NVMe-IP reference design manual

Rev1.0   23-Sep-20

1   NVMe

NVM Express (NVMe) defines the interface for the host controller to access solid state drives (SSD) by PCI Express. NVM Express optimizes the process to issue command and completion by using only two registers (Command issue and Command completion). Also, NVMe supports parallel operation by supporting up to 64K commands within single queue. 64K command entries improve transfer performance for both sequential and random access.

In PCIe SSD market, two standards are used, i.e. AHCI and NVMe. AHCI is the older standard to provide the interface for SATA hard disk drive while NVMe is optimized for non-volatile memory like SSD. The comparison between both AHCI and NVMe protocol in more details is described in "A Comparison of NVMe and AHCI" document.
https://sata-io.org/system/files/member-downloads/NVMe%20and%20AHCI_%20_long_.pdf

The example of NVMe storage device is shown in http://www.nvmexpress.org/products/.
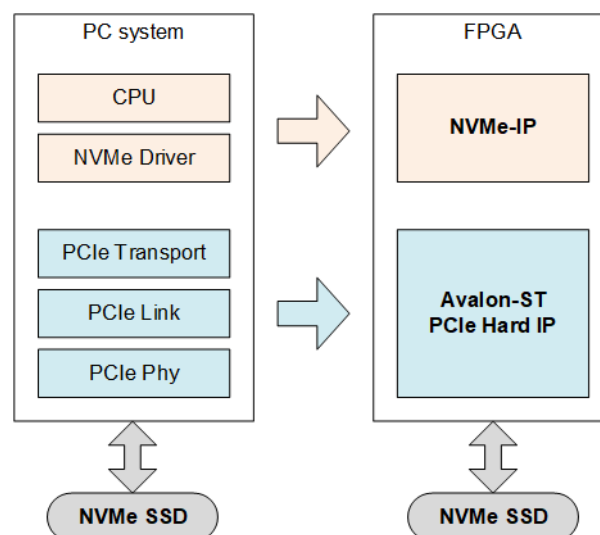

Figure 1-1 NVMe protocol layer

To access NVMe SSD, the general system implements NVMe driver running on the processor, as shown in the left side of Figure 1-1. The physical connection of NVMe standard is PCIe connector which is one-to-one type, so one PCIe host can connect to one PCIe device. NVMe-IP implements NVMe driver to access NVMe SSD by using pure-hardware logic. The user can access NVMe SSD without including processor and driver but using NVMe-IP in FPGA board. Using pure-hardware logic to be NVMe host controller can reduce the overhead time for software-hardware handshake, so using NVMe-IP can show very high performance for writing and reading with NVMe SSD.
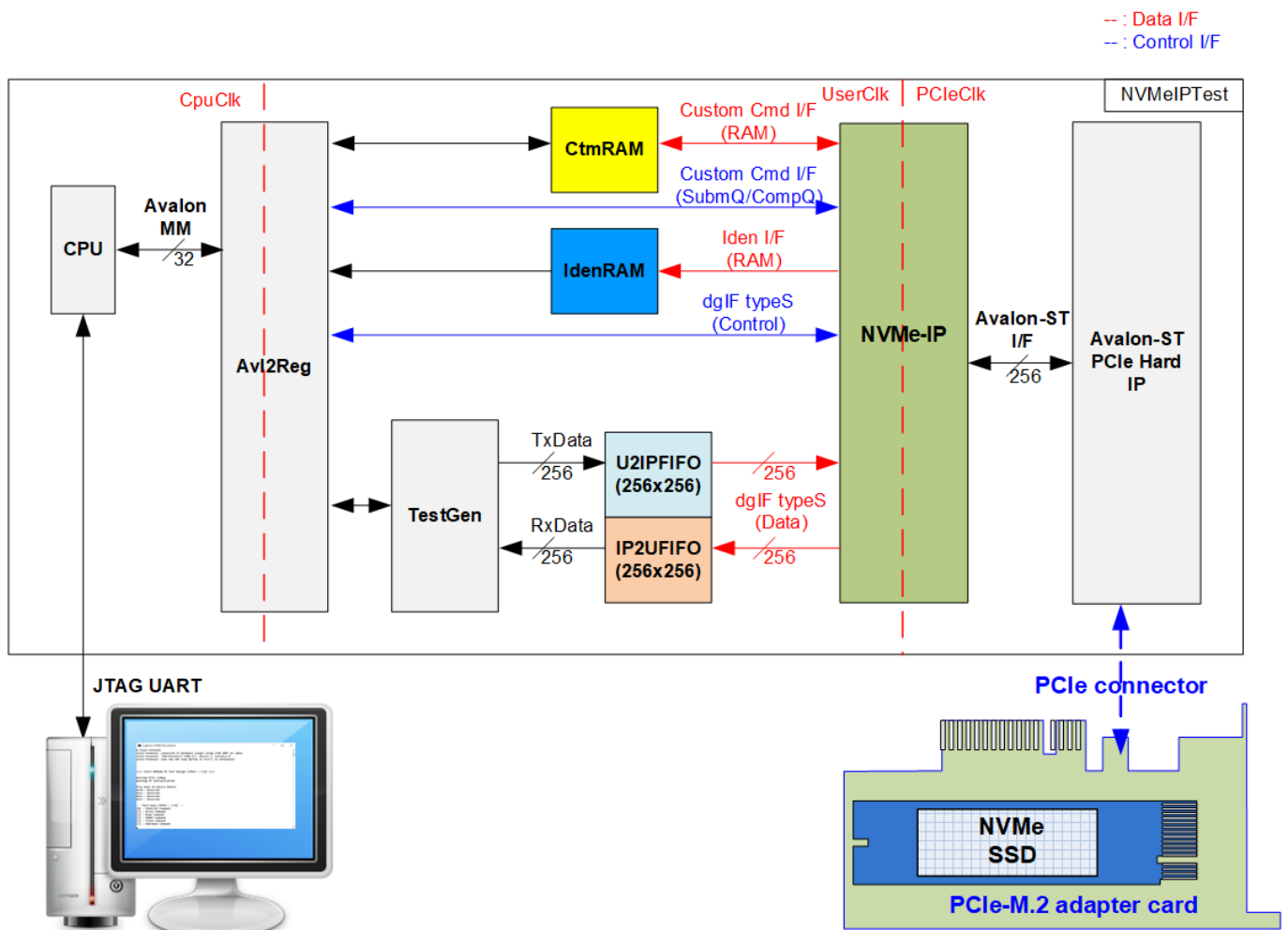
2 Hardware overview

Figure 2-1 NVMe-IP demo hardware

Following the function of each module, all hardware modules inside the test system are divided to three types, i.e. test function (TestGen), NVMe function (CtmRAM, IdenRAM, U2IPFIFO, IP2UFIFO, NVMe-IP and PCIe block) and CPU system (CPU and Avl2Reg).

TestGen is the test logic to generate test data stream for NVMe-IP via U2IPFIFO or read data stream output from NVMe-IP via IP2UFIFO to verify it. NVMe includes the NVMe-IP and the PCIe hard IP (Avalon-ST PCIe hard IP) for accessing NVMe SSD directly without PCIe switch. CPU and Avl2Reg are designed to interface with user via JTAG UART. User can set command and the test parameters on the console. Also, the current status of the test hardware is monitored by user on the console. The CPU firmware must be implemented to control the flow for operating each command.

