# *NVMe-IP for Gen5 reference design manual*

# NVMe-IP for Gen5 reference design manual

Rev1.0 23-May-23

## 1 NVMe

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 allows for improved 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 https://nvmexpress.org/compliance/.

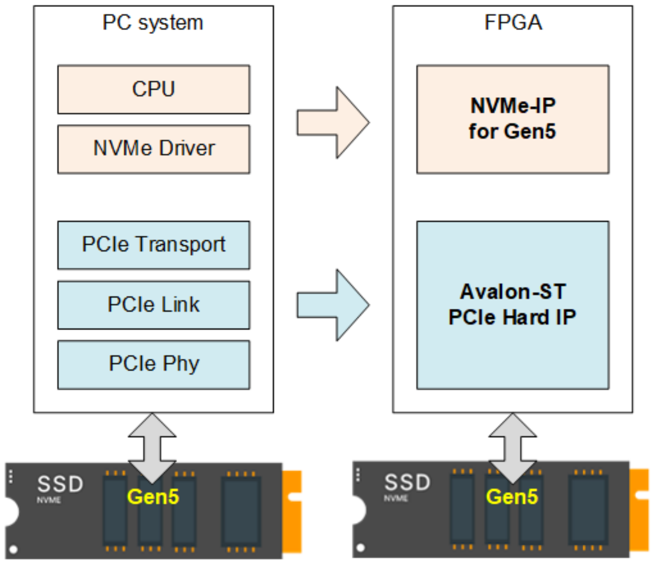

Figure 1-1 NVMe protocol layer

To access NVMe Gen5 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 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

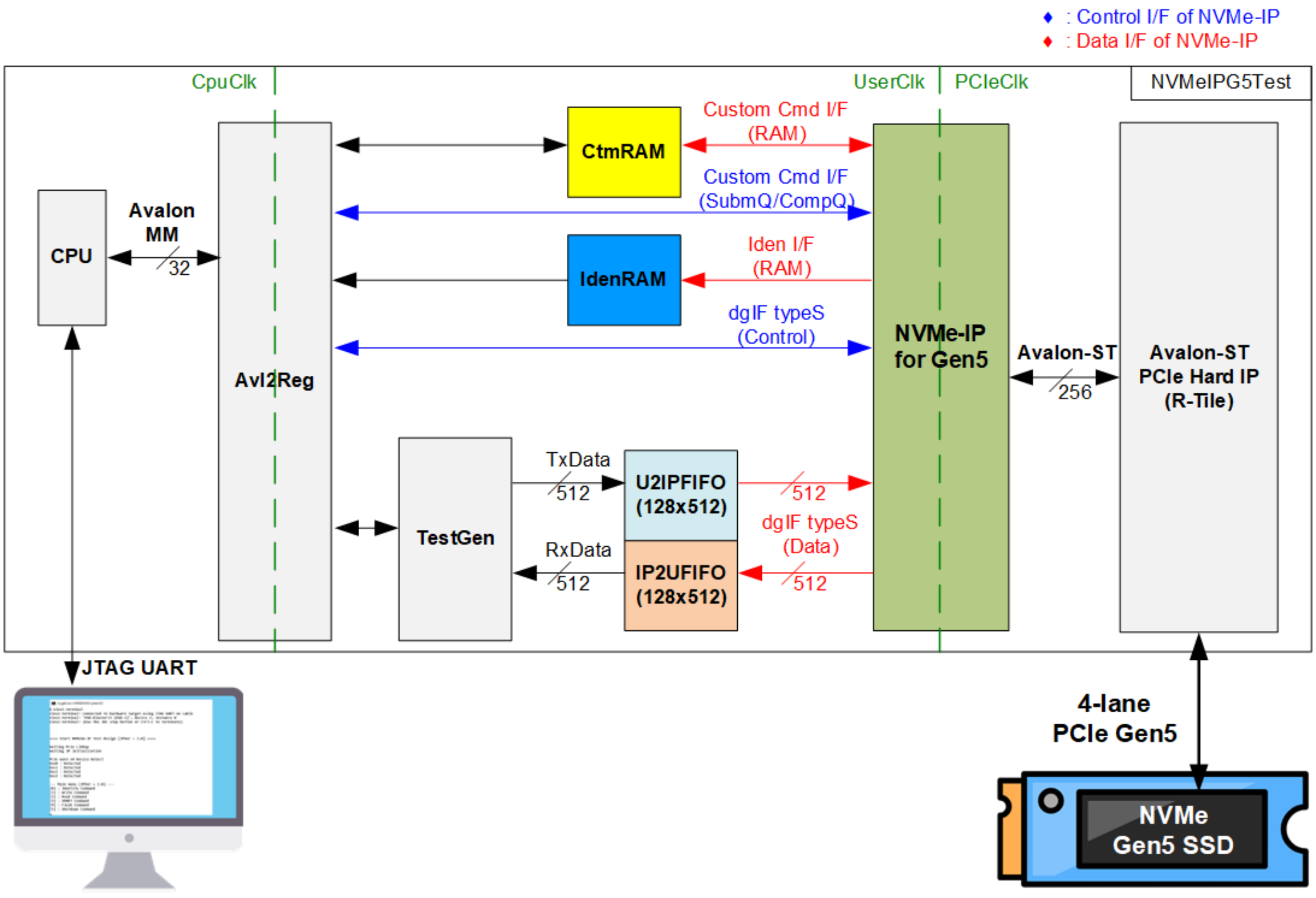

Figure 2-1 NVMe-IP for Gen5 demo hardware

The hardware modules in the test system are divided into three parts: test function (TestGen), NVMe function (CtmRAM, IdenRAM, U2IPFIFO, IP2UFIFO, NVMe-IP for Gen5, and PCIe block), and CPU system (CPU and Avl2Reg).

The TestGen connects to the user interface of NVMe-IP for Gen5 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 for Gen5 and the PCIe hard IP (R-Tile) for accessing an NVMe Gen5 SSD directly without PCIe switch. The command request and the parameters of each command, the inputs of NVMe-IP for Gen5, are controlled by the CPU through Avl2Reg 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 Avl2Reg 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 the console (JTAG UART). 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 for Gen5.

