NVMe-IP for PLDA PCIe reference design manual

Rev1.1 9-Oct-18

1 NVMe

NVM Express (NVMe) defines interface for the host controller to access solid state drives (SSD) through PCI Express. NVM Express optimizes the process to issue command and completion by using only 2 register writes (one for command and another for completion). Also, NVMe supports parallel operation by supporting up to 64K commands within single queue. So, performance for sequential and random access is improved.

In PCIe SSD market, two standards are found, i.e. AHCI and NVMe. AHCI is the older standard to interface with SATA hard disk drives while NVMe is designed for non volatile memory like SSD. The comparison between AHCI and NVMe protocol in more details can be found from "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 devices is shown in http://www.nvmexpress.org/products/.

Generally, user needs to install NVMe driver to access NVMe SSD as shown in Figure 1-1. Physical connector of NVMe SSD is PCIe type such as M.2 connector. NVMe-IP implements NVMe driver and the task running on CPU by pure-hardware logic. So, CPU is not required to access NVMe SSD when using NVMe-IP in FPGA board.

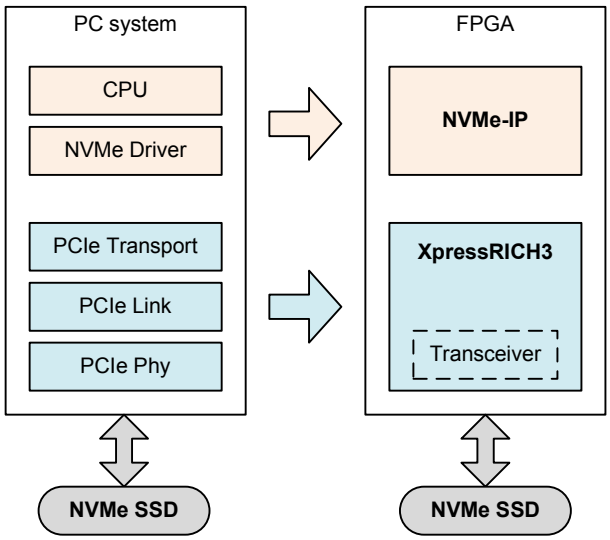

Figure 1-1 NVMe protocol layer

2 Hardware overview

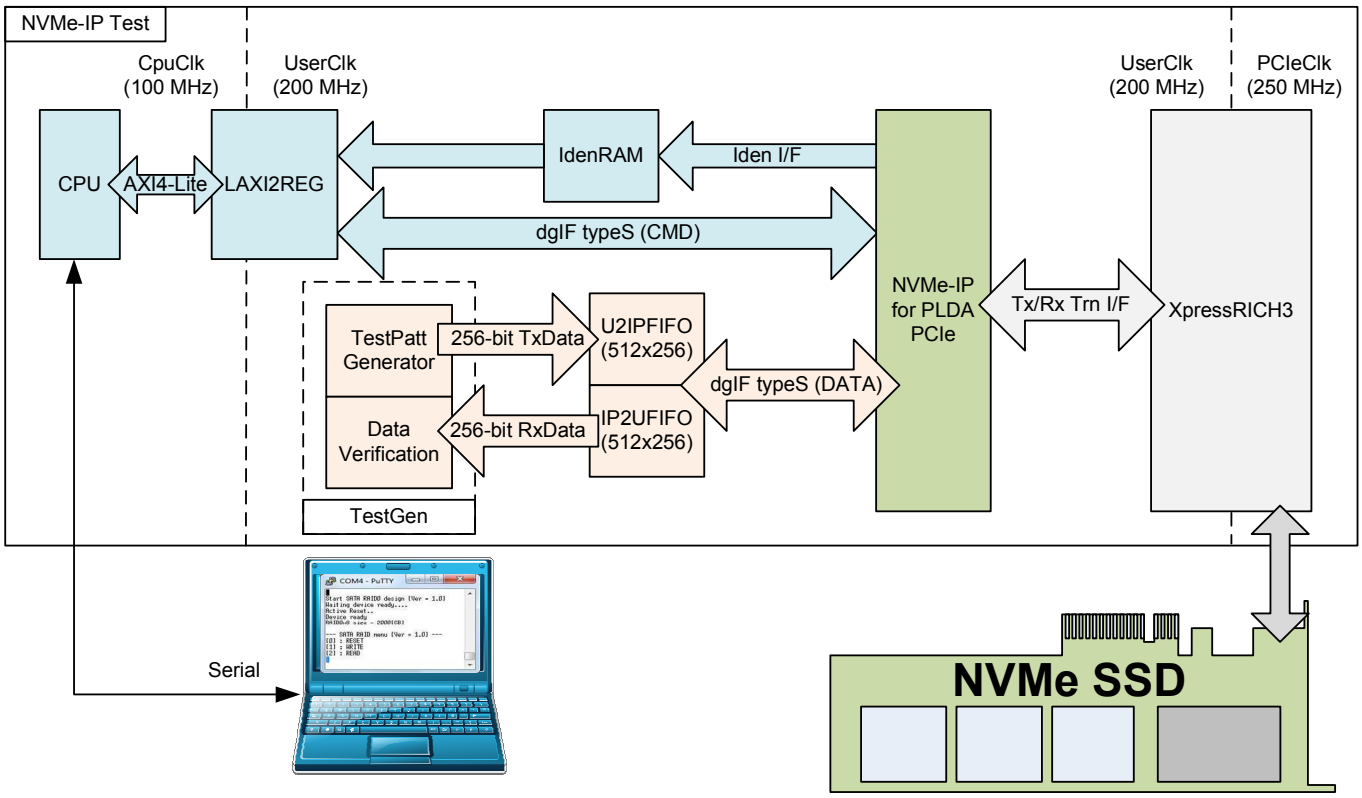

Figure 2-1 NVMe-IP demo hardware

The hardware system can be split into three groups following the interface.

