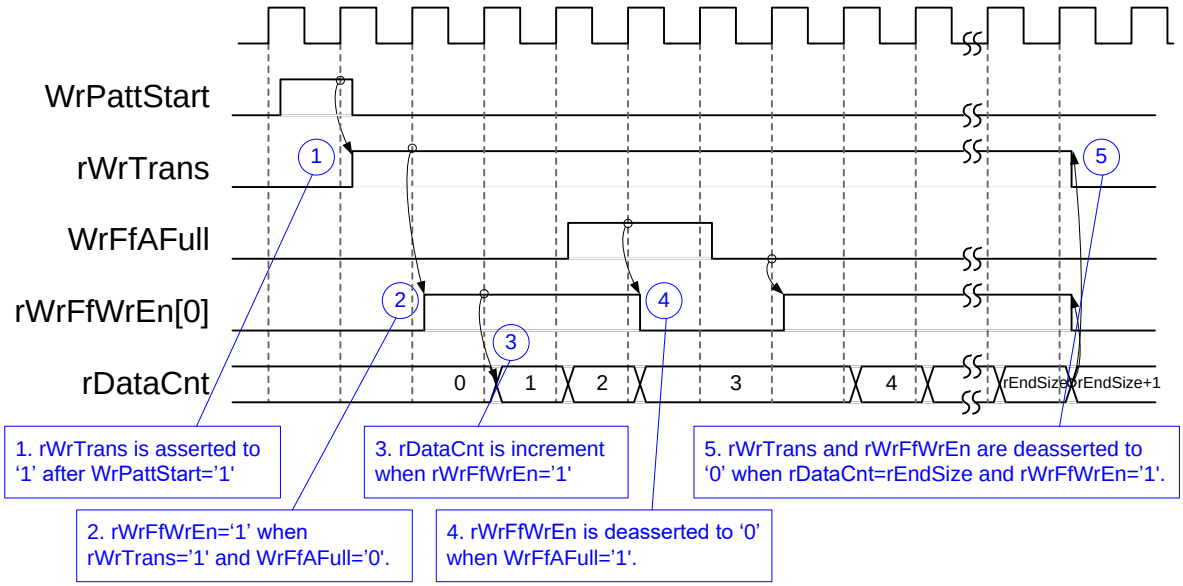

Figure 2-5 Timing diagram of Write operation in TestGen

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' to show the Write command is operating. Second, the FIFO must not be full by monitoring WrFfAFull='0'.
3) The write enable is fed to count total amount of data by rDataCnt in the Write command.
4) If FIFO is almost full (WrFfAFull='1'), the write process is paused by de-asserting rWrFfWrEn to '0'.
5) When rDataCnt is equal to the set value (rEndSize), rWrTrans is de-asserted to '0'. At the same time, rWrFfWrEn is also de-asserted to '0' to finish generating data.

For read operation, read enable of FIFO is controlled by empty flag of FIFO. Comparing to write enable, the read enable is not stopped by total amount of data and not started by start flag. The read enable is asserted to '1' when FIFO is not empty. The data counter and the address counter are increased when the read enable is asserted to '1' to count total amount of data and generate the header of expect value.

