rmNVMe-IP for Gen4 reference design manual

Rev1.0 28-Jun-23

1 Overview

Design Gateway's NVMe IP Core Series offers IP solutions for accessing an NVMe SSD without the need for a CPU or external memory integration. Each IP core within the series comes with distinct key features to match specific user applications.

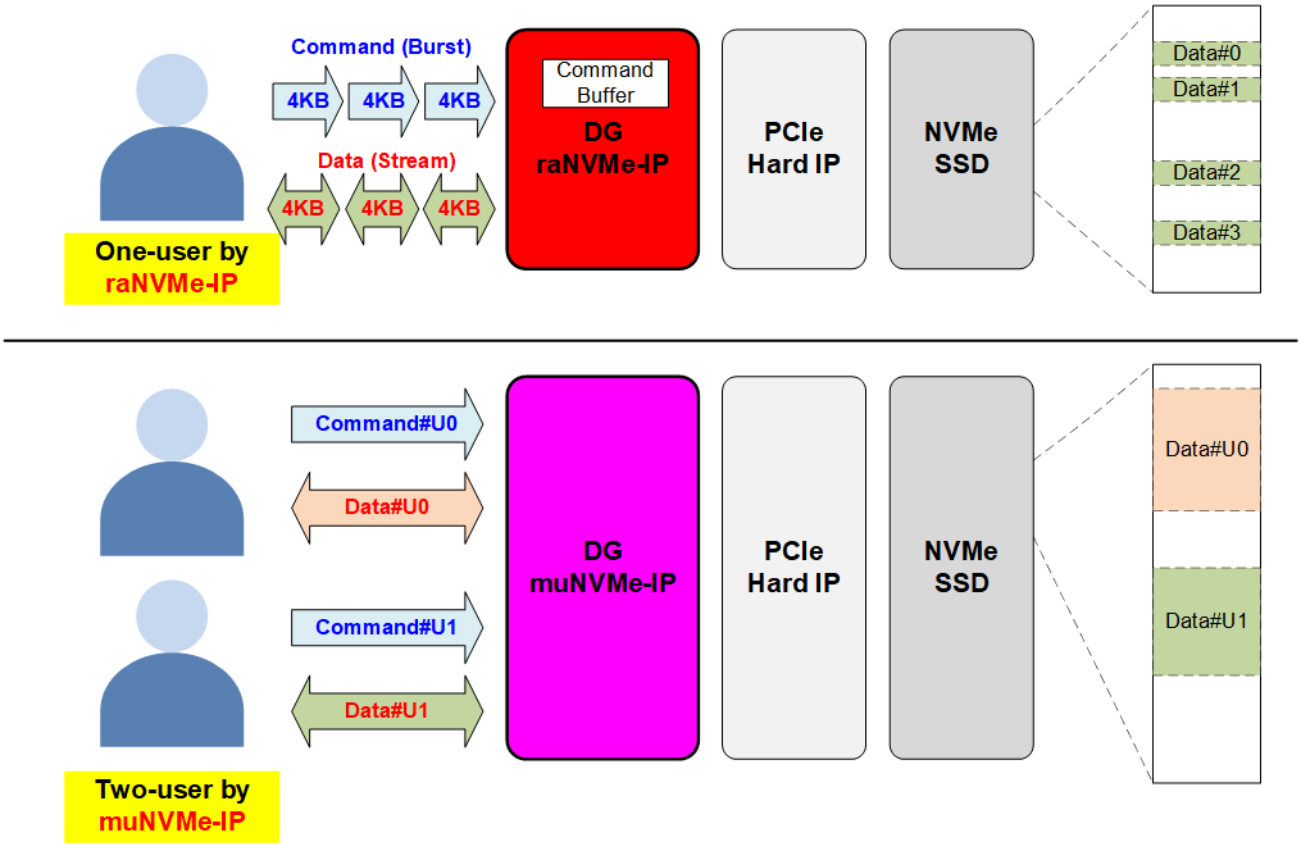


Figure 1-1 raNVMe-IP and muNVMe-IP features

raNVMe-IP is a specialized IP designed for supporting random access. It includes an internal Command Buffer, which can store up to 256 Write or Read commands with a data size of 4 Kbytes each. This feature enables users to send multiple Write or Read requests without waiting for the completion of the ongoing operation. However, it should be noted that the commands stored in the Command Buffer must be the same type (write or read), and mixed Write-Read operations are not supported.

As depicted in Figure 1-1, the data stored in the SSD using the raNVMe-IP can be stored at random address as each command has 4Kbyte data size. This makes the raNVMe-IP well-suited for applications that involve storing multiple small-sized data types at different locations within the same SSD. However, it is important to note that this IP Core offers a single user interface, requiring it to become idle before switching between command types (transitioning from Write to Read or vice versa).

Ref: raNVMe-IP for Gen3 reference design document
https://dgway.com/products/IP/NVMe-IP/dg_ranvmeip_refdesign_intel/

Design Gateway introduces the muNVMe-IP, which is specifically designed to support multi-user systems. As illustrated in Figure 1-1, two users can connect to the muNVMe-IP simultaneously and send commands to access the same SSD. The commands can be the same or different types, and the transfer size for each command can be configured to a large value, ensuring high performance. This feature makes the muNVMe-IP ideal for applications that require storing data in contiguous areas or sequential access.

Ref: muNVMe-IP for Gen4 reference design document
https://dgway.com/products/IP/NVMe-IP/dg_munvmeip_refdesign_g4_intel/

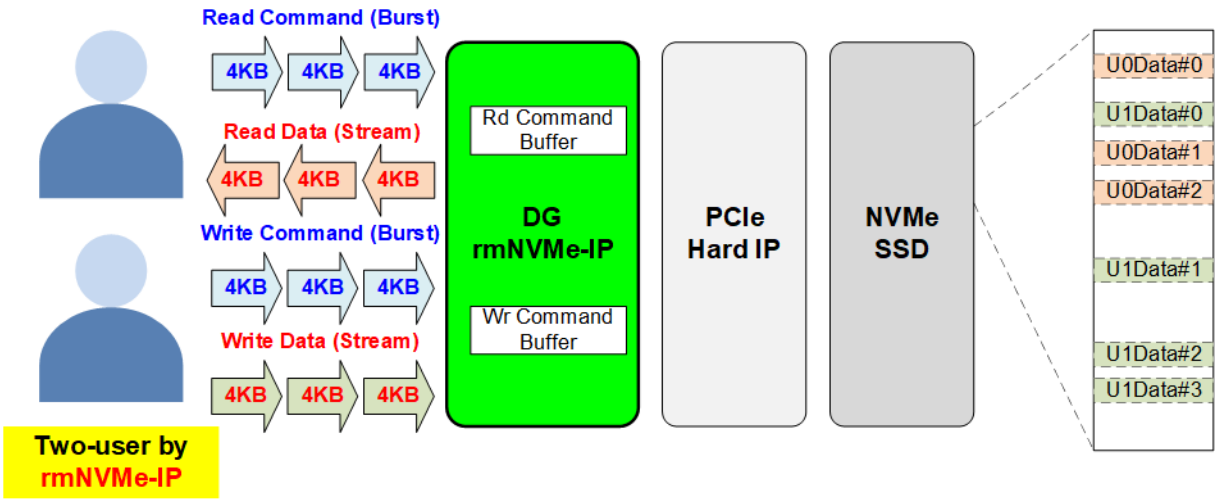


Figure 1-2 rmNVMe-IP features

The new solution offered by Design Gateway is the rmNVMe-IP (random access multiple-user NVMe-IP), which combines the key features of both raNVMe-IP and muNVMe-IP. The rmNVMe-IP provides random access capabilities and multiple-user interfaces for writing and reading from an NVMe SSD simultaneously. It includes two user interfaces, for writing and reading separately. Each interface supports up to 256 command requests with 4Kbyte data size per command.

2 Hardware overview

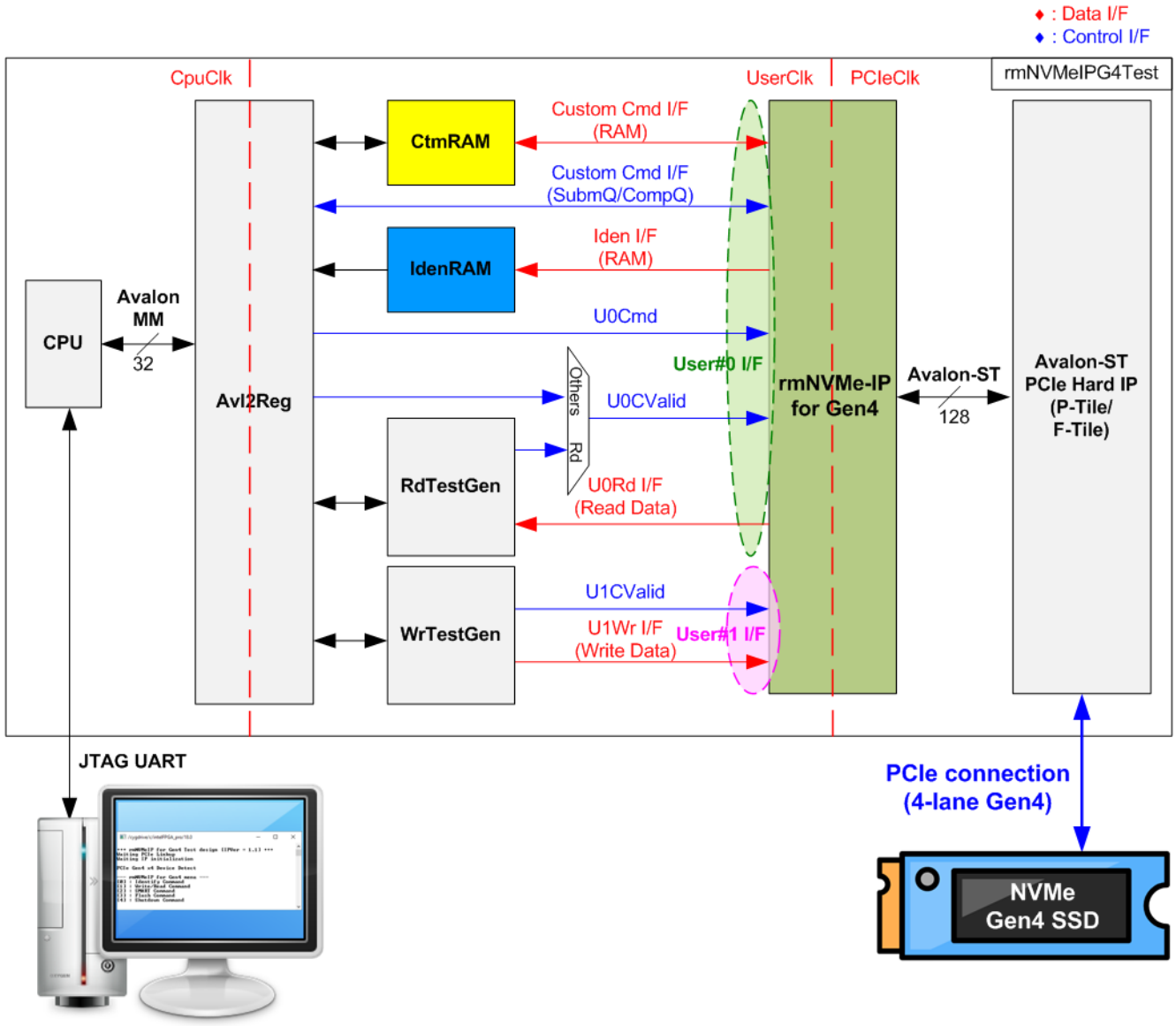


Figure 2-1 rmNVMe-IP for Gen4 demo hardware

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

The RdTestGen connects to the User#0 I/F of rmNVMe-IP. It is responsible for generating Read command request and verifying the Read data received from rmNVMe-IP. Conversely, the WrTestGen connects to the User#1 I/F of rmNVMe-IP and is responsible for generating Write command request and sending Write data to rmNVMe-IP.

The NVMe function consists of the rmNVMe-IP and the PCIe hard IP (P-Tile/F-Tile Avalon-ST Intel FPGA for PCIe), which allows direct access to an NVMe SSD without a PCIe switch. The generation of command requests to the User#0 I/F of the rmNVMe-IP (U0CValid) depends on the type of command. Single mode commands (Identify, Shutdown, Flush, or SMART) are generated by the CPU through the Avl2Reg module. On the other hand, multi-mode command requests (Read), are generated by WrTestGen. The data interface for both Custom and Identify commands is connected to RAMs that are accessible by the CPU.

