NVMe-IP for Gen5 reference design manual

Rev1.00 15-Aug-24

1 Overview

NVM Express (NVMe) is a specification that defines the interface between a host controller and a solid-state drive (SSD) through PCI Express. It optimizes the process of issuing commands and receiving completions by utilizing only two registers (Command Issue and Command Completion), and supports parallel operations with up to 64K commands in a single queue. This significantly improves transfer performance for both sequential and random access operations.

In the PCIe SSD market, two standards are commonly used: AHCI and NVMe. AHCI is an older standard designed for SATA hard disk drives, while NVMe is optimized specifically for non-volatile memory such as SSDs. For a detailed comparison between the AHCI and NVMe protocols, refere to the document titled “A Comparison of NVMe and AHCI” available at this link.
https://sata-io.org/system/files/member-downloads/NVMe%20and%20AHCI_%20_long_.pdf

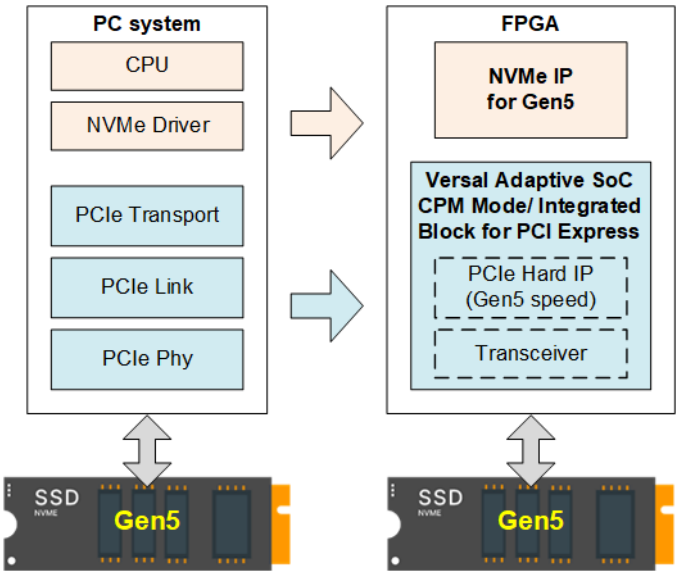

Figure 1-1 NVMe protocol layer

To access an NVMe Gen5 SSD, a typical system employs an NVMe driver running on a processor, as shown on the left side of Figure 1-1. The physical connection between the host and the NVMe device is made via as PCIe connector, which follows a one-to-one connection model, meaning that each PCIe host connects directly to a single PCIe device without requiring a PCIe switch.

The NVMe-IP implements the NVMe driver entirely in hardware logic, allowing access to NVMe SSDs without the need for a processor or software-based drivers. By integrating NVMe-IP into an FPGA, the system eliminates the overhead associated with software-hardware communication, resulting in higher performance for both write and read operations with NVMe SSDs.

2 Hardware

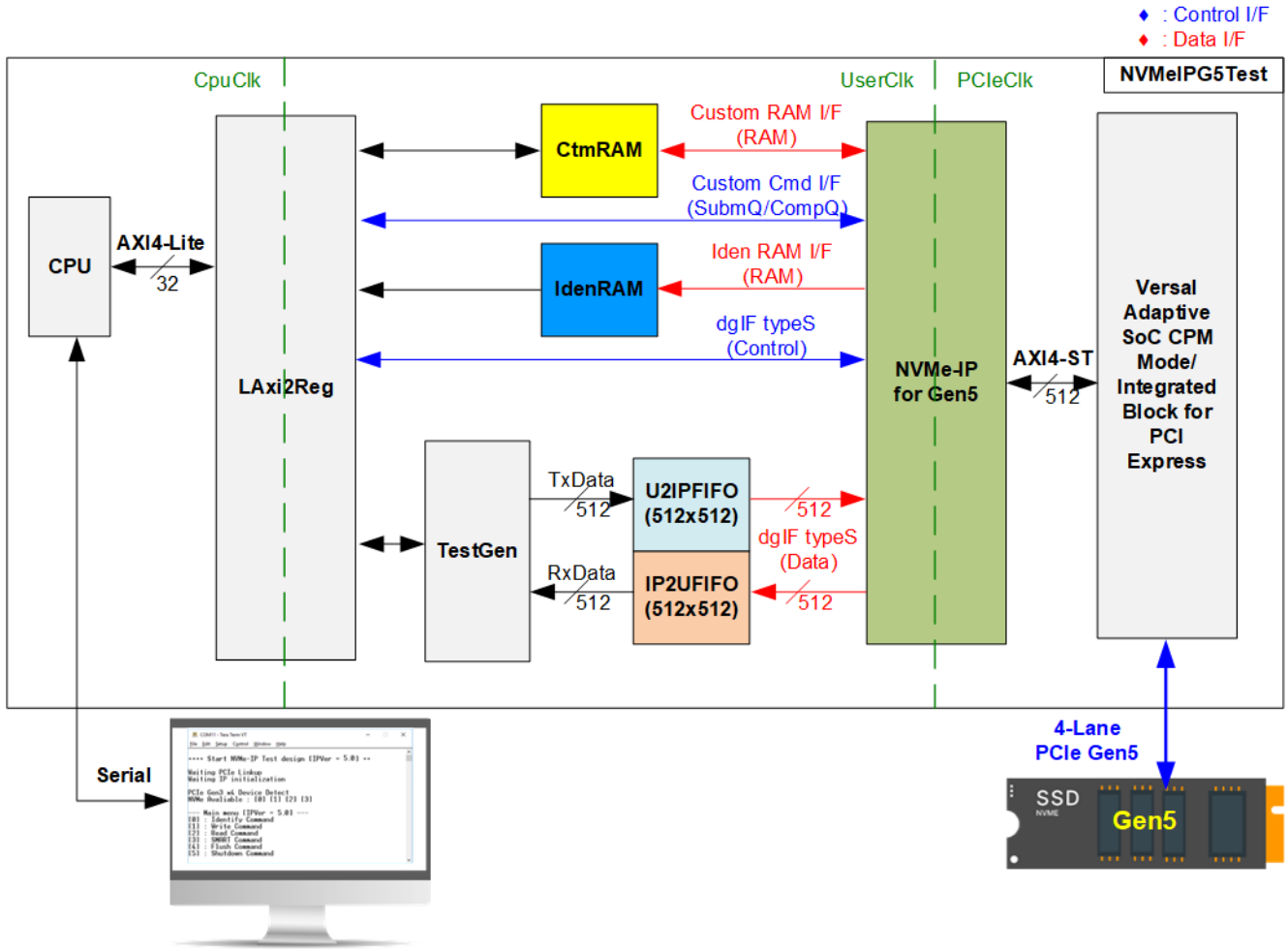

Figure 2-1 NVMe-IP for Gen5 demo hardware

The hardware modules in the test system are divided into three main parts: the test function (TestGen), the NVMe function (comprising CtmRAM, IdenRAM, U2IPFIFO, IP2UFIFO, NVMe-IP for Gen5, and the PCIe block), and the CPU system (including CPU and LAxi2Reg).

The TestGen module interfaces with the NVMe-IP for Gen5's user interface and is responsible for generating the data stream for Write commands and verifying the data stream of Read commands. The write and read data streams are stored in two FIFOs (U2IPFIFO and IP2UFIFO). TestGen continuously writes or reads data when the FIFO is ready, ensuring optimal transfer performance for system evaluation.

