NVMe-IP DDR reference design manual

Rev1.0 17-Apr-18

1 Introduction

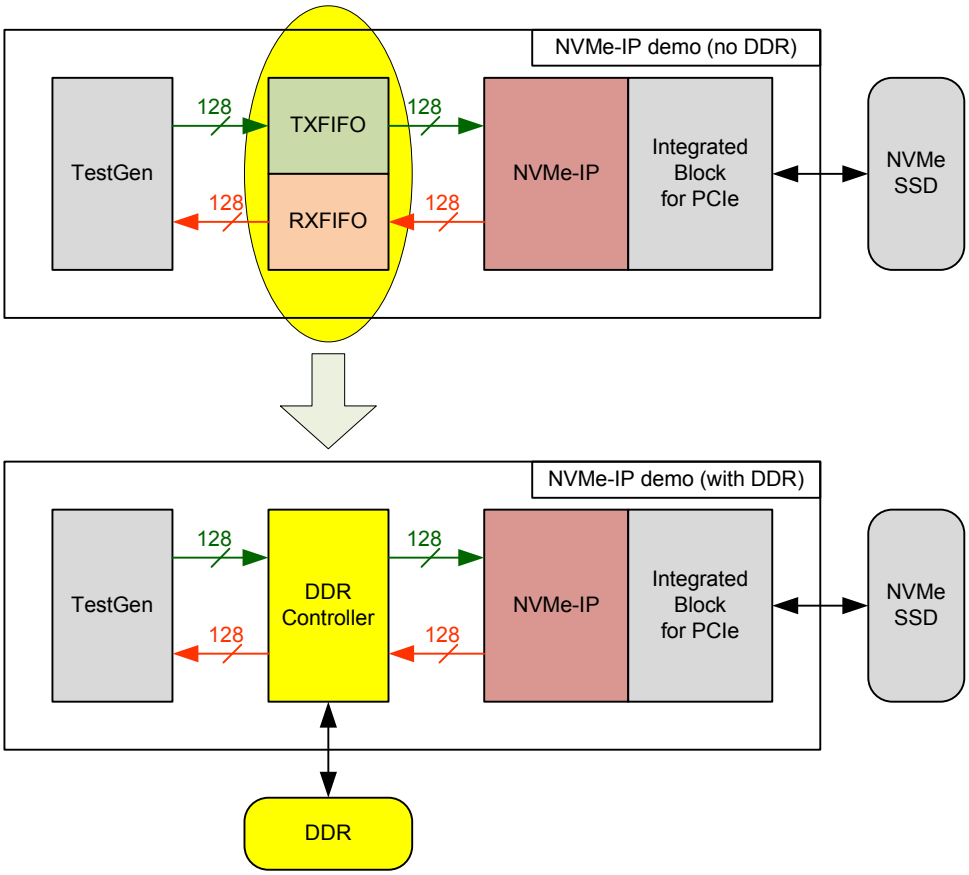

Figure 1-1 NVMe-IP with and without DDR demo comparison

This reference design is modified from standard NVMe-IP demo by using DDR to be data buffer instead of FIFO, as shown in yellow block of Figure 1-1 . DDR must be used for the system which requires big data buffer such as data logger from high-speed A/D which receives data at constant rate.

In Write command, when SSD is not ready, data from TestGen is stored to DDR. The data from DDR is flushed to SSD when SSD changes to ready status. So, minimum DDR buffer size in the system is equal to the maximum busy time of SSD x data input rate. This feature is found in data logger system.

In Read command, data from SSD is stored in DDR before forwarding to TestGen. Data in DDR must be much enough for user reading in constant rate when SSD is not ready to return data. Similar to Write command, minimum DDR buffer size is equal to the maximum busy time of SSD x data output rate. This feature is found for high resolution display system.