2.2 NVMe

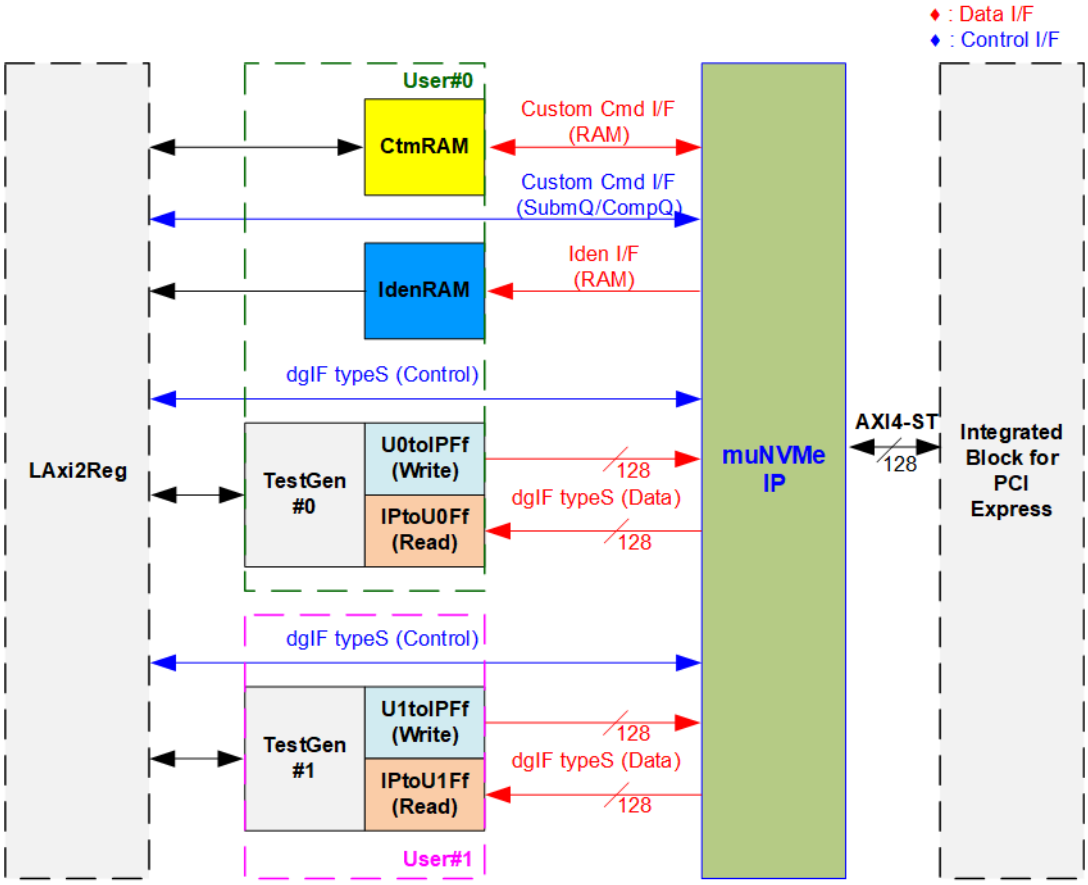

Figure 2-6 NVMe hardware

Figure 2-6 shows how to integrate muNVMe-IP in the reference design. Each user interface of muNVMe-IP consists of control interface and data interface. The control interface of User#0 receives the command and the parameters via custom command interface or dgIF typeS, depending on the command. Custom command interface is used when operating SMART command or Flush command. While User#1 control interface has only dgIF typeS to support Write command and Read command.

The data interface of User#0 has four interfaces, i.e., custom command RAM interface, Identify interface, FIFO input interface (dgIF typeS), and FIFO output interface (dgIF typeS). While User#1 data interface has FIFO interface that is dgIF typeS. Data bus width of all data interface is 128 bits. 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 for forwarding SMART data to LAxi2Reg.

2.2.1 muNVMe-IP

The muNVMe-IP implements NVMe protocol of the host side to access one NVMe SSD directly without PCIe switch connection. The muNVMe-IP supports two users. The first user (Main user) supports six commands, i.e., Write, Read, Identify, Shutdown, SMART, and Flush. The second user (Sub user) supports two commands - Write and Read. muNVMe-IP can connect to the PCIe hard IP directly. More details of muNVMe-IP are described in datasheet. https://dgway.com/products/IP/NVMe-IP/dg_munvme_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.
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. muNVMe-IP and LAxi2Reg have the different data bus size, 128 bits on muNVMe-IP but 32 bits on LAxi2Reg. Therefore, IdenRAM is asymmetric RAM that has the different bus size on Write interface and Read interface. Also, muNVMe-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.

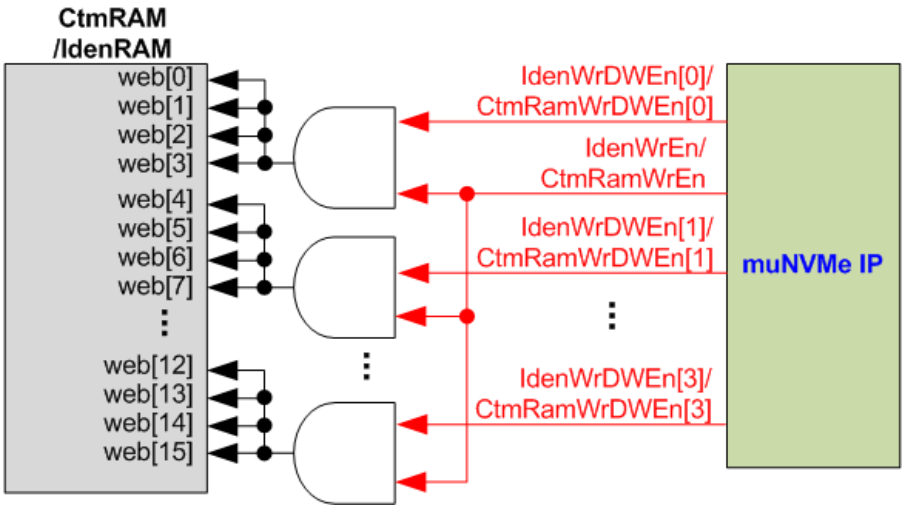

Figure 2-7 Byte write enable conversion logic

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 muNVMe-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.