There are three clock domains displayed in Figure 2-1: CpuClk, UserClk, and PCIeClk. CpuClk is the clock domain for the CPU and its peripherals. It must be a stable clock that can function independently from other hardware. UserClk is the clock domain utilized 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 greater than or equal to half of the PCIeClk. The reference design utilizes 280 MHz for UserClk at PCIe Gen5 speed. PCIeClk is the clock output generated by PCIe hard IP, which is synchronized with the 256-bit Avalon stream. The frequency of PCIeClk is 500 MHz for 4-lane PCIe Gen5. Using a lower frequency of PCIeClk is possible, but it will limit the maximum bandwidth on the user interface of PCIe hard IP to be lower than PCIe Gen5 performance. Further hardware details are described below.

## 2.1 TestGen

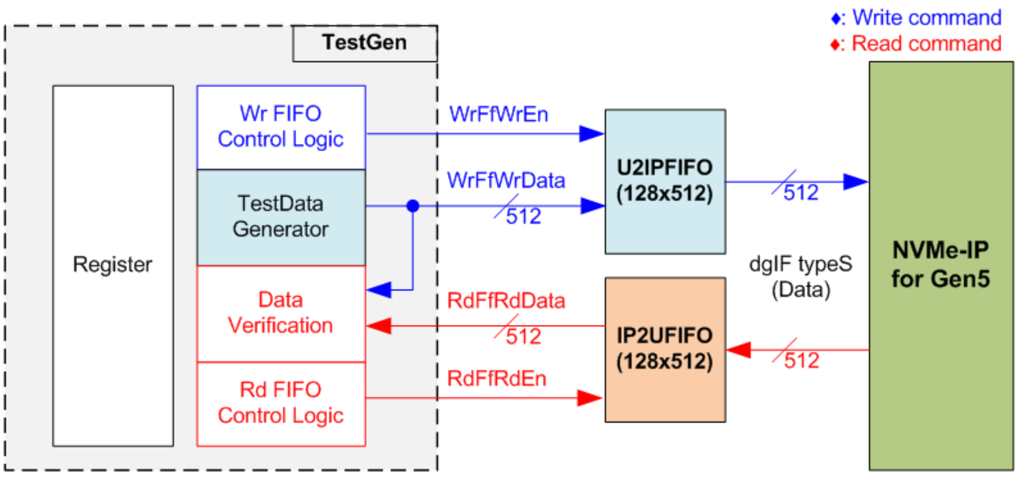

Figure 2-2 TestGen interface

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 512-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 ensures that both U2IPFIFO and IP2UFIFP are always ready to transfer data with NVMe-IP without delay, providing the best performance for writing and reading data with the SSD through NVMe-IP.

To provide a flexible test environment, the user can set some test parameters to control the TestGen module such as total transfer size, transfer direction, and test pattern selector via the console. These test parameters are stored in the Register block. The detailed hardware logic of TestGen is illustrated in Figure 2-3.

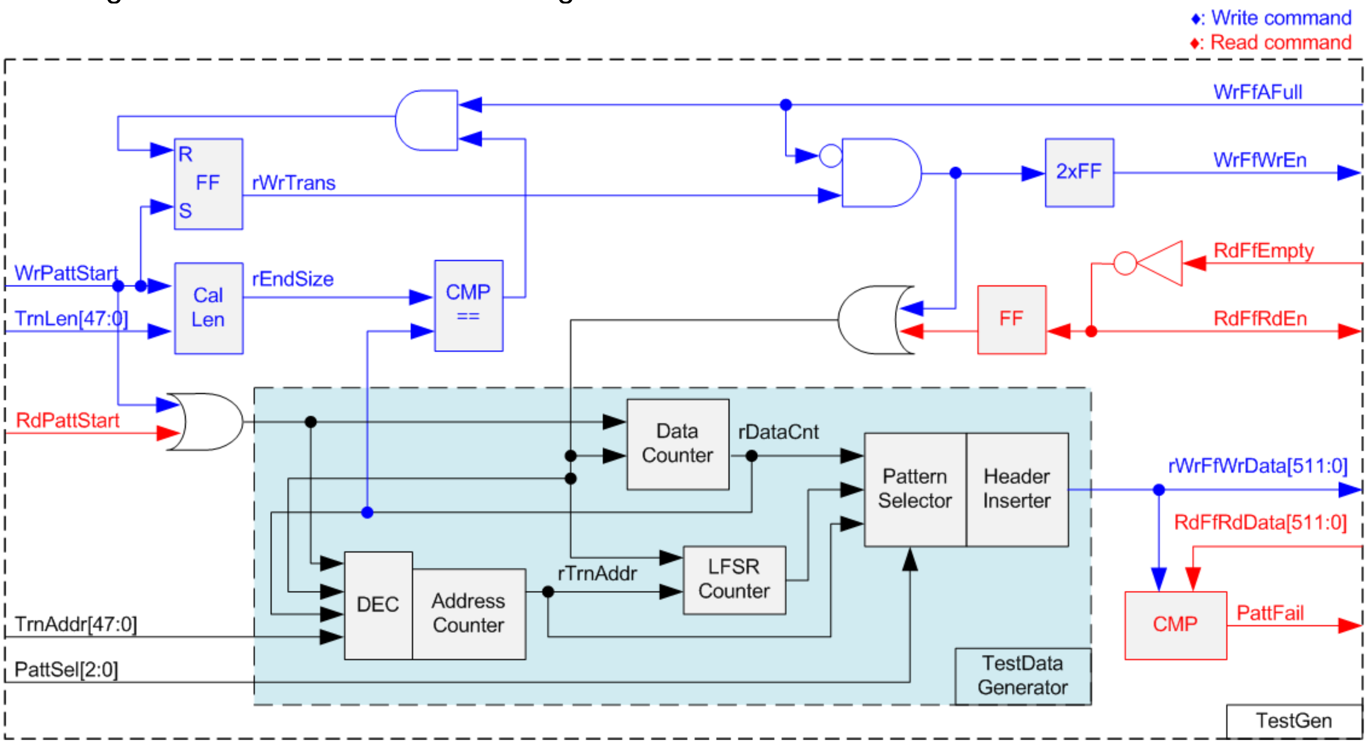

Figure 2-3 TestGen hardware

At the right side of Figure 2-3, the flow control signals of FIFO, WrFfAFull and RdFfEmpty, are used. During write operation, when FIFO is almost full (WrFfAFull=1b), WrFfWrEn is de-asserted to 0b to pause data sending to the FIFO. On the other hand, when there is data in the FIFO during read operation (RdFfEmpty=0b), the logic reads data from the FIFO to compare with the expected data by asserting RdFfRdEn to 1b.