Before running the reference design, it is recommended to study NVMe-IP datasheet and standard demo firstly from following link.
http://www.dgway.com/products/IP/NVMe-IP/dg_nvme_ip_data_sheet_en.pdf
http://www.dgway.com/products/IP/NVMe-IP/dg_nvmeip_refdesign_en.pdf
http://www.dgway.com/products/IP/NVMe-IP/dg_nvmeip_instruction_en.pdf

2 Hardware overview

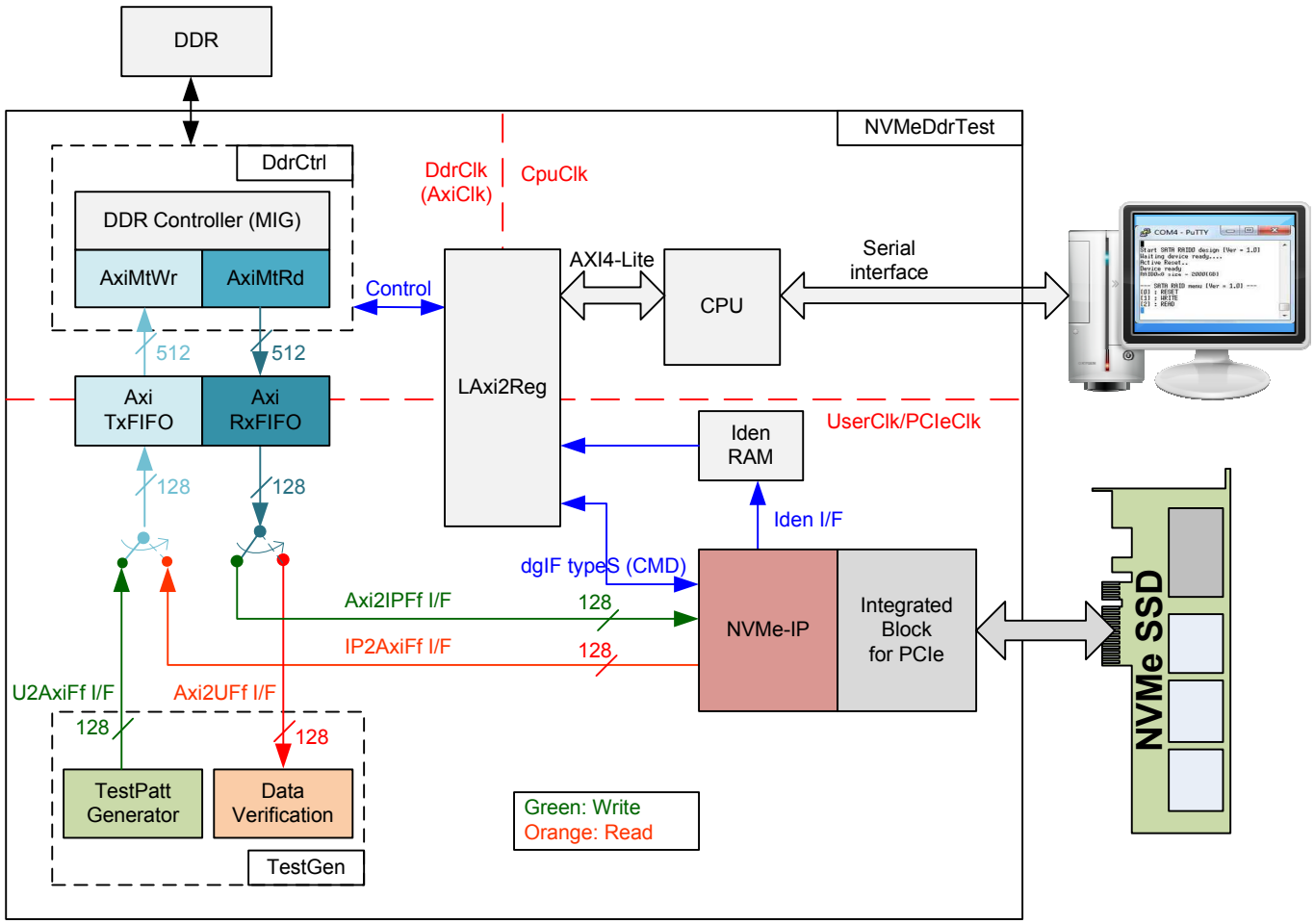

Figure 2-1 NVMe-IP with DDR demo hardware

The hardware system can be split into four groups following the interface i.e.