The CPU connects to the Avl2Reg module to interface with the NVMe test logics. Integrating the CPU into the test system allows users to set test parameters and monitor the test status via JTAG UART. The CPU also facilitates the execution of multiple test cases to verify the functionality of the IP. The default firmware for the CPU includes functions for executing NVMe commands using rmNVMe-IP.

Figure 2-1 shows three clock domains: CpuClk, UserClk, and PCIeClk. CpuClk is the clock domain for the CPU and its peripherals, and it must be a stable clock independent from other hardware. The UserClk is the user clock domain utilized for the operation of the rmNVMe-IP, RAM, and TestGen. As specified in the rmNVMe-IP datasheet, the clock frequency of UserClk must be greater than or equal to a half of the PCIeClk frequency. Finally, the PCIeClk is the clock output generated by the PCIe hard IP to synchronize with the data stream of the Avalon-ST interface. In this reference design, the interface is configured as four of 4-lane PCIe Gen4 at 500 MHz. Therefore, PCIeClk is set to 500 MHz while UserClk is set to 275 MHz (> 250 MHz).

Further details about the hardware are described below.

2.1 WrTestGen

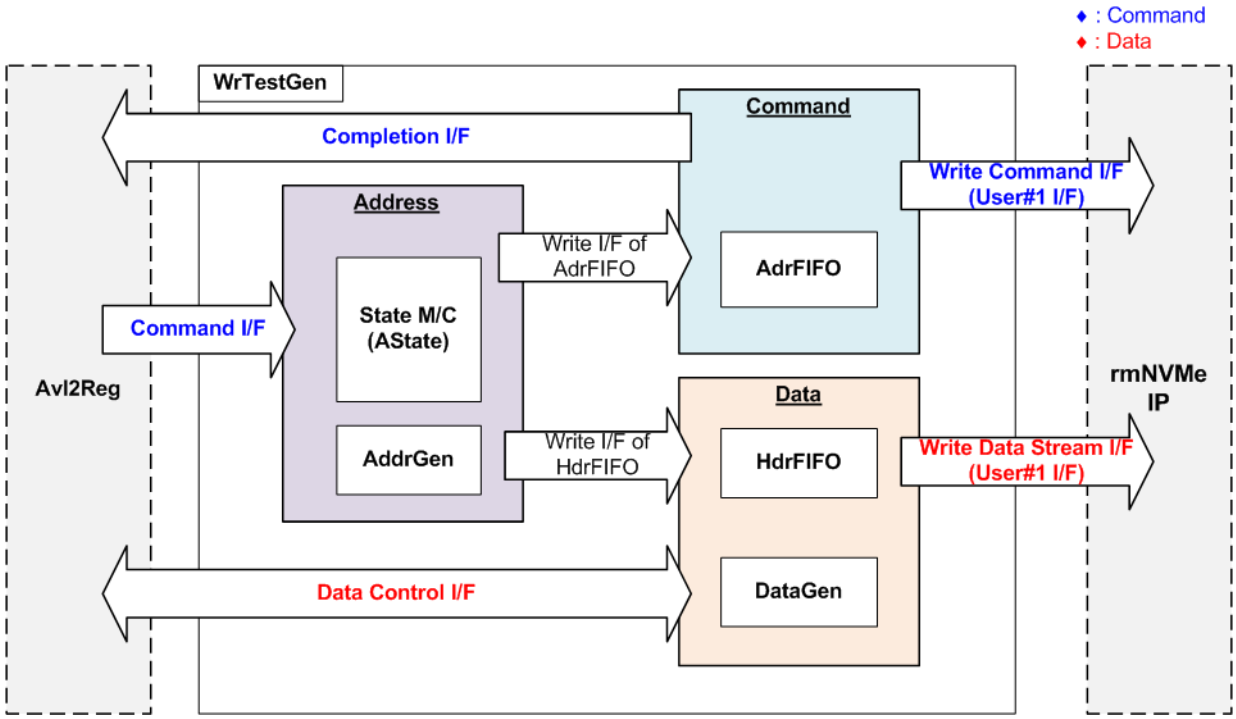


Figure 2-2 WrTestGen Block diagram

The WrTestGen module connects to rmNVMe-IP for generating Write command request and sending Write data stream via User#1 I/F. In Figure 2-2, the logic inside WrTestGen is divided into three groups – Address, Command, and Data. The Address block generates address values for multiple Write command requests, which are then transferred to rmNVMe-IP through Write Command I/F, handle by the Command block. Concurrently, the data block prepares the 4KB Write data of each Write command and transfers it to rmNVMe-IP via the Write Data Stream I/F. These three logic groups operate in parallel, so the Command and Data blocks include AdrFIFO and HdrFIFO to store the pre-generated addresses from the Address block.

The Address block receives the start address from user, generates address for each command, and stores it in the Command module (AdrFIFO) and Data module (HdrFIFO). Only 45-bit Address (bit[47:3]) is stored in both FIFOs because bit[2:0] is always set to 000b to align with 4Kbyte units. The generated addresses can be either sequential or random, based on the user's parameter (TrnMode). A state machine (AState) has been implemented to manage the operation flow and control the complexity of the Address block.

The Command block sends a Write command to rmNVMe-IP by setting CmdValid to 1b, along with CmdAddr (the output of AdrFIFO). If the internal Command buffer of rmNVMe-IP is available, CmdReady is asserted to 1b to accept the request. If rmNVMe-IP is unable to receive additional commands, CmdReady will become de-asserted.

Lastly, the Data block generates 4Kbyte Write data for each command by utilizing the address from the HdrFIFO to construct the header of each 4Kbyte Write data. The remaining 4Kbyte Write data is produced by the DataGen component within Data block. Additionally, the Write data transfer rate can be adjusted through the user's parameter (TrnRate), which affects the assertion and de-assertion of WrValid (the data flow control signals of rmNVMe-IP).

More details of each logic group in WrTestGen are described as follows.

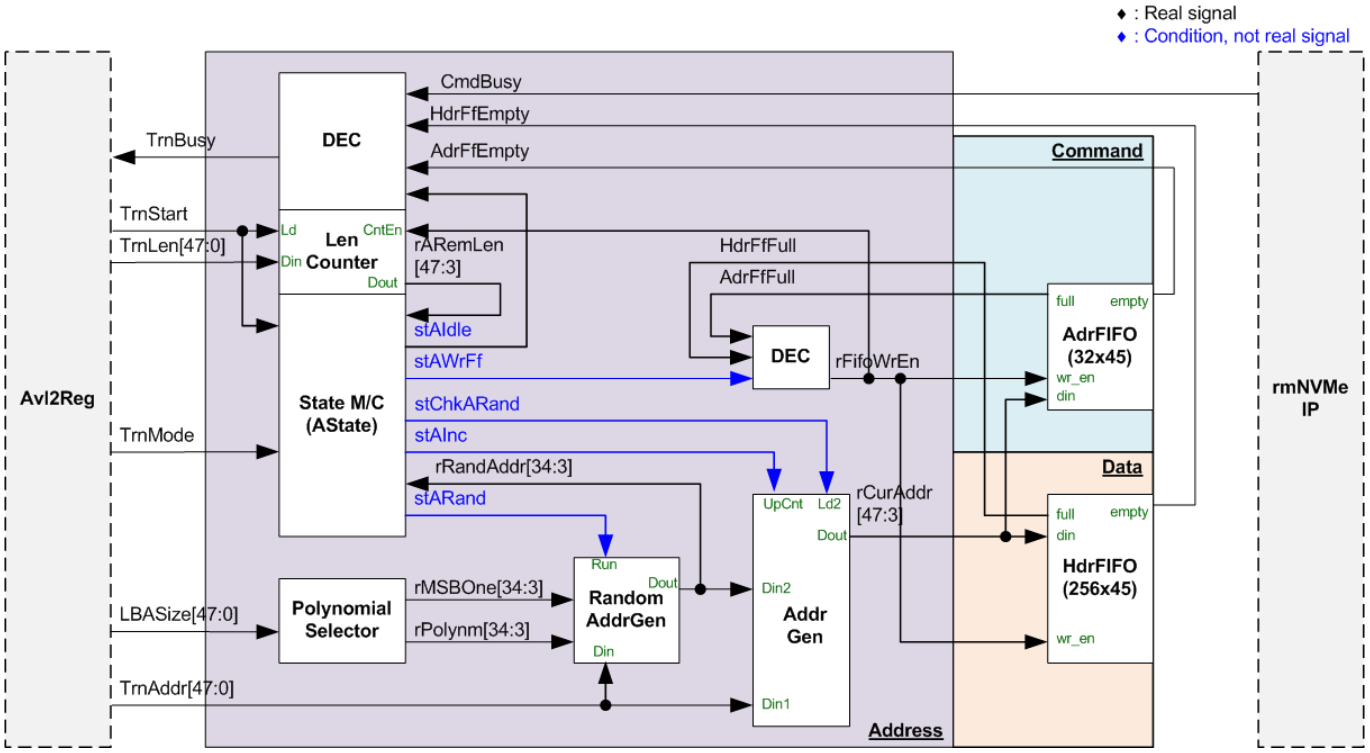


Figure 2-3 Logic diagram of Address block

Address

When user asserts TrnStart pulse for Write command, four parameters are loaded from Avl2Reg, i.e., the start address in 512-byte units (TrnAddr), the amount of transferred data in 512-byte units (TrnLen), the address mode (TrnMode: Random or Sequential), and the total disk capacity (LBASize). The operation of Address block is managed by a state machine, which will be discussed in further detail.

(1) stAIdle: This state is designed to wait for TrnStart pulse asserted when the new Write command is requested. The internal logic is initialized by the input parameters. Len Counter loads total transfer size from TrnLen, while the AddrGen and Random AddrGen load the initial address from TrnAddr (bit[2:0] are ignored for 4Kbyte unit address). After that, it transits to the next state: stAWrFf.

(2) stAWrFf: This state checks the FIFO full status. If both AdrFIFO and HdrFIFO are not full, the new address, rCurAddr (output from AddrGen), is written to both FIFOs by asserting rFifoWrEn to 1b. After that, the next state is determined based on the following conditions.
   a. If the current address is the final address (indicated by the remaining transfer length or rRemALen being equal to 1), the state returns to stAIdle (1).
   b. If the current address is not the final address, the next state depends on the address mode (TrnMode). For sequential access (TrnMode=0b), it transits to stAInc, while for random access (TrnMode=1b), it transits to stARand1.

(3) stAInc: This 1-clock state generates the next address for sequential access by up-counting rCurAddr value using AddrGen. Afterward, the state returns to stAWrFf(2).

(4) stARand: The 1-clock state is designed to generate the next address for random access. It utilizes the Polynomial Selector and Random AddrGen in this process, and the output value, rRandAddr, will be validated in the subsequent state, stChkARand.

(5) stChkARand: In this state, the random address (rRandAddr) is verified. If rRandAddr value exceeds the disk capacity (LBASize – 1), the state returns to stARand to re-generate the new address. Conversely, if the value is within an acceptable range, the state transits to stAWrFf to store the address result to both AdrFIFO and HdrFIFO.

The busy flag of the WrTestGen (TrnBusy) is de-asserted to 0b when all subblock operations have been finished. This occurs when the state within the Address block returns to stIdle, both the AdrFIFO and HdrFIFO are empty, and the CmdBusy of rmNVMe-IP is de-asserted. The Len Counter is a counter that decrements to indicate the number of remaining addresses for the 4KB command request to be generated. The output signal, rARemLen, is fed to the state machine to verify that the final address is completely generated. The key function of the Address block is how to generate sequential address or random address based on TrnMode setting. This function is performed by three components, i.e., Polynomial Selector, Random AddrGen, and AddrGen.

