

# NVMe-IP for AG5 Reference Design Manual

1	Overview .....	2
2	Hardware .....	3
2.1	TestGen .....	4
2.2	NVMe .....	7
2.2.1	NVMe-IP .....	7
2.2.2	GTS AXI Stream PCIe IP .....	7
2.2.3	Two-port RAM .....	8
2.3	CPU and Peripherals .....	9
2.3.1	AsyncAviReg .....	10
2.3.2	UserReg .....	12
3	CPU Firmware .....	15
3.1	Test firmware (nvmeiptest.c).....	15
3.1.1	Identify Command .....	15
3.1.2	Write/Read Command.....	16
3.1.3	SMART Command .....	16
3.1.4	Flush Command .....	17
3.1.5	Secure Erase Command.....	17
3.1.6	Shutdown Command.....	17
3.2	Function List in Test Firmware .....	18
4	Example Test Result .....	21
5	Revision History .....	22

# NVMe-IP for AG5 Reference Design Manual

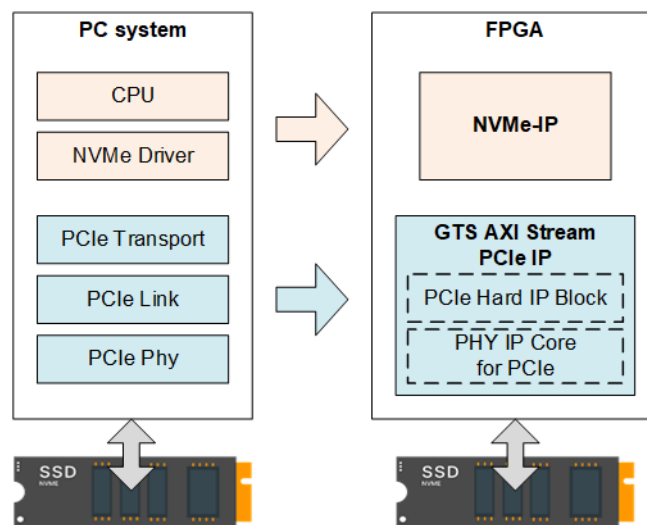
Rev1.00 30-May-2025

## 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 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. For a detailed comparison between the AHCI and NVMe protocols, refer 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](https://sata-io.org/system/files/member-downloads/NVMe%20and%20AHCI_%20long_.pdf)

