NVMe-IP reference design manual

Rev3.11 12-Dec-23

1 Overview

NVM Express (NVMe) is a specification that defines the interface between the host controller and solid state drive (SSD) through PCI Express. It optimizes the process of issuing commands and completions by utilizing only two registers (Command issue and Command completion), and enables parallel operation by supporting up to 64K commands within a single queue. This improves transfer performance for both sequential and random access.

In the PCIe SSD market, two standards are commonly used: AHCI and NVMe. AHCI is the older standard used for providing the interface to SATA hard disk drives while NVMe is optimized for non-volatile memory like SSDs. A detailed comparison between the AHCI and NVMe protocol is available in the "A Comparison of NVMe and AHCI" document at https://sata-io.org/system/files/member-downloads/NVMe%20and%20AHCI_%20_long_.pdf

An example of an NVMe storage device can be found at http://www.nvmexpress.org/products/.

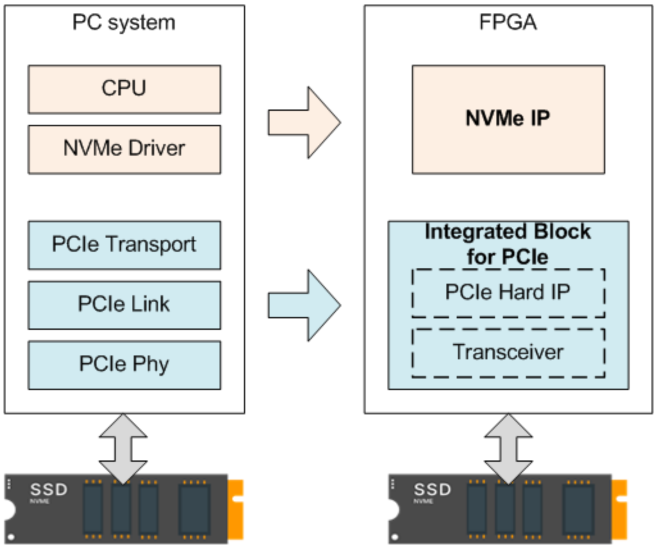
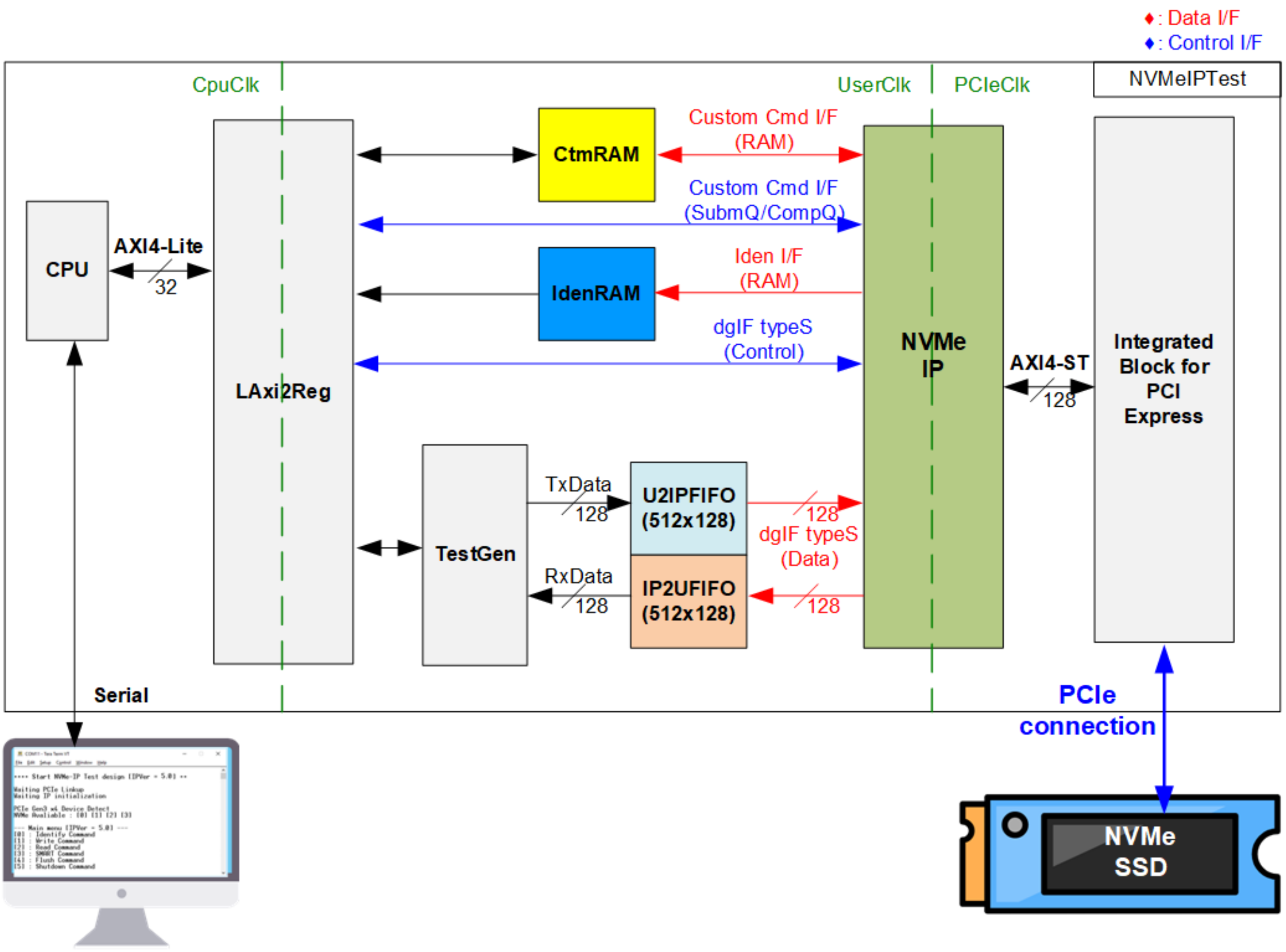


Figure 1-1 NVMe protocol layer

To access NVMe SSD, the general system implements an NVMe driver running on the processor, as shown on the left side of Figure 1-1. The physical connection of the NVMe standard is PCIe connector which is one-to-one type, allowing for each PCIe host to connect to one PCIe device without the use of a PCIe switch. NVMe-IP implements the NVMe driver for accessing NVMe SSD using pure hardware logic. This allows the user to access NVMe SSD without requiring any processor or driver, but instead using the NVMe IP in the FPGA board. The use of pure hardware logic for the NVMe host controller reduces the overhead time for software-hardware handshake, resulting in high performance for both writing and reading with NVMe SSD.

2 Hardware overview



```
++++ Start NVMe-IP Test design [IPVer = 5.0] ++

Waiting PCIe Linkup
Waiting IP initialization

PCIe Gen3 x4 Device Detect
NVMe Available : [0] [1] [2] [3]

--- Main menu [IPVer = 5.0] ---
[0] : Identify Command
[1] : Write Command
[2] : Read Command
[3] : SMART Command
[4] : Flush Command
[5] : Shutdown Command
```

Figure 2-1 NVMe-IP demo hardware

The hardware modules in the test system are divided into three parts: test function (TestGen), NVMe function (CtmRAM, IdenRAM, U2IPFIFO, IP2UFIFO, NVMe-IP and PCIe block), and CPU system (CPU and LAxi2Reg).