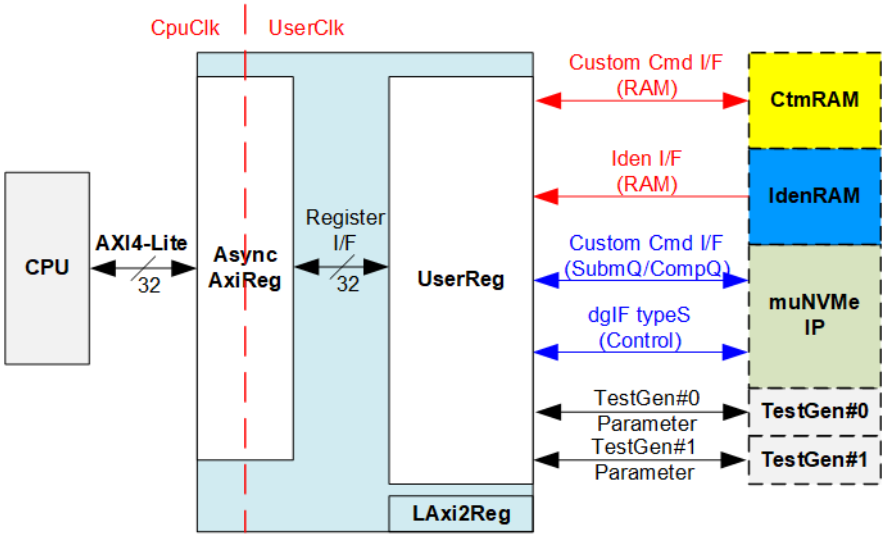

Figure 2-8 CPU and peripherals hardware

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

2.3.1 AsyncAxiReg

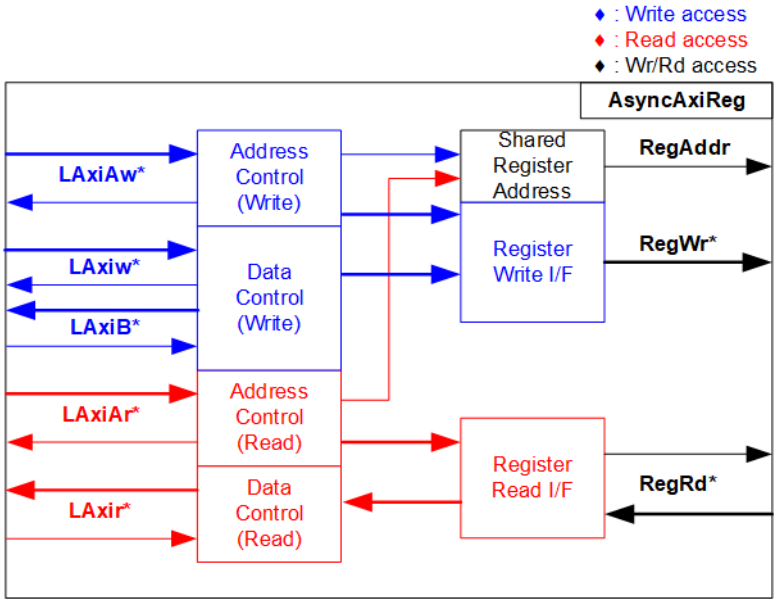

Figure 2-9 AsyncAxiReg Interface

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. 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 returned data from Register Read I/F is transferred to AXI4-Lite bus by using clock domain crossing registers. In register interface, RegAddr is shared signal for write and read access. Therefore, it loads the address 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.