1) TestGen: The example of user logic to write and read data in this reference design is TestGen module. TestGen module generates test data to U2IPFIFO at the highest speed with flow control in Write command. For Read command, TestGen reads and verifies test data from IP2UFIFO at the highest speed with flow control. TestGen uses 256-bit data bus and runs in UserClk domain which is equal to 200 MHz. Maximum bandwidth of TestGen is more than maximum performance of Gen3 SSD.
2) NVMe: NVMe-IP connecting with XpressRICH3 is used to interface with NVMe SSD. Command and data interface of NVMe-IP is dgIF typeS format. Command interface is controlled by CPU while data interface is connected to FIFO. IdenRAM (implemented by simple dual port RAM) is used to connect with Identify interface of NVMe-IP and CPU.
3) CPU: Test operation in the demo is controlled by user through Serial console. CPU firmware is designed to receive command and command parameters from user. Then, parameters are set to the hardware through AXI4-Lite bus. LAxi2Reg has the register sets of test parameters which are mapped to different address of CPU. LAxi2Reg decodes the address of AXI4-Lite bus to select the active parameter. For write access, Write data from AXI4-Lite bus is set to the selected parameter following the address. For read access, Read data from selected parameter is returned to AXI4-Lite bus. Read access is applied for CPU monitoring and displaying the hardware status to the user through Serial console.

   More details of the hardware are described as follows.

2.1 TestGen

This module is designed to generate Test pattern to WrFf in Write command or reads data from RdFf to verify in Read command at the fastest speed to check system performance. The details of hardware inside TestGen are shown in Figure 2-2.