The NVMe section includes the NVMe-IP for Gen5 and the PCIe hard IP (Versal Adaptive SoC CPM Mode/Integrated Block for PCI Express), providing direct access to an NVMe Gen5 SSD without requiring a PCIe switch. Command requests and parameters for each command, which are inputs to the NVMe-IP for Gen5, are controlled by the CPU through the LAxi2Reg module. Additionally, the data interface for both Custom and Identify commands connects to RAMs accessible by the CPU.

The CPU is connected to the LAxi2Reg module, interfacing with the NVMe test logics. Integrating the CPU into the test system allows users to set test parameters and monitor the status via a Serial console. The CPU also facilitates the execution of multiple test cases to verify the functionality of the NVMe-IP. The default firmware for the CPU includes functions for executing NVMe commands using the NVMe-IP for Gen5.

Figure 2-1 displays three clock domains used in the design: CpuClk, UserClk, and PCIeClk.

- CpuClk serves as the clock domain for the CPU and its peripherals, requiring a stable clock independent of other hardware components.
- UserClk is the clock domain used for the operation of the NVMe-IP for Gen5, RAM, and TestGen. According to the NVMe-IP for Gen5 datasheet, the frequency of UserClk must be equal to or greater than PCIeClk. In the reference design, UserClk is set to 280 MHz for PCIe Gen5 operation.
- PCIeClk is generated by the PCIe hard IP and is synchronized with the 512-bit AXI4-stream. It operates at 250 MHz for 4-lane PCIe Gen5.

Further hardware details are provided in the subsequent sections.

2.1 TestGen

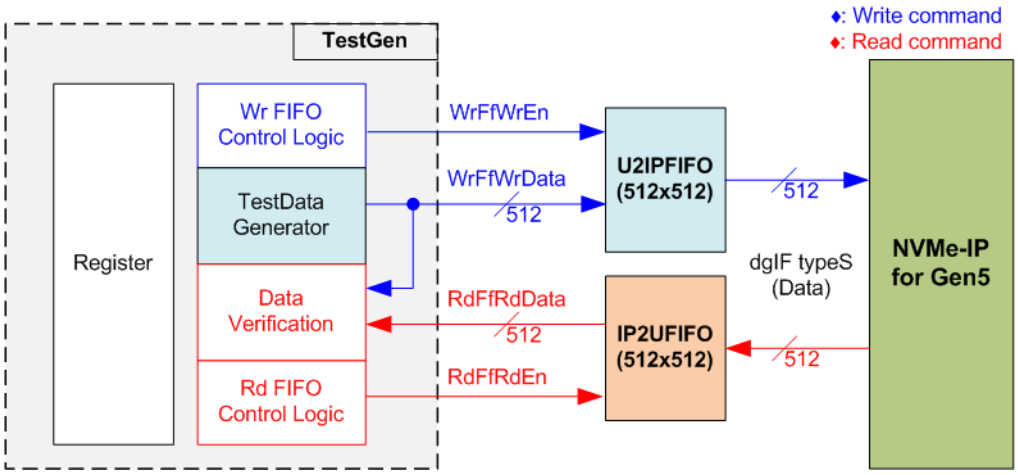

Figure 2-2 TestGen interface

The TestGen module manages the data interface of the NVMe-IP, facilitating data transfer for both Write and Read commands. During a Write command, TestGen generates a 512-bit test data and sends it to NVMe-IP via U2IPFIFO. In contrast, for a Read command, the data is received from IP2UFIFO and compared against the expected value to ensure data accuracy. TestGen's data bandwidth is set to match the NVMe-IP, running at the same clock and using the same data bus size to optimize performance.

TestGen's control logic ensures that the Write or Read enable is asserted to 1b whenever the corresponding FIFO is ready to transfer data. This allows both U2IPFIFO and IP2UFIFO to be ready for data transfer to and from the NVMe-IP, achieving optimal write and read performance with the SSD.

The module provides flexibility by allowing the user to configure test parameters through the console, including the start transfer address, total transfer size, transfer direction, and test pattern selector. These parameters are stored in the Register block. The detailed hardware logic of TestGen is illustrated in Figure 2-3.

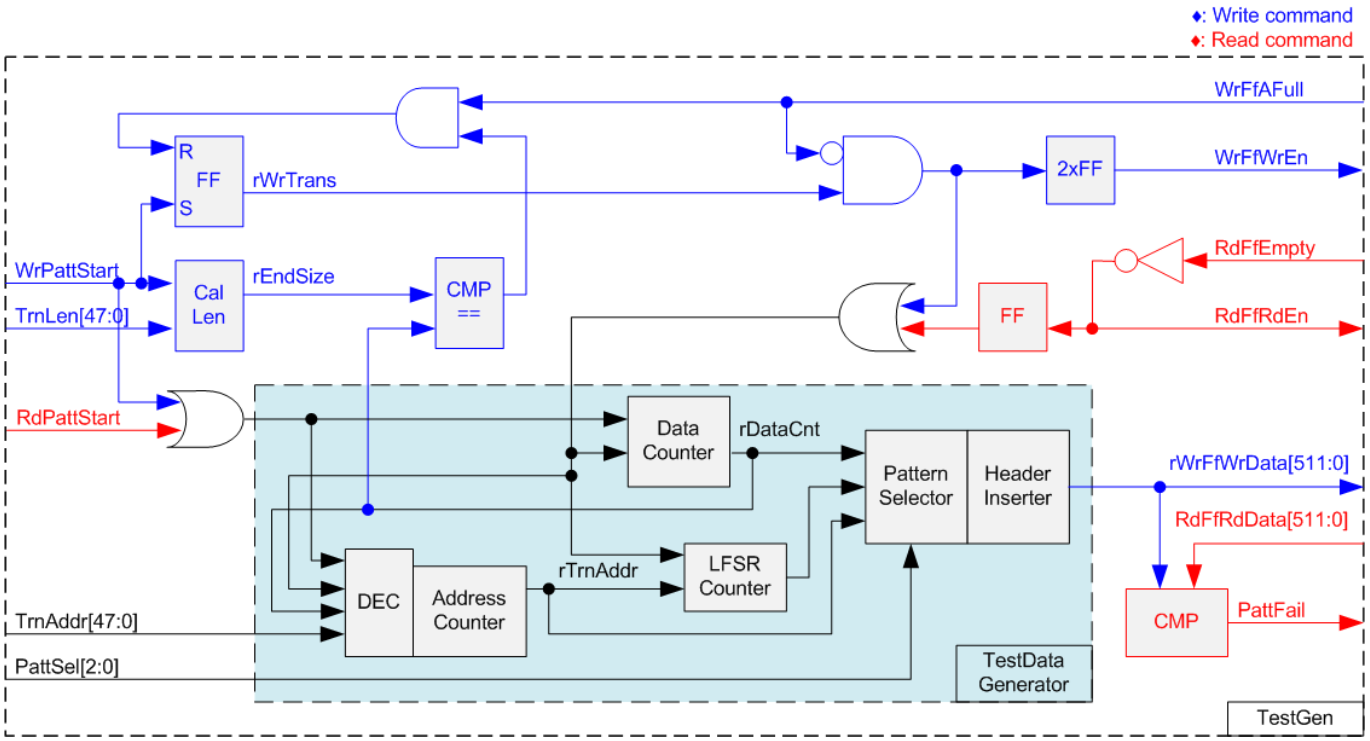

Figure 2-3 TestGen hardware

At the right side of Figure 2-3, the flow control signals for the FIFO, WrFfAFull and RdFfEmpty, are used to manage data flow. During a write operation, when the FIFO is nearly full (WrFfAFull=1b), WrFfWrEn is de-asserted to 0b to pause data transmission to the FIFO. On the other hand, during a read operation, if data is present in the FIFO (RdFfEmpty=0b), RdFfRdEn is asserted to 1b to read data from the FIFO for comparison against the expected data.