The data interface of NVMe-IP connects with four memory blocks, i.e. CtmRAM, IdenRAM, U2IPFIFO and IP2UFIFO. CtmRAM stores returned data from SMART command while IdenRAM stores returned data from Identify command. U2IPFIFO stores data for Write command while IP2UFIFO stores data for Read command. TestGen always writes data with U2IPFIFO or reads data with IP2UFIFO when the FIFO is ready. So, TestGen can check the performance of NVMe-IP system.

There are three clock domains displayed in Figure 2-1, i.e. CpuClk, UserClk and PCIeClk. CpuClk is the clock domain of CPU and its peripherals. This clock must be stable clock which is independent from the other hardware interface. UserClk is the example user clock domain which may be independent clock from the other clock domains for running the user interface of NVMe-IP, RAM, FIFO and TestGen. According to NVMe-IP datasheet, clock frequency of UserClk must be more than or equal to PCIeClk. So, this reference design uses 150 MHz for PCIe Gen3 speed. PCIeClk is the clock output from PCIe hard IP to synchronous with data stream of 256-bit Avalon-ST interface. When the PCIe hard IP is set to 4-lane PCIe Gen3, PCIeClk frequency is equal to 125 MHz.

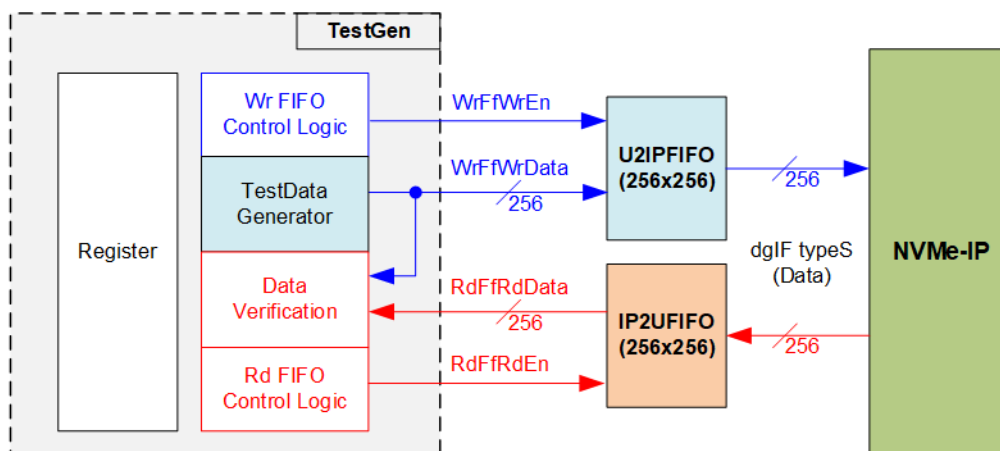More details of the hardware are described as follows.

2.1 TestGen

Figure 2-2 TestGen interface

TestGen module handles the data interface of NVMe-IP for transferring the data in Write and Read command. In Write command, TestGen sends 256-bit test data to NVMe-IP via U2IPFIFO. In Read command, the test data is fed from IP2UFIFO to compare with the expected value for data verification. Data bandwidth of TestGen is matched to NVMe-IP by running at the same clock and the same data bus size. Control logic asserts Write enable or Read enable to '1' for writing or reading the FIFO when it is ready. So, NVMe-IP can transfer data with U2IPFIFO and IP2UFIFO without waiting data ready. As a result, the test logic shows the best performance to write and read data with the SSD through NVMe-IP.

Register file in the TestGen receives test parameters from user, i.e. total transfer size, transfer direction, verification enable and test pattern selector. The internal logic includes the counter to control total transfer size of test data. The details of hardware logic of TestGen are shown in Figure 2-3.
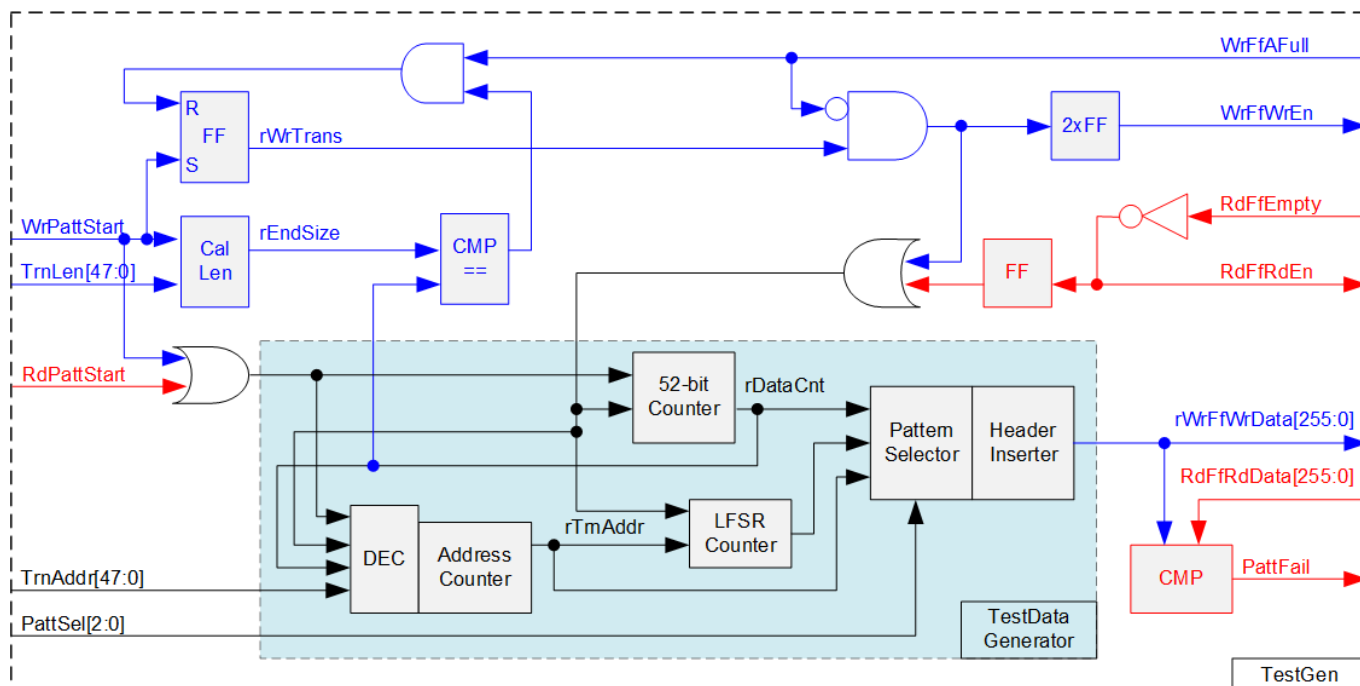

Figure 2-3 TestGen hardware

As shown in the right side of Figure 2-3, flow control signals of FIFO are WrFfAFull and RdFfEmpty. When FIFO is almost full during write operation (WrFfAFull='1'), WrFfWrEn is de-asserted to '0' to pause data sending to FIFO. For read operation, when FIFO has data (RdFfEmpty='0'), the logic reads data from FIFO to compare with the expected data by asserting RdFfRdEn to '1'.