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

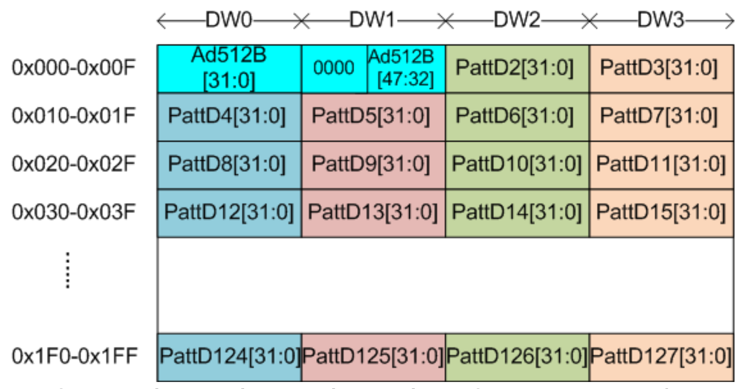

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

Each 512-byte data consists of a 64-bit header in Dword#0 and Dword#1, and the test data in remaining words of 512-byte data (Dword#2 – Dword#127). The header is created using the address in 512-byte unit (rTrnAddr) by the Address counter block. The initial value of the address counter is set by the user (TrnAddr), and the value is increased when finishing transferring 512-byte data. The remaining Dwords (DW#2 – DW#127) depends on the pattern selector, which may be 32-bit incremental data, 32-bit decremental data, or LFSR. The 32-bit incremental data is designed using Data counter, while the decremental data can be designed by connecting NOT logic to incremental data. The LFSR pattern is designed using the LFSR counter. The equation of LFSR is x^31 + x^21 + x + 1.

To generate 512-bit test data for LFSR pattern, the data is divided into four sets of 128-bit data, each using a different start value, as shown in Figure 2-5.

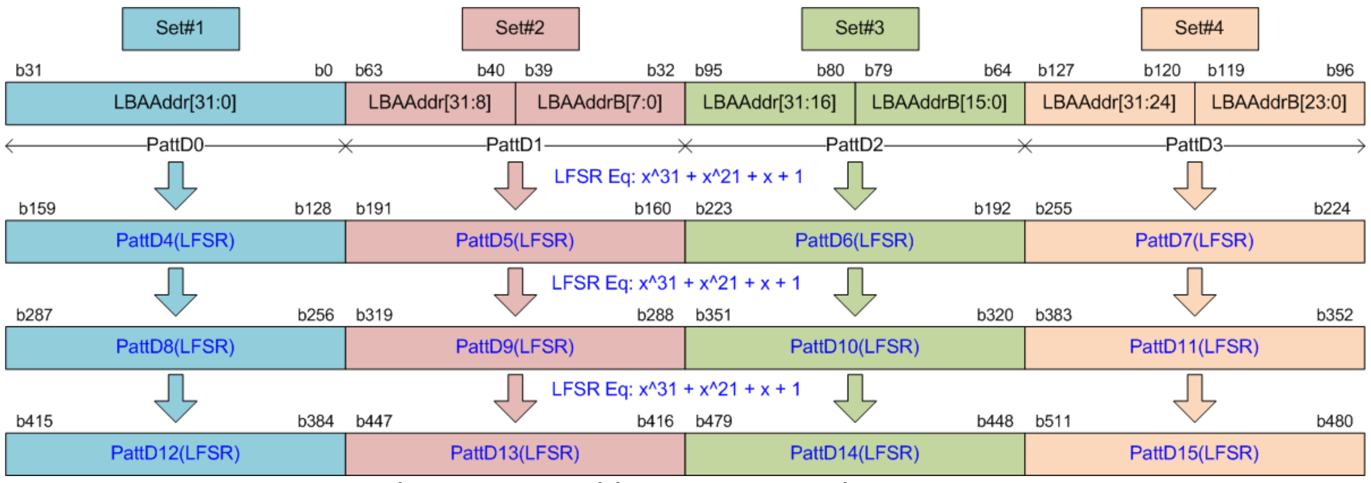

Figure 2-5 512-bit LFSR Pattern in TestGen

Using look-ahead technique, each clock cycle generates four 32-bit LFSR data or 128-bit data, represented by the same color in Figure 2-5. The start value of each data set consists of 32-bits of LBAAddr and NOT logic applied to some bit of LBAAddr (LBAAddrB) to create a different start value for each 128-bit data set. The generated test data is then written to the FIFO as write data or used as expected data for verification with the read data from the FIFO. The fail flag is set to 1b when the data verification process fails. The timing diagram for writing data to the FIFO is shown below.