The TestGen connects to the user interface of NVMe-IP and is responsible for generating test data stream of Write command and verifying test data stream of Read command. The write and read data stream are stored at two FIFOs (U2IPFIFO and IP2UFIFO). The TestGen always writes or reads data when the FIFO is ready to check the best transfer performance of the system.

NVMe consists of the NVMe-IP and the PCIe hard IP (Integrated Block for PCI Express) for accessing an NVMe SSD directly without PCIe switch. The command request and the parameters of each command (the inputs of NVMe-IP) are controlled by the CPU through LAxi2Reg module. While the data interface for both Custom and Identify commands is connected to RAMs that are accessible by the CPU.

CPU is connected to LAxi2Reg module for interface with the NVMe test logics. Integrating CPU to the test system allows the user to set the test parameters and monitor the test status via Serial console. Using CPU also facilitates the execution of many test cases to verify the functionality of the IP. The default firmware for the CPU includes the functions for executing the NVMe commands by using NVMe-IP.

There are three clock domains shown in Figure 2-1, i.e., CpuClk, UserClk, and PCIeClk. The CpuClk is the clock domain for the CPU and its peripherals, and it must be a stable clock that can be independent from other hardware. The UserClk is the user clock domain utilized for the operation of the NVMe-IP, RAM, and TestGen. As specified in the NVMe-IP datasheet, the clock frequency of UserClk must be equal to or greater than that of the PCIeClk. The reference design uses 275/280 MHz for UserClk at PCIe Gen3 design and 200 MHz for UserClk at PCIe Gen2 design. Finally, the PCIeClk is the clock output generated by the PCIe hard IP which is synchronized with the 128-bit AXI4 stream. The PCIeClk frequency is equal to 250 MHz for 4-lane PCIe Gen3 and 125 MHz for 4-lane PCIe Gen2.

More details of the hardware are described as follows.

2.1 TestGen

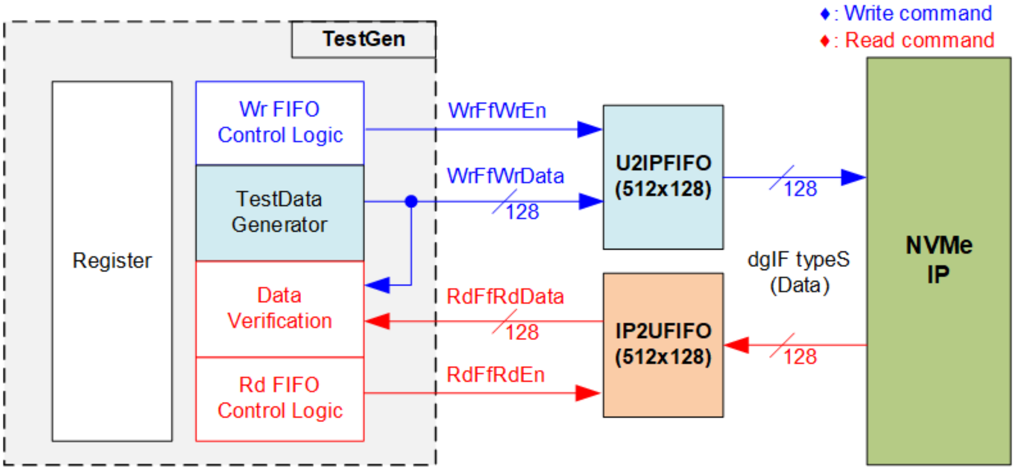


Figure 2-2 TestGen interface

The TestGen module handles the data interface of NVMe-IP, facilitating data transfer for both Write and Read commands. In case of a Write command, TestGen sends 128-bit test data to NVMe-IP via U2IPFIFO. In contrast, for a Read command, the test data is received from IP2UFIFO for comparison with the expected value, ensuring data accuracy. Data bandwidth of TestGen is set to match that of NVMe-IP by running at the same clock and data bus size. The control logic ensures that the Write or Read enable is always asserted to 1b when the FIFO is ready to write or read data, respectively. This allows NVMe-IP to transfer data through U2IPFIFO and IP2UFIFO without delay, providing the best performance for writing and reading data with the SSD through NVMe-IP.

The Register file in the TestGen receives user-defined test parameters, including total transfer size, transfer direction, verification enable, and test pattern selector. Additionally, the internal logic includes a counter to check total transfer size of test data. The detailed hardware logic of TestGen is illustrated in Figure 2-3.

