NVMe-IP reference design manual

Rev3.5 12-Dec-23

1 Overview

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

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

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

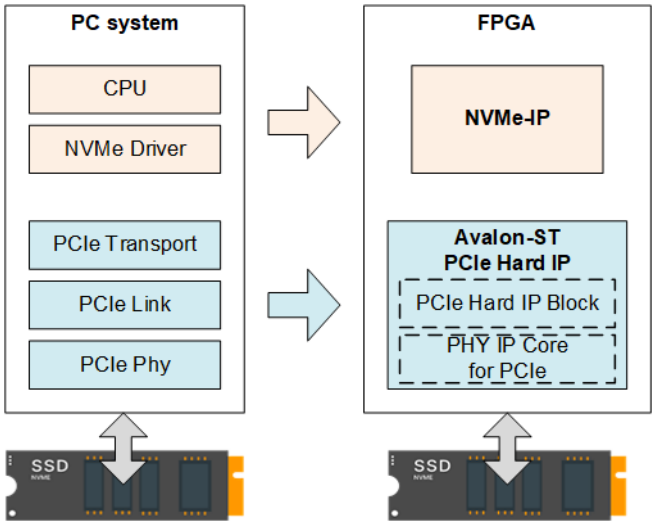

Figure 1-1 NVMe protocol layer

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

2 Hardware overview

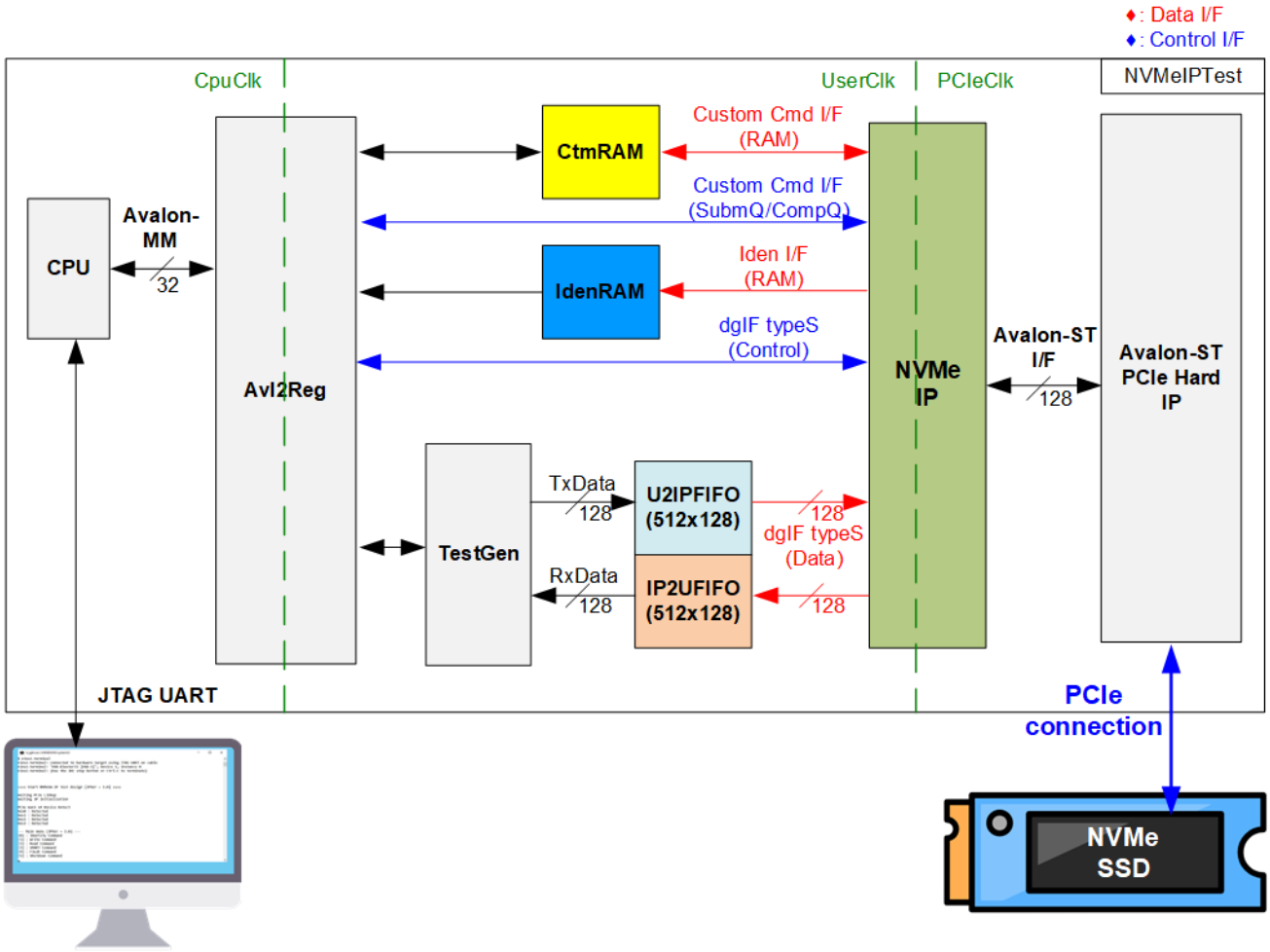

Figure 2-1 NVMe-IP demo hardware

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

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

NVMe consists of the NVMe-IP and the PCIe hard IP (Avalon-ST PCIe hard IP) for accessing an NVMe SSD directly without PCIe switch. The command request and the parameters of each command (the inputs of NVMe-IP) are controlled by the CPU through 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. Using CPU also facilitates the execution of many test cases to verify the functionality of the IP. The default firmware for the CPU includes the functions for executing the NVMe commands by using NVMe-IP.