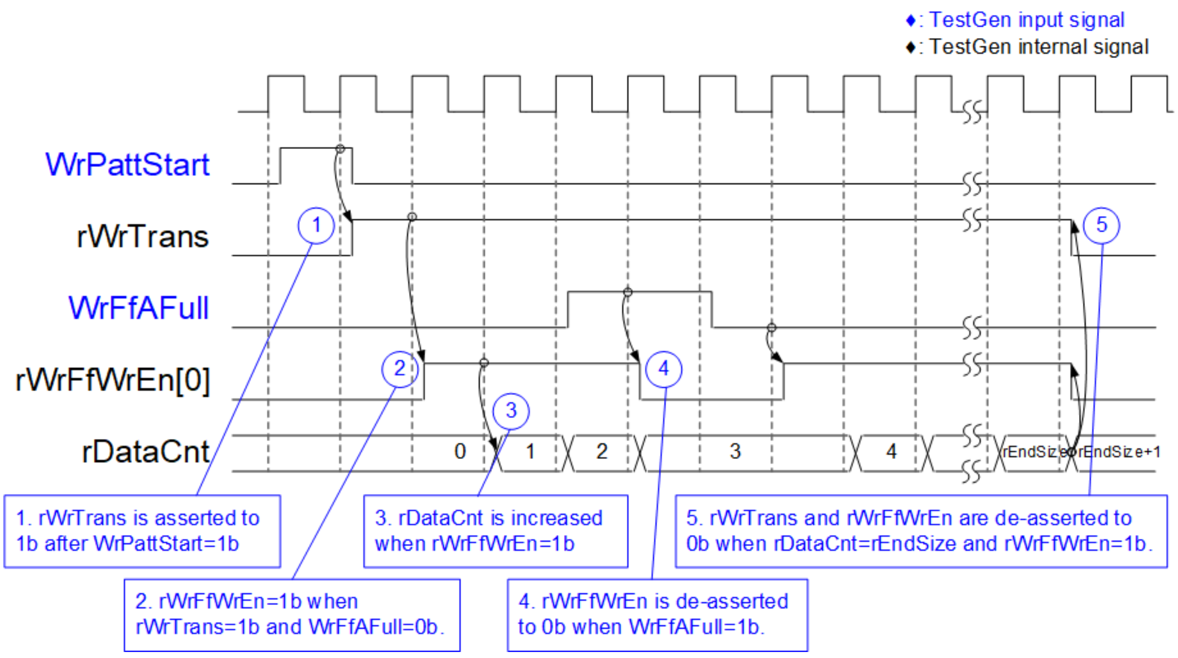
♦: TestGen input signal
♦: TestGen internal signal
WrPattStart
rWrTrans
WrFfAFull
rWrFfWrEn[0]
rDataCnt
0
1
2
3
4
rEndSize
rEndSize+1
1. rWrTrans is asserted to 1b after WrPattStart=1b
2. rWrFfWrEn=1b when rWrTrans=1b and WrFfAFull=0b.
3. rDataCnt is increased when rWrFfWrEn=1b
4. rWrFfWrEn is de-asserted to 0b when WrFfAFull=1b.
5. rWrTrans and rWrFfWrEn are de-asserted to 0b when rDataCnt=rEndSize and rWrFfWrEn=1b.

Figure 2-6 Timing diagram of Write command in TestGen

1) The write operation is initiated by setting WrPattStart signal to 1b for one 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 met (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 in 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 (rDataCnt) is equal to the set value (rEndSize). 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 expect value, respectively.

## 2.2 NVMe

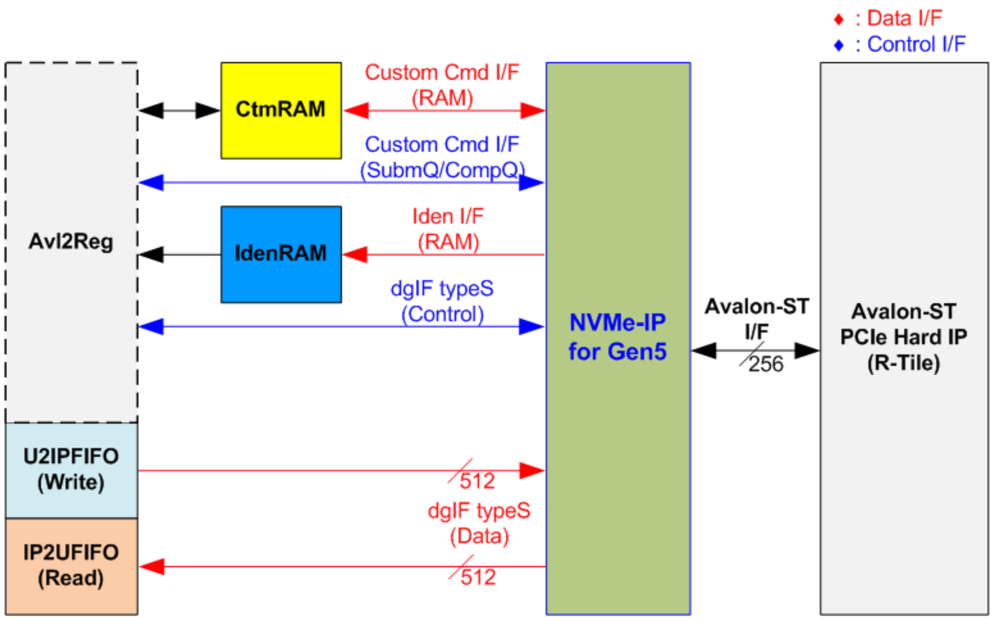

Figure 2-7 NVMe hardware

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 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 command, Flush command, or Secure Erase command.

On the other hand, the data interface of NVMe-IP has four different interfaces with a data bus width of 512-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 one directional interface. In the reference design, the Custom command RAM interface is used for one-directional data transfer when NVMe-IP sends SMART data to Avl2Reg.

### 2.2.1 NVMe-IP for Gen5

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, Flush, and Secure Erase. The NVMe-IP can be directly connected to the PCIe hard IP (R-Tile) directly. More details of NVMe-IP are described in datasheet.
https://dgway.com/products/IP/NVMe-IP/dg_nvme_datasheet_g5_intel.pdf

### 2.2.2 PCIe Hard IP (R-Tile Avalon-ST Intel FPGA for PCIe)

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

R-Tile Avalon-ST Intel FPGA for PCIe
https://www.intel.com/content/www/us/en/docs/programmable/683501/

### 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 512-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 2-8.

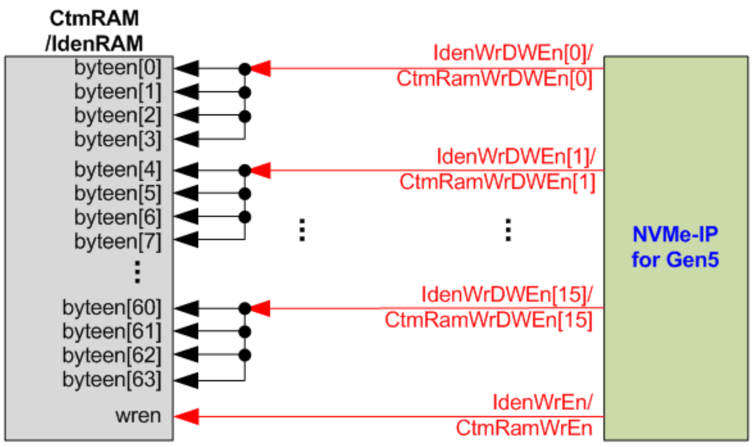

Figure 2-8 Word enable to be byte write enable connection

The 16 bits of the WrDWEn are used to drive the byte write enable of IdenRAM as follows: bits[0], [1], …, [15] of WrDWEn are fed to bits[3:0], [7:4], …, [63:60] of IdenRAM byte write enable.