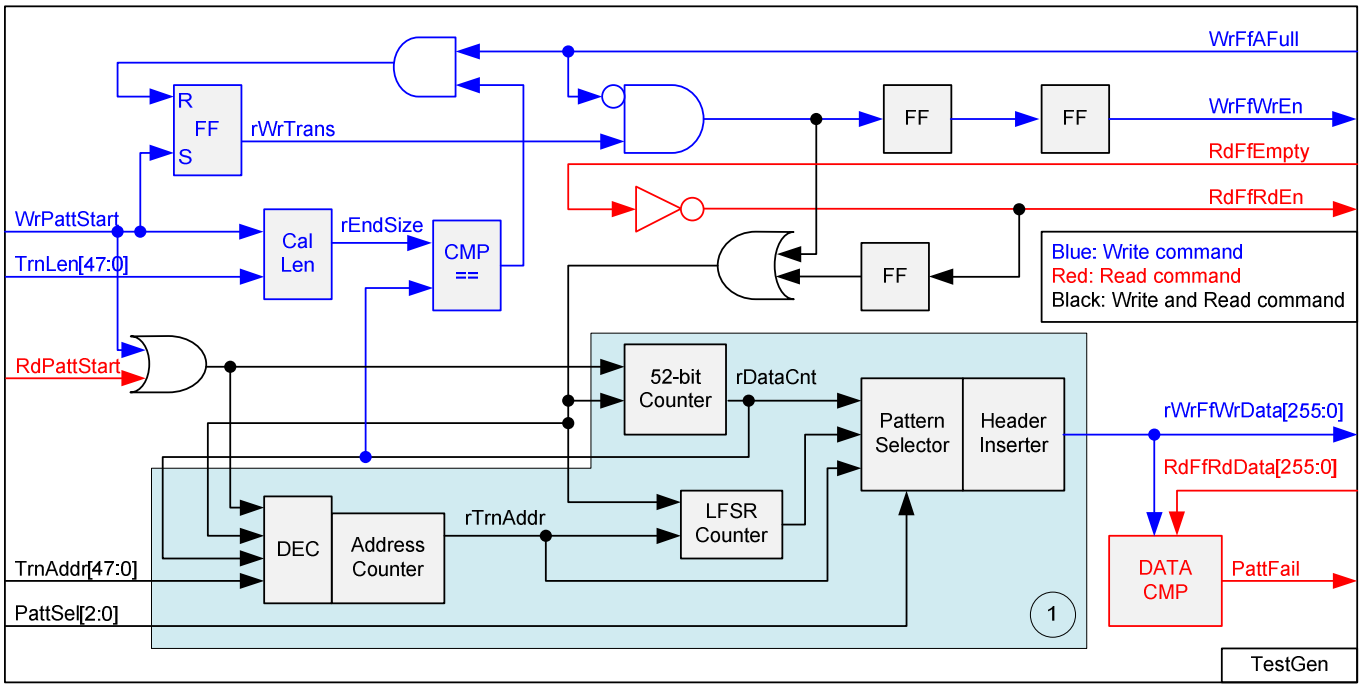

Figure 2-2 TestGen hardware

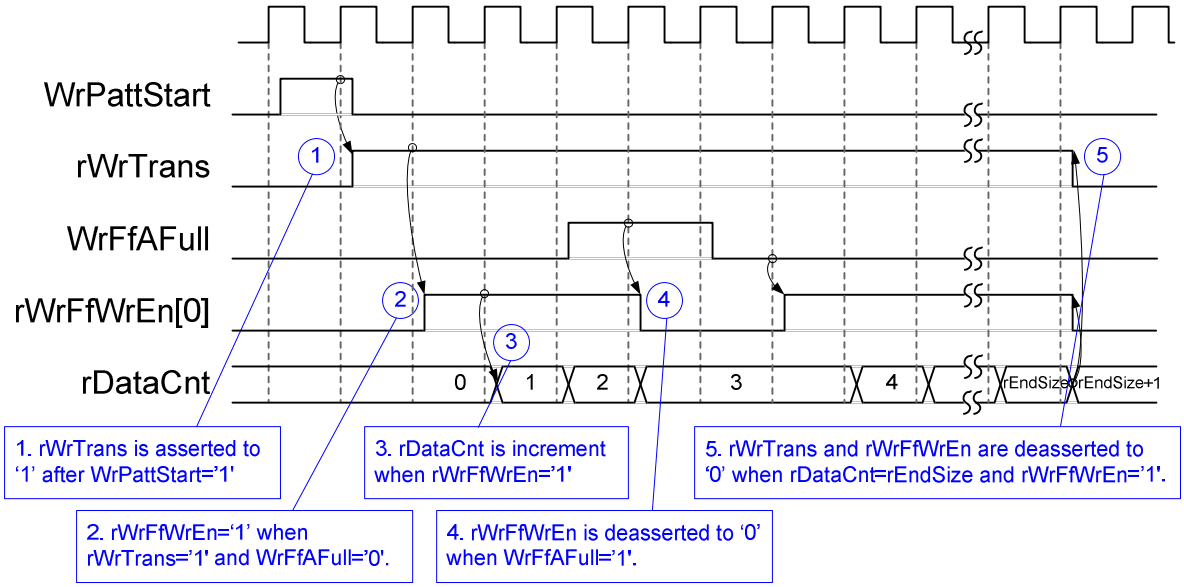

Figure 2-3 Timing diagram of Write operation in TestGen

To start Write operation, rWrTrans is asserted to '1' when WrPattStart from LAxi2Reg is asserted to '1'. If rWrTrans='1' (Write command is operating) and WrFfAFull='0' (WrFf is ready to receive new data), rWrFfWrEn[0] will be asserted to '1' to send test data to WrFf. If WrFfAFull='1', rWrFfWrEn[0] will be de-asserted to '0' to pause data transferring. rDataCnt is data counter to check total transfer size, increased by rWrFfWrEn[0]. When total data are transferred complete (rDataCnt=EndSize), rWrTrans and rWrFfWrEn[0] are de-asserted to '0' to stop data transferring.

For Read operation, RdFfRdEn signal is designed by using NOT logic to RdFfEmpty. rDataCnt is increased when RdFfRdEn is asserted to '1'.

Block no.1 in lower side of Figure 2-2 shows the logic for generating test pattern in TestGen module. To create unique test data for each 512-byte data, test pattern is designed as shown in Figure 2-4.