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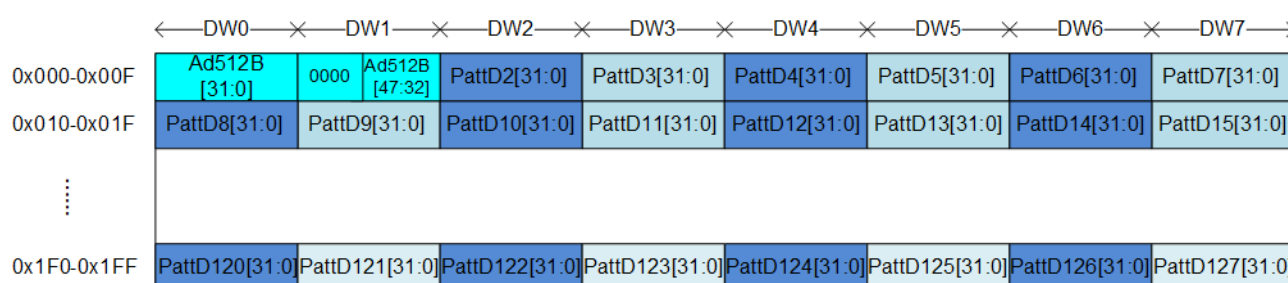

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

512-byte data consists of 64-bit header in Dword#0 and Dword#1 and the test data in remaining words of 512-byte data (Dword#2 – Dword#127). The header is created by using the address in 512-byte unit (rTrnAddr), output from the Address counter. The address counter loads the start value from user (TrnAddr) and then increases the value after finishing 512-byte data transferring. Remaining Dwords depends on pattern selector. 32-bit incremental data is designed by using 52-bit counter. The decremental data can be designed by connecting NOT logic to the incremental data. The LFSR pattern is designed by using LFSR counter. The equation of LFSR is $x^{31} + x^{21} + x + 1$.

To implement 256-bit LFSR pattern, the data is split to be two sets of 128-bit data which uses the different start value. 128-bit data uses look-ahead technique to calculate four 32-bit LFSR data in one clock cycle. As shown Figure 2-5, start value of each LFSR set is designed by using combination signal of 32-bit LBA address and NOT logic of LBA address (LBAAddrB means NOT logic of LBAAddr signal).
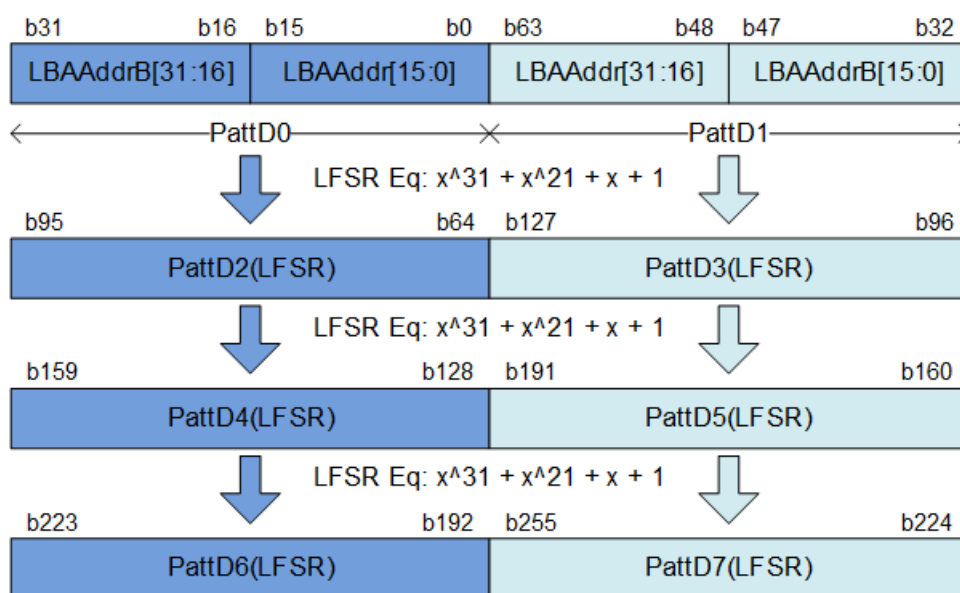
Figure 2-5 256-bit LFSR Pattern in TestGen

Test data is fed to be write data to the FIFO or the expected data for verifying with the read data from FIFO. Fail flag is asserted to '1' when data verification is failed. The example of timing diagram to write data to FIFO is shown as follows.
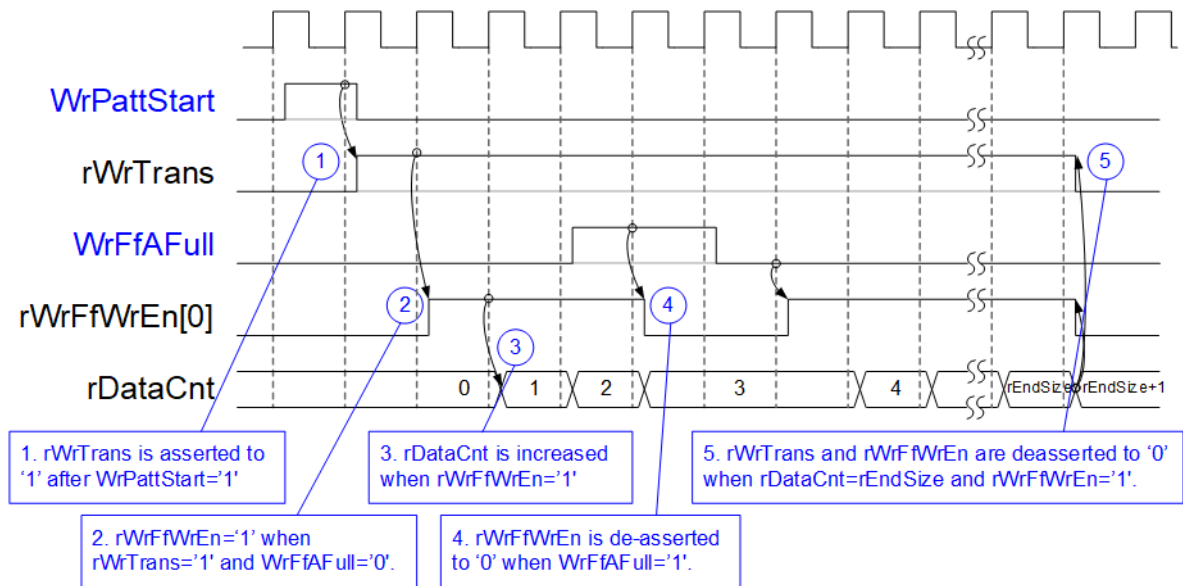
Figure 2-6 Timing diagram of Write operation in TestGen

1) WrPattStart is asserted to '1' for one clock cycle when user sets the register to start write operation. In the next clock, rWrTrans is asserted to '1' to enable the control logic for generating write enable to FIFO.

2) Write enable to FIFO (rWrFfWrEn) is asserted to '1' when two conditions are met. First, rWrTrans must be asserted to '1' during the write operation being active. Second, the FIFO must not be full by monitoring WrFfAFull='0'.

3) The write enable is also applied to be counter enable for counting total data (rDataCnt) in the write operation.

4) If FIFO is almost full (WrFfAFull='1'), the write process is paused by de-asserting rWrFfWrEn to '0'.

5) When total data count is equal to the set value, rWrTrans is de-asserted to '0'. At the same time, rWrFfWrEn is also de-asserted to '0' to finish data generating.