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 8Kbyte 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 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 2-9, is designed to connect the CPU system via the Avalon-MM interface, in compliance with the standard.

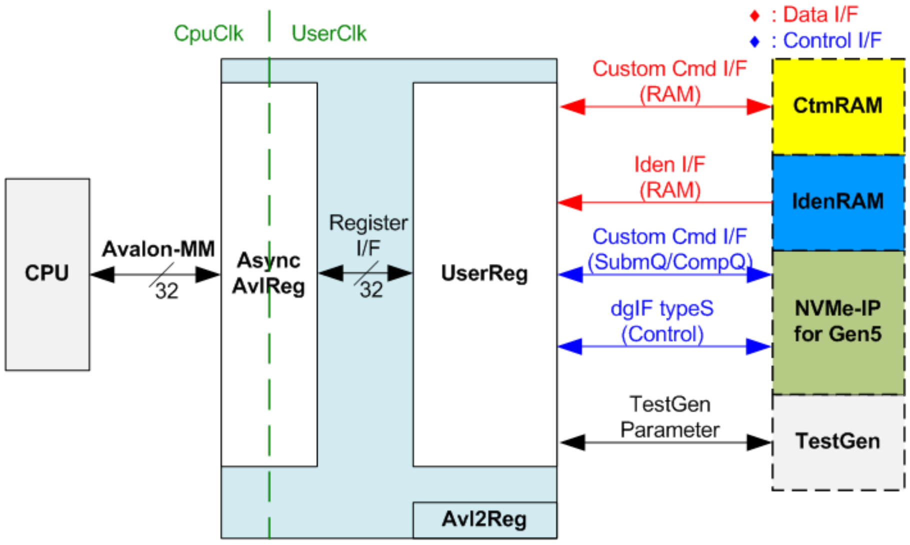

Figure 2-9 CPU and peripherals hardware

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

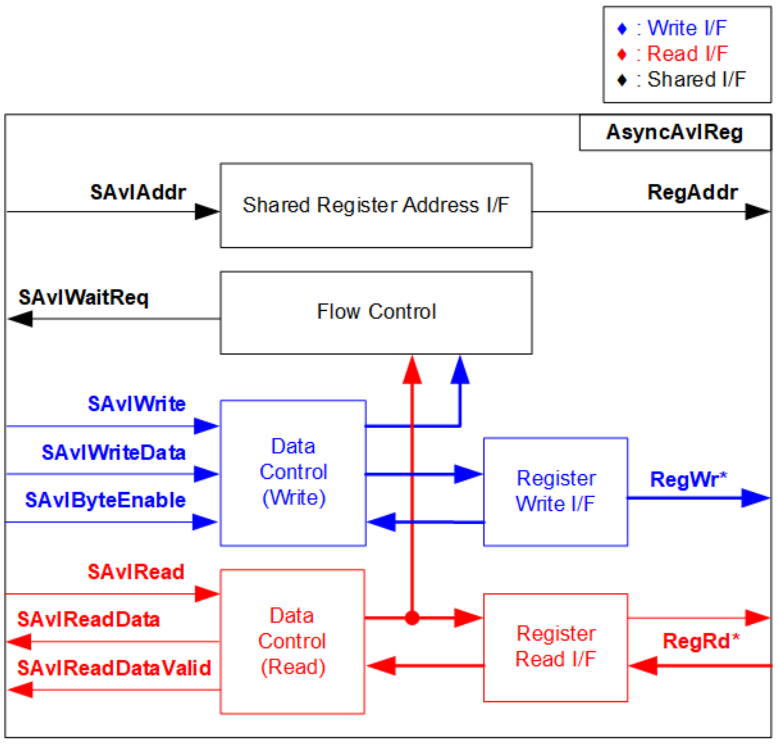

Figure 2-10 AsyncAvlReg Interface

The Avalon-MM bus interface signal can be grouped 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

According to Avalon-MM specification, only one command (write or read) can be executed at a time. AsyncAvlReg's logic is divided into three groups: Write control logic, Read control logic, and Flow control logic. The flow control logic asserts SAvlWaitReq to hold the next request from the Avalon-MM interface if the current request has not finished. Write control and Write data I/F of the Avalon-MM bus are latched and transferred to the Write register interface with clock domain crossing registers. Similarly, Read control I/F are latched and transferred to be Read register interface. Afterward, the data returned from Register Read I/F is transferred to Avalon-MM bus with 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 the single-port RAM interface for write transactions. However, the read transaction of the Register interface is slightly modified from RAM interface by adding RdReq and RdValid signals to control the read latency time. Since the address of the Register interface is shared for write and read transactions, the user cannot write and read the register at the same time. The timing diagram of the Register interface is shown in Figure 2-11.