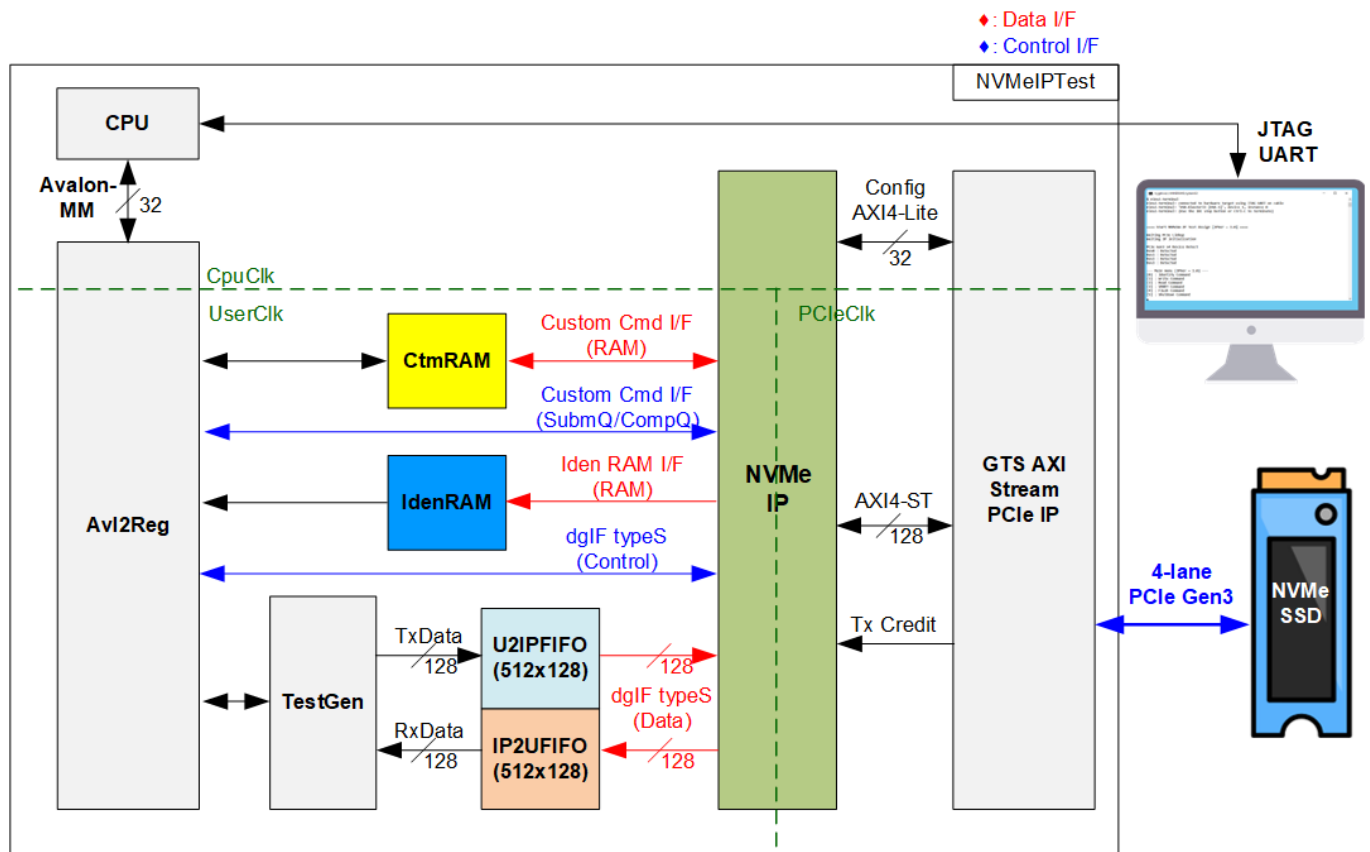


**Figure 1 NVMe Protocol Layer**

To access an NVMe SSD, a typical system employs an NVMe driver running on a processor, as shown on the left side of Figure 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 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 Avl2Reg).

The TestGen module interfaces with the NVMe-IP 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 and the PCIe hard IP (GTS AXI Stream PCIe IP), providing direct access to an NVMe SSD without requiring a PCIe switch. Command requests and parameters for each command, which are inputs to the NVMe-IP, are controlled by the CPU through the Avl2Reg module. Additionally, the data interface for both Custom and Identify commands connects to RAMs accessible by the CPU.

The CPU is connected to the Avl2Reg 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 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.

Figure 2 illustrates the three clock domains used in the NVMe reference design: CpuClk, UserClk, and PCIeClk.

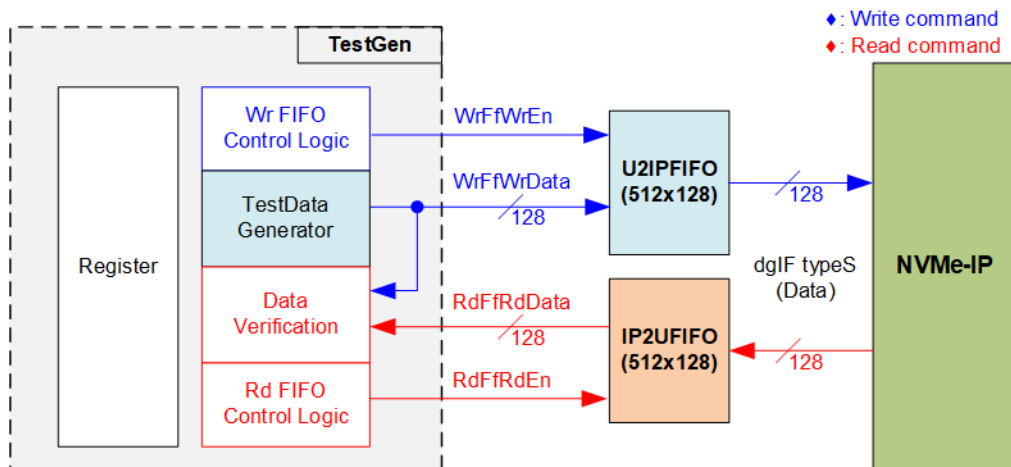
- CpuClk is the clock domain for the CPU, its peripherals, and the AXI4-Lite configuration interface of the PCIe. It must be a stable, independent clock that is not depend on other hardware components.
- UserClk is used for the operation of the NVMe-IP, RAM, and TestGen. As specified in the NVMe-IP datasheet, the UserCLk frequency must be equal to or greater than PCIeClk. In this reference design, UserClk is set to 310MHz for PCIe Gen3 operation.

*Note: While it is possible to use the same clock for UserClk and PCIeClk, this demo uses a separate clock to accommodate base systems where user logic may operate in a different clock domain.*

- PCIeClk is generated by the PCIe hard IP and is synchronized with the 128-bit AXI Stream interface. It operates at 300 MHz for a 4-lane PCIe Gen3 configuration.

Further hardware details are provided in the subsequent sections.

