NVMe-IP reference design manual

Rev3.3 25-Mar-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 improves 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/.

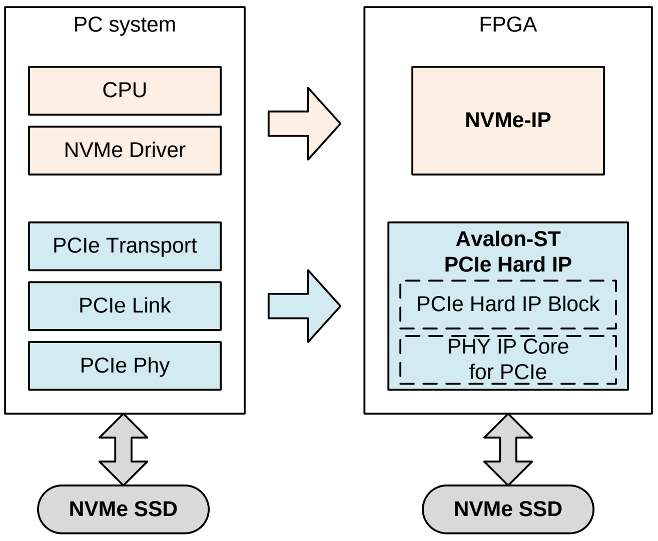

Figure 1-1 NVMe protocol layer

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. So, user can access NVMe SSD without including any processor and driver by using NVMe-IP in FPGA board.