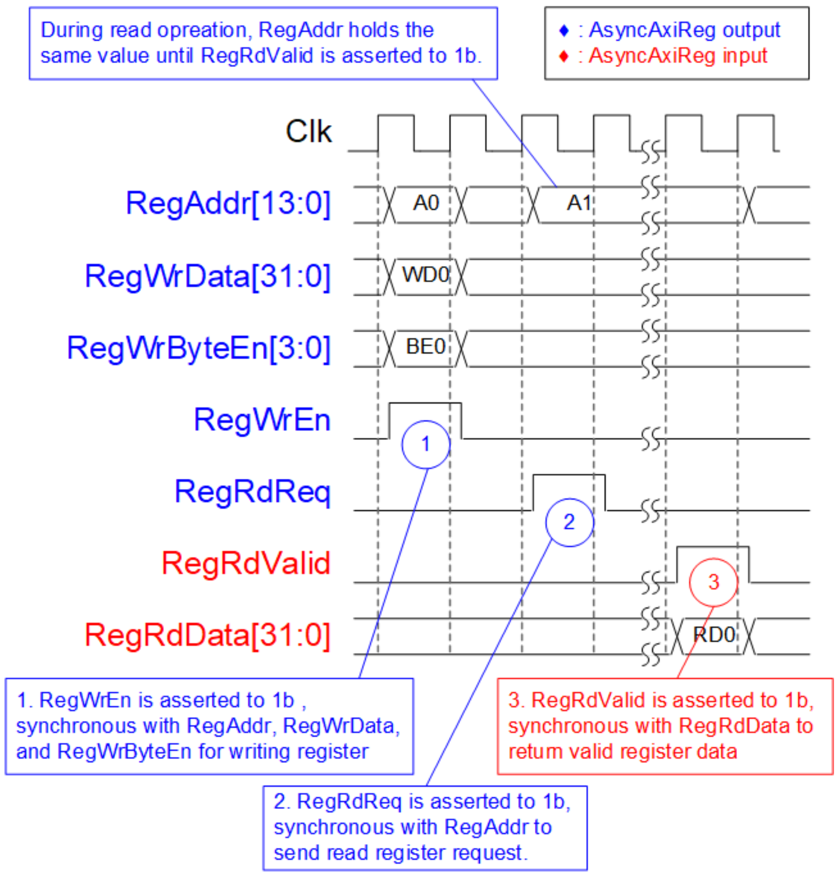
During read opreation, RegAddr holds the same value until RegRdValid is asserted to 1b.
♦ : AsyncAxiReg output
♦ : AsyncAxiReg input
Clk
RegAddr[13:0]
A0
A1
RegWrData[31:0]
WD0
RegWrByteEn[3:0]
BE0
RegWrEn
RegRdReq
RegRdValid
RegRdData[31:0]
RD0
1
2
3
1. RegWrEn is asserted to 1b , synchronous with RegAddr, 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, AsyncAvlReg 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, AsyncAvlReg forwards the read value to the SAvlRead interface.

### 2.3.2 UserReg

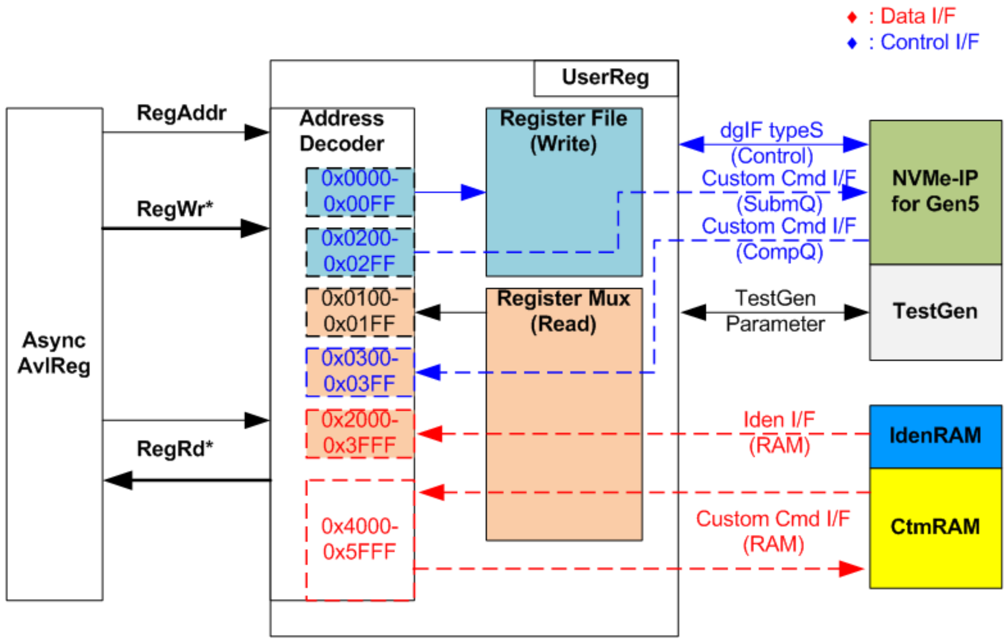

Figure 2-12 UserReg Interface

The UserReg module consists of an Address decoder, a Register File, and a Register Mux. The Address decoder decodes the address requested by AsyncAvlReg 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 the 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 using Custom command RAM interface. This area allows both write-access and read access. However, the demo shows only read-access by running the SMART command.

The Address decoder decodes the upper bits of RegAddr to select the active hardware (NVMe-IP, RdTestGen, WrTestGen, 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 a 32-bit pointer to set the hardware register.

To read the register, multi-level multiplexers (mux) select the data to return to the 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. Consequently, the latency time of read data equals to three clock cycles. Therefore, RegRdValid is created by RegRdReq, with three D Flip-flops asserted. More details of the address mapping within the UserReg module are shown in Table 2-1.