## 2.1 TestGen

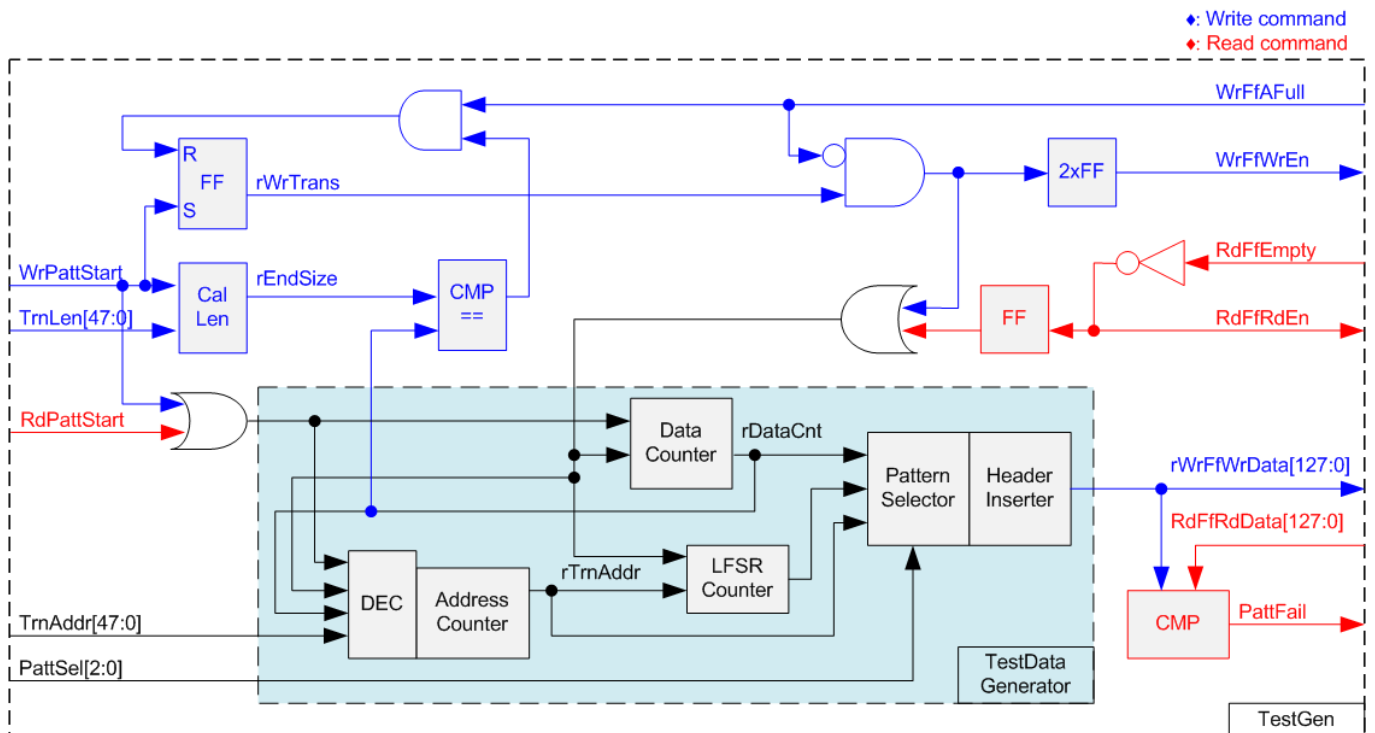


**Figure 3 TestGen Interface**

The TestGen module handles the data interface of NVMe-IP, facilitating data transfer for both Write and Read commands. During a Write command, TestGen generates a 128-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 4.



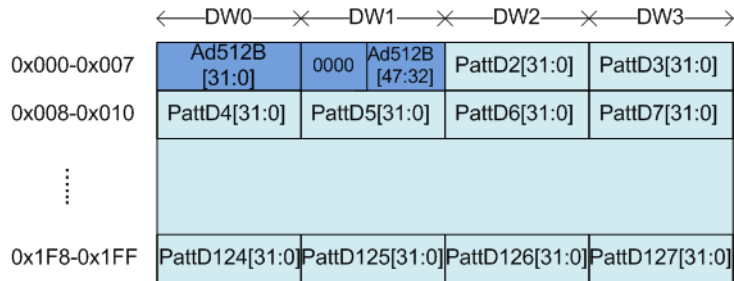
**Figure 4 TestGen Hardware**

In Figure 4, two key aspects of the system are depicted. The first part (upper section) illustrates the control of data flow, while the second part (lower section) details the generation of test data for use with the FIFO interface.

In the upper section of Figure 4, 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 4 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 5.

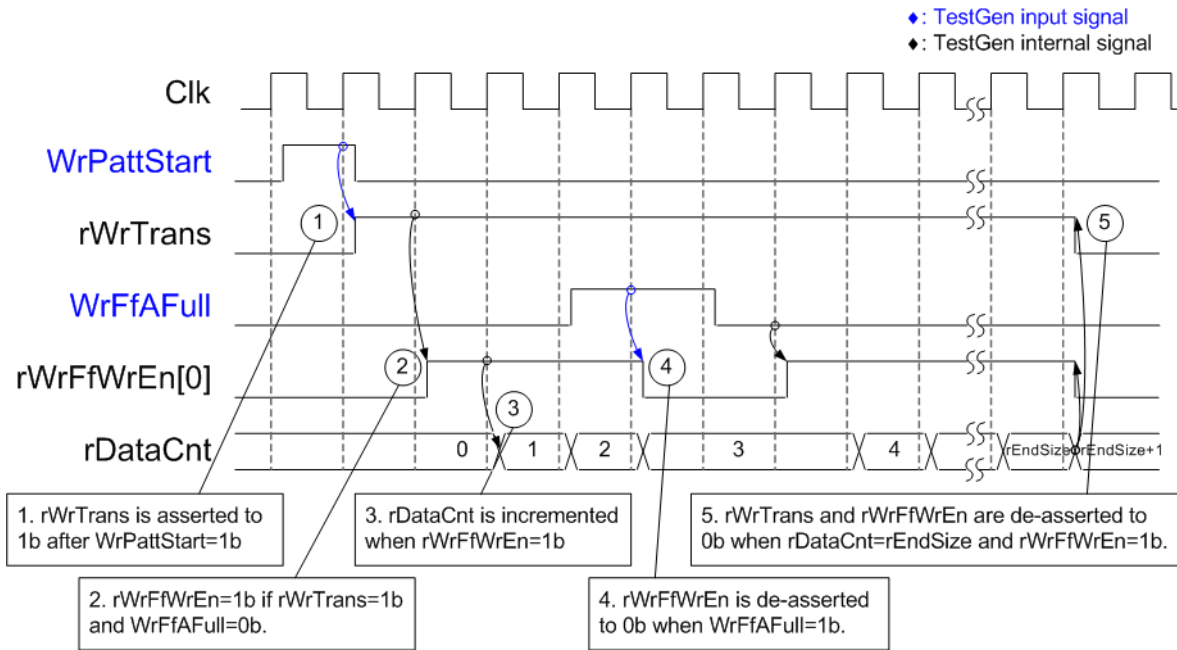


**Figure 5 Test Pattern Format for Increment/Decrement/LFSR Patterns in Each 512-Byte Data Block**

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 (Dword#2 – Dword#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.