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

More details of the hardware are described as follows.

2.1 TestGen

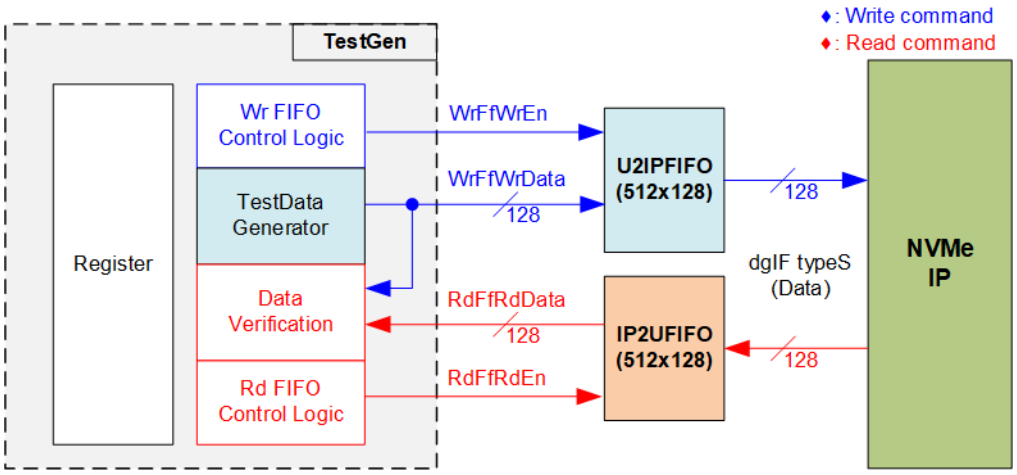

Figure 2-2 TestGen interface

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

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

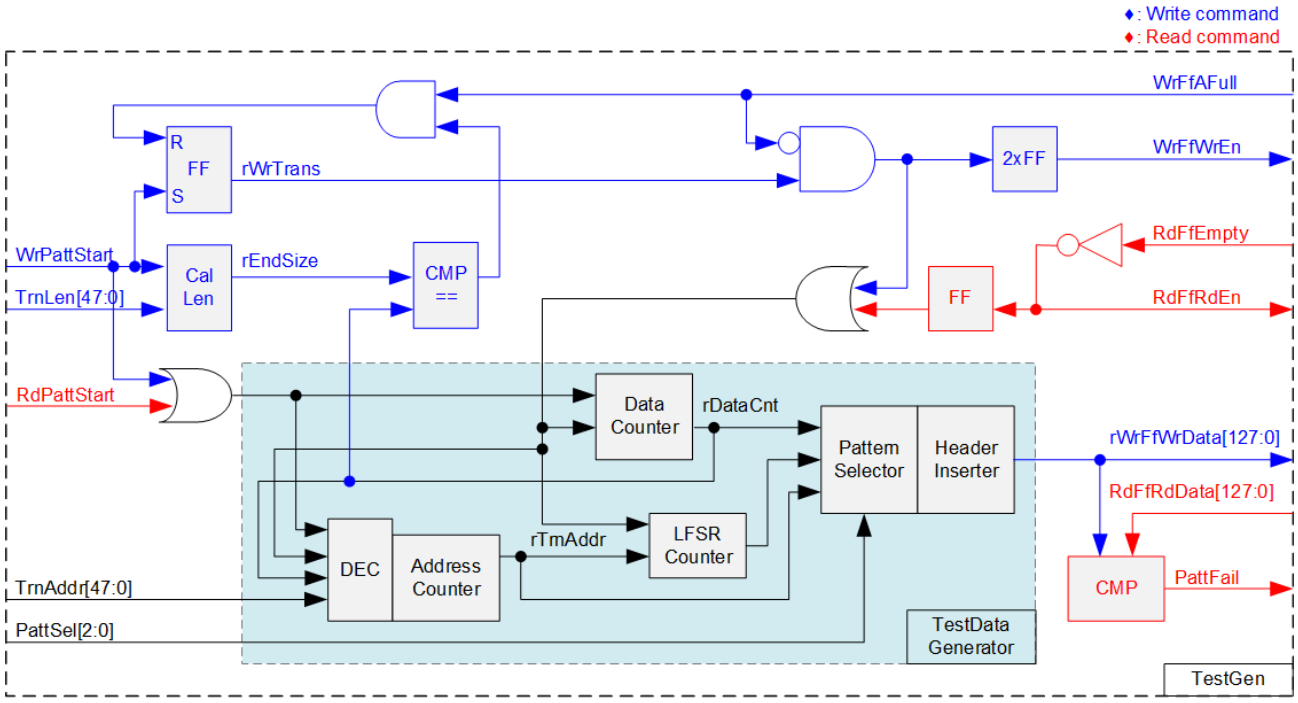

Figure 2-3 TestGen hardware

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

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

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

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

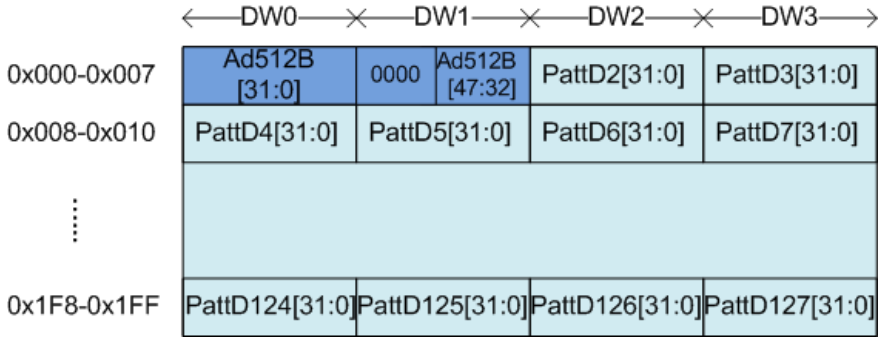

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

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

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

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

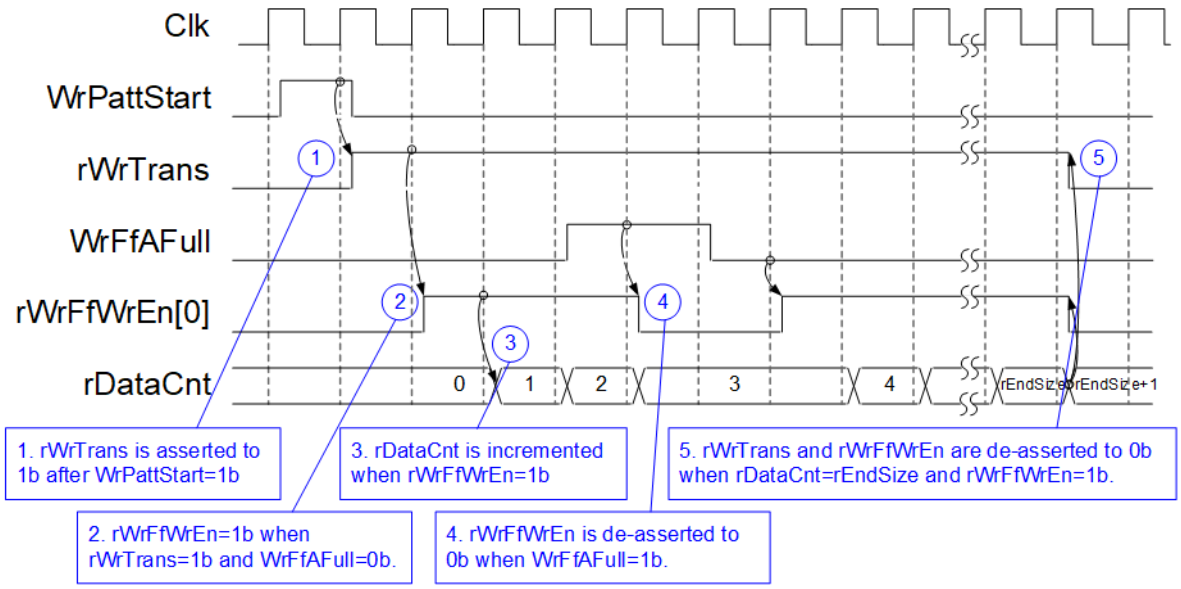

Figure 2-5 Timing diagram of Write operation in TestGen

1) The write operation is initiated by setting WrPattStart signal to 1b for a single clock cycle. 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 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

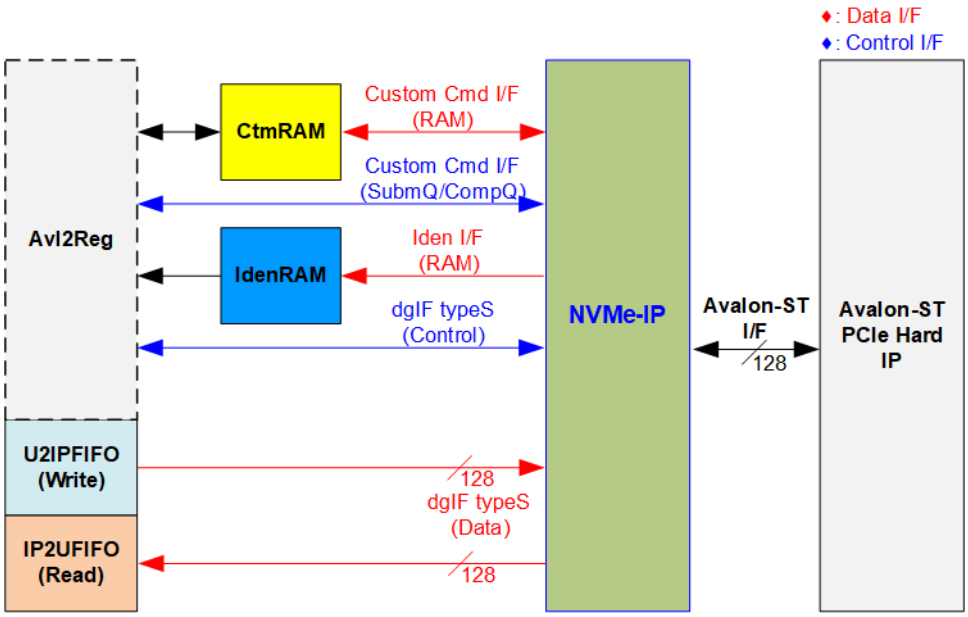

Figure 2-6 NVMe hardware

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

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 NVMe protocol of the host side to direct access an NVMe SSD without PCIe switch connection. It supports seven commands, i.e., Write, Read, Identify, Shutdown, SMART, Secure Erase, and Flush. The NVMe-IP can be directly connected to the PCIe hard IP directly. More details of NVMe-IP are described in datasheet.
https://dgway.com/products/IP/NVMe-IP/dg_nvmeip_datasheet_intel_en/

2.2.2 Avalon-ST PCIe Hard IP

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

ArriaV Avalon-ST Interface for PCIe Solutions User Guide
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_a5_pcie_avst.pdf

Stratix V Avalon-ST Interface for PCIe Solutions User Guide
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_s5_pcie_avst.pdf

Intel Arria10 and Intel Cyclone 10 GX Avalon-ST Interface for PCIe Solutions User Guide
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_a10_pcie_avst.pdf

2.2.3 Two-port RAM

Within the system, two 2-Port RAMs, CtmRAM and IdenRAM, store the returned data from Identify command and SMART command, respectively. Specifically, IdenRAM is designed with an 8 KB capacity, storing the 8 KB data response 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 configured as an asymmetric RAM, with different bus sizes for its Write and Read interfaces.