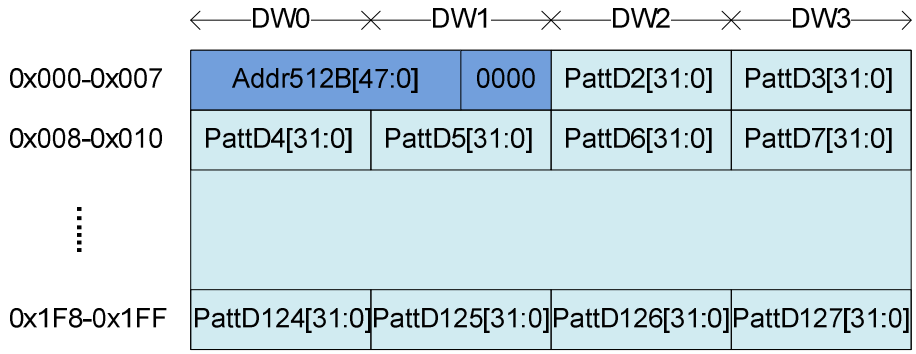

Figure 2-4 Test pattern format in each 512-byte unit

Test pattern consists of two parts, i.e. 64-bit header in Dword#0 and Dword#1 of each 512-byte and test data in Dword#2 – Dword#127. 64-bit header is created by using address value in 512-byte unit. As shown in Figure 2-2, TrnAddr is loaded to be initial value of rTrnAddr. rTrnAddr is applied to be 64-bit header of each 512-byte data and increased every 512-byte transferring. rDataCnt and write/read enable signal are monitored to check end of 512-byte transferring.

TestGen supports to generate five patterns, i.e. 32-bit increment, 32-bit decrement, all 0, all 1, and 32-bit LFSR. 32-bit increment is generated by using lower-bit of rTrnAddr and rDataCnt. Decrement pattern is designed by using NOT logic of increment data. The equation of 32-bit LFSR is x^31 + x^21 + x + 1. To create 256-bit LFSR pattern, two sets of 32-bit LFSR are designed as shown in Figure 2-5.

The 1st DW data of set#1 uses 16 lower bit of Addr512B (address in 512-byte unit) and 16 higher bit of NAddr512B (not logic of Addr512B) to be initial value for generating test pattern. Otherwise, the 1st DW data of set#2 uses 16 lower bit of NAddr512B and 16 higher bit of Addr512B.

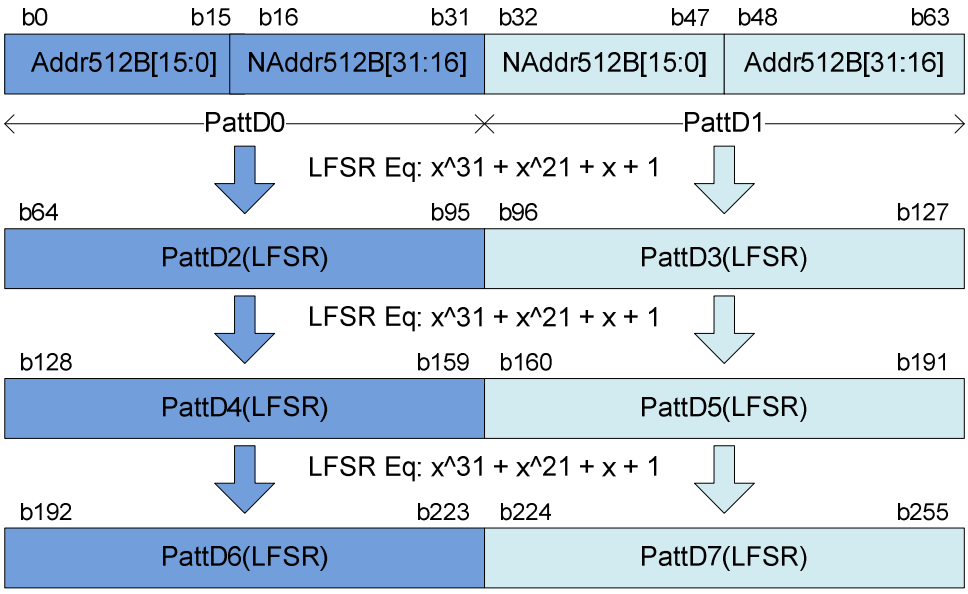

Figure 2-5 LFSR pattern in TestGen

Each LFSR logic set is designed to generate 128-bit LFSR data, so four 32-bit LFSR data must be generated within one clock. The logic to design LFSR must use look-ahead style to generate PattD0/D2/D4/D6 or PattD1/D3/D5/D7 in the same clock.