The left side of Figure 2-3 shows the logic that counts the transfer size. Once the total data count (rDataCnt) matches the user-defined end size (rEndSize), the FIFO's write or read enable signal is de-asserted to 0b. The lower portion of Figure 2-3 details the logic for generating test data during write operations or verifying data during read operations. User can choose from five test patterns: all-zero, all-one, 32-bit incremental data, 32-bit decremental data, and LFSR, selected by Pattern Selector. When generating all-zero or all-one pattern, the entire data block is filled with zero or one, respectively. Other patterns are generated by splitting the data into two distinct sections to ensure uniqueness across each 512-byte block, as shown in Figure 2-4.

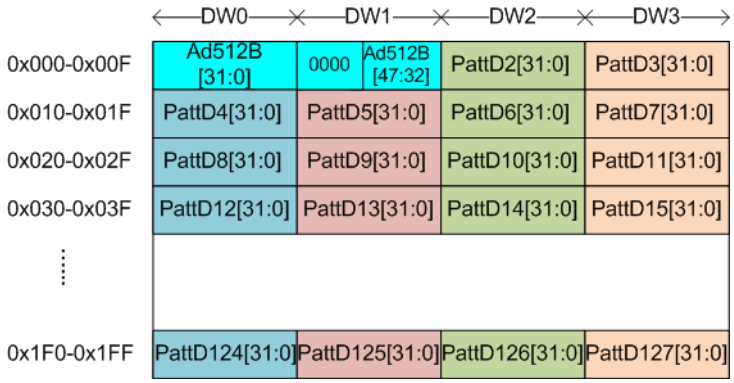

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

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 (from Dword#2 to Dword#127). The header is generated using the address in 512-byte units (rTrnAddr), controlled by the Address counter block. The initial value of the address counter is set by the user (TrnAddr) and is incremented after each 512-byte data transfer. The data in the remaining Dwords (DW#2 – DW#127) depends on the pattern selector, which can be 32-bit incremental data, 32-bit decremental data, or LFSR. The 32-bit incremental data is generated using the Data counter, while the decremental data is created by applying a NOT operation to the incremental data. The LFSR pattern is generated using the LFSR counter, based on the equation: x^31 + x^21 + x + 1.

To generate 512-bit test data for the LFSR pattern, the data is split into four 128-bit sets, each with different start value, as illustarted in Figure 2-5.

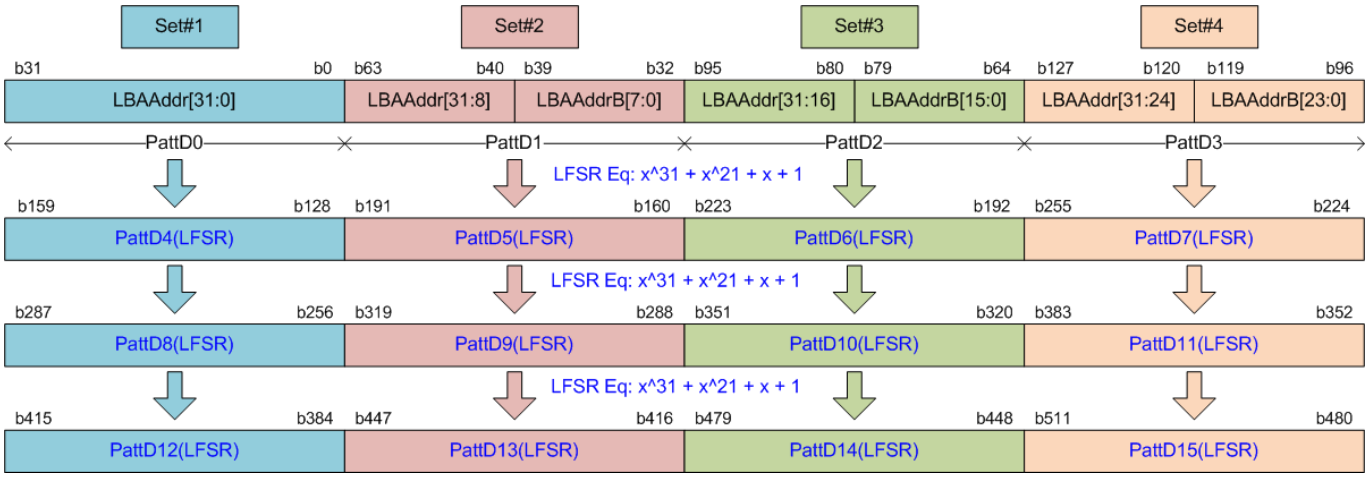

Figure 2-5 512-bit LFSR Pattern in TestGen

Using a look-ahead technique, each clock cycle generates four 32-bit LFSR values or 128-bit data, represented by the same color in Figure 2-5. The start value of each data set is derived from combination of the 32-bit of LBAAddr and the NOT operation applied to specific bits of the LBAAddr (LBAAddrB), creating unique start values for each 128-bit data set. The generated test data is either written to the FIFO as write data or used as expected data for verification against the read data from the FIFO. If a mismatch occurs during data verification, the failure flag is set to 1b.

The timing diagram for writing data to the FIFO is outlined below.