The random value is generated through Galois LFSR, which uses XNOR operation to overcome the issue of a start value being set to all zeros. Bit rotation is performed using a right-shift operation. The degree of polynomial is determined by the disk capacity to minimize the chance of the output value exceeding it. We apply LFSR-4, as mentioned in the following website, is applied.
https://web.archive.org/web/20161007061934/http://courses.cse.tamu.edu/csce680/walker/lfsr_table.pdf

Table 2-1 Implemented LFSR polynomial and pre-generated parameters

Disk size	Polynomial	rMSBOne (hex)	rPolynm (hex)
≥8TB	$x^{31}+x^{29}+x^{25}+x^{24}$	8000 0000h	A300_0000h
8TB> and >4TB	$x^{30}+x^{29}+x^{28}+x^{27}$	4000 0000h	7800_0000h
4TB> and >2TB	$x^{29}+x^{28}+x^{25}+x^{23}$	2000 0000h	3280_0000h
2TB> and >1TB	$x^{28}+x^{27}+x^{26}+x^{24}$	1000 0000h	1D00_0000h
1TB> and >512GB	$x^{27}+x^{26}+x^{23}+x^{21}$	800 0000h	0CA0_0000h
512GB> and >256GB	$x^{26}+x^{25}+x^{24}+x^{21}$	400 0000h	0720_0000h
≤256GB	$x^{25}+x^{24}+x^{23}+x^{19}$	200 0000h	0388_0000h

To support multiple polynomials, two constant values, as listed in Table 2-1, are pre-generated by the Polynomial Selector. These values are inputted into the Random AddrGen to perform OR and XOR operations with the current address value to generate the next address value. The process for generating a random address is as follows.

(1) Read the LSB of the current address value for the next operation.
(2) Right-shift the current address value and set the MSB to 1 by using OR with rMSBOne.
(3) If LSB result from step (1) is 1b, the next address is equal to the result from step (2). If LSB is 0b, the next address is determined by XOR-ing the result from step (2) with rPolynm.
(4) The next address (rRanAddr) is inputted to the State machine to check if it exceeds the disk capacity. If it is within limit, it is fed into AddrGen for the next step. In case the result exceeds the disk capacity, the random value is re-generated.

The AddrGen determines the next address value (rCurAddr), either as a random value (rRanAddr) or as an incremental value (rCurAddr + 1). The address result is finally written into AdrFIFO and HdrFIFO.

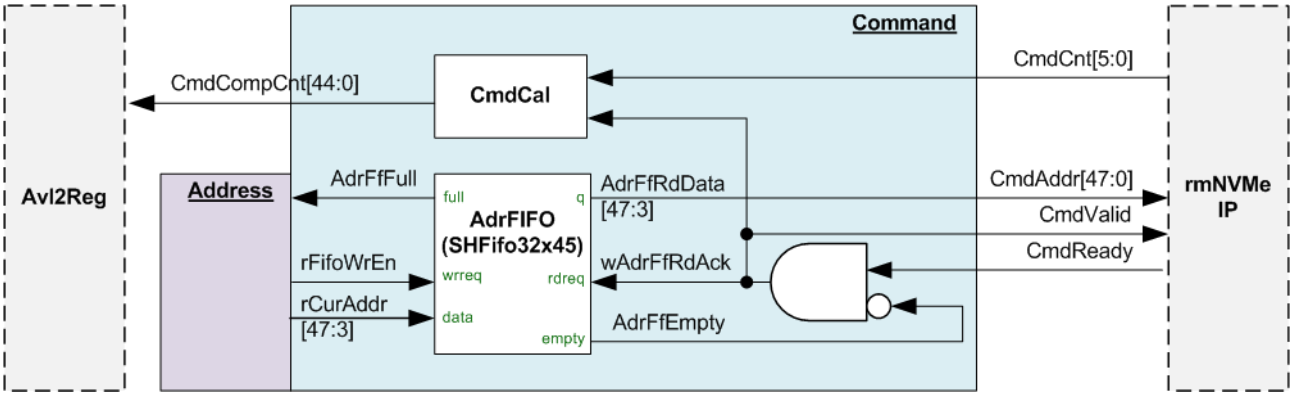


Figure 2-4 Logic diagram of Command block

Command

The AdrFIFO is read when the FIFO is not empty (AdrFfEmpty=0b) and the rmNVMe-IP is ready to receive new command (CmdReady=1b). The AdrFIFO is Show-ahead type, providing the read data (CmdAddr) at the same time as the read enable signal (wAdrFfRdAck) being asserted. As a result, the CmdValid can be derived from wAdrFfRdAck.

To display the write performance in IOPs, a logic has been designed to count the total number of completed commands. The CmdCal block calculates the CmdCompCnt, which represents the total number of commands sent to rmNVMe-IP (counted by the wAdrFfRdAck signal) minus the number of incomplete commands at the rmNVMe-IP (CmdCnt).

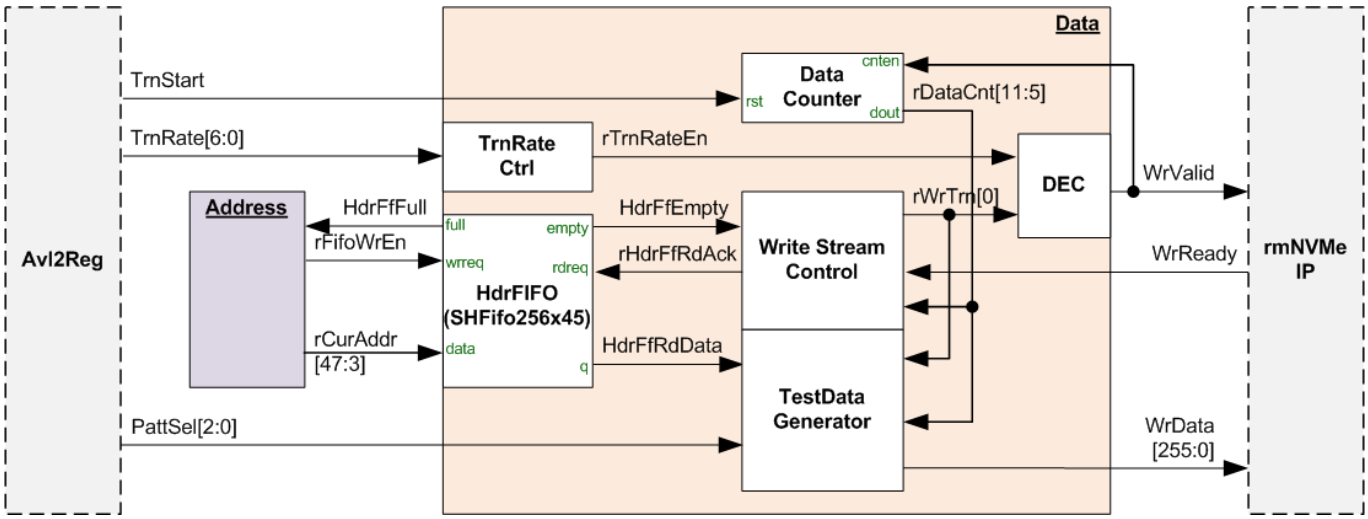


Figure 2-5 Logic diagram of Data block

Data

The Write Stream Control manages the core functionality of the Data block, responsible for transferring 4KB Write data to the rmNVMe-IP. The header for each 4KB data segment is read from the HdrFIFO, while the rest of the data is generated by the TestData Generator. The maximum data rate for the data stream can be configured using the TrnRate signal, with the TrnRateCtrl managing the duty cycle to set the rTrnRateEn signal to either 1b or 0b, thereby limiting the maximum Write data rate. The WrValid signal can only be set to 1b when the rTrnRateEn is set to 1b. The Data Counter is designed to check the total amount of Write data that has been transferred to the rmNVMe-IP.

The Write Stream Control initiates the transfer of 4KB Write data by waiting until the data is available in the HdrFIFO (HdrFfEmpty=0b) and rmNVMe-IP has space to receive new data (WrReady=1b). Once the conditions are met, the rWrTrn is set to 1b to initiate the transfer. For every 4KB data transfer, the rHdrFfRdAck signal is set to 1b to read a single data (HdrFfRdData) from the HdrFIFO, which is then passed to the TestData Generator.

The WrValid signal remains asserted for 128 clock cycles to transfer 4 KB data. However, the signal may not be asserted continuously, depending on the TrnRate parameter which determines the transfer rate of the Data Stream I/F. The RateCal logic asserts the rTrnRateEn to 1b for the specified number of clock cycles based on “TrnRate” clock cycles, and then de-asserts it to 0b for the remaining cycles in every 100 clock cycles.

A 7-bit counter (rDataCnt) is applied to track the completion of each 4KB data transfer and to create the Write data. After finishing transferring each 4KB data segment, a pause of two clock cycles is introduced to allow time for WrReady to be updated from the pipeline processing.

The TestData Generator generates the test data (WrData) to be sent to rmNVMe-IP in the Write command. Each 4 KB data segment consists of a 64-bit header data and the test pattern, selected by the PattSel parameter.

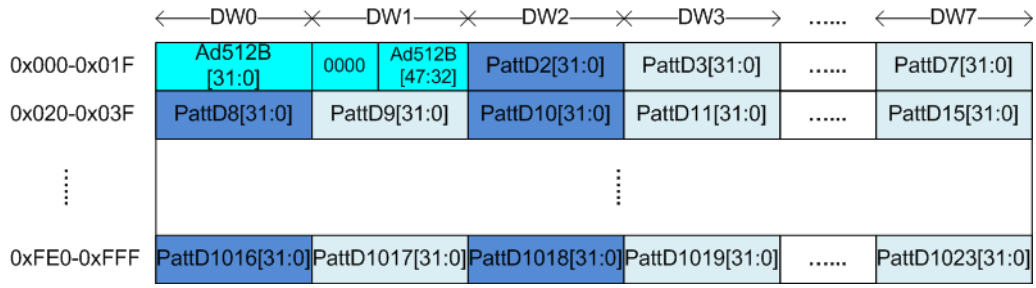


Figure 2-6 Test pattern format of 4096-byte data for Increment/Decrement/LFSR pattern

Figure 2-6 demonstrates the generation of the 64-bit header in DW#0 and DW#1 by combining a 48-bit address read from HdrFIFO with a zero value. The remaining data (DW#2 – DW#1023) represents the test pattern, which can be selected from three formats: 32-bit incremental data, 32-bit decremental data, and 32-bit LFSR counter. The 32-bit incremental data is obtained from the Data Counter. The decremental data is derived by applying the NOT logic to the incremental data. The LFSR data is generated using a Fibonacci LFSR with the equation is $x^{31} + x^{21} + x + 1$.

To implement the 256-bit LFSR pattern, the data is divided into two sets of 128-bit data, each with a different initial value. The 128-bit data uses a look-ahead technique to calculate four 32-bit LFSR data in one clock cycle. Figure 2-7 illustrates that the initial value of LFSR is obtained by combining a part of 32 lower bits of the address (LBAAddr) with the NOT logic of the 32 lower bits of the LBA address (LBAAddrB).

For both all zero and all one patterns, a 64-bit header is not inserted into the 4KB data. These patterns show the best Write/Read performance of certain SSDs.