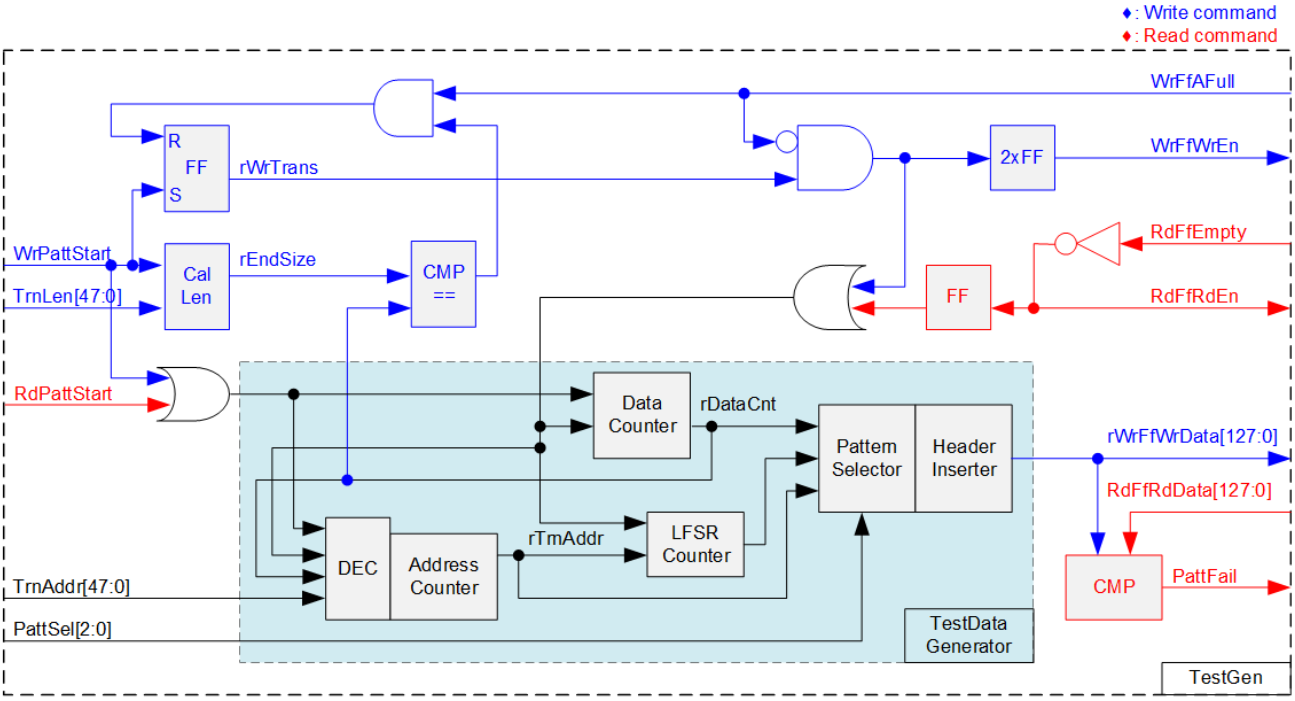


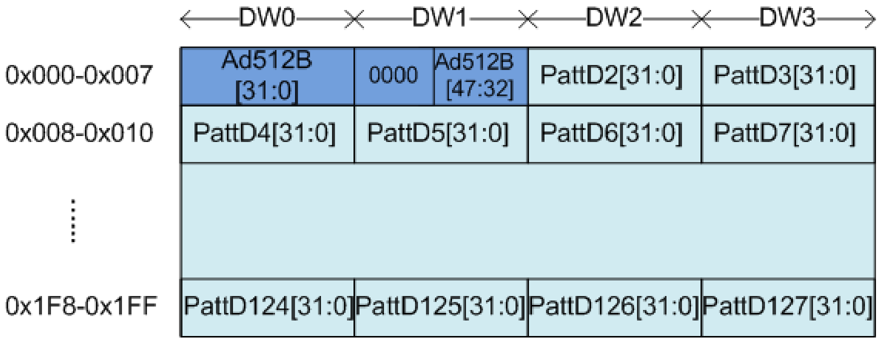
Figure 2-3 TestGen hardware

In Figure 2-3, two key aspects of the system are depicted. The first part illustrates the control of data flow, while the second part details the generation of test data for use with the FIFO interface.

In the upper portion of Figure 2-3, we focus on the control of data flow. Two signals, the WrFfAFull and RdFfEmpty, are integral to the FIFO interface for flow control. When the FIFO reaches its capacity (indicated by WrFfAFull=1b), the WrFfWrEn signal is set to 0b, effectively pausing data transfer into the FIFO. In a read operation, when data is available within the FIFO (indicated by RdFfEmpty=0b), the system retrieves this data for comparison by setting the RdFfRdEn to 1b. Furthermore, it is important to note that both write and read operation are completed when the total transferred data matches the user-defined value. Consequently, the counter logic is designed to track the amount of data transferred during this command, and upon command completion, both WrFfWrEn and RdFfRdEn must be de-asserted.

The lower section of Figure 2-3 outlines the methods for generating test data, either for writing to the FIFO or for data verification. There are five available test patterns: all-zero, all-one, 32-bit incremental data, 32-bit decremental data, and LFSR. These patterns are selected by the Pattern Selector.

For the all-zero or all-one pattern, every bit of the data is set to zero or one, respectively. Conversely, the other test patterns are designed by separating the data into two parts to create unique test data within every 512-byte data, as shown in Figure 2-4.



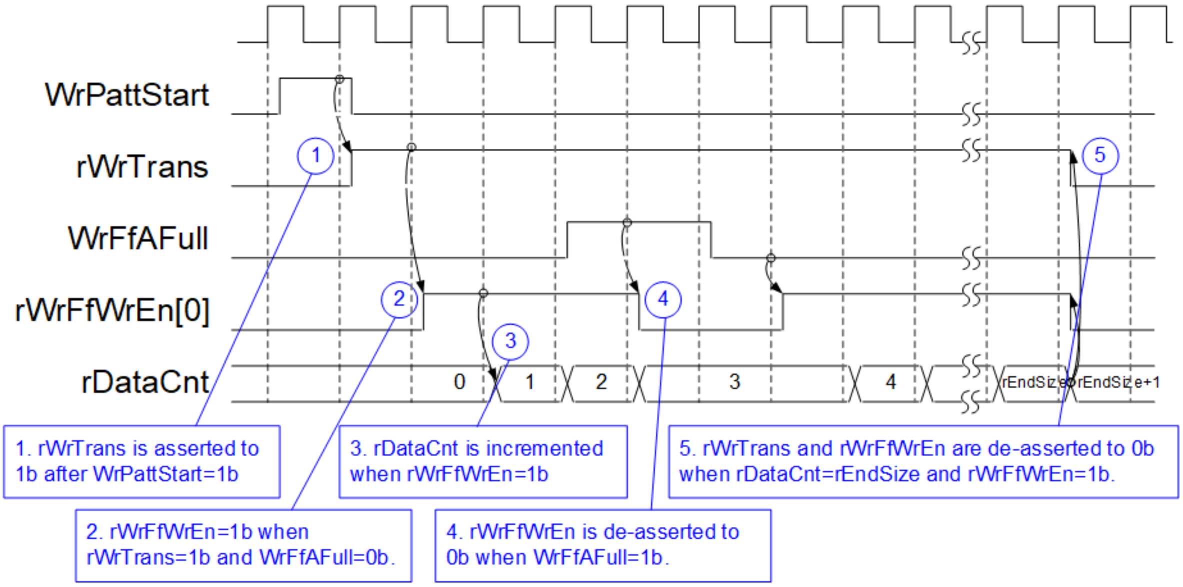
	DW0	DW1	DW2	DW3
0x000-0x007	Ad512B [31:0]	0000 Ad512B [47:32]	PattD2[31:0]	PattD3[31:0]
0x008-0x010	PattD4[31:0]	PattD5[31:0]	PattD6[31:0]	PattD7[31:0]
⋮				
0x1F8-0x1FF	PattD124[31:0]	PattD125[31:0]	PattD126[31:0]	PattD127[31:0]

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

Each 512-byte data block consists of a 64-bit header in Dword#0 and Dword#1, followed by the test data in the remaining words of the 512-byte data (Dword#2 – Dword#127). The header is created using the Address counter block, which operates in 512-byte units. The initial value of the Address counter is configured by the user and increases after transferring each 512-byte data.

The content of the remaining Dwords (DW#2 – DW#127) depend on the pattern selector, which could be 32-bit incremental data, 32-bit decremental data, or the LFSR pattern. The 32-bit incremental data is designed using the Data counter, while the decremental data can be created by connecting NOT logic to the incremental data. The LFSR pattern is generated using the LFSR counter, using the equation x^31 + x^21 + x + 1. To generate 128-bit test data, four 32-bit LFSR data are produced within a single clock cycle using look-ahead logic.

This Test data is used either as write data for the FIFO or for comparison with the data read from the FIFO. When data verification fails, the Fail flag is asserted to 1b. Below is an example of timing diagram illustrating the process of writing data to the FIFO.



1. rWrTrans is asserted to 1b after WrPattStart=1b

3. rDataCnt is incremented when rWrFfWrEn=1b

5. rWrTrans and rWrFfWrEn are de-asserted to 0b when rDataCnt=rEndSize and rWrFfWrEn=1b.

2. rWrFfWrEn=1b when rWrTrans=1b and WrFfAFull=0b.

4. rWrFfWrEn is de-asserted to 0b when WrFfAFull=1b.

Figure 2-5 Timing diagram of Write operation in TestGen

1) The write operation is initiated by setting WrPattStart signal to 1b for a single clock cycle, which is followed by the assertion of rWrTrans to enable the control logic for generating write enable to FIFO.
2) If two conditions are satisfied (rWrTrans is asserted to 1b during the write operation and the FIFO is not full, indicated by WrFfAFull=0b), the write enable (rWrFfWrEn) to FIFO is asserted to 1b.
3) The write enable is fed back to the counter to count the total amount of data during the write operation.
4) If FIFO is almost full (WrFfAFull=1b), the write process is paused by de-asserting rWrFfWrEn to 0b.
5) The write operation is finished when the total data count is equal to the set value. At this point, both rWrTrans and rWrFfWrEn are de-asserted to 0b.

For read transfer, the read enable of FIFO is controlled by the empty flag of FIFO. Unlike the 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 1b, the data counter and the address counter are increased for counting the total amount of data and generating the header of expected value, respectively.