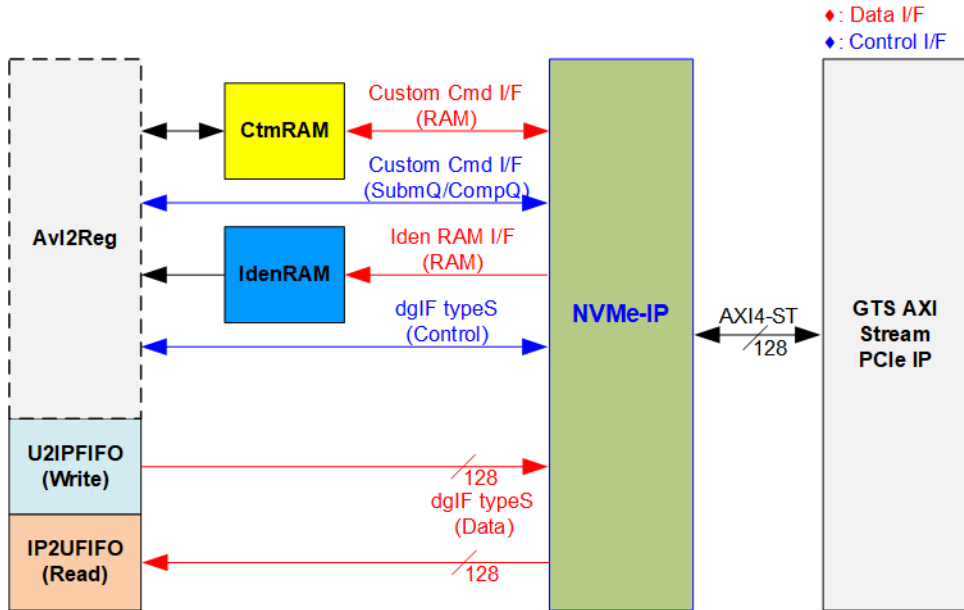


**Figure 6 Timing Diagram of Write Operation in TestGen**

- 1) The write operation is initiated by setting WrPattStart signal to 1b for a single clock cycle. Subsequently, the rWrTrans signal is asserted, enabling the control logic for asserting write enable signal to the 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 data transferred during the write operation.
- 4) If FIFO is almost full (WrFfAFull=1b), the write process is temporarily paused by de-asserting the rWrFfWrEn to 0b.
- 5) The write operation is completed when the total data count matches the set value. At this point, both rWrTrans and rWrFfWrEn are de-asserted to 0b.

For read transfer, the read enable for the FIFO is controlled by the empty flag of FIFO. Unlike the write enable, the read enable signal is not stopped by the total data count and is not started by start flag. When the read enable is asserted to 1b, both the data counter and the address counter are increased for counting the total transferred data and generating the header of expected value, respectively.

## 2.2 NVMe



**Figure 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 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.

Besides, 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-way data transfer when NVMe-IP sends SMART data to Avl2Reg.

### 2.2.1 NVMe-IP

The NVMe-IP implements the NVMe protocol on the host side, enabling direct access to an NVMe 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/NVMe-IP-datasheet-ag5/>

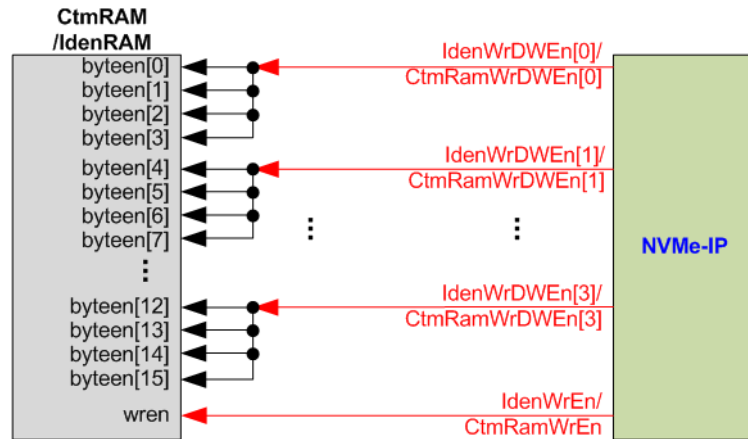
### 2.2.2 GTS AXI Stream PCIe IP

The GTS AXI Stream PCIe IP is a hard IP block integrated in Altera FPGAs, responsible for implementing the Physical, Data Link, and Transaction Layers of the PCIe specification. More information is available in the following document:

<https://www.intel.com/content/www/us/en/docs/programmable/813754/>

### 2.2.3 Two-port RAM

Two of two-port RAMs, CtmRAM and IdenRAM, store the returned data from Identify and SMART commands, respectively. IdenRAM is simple dual-port RAM with one read port and one write port and has a data size of 8 Kbytes to store the 8 Kbyte output from the Identify command. The data bus size for NVMe-IP and Avl2Reg differ, with NVMe-IP having a 128-bit size and Avl2Reg having a 32-bit size. As a result, IdenRAM is 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 the IP catalog of Quartus enables write byte enable, so each bit of the double word enable is extended to a 4-bit write byte enable, as shown in Figure 8.