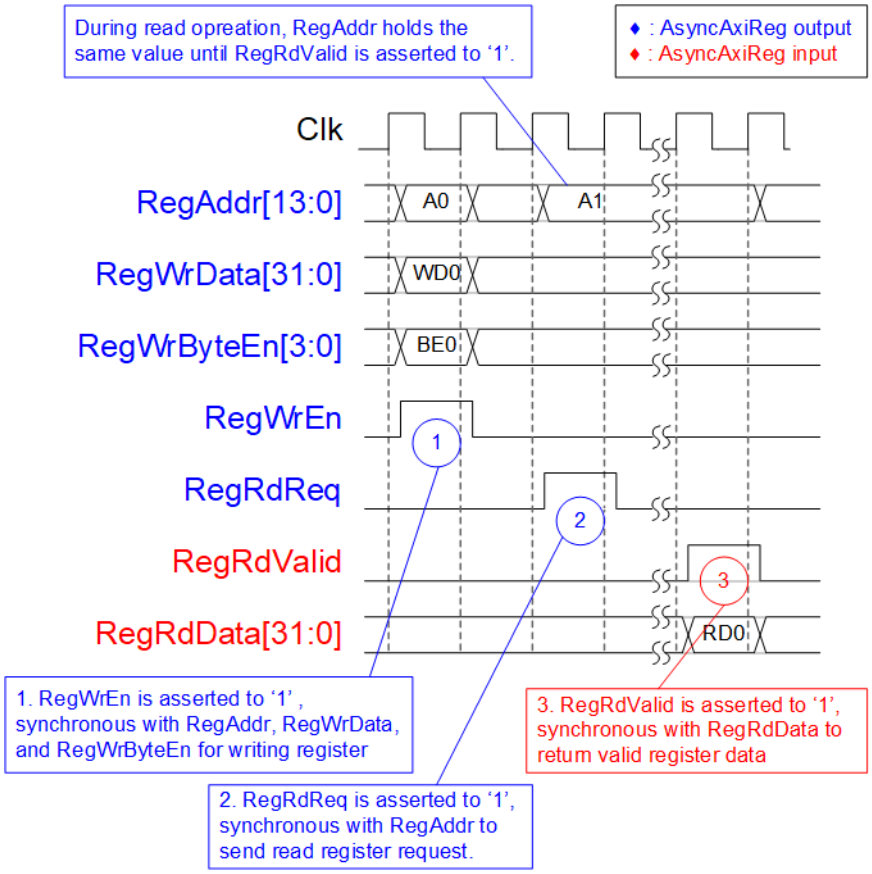

Figure 2-10 Register interface timing diagram

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

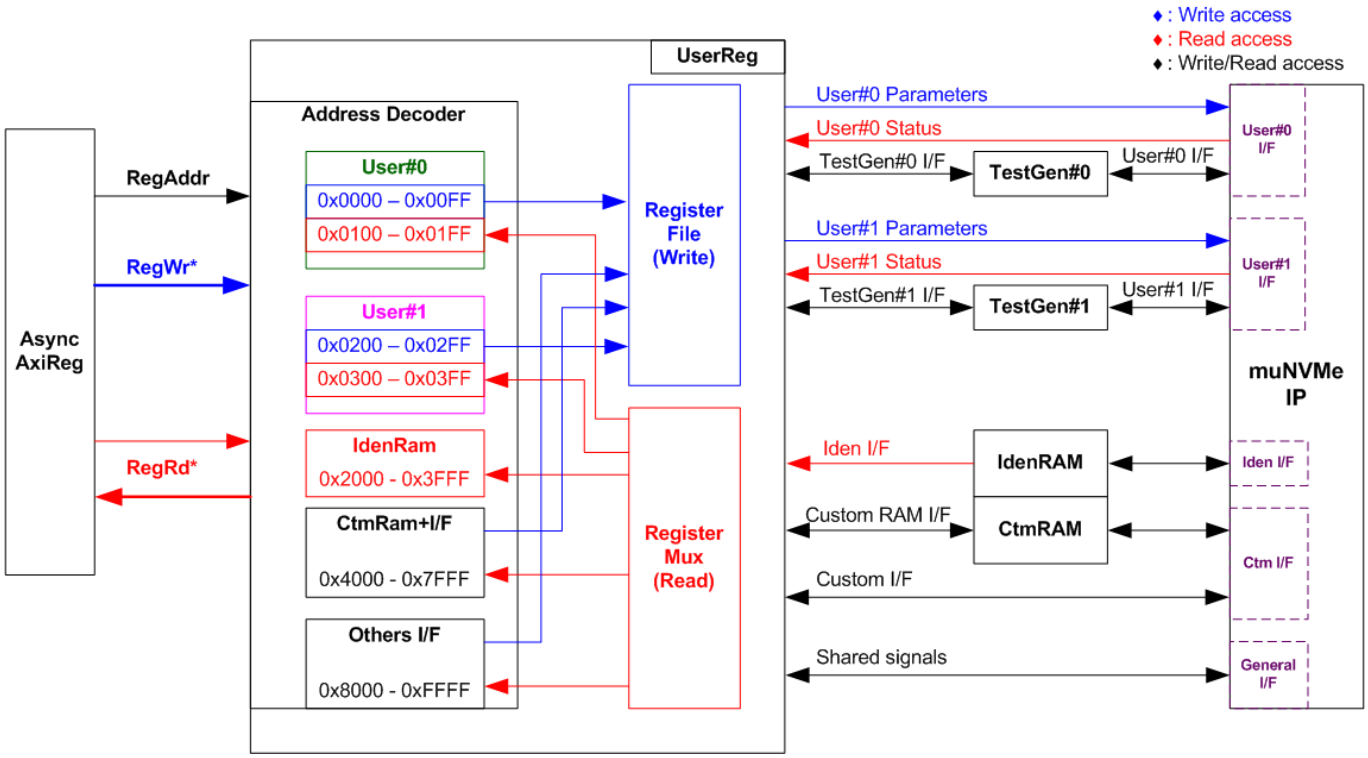

Figure 2-11 UserReg Interface

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 – 0x01FF: mapped to User#0 and TestGen#0
2) 0x0200 – 0x03FF: mapped to User#1 and TestGen#1
3) 0x2000 – 0x3FFF: mapped to read data from IdenRAM. This area is read-access only.
4) 0x4000 – 0x5FFF: mapped to write or read data with custom command RAM interface. This area supports write-access and read-access but the demo shows only read access by running SMART command.
5) 0x6000 – 0x7FFF: mapped to Custom command interface
6) 0x8000 – 0xFFFF: mapped to other interfaces such as shared parameters for all Users, PCIe status, and IP version.

Address decoder decodes the upper bit of RegAddr for selecting the active hardware that is muNVMe-IP, TestGen#0, TestGen#1, 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 uses 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 defintion

Address Rd/Wr	Register Name (Label in the “munvmeiptest.c”)	Description
0x0000 – 0x01FF: Signal Interface of User#0 (muNVMe-IP) and TestGen#0		
0x0000 – 0x00FF: Control signals of User#0 and TestGen#0 (Write access only)		
BA+0x0000	User#0 Address (Low) Reg (U0ADRL_INTREG)	[31:0]: Input to be bit[31:0] of User#0 start address as 512-byte unit (U0Addr[31:0] of muNVMe-IP)
BA+0x0004	User#0 Address (High) Reg (U0ADRH_INTREG)	[15:0]: Input to be bit[47:32] of User#0 start address as 512-byte unit (U0Addr[47:32] of muNVMe-IP)
BA+0x0008	User#0 Length (Low) Reg (U0LENL_INTREG)	[31:0]: Input to be bit[31:0] of User#0 transfer length as 512-byte unit (U0Len[31:0] of muNVMe-IP)
BA+0x000C	User#0 Length (High) Reg (U0LENH_INTREG)	[15:0]: Input to be bit[47:32] of User#0 transfer length as 512-byte unit (U0Len[47:32] of muNVMe-IP)
BA+0x0010	User#0 Command Reg (U0CMD_INTREG)	[2:0]: Input to be User#0 command (U0Cmd of muNVMe-IP) 000b: Identify, 001b: Shutdown, 010b: Write SSD, 011b: Read SSD, 100b: SMART, 110b: Flush, 101b/111b: Reserved When this register is written, the command request is sent to muNVMe-IP via User#0 I/F to start the operation.