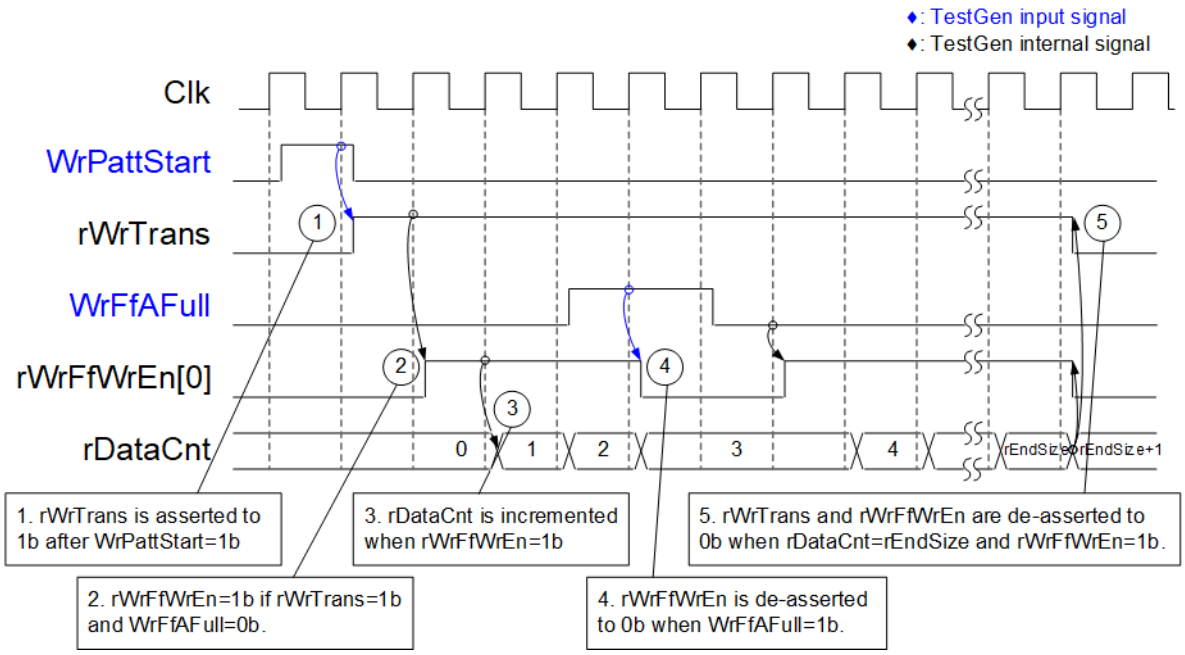

Figure 2-6 Timing diagram of Write operation in TestGen

1) The write operation is initiated by setting the WrPattStart signal to 1b for a single clock cycle, which triggers the assertion of rWrTrans, enabling the control logic to generate the write enable signal for the FIFO.
2) If two conditions are met - rWrTrans is asserted to 1b during the write operation, and the FIFO is not full (indicated by WrFfAFull=0b) - the write enable (rWrFfWrEn) for the FIFO is asserted to 1b.
3) The write enable signal is fed back to the counter to track the total amount of data written during the operation.
4) If the FIFO becomes almost full (WrFfAFull=1b), the write process is paused by de-asserting rWrFfWrEn to 0b.
5) The write operation is completed when the total data count (rDataCnt) reaches the user-defined end value (rEndSize). At this point, both rWrTrans and rWrFfWrEn are de-asserted to 0b.

For the read operation, the read enable signal for the FIFO is controlled by the FIFO's empty flag. Unlike the write process, the read enable signal is not controlled by the total data count and is not initiated by a start flag. Once the read enable is asserted to 1b, both the data counter and the address counter are incremented to track the total amount of data read and generate the expected header values for data verification.

2.2 NVMe

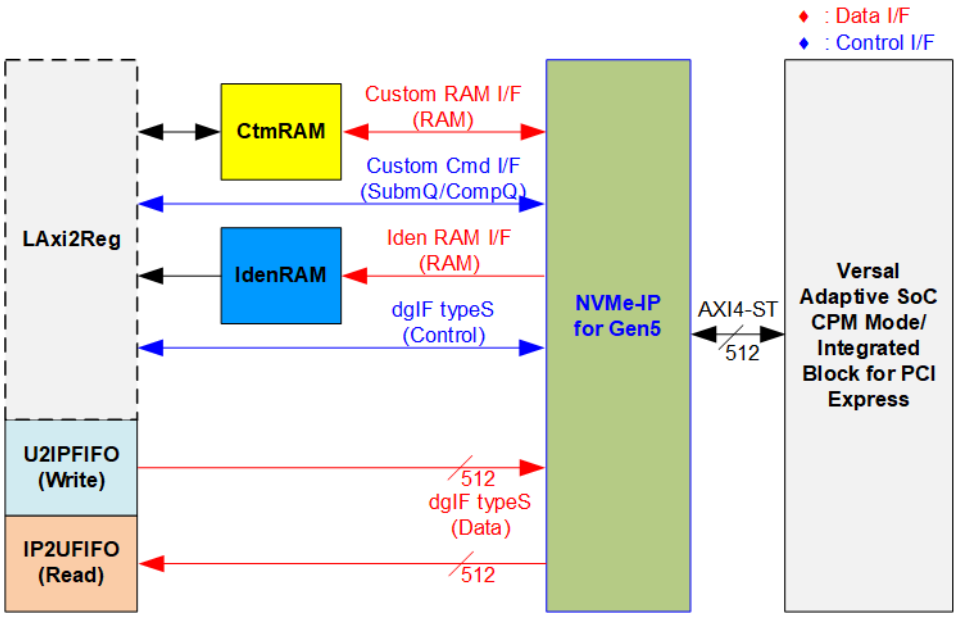

Figure 2-7 NVMe hardware

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 either from the Custom command interface or the dgIF typeS interface, depending on the type of command. For instance, the Custom command interface is used for operating SMART, Secure Erase, or Flush command.

On the other hand, the data interface of the NVMe-IP features four different interfaces, each with a 512-bit data bus width. These interfaces include Custom command RAM interface (bi-directional), Identify interface (unidirectional), FIFO input interface (dgIF typeS and unidirectional), and FIFO output interface (dgIF typeS and unidirectional).

In the reference design, the Custom command RAM interface is typically used for one-way data transfer, such as when the NVMe-IP sends SMART data to the LAxi2Reg module.

2.2.1 NVMe-IP

The NVMe-IP implements the NVMe protocol on the host side, enabling direct access to an NVMe Gen5 SSD without the need for a PCIe switch. It supports seven commands: Write, Read, Identify, Shutdown, SMART, Secure Erase, and Flush. Further details about the NVMe-IP can be found in the datasheet.
https://dgway.com/products/IP/NVMe-IP/dg_nvmeip_datasheet_g5_amd/

2.2.2 Versal Adaptive SoC CPM Mode / Integrated Block for PCI Express

This block is a hard IP integrated into certain AMD Xilinx Versal devices, supporting PCIe Gen5 speed. It implements the Physical, Data Link, and Transaction Layers of the PCIe specification. More details can be found in the following AMD Xilinx documents.

PG343: Versal Adaptive SoC Integrated Block for PCI Express
https://www.xilinx.com/products/intellectual-property/pcie-versal.html

PG346: Versal Adaptive SoC CPM Mode for PCI Express
https://www.xilinx.com/products/intellectual-property/cpm-pcie.html

When using the Versal Adaptive SoC Integrated Block for PCI Express, the PCIe hard IP is created using the IP wizard. It is crucial to ensure that the PCIe Block Location is adjacent to the transceiver pin connected to the SSD. More details about the location of the PCIe hard IP and transceiver can be found in the following document.

AM013: Versal Adaptive SoC Packaging and Pinouts
https://docs.amd.com/r/en-US/am013-versal-pkg-pinout

An example of the PCIe hard IP location on the XCVH1582-VSVA3697 device is shown in Figure 2-8.

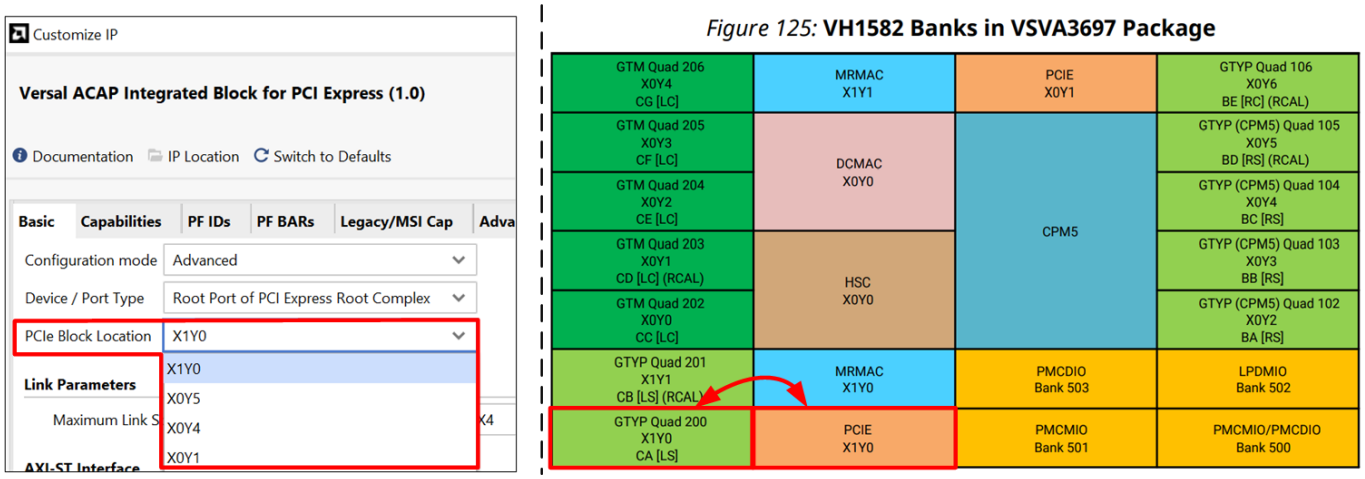

Figure 2-8 PCIe Hard IP pin location

2.2.3 Dual port RAM

Two dual port RAMs, CtmRAM and IdenRAM, are used to store data returned from the Identify and SMART commands, respectively. IdenRAM has an 8 KB capacity, sufficient to store the 8 KB output from the Identify command.