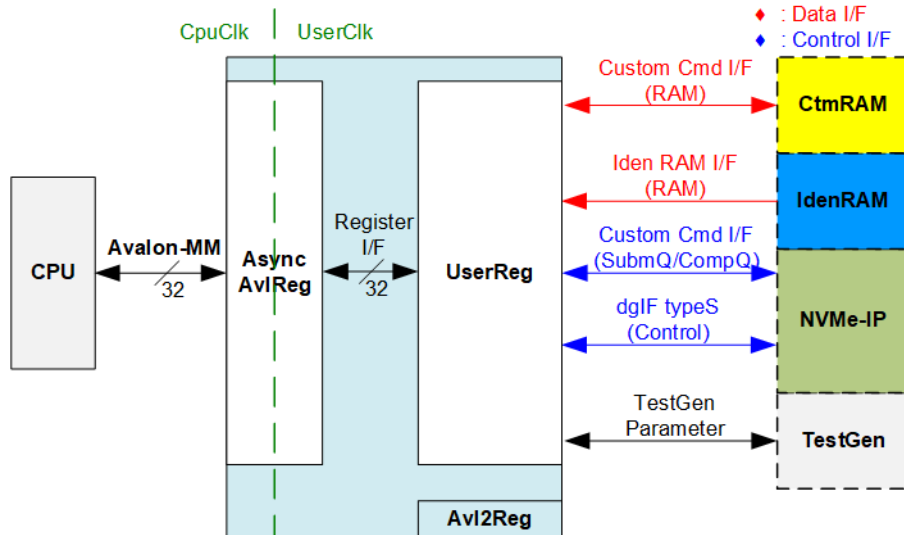
**Figure 8 Byte Write Enable Conversion Logic**

Each bit of WrDWEEn is extended to be 4-bit of IdenWrEn, so bit[0], [1], [2], and [3] are then used to drive bits[3:0], [7:4], [11:8], and [15:12] of IdenWrEn, respectively.

On the other hand, CtmRAM is implemented as a two-Port RAM with two read ports and two write ports, and with byte write enable. The connection from the double-word enable of NVMe-IP to byte enable of CtmRAM is similar to that of IdenRAM. The two-port RAM is utilized to support additional features when the customized Custom command requires data input. For supporting SMART command, a simple dual-port RAM is sufficient, even though the data size returned from the SMART command is 512 bytes. However, CtmRAM is implemented with an 8KB RAM for the customized Custom command.

### 2.3 CPU and Peripherals

The CPU system uses a 32-bit Avalon-MM bus as the interface to access peripherals such as the Timer and JTAG UART. The system also integrates an additional peripheral to access the 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 Avalon-MM bus standard. Avl2Reg module, as shown in Figure 9, is designed to connect the CPU system via the Avalon-MM interface, in compliance with the standard.



**Figure 9 CPU and Peripherals Hardware**

Avl2Reg consists of AsyncAvlReg and UserReg. AsyncAvlReg converts Avalon-MM signals into a simple Register interface with a 32-bit data bus size, similar to the Avalon-MM 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 AsyncAvlReg and UserReg are explained below.