BA+0x0014	User#0 Test Pattern Reg (U0PATTSEL_INTREG)	[2:0]: Select test pattern of TestGen#0 000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1, 100b-LFSR
0x0100 – 0x01FF: Status signals of User#0 and TestGen#0 (Read access only)		
BA+0x0100	User#0 Status Reg (U0STS_INTREG)	[0]: User#0 Busy of muNVMe-IP (‘0’: Idle, ‘1’: Busy) [1]: U0Error of muNVMe-IP (‘0’: Normal, ‘1’: Error) [2]: Data verification fail in TestGen#0 (‘0’: Normal, ‘1’: Error)
BA+0x0104	User#0 Error Type Reg (U0ERRTYPE_INTREG)	[31:0]: Mapped to U0ErrorType[31:0] of muNVMe-IP to show error status
BA+0x0108	User#0 Completion Status Reg (U0COMPSTS_INTREG)	[15:0]: Mapped to U0AdmCompStatus[15:0] of muNVMe-IP [31:16]: Mapped to U0IOCompStatus[15:0] of muNVMe-IP
BA+0x0110	User#0 Test pin (Low) Reg (U0TESTPINL_INTREG)	[31:0]: Mapped to U0TestPin[31:0] of muNVMe-IP
BA+0x0114	User#0 Test pin (High) Reg (U0TESTPINH_INTREG)	[15:0]: Mapped to U0TestPin[47:32] of muNVMe-IP
BA+0x0140- BA+0x014C	User#0 Expected value Word0-3 Reg (U0EXPPATW0-W3_INTREG)	128-bit of the expected data at the 1st failure data in TestGen#0 when operating Read command 0x0130: Bit[31:0], 0x0134: Bit[63:32], …, 0x013C: Bit[127:96]
BA+0x0160- BA+0x016C	User#0 Read value Word0-3 Reg (U0RDPATW0-W3_INTREG)	128-bit of the read data at the 1st failure data in TestGen#0 when operating Read command 0x0150: Bit[31:0], 0x0154: Bit[63:32], …, 0x015C: Bit[127:96]
BA+0x0180	User#0 Data Failure Address(Low) Reg (U0RDFAILNOL_INTREG)	[31:0]: Bit[31:0] of the byte address of the 1st failure data in TestGen#0 when operating Read command
BA+0x0184	User#0 Data Failure Address(High) Reg (U0RDFAILNOH_INTREG)	[24:0]: Bit[56:32] of the byte address of the 1st failure data in TestGen#0 when operating Read command
BA+0x0188	User#0 Current test byte (Low) Reg (U0CURTESTSIZEL_INTREG)	[31:0]: Bit[31:0] of the current test data size in TestGen#0 module
BA+0x018C	User#0 Current test byte (High) Reg (U0CURTESTSIZEH_INTREG)	[24:0]: Bit[56:32] of the current test data size of TestGen#0 module