1) TestGen: TestGen is designed to be the example of user logic to write and read data with NVMe-IP. TestGen generates test data at constant rate to DDR in Write command. For Read command, TestGen also reads and verifies test data from DDR at constant rate. TestGen uses 128-bit data bus and runs in UserClk domain which is equal to 250 MHz.

2) NVMe: NVMe-IP and Integrated Block for PCIe implement NVMe protocol to transfer data with NVMe SSD at ultra high speed rate. Command interface of NVMe-IP is controlled by CPU through LAxi2Reg module. Data interface of NVMe-IP is 128-bit and connects to AxiTxFIFO and AxiRxFIFO. NVMe-IP runs in UserClk domain like TestGen module.

3) DdrCtrl: DDR controller in reference design uses 512-bit AXI4 bus to be user interface. AxiMtWr is designed to transfer data from AxiTxFIFO to DDR while AxiMtRd is designed to transfer data from DDR to AxiRxFIFO. The source of AxiTxFIFO and the destination of AxiRxFIFO are switched between TestGen and NVMe-IP following the command. For Write command, Data from TestGen is stored to DDR through AxiTxFIFO and Read data from DDR is forwarded to NVMe-IP through AxiRxFIFO. For Read command, DDR is written by NVMe-IP and read by TestGen.

4) CPU: Test operation in the demo is controlled by user through Serial console. CPU firmware is designed to receive test parameters and the command from user. CPU sets parameters to the hardware through AXI4-Lite bus. LAxi2Reg has the register sets of test parameters which are mapped as CPU memory. Each test parameter is defined in different address. LAxi2Reg decodes the address of AXI4-Lite bus to select the active parameter. For write access, write data from AXI4-Lite bus is loaded to set the parameter, selected by the write address. For read access, read data from all parameters are fed to data multiplexer to select the returned data following the read address. The parameter and system status are monitored by CPU through AXI4-Lite bus and displayed to 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 as constant rate in Write command. For Read command, this module reads data from RdFf as constant rate and verifies the data with Test pattern. The details of hardware inside TestGen are shown in Figure 2-2.