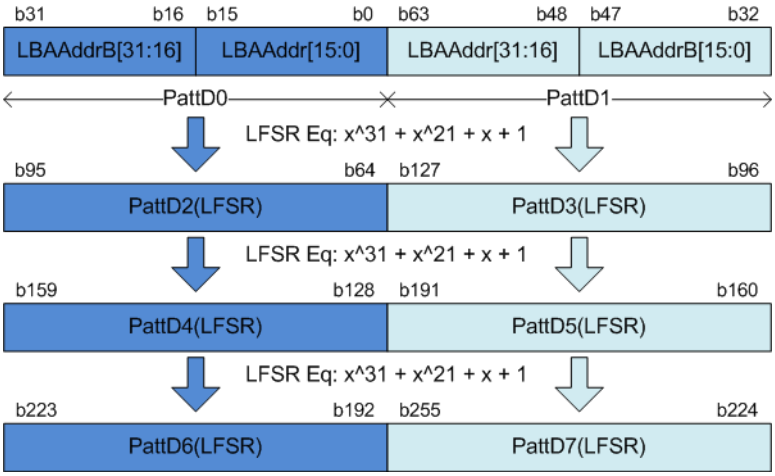
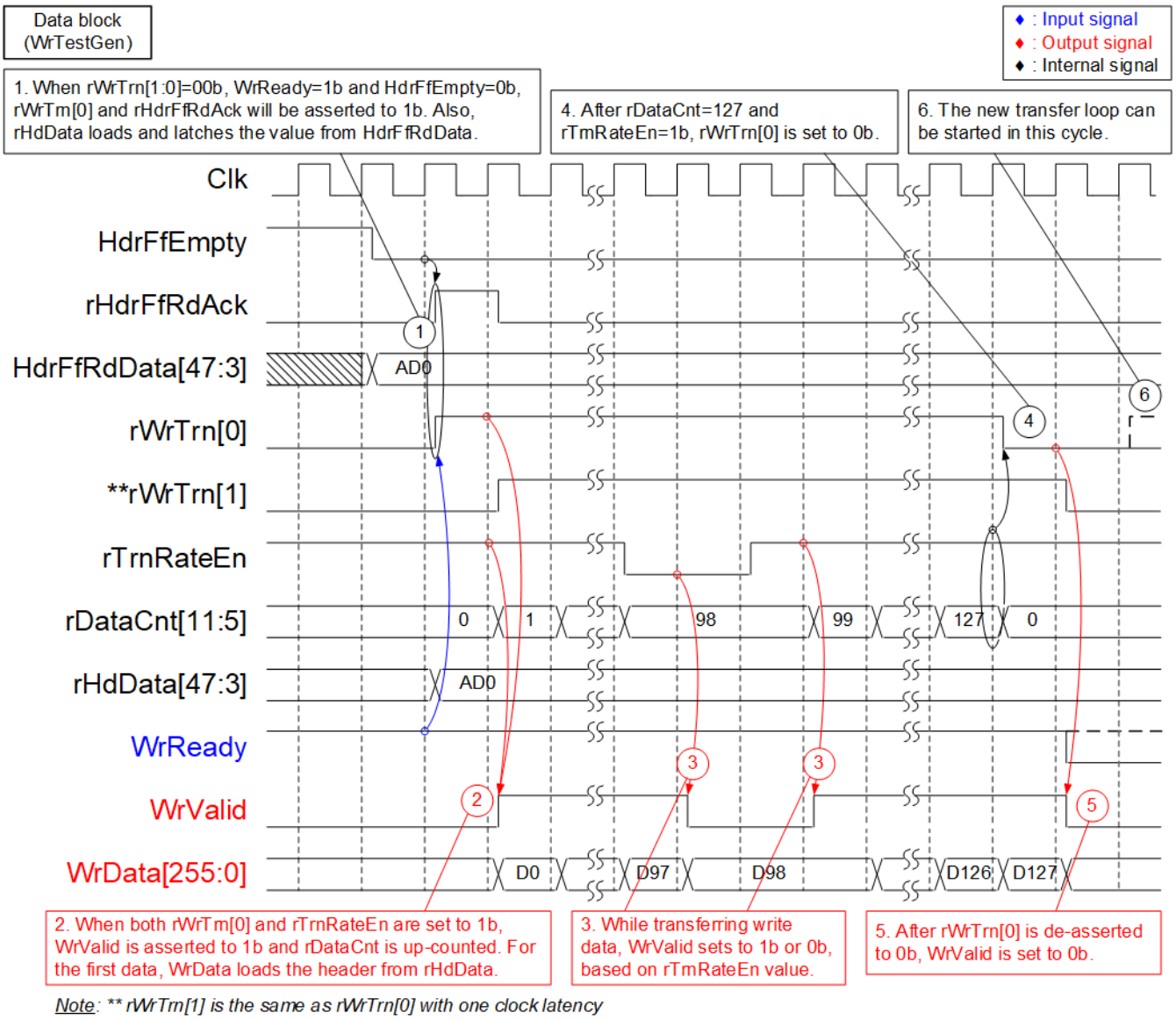


Figure 2-7 256-bit LFSR Pattern in TestGen

Timing diagram to show the operation of Data block inside WrTestGen is shown in Figure 2-8.



*Note: ** rWrTrn[1] is the same as rWrTrn[0] with one clock latency*

Figure 2-8 Timing diagram to show Data block operation within WrTestGen module

(1) To initiate a new 4KB Write data transfer, three conditions must be satisfied.
- HdrFfEmpty=0b to ensure that a new address has been stored in HdrFIFO.
- WrReady=1b to indicate the rmNVMe-IP has a capability to receive 4KB Write data.
- rWrTrn[1:0]=00b to add the latency time for waiting until the WrReady signal is updated after finishing the previous transfer.

The new data transfer is initiated by setting rHdrFfRdAck to 1b for one clock cycle to read the HdrFfRdData and will be used as the header data for each 4KB data segment. Simultaneously, rWrTrn[0] is set to 1b to indicate the ongoing data transfer. This value remains set until finishing the current data transfer.

(2) The Write data is sent to the rmNVMe-IP using the following guidelines.
- Asserting WrValid to 1b along with valid WrData.
- Setting WrValid to 1b when both rWrTrn[0] and rTrnRateEn are asserted to 1b, indicating that the Write command is active and the Write data rate has not reached the maximum limit.
- Including 48-bit header data from rHdData in the first data of each 4KB data segment.
- Incrementing rDataCnt after transferring each Write data to indicate the amount of Write data transferred within this transfer (ranging from 0 to 127).

(3) While transferring a 4KB data segment, the value of WrValid value is controlled by rTrnRateEn signal and can only be asserted when rTrnRateEn is set to 1b.
(4) When the final data of the current data segment is transferred in the next clock cycle, indicated by rDataCnt=127 and rTrnRateEn=1b, rWrTrn[0] is set to 0b, marking the completion of the current data transfer.
(5) The final data transfer occurs by asserting WrValid to 1b along with the last data (D127) on WrData. rWrTrn[1] is set to 0b in the next clock cycle.
(6) The updated value of WrReady is provided after finishing transferring all 4KB data segment when both rWrTrn[0] and rWrTrn[1] are de-asserted to 0b. Therefore, WrValid is de-asserted to 0b at least two clock cycles before sending the next 4KB data segment.

2.2 RdTestGen

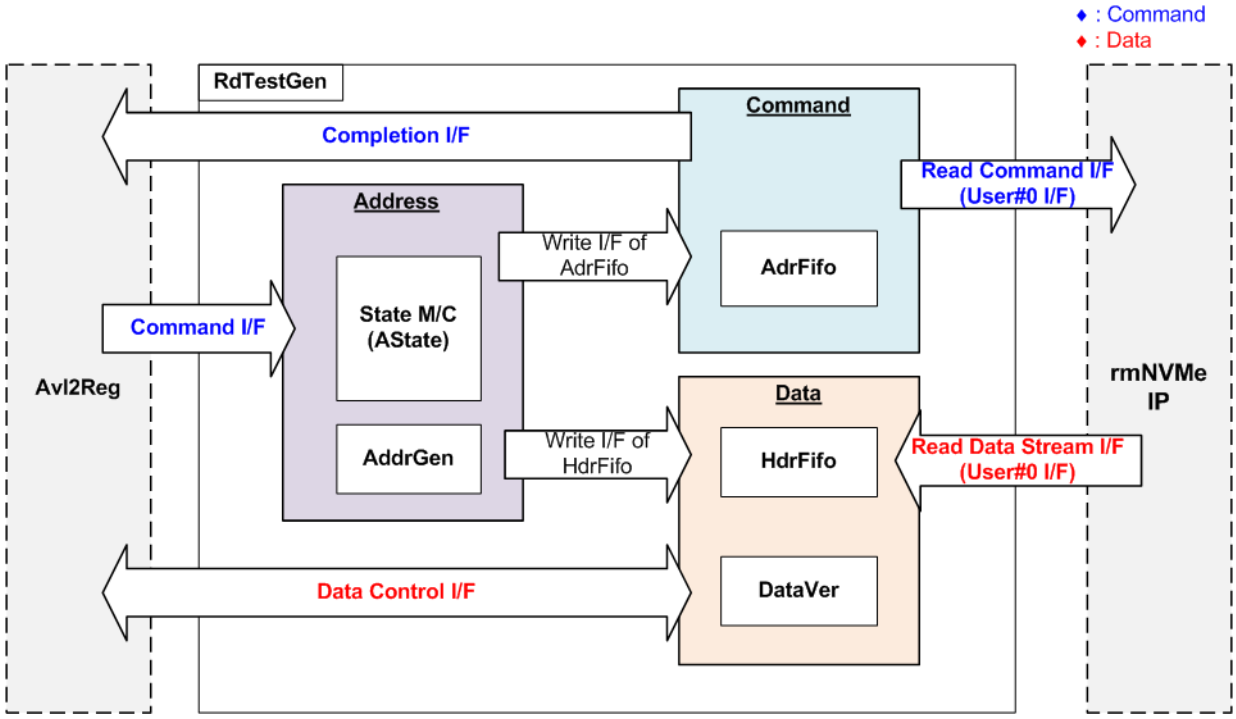


Figure 2-9 RdTestGen Block diagram

While WrTestGen connects to User#1 I/F for handling Write command operation, the RdTestGen connects to User#0 I/F for handling Read command operation. Similar to WrTestGen, the RdTestGen module consists of three logic groups: Address, Command, and Data. The function and the logic of the Address and Command groups are similar to WrTestGen. Please refer to the previous topic for more details. This topic focuses only the details of the Data block.

Note: As the User#0 I/F supports multiple command types, the U0Cmd which specifies the command type is directly set by Avl2Reg. The Command block in the RdTestGen module sends only the command request for Read commands.

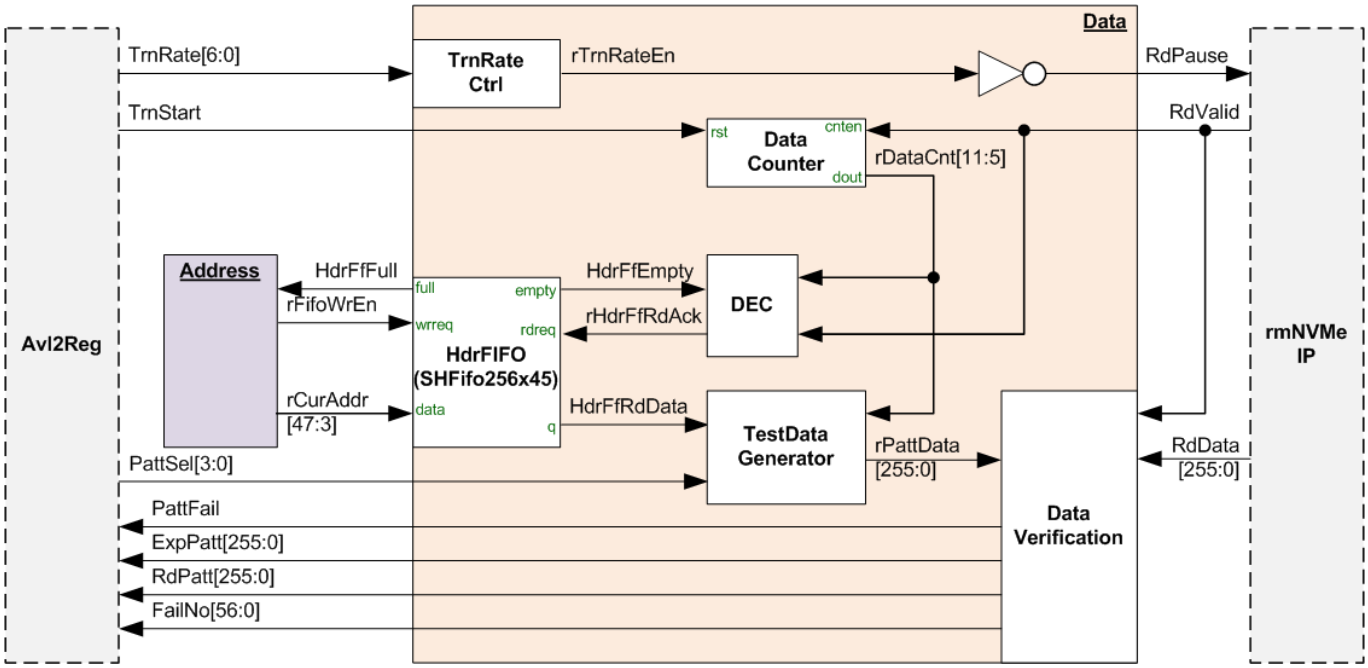


Figure 2-10 Logic diagram of Data block

Data

The maximum data transfer rate for Read data is controlled by the RdPause signal. According to the rmNVMe-IP datasheet, setting RdPause to 1b will cause RdValid to be de-asserted to 0b within 4 clock cycles. RdPause can be designed using the same logic as rTrnRateEn in WrTestGen, by adding NOT logic.