2 Hardware overview

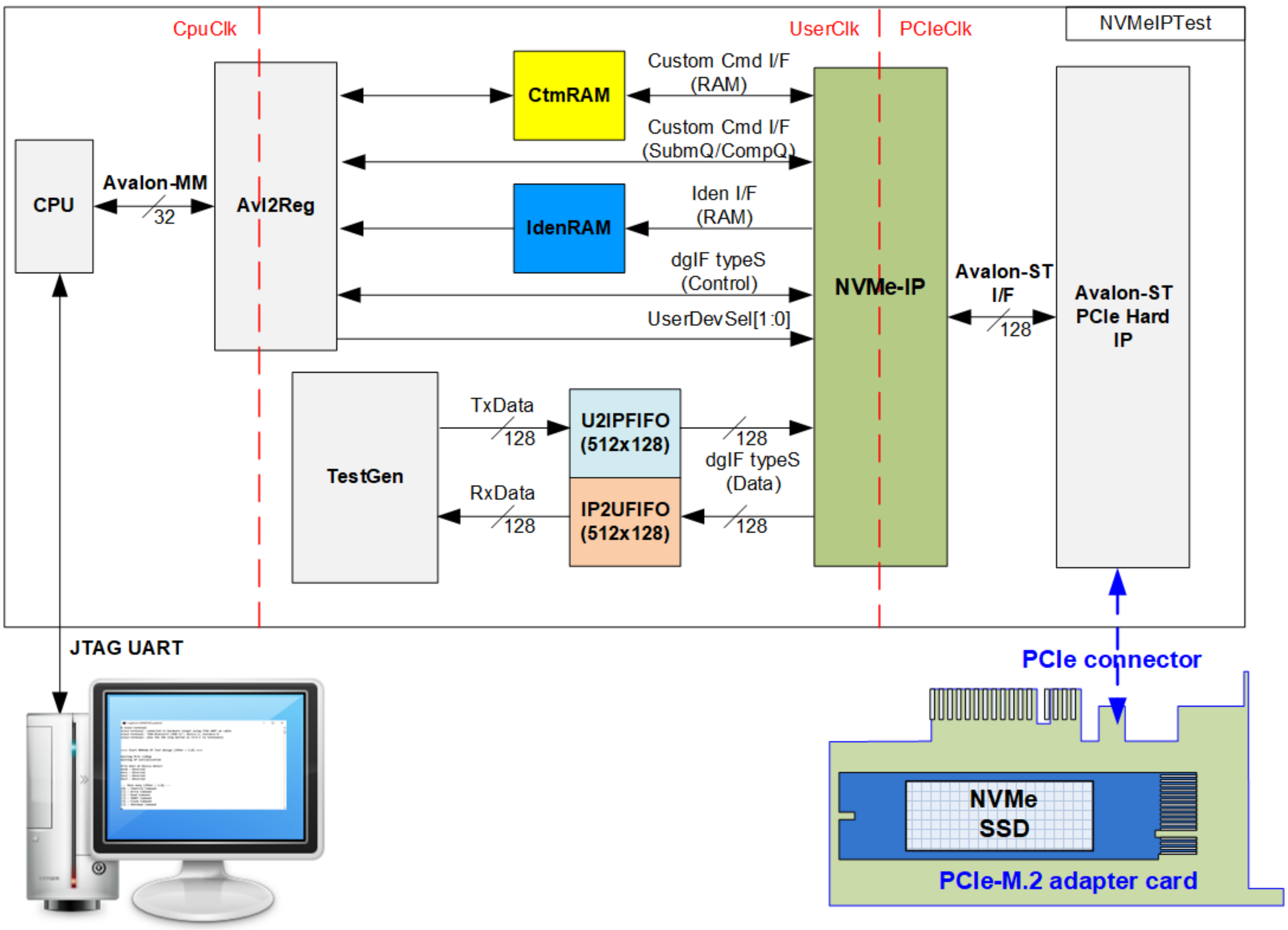

Figure 2-1 NVMe-IP demo hardware

The hardware system can be split into three groups, i.e. TestGen, NVMe, and CPU. TestGen is the test logic to generate test data stream for NVMe-IP through U2IPFIFO. Also, TestGen reads data stream output from NVMe-IP through IP2UFIFO, and then verify it. NVMe includes the NVMe-IP and the PCIe hard IP (Avalon-ST PCIe hard IP). NVMe supports to access NVMe SSD directly. CPU and Avl2Reg are designed to interface with user through JTAG UART. User can set command and the test parameters through JTAG UART. Also, the current status of the test hardware is monitored by user through JTAG UART. The CPU firmware must be implemented to control the sequence for operating each command.

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. PCIeClk is the clock output from PCIe hard IP to synchronous with data stream of 128-bit Avalon-ST interface. When the PCIe hard IP is set to 4 lane PCIe Gen3, PCIeClk frequency is equal to 250 MHz (125 MHz is applied for PCIe Gen2). UserClk is the example user clock domain which is independent from the other clock domains. So, UserClk is the main clock domain 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. In this reference design, UserClk is equal to 275 MHz for PCIe Gen3 speed.

There are six memories implemented in the test system, i.e. CtmRAM, IdenRAM, TxFIFO, RxFIFO, U2IPFIFO, and IP2UFIFO.

CtmRAM stores returned data of each SSD from SMART command while IdenRAM stores returned data from Identify command. When running SMART command or Identify command, the returned data from the command is decoded by CPU which is read through Avl2Reg module by using Avalon-MM bus.

U2IPFIFO and IP2UFIFO are connected between TestGen and NVMe-IP for storing data when running Write command and Read command respectively. TestGen is designed to monitor flow control signal of U2IPFIFO and IP 2UFIFO to be always read and write data when the FIFO is ready. The FIFO depth is set to 512 x 128 bit. More details of the hardware are described as follows.

2.1 TestGen

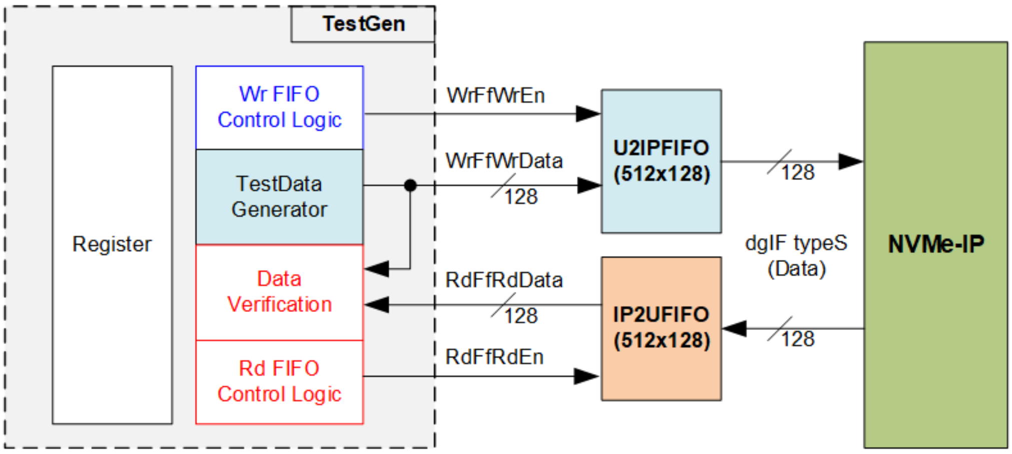

Figure 2-2 TestGen interface

TestGen module is the test logic to send test data to NVMe-IP through U2IPFIFO when operating Write command. Also, the test data is fed to be the expected value to verify the read data from NVMe-IP through IP2UFIFO when operating Read command. Control logic asserts Write enable and Read enable to '1' when the FIFOs are ready. Data bandwidth of TestGen is matched to NVMe-IP by running at the same clock and using same data bus size, 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. So, 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.