3-bit PattSel signal is used to select one of five test patterns. Header Inserter logic inserts 64-bit header to be the 1st and 2nd data of each 512 byte. After that, test data from pattern counter is used to be rWrFfWrData. In Read command, rWrFfWrData is used to be expected value to compare with read data from FIFO (RdFfRdData). PattFail is asserted to '1' when data verification is failed.

2.2 NVMe

User interface of NVMe-IP is designed by using dgIF typeS format. CMD interface is connected to LAxi2Reg to receive the parameter from user through Serial console. 256-bit data bus is connected with U2IPFIFO and IP2UFIFO. NVMe-IP connects to XpressRICH3 for creating PCIe packet and converting to PCIe signals. SSD is directly connected to XpressRICH3.

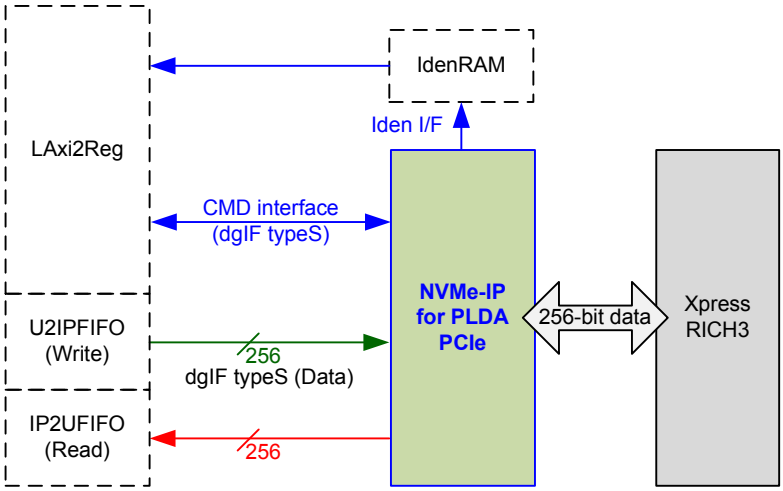

Figure 2-6 NVMe hardware

To support Identify command, one additional RAM is connected in the system. IdenRAM is simple dual-port RAM which is used to store 8K byte data output from Identify command. IdenRAM is read by CPU through LAxi2Reg.

2.2.1 NVMe-IP for PLDA PCIe

NVMe-IP for PLDA PCIe implements NVMe protocol of Host side to access NVMe SSD. User interface is simple designed by using dgIF typeS format. NVMe-IP is designed to connect with XpressRICH3. More details of NVMe-IP are described in datasheet.
https://dgway.com/products/IP/NVMe-IP/dg_nvmeip_pldapcie_data_sheet_en.pdf

2.2.2 XpressRICH3 for Xilinx

This block is PCIe soft IP core from PLDA. More details are described in following website.
https://www.plda.com/node/403

2.3 CPU and Peripherals

The hardware is connected to CPU through AXI4-Lite bus, similar to other CPU peripherals. The hardware registers are mapped to CPU memory address, as shown in Table 2-1. LAxi2Reg is the module to interface with CPU following memory map.

LAxi2Reg connects to many hardwares in the system such as TestGen, NVMe-IP, and IdenRAM to interface control and status signals of each module. As shown in Figure 2-7, there are two clock domains applied in this block, i.e. CpuClk (CPU Clock and AXI4-Lite bus) and UserClk (User clock domain for TestGen and NVMe block).

AsyncAxiReg includes asynchronous circuit between CpuClk and UserClk. More details of each hardware are described as follows.