When running read operation, read enable of FIFO is controlled by empty flag of FIFO. Data is read when FIFO has the data. Comparing to write enable, the read enable signal is not stopped by total count and not started by start flag. When the read enable is asserted to '1', the data counter and the address counter are increased for counting total data and generating the header of expect value respectively.
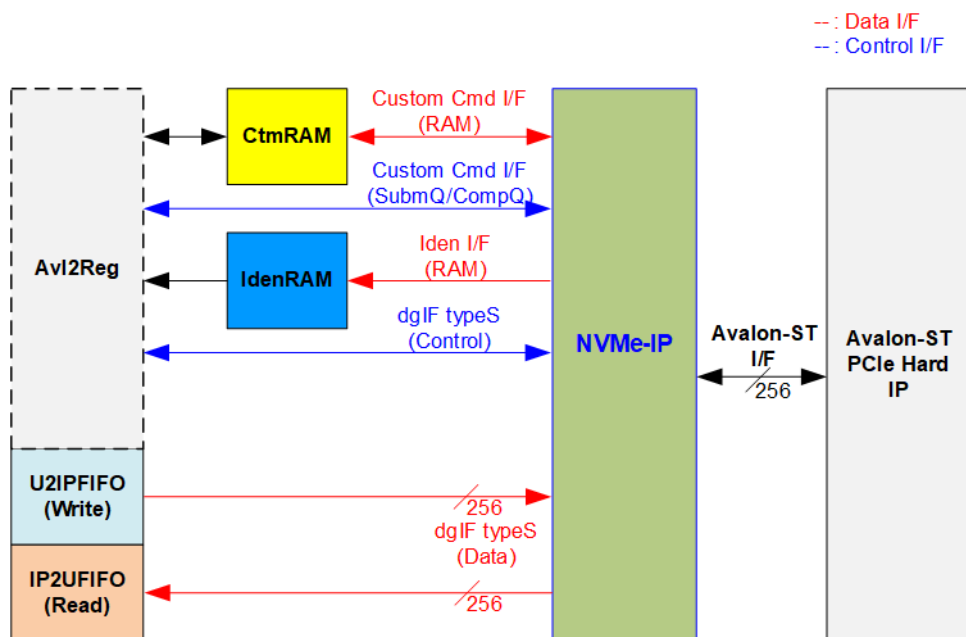
2.2 NVMe

Figure 2-7 NVMe hardware

Figure 2-7 shows the example to interface NVMe-IP in the reference design. The user interface of NVMe-IP consists of control interface and data interface. The control interface receives the command and the parameters from Custom command interface or dgIF typeS, depending on the command. Custom command interface is used when operating SMART command or Flush command.

The data interface of NVMe-IP has four interfaces, i.e. Custom command RAM interface, Identify interface, FIFO input interface (dgIF typeS) and FIFO output interface (dgIF typeS). Data bus width of all interfaces is 256-bit. The custom command RAM interface is bi-directional interface while the other interfaces are one directional interface. In the reference design, the Custom command RAM interface is used for transferring one direction only when NVMe-IP sends SMART data to Avl2Reg.

2.2.1 NVMe-IP

NVMe-IP implements NVMe protocol of the host side to access one NVMe SSD directly without PCIe switch connection. The NVMe-IP supports six commands, i.e. Write, Read, Identify, Shutdown, SMART and Flush. NVMe-IP can connect 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.pdf

2.2.2 Avalon-ST Intel Stratix10 Hard IP for PCIe

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

https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_s10_pcie_avst.pdf

2.2.3 Two-port RAM

Two of 2-Port RAMs, CtmRAM and IdenRAM, store data from Identify command and SMART command respectively. IdenRAM is simple dual-port RAM which has one read port and one write port. The data size of Identify command is 8Kbyte, so IdenRAM size is 8Kbyte. NVMe-IP and Avl2Reg have different data bus size, so IdenRAM sets the different bus size for write port and read port. The data interface of NVMe-IP (write port) is 256-bit while the interface of Avl2Reg (read port) is 32-bit. Furthermore, NVMe-IP has double word enable to write only 32-bit data in some cases. The RAM setting on IP catalog of QuartusII supports the write byte enable. So, one bit of double word enable is extended to be 4-bit write byte enable as shown in Figure 2-8.
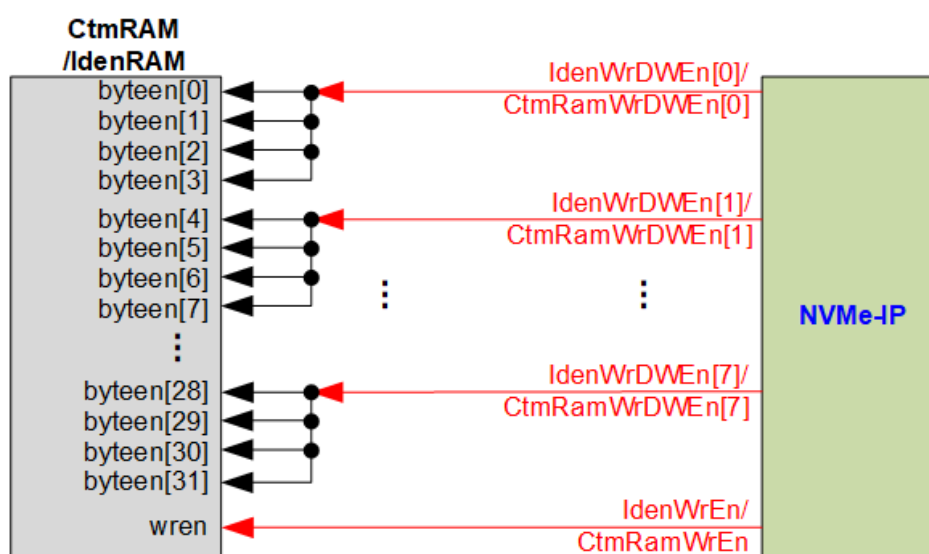
Figure 2-8 Word enable to be byte write enable connection

Bit[0], [1], …, [7] of WrDWEn is fed to bit[3:0], [7:4], …, [31:28] of IdenRAM byte write enable respectively.

Comparing with IdenRAM, CtmRAM is implemented by true dual-port RAM (two read ports and two write ports) with byte write enable. The connection to convert from word enable of NVMe-IP to byte enable of CtmRAM is similar to IdenRAM. True dual-port RAM is used to support the additional features when the customized custom command needs the data input. To support SMART command, using simple dual port RAM is enough. Though the data size returned from SMART command is 512 bytes, CtmRAM is implemented by 8Kbyte RAM for customized custom command.

2.3   CPU and Peripherals

32-bit Avalon-MM bus is applied to be the bus interface for CPU accessing the peripherals such as Timer and JTAG UART. The test system of NVMe-IP is connected with CPU as a peripheral on 32-bit Avalon-MM bus for CPU controlling and monitoring. CPU assigns the different base address and the address range to each peripheral for accessing one peripheral at a time.