Table 2-1 Register Map

| Address<br>Wr/Rd | Register Name<br>(Label in the "nvmeipg5test.c") | Description |
|---|---|---|
| **0x0000 – 0x00FF: Control signals of NVMe-IP and TestGen (Write access only)** | | |
| BA+0x0000 | User Address (Low) Reg<br>(USRADRL_INTREG) | [31:0]: Input to be bit[31:0] of start address as 512-byte unit<br>(UserAddr[31:0] of dgIF typeS) |
| BA+0x0004 | User Address (High) Reg<br>(USRADRH_INTREG) | [15:0]: Input to be bit[47:32] of start address as 512-byte unit<br>(UserAddr[47:32] of dgIF typeS) |
| BA+0x0008 | User Length (Low) Reg<br>(USRLENL_INTREG) | [31:0]: Input to be bit[31:0] of transfer length as 512-byte unit<br>(UserLen[31:0] of dgIF typeS) |
| BA+0x000C | User Length (High) Reg<br>(USRLENH_INTREG) | [15:0]: Input to be bit[47:32] of transfer length as 512-byte unit<br>(UserLen[47:32] of dgIF typeS) |
| BA+0x0010 | User Command Reg<br>(USRCMD_INTREG) | [2:0]: Input to be user command (UserCmd of dgIF typeS for NVMe-IP)<br>000b: Identify, 001b: Shutdown, 010b: Write SSD, 011b: Read SSD,<br>100b: SMART/Secure Erase, 110b: Flush, 101b/111b: Reserved<br>When this register is written, the command request is sent to NVMe-IP to start the operation. |
| BA+0x0014 | Test Pattern Reg<br>(PATTSEL_INTREG) | [2:0]: Select test pattern<br>000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1, 100b-LFSR |
| BA+0x0020 | NVMe Timeout Reg<br>(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<br>(USRSTS_INTREG) | [0]: UserBusy of dgIF typeS (0b: Idle, 1b: Busy)<br>[1]: UserError of dgIF typeS (0b: Normal, 1b: Error)<br>[2]: Data verification fail (0b: Normal, 1b: Error) |
| BA+0x0104 | Total disk size (Low) Reg<br>(LBASIZEL_INTREG) | [31:0]: Mapped to LBASize[31:0] of NVMe-IP |
| BA+0x0108 | Total disk size (High) Reg<br>(LBASIZEH_INTREG) | [15:0]: Mapped to LBASize[47:32] of NVMe-IP<br>[31]: Mapped to LBAMode of NVMe-IP |
| BA+0x010C | User Error Type Reg<br>(USRERRTYPE_INTREG) | [31:0]: Mapped to UserErrorType[31:0] of NVMe-IP to show error status |
| BA+0x0110 | PCIe Status Reg<br>(PCIESTS_INTREG) | [0]: PCIe linkup status from PCIe hard IP (0b: No linkup, 1b: linkup)<br>[3:2]: Two lower bits to show PCIe link speed.<br>Two upper bits are bit[17:16].<br>(0000b: Not linkup, 0001b: PCIe Gen1, 0010b: PCIe Gen2,<br>0011b: PCIe Gen3, 0111b: PCIe Gen4, 1111b: PCIe Gen5)<br>[6:4]: PCIe link width status from PCIe hard IP<br>(001b: 1-lane, 010b: 2-lane, 100b: 4-lane)<br>[13:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in Avalon-ST PCIe Hard IP datasheet<br>[17:16]: Two upper bits to show PCIe link speed.<br>Two lower bits are bit[3:2]. |
| BA+0x0114 | Completion Status Reg<br>(COMPSTS_INTREG) | [15:0]: Mapped to AdmCompStatus[15:0] of NVMe-IP<br>[31:16]: Mapped to IOCompStatus[15:0] of NVMe-IP |
| BA+0x0118 | NVMe CAP Reg<br>(NVMCAP_INTREG) | [31:0]: Mapped to NVMeCAPReg[31:0] of NVMe-IP |
| (BA+0x0120) –<br>(BA+0x012F) | NVMe IP Test pin Word0-3 Reg<br>(NVMTESTPIN0-3_INTREG) | Mapped to TestPin[127:0] of NVMe-IP<br>0x0120: Bit[31:0], 0x0124: Bit[63:32],<br>0x0128: Bit[95:64], 0x012C: Bit[127:96] |

| Address<br>Wr/Rd | Register Name<br>(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<br>(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<br>(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<br>(CURTESTSIZEL_INTREG) | [31:0]: Bit[31:0] of the current test data size in TestGen module |
| BA+0x017C | Current test byte (High) Reg<br>(CURTESTSIZEH_INTREG) | [24:0]: Bit[56:32] of the current test data size of TestGen module |
| (BA+0x0180) – (BA+0x01BF) | Expected value Word0-15 Reg<br>(EXPPATW0-W15_INTREG) | 512-bit of the expected data at the 1st failure data in Read command<br>0x0180: Bit[31:0], 0x0184: Bit[63:32],…, 0x01BC: Bit[511:480] |
| (BA+0x01C0) – (BA+0x01FF) | Read value Word0-15 Reg<br>(RDPATW0-W15_INTREG) | 512-bit of the read data at the 1st failure data in Read command<br>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)<br>Wr | Custom Submission Queue Reg<br>(CTMSUBMQ_STRUCT) | [31:0]: Submission queue entry of SMART and Flush command.<br>Input to be CtmSubmDW0-DW15 of NVMe-IP.<br>0x200: DW0, 0x204: DW1, …, 0x23C: DW15 |
| (BA+0x0300) – (BA+0x030F)<br>Rd | Custom Completion Queue Reg<br>(CTMCOMPQ_STRUCT) | [31:0]: CtmCompDW0-DW3 output from NVMe-IP.<br>0x300: DW0, 0x304: DW1, …, 0x30C: DW3 |
| BA+0x0800<br>Rd | IP Version Reg<br>(IPVERSION_INTREG) | [31:0]: Mapped to IPVersion[31:0] of NVMe-IP |
| (BA+0x2000) – (BA+0x2FFF)<br>Rd | Identify Controller Data<br>(IDENCTRL_CHARREG) | 4Kbyte Identify controller data structure |
| (BA+0x3000) – (BA+0x3FFF)<br>Rd | Identify Namespace Data<br>(IDENNAME_CHARREG) | 4Kbyte Identify Namespace Data Structure |
| (BA+0x4000) – (BA+0x5FFF)<br>Wr/Rd | Custom command Ram<br>(CTMRAM_CHARREG) | Connect to 8K byte CtmRAM interface for storing 512-byte data output from SMART command. |

# 3 CPU Firmware

## 3.1 Test firmware (nvmeipg5test.c)

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

1) Initializes JTAG UART and Timer parameters.
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). The error message is displayed and the process stops when some errors are found.
4) CPU displays PCIe link status (the number of PCIe lanes and the PCIe speed) by reading PCIESTS_INTREG[17:2].
5) CPU displays the main menu. There are seven menus for running seven commands of NVMe-IP, i.e., Identify, Write, Read, SMART, Flush, Secure Erase and Shutdown.

More details of the operation flow in each command are described as follows.

### 3.1.1 Identify Command

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

1) Set USRCMD_INTREG[2:0]=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 will wait until the operation is completed or an error is detected by monitoring USRSTS_INTREG[1:0].