NVMe-IP is equipped with a double-word enable, enabling it to write only 32-bit data under certain cases. In the Quartus tool, the RAM settings support write byte enable functionality, and this double-word enable can be extended to a 4-bit write byte enable, as shown in Figure 2-7.

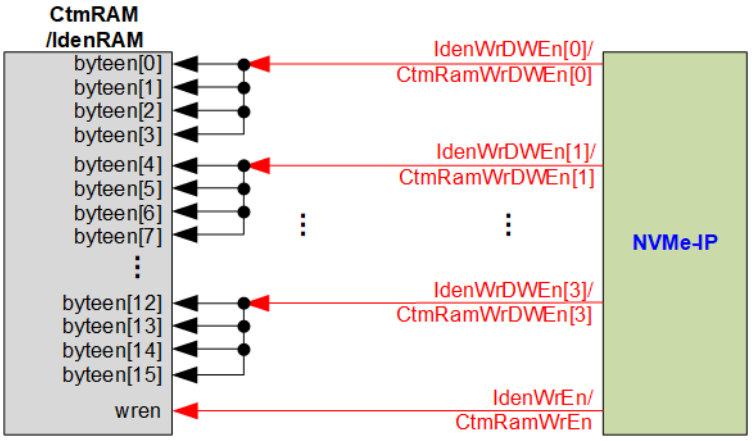

Figure 2-7 Word enable to be byte write enable connection

Each bit of WrDWEn 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 2-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 2-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 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 2-8, is designed to connect the CPU system via the Avalon-MM interface, in compliance with the standard.

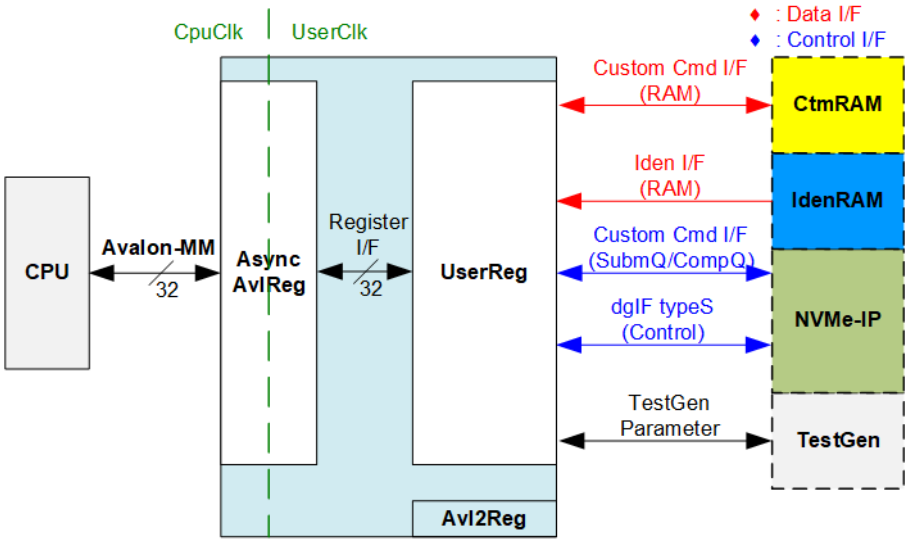

Figure 2-8 CPU and peripherals hardware

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

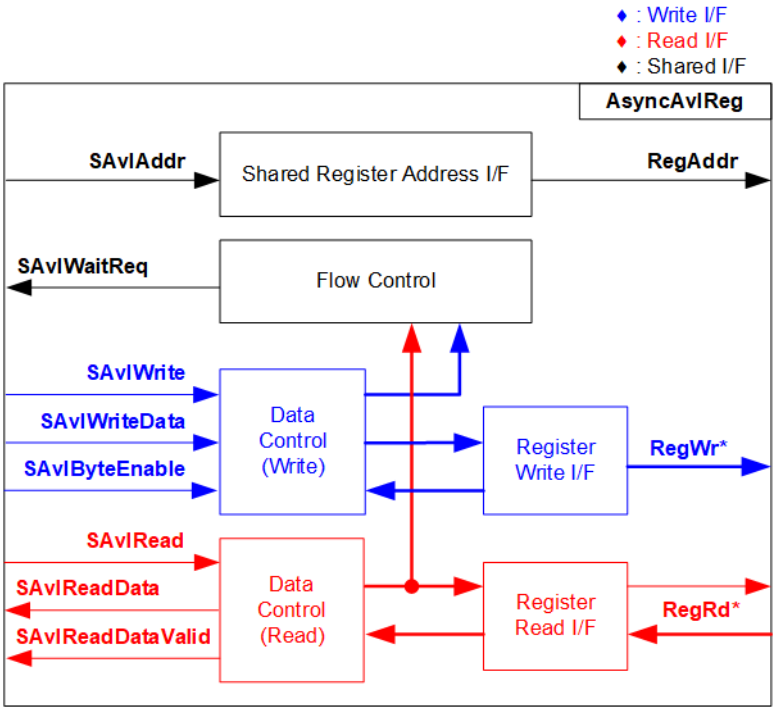

Figure 2-9 AsyncAvlReg Interface

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 2-10.