In the reference design, the CPU system is built with one additional peripheral to access the test logic. The base address and the range for accessing the test logic are defined in the CPU system. So, the hardware logic must be designed to support Avalon-MM bus standard for CPU writing and reading. Avl2Reg module is designed to connect the CPU system as shown in Figure 2-9.
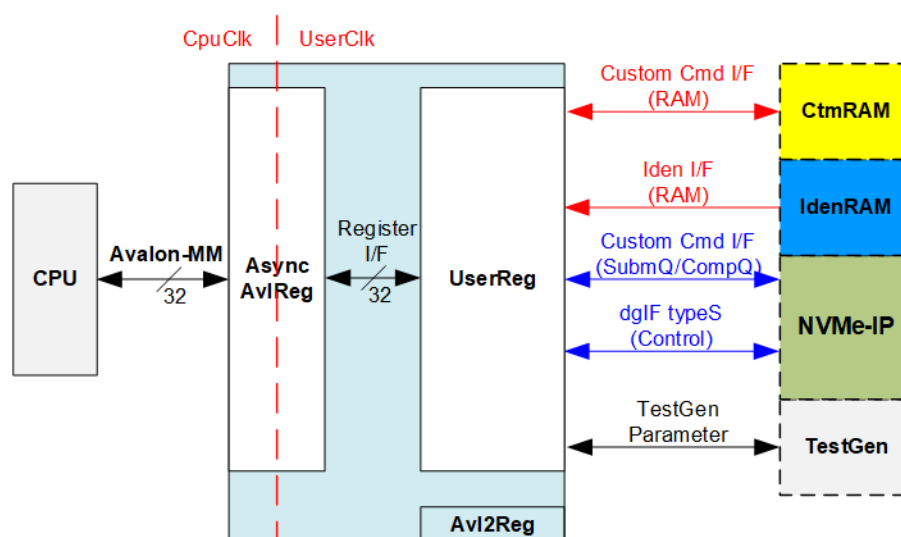
Figure 2-9 CPU and peripherals hardware

Avl2Reg consists of AsyncAvlReg and UserReg. AsyncAvlReg is designed to convert the Avalon-MM signals to be the simple register interface which has 32-bit data bus size, similar to Avalon-MM data bus size. In addtition, AsyncAvlReg includes asynchronous logic to support clock crossing between CpuClk domain and UserClk domain.

UserReg includes the register file of the parameters and the status signals of other modules in the test system, i.e. CtmRAM, IdenRAM, NVMe-IP and TestGen. More details of AsyncAvlReg and UserReg are described as follows.

2.3.1 AsyncAvlReg

Figure 2-10 AsyncAvlReg Interface

The signal on Avalon-MM bus interface can be split into three groups, i.e. Write channel (blue color), Read channel (red color) and Shared control channel (black color). More details of Avalon-MM interface specification is described in following document.
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf

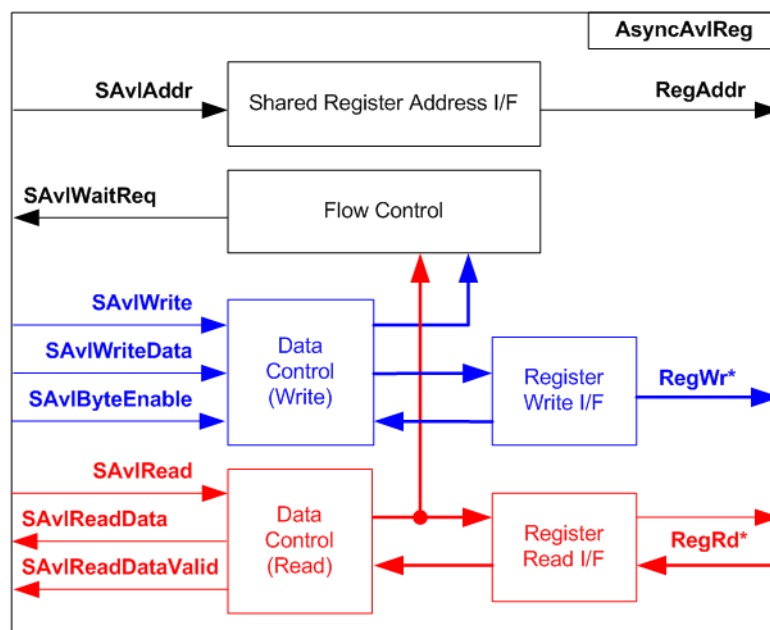According to Avalon-MM specification, one command (write or read) can be operated at a time. The logics inside AsyncAvlReg are split into three groups, i.e. Write control logic, Read control logic and Flow control logic. Flow control logic controls SAvlWaitReq to hold the next request from Avalon-MM interface if the current request does not finish. Write control and Write data I/F of Avalon-MM bus are latched and transferred to be Write register interface with clock-crossing registers. In the same way, Read control I/F are latched and transferred to be Read register interface with clock-crossing registers. After that, the returned data from Register Read I/F is transferred to Avalon-MM bus by using clock-crossing registers. Address I/F of Avalon-MM is latched and transferred to Address register interface as well.

The simple register interface is compatible with single-port RAM interface for write transaction. The read transaction of the register interface is slightly modified from RAM interface by adding RdReq and RdValid signals for controlling read latency time. The address of register interface is shared for write and read transaction, so user cannot write and read the register at the same time. The timing diagram of the register interface is shown in Figure 2-11.
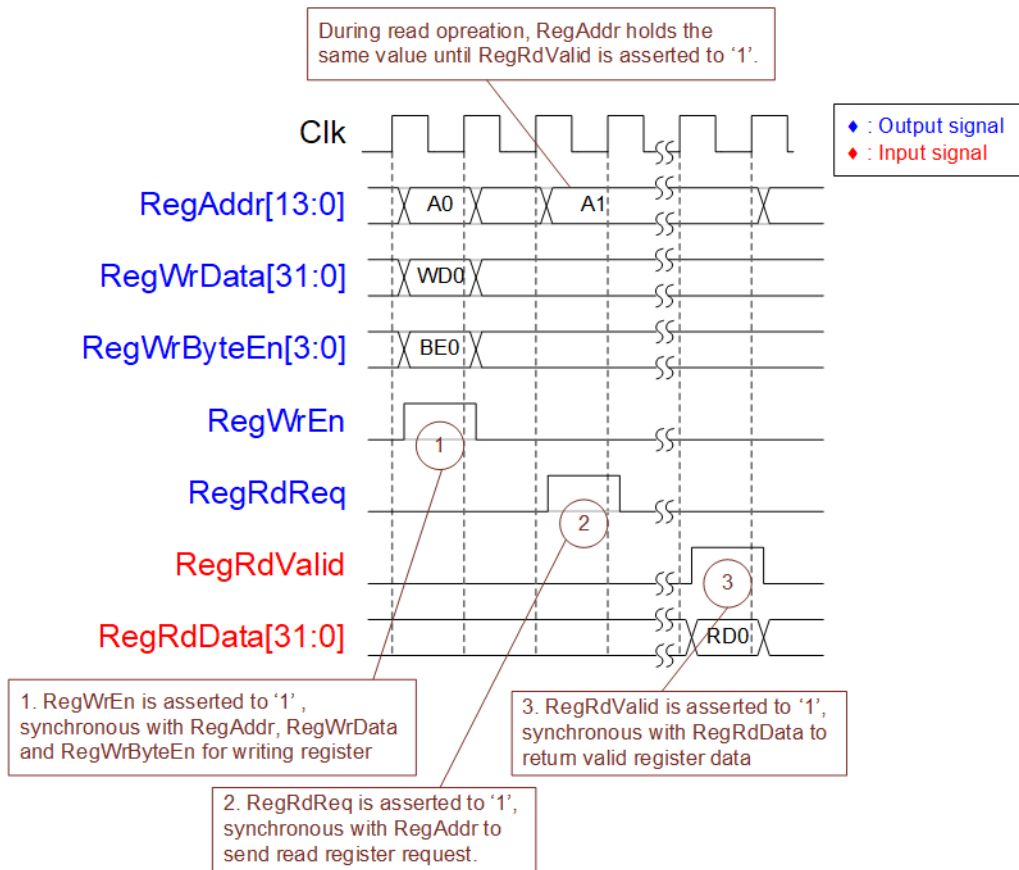
Figure 2-11 Register interface timing diagram

1) To write register, the timing diagram is similar to single-port RAM interface. RegWrEn is asserted to '1' with the valid signal of RegAddr (Register address in 32-bit unit), RegWrData (write data of the register) and RegWrByteEn (the write byte enable). Byte enable has four bits to be the byte data valid. Bit[0], [1], [2] and [3] is equal to '1' when RegWrData[7:0], [15:8], [23:16] and [31:24] is valid respectively.