   - If bit[0] is de-asserted to 0b after the operation is finished, the data of Identify command returned by NVMe-IP will be stored in IdenRAM.
   - If 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 the busy flag (USRSTS_INTREG[0]) is de-asserted to 0b, the CPU will display some information decoded from IdenRAM (IDENCTRL_CHARREG), such as the SSD model name and the information from NVMe-IP output such as SSD capacity, LBA unit size (LBASIZEH/L_INTREG) and Secure Erase command support.

### 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 JTAG UART. If any inputs are invalid, the operation will be cancelled.
   *Note: If LBA unit size = 4 Kbyte, start address and transfer length must be aligned to 8.*
2) After obtaining all the inputs, set them to USRADRL/H_INTREG, USRLENL/H_INTREG, and PATTSEL_INTREG.
3) To execute the Write or Read command, set USRCMD_INTREG[2:0]= 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 will wait until the operation is completed or an error (excluding verification error) is detected by monitoring USRSTS_INTREG[2:0].

   - Bit[0] will be de-asserted to 0b when the command is completed.
   - Bit[1] will be asserted when an error is detected. In this case, the error message will be displayed on the console, which will show the decoded error details from USRERRTYPE_INTREG[31:0], and the process will be stopped.
   - Bit[2] will be asserted when data verification fails. In this case, the verification error message will be displayed on the console, but the CPU will continue to run until the operation is complete or the user cancels the operation by pressing any key.

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

5) After the busy flag (USRSTS_INTREG[0]) is de-asserted to 0b, CPU will calculates 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 USRCMD_INTREG[2:0]=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 will wait until the operation is completed or an error is detected by monitoring USRSTS_INTREG[1:0].

   - If 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.
   - If 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.

4) After the busy flag (USRSTS_INTREG[0]) is de-asserted to 0b, CPU the CPU will display information decoded from CtmRAM (CTMRAM_CHARREG), i.e., 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/resources/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 USRCMD_INTREG[2:0]=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 will wait until the operation is completed or an error is detected by monitoring USRSTS_INTREG[1:0].

   - If bit[0] is de-asserted to 0b after the operation is finished, the CPU will then return to the main menu.
   - If 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.

### 3.1.5 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 USRCMD_INTREG[2:0]=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 will wait until the operation is completed or an error is detected by monitoring USRSTS_INTREG[1:0].

   - If bit[0] is de-asserted to 0b after the operation is finished, the CPU will then proceed to the next step.
   - If bit[1] is asserted, indicating an error, the error message will be displayed on the console with the 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 step to operate Shutdown command is described as follows.

1) Set USRCMD_INTREG[2:0]=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 will wait until the operation is completed or an errors is detected by monitoring USRSTS_INTREG[1:0].

   - If bit[0] is de-asserted to 0b after the operation is finished, the CPU will then proceed to the next step.
   - If 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,), execute Flush command as outlined in section 3.1.4 (Flush Command), or execute Secure Erase command as outlined in section 3.1.5 (Secure Erase Command). |

| unsigned long long get_cursize(void) | |
|---|---|
| Parameters | None |
| Return value | Read value of CURTESTSIZEH/L_INTREG |
| Description | The value of CURTESTSIZEH/L_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 exectute 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 signal is read by NVMTESTPIN0-3_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 MByte, GByte, or TByte unit |

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

| void show_smart_int8byte(volatile unsigned char *char_ptr) | |
|---|---|
| Parameters | *char_ptr: Pointer of 8-byte SMART data |
| Return value | None |
| Description | When the input value is less than 4 billion (32-bit), 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-W15_INTREG (expected value), and RDPATW0-W15_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 3.1.5. (Shutdown Command) |

| int wrrd_dev(unsigned int user_cmd) | |
|---|---|
| Parameters | user_cmd: 2-Write command, 3-Read command |
| Return value | 0: No error, -1: Receive invalid input or some errors are found. |
| Description | Execute Write command or Read command as outlined in section 3.1.2. (Write/Read Command). Show_result function is called to calculate and display transfer performance in Write/Read command. |

## 4 Example Test Result

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

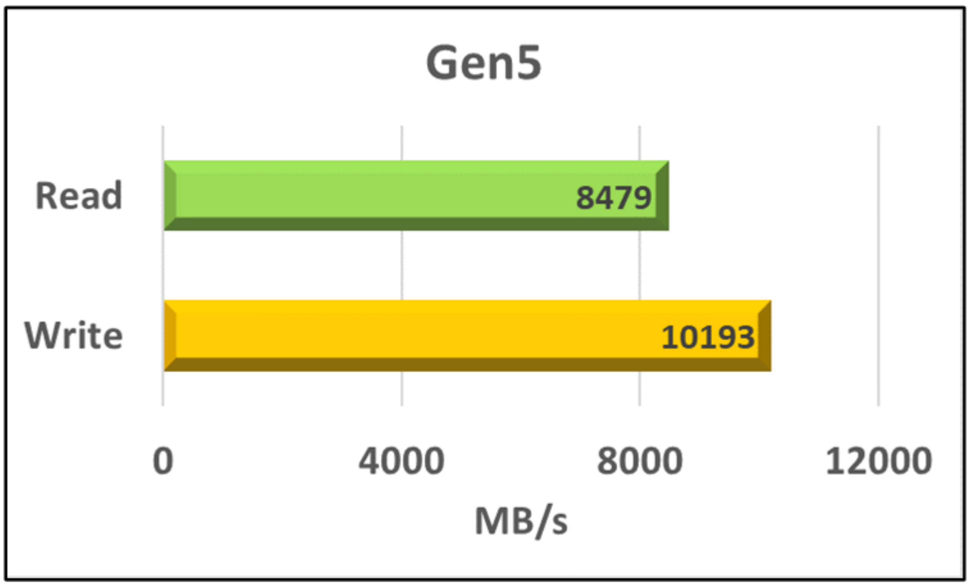

Figure 4-1 Test Performance of NVMe-IP demo by using CFD Gaming PG5NFZ SSD

Utilizing the Agilex 7 I-Series board with PCIe Gen5, the system demonstrates a remarkable write performance of approximately 10,000 Mbyte/sec, along with a read performance of around 8,400 Mbyte/sec.

# 5 Revision History

| Revision | Date | Description |
|---|---|---|
| 1.0 | 18-May-23 | Initial Release |