Address Rd/Wr	Register Name (Label in the “munvmeiptest.c”)	Description
0x0200 – 0x03FF: Signal Interface of User#1 (muNVMe-IP) and TestGen#1		
0x0200 – 0x02FF: Control signals of User#1 and TestGen#1 (Write access only)		
BA+0x0200- BA+0x0217	Control signals of User#1/TestGen#1 (U1ADRL_INTREG – U1PATTSEL_INTREG)	Similar to 0x0000 – 0x0017 which are the registers of User#0 and TestGen#0, these are mapped to User#1 and TestGen#1 instead. However, U1CMD_INTREG can be assigned by two values for two commands only – 010b: Write SSD and 011b: Read SSD.
0x0300 – 0x03FF: Status signals of User#1 and TestGen#1 (Read access only)		
BA+0x0300- BA+0x037F	Status signals of User#1/TestGen#1 (U1STS_INTREG – U1CURTESTSIZEH_INTREG)	Similar to 0x0100 – 0x018F which are the registers of User#0 and TestGen#0, these are mapped to User#1 and TestGen#1 instead. However, U1COMPSTS_INTREG[31:16] is applied to map to U1IOCompStatus[15:0] of muNVMe-IP while bit[15:0] is reserved and U1TestPin[15:0] is map to U1TESTPINL_INTREG while U1TESTPINL_INTREG[31:16] and U1TESTPINH_INTREG is reserved.
0x2000 – 0x3FFF: IdenRAM (Read access only)		
BA+0x2000- BA+0x2FFF	Identify Controller Data (IDENCTRL_CHARREG)	4Kbyte Identify Controller Data Structure
BA+0x3000- BA+0x3FFF	Identify Namespace Data (IDENNAME_CHARREG)	4Kbyte Identify Namespace Data Structure
0x4000 – 0x5FFF: CtmRAM (Write/Read access)		
BA+0x4000- BA+0x5FFF	Custom command Ram (CTMRAM_CHARREG)	Connect to 8K byte CtmRAM interface for storing 512-byte data that is output from SMART Command.
0x6000 – 0x7FFF: Custom Command Interface		
BA+0x6000- BA+0x603F Wr	Custom Submission Queue Reg (CTMSUBMQ_STRUCT)	[31:0]: Submission queue entry of SMART and Flush command. Input to be CtmSubmDW0-DW15 of muNVMe-IP. 0x200: DW0, 0x204: DW1, …, 0x23C: DW15
BA+0x6100- BA+0x610F Rd	Custom Completion Queue Reg (CTMCOMPQ_STRUCT)	[31:0]: CtmCompDW0-DW3 output from muNVMe-IP. 0x300: DW0, 0x304: DW1, …, 0x30C: DW3
0x8000 – 0xFFFF: Other Interfaces		
BA+0x8000 Wr	NVMe Timeout Reg (NVMTIMEOUT_INTREG)	[31:0]: Mapped to TimeOutSet[31:0] of muNVMe-IP
BA+0x8100 Rd	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+0x8110 Rd	NVMe CAP Reg (NVMCAP_INTREG)	[31:0]: Mapped to NVMeCAPReg[31:0] of muNVMe-IP
BA+0x8120 Rd	Total disk size (Low) Reg (LBASIZEL_INTREG)	[31:0]: Mapped to LBASize[31:0] of muNVMe-IP
BA+0x8124 Rd	Total disk size (High) Reg (LBASIZEH_INTREG)	[15:0]: Mapped to LBASize[47:32] of muNVMe-IP [31]: Mapped to LBAMode of muNVMe-IP
BA+0x8200 Rd	IP Version Reg (IPVERSION_INTREG)	[31:0]: Mapped to IPVersion[31:0] of muNVMe-IP

3 CPU Firmware

3.1 Test firmware (munvmeiptest.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 muNVMe-IP completes initialization process (U0/1STS_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 five menus for running six commands of muNVMe-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

This command can be requested by User#0 I/F. The sequence of the firmware when user selects Identify command is below.