The RdValid signal, an input of rmNVMe-IP, controls the Data Counter that shows the total amount of Read data sent. When the first data of each 4KB data segment is received, Decoder logic sets rHdrFfRdAck to 1b for one clock cycle. The header data, HdrFfRdData, is input to TestData Generator to generate the expected data, rPattData. The logic of TestData Generator is almost similar to that of WrTestGen. The Data Verification module compares the RdData with the rPattData. If an error is detected, PattFail is set to 1b along with the expected value (ExpPatt), received value (RdPatt), and failure position (FailNo).

Timing diagram to show the operation of Data block within RdTestGen is shown in Figure 2-11.

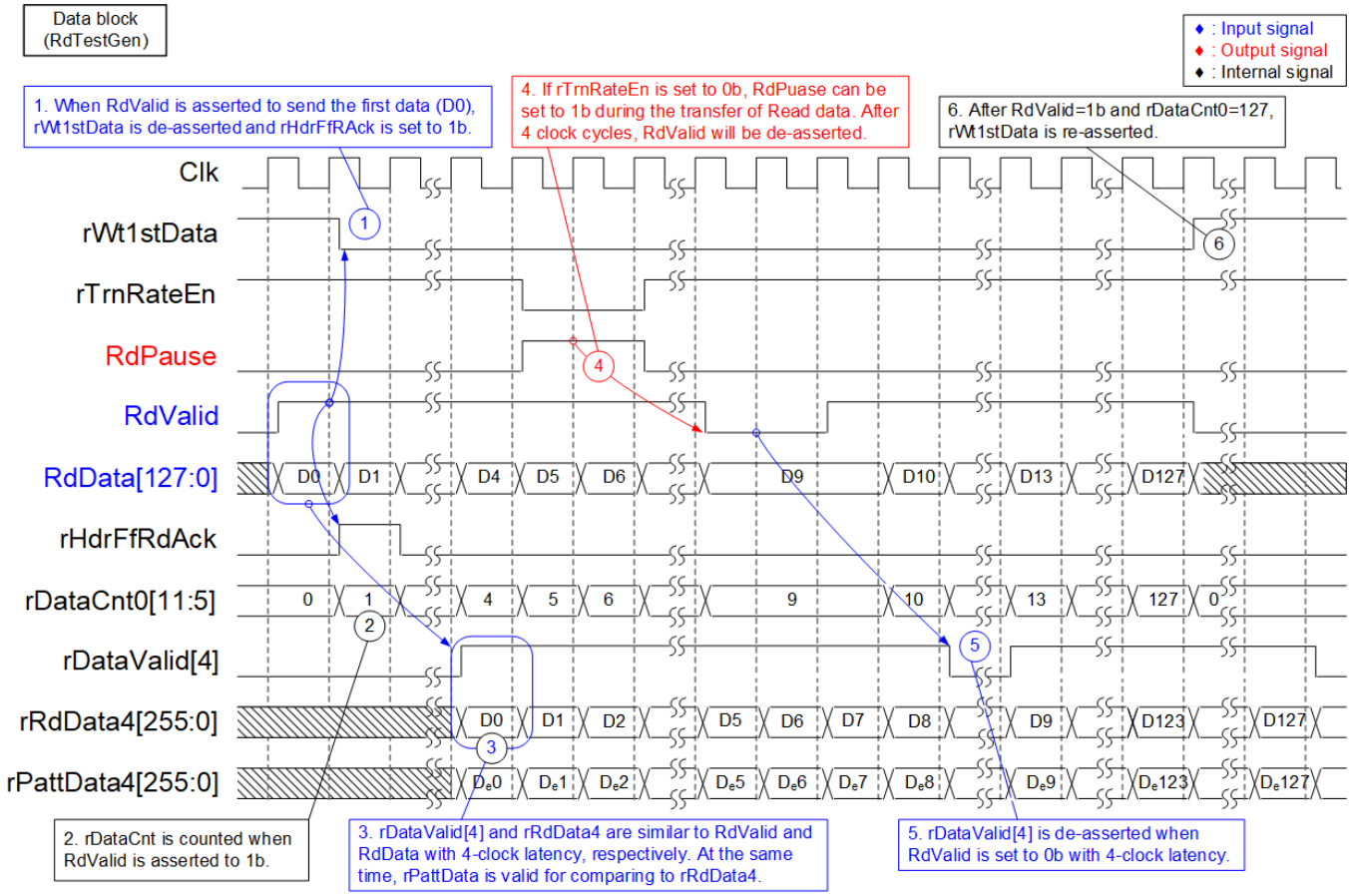


Figure 2-11 Timing diagram to show Data block operation within RdTestGen module

(1) When the Command block generates a Read command request to rmNVMe-IP, the Data block receives the 4KB Read data. The data is sent via RdData, and RdValid is set to 1b. If RdValid=1b and rWt1stData=1b, it indicates that the first data (D0) is being sent. At this point, rWt1stData signal is de-asserted, and rHdrFfRdAck is asserted. The output data of HdrFIFO (HdrFfRdData) is input to the TestData Generator to create the expected value (D_e0), similar to the process used in WrTestGen.

(2) While receiving each Read data (RdValid=1b), rDataCnt is incremented to track the total amount of Read data in each 4KB segment. Its value ranges from 0 to 127.

(3) The TestData Generator requires 4 clock cycles to process and generate the expected value for the first data (D_e0) after receiving it on the RdData. D_e0 includes the 48-bit address (HdrFfRdData), which is valid when rHdrFfRdAck=1b

(4) After the TestData Generator completes generating the expected value of the first data (D_e0), which includes the 48-bit address (HdrFfRdData) that is valid when rHdrFfRdAck=1b, several Flip-Flops are added to synchronize RdData signal with rPattData4. The expected value (D_e0 – D_e127) are then compared to the Read data (D0 – D127) by the Data Verification module. If rRdData4 ≠ rPattData4, PattFail is asserted.

(5) To control the data transfer rate, rTrnRateEn can be set to 0b to assert RdPause to 1b. After four clock cycles, RdValid will be de-asserted to 0b to pause data transmission.

(6) Data Verification also pauses the operation when rDataValid [4] (the RdValid signal with 4-clock latency) is de-asserted.

(7) When the last data of each 4KB block is received (DataCnt0=127 and RdValid=1b), rWt1stData is re-asserted to 1b to scan the first data of the next 4KB data segment.

2.3 NVMe

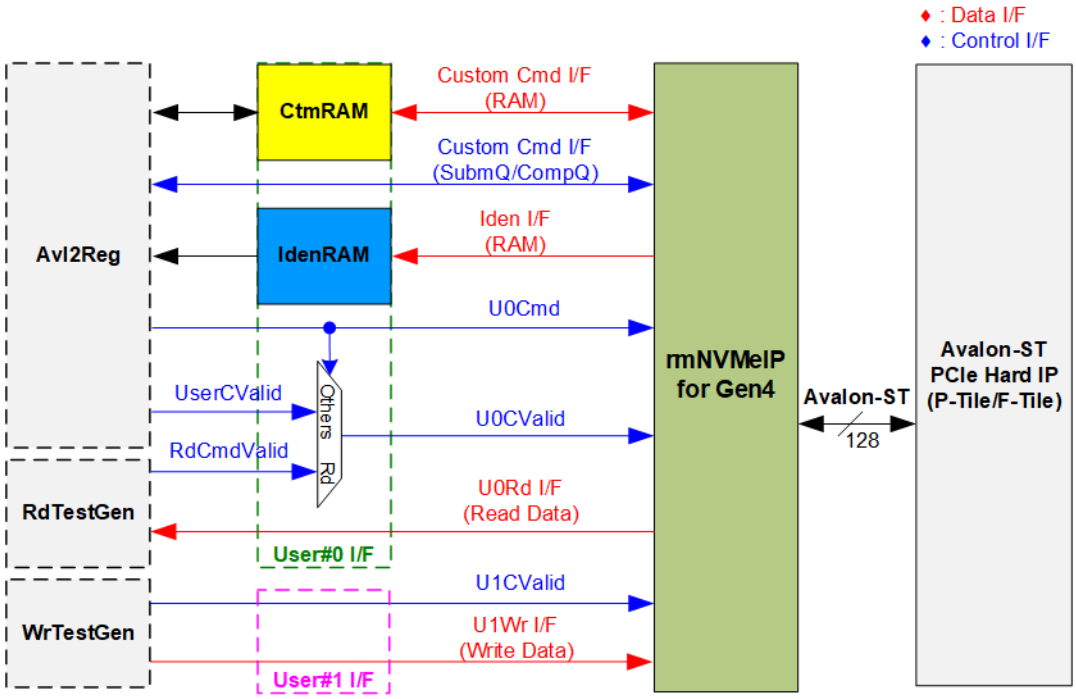


Figure 2-12 NVMe hardware

Figure 2-12 shows how to integrate rmNVMe-IP into the reference design. Each rmNVMe-IP user interface consists of a Control interface and a Data interface. The Control interface receives commands and parameters from the user, while the Data interface transfers data when a command requires data transfer.

There are two types of commands – Single mode and Multi-mode. For User#0 I/F, the command value (U0Cmd) is set by CPU firmware via Avl2Reg, whereas the command request (U0CValid) is controlled by two sources – UserCValid and RdCmdValid. In Single mode, the command request (UserCValid) is generated by the CPU firmware. On the other hand, in Multi-mode, the command request (RdCmdValid) is generated by RdTestGen. SMART and Flush commands are Custom commands that require additional parameters set via the Custom Cmd I/F, which are also set by the CPU firmware via Avl2Reg module. For User#1 I/F, only Write command, a Multi-mode command, is requested and the command request (U1CValid) to the IP is generated by WrTestGen.

There are four commands that involve data transfer via specific interfaces.

- Custom Cmd I/F (RAM) transfers SMART data to CtmRAM in the SMART command.
- Iden I/F (RAM) transfers Identify data to IdenRAM in the Identify command.
- U0Rd I/F transfers Read data from rmNVMe-IP in the Read command.
- U1Wr I/F transfers Write data from WrTestGen in the Write command.

Although each command uses a different interface for data transfer, all data interfaces have the same data bus size of 256 bits.

2.3.1 rmNVMe-IP

rmNVMe-IP implements NVMe protocol on the host side, enabling direct access to an NVMe SSD without the need for a PCIe switch connection. It supports two users, with the Main user having access for five commands (Read, Identify, Shutdown, SMART, and Flush), and the Sub user able to access only the Write command. rmNVMe-IP can handle up to 256 Read commands and 256 Write commands with random addressing, allowing to execute the additional commands without the need to wait for the previous command completion. Additionally, rmNVMe-IP can be directly connected to the PCIe hard IP. For more detailed information about rmNVMe-IP, please refer to the datasheet available at the following link.

https://dgway.com/products/IP/NVMe-IP/dg_rmnvme_ip_datasheet_g4_intel/

2.3.2 PCIe Hard IP (P-Tile/F-Tile Avalon-ST Intel FPGA for PCIe)

This block is the hard IP integrated into Intel FPGA devices, which implements Physical, Data Link, and Transaction Layers of the PCIe protocol. Further details can be found from Intel FPGA website.

P-Tile Avalon-ST Intel FPGA for PCIe
https://www.intel.com/content/www/us/en/docs/programmable/683059/
F-Tile Avalon-ST Intel FPGA for PCIe
https://www.intel.com/content/www/us/en/docs/programmable/683140/

2.3.3 Two-port RAM

The system includes two of two-Port RAMs, CtmRAM and IdenRAM, which store data from Identify command and SMART command, respectively. IdenRAM is a simple dual-port RAM with one read port and one write port. The data size of Identify command is 8 Kbytes, so the size of IdenRAM is 8 Kbytes. rmNVMe-IP and Avl2Reg have different data bus sizes, 256-bit data bus for rmNVMe-IP and 32-bit data bus for Avl2Reg. Therefore, IdenRAM has a different bus size for its Write interface and Read interface. Besides, rmNVMe-IP features a double word enable that allows for writing 32-bit data in specific cases. To accommodate this, the RAM settings in the IP catalog of Quartus tool support write byte enable. As shown in Figure 2-13, each bit (bit[0], [1], …, [7]) of double word enable (WrDWEn) is mapped to 4-bit write byte enable (bit[3:0], [7:4], …, [31:28]) of IdenRAM byte write enable.