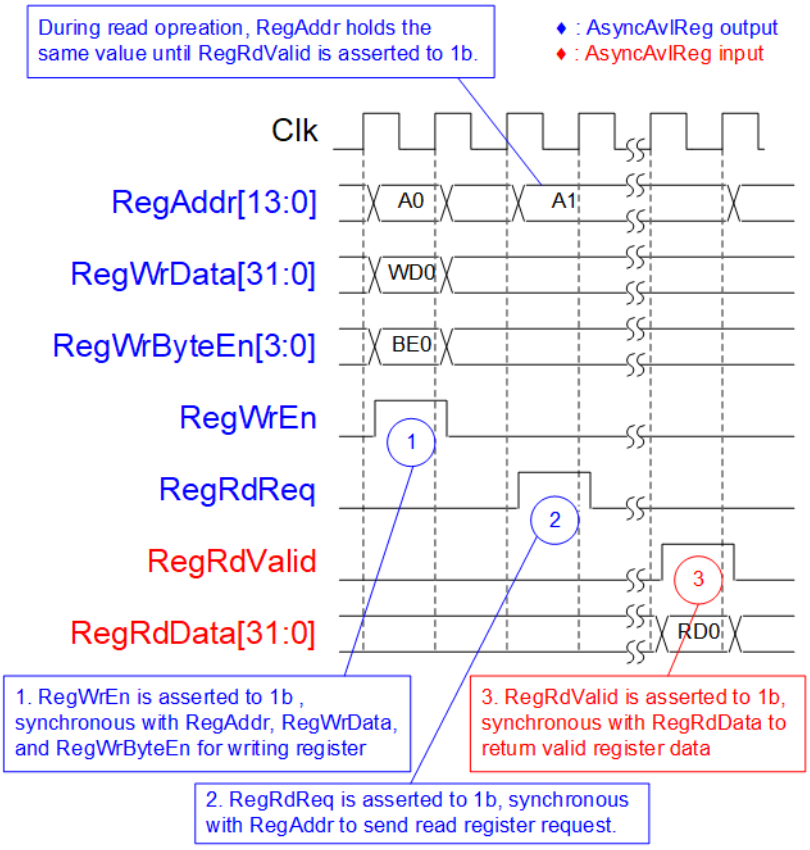

Figure 2-10 Register interface timing diagram

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

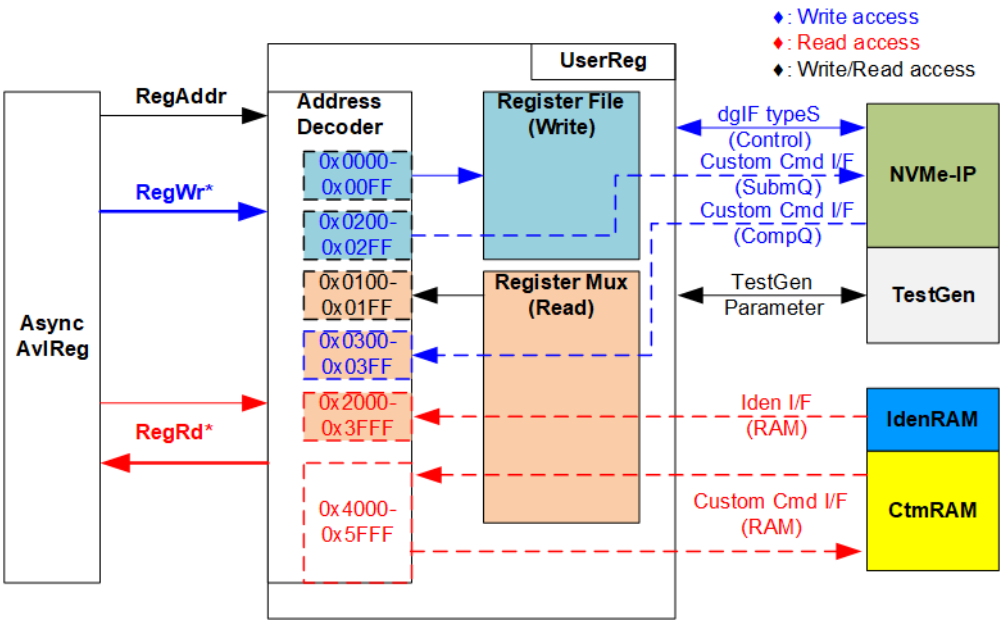

Figure 2-11 UserReg Interface

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 2-11.