The data bus size for the NVMe-IP and LAxi2Reg differ: NVMe-IP uses a 512-bit bus, while LAxi2Reg uses a 32-bit size. Consequently, IdenRAM is configured as an asymmetric RAM, with different bus sizes for its Write and Read interfaces.

The NVMe-IP also includes a double-word enable feature, which allows for writing only 32-bit data in certain cases. To accommodate this, IdenRAM is configured with a write byte enable feature, which is supported by AMD Xilinx IP tools. A small logic circuit converts the double-word enable to a write byte enable, as shown in Figure 2-9.

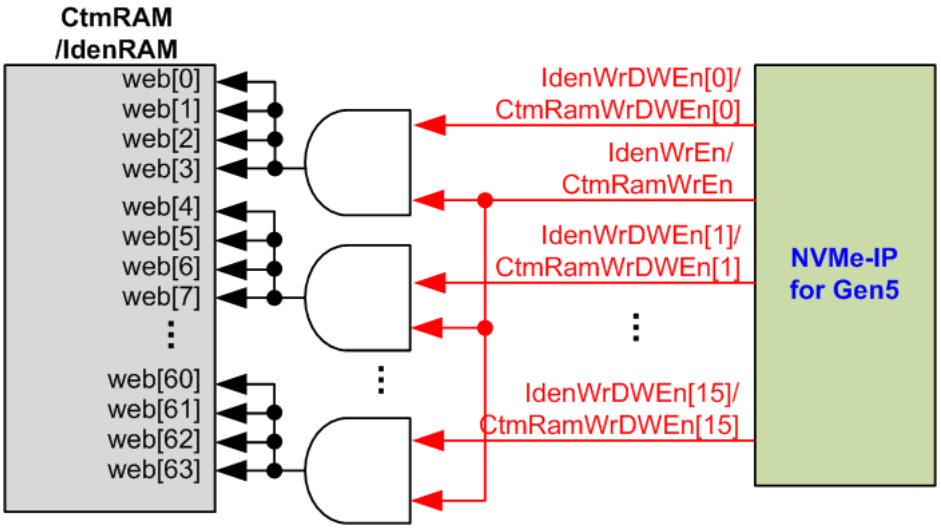

Figure 2-9 Byte write enable conversion logic

The input to the AND gate is bit[0] of WrDWEn and the WrEn signal. The output of the AND gate is connected to bits[3:0] of IdenRAM's write byte enable. Similarly, bit[1], [2], ..., [15] of WrDWEn are connected to bits[7:4], [11:8], ..., [63:60] of IdenRAM's write byte enable.

On the other hand, CtmRAM is implemented as a true dual-port RAM with two read ports and two write ports, along with byte write enable functionality. Similar to IdenRAM, a small logic circuit is used to convert the double-word enable from the Custom interface to byte write enable. The true dual-port RAM architecture allows for additional flexibility when a customized Custom command requires data input. While a simple dual-port RAM would be sufficient to support the SMART command (which returns only 512 bytes of data), CtmRAM is implemented with an 8KB capacity to handle the customized Custom command requirements.

2.3 CPU and Peripherals

The CPU system uses a 32-bit AXI4-Lite bus interface to access peripherals, such as the Timer and UART. Additionally, the system integrates a 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. The LAxi2Reg module, as shown in Figure 2-10, connects the CPU system via the AXI4-Lite interface in full compliance with the standard.

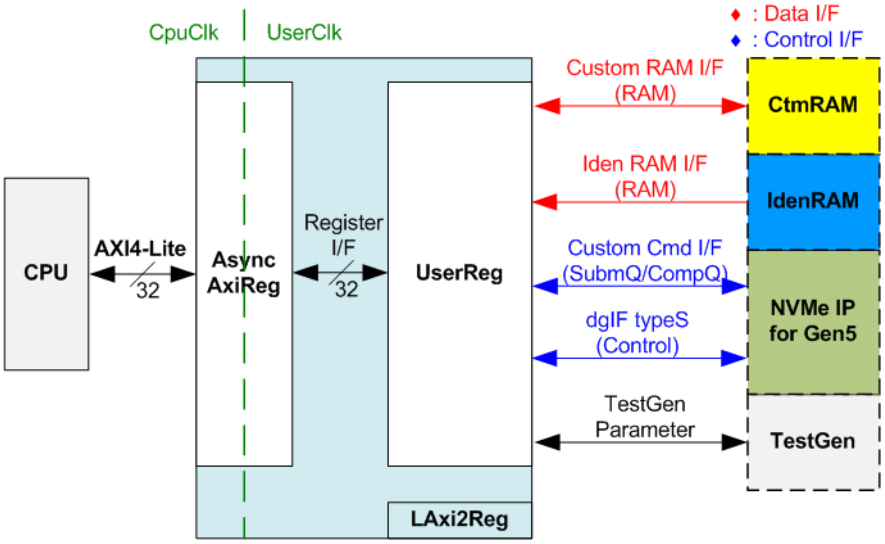

Figure 2-10 CPU and peripherals hardware

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

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

2.3.1 AsyncAxiReg

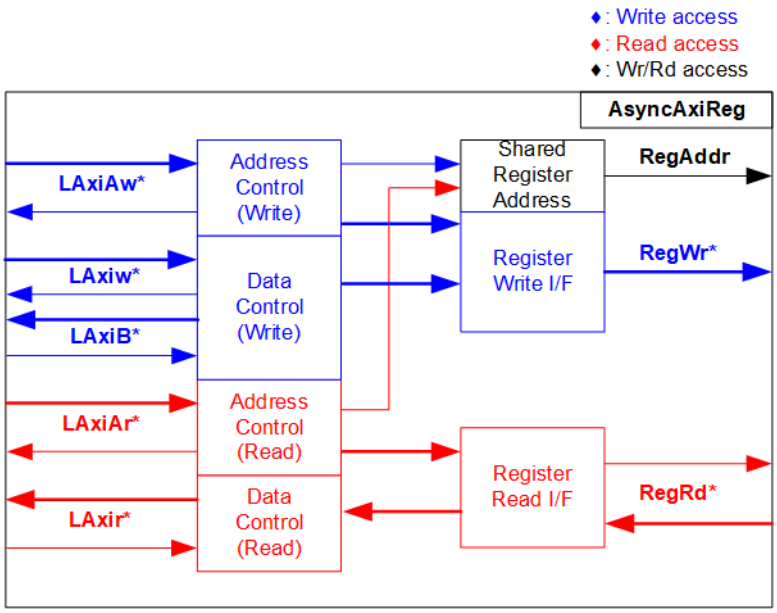

Figure 2-11 AsyncAxiReg Interface

The signal on AXI4-Lite bus interface can be grouped into five categories, 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 on building custom logic for the AXI4-Lite bus can be found in the following document.
https://github.com/Architech-Silica/Designing-a-Custom-AXI-Slave-Peripheral/blob/master/designing_a_custom_axi_slave_rev1.pdf

According to the AXI4-Lite standard, the write and read channels operate independently for both control and data interfaces. Therefore, the logic within AsyncAxiReg to interface with AXI4-Lite bus is divided into four groups: Write control logic, Write data logic, Read control logic, and Read data logic, as shown on the left side of Figure 2-11. The Write control and Write data interfaces of the AXI4-Lite bus are latched and transferred to the Write register interface using clock domain crossing registers. Similarly, the Read control I/F of AXI4-Lite bus is latched and transferred to the Read register interface, while the Read data is returned from Register interface to the AXI4-Lite bus via clock domain crossing registers. In the Register interface, RegAddr is a shared signal for both write and read operations. It loads the value from LAxiAw* for write access or LAxiAr* for read access.