2.2 NVMe

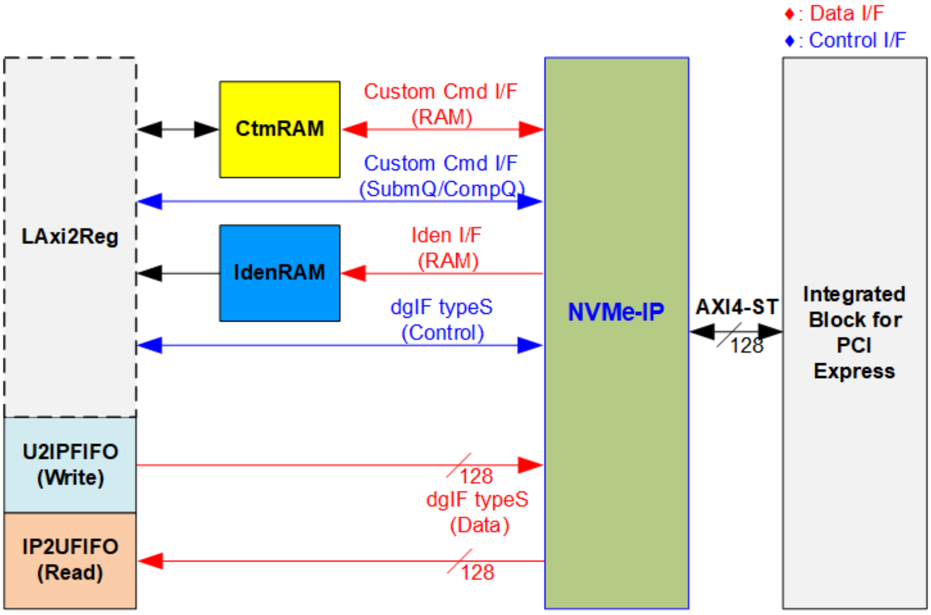


Figure 2-6 NVMe hardware

In the reference design, the NVMe-IP's user interface consists of a control interface and a data interface. The control interface receives commands and parameters from either the Custom command interface or dgIF typeS, depending on the type of command. For instance, Custom command interface is used when operating SMART, Secure Erase, or Flush command.

On the other hand, the data interface of NVMe-IP has four different interfaces with a data bus width of 128-bit. These interfaces include Custom command RAM interface, Identify interface, FIFO input interface (dgIF typeS), and FIFO output interface (dgIF typeS). While the Custom command RAM interface is a bi-directional interface, the other interfaces are unidirectional interface. In the reference design, the Custom command RAM interface is used for one-directional data transfer when NVMe-IP sends SMART data to LAxi2Reg.

2.2.1 NVMe-IP

The NVMe-IP implements NVMe protocol of the host side to direct access an NVMe SSD without PCIe switch connection. It supports seven commands, i.e., Write, Read, Identify, Shutdown, SMART, Secure Erase, and Flush. The NVMe-IP can be directly connected 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/

2.2.2 Integrated Block for PCIe

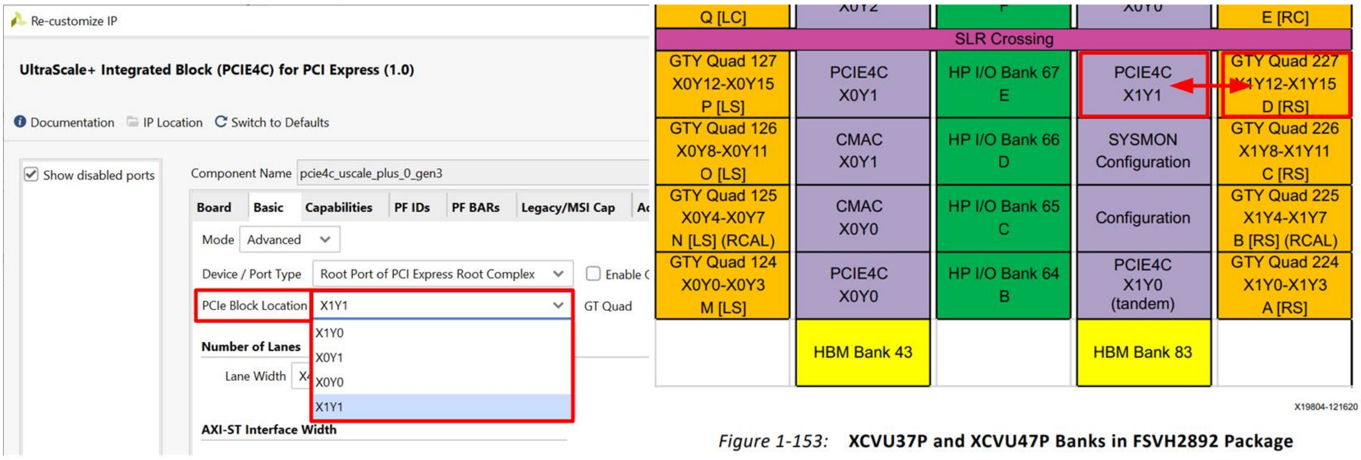
This block is the hard IP integrated in Xilinx FPGAs which implements Physical, Data Link, and Transaction Layers of PCIe specification. More details are described in Xilinx document.

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

The PCIe hard IP is created by using IP wizard. It is recommended for user to select "PCIe Block Location" which is closed to the transceiver pins that connect to the SSD. Please see more details about the location of PCIe Hard IP and transceiver from following document.

UG575: UltraScale and UltraScale+ FPGAs Packaging and Pinouts
https://www.xilinx.com/support/documentation/user_guides/ug575-ultrascale-pkg-pinout.pdf

The example of PCIe hard IP location on XCVC39P-FSVH2892 is show in Figure 2-7



Re-customize IP

UltraScale+ Integrated Block (PCIE4C) for PCI Express (1.0)

Component Name: pcie4c_uscale_plus_0_gen3

Mode: Advanced

Device / Port Type: Root Port of PCI Express Root Complex

□ Enable C

PCIe Block Location: X1Y1

Q [LC]	X0Y2	F	X0Y0	E [RC]
SLR Crossing				
GTY Quad 127 X0Y12-X0Y15 P [LS]	PCIE4C X0Y1	HP I/O Bank 67 E	PCIE4C X1Y1	GTY Quad 227 X1Y12-X1Y15 D [RS]
GTY Quad 126 X0Y8-X0Y11 O [LS]	CMAC X0Y1	HP I/O Bank 66 D	SYSMON Configuration	GTY Quad 226 X1Y8-X1Y11 C [RS]
GTY Quad 125 X0Y4-X0Y7 N [LS] (RCAL)	CMAC X0Y0	HP I/O Bank 65 C	Configuration	GTY Quad 225 X1Y4-X1Y7 B [RS] (RCAL)
GTY Quad 124 X0Y0-X0Y3 M [LS]	PCIE4C X0Y0	HP I/O Bank 64 B	PCIE4C X1Y0 (tandem)	GTY Quad 224 X1Y0-X1Y3 A [RS]
	HBM Bank 43		HBM Bank 83	

X19804-121620

Figure 1-153: **XCVU37P and XCVU47P Banks in FSVH2892 Package**

Figure 2-7 PCIe Hard IP Pin location

2.2.3 Dual port RAM

Two dual port RAMs, CtmRAM and IdenRAM, store the returned data from Identify command and SMART command, respectively. IdenRAM has an 8 Kbyte size and is used to store the 8 Kbyte output from the Identify command.