2) To read register, AsyncAvlReg asserts RegRdReq to '1' with the valid value of RegAddr. 32-bit data must be returned after receiving the read request. The slave must monitor RegRdReq signal to start the read transaction. During read operation, the address value (RegAddr) does not change the value until RegRdValid is asserted to '1'. So, the address can be used for selecting the returned data by using multiple layers of multiplexer.

3) The read data is returned on RegRdData bus by the slave with asserting RegRdValid to '1'. After that, AsyncAvlReg forwards the read value to SAvlRead interface.

2.3.2 UserReg

Figure 2-12 UserReg Interface

The address range to map to UserReg is split into six areas, as shown in Figure 2-12.
1) 0x0000 – 0x00FF: mapped to set the command with the parameters of NVMe-IP and raTestGen. 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.
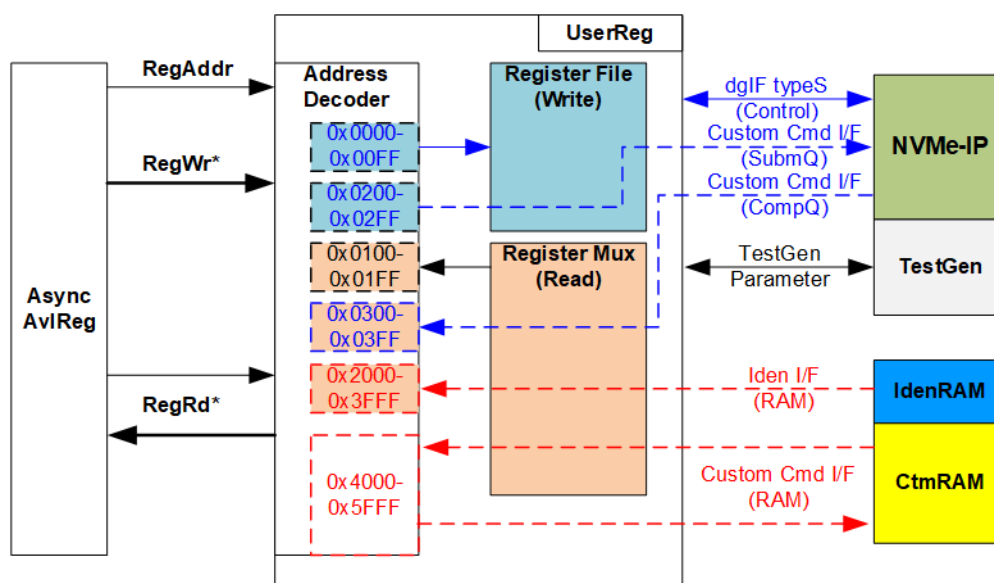3) 0x0100 – 0x01FF: mapped to read the status signals of NVMe-IP and TestGen. This area is read access only.
4) 0x0300 – 0x03FF: mapped to read the status of custom command interface (NVMe-IP). This area is read access only.
5) 0x2000 – 0x3FFF: mapped to read data from IdenRAM. This area is read access only.
6) 0x4000 – 0x5FFF: mapped to custom command RAM interface (NVMe-IP). This area supports write access and read access. The demo shows only read access by running SMART command.

Address decoder decodes the upper bit of RegAddr for selecting the active hardware. The register file inside UserReg is 32-bit bus size, so write byte enable (RegWrByteEn) is not used. To write hardware registers, the CPU must use 32-bit pointer to place 32-bit valid value on the write data bus.

To read register, two-step multiplexer is designed to select the read data within each address area. The lower bit of RegAddr is applied in each Register area to select the data. Next, the address decoder uses the upper bit to select the read data from each area for returning to CPU. Totally, the latency of read data is equal to two clock cycles, so RegRdValid is created by RegRdReq with asserting two D Flip-flips. More details of the address mapping within UserReg module is 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_REG)	[31:0]: Input to be start address as 512-byte unit (UserAddr[31:0] of dgIF typeS)
BA+0x0004	User Address (High) Reg (USRADRH_REG)	[15:0]: Input to be start address as 512-byte unit (UserAddr[47:32] of dgIF typeS)
BA+0x0008	User Length (Low) Reg (USRLENL_REG)	[31:0]: Input to be transfer length as 512-byte unit (UserLen[31:0] of dgIF typeS)
BA+0x000C	User Length (High) Reg (USRLENH_REG)	[15:0]: Input to be transfer length as 512-byte unit (UserLen[47:32] of dgIF typeS)
BA+0x0010	User Command Reg (USRCMD_REG)	[2:0]: Input to be user command (UserCmd of dgIF typeS for NVMe-IP) "000": Identify, "001": Shutdown, "010": Write SSD, "011": Read SSD, "100": SMART, "110": Flush, "101"/"111": Reserved When this register is written, the command request is sent to NVMe-IP to start the operation.
BA+0x0014	Test Pattern Reg (PATTSEL_REG)	[2:0]: Select test pattern "000"-Increment, "001"-Decrement, "010"-All 0, "011"-All 1, "100"-LFSR
BA+0x0020	NVMe Timeout Reg (NVMTIMEOUT_REG)	[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_REG)	[0]: UserBusy of dgIF typeS ('0': Idle, '1': Busy) [1]: UserError of dgIF typeS ('0': Normal, '1': Error) [2]: Data verification fail ('0': Normal, '1': Error)
BA+0x0104	Total disk size (Low) Reg (LBASIZEL_REG)	[31:0]: Mapped to LBASize[31:0] of NVMe-IP
BA+0x0108	Total disk size (High) Reg (LBASIZEH_REG)	[15:0]: Mapped to LBASize[47:32] of NVMe-IP [31]: Mapped to LBAMode of NVMe-IP
BA+0x010C	User Error Type Reg (USRERRTYPE_REG)	[31:0]: Mapped to UserErrorType[31:0] of NVMe-IP to show error status
BA+0x0110	PCIe Status Reg (PCISTS_REG)	[0]: PCIe linkup status from PCIe hard IP ('0': No linkup, '1': linkup) [3:2]: PCIe link speed from PCIe hard IP ("00": Not linkup, "01": PCIe Gen1, "10": PCIe Gen2, "11": PCIe Gen3) [7:4]: PCIe link width status from PCIe hard IP ("0001": 1-lane, "0010": 2-lane, "0100": 4-lane, "1000": 8-lane) [13:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in Avalon-ST PCIe Hard IP datasheet
BA+0x0114	Completion Status Reg (COMPSTS_REG)	[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_REG)	[31:0]: Mapped to NVMeCAPReg[31:0] of NVMe-IP
BA+0x011C	NVMe IP Test pin Reg (NVMTESTPIN_REG)	[31:0]: Mapped to TestPin[31:0] of NVMe-IP
BA+0x0130	Expected value Word0 Reg (EXPPATW0_REG)	[31:0]: Bit[31:0] of the expected data at the 1st failure data in Read command
BA+0x0134	Expected value Word1 Reg (EXPPATW1_REG)	[31:0]: Bit[63:32] of the expected data at the 1st failure data in Read command
BA+0x0138	Expected value Word2 Reg (EXPPATW2_REG)	[31:0]: Bit[95:64] of the expected data at the 1st failure data in Read command
BA+0x013C	Expected value Word3 Reg (EXPPATW3_REG)	[31:0]: Bit[127:96] of the expected data at the 1st failure data in Read command