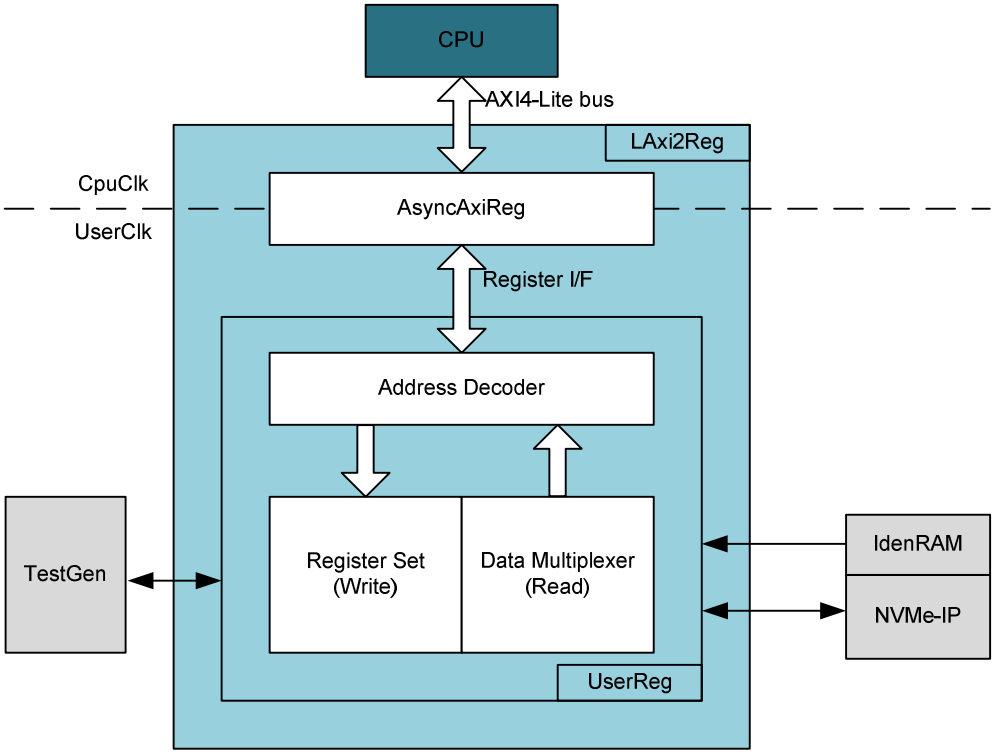

Figure 2-7 CPU and peripherals hardware

2.3.1 AsyncAxiReg

This module is designed to convert the signal interface of AXI4-Lite to be register interface. Also, it transfers signals in CpuClk domain to be UserClk domain. Timing diagram of register interface is shown in Figure 2-8.

To write register, timing diagram is same as 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 byte enable of this access: bit[0] is write enable for RegWrData[7:0], bit[1] is used for RegWrData[15:8], …, and bit[3] is used for RegWrData[31:24]).

To read register, AsyncAxiReg asserts RegRdReq to '1' with the valid value of RegAddr (the register address is used for 32-bit data). After that, the read data is valid on RegRdData bus with asserting RegRdValid to '1'.

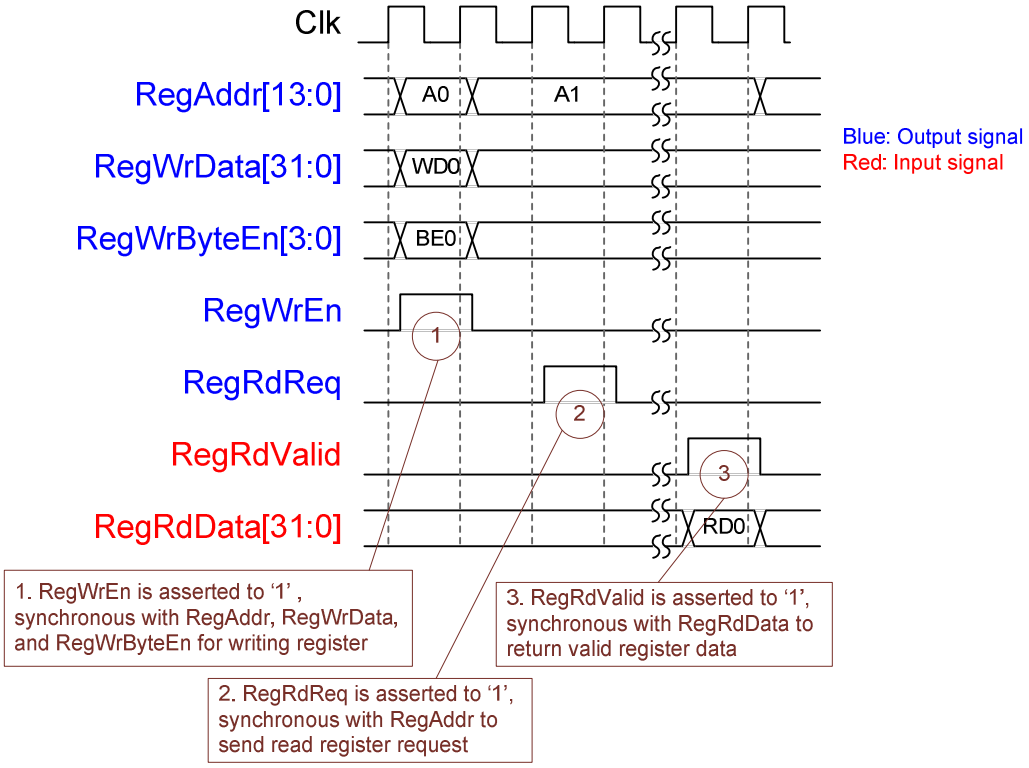

Figure 2-8 Register interface timing diagram

2.3.2 UserReg

As shown in Figure 2-7, after RegWrEn or RegRdReq is asserted to '1' to request write or read register, RegAddr is loaded to Address decoder to select the active register. For write register, RegWrData signal is loaded to be the new value of active register. In this module, RegWrByteEn is not used, so CPU firmware needs to access the hardware register by using 32-bit pointer only.