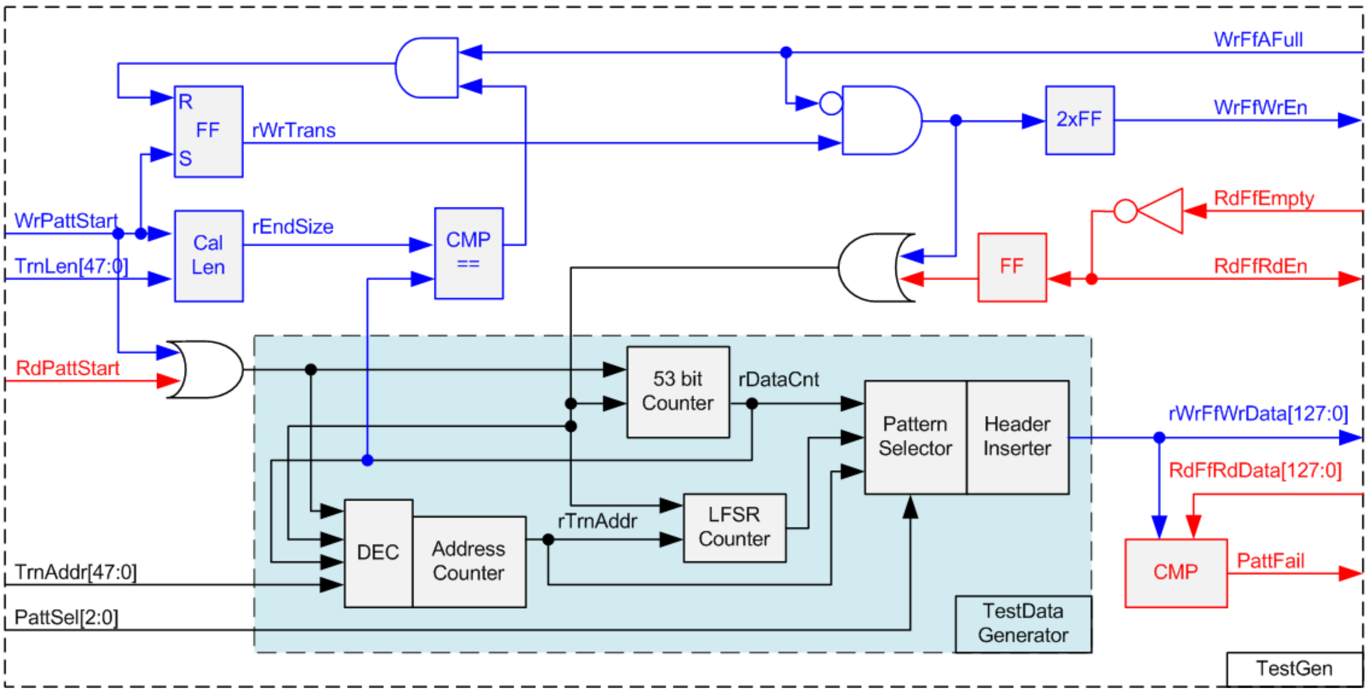

Figure 2-3 TestGen hardware

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. All zero and all one are fixed value to select test data through Pattern Selector. While 32-bit incremental data is designed by using 53-bit counter. The decremental data can be designed by connecting NOT logic to increment data. The LFSR pattern is designed by using LFSR counter. The equation of LFSR is x^31 + x^21 + x + 1. Data bus size of TestGen is 128-bit, so four 32-bit LFSR data must be generated within one clock. The logic to design LFSR must use look-ahead style to generate four LFSR data in the same clock.

When creating all zero or all one pattern, every bit of data is fixed 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, 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).

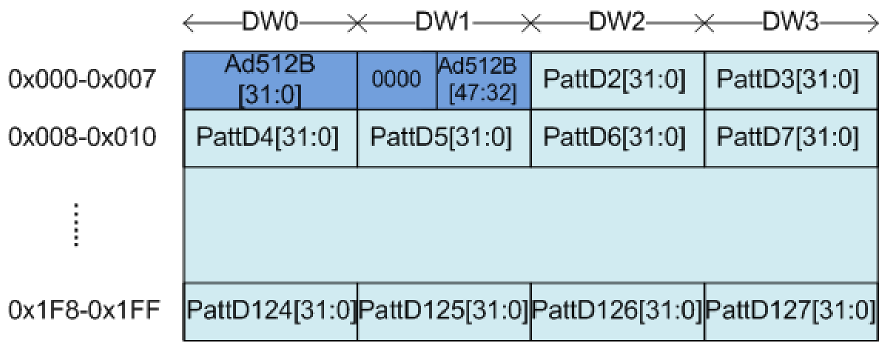

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

64-bit 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

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.

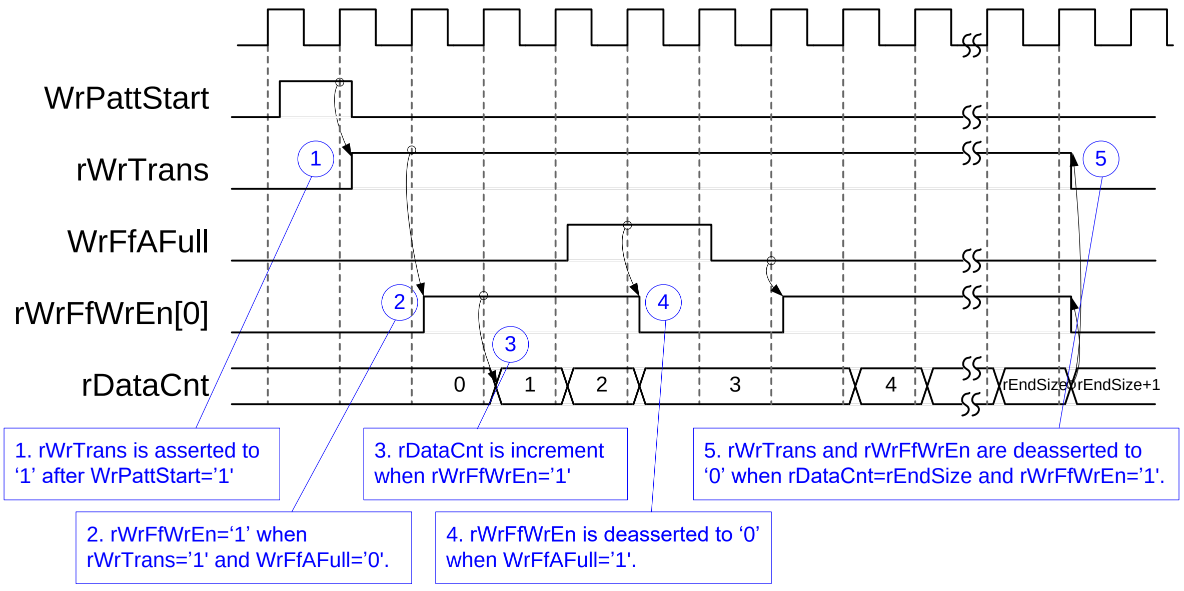
WrPattStart
rWrTrans
WrFfAFull
rWrFfWrEn[0]
rDataCnt
0
1
2
3
4
rEndSize
rEndSize+1
1. rWrTrans is asserted to '1' after WrPattStart='1'
2. rWrFfWrEn='1' when rWrTrans='1' and WrFfAFull='0'.
3. rDataCnt is increment when rWrFfWrEn='1'
4. rWrFfWrEn is deasserted to '0' when WrFfAFull='1'.
5. rWrTrans and rWrFfWrEn are deasserted to '0' when rDataCnt=rEndSize and rWrFfWrEn='1'.