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

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

To read the register, multi-level multiplexers (mux) select the data to return to CPU by using the address. The lower bits of RegAddr are fed to the submodule to select the active data from each submodule. While the upper bits are used in UserReg to select the returned data from each submodule. Consequently, the latency time of read data equals to two clock cycles, and RegRdValid is created by RegRdReq, with two 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 Wr/Rd	Register Name (Label in the “nvmeiptest.c”)	Description
0x0000 – 0x00FF: Control signals of NVMe-IP and TestGen (Write access only)		
BA+0x0000	User Address (Low) Reg (USRADRL_INTREG)	[31:0]: Input to be bit[31:0] of start address as 512-byte unit (UserAddr[31:0] of dgIF typeS)
BA+0x0004	User Address (High) Reg (USRADRH_INTREG)	[15:0]: Input to be bit[47:32] of start address as 512-byte unit (UserAddr[47:32] of dgIF typeS)
BA+0x0008	User Length (Low) Reg (USRLENL_INTREG)	[31:0]: Input to be bit[31:0] of transfer length as 512-byte unit (UserLen[31:0] of dgIF typeS)
BA+0x000C	User Length (High) Reg (USRLENH_INTREG)	[15:0]: Input to be bit[47:32] of transfer length as 512-byte unit (UserLen[47:32] of dgIF typeS)
BA+0x0010	User Command Reg (USRCMD_INTREG)	[2:0]: Input to be user command (UserCmd of dgIF typeS for NVMe-IP) 000b: Identify, 001b: Shutdown, 010b: Write SSD, 011b: Read SSD, 100b: SMART/Secure Erase, 110b: Flush, 101b/111b: Reserved When this register is written, the command request is sent to NVMe-IP to start the operation.
BA+0x0014	Test Pattern Reg (PATTSEL_INTREG)	[2:0]: Select test pattern 000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1, 100b-LFSR
BA+0x0020	NVMe Timeout Reg (NVMTIMEOUT_INTREG)	[31:0]: Mapped to TimeOutSet[31:0] of NVMe-IP
0x0100 – 0x01FF: Status signals of NVMe-IP and TestGen (Read access only)		
BA+0x0100	User Status Reg (USRSTS_INTREG)	[0]: UserBusy of dgIF typeS (0b: Idle, 1b: Busy) [1]: UserError of dgIF typeS (0b: Normal, 1b: Error) [2]: Data verification fail (0b: Normal, 1b: Error)
BA+0x0104	Total disk size (Low) Reg (LBASIZEL_INTREG)	[31:0]: Mapped to LBASize[31:0] of NVMe-IP
BA+0x0108	Total disk size (High) Reg (LBASIZEH_INTREG)	[15:0]: Mapped to LBASize[47:32] of NVMe-IP [31]: Mapped to LBAMode of NVMe-IP
BA+0x010C	User Error Type Reg (USRERRTYPE_INTREG)	[31:0]: Mapped to UserErrorType[31:0] of NVMe-IP to show error status
BA+0x0110	PCIe Status Reg (PCIESTS_INTREG)	[0]: PCIe linkup status from PCIe hard IP (0b: No linkup, 1b: linkup) [3:2]: PCIe link speed from PCIe hard IP (00b: Not linkup, 01b: PCIe Gen1, 10b: PCIe Gen2, 11b: PCIe Gen3) [7:4]: PCIe link width status from PCIe hard IP (0001b: 1-lane, 0010b: 2-lane, 0100b: 4-lane, 1000b: 8-lane) [12:8]: 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
0x0100 – 0x01FF: Status signals of NVMe-IP and TestGen (Read access only)		
BA+0x0130 - BA+0x013F	Expected value Word0-3 Reg (EXPPATW0-W3_INTREG)	The 128-bit expected data of the 1st failure when executing a Read command. 0x0130: Bit[31:0], 0x0134[31:0]: Bit[63:32], …, 0x013C[31:0]: Bit[127:96]
BA+0x0150 - BA+0x015F	Read value Word0-3 Reg (RDPATW0-W3_INTREG)	The 128-bit read data of the 1st failure when executing a Read command. 0x0150: Bit[31:0], 0x0154[31:0]: Bit[63:32], …, 0x015C[31:0]: Bit[127:96]
BA+0x0170	Data Failure Address(Low) Reg (RDFAILNOL_INTREG)	[31:0]: Bit[31:0] of the byte address of the 1st failure when executing a Read command
BA+0x0174	Data Failure Address(High) Reg (RDFAILNOH_INTREG)	[24:0]: Bit[56:32] of the byte address of the 1st failure when executing a Read command
BA+0x0178	Current test byte (Low) Reg (CURTESTSIZEL_INTREG)	[31:0]: Bit[31:0] of the current test data size in TestGen module
BA+0x017C	Current test byte (High) Reg (CURTESTSIZEH_INTREG)	[24:0]: Bit[56:32] of the current test data size of TestGen module
Other interfaces (Custom command of NVMe-IP, IdenRAM, and Custom RAM)		
BA+0x0200- BA+0x023F Wr	Custom Submission Queue Reg (CTMSUBMQ_STRUCT)	[31:0]: Submission queue entry of SMART, Secure Erase, and Flush commands. Input to be CtmSubmDW0-DW15 of NVMe-IP. 0x200: DW0, 0x204: DW1, …, 0x23C: DW15
BA+0x0300– BA+0x030F Rd	Custom Completion Queue Reg (CTMCOMPQ_STRUCT)	[31:0]: CtmCompDW0-DW3 output from NVMe-IP. 0x300: DW0, 0x304: DW1, …, 0x30C: DW3
BA+0x0800 Rd	IP Version Reg (IPVERSION_INTREG)	[31:0]: Mapped to IPVersion[31:0] of NVMe-IP
BA+0x2000- BA+0x2FFF Rd	Identify Controller Data (IDENCTRL_CHARREG)	4KB Identify Controller Data structure
BA+0x3000– BA+0x3FFF Rd	Identify Namespace Data (IDENNAME_CHARREG)	4KB Identify Namespace Data structure
BA+0x4000– BA+0x5FFF Wr/Rd	Custom command Ram (CTMRAM_CHARREG)	Connect to 8KB CtmRAM interface for storing 512-byte data output from SMART Command.

3 CPU Firmware

3.1 Test firmware (nvmeiptest.c)

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

1) Initialize JTAG UART and Timer settings.
2) Wait for the PCIe connection to become active (PCIESTS_INTREG[0]=1b).
3) Wait for NVMe-IP to complete its own initialization process (USRSTS_INTREG[0]=0b). If any errors occur during this phase, the process is halted, and an error message is displayed.
4) Display the status of the PCIe link, which includes details such as the number of lanes and the speed. This information is obtained by reading the status in PCIESTS_INTREG[7:2].
5) Display the main menu, which offers options to execute seven commands for NVMe-IP: Identify, Write, Read, SMART, Flush, Secure Erase, and Shutdown.