The Register interface is compatible with single-port RAM interface during write transaction. However, for read transactions, the Register interface includes additional signals, RdReq and RdValid, to control read latency. Since the address of Register interface is shared for both write and read operations, the user cannot write to and read from the register simultaneously. The timing diagram for the Register interface is shown in Figure 2-12.

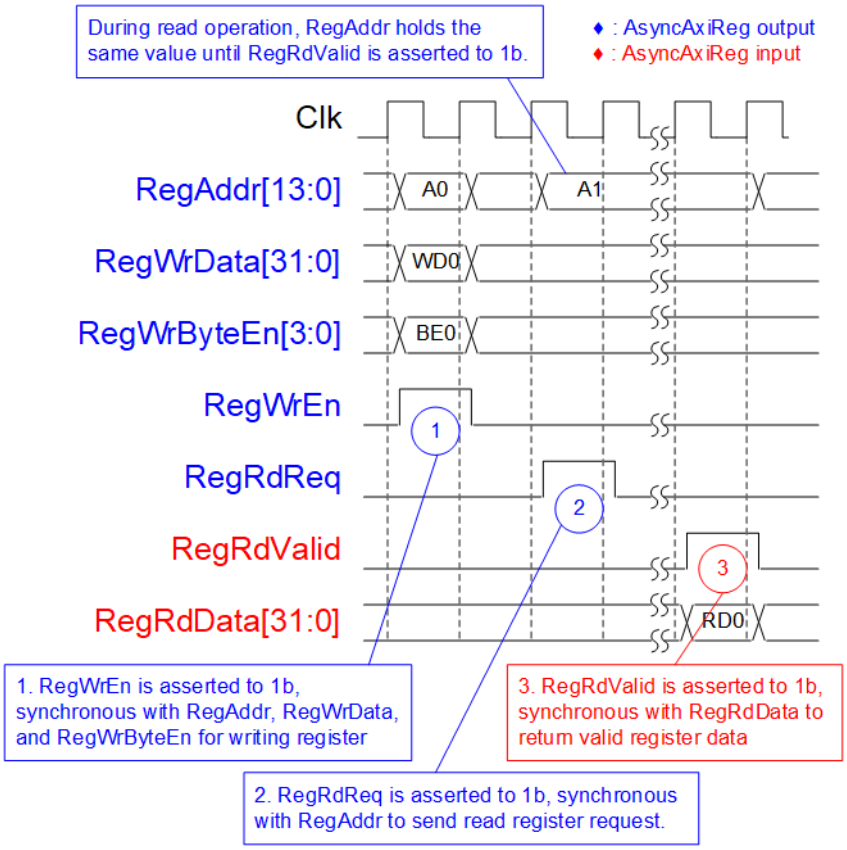

Figure 2-12 Register interface timing diagram

1) The timing for writing to a register is similar to that of a single-port RAM. The RegWrEn signal is set to 1b, along with valid values for RegAddr (Register address in 32-bit units), RegWrData (data to be written to the register), and RegWrByteEn (write byte enable). The RegWrByten signal consists of four bits, each corresponding to a byte within the 32-bit RegWrData value. For example, bits[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 a register, AsyncAxiReg sets the RegRdReq signal to 1b with a valid value for RegAddr. Once the read request is received, the 32-bit data is returned. The slave detects the RegRdReq signal to initiate the read transaction. The address value (RegAddr) remains constant 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 via the RegRdData bus by setting the RegRdValid signal to 1b. Afterward, AsyncAxiReg forwards the read value to the LAxir* interface.

2.3.2 UserReg

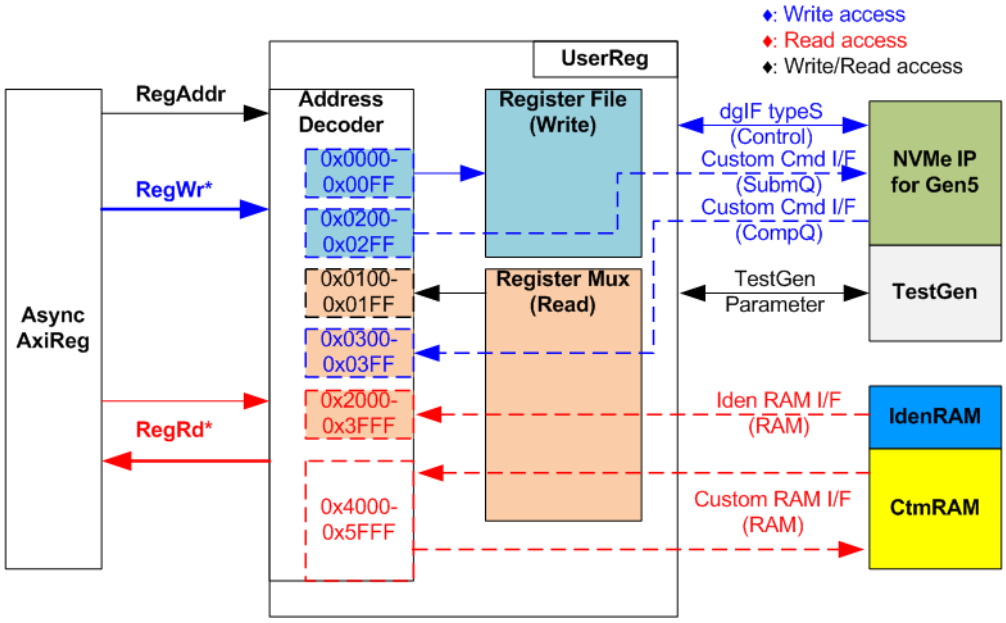

Figure 2-13 UserReg Interface

The UserReg module consists of three components: an Address decoder, a Register File, and a Register Multiplexer (Mux). The Address decoder decodes the address requested by AsyncAxiReg and selects the active register for either write or read transactions. The address range in UserReg is divided into six areas, as illustrated in Figure 2-13.

1) 0x0000 – 0x00FF: Mapped for setting commands and parameters for NVMe-IP and TestGen. This area is write-access only.
2) 0x0200 – 0x02FF: Mapped for setting parameters for the Custom command interface of NVMe-IP. This area is write-access only.
3) 0x0100 – 0x01FF: Mapped for reading status signals of NVMe-IP and TestGen. This area is read-access only.
4) 0x0300 – 0x03FF: Mapped for reading the status of Custom command interface (NVMe-IP). This area is read-access only.
5) 0x2000 – 0x3FFF: Mapped for reading data from IdenRAM. This area is read-access only.
6) 0x4000 – 0x5FFF: Mapped for writing or reading data using the Custom command RAM interface. Both write-access and read access are allowed, though in the demo, only read-access is demonstrated through the SMART command.