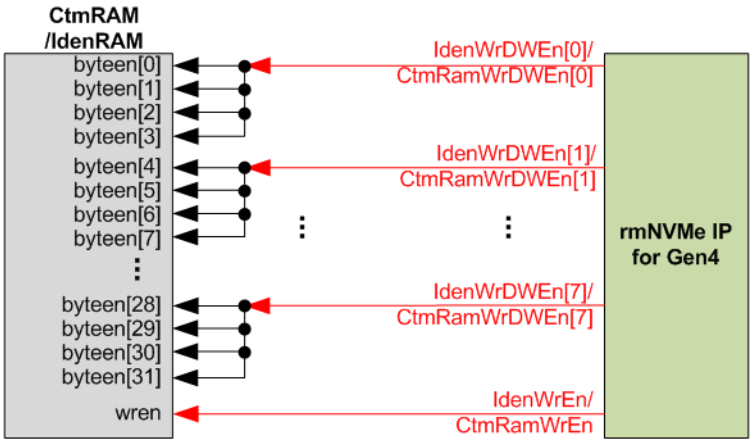


Figure 2-13 Word enable to byte write enable connection

On the other hand, CtmRAM is implemented using a Two-Port RAM (two read ports and two write ports), along with byte write enable feature. The connection to convert from word enable of rmNVMe-IP to byte enable of CtmRAM is similar to IdenRAM. Two-Port RAM is used to support additional features when a customized Custom command needs the data input. In case of SMART command, a simple dual port RAM is sufficient. Although the data size returned from SMART command is 512 bytes, CtmRAM is implemented using an 8 Kbyte RAM for customized Custom command in future use.

2.4 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 rmNVMe-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 2-14, is designed to connect the CPU system via the Avalon-MM interface, in compliance with the standard.

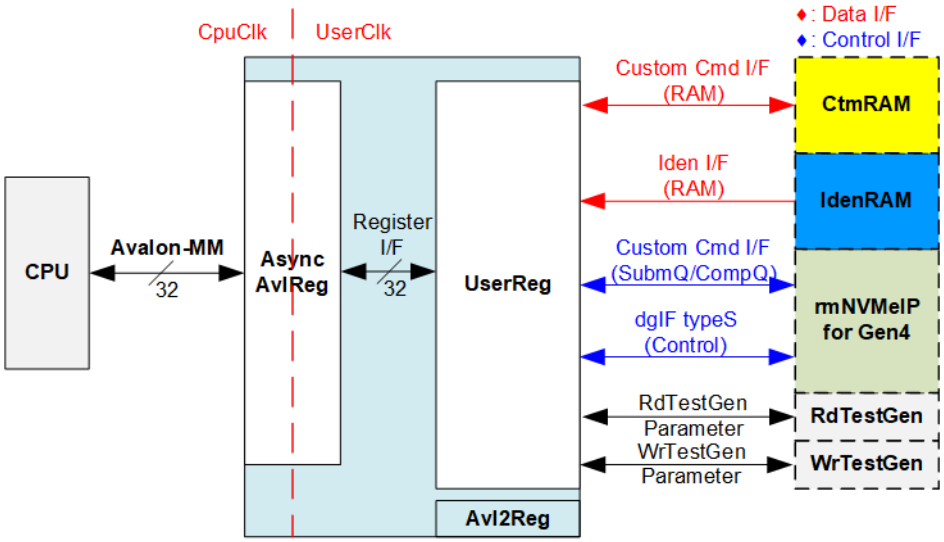


Figure 2-14 CPU and peripherals hardware

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 parameters and the status signals of other modules in the test system, including the CtmRAM, IdenRAM, rmNVMe-IP, RdTestGen, and WrTestGen. More details of AsyncAvlReg and UserReg are explained below.

2.4.1 AsyncAvlReg

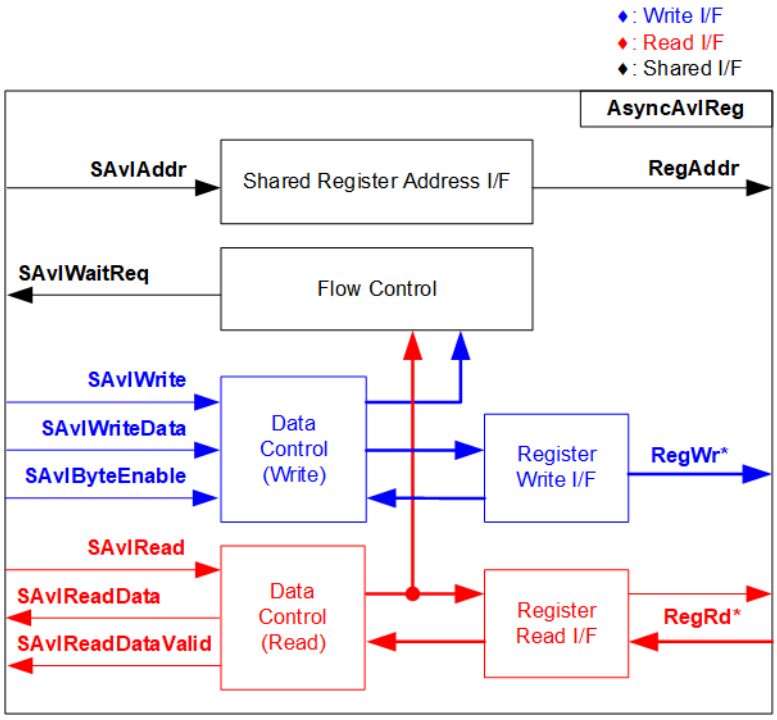


Figure 2-15 AsyncAvlReg Interface

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 be 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 2-16.

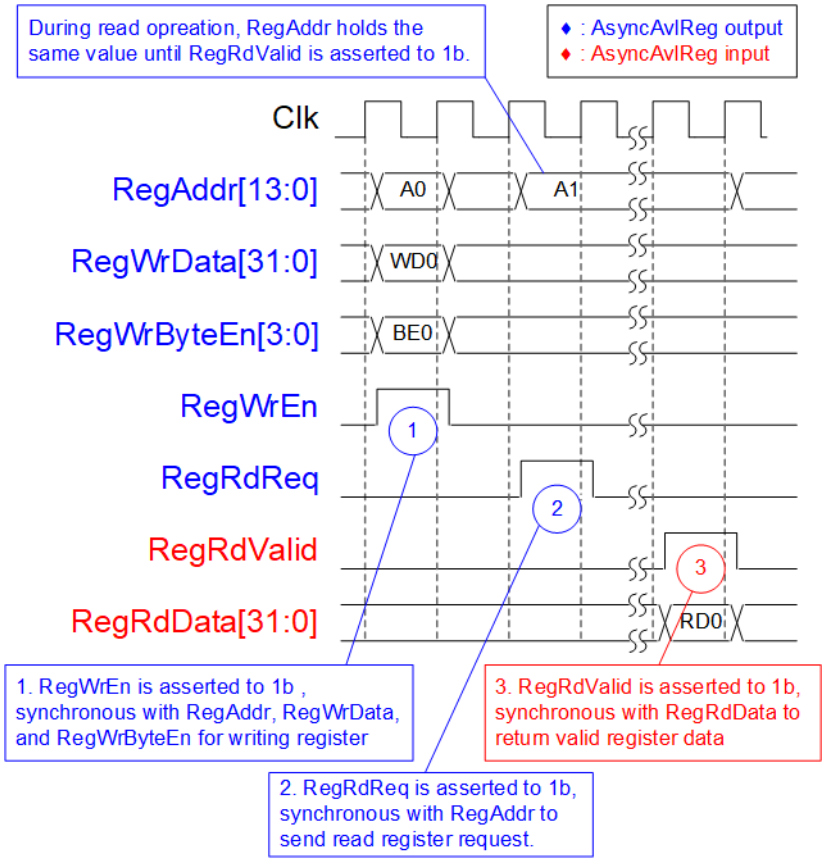


Figure 2-16 Register interface timing diagram

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.

2.4.2 UserReg

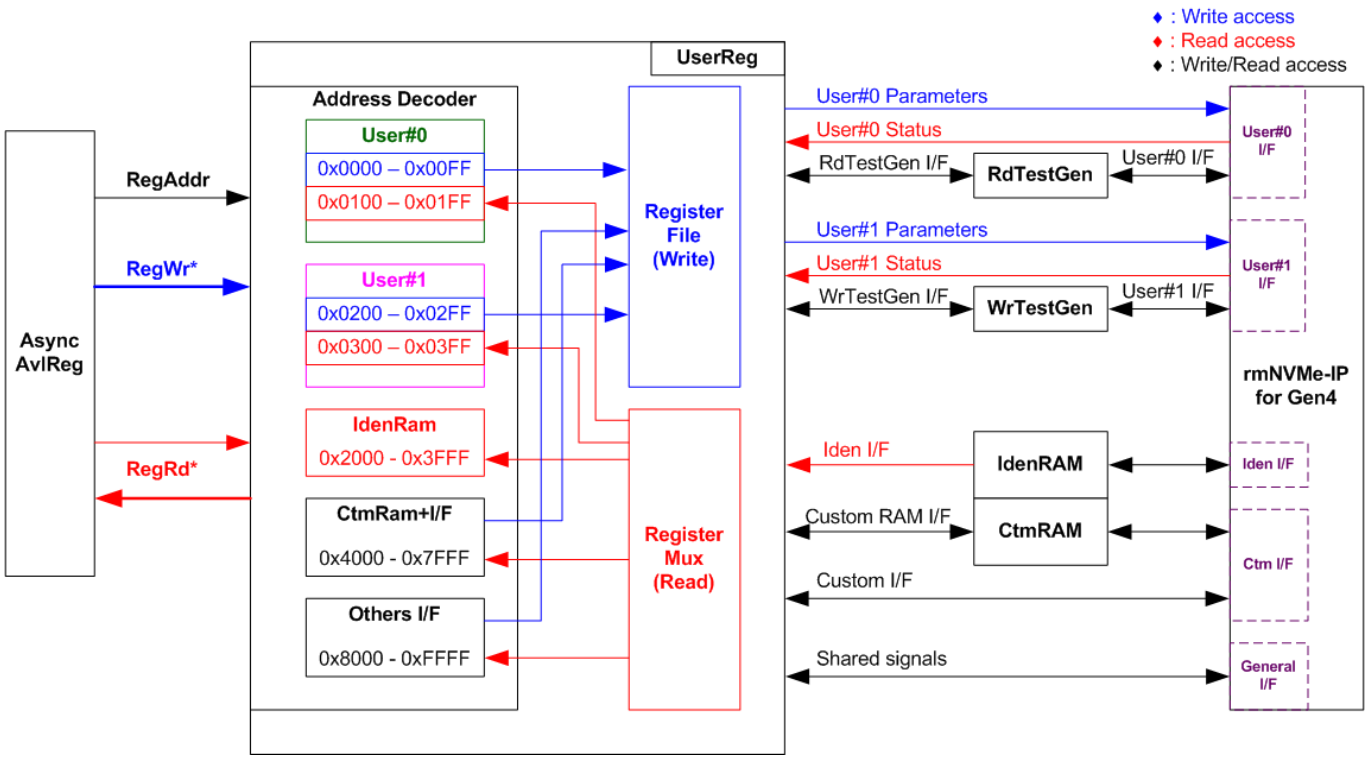


Figure 2-17 UserReg Interface

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 to be accessed for write or read transactions. The address range assigned within UserReg is divided into six areas, as shown in Figure 2-17.

1) 0x0000 – 0x01FF: Mapped to User#0 and RdTestGen
2) 0x0200 – 0x03FF: Mapped to User#1 and WrTestGen
3) 0x2000 – 0x3FFF: Mapped to read data from IdenRAM (read-only access).
4) 0x4000 – 0x5FFF: Mapped to write and read access with Custom command RAM interface. However, the current demo uses only read access through SMART command.
5) 0x6000 – 0x7FFF: Mapped to Custom command interface
6) 0x8000 – 0xFFFF: Mapped to other interfaces such as shared parameters for all Users, PCIe status, and IP version.

The Address decoder decodes the upper bits of RegAddr to select the active hardware (rmNVMe-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 32-bit pointer to set the hardware register.

For reading a register, multi-level multiplexers (mux) select the data to be returned to the CPU based on the address. The lower bits of RegAddr are fed to the submodule, enabling to select the active data from each submodule. Meanwhile, the upper bits are used within UserReg to select the returned data from each submodule. The total latency time for reading data is equal to three clock cycles, and RegRdValid is created by RegRdReq through the use of three D Flip-flops. Further details of the address mapping within the UserReg module can be found in Table 2-2.