### 2.3.1 AsyncAvlReg

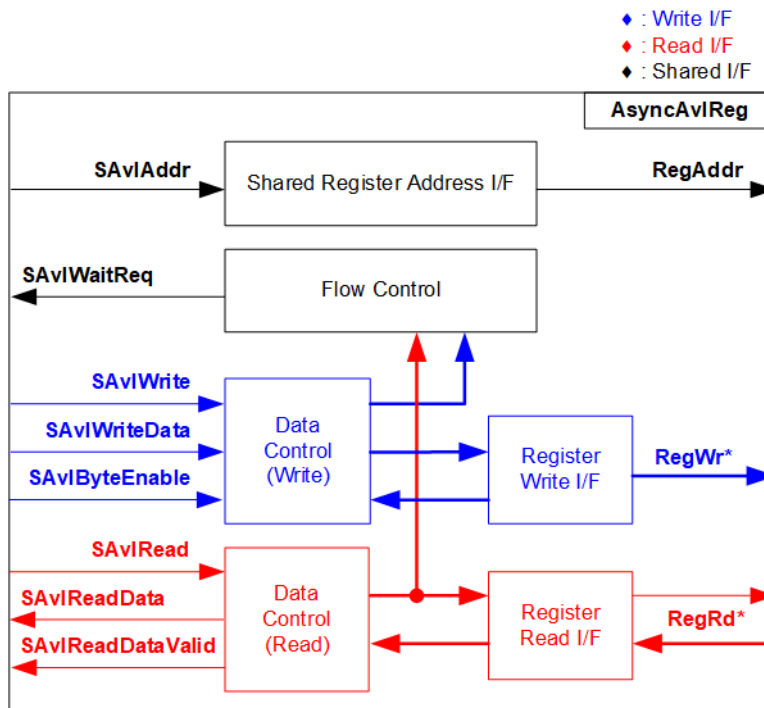


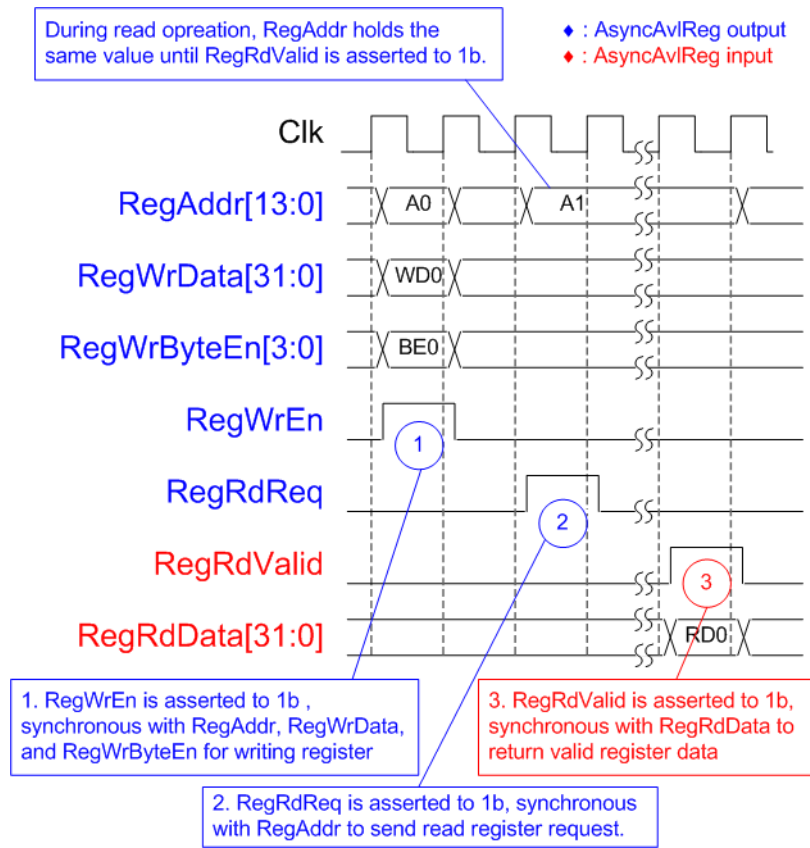
Figure 10 AsyncAvlReg Interface

The Avalon-MM bus interface signals are divided into three categories: Write channel (blue), Read channel (red), and Shared control channel (black). More details about the Avalon-MM interface specification can be found in the following document.

[https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl\\_avalon\\_spec.pdf](https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf)

According to the Avalon-MM specification, only one command (write or read) can be executed at a time. AsyncAvlReg’s logic is organized into three groups: Write control logic, Read control logic, and Flow control logic. The flow control logic asserts SAvIWaitReq to hold off subsequent requests from the Avalon-MM interface until the current request completes. Write control and Write data signals of the Avalon-MM bus are latched and transferred to the Write register interface through clock domain crossing registers. Similarly, Read control signals are latched and transferred to be Read register interface. Afterward, the data returned from Register Read I/F is transferred back to Avalon-MM bus using clock domain crossing registers. The Address I/F of Avalon-MM is also latched and transferred to the Address register interface.

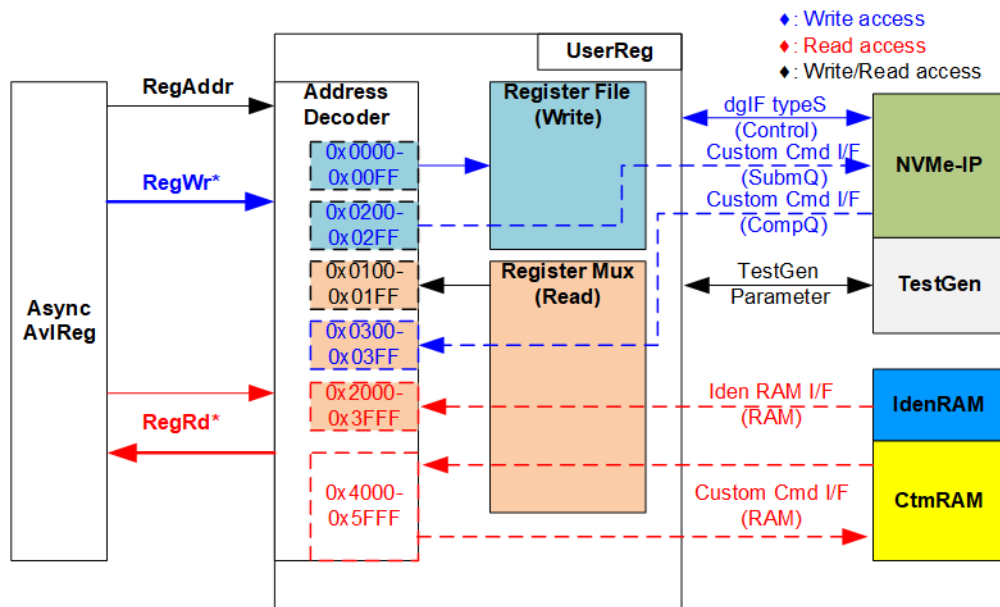
The Register interface is compatible with single-port RAM interface for write transactions. However, the read transaction has a slight modification from the RAM interface by adding RdReq and RdValid signals to manage read latency. Since the address of the Register interface is shared for write and read transactions, it cannot handle simultaneous write and read operations. The timing diagram of the Register interface is shown in Figure 11.