Figure 2-5 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 stop data generating.

When running read operation, read enable of FIFO is only controlled by empty flag of FIFO. Data is read when FIFO has the data. It does not wait start flag for beginning the read operation and also not monitor total count for stopping the read operation. When the read enable is asserted to '1', the address counter is increased for generating the header of expected value and for checking total read data in read operation.

2.2 NVMe

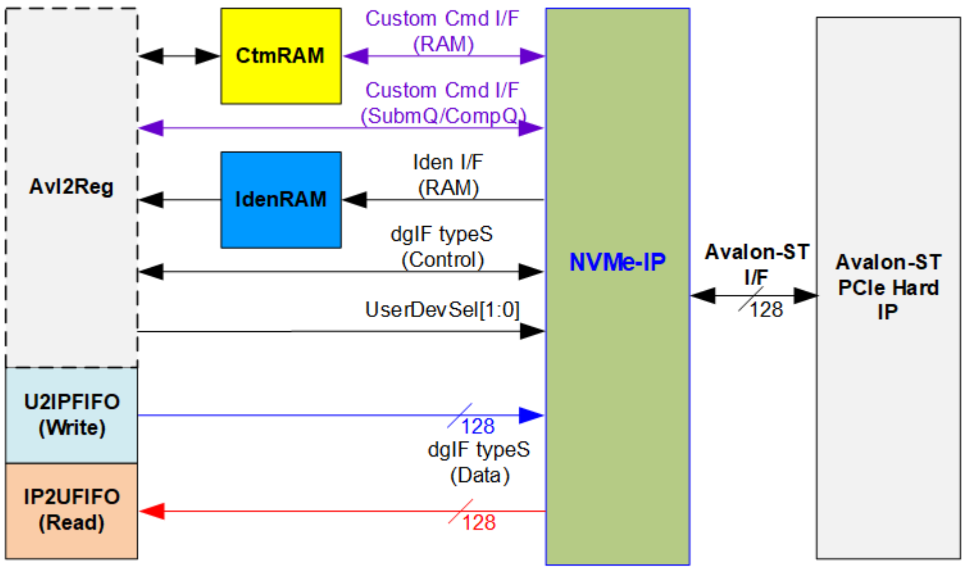

Figure 2-6 NVMe hardware

Figure 2-6 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 parameters through dgIF typeS or custom command interface, depending on the command. Custom command interface is applied when the command is SMART command or Flush command. Otherwise, dgIF typeS is applied.

The data interface of NVMe-IP has four signal groups, i.e. FIFO input interface (dgIF types), FIFO output interface (dgIF types), custom command RAM interface, and Identify interface. Data bus width of all signal groups is 128-bit. The custom command RAM interface is bi-directional interface while others are one directional interface. In the reference design, the custom command RAM interface is used to transfer data of SMART command from NVMe-IP to Avl2Reg only. Another direction is not used.

2.2.1 NVMe-IP

NVMe-IP implements NVMe protocol of the host side to access NVMe SSD. User interface is designed by using dgIF typeS interface. NVMe-IP connects with Avalon-ST PCIe Hard IP (Hard IP in Intel FPGA device). 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 PCIe Hard IP

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

ArriaV Avalon-ST Interface for PCIe Solutions User Guide
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_a5_pcie_avst.pdf
Stratix V Avalon-ST Interface for PCIe Solutions User Guide
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_s5_pcie_avst.pdf

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

2.2.3 Two-port RAM

Two of 2-Port RAMs are implemented in the reference design to store data from Identify command and SMART command.

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 128-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 2-7.