The data bus size for NVMe-IP and LAxi2Reg differ, with NVMe-IP having a 128-bit size and LAxi2Reg having a 32-bit size. As a result, IdenRAM is configured as an asymmetric RAM with different bus sizes for its Write and Read interfaces.

NVMe-IP also has a double-word enable, which allows it to write only 32-bit data in certain cases. The RAM setting on Xilinx IP tool supports write byte enable, so a small logic circuit was designed to convert the double word enable to be write byte enable, as shown in Figure 2-8.

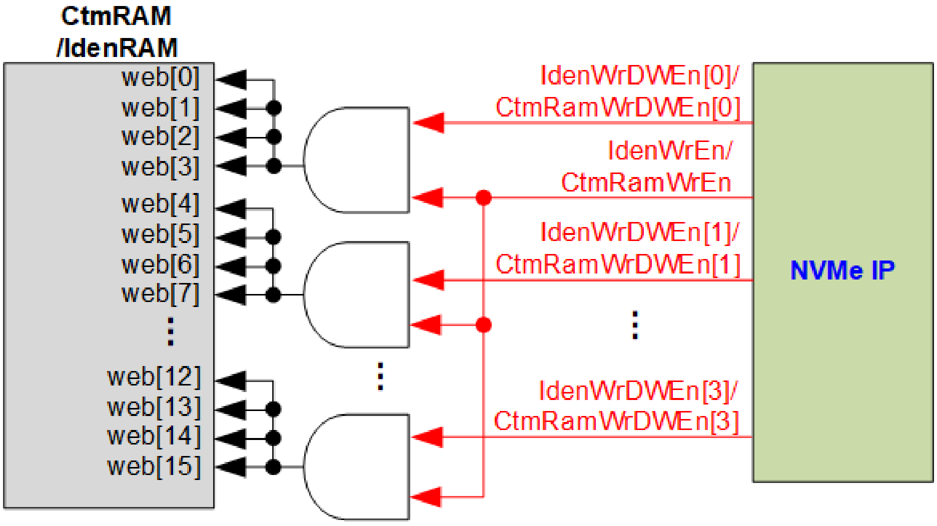


Figure 2-8 Byte write enable conversion logic

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

On the other hand, CtmRAM is implemented as a true dual-port RAM with two read ports and two write ports, and with byte write enable. A small logic circuit must be used to convert the double word enable of Custom interface to byte write enable, similar to IdenRAM. The true dual-port RAM is used to support additional features when a customized Custom command requires data input. A simple dual-port RAM is sufficient to support the SMART command, even though the data size returned from the SMART command is 512 bytes. However, CtmRAM is implemented with an 8Kbyte RAM for the customized Custom command.

2.3 CPU and Peripherals

The CPU system uses a 32-bit AXI4-Lite bus as the interface to access peripherals such as the Timer and UART. The system also integrates an additional peripheral to access NVMe-IP test logic by assigning a unique base address and address range. To support CPU read and write operations, the hardware logic must comply with the AXI4-Lite bus standard. LAxi2Reg module, as shown in Figure 2-9, is designed to connect the CPU system via the AXI4-Lite interface, in compliance with the standard.

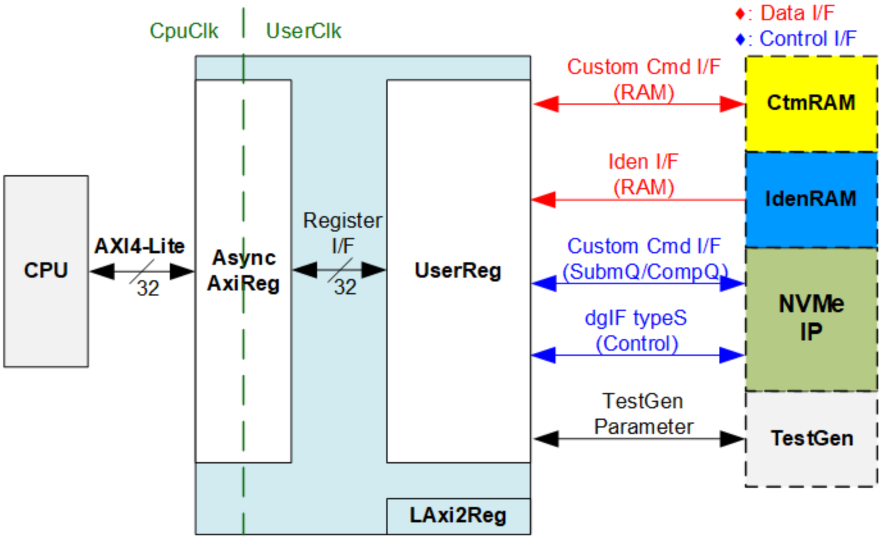


Figure 2-9 CPU and peripherals hardware

LAxi2Reg consists of AsyncAxiReg and UserReg. AsyncAxiReg converts AXI4-Lite signals into a simple Register interface with a 32-bit data bus size, similar to the AXI4-Lite data bus size. It also includes asynchronous logic to handle clock domain crossing between the CpuClk and UserClk domains.

UserReg includes the register file of the parameters and the status signals of other modules in the test system, including the CtmRAM, IdenRAM, NVMe-IP, and TestGen. More details of AsyncAxiReg and UserReg are explained below.

2.3.1 AsyncAxiReg

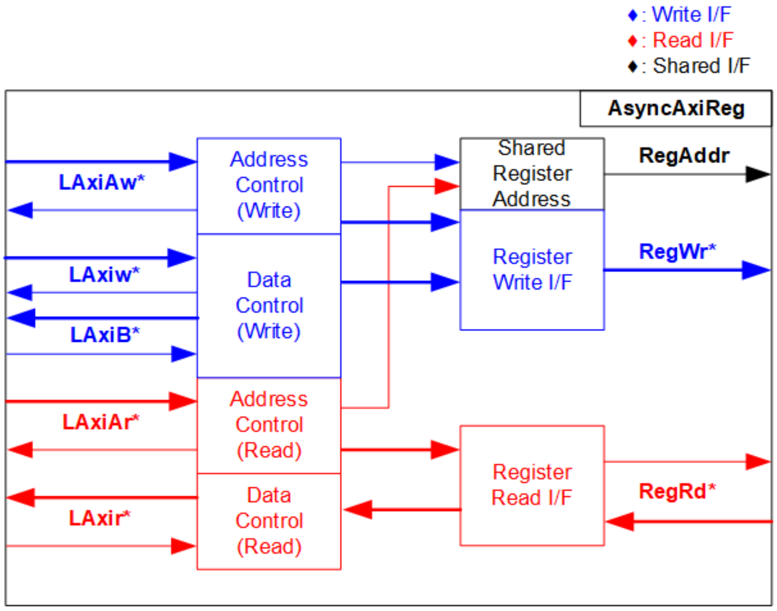


Figure 2-10 AsyncAxiReg Interface

The signal on AXI4-Lite bus interface can be grouped 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://github.com/Architech-Silica/Designing-a-Custom-AXI-Slave-Peripheral/blob/master/designing_a_custom_axi_slave_rev1.pdf

According to AXI4-Lite standard, the write channel and read channel operate independently for both control and data interfaces. Therefore, the logic within AsyncAxiReg to interface with AXI4-Lite bus is divided into four groups, i.e., Write control logic, Write data logic, Read control logic, and Read data logic, as shown in the left side of Figure 2-10. The Write control I/F and Write data I/F of the AXI4-Lite bus are latched and transferred to become the Write register interface with clock domain crossing registers. Similarly, the Read control I/F of AXI4-Lite bus is latched and transferred to the Read register interface, while Read data is returned from Register interface to AXI4-Lite via clock domain crossing registers. In the Register interface, RegAddr is a shared signal for write and read access, so it loads the value from LAxiAw for write access or LAxiAr for read access.