For read request, CPU monitors status signals of many modules such as TestGen, NVMe-IP, and IdenRAM. To avoid timing constraint problem, many status signals are selected by using multiplexer with two-stage pipeline registers. So, RegRdValid is asserted to '1' after RegRdReq is asserted for two clock cycles. Two latency clock cycles is designed by adding two D Flip-flops to generate RegRdValid from RegRdReq.

Memory map of control and status signals inside UserReg module is shown in Table 2-1.

Table 2-1 Register Map

Address Rd/Wr	Register Name (Label in the “nvmexr1iptest.c”)	Description
BA+0x0000 Wr	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 Wr	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 Wr	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 Wr	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 Wr	User Command Reg (USRCMD_REG)	[1:0]: Input to be user command (UserCmd of dgIF typeS for NVMe-IP) “00”: Identify, “10”: Write SSD, “11”: Read SSD When this register is written, the design generates command request to NVMe-IP to start new command operation.
BA+0x0014 Wr	Test Pattern Reg (PATTSEL_REG)	[2:0]: Test pattern select “000”-Increment, “001”-Decrement, “010”-All 0, “011”-All 1, “100”-LFSR
BA+0x0100 Rd	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) [4:3]: PCIe speed from IP (“00”: No linkup, “01”: PCIe Gen1, “10”: PCIe Gen2, “11”: PCIe Gen3)
BA+0x0104 Rd	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 Rd	Total disk size (High) Reg (LBASIZEH_REG)	[15:0]: Total capacity of SSD in 512-byte unit (LBASize[47:32] of dgIF typeS)
BA+0x010C Rd	User Error Type Reg (USRERRTYPE_REG)	[31:0]: User error status (UserErrorType[31:0] of dgIF typeS)
BA+0x0114 Rd	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 Rd	NVMe CAP Reg (NVMCAP_REG)	[31:0]: NVMeCAPReg[31:0] output from NVMe-IP
BA+0x011C Rd	NVMe IP Test pin Reg (NVMTESTPIN_REG)	[31:0]: TestPin[31:0] output from NVMe-IP
BA+0x0120 Rd	Data Failure Address (Low) Reg (RDFAILNOL_REG)	[31:0]: Latch value of failure address[31:0] in byte unit from read command
BA+0x0124 Rd	Data Failure Address (High) Reg (RDFAILNOH_REG)	[24:0]: Latch value of failure address [56:32] in byte unit from read command

Address Rd/Wr	Register Name (Label in the “nvmeiptest.c”)	Description
BA+0x0140 Rd	Expected value Word0 Reg (EXPPATW0_REG)	[31:0]: Latch value of expected data [31:0] from read command
BA+0x0144 Rd	Expected value Word1 Reg (EXPPATW1_REG)	[31:0]: Latch value of expected data [63:32] from read command
BA+0x0148 Rd	Expected value Word2 Reg (EXPPATW2_REG)	[31:0]: Latch value of expected data [95:64] from read command
BA+0x014C Rd	Expected value Word3 Reg (EXPPATW3_REG)	[31:0]: Latch value of expected data [127:96] from read command
BA+0x0150 Rd	Expected value Word4 Reg (EXPPATW4_REG)	[31:0]: Latch value of expected data [159:128] from read command
BA+0x0154 Rd	Expected value Word5 Reg (EXPPATW5_REG)	[31:0]: Latch value of expected data [191:160] from read command
BA+0x0158 Rd	Expected value Word6 Reg (EXPPATW6_REG)	[31:0]: Latch value of expected data [223:192] from read command
BA+0x015C Rd	Expected value Word7 Reg (EXPPATW7_REG)	[31:0]: Latch value of expected data [255:224] from read command
BA+0x0180 Rd	Read value Word0 Reg (RDPATW0_REG)	[31:0]: Latch value of read data [31:0] from read command
BA+0x0184 Rd	Read value Word1 Reg (RDPATW1_REG)	[31:0]: Latch value of read data [63:32] from read command
BA+0x0188 Rd	Read value Word2 Reg (RDPATW2_REG)	[31:0]: Latch value of read data [95:64] from read command
BA+0x018C Rd	Read value Word3 Reg (RDPATW3_REG)	[31:0]: Latch value of read data [127:96] from read command
BA+0x0190 Rd	Read value Word4 Reg (RDPATW4_REG)	[31:0]: Latch value of read data [159:128] from read command
BA+0x0194 Rd	Read value Word5 Reg (RDPATW5_REG)	[31:0]: Latch value of read data [191:160] from read command
BA+0x0198 Rd	Read value Word6 Reg (RDPATW6_REG)	[31:0]: Latch value of read data [223:192] from read command
BA+0x019C Rd	Read value Word7 Reg (RDPATW7_REG)	[31:0]: Latch value of read data [255:224] from read command
BA+0x01C0 Rd	Current test byte (Low) Reg (CURTESTSIZEL_REG)	[31:0]: Current test data size of TestGen module in byte unit (bit[31:0])
BA+0x01C4 Rd	Current test byte (High) Reg (CURTESTSIZEH_REG)	[24:0]: Current test data size of TestGen module in byte unit (bit[56:32])
BA+0x2000 – 0x2FFF	Identify Controller Data (IDENCTRL_REG)	4Kbyte Identify Controller Data Structure
BA+0x3000 – 0x3FFF	Identify Namespace Data (IDENNAME_REG)	4Kbyte Identify Namespace Data Structure