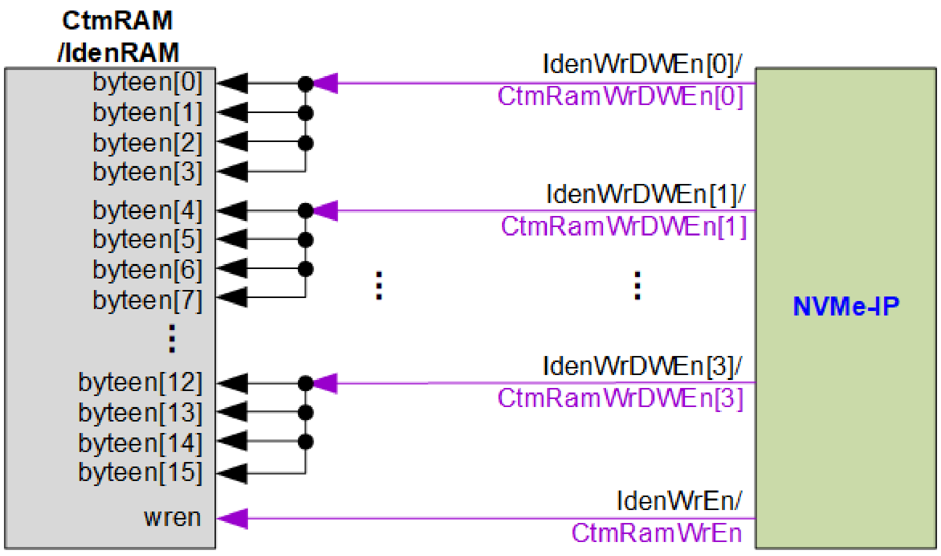

Figure 2-7 Word enable to be byte write enable connection

Bit[0] of WrDWEn is fed to bit[3:0] of IdenRAM byte write enable. Bit[1] of WrDWEn is applied to generate bit[7:4] of IdenRAM write byte enable, and so on.

Similar to IdenRAM, CtmRAM supports byte write enable and double word enable of custom interface is connected to 4-bit write byte enable. Unlike IdenRAM, CtmRAM is true dual-port RAM (two read ports and two write ports) to allow NVMe-IP and Avl2Reg writing and reading data. The data width of both interfaces generated by IP catalog is 128-bit. The small logic is designed to convert data bus from 128-bit to be 32-bit for Avl2Reg access in write and read transaction. Two lower address bits are applied to select the active double word from 128-bit data. CtmRAM is implemented by 8Kbyte RAM for the future support though the data size of SMART command is 512-byte.

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. To control and monitor the test logic of NVMe-IP, the test logic is connected to CPU as a peripheral on 32-bit Avalon-MM bus. CPU assigns the different base address and the address range for each peripheral.

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 writing and reading the register. Avl2Reg module is designed to connect the CPU system as shown in Figure 2-8.

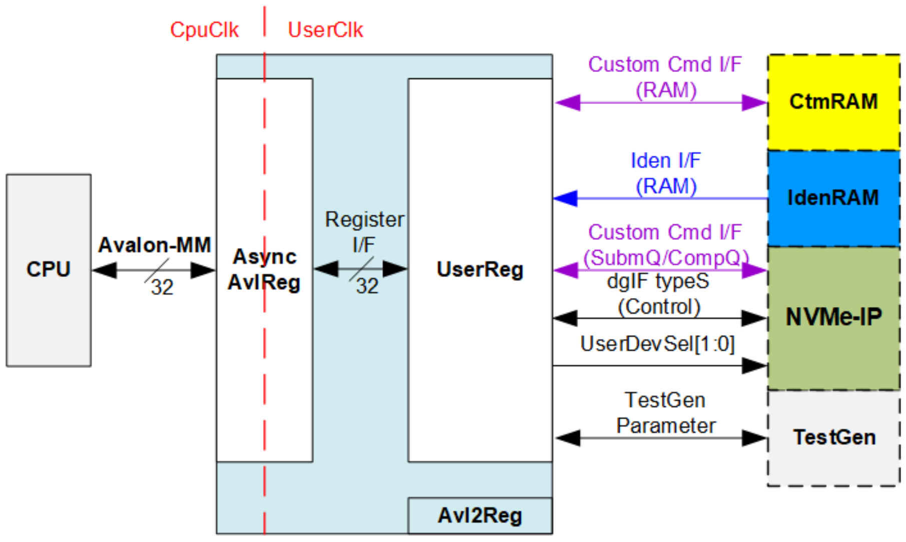

Figure 2-8 CPU and peripherals hardware

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 to control the other modules, i.e. CtmRAM, IdenRAM, NVMe-IP, and TestGen. More details of AsyncAvlReg and UserReg are described as follows.