Address	Register Name	Description
Wr/Rd	(Label in the "nvmeiptest.c")	
0x0100 – 0x01FF: Status signals of NVMe-IP and TestGen (Read access only)		
BA+0x0140	Read value Word4 Reg (EXPPATW4_REG)	[31:0]: Bit[159:128] of the expected data at the 1st failure data in Read command
BA+0x0140	Read value Word5 Reg (EXPPATW5_REG)	[31:0]: Bit[191:160] of the expected data at the 1st failure data in Read command
BA+0x0144	Read value Word6 Reg (EXPPATW6_REG)	[31:0]: Bit[223:192] of the expected data at the 1st failure data in Read command
BA+0x0148	Read value Word7 Reg (EXPPATW7_REG)	[31:0]: Bit[255:224] of the expected data at the 1st failure data in Read command
BA+0x0150	Read value Word0 Reg (RDPATW0_REG)	[31:0]: Bit[31:0] of the read data at the 1st failure data in Read command
BA+0x0154	Read value Word1 Reg (RDPATW1_REG)	[31:0]: Bit[63:32] of the read data at the 1st failure data in Read command
BA+0x0158	Read value Word2 Reg (RDPATW2_REG)	[31:0]: Bit[95:64] of the read data at the 1st failure data in Read command
BA+0x015C	Read value Word3 Reg (RDPATW3_REG)	[31:0]: Bit[127:96] of the read data at the 1st failure data in Read command
BA+0x0160	Read value Word4 Reg (RDPATW4_REG)	[31:0]: Bit[159:128] of the read data at the 1st failure data in Read command
BA+0x0164	Read value Word5 Reg (RDPATW5_REG)	[31:0]: Bit[191:160] of the read data at the 1st failure data in Read command
BA+0x0168	Read value Word6 Reg (RDPATW6_REG)	[31:0]: Bit[223:192] of the read data at the 1st failure data in Read command
BA+0x016C	Read value Word7 Reg (RDPATW7_REG)	[31:0]: Bit[255:224] of the read data at the 1st failure data in Read command
BA+0x0170	Data Failure Address(Low) Reg (RDFAILNOL_REG)	[31:0]: Bit[31:0] of the byte address of the 1st failure data in Read command
BA+0x0174	Data Failure Address(High) Reg (RDFAILNOH_REG)	[24:0]: Bit[56:32] of the byte address of the 1st failure data in Read command
BA+0x0178	Current test byte (Low) Reg (CURTESTSIZEL_REG)	[31:0]: Bit[31:0] of the current test data size in TestGen module
BA+0x017C	Current test byte (High) Reg (CURTESTSIZEH_REG)	[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	Custom Submission Queue Reg	[31:0]: Submission queue entry of SMART and Flush command. Input to be CtmSubmDW0-DW15 of NVMe-IP.
Wr	(CTMSUBMQ_REG)	0x200: DW0, 0x204: DW1, …, 0x23C: DW15
BA+0x0300 – BA+0x030F	Custom Completion Queue Reg	[31:0]: CtmCompDW0-DW3 output from NVMe-IP. 0x300: DW0, 0x304: DW1, …, 0x30C: DW3
Rd	(CTMCOMPQ_REG)	
BA+0x0800	IP Version Reg	[31:0]: Mapped to IPVersion[31:0] of NVMe-IP
Rd	(IPVERSION_REG)	
BA+0x2000 – BA+0x2FFF	Identify Controller Data	4Kbyte Identify controller data structure
Rd	(IDENCTRL_REG)	
BA+0x3000 – BA+0x3FFF	Identify Namespace Data	4Kbyte Identify Namespace Data Structure
Rd	(IDENNAME_REG)	
BA+0x4000 – BA+0x5FFF	Custom command Ram	Connect to 8K byte CtmRAM interface. Used to store 512-byte data output from SMART command.
Wr/Rd	(CTMRAM_REG)	

3   CPU Firmware

3.1   Test firmware (nvmeiptest.c)

After system boot-up, CPU runs following steps to finish the initialization process.
1) CPU initializes JTAG UART and Timer parameters.
2) CPU waits until PCIe connection links up (PCISTS_REG[0]='1').
3) CPU waits until NVMe-IP completes initialization process (USRSTS_REG[0]='0'). The error message is displayed and the process stops when some errors are found.
4) CPU displays PCIe link status (the number of PCIe lanes and the PCIe speed) by reading PCISTS_REG[7:2].
5) CPU displays the main menu. There are six menus for running six commands of NVMe-IP, i.e. Identify, Write, Read, SMART, Flush and Shutdown.

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

3.1.1   Identify Command

The step to operate Identify command is described as follows.
1) Set USRCMD_REG[2:0]="000". Next, Test logic generates command and asserts command request to NVMe-IP. After that, Busy flag (USRSTS_REG[0]) changes from '0' to '1'.
2) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_REG[1:0].

   Bit[0] is de-asserted to '0' when command is completed. After that, the data from Identify command is stored to IdenRAM.
   Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE_REG[31:0]. Finally, the process is stopped.

3) After busy flag (USRSTS_REG[0]) is de-asserted to '0', CPU displays some information decoded from IdenRAM (IDENCTRL_REG) such as SSD model name and the information from NVMe-IP output such as SSD capacity and LBA unit size (LBASIZE_REG).

3.1.2 Write/Read Command

The step to operate Write/Read command is described as follows.

1) Receive start address, transfer length and test pattern from JTAG UART. If some inputs are invalid, the operation is cancelled.

   Note: If LBA unit size = 4 Kbyte, start address and transfer length must be aligned to 8.

2) Get all inputs and set to USRADRL/H_REG, USRLENL/H_REG and PATTSEL_REG.

3) Set USRCMD_REG[2:0]="010" for Write command or "011" for Read command. After that, the new command request is sent to NVMe-IP for running Write or Read command. Busy flag (USRSTS_REG[0]) changes from '0' to '1'.

4) CPU waits until the operation is completed or some errors (except verification error) are found by monitoring USRSTS_REG[2:0].

   Bit[0] is de-asserted to '0' when command is completed.
   Bit[1] is asserted to '1' when some errors are detected. After that, the error message is displayed on the console to show the error details. Finally, the process is stopped.
   Bit[2] is asserted to '1' when data verification is failed. The verification error message is displayed on the console to show the error details. In this condition, CPU is still running until the operation is done or user presses any key(s) to cancel operation.

   When the operation does not finish, current transfer size read from CURTESTSIZE_REG is displayed every second.

5) After busy flag (USRSTS_REG[0]) is de-asserted to '0', CPU calculates and displays the test result on the console, i.e. total time usage, total transfer size and transfer speed.