Additional details of the sequence for each command in the CPU firmware are described in the following sections.

3.1.1 Identify Command

When the user selects the Identify command, the firmware follows this sequence.

1) Set bits[2:0] of USRCMD_INTREG to 000b to send the Identify command request to NVMe-IP. Subsequently, the busy flag (USRSTS_INTREG[0]) changes from 0b to 1b.
2) The CPU enters a waiting state, monitoring USRSTS_INTREG[1:0] for the operation status or any error.
   - Bit[0] is de-asserted to 0b when the command is completed. Also, the data response from Identify command, output from NVMe-IP, is stored in IdenRAM.
   - Bit[1] is asserted when an error is detected. In this case, the error message will be displayed on the console, revealing error 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 proceeds to display information that has been decoded from LBASIZEL/H_INTREG, which includes the SSD capacity and LBA unit size. Besides, further information, such as the SSD model, can be retrieved from the IdenRAM (IDENCTRL_CHARREG).

3.1.2 Write/Read Command

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

1) Receive start address, transfer length, and test pattern from the console. If any of these inputs are invalid, the operation will be cancelled.
   Note: If LBA unit size = 4 KB, the start address and transfer length must align to 8.
2) Once all the required inputs are obtained, set them to their respective registers: USRADRL/H_INTREG, USRLENL/H_INTREG, and PATTSEL_INTREG.
3) To execute either the Write or Read command, set bits[2:0] of USRCMD_INTREG to 010b or 011b, respectively. This sends the command request to the NVMe-IP. Once the command is issued, the busy flag of NVMe-IP (USRSTS_INTREG[0]) will change from 0b to 1b.
4) The CPU enters a waiting state, monitoring USRSTS_INTREG[2:0] for the operation status or any error (excluding verification error).
   - Bit[0] is de-asserted to 0b when the command is completed.
   - Bit[1] is asserted when an error is detected. In this case, the error message will be displayed on the console, revealing error details decoded from USRERRTYPE_INTREG[31:0]. The process will then be terminated.
   - Bit[2] is asserted when data verification fails. In this case, a verification error message will be displayed on the console, but the CPU will continue running until the operation is completed or until the user inputs any key to cancel the operation.

   Additionally, during the command's execution, the current transfer size, as read from CURTESTSIZEL/H_INTREG, will be displayed every second.

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

3.1.3 SMART Command

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

1) Configure the 16-Dword Submission Queue entry (CTMSUBMQ_STRUCT) with 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. This action causes the busy flag (USRSTS_INTREG[0]) to change from 0b to 1b.
3) The CPU enters a waiting state, monitoring USRSTS_INTREG[1:0] for the operation status or any error.
   - Bit[0] is de-asserted to 0b when the command is completed. Also, the data response from SMART command, output from NVMe-IP, is stored in CtmRAM.
   - Bit[1] is asserted when an error is detected. In this case, the error message will be displayed on the console, revealing error details decoded 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 proceeds to display information that has been 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 additional details regarding the SMART log, refer to the NVM Express Specification at https://nvmexpress.org/specifications/

3.1.4 Flush Command

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

1) Configure the 16-Dword Submission Queue entry (CTMSUBMQ_STRUCT) 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. This action causes the busy flag (USRSTS_INTREG[0]) to change from 0b to 1b.
3) The CPU enters a waiting state, monitoring USRSTS_INTREG[1:0] for the operation status or any error.
   - Bit[0] is de-asserted to 0b when the command is completed. The CPU will then return to the main menu.
   - Bit[1] is asserted when an error is detected. In this case, the error message will be displayed on the console, revealing error details decoded 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) Configure the 16-Dword of Submission Queue entry (CTMSUBMQ_STRUCT) with the Secure Erase command value.
2) Set NVMTIMEOUT_INTREG to 0 to disable timer to prevent the timeout error.
3) Set bits[2:0] of USRCMD_INTREG[2:0] to 100b to send Secure Erase command request to NVMe-IP. This action causes the busy flag (USRSTS_INTREG[0]) to change from 0b to 1b.
4) The CPU enters a waiting state, monitoring USRSTS_INTREG[1:0] for the operation status or any error.
   - Bit[0] is de-asserted to 0b when the command is completed.
   - Bit[1] is asserted when an error is detected. In this case, the error message will be displayed on the console, revealing error details decoded 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 timer is re-enabled to generate timeout error in NVMe-IP by setting NVMTIMEOUT_INTREG to the default value. 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. This action causes the busy flag (USRSTS_INTREG[0]) to change from 0b to 1b.
2) The CPU enters a waiting state, monitoring USRSTS_INTREG[1:0] for the operation status or any error.
   - Bit[0] is de-asserted to 0b when the command is completed.
   - Bit[1] is asserted when an error is detected. In this case, the error message will be displayed on the console, revealing error 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, both the SSD and NVMe-IP 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

int exec_ctm(unsigned int user_cmd)	
Parameters	user_cmd: 4-SMART command, 6-Flush command
Return value	0: No error, -1: Some errors are found in the NVMe-IP
Description	Execute SMART command as outlined in section 3.1.3 (SMART Command) or execute Flush command as outlined in section 3.1.4 (Flush Command).