2.3.1 AsyncAvlReg

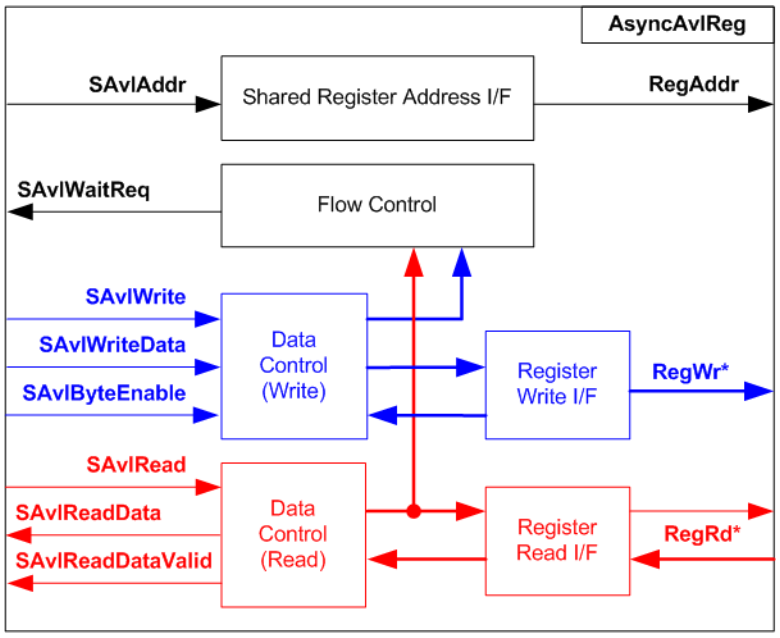

Figure 2-9 AsyncAvlReg Interface

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 logic inside AsyncAvlReg is split into three groups, i.e. Write control logic, Read control logic, and Flow control logic. Flow control logic to control SAvlWaitReq is designed to hold the next request from Avalon-MM interface while the current request is operating. Write control I/F and Write data I/F of Avalon-MM bus are latched and transferred as Write register. While Read control I/F and Read data I/F of Avalon-MM bus are latched and transferred as Read register. Address I/F of Avalon-MM is latched and transferred to Address register interface as well.

The simple register interface is designed to compatible to general RAM interface for write transaction. The read transaction of the register interface is little modified from RAM interface by adding RdReq signal. 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 2-10

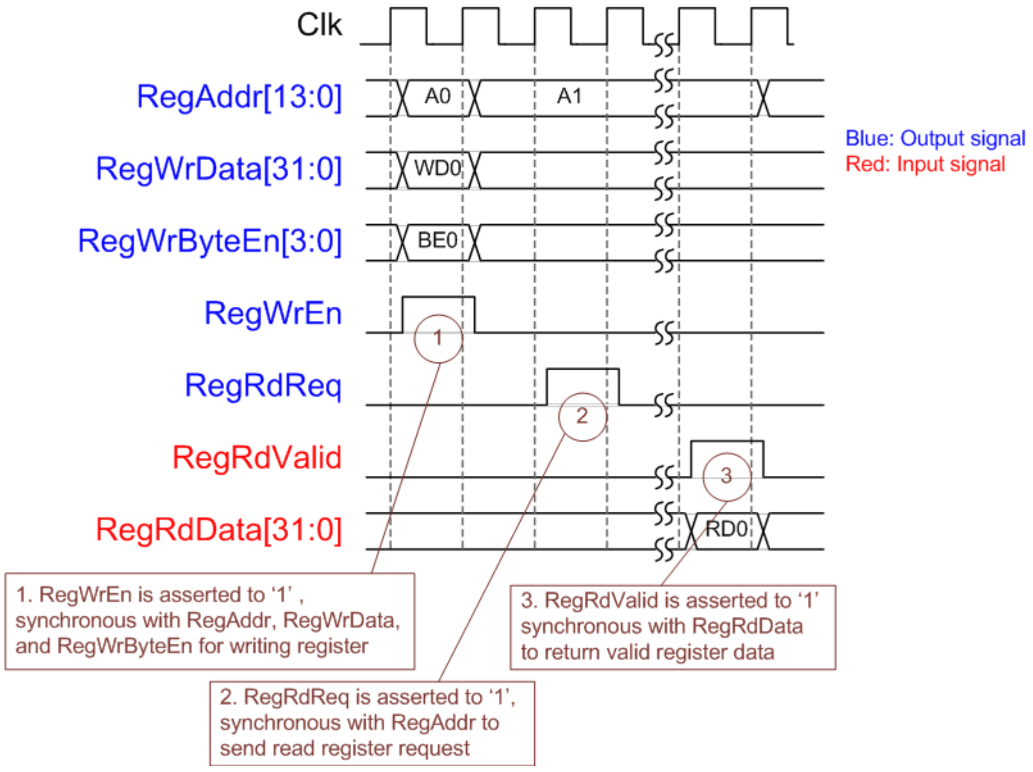
Clk
RegAddr[13:0]
A0
A1
RegWrData[31:0]
WD0
RegWrByteEn[3:0]
BE0
RegWrEn
RegRdReq
RegRdValid
RegRdData[31:0]
RD0
1
2
3
Blue: Output signal
Red: Input signal
1. RegWrEn is asserted to '1' , synchronous with RegAddr, RegWrData, and RegWrByteEn for writing register
2. RegRdReq is asserted to '1', synchronous with RegAddr to send read register request
3. RegRdValid is asserted to '1' synchronous with RegRdData to return valid register data

Figure 2-10 Register interface timing diagram

1) To write register, the timing diagram is same as general 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 bit to be the byte data valid, i.e. bit[0] for RegWrData[7:0], bit[1] for RegWrData[15:8], and so on.
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.
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