**Figure 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 (the Register address in 32-bit units), RegWrData (write data for the register), and RegWrByteEn (write byte enable). The byte enable is four bits wide, where each bit indicates the validity of a specific byte within RegWrData. For example, if RegWrByteEn[0], [1], [2], and [3] are set to 1b, then RegWrData[7:0], [15:8], [23:16], and [31:24] are valid, respectively.
- 2) To read from a register, AsyncAvlReg sets the RegRdReq signal to 1b, along with a valid value for RegAddr. After the read request is processed, the 32-bit data is returned. The slave detects the RegRdReq being asserted to start the read transaction. During the read operation, the address value (RegAddr) remains unchanged until RegRdValid is set to 1b. Once valid, the address is used to select the returned data through multiple layers of multiplexers.
- 3) The slave returns the read data on RegRdData bus by setting the RegRdValid signal to 1b. After that, AsyncAvlReg forwards the read value to the SAVIRead interface.

### 2.3.2 UserReg



**Figure 12 UserReg Interface**

The UserReg module comprises three components: an Address Decoder, a Register File, and a Register Mux. The Address Decoder interprets the address issued by AsyncAvlReg and selects the corresponding register for either write or read operations. The address space assigned to the UserReg module is divided into six distinct regions, as illustrated in Figure 12.

- 1) 0x0000 – 0x00FF: Mapped to set the command with the parameters of NVMe-IP and TestGen. This region is write-access only.
- 2) 0x0200 – 0x02FF: Mapped to set the parameters for Custom command interface of NVMe-IP. This region is write-access only.
- 3) 0x0100 – 0x01FF: Mapped to read the status signals of NVMe-IP and TestGen. This region is read-access only.
- 4) 0x0300 – 0x03FF: Mapped to read the status of Custom command interface (NVMe-IP). This region is read-access only.
- 5) 0x2000 – 0x3FFF: Mapped to read data from IdenRAM. This region is read-access only.
- 6) 0x4000 – 0x5FFF: Mapped to write and read access to the Custom command RAM interface. While the interface supports both access types, the demo uses read access only when executing the SMART command.

The Address decoder decodes the upper bits of RegAddr to select the appropriate hardware module (NVMe-IP, TestGen, IdenRAM, or CtmRAM). The Register File within the UserReg module operates on a 32-bit bus size, so the write byte enable (RegWrByteEn) is not used in the test system. The CPU interacts with the hardware registers using 32-bit aligned pointers.

For read operations, multi-level multiplexers (mux) are used to select the correct data to return to the CPU based on the input address. The lower bits of RegAddr are passed to the respective submodules to select specific register values, while the upper bits are used within UserReg to determine which submodule’s data is returned.

As a result, the read data latency is two clock cycles. The signal “RegRdValid” is generated by delaying “RegRdReq”, through two D Flip-flops, ensuring proper timing and synchronization. Additional details on the address mapping within the UserReg module are provided in Table 1.

**Table 1 Register Map**

Address	Register Name	Description
Rd/Wr	(Label in "nvmeiptest.c")	
<b>0x0000 – 0x00FF: Control Signals of NVMe-IP and TestGen (Write-Access Only)</b>		
BA+0x0000	User Address (Low) Reg (USRADRL_INTREG)	[31:0]: Input to be bits[31:0] of start address as 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 as 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 as 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 as 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
<b>BA+0x1000 – BA+0x10FF: Status Signals of NVMe-IP and TestGen (Read-Access Only)</b>		
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 fails (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) [6:4]: PCIe link width status from PCIe hard IP (001b: 1-lane, 010b: 2-lane, 100b: 4-lane) [13:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in Avalon-ST PCIe Hard IP 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
<b>0x0100 – 0x01FF: Status Signals of NVMe-IP and TestGen (Read-Access Only)</b>		
BA+0x0130 - BA+0x013F	Expected value Word0-3 Reg (EXPPATW0-W3_INTREG)	128-bit expected data at the 1 <sup>st</sup> failure data during 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)	128-bit actual read data at the 1 <sup>st</sup> failure data during 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]: Bits[31:0] of the byte address of the 1 <sup>st</sup> 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 1 <sup>st</sup> failure when executing a Read Command
BA+0x0178	Current test byte (Low) Reg (CURTESTSIZE_L_INTREG)	[31:0]: Bits[31:0] of the current test data size in TestGen module
BA+0x017C	Current test byte (High) Reg (CURTESTSIZE_H_INTREG)	[24:0]: Bits[56:32] of the current test data size in TestGen module
<b>Other Interfaces (Custom Command of NVMe-IP, IdenRAM, and Custom RAM)</b>		
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 the 8KB CtmRAM interface for storing 512-byte data output from the 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 JTAG 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[6:2]` value.
- 5) Display the main menu, providing options to execute seven NVMe-IP commands: Identify, Write, Read, SMART, Flush, Secure Erase, and Shutdown.

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

#### 3.1.1 Identify Command

When the user selects the Identify command, 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]`) then changes 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 be terminated.
- 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

When the Write/Read command is selected, the firmware follows this sequence.

- 1) The CPU receives the start address, transfer length, and test pattern from the console. If any inputs are invalid, the operation will be cancelled.
 

*Note: If 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 be terminated.
  - 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 CURTESTSIZE/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

When the SMART command is selected, the firmware follows this sequence.

- 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[2:0] to 100b to send the SMART command request to NVMe-IP. The busy flag (USRSTS\_INTREG[0]) then changes 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 be terminated.
- 4) Once 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

When the user selects the Flush command, the firmware follows this sequence

- 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[2:0] to 110b to send Flush command request to NVMe-IP. The busy flag of NVMe-IP (USRSTS\_INTREG[0]) then changes 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 be terminated.