The Register interface is compatible with single-port RAM interface for write transaction. The read transaction of the Register interface has been slightly modified from RAM interface by adding the RdReq and RdValid signals to control read latency time. The address of Register interface is shared for both write and read transactions, so user cannot write and read the register at the same time. The timing diagram of the Register interface is shown in Figure 2-11.

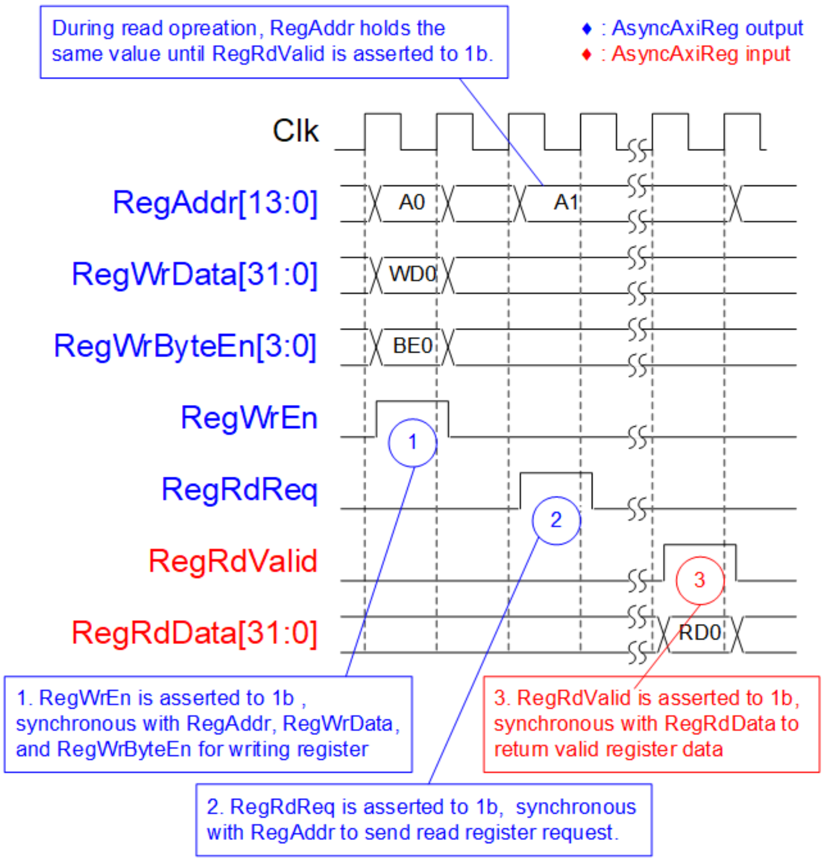

During read opreation, RegAddr holds the same value until RegRdValid is asserted to 1b.
♦ : AsyncAxiReg output
♦ : AsyncAxiReg input
Clk
RegAddr[13:0]
A0
A1
RegWrData[31:0]
WD0
RegWrByteEn[3:0]
BE0
RegWrEn
RegRdReq
RegRdValid
RegRdData[31:0]
RD0
1
2
3
1. RegWrEn is asserted to 1b , synchronous with RegAddr, RegWrData, and RegWrByteEn for writing register
3. RegRdValid is asserted to 1b, synchronous with RegRdData to return valid register data
2. RegRdReq is asserted to 1b, synchronous with RegAddr to send read register request.

Figure 2-11 Register interface timing diagram

1) Timing diagram to write register is similar to that of a single-port RAM. The RegWrEn signal is set to 1b, along with a valid RegAddr (Register address in 32-bit units), RegWrData (write data for the register), and RegWrByteEn (write byte enable). The byte enable consists of four bits that indicate the validity of the byte data. For example, bit[0], [1], [2], and [3] are set to 1b when RegWrData[7:0], [15:8], [23:16], and [31:24] are valid, respectively.
2) To read register, AsyncAxiReg sets the RegRdReq signal to 1b with a valid value for RegAddr. The 32-bit data is returned after the read request is received. The slave detects the RegRdReq signal being set to start the read transaction. In the read operation, the address value (RegAddr) remains unchanged until RegRdValid is set to 1b. The address can then be used to select the returned data using multiple layers of multiplexers.
3) The slave returns the read data on RegRdData bus by setting the RegRdValid signal to 1b. After that, AsyncAxiReg forwards the read value to the LAxir* interface.

2.3.2 UserReg

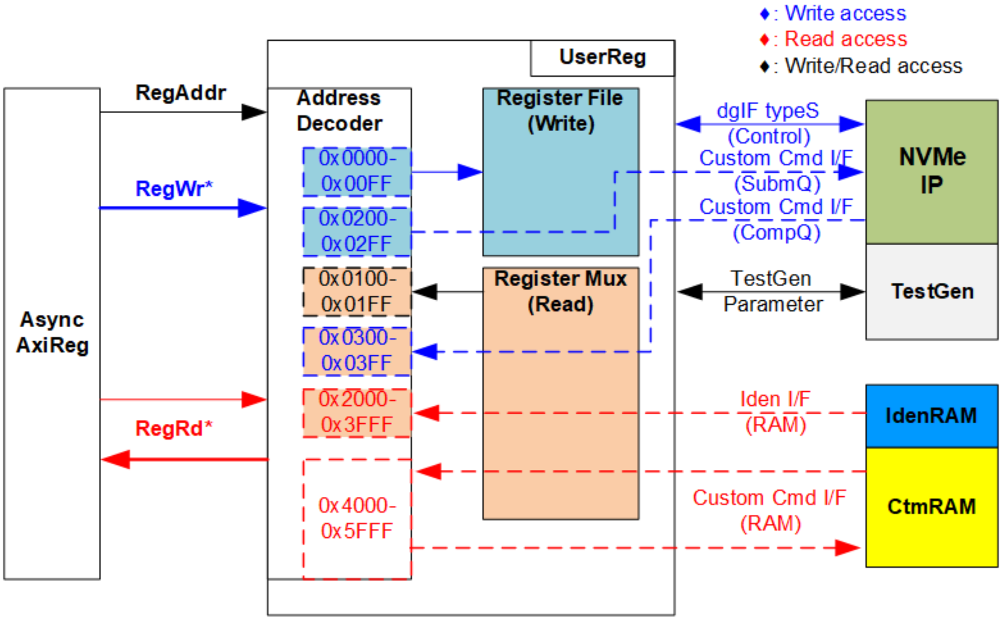


Figure 2-12 UserReg Interface

The UserReg module consists of an Address decoder, a Register File, and a Register Mux. The Address decoder decodes the address requested by AsyncAxiReg and selects the active register for either write or read transactions. The assigned address range in UserReg is divided into six areas, as shown in Figure 2-12.

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 when running SMART command.

The Address decoder decodes the upper bits of RegAddr to select the active hardware (NVMe-IP, TestGen, IdenRAM, or CtmRAM). The Register File within UserReg has a 32-bit bus size, so the write byte enable (RegWrByteEn) is not used in the test system and the CPU uses 32-bit pointer to set the hardware register.