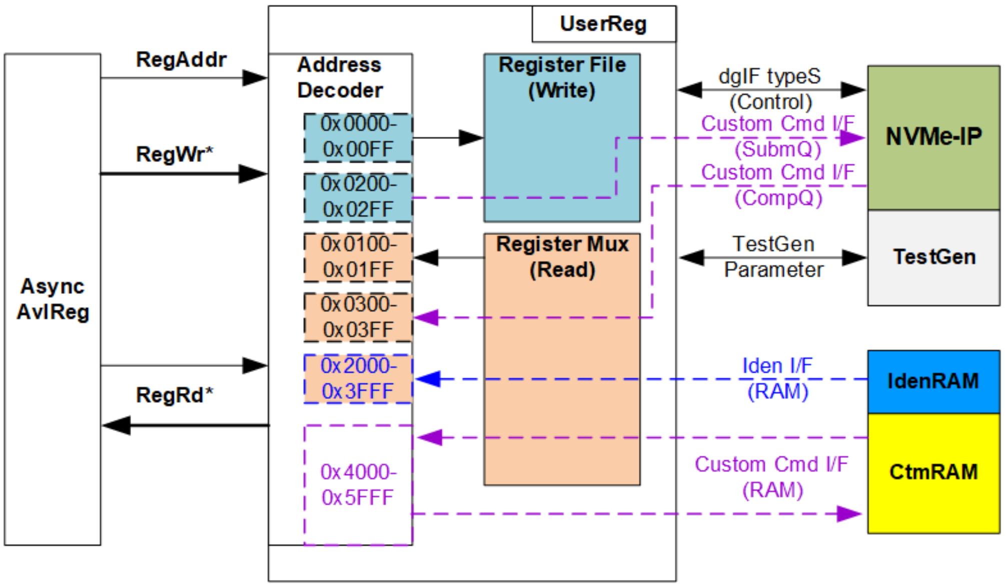

Figure 2-11 UserReg Interface

The address range to map to UserReg is split into six areas, as shown in Figure 2-11.

1) 0x0000 – 0x00FF: mapped to set the test parameters of NVMe-IP and TestGen. This area is write access only.
2) 0x0200 – 0x02FF: mapped to set the test parameters of custom command interface (NVMe-IP). This area is write access only.
3) 0x0100 – 0x01FF: mapped to read the status 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 set the parameters in the hardware, the CPU must use 32-bit pointer to force 32-bit valid value of the write data.

To read register, two step multiplexers are designed. Register Mux is the data multiplexer to select the read data within each address area. The lower bit of RegAddr is applied in the Register Mux. 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]: Test pattern select "000"-Increment, "001"-Decrement, "010"-All 0, "011"-All 1, "100"-LFSR
BA+0x0020	NVMe Timeout Reg (NVMTIMEOUT_REG)	[31:0]: Timeout value of NVMe-IP (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]: Total capacity of SSD in 512-byte unit (LBASize[31:0] of dgIF typeS)
BA+0x0108	Total disk size (High) Reg (LBASIZEH_REG)	[15:0]: Total capacity of SSD in 512-byte unit (LBASize[47:32] of dgIF typeS) [31]: LBA unit ('0':512Byte, '1':4KByte), mapped from LBAMode of NVMe-IP
BA+0x010C	User Error Type Reg (USRERRTYPE_REG)	[31:0]: User error status (UserErrorType[31:0] of dgIF typeS)
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) [12:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in Avalon-ST PCIe Hard IP datasheet
BA+0x0114	Completion Status Reg (COMPSTS_REG)	[15:0]: Status from Admin completion (AdmCompStatus[15:0] of NVMe-IP) [31:16]: Status from I/O completion (IOCompStatus[15:0] of NVMe-IP)
BA+0x0118	NVMe CAP Reg (NVMCAP_REG)	[31:0]: NVMeCAPReg[31:0] output from NVMe-IP
BA+0x011C	NVMe IP Test pin Reg (NVMTESTPIN_REG)	[31:0]: TestPin[31:0] output from 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 Wr/Rd	Register Name (Label in the "nvmeiptest.c")	Description
0x0100 – 0x01FF: Status signals of NVMe-IP and TestGen (Read access only)		
BA+0x0140 Rd	Read value Word0 Reg (RDPATW0_REG)	[31:0]: Bit[31:0] of the read data at the 1st failure data in Read command
BA+0x0144 Rd	Read value Word1 Reg (RDPATW1_REG)	[31:0]: Bit[63:32] of the read data at the 1st failure data in Read command
BA+0x0148 Rd	Read value Word2 Reg (RDPATW2_REG)	[31:0]: Bit[95:64] of the read data at the 1st failure data in Read command
BA+0x014C Rd	Read value Word3 Reg (RDPATW3_REG)	[31:0]: Bit[127:96] of the read data at the 1st failure data in Read command
BA+0x0150 Rd	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+0x0154 Rd	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+0x0158 Rd	Current test byte (Low) Reg (CURTESTSIZEL_REG)	[31:0]: Bit[31:0] of the current test data size in TestGen module
BA+0x015C Rd	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 Wr	Custom Submission Queue Reg (CTMSUBMQ_REG)	[31:0]: Submission queue entry of SMART and Flush command. Input to be CtmSubmDW0-DW15 of NVMe-IP. 0x200: DW0, 0x204: DW1, …, 0x23C: DW15
BA+0x0300 – BA+0x030F Rd	Custom Completion Queue Reg (CTMCOMPQ_REG)	[31:0]: CtmCompDW0-DW3 output from NVMe-IP. 0x300: DW0, 0x304: DW1, …, 0x30C: DW3
BA+0x0800 Rd	IP Version Reg (IPVERSION_REG)	[31:0]: IP version number (IPVersion[31:0] of NVMe-IP)
BA+0x2000 – BA+0x2FFF Rd	Identify Controller Data (IDENCTRL_REG)	4Kbyte Identify controller data structure
BA+0x3000 – BA+0x3FFF Rd	Identify Namespace Data (IDENNAME_REG)	4Kbyte Identify Namespace Data Structure
BA+0x4000 – BA+0x5FFF Wr/Rd	Custom command Ram (CTMRAM_REG)	Connect to 8K byte CtmRAM interface. Used to store 512-byte data output from SMART command.