1) Set U0CMD_INTREG=000b to send Identify command request on User#0 I/F of muNVMe-IP. After that, busy flag of User#0 I/F (U0STS_INTREG[0]) changes from '0' to '1'.
2) CPU waits until the operation is completed or some errors are found by monitoring U0STS_INTREG[1:0].

   Bit[0] is de-asserted to '0' after finishing operating the command. Next, the data from Identify command of muNVMe-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 U0ERRTYPE_INTREG[31:0]. Finally, the process is stopped.

3) After busy flag (U0STS_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 muNVMe-IP such as SSD capacity (LBASIZEL/H_INTREG).

3.1.2 Write/Read Command

Write and Read command can be requested by User#0 I/F and User#1 I/F. The sequence of the firmware when user selects Write/Read command is below.

1) Receive 2-user parameters such as command (Write, Read, or Disable), 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 UxADRL/H_INTREG, UxLENL/H_INTREG, and UxPATTSEL_INTREG.
3) Set UxCMD_INTREG[2:0] = 010b for Write command or 011b for Read command and then the command request is asserted to the user who runs the Write command or Read command. After that, busy flag of the active user (UxSTS_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 UxSTS_INTREG[2:0].

   Bit[0] is de-asserted to '0' when command of User#x is completed.
   Bit[1] is asserted when error is detected in User#x. 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 in User#x. After that, the verification error message is displayed. However, CPU is still running until the operation is done or user inputs any key to cancel operation.

   While the command is running, current transfer size of the active user read from UxCURTESTSIZEL/H_INTREG is displayed every second.

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

3.1.3 SMART Command

This command can be requested by User#0 I/F. 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 U0CMD_INTREG[2:0]=100b to send SMART command request on User#0 I/F of muNVMe-IP. After that, busy flag of User#0 I/F (U0STS_INTREG[0]) changes from '0' to '1'.
3) CPU waits until the operation is completed or some errors are found by monitoring U0STS_INTREG[1:0].

   Bit[0] is de-asserted to '0' after finishing operating the command. Next, the data from SMART command of muNVMe-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 U0ERRTYPE_INTREG[31:0]. Finally, the process is stopped.

4) After busy flag (U0STS_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

This command can be requested by User#0 I/F. 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 U0CMD_INTREG[2:0]=110b to send Flush command request on User#0 I/F of muNVMe-IP. After that, busy flag of User#0 I/F (U0STS_INTREG[0]) changes from '0' to '1'.
3) CPU waits until the operation is completed or some errors are found by monitoring U0STS_INTREG[1:0].

   Bit[0] is de-asserted to '0' after finishing operating the command. Next, 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 U0ERRTYPE_INTREG[31:0]. Finally, the process is stopped.

3.1.5 Shutdown Command

This command can be requested by User#0 I/F. The sequence of the firmware when user selects Shutdown command is below.

1) Set U0CMD_INTREG[2:0]=001b to send Shutdown command request on User#0 I/F of muNVMe-IP. After that, busy flag of User#0 I/F (U0STS_INTREG[0]) changes from '0' to '1'.
2) CPU waits until the operation is completed or some errors are found by monitoring U0STS_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 U0ERRTYPE_INTREG[31:0]. Finally, the process is stopped.

3) After busy flag (U0STS_INTREG[0]) is de-asserted to '0', the SSD and muNVMe-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 muNVMe-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(unsigned int user)	
Parameters	user: 0-User#0, 1-User#1
Return value	Read value of U0/1CURTESTSIZEH/L_INTREG
Description	Read U0/1CURTESTSIZEH/L_INTREG and return read value as function result.

int get_param(userin_struct* userin)	
Parameters	userin: Four inputs from user, i.e., command, 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 print_space(unsigned long long size_input)	
Parameters	size_input: test size for displaying on the console
Return value	None
Description	Calculate the number of digits and the number of spaces to display size_input with alignment on the console.

int setctm_flush(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the muNVMe-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 muNVMe -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(unsigned int user)	
Parameters	user: 0-User#0, 1-User#1
Return value	None
Description	Read U0/1ERRTYPE_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(unsigned int user, unsigned int cmd, unsigned int timeuseh, unsigned int timeusel)	
Parameters	user, cmd, timeuseh, timeusel
Return value	None
Description	Print user channel, command, and total size by calling get_cursize and show_size function. After that, calculate total time usage from timeuseh and timeusel and then display in usec, msec, or sec unit. Finally, transfer performance is calculated and displayed in MB/s unit.