### 3.1.5 Secure Erase Command

When the user selects the Secure Erase command, the firmware follows this sequence.

- 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 to prevent a 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]) changes 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 be terminated.
- 5) Once the command complete, the timer is re-enabled by setting NVMTIMEOUT\_INTREG to its default value to generate timeout error in NVMe-IP. Following this, the CPU returns to the main menu.

### 3.1.6 Shutdown Command

When the Shutdown command is selected, the firmware follows this sequence.

- 1) Set bits[2:0] of USRCMD\_INTREG[2:0] to 001b to send Shutdown command request to NVMe-IP. The busy flag of NVMe-IP (USRSTS\_INTREG[0]) then changes 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 be terminated.
- 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 resume testing, the user must power off and subsequently power on the system.

### 3.2 Function List in Test Firmware

This section describes the list of functions used to operate NVMe-IP.

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 SMART command as outlined in section 3.1.3 (SMART Command), execute Flush command as outlined in section 3.1.4 (Flush Command), or execute Secure Erase command as outline in section 3.1.5 (Secure Erase Command)

void get_cursize(unsigned long long* cursize)	
Parameters	cursize: A pointer to store read value
Return value	None
Description	Read current transfer size from CURTESTSIZE/H_INTREG and update the value to 'cursize' parameter.

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 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 Secure Erase command to 'CTMSUBMQ_STRUCT' and call 'exec_ctm' function to execute Secure Erase command.

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

int setctm_smart(void)	
Parameters	None
Return value	STATUS_SUCCESS: Operation was successful. STATUS_ERROR: An error occurred.
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 values from two consecutive reads are stable. After that, the read value is displayed on the console. Also, debug signals are read from 'NVMTESTPIN_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 of 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 of 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 of 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 exceeds 500 PB, an overflow message is displayed.

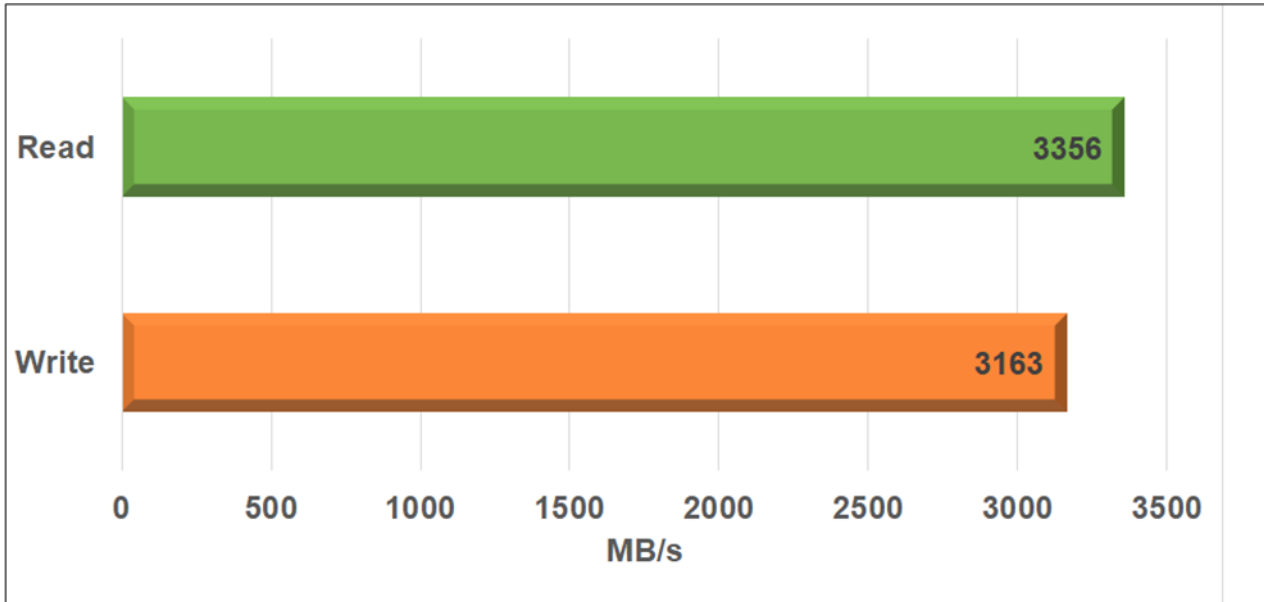
void show_vererr(void)	
Parameters	None
Return value	None
Description	Read the error byte address from 'RDFAILNOL/H_INTREG', the expected value from 'EXPPATW0-3_INTREG', and the read value from 'RDPATW0-3_INTREG'. These details are displayed on the console to provide information on verification errors.

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 wrdd_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 2 TB Addlink S95 SSD is shown in Figure 13. The system's performance was measured using the Write and Read commands, with the test data pattern set to all-zero and a transfer size of 128 GB.



**Figure 13 Test Performance of NVMe-IP Demo Using 2 TB Addlink S95 NVMe SSD**

Utilizing the Sulfur Agilex5 E-Series board with PCIe Gen3, the system achieves a remarkable write performance of approximately 3,163 MB/s and a read performance of around 3,356 MB/s.

## 5 Revision History

Revision	Date (D-M-Y)	Description
1.00	30-May-25	Initial version release