Table 2-2 Register Map

Address	Register Name (Label in the "rmnvmeiptest.c")	Description
0x0000 – 0x01FF: Signal interface of User#0 (rmNVMe-IP) and RdTestGen		
0x0000 – 0x00FF: Control signals of User#0 and RdTestGen (Write access only)		
BA+0x0000	User#0 Address (Low) Reg (U0ADRL_INTREG)	[31:0]: Input to be bit[31:0] of User#0 start address as 512-byte unit (TrnAddr[31:0] of RdTestGen)
BA+0x0004	User#0 Address (High) Reg (U0ADRH_INTREG)	[15:0]: Input to be bit[47:32] of User#0 start address as 512-byte unit (TrnAddr[47:32] of RdTestGen)
BA+0x0008	User#0 Length (Low) Reg (U0LENL_INTREG)	[31:0]: Input to be bit[31:0] of User#0 transfer length as 512-byte unit (TrnLen[31:0] of RdTestGen)
BA+0x000C	User#0 Length (High) Reg (U0LENH_INTREG)	[15:0]: Input to be bit[47:32] of User#0 transfer length as 512-byte unit (TrnLen[47:32] of RdTestGen)
BA+0x0010	User#0 Command Reg (U0CMD_INTREG)	[2:0]: Input to be User#0 command (U0Cmd of rmNVMe-IP) 000b: Identify, 001b: Shutdown, 011b: Read SSD, 100b: SMART, 110b: Flush, 010b/101b/111b: Reserved When a Single mode command (not Read command) is written to this register, the new command request (UserCValid) is asserted to rmNVMe-IP. However, if it is a Read command, the start flag for Read command is asserted to RdTestGen, which then asserts the read command request (RdCmdValid) to rmNVMe-IP.
BA+0x0014	User#0 Test Pattern Reg (U0PATTSEL_INTREG)	[2:0]: Select test pattern of RdTestGen 000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1, 100b-LFSR [3]: Verification enable. 0b -No verification, 1b -Enable verification. [4]: Address mode. 0b -Sequential access, 1b -Random access
BA+0x0018	User#0 Transfer Rate Reg (U0TRNRATE_INTREG)	[6:0]: Transfer rate in percentage unit of RdTestGen (TrnRate[6:0] of RdTestGen) Valid from 1 – 100. For example, when this value=40, the maximum data rate is 40% of 275 x 256-bit (8.8 GB/s) = 3520 MB/s
0x0100 – 0x01FF: Status signals of User#0 and RdTestGen (Read access only)		
BA+0x0100	User#0 Status Reg (U0STS_INTREG)	[0]: Mapped to U0Busy of rmNVMe-IP. 0b: IP is Idle, 1b: IP is busy. [1]: Mapped to U0Error of rmNVMe-IP. 0b: No error, 1b: Error is found. [2]: Data verification failure. 0b: Normal, 1b: Error detected. [3]: Busy status of RdTestGen. 0b: Idle, 1b: Busy. [11:4]: Mapped to U0CId of rmNVMe-IP to show current command ID. [19:12]: Mapped to U0DId of rmNVMe-IP to show command ID which currently transfers data on Data stream interface. [28:20]: Mapped to U0CCnt of rmNVMe-IP to show remaining command count stored in rmNVMe-IP when executing Read command. [31]: Mapped to U0CReady of rmNVMe-IP to show command ready.
BA+0x0104	User#0 Error Type Reg (U0ERRTYPE_INTREG)	[31:0]: Mapped to U0ErrorType[31:0] of rmNVMe-IP to show error status
BA+0x0108	User#0 Admin Completion Status Reg U0AMCOMPSTS_INTREG)	[15:0]: Mapped to U0AdmCompStatus[15:0] of rmNVMe-IP to show status of Admin completion
BA+0x010C	User#0 IO Completion Status Reg U0IOCOMPSTS_INTREG)	[31:0]: Mapped to U0IOCompStatus[31:0] of rmNVMe-IP to show status of I/O completion.
BA+0x0110	User#0 Test pin (Low) Reg (U0TESTPINL_INTREG)	[31:0]: Mapped to U0TestPin[31:0] of rmNVMe-IP
BA+0x0114	User#0 Test pin (High) Reg (U0TESTPINH_INTREG)	[15:0]: Mapped to U0TestPin[47:32] of rmNVMe-IP
BA+0x0140-BA+0x015F	User#0 Expected value Word0-7 Reg (U0EXPPATW0-W7_INTREG)	The 256-bit expected data of the 1st failure in RdTestGen when executing a Read command. 0x0140: Bit[31:0], 0x0144: Bit[63:32], …, 0x015C: Bit[255:224]

Address	Register Name (Label in the "rmnvmeiptest.c")	Description
0x0100 – 0x01FF: Status signals of User#0 and RdTestGen (Read access only)		
BA+0x0160-BA+0x017F	User#0 Read value Word0-7 Reg (U0RDPATW0-W7_INTREG)	The 256-bit read data of the 1st failure in RdTestGen when executing a Read command. 0x0160: Bit[31:0], 0x0164: Bit[63:32], …, 0x017C: Bit[255:224]
BA+0x0180	User#0 Data Failure Address(Low) Reg (U0RDFAILNOL_INTREG)	[31:0]: Bit[31:0] of the byte address of the 1st failure when executing a Read command
BA+0x0184	User#0 Data Failure Address(High) Reg (U0RDFAILNOH_INTREG)	[24:0]: Bit[56:32] of the byte address of the 1st failure when executing a Read command
BA+0x0188	User#0 Completed Count (Low) Reg (U0CMDCMPCNTL_INTREG)	[31:0]: Bit[31:0] of the completed command count in RdTestGen
BA+0x018C	User#0 Completed Count (High) Reg (U0CMDCMPCNTH_INTREG)	[12:0]: Bit[44:32] of the completed command count in RdTestGen
0x0200 – 0x03FF: Signal interface of User#1 (rmNVMe-IP) and WrTestGen		
0x0200 – 0x02FF: Control signals of User#1 and WrTestGen (Write access only)		
BA+0x0200	User#1 Address (Low) Reg (U1ADRL_INTREG)	[31:0]: Input to be bit[31:0] of User#1 start address as 512-byte unit (TrnAddr[31:0] of WrTestGen)
BA+0x0204	User#1 Address (High) Reg (U1ADRH_INTREG)	[15:0]: Input to be bit[47:32] of User#1 start address as 512-byte unit (TrnAddr[47:32] of WrTestGen)
BA+0x0208	User#1 Length (Low) Reg (U1LENL_INTREG)	[31:0]: Input to be bit[31:0] of User#1 transfer length as 512-byte unit (TrnLen[31:0] of WrTestGen)
BA+0x020C	User#1 Length (High) Reg (U1LENH_INTREG)	[15:0]: Input to be bit[47:32] of User#1 transfer length as 512-byte unit (TrnLen[47:32] of WrTestGen)
BA+0x0210	User#1 Command Reg (U1CMD_INTREG)	[2:0]: Input to be User#1 command 010b: Write SSD, Others: Reserved When Write command is written to this register, the start flag for Write command is asserted to WrTestGen, which then asserts the command request (U1CValid) to rmNVMe-IP.
BA+0x0214	User#1 Test Pattern Reg (U1PATTSEL_INTREG)	[2:0]: Select test pattern of WrTestGen 000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1, 100b-LFSR [3]: Reserved [4]: Address mode. 0b-Sequential access, 1b-Random access
BA+0x0218	User#1 Transfer Rate Reg (U1TRNRATE_INTREG)	[6:0]: Transfer rate in percentage unit of WrTestGen (TrnRate[6:0] of WrTestGen)
0x0300 – 0x03FF: Status signals of User#1 and WrTestGen (Read access only)		
BA+0x0300	User#1 Status Reg (U1STS_INTREG)	[0]: Mapped to U1Busy of rmNVMe-IP. 0b: IP is Idle, 1b: IP is busy. [1]: Mapped to U1Error of rmNVMe-IP. 0b: No error, 1b: Error is found. [3]: Busy status of WrTestGen. 0b: Idle, 1b: Busy. [11:4]: Mapped to U1CId of rmNVMe-IP to show current command ID. [19:12]: Mapped to U1DId of rmNVMe-IP to show command ID which currently transfers data on Data stream interface. [28:20]: Mapped to U1CCnt of rmNVMe-IP to show remaining command count stored in rmNVMe-IP when executing Write command. [31]: Mapped to U1CReady of rmNVMe-IP to show command ready.
BA+0x0304	User#1 Error Type Reg (U1ERRTYPE_INTREG)	[31:0]: Mapped to U1ErrorType[31:0] of rmNVMe-IP to show error status
BA+0x030C	User#1 IO Completion Status Reg U1IOCOMPSTS_INTREG)	[31:0]: Mapped to U1IOCompStatus[31:0] of rmNVMe-IP to show status of I/O completion.
BA+0x0310	User#1 Test pin (Low) Reg (U1TESTPINL_INTREG)	[15:0]: Mapped to U1TestPin[15:0] of rmNVMe-IP
BA+0x0388	User#1 Completed Count (Low) Reg (U1CMDCMPCNTL_INTREG)	[31:0]: Bit[31:0] of the completed command count in WrTestGen
BA+0x038C	User#1 Completed Count (High) Reg (U1CMDCMPCNTH_INTREG)	[12:0]: Bit[44:32] of the completed command count in WrTestGen

Address	Register Name (Label in the "rmnvmeiptest.c")	Description
0x2000 – 0x3FFF: IdenRAM (Read access only)		
BA+0x2000-BA+0x2FFF	Identify Controller Data (IDENCTRL_CHARREG)	4Kbyte Identify Controller Data Structure
BA+0x3000-BA+0x3FFF	Identify Namespace Data (IDENNAME_CHARREG)	4Kbyte Identify Namespace Data Structure
0x4000 – 0x5FFF: CtmRAM (Write/Read access)		
BA+0x4000-BA+0x5FFF	Custom command Ram (CTMRAM_CHARREG)	Connect to 8Kbyte CtmRAM for storing 512-byte data output from SMART Command.
0x6000 – 0x7FFF: Custom Command Interface		
BA+0x6000-BA+0x603F Wr	Custom Submission Queue Reg (CTMSUBMQ_STRUCT)	[31:0]: Submission queue entry of SMART and Flush commands. Input to be CtmSubmDW0-DW15 of rmNVMe-IP. 0x6000: DW0, 0x6004: DW1, …, 0x603C: DW15
BA+0x6100-BA+0x610F Rd	Custom Completion Queue Reg (CTMCOMPQ_STRUCT)	[31:0]: CtmCompDW0-DW3 output from rmNVMe-IP. 0x6100: DW0, 0x6104: DW1, …, 0x610C: DW3
0x8000 – 0xFFFF: Other Interfaces		
BA+0x8000 Wr	NVMe Timeout Reg (NVMTIMEOUT_INTREG)	[31:0]: Mapped to TimeOutSet[31:0] of rmNVMe-IP
BA+0x8100 Rd	PCIe Status Reg (PCIESTS_INTREG)	[0]: PCIe linkup status from PCIe hard IP (0b: No linkup, 1b: linkup) [3:2]: Two lower bits to show PCIe link speed of PCIe hard IP. MSB is bit[16]. (000b: Not linkup, 001b: PCIe Gen1, 010b: PCIe Gen2, 011b: PCIe Gen3, 111b: PCIe Gen4) [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 PCIe hard IP datasheet [16]: The upper bit to show PCIe link speed of PCIe hard IP. Two lower bits are bit[3:2].
BA+0x8110 Rd	NVMe CAP Reg (NVMCAP_INTREG)	[31:0]: Mapped to NVMeCAPReg[31:0] of rmNVMe-IP
BA+0x8120 Rd	Total disk size (Low) Reg (LBASIZEL_INTREG)	[31:0]: Mapped to LBASize[31:0] of rmNVMe-IP
BA+0x8124 Rd	Total disk size (High) Reg (LBASIZEH_INTREG)	[15:0]: Mapped to LBASize[47:32] of rmNVMe-IP
BA+0x8200 Rd	IP Version Reg (IPVERSION_INTREG)	[31:0]: Mapped to IPVersion[31:0] of rmNVMe-IP

3 CPU Firmware

3.1 Test firmware (rmnvmeiptest.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 rmNVMe-IP to completes its own initialization process (U0-U1STS_INTREG[0]=0b). If errors are encountered, the process will stop and display an error message.
4) Display the status of the PCIe link, including the number of lanes and the speed, by reading PCIESTS_INTREG[16:2] status.
5) Display the main menu with options to run six commands for rmNVMe-IP, i.e., Identify, Write, Read, SMART, Flush, and Shutdown.