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

To read the register, the address is passed through multi-level multiplexers (mux) to select the data to be returned to the CPU. The lower bits of RegAddr are used within the submodules to select the active data, while the upper bits are used in UserReg to choose the returned data from the appropriate submodule. Consequently, the latency time for reading data is two clock cycles, and the RegRdValid is created by RegRdReq, with two D Flip-flops asserted.

More details regarding the address mapping within the UserReg module can be found in Table 2-1.

Table 2-1 Register Map

Address Rd/Wr	Register Name (Label in the “nvmeipg5test.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 bits[31:0] of the start address in 512-byte units (UserAddr[31:0] of dgIF typeS)
BA+0x0004	User Address (High) Reg (USRADRH_INTREG)	[15:0]: Input to be bits[47:32] of start address in 512-byte units (UserAddr[47:32] of dgIF typeS)
BA+0x0008	User Length (Low) Reg (USRLENL_INTREG)	[31:0]: Input to be bits[31:0] of transfer length in 512-byte units (UserLen[31:0] of dgIF typeS)
BA+0x000C	User Length (High) Reg (USRLENH_INTREG)	[15:0]: Input to be bits[47:32] of transfer length in 512-byte units (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 Writing to this register triggers the command request 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]: Two lower bits to show PCIe link speed of PCIe hard IP. Two upper bits are bit[17:16]. (0000b: Not linkup, 0001b: PCIe Gen1, 0010b: PCIe Gen2, 0011b: PCIe Gen3, 0111b: PCIe Gen4, 1111b: PCIe Gen5) [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 PCIe hard IP datasheet. [17:16]: Two upper bits to show PCIe link speed of PCIe hard IP. Two lower bits are bit[3:2].
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+0x0120) – (BA+0x012F)	NVMe IP Test pin DW0-3 Reg (NVMTESTPIN0-3_INTREG)	[31:0]: Mapped to TestPin[127:0] of NVMe-IP 0x0120: Bit[31:0], 0x0124: Bit[63:32], 0x0128: Bit[95:64], 0x012C: Bit[127:96]

Address Rd/Wr	Register Name (Label in the “nvmeipg5test.c”)	Description
0x0100 – 0x01FF: Status signals of NVMe-IP and TestGen (Read access only)		
BA+0x0170	Data Failure Address(Low) Reg (RDFAILNOL_INTREG)	[31:0]: Bits[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]: Bits[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]: Bits[31:0] of the current test data size in the TestGen module
BA+0x017C	Current test byte (High) Reg (CURTESTSIZEH_INTREG)	[24:0]: Bits[56:32] of the current test data size in the TestGen module
(BA+0x0180) – (BA+0x01BF)	Expected value Word0-15 Reg (EXPPATW0-W15_INTREG)	512-bit expected data at the 1st failure data during a Read command 0x0180: Bit[31:0], 0x0184: Bit[63:32],…, 0x01BC: Bit[511:480]
(BA+0x01C0) – (BA+0x01FF)	Read value Word0-15 Reg (RDPATW0-W15_INTREG)	512-bit actual read data at the 1st failure data during a Read command 0x01C0: Bit[31:0], 0x01C4: Bit[63:32],…, 0x01FC: Bit[511:480]
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 for 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 the 8KB CtmRAM interface for storing 512-byte data output from the SMART Command.

3 CPU Firmware

3.1 Test firmware (nvmeipg5test.c)

Upon system startup, the CPU follows these steps to complete the initialization process.

1) Initialize the UART and Timer settings.
2) Wait for the PCIe connection to become active by checking if PCIESTS_INTREG[0]=1b.
3) Wait for the NVMe-IP to complete its initialization process by monitoring USRSTS_INTREG[0]=0b. If an error is encountered, the process will stop and display an error message.
4) Display the PCIe link status, including the number of lanes and the speed, by reading PCIESTS_INTREG[17:2] value.
5) Display the main menu, providing options to execute seven NVMe-IP commands: Identify, Write, Read, SMART, Flush, Secure Erase, and Shutdown.

Details for each command sequence in the CPU firmware are described in the following sections.

3.1.1 Identify Command

When the Identify command is selected, the firmware executes the following sequence.

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 monitors USRSTS_INTREG[1:0] to determine whether the operation completes or an error occurs.
   - Bit[0] is de-asserted to 0b when the command is completed. The Identify command data returned by NVMe-IP is stored in IdenRAM.
   - Bit[1] is asserted to 1b, indicating an error. In this case, the CPU displays an error message on the console with details decoded from USRERRTYPE_INTREG[31:0]. The process will then stop.
3) Once the busy flag (USRSTS_INTREG[0]) is de-asserted to 0b, the CPU displays the SSD capacity and LBA unit size, decoded from LBASIZEL/H_INTREG. Additional information, such as the SSD model, can be retrieved from IdenRAM (IDENCTRL_CHARREG).

3.1.2 Write/Read Command

The firmware sequence for Write or Read command is as follows.

1) The CPU receives the start address, transfer length, and test pattern from the Serial console. If any inputs are invalid, the operation will be cancelled.
   Note: If the LBA unit size is 4 KB, the start address and transfer length must align to 8.
2) Once all inputs are validated, the values are written to USRADRL/H_INTREG, USRLENL/H_INTREG, and PATTSEL_INTREG.
3) To execute a Write command, set bits[2:0] of USRCMD_INTREG to 010b, or for a Read command, set it to 011b. 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 to 1b, indicating an error. An error message is displayed on the console with details from USRERRTYPE_INTREG[31:0], and the process will then stop.
   - Bit[2] is asserted when data verification fails. The verification error message is displayed on the console, but the CPU will continue running until the operation is completed.

   While the command is running, the current transfer size is read from CURTESTSIZEL/H_INTREG and displayed every second.

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

3.1.3 SMART Command

The firmware sequence for the f SMART command is as follows.

1) The CPU sets the 16-Dword of the Submission Queue entry (CTMSUBMQ_STRUCT) to the SMART command value.
2) Set bits[2:0] of USRCMD_INTREG 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 SMART command data returned by NVMe-IP will be stored in CtmRAM.
   - Bit[1] is asserted to 1b, indicating an error. An error message is displayed on the console with details 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 retrieve and 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 details on the SMART log, refer to the NVM Express Specification.
   https://nvmexpress.org/specifications/

3.1.4 Flush Command

The firmware sequence for the Flush command is as follows.

1) The 16-Dword of the Submission Queue entry (CTMSUBMQ_STRUCT) is configured with 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. An error message is displayed on the console with details from USRERRTYPE_INTREG[31:0]. The process will then stop.

3.1.5 Secure Erase Command

The firmware sequence for the Secure Erase command is as follows.

1) The 16-Dword of the Submission Queue entry (CTMSUBMQ_STRUCT) is configured with the Secure Erase command value.
2) Set NVMTIMEOUT_INTREG to 0 to disable the timer and prevent a timeout error.
3) Set bits[2:0] of USRCMD_INTREG 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 finishes. The CPU proceeds to the next step.
   - Bit[1] is asserted to 1b, indicating an error. An error message is displayed on the console with details from USRERRTYPE_INTREG[31:0]. The process will then stop.