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 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 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 (read 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 from IdenRAM (IDENCTRL_REG) such as SSD model name and the information from NVMe-IP output, i.e. 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 input parameter from user through JTAG UART, i.e. start address, transfer length, and test pattern. When 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) Set the input parameters to hardware registers, i.e. USRADRL/H_REG, USRLENL/H_REG, 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. The error message is displayed on the console to show the error details and 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 Dwords of Submission queue entry (CTMSUBMQ_REG) to be SMART command value.
2) Set USRCMD_REG[2:0]=”100”. Next, Test logic generates command and 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 (read 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 Dwords of Submission queue entry (CTMSUBMQ_REG) to be Flush command value.
2) Set USRCMD_REG[2:0]=”110”. Next, Test logic generates command and 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 (read 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 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. If the command is completed, CPU does not receive the new command from user. The user must power off the test system.

   Bit[1] is asserted to ‘1’ when some errors are detected. The error message is displayed on the console to show the error details (read from USRERRTYPE_REG[31:0]). Finally, the process is stopped.

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 start 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 shutdown_dev(void)	
Parameters	None
Return value	None
Description	Run Shutdown command, following in topic 3.1.5. (Shutdown 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 of each channel.

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	Read CURTESTSIZEL/H _REG to display total transfer size. Read the global parameters (timer_val and timer_upper_val) and calculate total time usage to display in usec, msec, or sec unit. Finally, transfer performance is calculated and displayed on 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), EXPPATW0-3_REG (expected value), and RDPATW0-3_REG (read value) to display verification error details on the console.

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 960 Pro is shown in Figure 4-1.

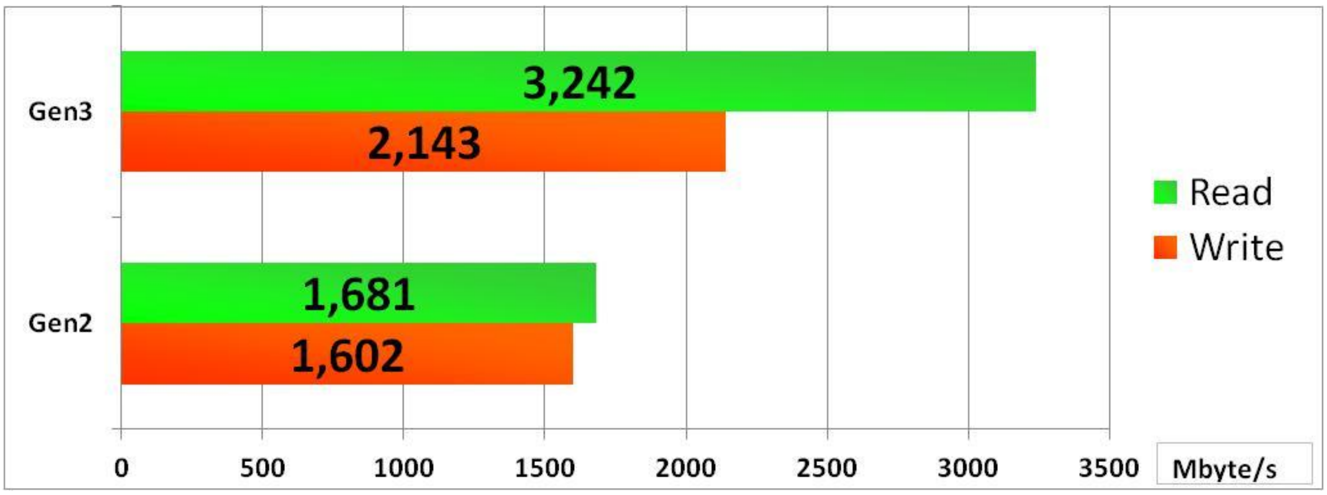

Figure 4-1 Test Performance of NVMe IP demo by using Samsung 960 Pro SSD

By using PCIe Gen3 on Arria10 SoC board, write performance is about 2100 Mbyte/sec and read performance is about 3200 Mbyte/sec. Performance by using PCIe Gen2 on ArriaV GX board is slower than Gen3. Write and read performance on Gen2 are about 1600 Mbyte/sec.

5 Revision History

Revision	Date	Description
1.0	5-Aug-16	Initial Release
1.1	16-Dec-16	Change buffer from DDR to Block Memory
1.2	9-May-17	- Use Avalon-ST PCIe Hard IP instead of Avalon-MM - Add Example test result
2.0	8-Jun-17	Support only 256 Kbyte buffer
2.1	31-Jul-17	Add LFSR pattern
3.0	24-Jul-18	Support Shutdown, SMART and Flush command
3.1	23-Nov-2018	Add dword enable for RAM interface
3.2	11-Feb-2020	Update description
3.3	25-Mar-2020	Add Function list in Test firmware