3 CPU Firmware

After system boot-up, CPU initializes its peripherals such as UART and Timer. Next, CPU waits until PCIe connection links up (PCISTS_REG[0]='1'). Finally, CPU waits until NVMe-IP completes initialization process (USRSTS_REG[0]='0').

To receive command from user, Main menu is displayed on the console for user selecting one of six commands (Identify, Write, or Read). More details of the sequence in each command are described as follows.

3.1 Identify Command

The sequence of the firmware when user selects Identify command is below.

1) Set USRCMD_REG="00". 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. Bit[1] is asserted to '1' when some errors are detected. In case of error condition, there is error message displayed on the console. If the command is completed, the data from Identify command of NVMe-IP will be stored in IdenRAM.
3) CPU reads Identify data from IdenRAM (IDENCTRL_REG) and displays SSD model name. Otherwise, SSD capacity and LBA unit size are also displayed by reading from NVMe-IP output (LBASIZEL_REG and LBASIZEH_REG).

3.2 Write/Read Command

The sequence of the firmware when user selects Write/Read command is below.

1) Receive start address, transfer length, and test pattern through Serial console. If some inputs are invalid, the operation will be cancelled.
2) Get all inputs and set the value to USRADRL/H_REG, USRLENL/H_REG, PATTSEL_REG, and USRCMD_REG (USRCMD_REG="10" for Write command, and "11" for Read command).
3) CPU waits until the operation is completed or some errors (except verification error) are found by monitoring USRSTS_REG[2:0]. If USRSTS_REG[2] (verification error) is asserted to '1', verification error message will be displayed. After that, CPU still runs until end of operation or user inputs any key to cancel operation.
4) During running command, current transfer size reading from CURTESTSIZE_REG is displayed every second. Finally, test performance is displayed on Serial console when command is completed.

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.

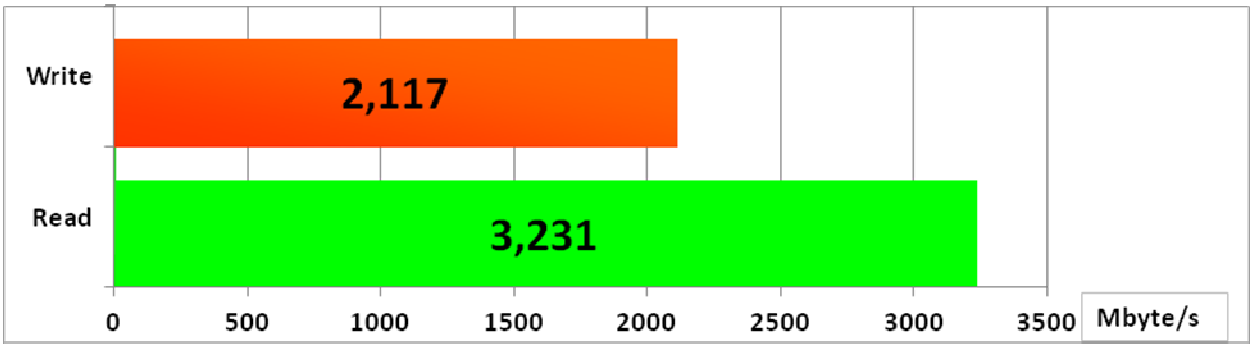

Figure 4-1 Test Performance of NVMe IP demo by using Samsung 960 Pro SSD

By using PCIe Gen3 on ZCU102 board, write performance is about 2100 Mbyte/sec and read performance is about 3200 Mbyte/sec.

5 Revision History

Revision	Date	Description
1.0	5-Feb-18	Initial Release
1.1	9-Oct-18	Add more information