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

3.1.6 Shutdown Command

The firmware sequence for the Shutdown command 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 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 completed. The CPU proceeds to the next step.
   - Bit[1] is asserted to 1b, indicating an error. An error message is displayed on the console with details from USRERRTYPE_INTREG[31:0]. The process will then stop.

3) After the Shutdown command completes, both the SSD and NVMe-IP will become inactive, and the CPU will no longer accept 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

void error_handler(void)	
Parameters	None
Return value	None
Description	This function is invoked when the system encounters an error. By default, it waits indefinitely, halting all further operations. Users can customize this function to implement specific error-handling procedures according to their requirements.

int exec_ctm(unsigned int user_cmd)	
Parameters	user_cmd: Command type (4-SMART/Secure Erase command, 6-Flush command)
Return value	STATUS_SUCCESS: Operation was successful. STATUS_ERROR: An error occurred.
Description	Execute the SMART command as outlined in section 3.1.3 (SMART Command) or the Flush command as outlined in section 3.1.4 (Flush Command).

unsigned long long get_cursize(void)	
Parameters	None
Return value	The read value of CURTESTSIZEL/H_INTREG
Description	Read the value from ‘CURTESTSIZEL/H_INTREG’ and converts it to byte units before returning as the result of the function.

int get_param(userin_struct* userin)	
Parameters	userin: Structure that holds three inputs from the user: start address, total length in 512-byte units, and test pattern
Return value	STATUS_SUCCESS: Operation was successful. STATUS_INVALIDINPUT: Invalid input was received.
Description	Receive input parameters from the user and verify them. If the input is invalid, the function returns ‘STATUS_INVALIDINPUT’. Otherwise, the inputs are updated in the userin structure.

int iden_dev(void)	
Parameters	None
Return value	STATUS_SUCCESS: Operation was successful. STATUS_ERROR: An error occurred.
Description	Execute the Identify command as outlined in section 3.1.1 (Identify Command).

int setctm_erase(void)	
Parameters	None
Return value	STATUS_SUCCESS: Operation was successful. STATUS_ERROR: An error occurred.
Description	Set the Secure Erase command in ‘CTMSUBMQ_STRUCT’ and call ‘exec_ctm’ function to execute the Secure Erase command.

int setctm_flush(void)	
Parameters	None
Return value	STATUS_SUCCESS: Operation was successful. STATUS_ERROR: An error occurred.
Description	Set the Flush command in ‘CTMSUBMQ_STRUCT’ and call ‘exec_ctm’ function to execute the Flush command.

int setctm_smart(void)	
Parameters	None
Return value	STATUS_SUCCESS: Operation was successful. STATUS_ERROR: An error occurred.
Description	Set the SMART command in ‘CTMSUBMQ_STRUCT’ and call ‘exec_ctm’ function to execute the SMART command, and then decode and display the SMART information on the console

void show_error(void)	
Parameters	None
Return value	None
Description	Read ‘USRERRTYPE_INTREG’, decode the error flags, and display the corresponding error message. Additionally, 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 values from two consecutive reads are stable. After that, the read value is displayed on the console. Additionally, debug signals are read from ‘NVMTESTPIN0-3_INTREG’.

void show_result(void)	
Parameters	None
Return value	None
Description	Display the total transfer size by calling ‘get_cursize’ and ‘show_size’ functions. The total time usage is calculated from global parameters (‘timer_val’ and ‘timer_upper_val’) and displayed in usec, msec, or sec. Finally, the transfer performance is calculated and displayed in MB/s.

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 ‘size_input’ value in MB or GB unit.

void show_smart_hex16byte(volatile unsigned char *char_ptr)	
Parameters	*char_ptr: Pointer to a 16-byte SMART data block
Return value	None
Description	Display 16-byte SMART data as a hexadecimal value on the console.

void show_smart_int8byte(volatile unsigned char *char_ptr)	
Parameters	*char_ptr: Pointer to an 8-byte SMART data block
Return value	None
Description	Displays the 8-byte SMART data in decimal units if the input value is less than 4 billion (32-bit). 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 to an 8-byte SMART data block
Return value	None
Description	Display the 8-byte SMART data in GB or TB unit. If the input value excceds 500 PB, an overflow message is displayed.

void show_vererr(void)	
Parameters	None
Return value	None
Description	Read the error byte address from ‘RDFAILNOL/H_INTREG’, the expected value from ‘EXPPATW0-15_INTREG’, and the read value from ‘RDPATW0-15_INTREG’. These details are displayed on the console to provide information on verification errors.

int shutdown_dev(void)	
Parameters	None
Return value	STATUS_SUCCESS: Operation was successful. STATUS_ERROR: An error occurred.
Description	Execute the Shutdown command as outlined in section 3.1.6 (Shutdown Command).

int wrrd_dev(unsigned int user_cmd)	
Parameters	user_cmd: Command type (2-Write command, 3-Read command)
Return value	STATUS_SUCCESS: Operation was successful. STATUS_INVALIDINPUT: Invalid input was received. STATUS_ERROR: An error occurred.
Description	Execute either the Write command or Read command as outlined in section 3.1.2 (Write/Read Command). This function calls ‘show_result’ function to compute and display the transfer performance for the Write or Read operation.

4 Example Test Result

An example test result obtained from running the demo system with a 2 TB CFD Gaming PG5NFZ SSD is shown in Figure 4-1. The system's performance was measured using the Write and Read commands, with the test data pattern set to LFSR and a transfer size of 128 GB.

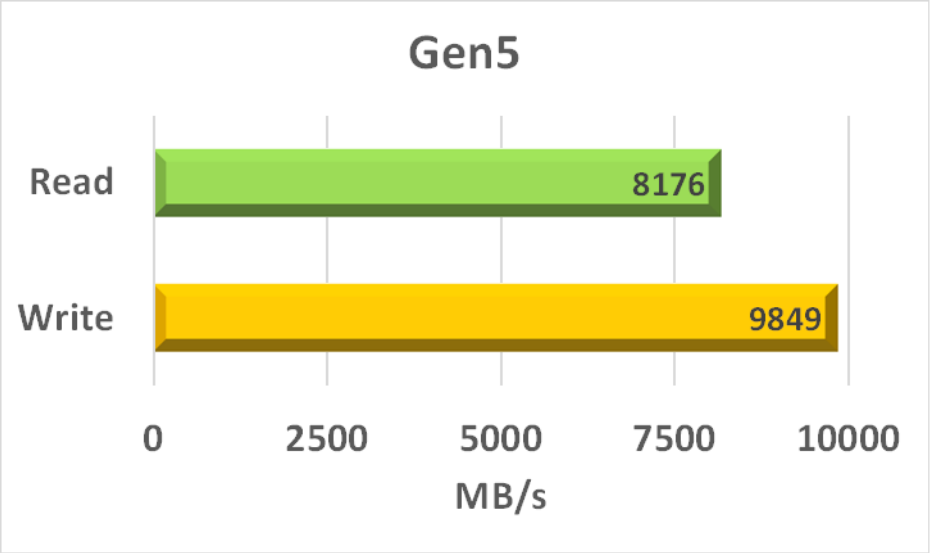

Figure 4-1 Test Performance of NVMe-IP for Gen5 demo using 2TB CFD Gaming PG5NFZ SSD

Utilizing the VHK158 board with PCIe Gen5, the system demonstrates impressive performance. Write performance is approximately 9,800 MB/sec, while read performance is approximately 8,100 MB/sec.

5 Revision History

Revision	Date (D/M/Y)	Description
1.00	15-Aug-24	Initial Release