To read the register, multi-level multiplexers (mux) select the data to return to CPU by using the address. The lower bits of RegAddr are fed to the submodule to select the active data from each submodule. While the upper bits are used in UserReg to select the returned data from each submodule. The total latency time of read data is equal to two clock cycles, and RegRdValid is created by RegRdReq by asserting two D Flip-flops. More details of the address mapping within the 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) 000b: Identify, 001b: Shutdown, 010b: Write SSD, 011b: Read SSD, 100b: SMART/Secure Erase, 110b: Flush, 101b/111b: 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 000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1, 100b-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 (0b: Idle, 1b: Busy) [1]: UserError of dgIF typeS (0b: Normal, 1b: Error) [2]: Data verification fail (0b: Normal, 1b: 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 (PCIESTS_INTREG)	[0]: PCIe linkup status from PCIe hard IP (0b: No linkup, 1b: linkup) [3:2]: PCIe link speed from PCIe hard IP (00b: Not linkup, 01b: PCIe Gen1, 10b: PCIe Gen2, 11b: PCIe Gen3) [7:4]: PCIe link width status from PCIe hard IP (0001b: 1-lane, 0010b: 2-lane, 0100b: 4-lane, 1000b: 8-lane) [13:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in Integrated Block for PCIe datasheet
BA+0x0114	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+0x013F	Expected value Word0-3 Reg (EXPPATW0-W3_INTREG)	The 128-bit expected data of the 1st failure when executing a Read command. 0x0130: Bit[31:0], 0x0134[31:0]: Bit[63:32], …, 0x013C[31:0]: Bit[127:96]
BA+0x0150 - BA+0x015F	Read value Word0-3 Reg (RDPATW0-W3_INTREG)	The 128-bit read data of the 1st failure when executing a 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 when executing a Read command
BA+0x0174	Data Failure Address(High) Reg (RDFAILNOH_INTREG)	[24:0]: Bit[56:32] of the byte address of the 1st failure when executing a 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, Secure Erase, and Flush commands. 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)	4KB Identify Controller Data structure
BA+0x3000– BA+0x3FFF Rd	Identify Namespace Data (IDENNAME_CHARREG)	4KB Identify Namespace Data structure
BA+0x4000– BA+0x5FFF Wr/Rd	Custom command Ram (CTMRAM_CHARREG)	Connect to 8KB CtmRAM interface for storing 512-byte data output from SMART Command.

3 CPU Firmware

3.1 Test firmware (nvmeiptest.c)

The CPU follows these steps upon system startup to complete the initialization process.

1) Initialize UART and Timer settings.
2) Wait for the PCIe connection to become active (PCIESTS_INTREG[0]=1b).
3) Wait for NVMe-IP to complete its own initialization process (USRSTS_INTREG[0]=0b). If errors are encountered, the process will stop and display an error message.
4) Display the status of the PCIe link, including the number of lanes and the speed, by reading PCIESTS_INTREG[7:2] status.
5) Display the main menu with options to run seven commands for NVMe-IP, i.e., Identify, Write, Read, SMART, Flush, Secure Erase, and Shutdown.

More details on the sequence for each command in the CPU firmware are described in the following sections.

3.1.1 Identify Command

The sequence for the firmware when the Identify command is selected by user is as follows.

1) Set bits[2:0] of USRCMD_INTREG to 000b to send the Identify command request to NVMe-IP. The busy flag (USRSTS_INTREG[0]) will then change from 0b to 1b.
2) The CPU waits until the operation is completed or an error is detected by monitoring USRSTS_INTREG[1:0].
   - Bit[0] is de-asserted to 0b when the command is completed. The data of Identify command returned by NVMe-IP will be stored in IdenRAM.
   - Bit[1] is asserted when an error is detected. In this case, the error message will be displayed on the console to show the error details, and the process will be stopped.

3) Once the busy flag (USRSTS_INTREG[0]) is de-asserted to 0b, the CPU proceeds to display information that has been decoded from LBASIZEL/H_INTREG, which includes the SSD capacity and LBA unit size. Besides, further information, such as the SSD model, can be retrieved from the IdenRAM (IDENCTRL_CHARREG).

3.1.2 Write/Read Command

The sequence for the firmware when the Write/Read command is selected is as follows.

1) Receive start address, transfer length, and test pattern from Serial console. If any inputs are invalid, the operation will be cancelled.
   Note: If LBA unit size = 4 KB, the start address and transfer length must align to 8.
2) After obtaining all the inputs, set them to USRADRL/H_INTREG, USRLENL/H_INTREG, and PATTSEL_INTREG.
3) To execute either the Write or Read command, set bits[2:0] of USRCMD_INTREG to 010b or 011b, respectively. This sends the command request to the NVMe-IP. Once the command is issued, the busy flag of NVMe-IP (USRSTS_INTREG[0]) will change from 0b to 1b.
4) The CPU waits until the operation is completed or an error (excluding verification error) is detected by monitoring USRSTS_INTREG[2:0].
   - Bit[0] is de-asserted to 0b when the command is completed.
   - Bit[1] is asserted when an error is detected. In this case, the error message will be displayed on the console to show the error details, and the process will be stopped.
   - Bit[2] is asserted when data verification fails. In this case, the verification error message will then be displayed on the console, but the CPU will continue to run until the operation is completed or the user inputs any key to cancel the operation.

   While the command is running, the current transfer size, read from CURTESTSIZE_INTREG, will be displayed every second.

5) Once the busy flag (USRSTS_INTREG[0]) is de-asserted to 0b, CPU will calculate and display the test result on the console including the total time usage, total transfer size, and transfer speed.

3.1.3 SMART Command

The sequence of the firmware when the SMART command is selected is as follows.

1) The 16-Dword of the Submission Queue entry (CTMSUBMQ_STRUCT) is set to the SMART command value.
2) Set bits[2:0] of USRCMD_INTREG[2:0] to 100b to send the SMART command request to NVMe-IP. The busy flag (USRSTS_INTREG[0]) will then change from 0b to 1b.
3) The CPU waits until the operation is completed or an error is detected by monitoring USRSTS_INTREG[1:0].
   - Bit[0] is de-asserted to 0b after the operation is finished. The data of SMART command returned by NVMe IP will be stored in CtmRAM.
   - Bit[1] is asserted to 1b, indicating an error. In this case, the error message will be displayed on the console with details decoded from USRERRTYPE_INTREG[31:0]. The process will then stop.

4) After the busy flag (USRSTS_INTREG[0]) is de-asserted to 0b, the CPU will display information decoded from CtmRAM (CTMRAM_CHARREG), including Remaining Life, Percentage Used, Temperature, Total Data Read, Total Data Written, Power On Cycles, Power On Hours, and Number of Unsafe Shutdown.

   For more information on the SMART log, refer to the NVM Express Specification.
   https://nvmexpress.org/specifications/

3.1.4 Flush Command

The sequence of the firmware when the Flush command is selected is as follows.

1) The 16-Dword of the Submission Queue entry (CTMSUBMQ_STRUCT) is set to the Flush command value.
2) Set bits[2:0] of USRCMD_INTREG to 110b to send Flush command request to NVMe-IP. The busy flag of NVMe-IP (USRSTS_INTREG[0]) will then change from 0b to 1b.
3) The CPU waits until the operation is completed or an error is detected by monitoring USRSTS_INTREG[1:0].
   - Bit[0] is de-asserted to 0b after the operation is finished. The CPU will then return to the main menu.
   - Bit[1] is asserted to 1b, indicating an error. In this case, the error message will be displayed on the console with details decoded from USRERRTYPE_INTREG[31:0]. The process will then stop.