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

3.1.1 Identify Command

The sequence for the firmware when the Identify command is selected by User#0 I/F is as follows.

1) Set U0CMD_INTREG=000b to send the Identify command request on User#0 I/F of rmNVMe-IP. The busy flag of User#0 I/F (U0STS_INTREG[0]) will then change from 0b to 1b.
2) The CPU waits until the operation is completed or an error is detected by monitoring U0STS_INTREG[1:0].

   - If Bit[0] is de-asserted to 0b after the operation is finished, the data of Identify command returned by rmNVMe-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 U0ERRTYPE_INTREG[31:0]. The process will then stop.

3) After the busy flag (U0STS_INTREG[0]) is de-asserted to 0b, the CPU will display information decoded from IdenRAM (IDENCTRL_CHARREG), such as the SSD model name and the information from rmNVMe-IP output, such as SSD capacity (LBASIZEH/L_INTREG).

3.1.2 Write/Read Command

This menu allows the user to execute Write and Read commands. The user has the option to enable Write or Read operation separately with specific parameters. The sequence of the firmware for this menu is as follows.

1) Input two sets of parameters (Write command parameters and Read command parameters) from the console, including the command (enable or disable), the address mode (Sequential or Random), Data verification (enable or disable for Read command only), the start address, transfer length, test pattern, and transfer rate from the console. If any inputs are invalid, the operation will be cancelled.
2) After obtaining all the inputs, set them to U0-U1ADRL/H_INTREG, U0-U1LENL/H_INTREG, U0-U1PATTSEL_INTREG, and U0-U1TRNRATE_INTREG.
3) To execute the Read command, set U0CMD_INTREG[2:0] = 011b. To execute the Write command, set U1CMD_INTREG[2:0] = 010b. This sends the command request to the corresponding User I/F. Once the command is issued, the busy flags for both rmNVMe-IP and TestGen of the active user (U0-U1STS_INTREG[0] and U0-U1STS_INTREG[3], respectively) 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 U0-U1STS_INTREG[3:0].

   - Bit[0] will be de-asserted to 0b when the User#0-#1 command is completed.
   - Bit[1] will be asserted when an error is detected in User#0-#1. In this case, the error message will be displayed on the console to show the decoded error details from U0-U1ERRTYPE_INTREG, and the process will be stopped.
   - Bit[2] will be asserted when data verification fails for User#0. In this case, the verification error message will then be displayed on the console, but the CPU will continue to run until the operation is complete or the user inputs any key to cancel the operation.
   - Bit[3] will be de-asserted to 0b when RdTestGen/WrTestGen is completed.

   While the command is running, the current transfer size of the active user, read from U0-U1CMDCMPCNTL/H_INTREG, will be displayed every second.

5) Once the busy flags (U0-U1STS_INTREG[0] and U0-U1STS_INTREG[3]) are de-asserted to 0b, CPU will calculate and display the test result of the active user on the console including the total time usage, total transfer size, transfer speed, and IOPS.

3.1.3 SMART Command

The sequence for the firmware when the SMART command is selected by User#0 I/F is as follows.

1) The 16-Dword of the Submission Queue entry (CTMSUBMQ_STRUCT) is set to the SMART command value.
2) Set U0CMD_INTREG[2:0]=100b to send the SMART command request on User#0 I/F of rmNVMe-IP. The busy flag of User#0 I/F (U0STS_INTREG[0]) will then change from 0b to 1b.
3) The CPU waits until the operation is completed or an error is detected by monitoring U0STS_INTREG[1:0].

    - If Bit[0] is de-asserted to 0b after the operation is finished, the data of SMART command returned by rmNVMe-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 U0ERRTYPE_INTREG[31:0]. The process will then stop.

4) After the busy flag (U0STS_INTREG[0]) is de-asserted to 0b, the CPU will display information decoded from CtmRAM (CTMRAM_CHARREG) including Remaining Life, Percentage Used, Temperature, Total Data Read, Total Data Written, Power On Cycles, Power On Hours, and Number of Unsafe Shutdown.

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

3.1.4 Flush Command

The sequence for the firmware when the Flush command is selected by User#0 I/F is as follows.

1) The 16-Dword of the Submission Queue entry (CTMSUBMQ_STRUCT) is set to the Flush command value.
2) Set U0CMD_INTREG[2:0]=110b to send Flush command request on User#0 I/F of rmNVMe-IP. The busy flag of User#0 I/F (U0STS_INTREG[0]) will then change from 0b to 1b.
3) The CPU waits until the operation is completed or an error is detected by monitoring U0STS_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 U0ERRTYPE_INTREG[31:0]. The process will then stop.

3.1.5 Shutdown Command

The sequence for the firmware when the Shutdown command is selected by User#0 I/F is as follows.

1) Set U0CMD_INTREG=001b to send the Shutdown command request on User#0 I/F of rmNVMe-IP. The busy flag of User#0 I/F (U0STS_INTREG[0]) will then change from 0b to 1b.
2) The CPU waits until the operation is completed or an error is detected by monitoring U0STS_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 U0ERRTYPE_INTREG[31:0]. The process will then stop.

3) After Shutdown command completes, both the SSD and rmNVMe-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 rmNVMe-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(unsigned int user)	
Parameters	user: 0-1 for User#0-#1, respectively
Return value	Read value of U0-U1CMDCMPCNTH/L_INTREG
Description	The value of U0-U1CMDCMPCNTH/L_INTREG is read and converted to byte units before being returned as the result of the function.

int get_param(unsigned int user, userin_struct* userin)	
Parameters	user: 0-1 for User#0-#1, respectively userin: Seven inputs from user, i.e., command, address mode, data verification, start address, total length in 512-byte unit, test pattern, and transfer rate
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_flush(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the rmNVMe-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 rmNVMe -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(unsigned int user)	
Parameters	user: 0-1 for User#0-#1, respectively
Return value	None
Description	Read U0-U1ERRTYPE_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 U0-U1TESTPINL/H_INTREG.

void show_result(unsigned int user, unsigned int timeuseh, unsigned int timeusel)	
Parameters	user: 0-1 for User#0-#1, respectively timeuseh, timeusel: 64-bit read value of timer
Return value	None
Description	Print command and total size by calling show_size function. After that, calculate total time usage from timeuseh and timeusel 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 or GByte 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 U0RDFAILNOL/H_INTREG (error byte address), U0EXPPATW0-W7_INTREG (expected value), and U0RDPATW0-W7_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(void)	
Parameters	None
Return value	0: No error, -1: Receive invalid input
Description	Execute Write command and Read command as outlined in section 3.1.2 (Write/Read Command). Show_result function is called to calculate and display transfer performance of the Write and Read command.

4 Example Test Result

The demo system was tested using an 800 GB Intel Optane P5800X SSD on Agilex 7 F-Series development board. The results of the tests are presented in Figure 4-1 and Figure 4-2.

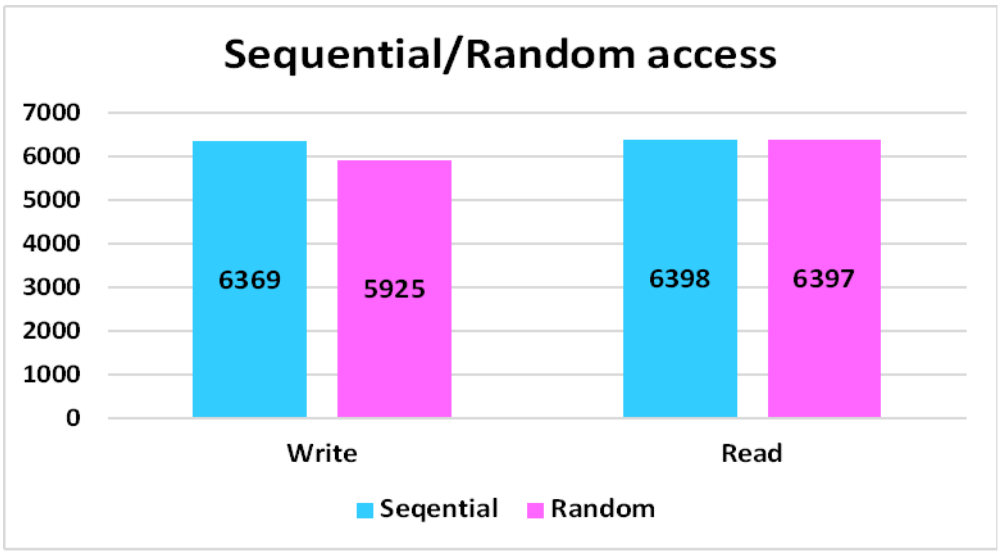


Figure 4-1 Test Performance of Write and Read commands

Figure 4-1 shows the performance of Random access and Sequential access when executing either Write or Read command. The Write performance for Random access is slightly lower than that of Sequential access. However, the Read performance for both Random access and Sequential access is the same. It is important to recognize that while some SSDs exhibit excellent performance with Sequential access, their performance significantly deteriorates for Random access.

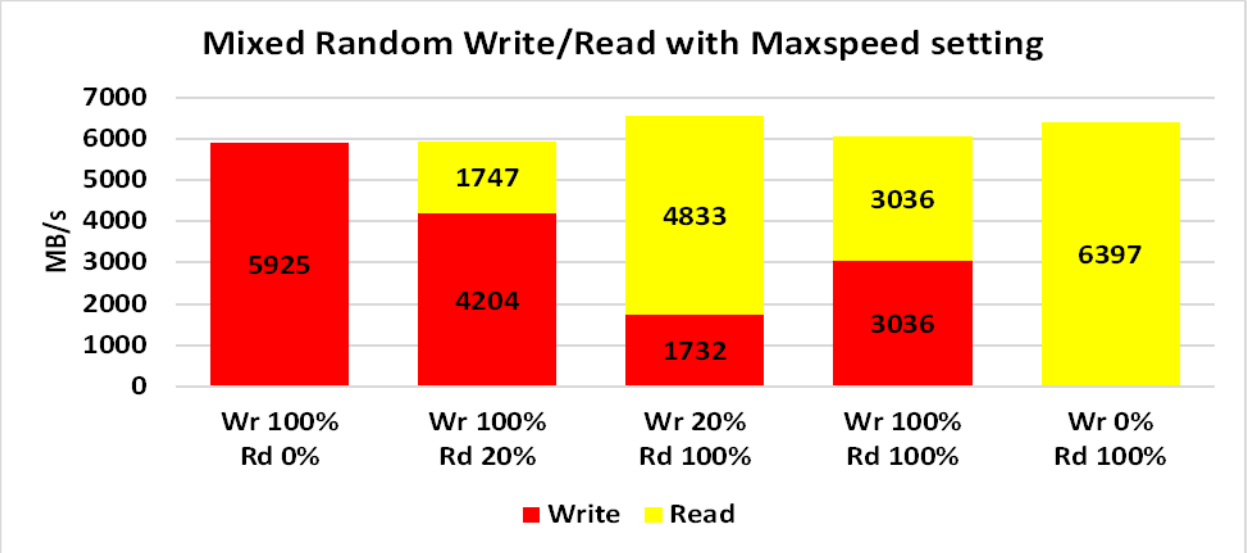


Figure 4-2 Test Performance of Mixed Random Write Read command

The demo features the ability to set the maximum data rate for both write and read operation. Figure 4-2 displays the results by adjusting the percentage of the maximum data rate for each operation. Five data sets are presented, including scenarios where only Write or Read is executed, where Write or Read is limited to 20% (1760 MB/s), and where there are no limits for both Write and Read. The total sum of write and read speeds across all settings remain consistent, averaging around 6000 – 6500 MB/s.

This SSD demonstrates a well-balanced performance for both write and read operations. When either Write or Read commands are executed alone, the read speed exceeds the write speed. However, when both write and read commands are set to the same rate, the performance of both becomes equal. As a result, write-sensitive systems should reduce the number of Read command requests to the rmNVMe-IP, while read-intensive systems should minimize the number of Write command requests.

It is worth mentioning that some SSDs may exhibit varying and unbalanced write-read performance when both commands are run simultaneously. The rmNVMe-IP demo is available for use to assess the characteristics of your own SSD.

5 Revision History

Revision	Date	Description
1.0	22-Jun-23	Initial Release