3.1.3 SMART Command,

The step to operate SMART command is described as follows.

1) Set 16-Dword of Submission queue entry (CTMSUBMQ_REG) to be SMART command value.

2) Set USRCMD_REG[2:0]="100". Next, Test logic generates command and asserts the request to NVMe-IP. After that, Busy flag (USRSTS_REG[0]) changes from '0' to '1'.

3) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_REG[1:0].

   Bit[0] is de-asserted to '0' when command is completed. After that, the data from SMART command is stored to CtmRAM.
   Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE_REG[31:0]. Finally, the process is stopped.

4) After busy flag (USRSTS_REG[0]) is de-asserted to '0', CPU decodes SMART command information from CtmRAM (CTMRAM_REG), i.e. Temperature, Total Data Read, Total Data Written, Power On Cycles, Power On Hours and Number of Unsafe Shutdown.

   More details of SMART log are described in NVM Express Specification.
   https://nvmexpress.org/resources/specifications/

3.1.4 Flush Command

The step to operate Flush command is described as follows.

1) Set 16-Dword of Submission queue entry (CTMSUBMQ_REG) to be Flush command value.
2) Set USRCMD_REG[2:0]="110". Next, Test logic generates command and asserts the request to NVMe-IP. After that, Busy flag (USRSTS_REG[0]) changes from '0' to '1'.
3) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_REG[1:0].

Bit[0] is de-asserted to '0' when command is completed. After that, CPU goes back to the main menu.
Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE_REG[31:0]. Finally, the process is stopped.

3.1.5 Shutdown Command

The step to operate Shutdown command is described as follows.

1) Set USRCMD_REG[2:0]="001". Next, Test logic generates command and asserts the request to NVMe-IP. After that, Busy flag (USRSTS_REG[0]) changes from '0' to '1'.
2) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_REG value.

Bit[0] is de-asserted to '0' when command is completed. After that, the CPU goes to the next step.
Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE_REG[31:0]. Finally, the process is stopped.

3) After busy flag (USRSTS_REG[0]) is de-asserted to '0', the SSD and NVMe-IP change to inactive status. The CPU cannot receive the new command from user. The user must power off the test 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	Run SMART command or Flush command, following in topic 3.1.3 (SMART Command,) and 3.1.4 (Flush Command).

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

unsigned long long get_cursize(void)	
Parameters	None
Return value	Read value of CURTESTSIZEH/L_REG
Description	Read CURTESTSIZEH/L_REG and return read value as function result.

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	Run Identify command, following in topic 3.1.1. (Identify Command).

void show_error(void)	
Parameters	None
Return value	None
Description	Read USRERRTYPE_REG, decode the error flag and display error message following the error flag.

void show_pciestat(void)	
Parameters	None
Return value	None
Description	Read PCIESTS_REG until the read value from two read times is stable. After that, display the read value on the console.

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

void show_size(unsigned long long size_input)	
Parameters	size_input: transfer size to display on the console
Return value	None
Description	Calculate and display the input value in MByte, GByte, or TByte unit

void show_smart_hex(unsigned char *char_ptr16B)	
Parameters	*char_ptr16B
Return value	None
Description	Display SMART data as hexadecimal unit.

void show_smart_raw(unsigned char *char_ptr16B)	
Parameters	*char_ptr16B
Return value	None
Description	Display SMART data as decimal unit when the input value is less than 4 MB. Otherwise, display overflow message.

void show_smart_unit(unsigned char *char_ptr16B)	
Parameters	*char_ptr16B
Return value	None
Description	Display SMART data as GB or TB unit. When the input value is more than limit (500 PB), overflow message is displayed instead.

void show_vererr(void)	
Parameters	None
Return value	None
Description	Read RDFAILNOL/H_REG (error byte address), EXPPATW0-7_REG (expected value), and RDPATW0-7_REG (read value) to display verification error details on the console.

void shutdown_dev(void)	
Parameters	None
Return value	None
Description	Run Shutdown command, following in topic 3.1.5. (Shutdown Command)

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

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	Run Write command or Read command, following in topic 3.1.2. (Write/Read Command)

4   Example Test Result

The example test result when running demo system by using 512 GB Samsung 970 Pro is shown in Figure 4-1.
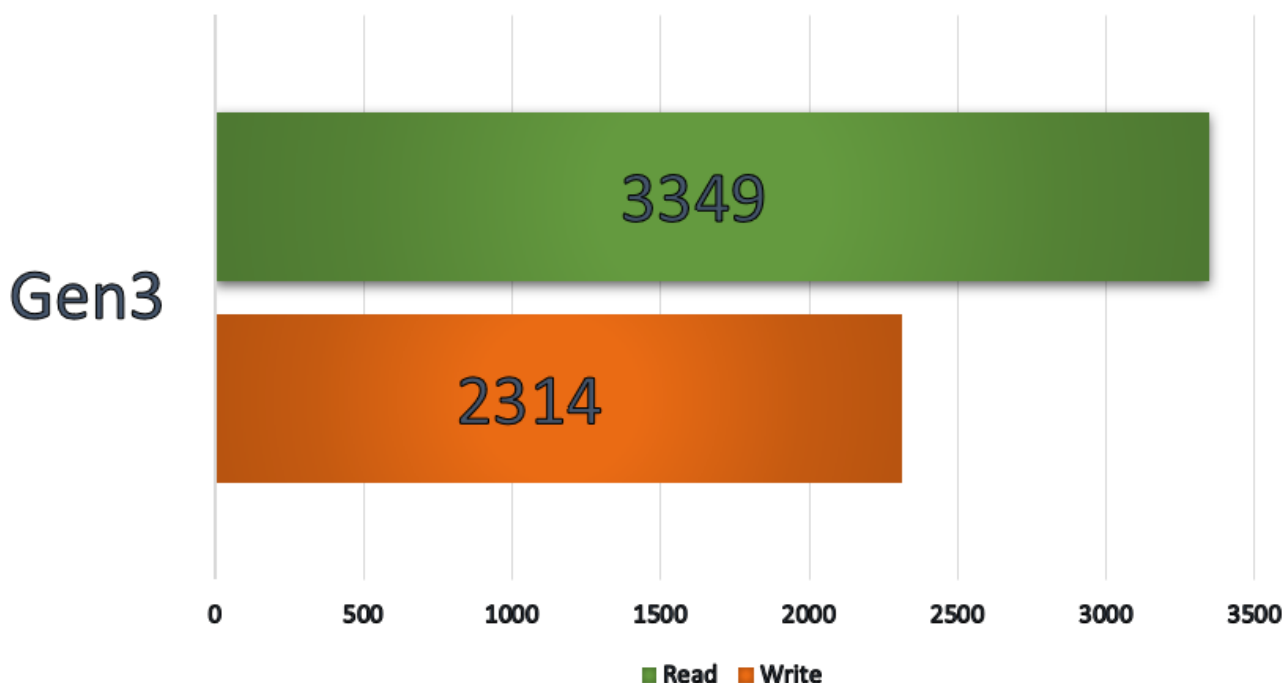

Figure 4-1 Test Performance of NVMe-IP demo by using Samsung 970 Pro SSD (Mbyte/s)

By using PCIe Gen3 on Stratix10 GX board, write performance is about 2300 Mbyte/sec and read performance is about 3300 Mbyte/sec.

5 Revision History

Revision	Date	Description
1.0	23-Sep-20	Initial Release

Copyright: 2020 Design Gateway Co,Ltd.