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

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

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

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

void show_vererr(unsigned int user)	
Parameters	user: 0-User#0, 1-User#1
Return value	None
Description	Read U0/1RDFAILNOL/H_INTREG (error byte address), U0/1EXPPATW0-W3_INTREG (expected value), and U0/1RDPATW0-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(void)	
Parameters	None
Return value	0: No error, -1: Receive invalid input
Description	Run Write command or Read command, following in topic 3.1.2 (Write/Read Command)

4 Example Test Result

The test results when running demo system by 1-2 users are shown in Figure 4-1. The test environment uses 280 GB Intel 900P and ZCU106 board (PCIe Gen3).

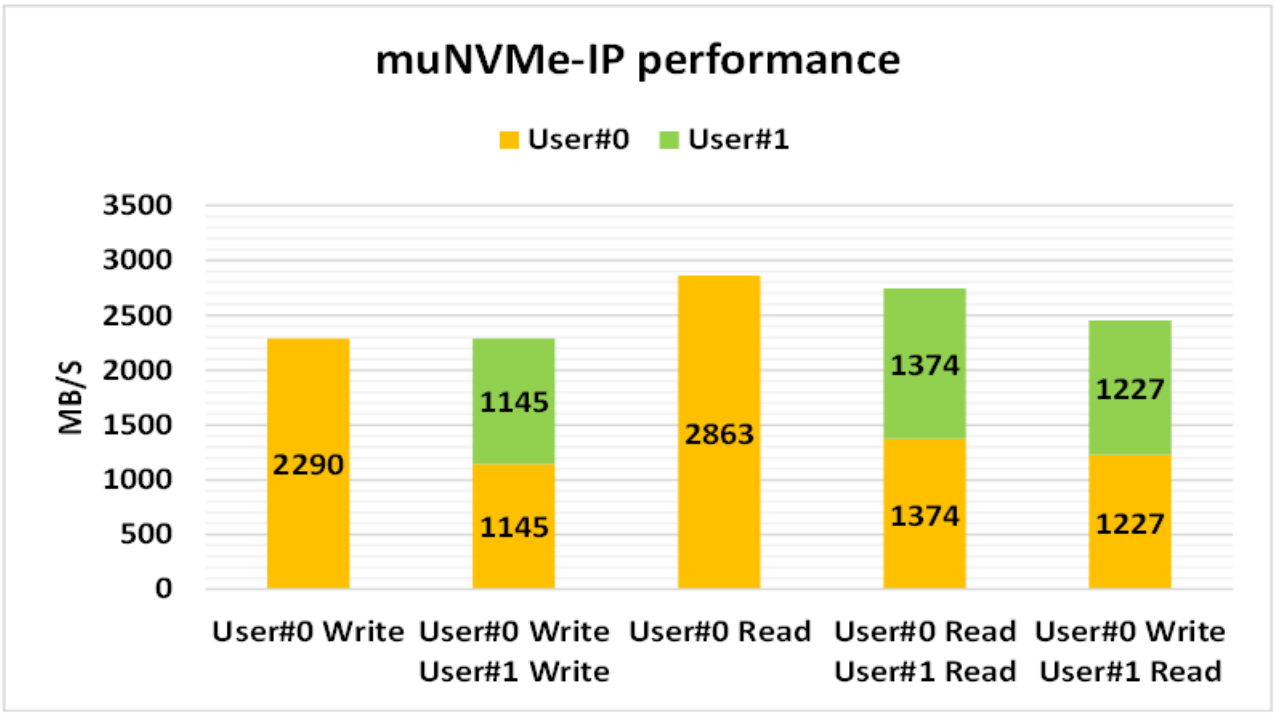

Figure 4-1 Test Performance of muNVMe-IP demo

Some SSDs can balance the device loading when Write and Read command are requested at the same time. As shown in Figure 4-1, the result shows the same performance when two users run the same command or different command. Also, total performance of two users is almost equal to the performance of one user.

While some SSDs has different characteristic. The load balancing for running Write command and Read command is different. The test result of these SSDs is different when running two users by mixed Write/Read command, comparing to using the same commands.

5 Revision History

Revision	Date	Description
1.0	17-Jun-22	Initial Release