3.1.5 Secure Erase Command

The sequence of the firmware when the Secure Erase command is selected is as follows.

1) The 16-Dword of the Submission Queue entry (CTMSUBMQ_STRUCT) is set to the Secure Erase command value.
2) Set NVMTIMEOUT_INTREG to 0 to disable timer to prevent the timeout error.
3) Set bits[2:0] of USRCMD_INTREG[2:0] to 100b to send Secure Erase command request to NVMe-IP. The busy flag of NVMe-IP (USRSTS_INTREG[0]) will then change from 0b to 1b.
4) The CPU waits until the operation is completed or an error is detected by monitoring USRSTS_INTREG[1:0].
   - Bit[0] is de-asserted to 0b after the operation is finished. The CPU will then proceed to the next step.
   - Bit[1] is asserted to 1b, indicating an error. In this case, the error message will be displayed on the console with details decoded from USRERRTYPE_INTREG[31:0]. The process will then stop.

5) After completing the command, the timer is re-enabled to generate timeout error in NVMe-IP by setting NVMTIMEOUT_INTREG to the default value.

3.1.6 Shutdown Command

The sequence of the firmware when the Shutdown command is selected is as follows.

1) Set bits[2:0] of USRCMD_INTREG to 001b to send the Shutdown command request to NVMe-IP. The busy flag of NVMe-IP (USRSTS_INTREG[0]) will then change from 0b to 1b.
2) The CPU waits until the operation is completed or an error is detected by monitoring USRSTS_INTREG[1:0].
   - Bit[0] is de-asserted to 0b after the operation is finished. The CPU will then proceed to the next step.
   - Bit[1] is asserted to 1b, indicating an error. The error message will be displayed on the console with details decoded from USRERRTYPE_INTREG[31:0]. The process will then stop.

3) After Shutdown command completes, both the SSD and NVMe-IP will become inactive and the CPU will be unable to receive any new commands from the user. To continue testing, the user must power off and power on the 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	Execute SMART command as outlined in section 3.1.3 (SMART Command) or execute Flush command as outlined in section 3.1.4 (Flush Command).

unsigned long long get_cursize(void)	
Parameters	None
Return value	Read value of CURTESTSIZEL/H_INTREG
Description	The value of CURTESTSIZEL/H_INTREG is read and converted to byte units before being returned as the result of the function.

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	Execute Identify command as outlined in section 3.1.1 (Identify Command).

int setctm_erase(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the NVMe-IP
Description	Set Secure Erase command to CTMSUBMQ_STRUCT and call exec_ctm function to execute Secure Erase 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 execute 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 execute 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 the corresponding error message. Also, call show_pciestat function to check the hardware's debug signals.

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. Also, debug signals are read from NVMTESTPIN_INTREG.

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 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 MB or GB 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), the 8-byte SMART data is displayed in decimal units. If the input value exceeds this limit, an overflow message is displayed.

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

void show_vererr(void)	
Parameters	None
Return value	None
Description	Read RDFAILNOL/H_INTREG (error byte address), EXPPATW0-W3_INTREG (expected value), and RDPATW0-W3_INTREG (read value) to display verification error details on the console.

Void shutdown_dev(void)	
Parameters	None
Return value	None
Description	Execute Shutdown command as outlined in section 3.1.6 (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	Execute Write command or Read command as outlined in section 3.1.2 (Write/Read Command). In this function, 'show_result' is called to compute and display transfer performance in Write/Read command.

4 Example Test Result

The performance results of executing Write and Read commands on the demo system using a 512 GB Samsung 970 Pro are shown in Figure 4-1.

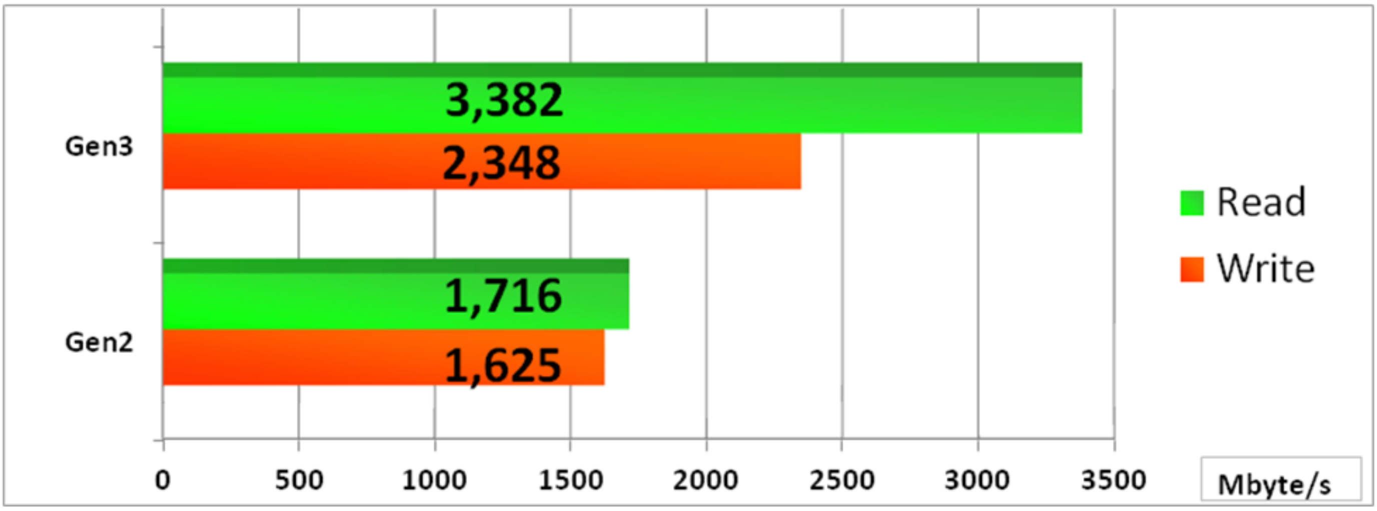


Figure 4-1 Test Performance of NVMe-IP demo by using Samsung 970 Pro SSD

When using the KCU105 board with PCIe Gen3, the system achieves a write performance of approximately 2300 MB/s and a read performance of around 3300 MB/s. However, using PCIe Gen2 on the ZC706 board results in slower performance compared to Gen3. On Gen2, the write and read performance is around 1600-1700 MB/sec.

5 Revision History

Revision	Date	Description
3.11	12-Dec-23	Add Secure Erase feature
3.10	23-Jun-23	Update show_size description in firmware function list
3.09	10-May-23	Update PCIe hard IP Pin location
3.08	17-Mar-22	Update firmware and register type
3.07	7-Jun-21	Update Register map
3.06	5-Mar-21	Update Function name
3.05	27-Aug-20	Correct the information
3.04	21-Feb-20	Add Function list in Test firmware
3.03	28-Jan-20	Remove header of all zero and all one patterns
3.02	3-May-19	Update SSD performance and system information
3.01	23-Nov-18	Add dword enable for RAM interface
3.00	19-Jul-18	Support Shutdown, SMART and Flush command
2.01	27-Jul-17	Add LFSR pattern
2.00	12-Jun-17	New NVMe-IP version
1.02	6-Dec-16	Change buffer from DDR to BRAM
1.01	5-Sep-16	Add CURTESTSIZE register
1.00	1-Jun-16	Initial Release