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

int get_param(userin_struct* userin)	
Parameters	userin: Three inputs from user, i.e., start address, total length in 512-byte unit, and test pattern
Return value	0: Valid input, -1: Invalid input
Description	Receive the input parameters from the user and verify the value. When the input is invalid, the function returns -1. Otherwise, all inputs are updated to userin parameter.

void iden_dev(void)	
Parameters	None
Return value	None
Description	Execute Identify command as outlined in section 3.1.1 (Identify Command).

int setctm_erase(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the NVMe-IP
Description	Set Secure Erase command to CTMSUBMQ_STRUCT and call exec_ctm function to execute Secure Erase command.

int setctm_flush(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the NVMe-IP
Description	Set Flush command to CTMSUBMQ_STRUCT and call exec_ctm function to execute Flush command.

int setctm_smart(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the NVMe-IP
Description	Set SMART command to CTMSUBMQ_STRUCT and call exec_ctm function to execute SMART command. Finally, decode and display SMART information on the console

void show_error(void)	
Parameters	None
Return value	None
Description	Read USRERRTYPE_INTREG, decode the error flag, and display the corresponding error message. Also, call show_pciestat function to check the hardware's debug signals.

void show_pciestat(void)	
Parameters	None
Return value	None
Description	Read PCIESTS_INTREG until the read value from two read times is stable. After that, display the read value on the console. Also, debug signals are read from NVMTESTPIN_INTREG.

void show_result(void)	
Parameters	None
Return value	None
Description	Print total size by calling get_cursize and show_size functions. After that, calculate total time usage from global parameters (timer_val and timer_upper_val) and display in usec, msec, or sec unit. Finally, compute and display the transfer performance in MB/s unit.

void show_size(unsigned long long size_input)	
Parameters	size_input: transfer size to display on the console
Return value	None
Description	Calculate and display the input value in MB or GB unit.

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

void show_smart_int8byte(volatile unsigned char *char_ptr)	
Parameters	*char_ptr: Pointer of 8-byte SMART data
Return value	None
Description	When the input value is less than 4 billion (32-bit), the 8-byte SMART data is displayed in decimal units. If the input value exceeds this limit, an overflow message is displayed.

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

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

Void shutdown_dev(void)	
Parameters	None
Return value	None
Description	Execute Shutdown command as outlined in section 3.1.6 (Shutdown Command).

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

4 Example Test Result

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

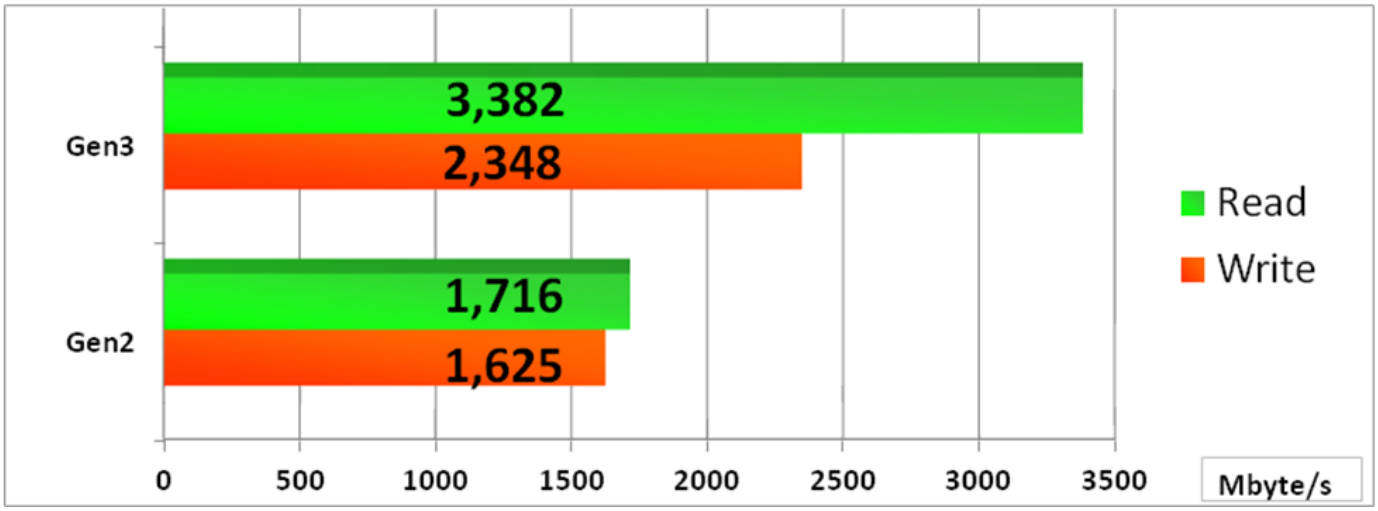

Figure 4-1 Test Performance of NVMe-IP demo by using Samsung 970 Pro SSD

When using the Arria10 GX board with PCIe Gen3, the system achieves a write performance of approximately 2300 MB/s and a read performance of around 3300 MB/s. However, using PCIe Gen2 on the ArriaV GX board results in slower performance compared to Gen3. On Gen2, the write and read performance is around 1600-1700 MB/sec.

5 Revision History

Revision	Date	Description
3.5	12-Dec-23	Add Secure Erase feature
3.4	25-Oct-23	Update Register map, CPU firmware, and Function list.
3.3	25-Mar-20	Add Function list in Test firmware
3.2	11-Feb-20	Update description
3.1	23-Nov-18	Add dword enable for RAM interface
3.0	24-Jul-18	Support Shutdown, SMART and Flush command
2.1	31-Jul-17	Add LFSR pattern
2.0	8-Jun-17	Support only 256 Kbyte buffer
1.2	9-May-17	- Use Avalon-ST PCIe Hard IP instead of Avalon-MM - Add Example test result
1.1	16-Dec-16	Change buffer from DDR to Block Memory
1.0	5-Aug-16	Initial Release