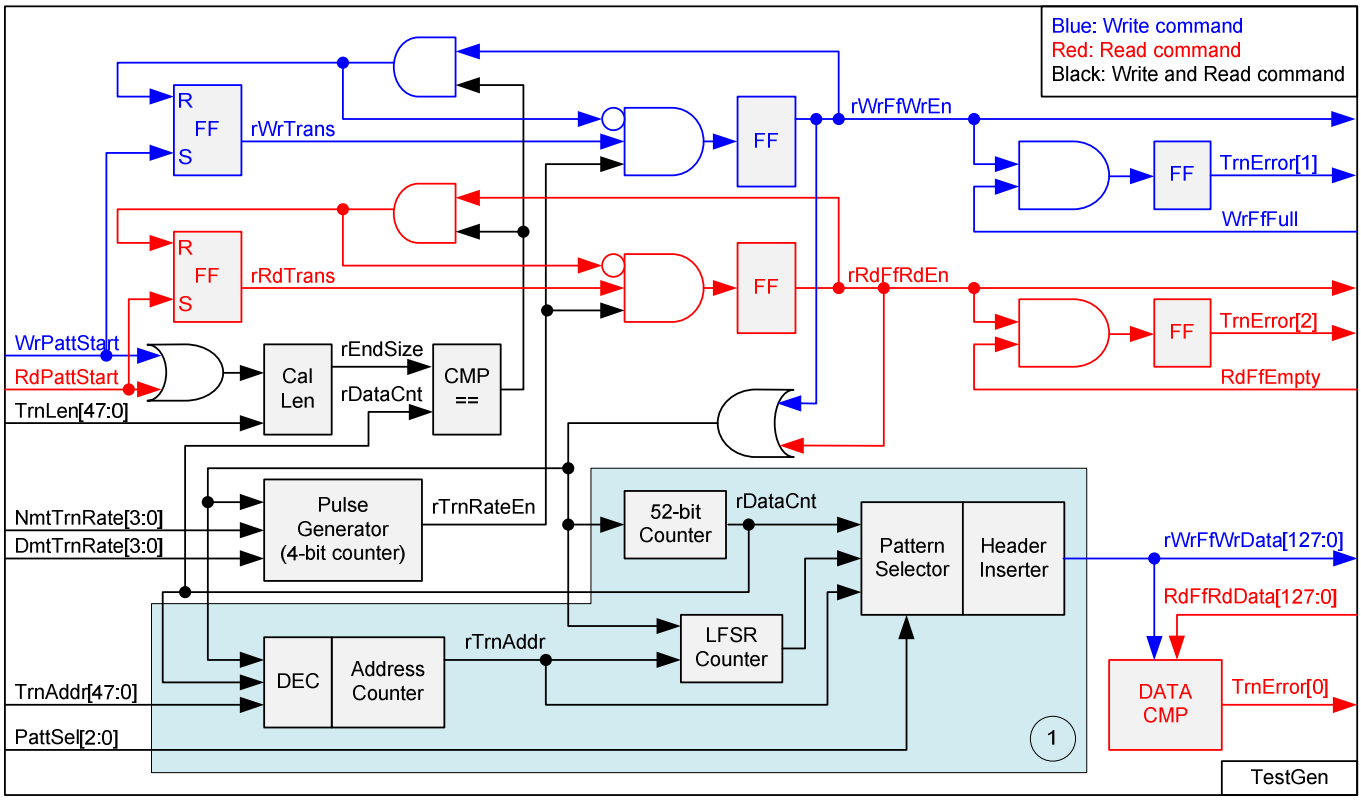

Figure 2-2 TestGen hardware

Pulse Generator (bottom left block) is designed to generate enable pulse (rTrnRateEn) to control data rate for writing and reading FIFO. As shown in Figure 2-3, duty cycle of rTrnRateEn is controlled by NmtTrnRate and DmtTrnRate value, set by user. rTrnRateEn is used to be enable signal to assert rWrFfWrEn and rRdFfRdEn to '1'. Maximum data rate of rWrFfWrEn and rRdFfRdEn is equal to (NmtTrnRate/DmtTrnRate) x Clock Frequency (250 MHz) x 128-bit.

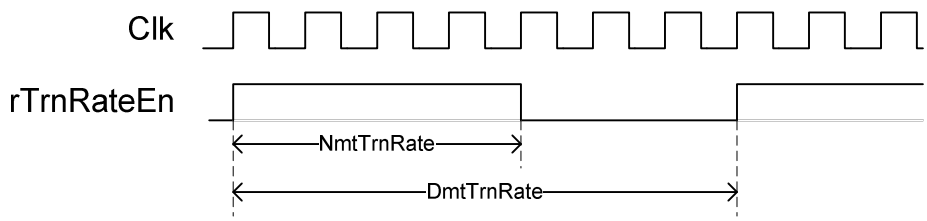

Figure 2-3 rTrnRateEn Timing diagram

The logic to generate rWrFfWrEn and rRdFfRdEn is shown in upper side as blue and red color sequentially. To start Write operation, WrPattStart is asserted to '1' as a pulse. rWrTrans is asserted to '1' by WrPattStart to start writing FIFO signal. rWrTrans is de-asserted to '0' when last data is written to FIFO, monitored by comparing rDataCnt (counter to check current data size) with rEndSize (total data size setting from user). Timing diagram of rWrTrans is shown in Figure 2-4.

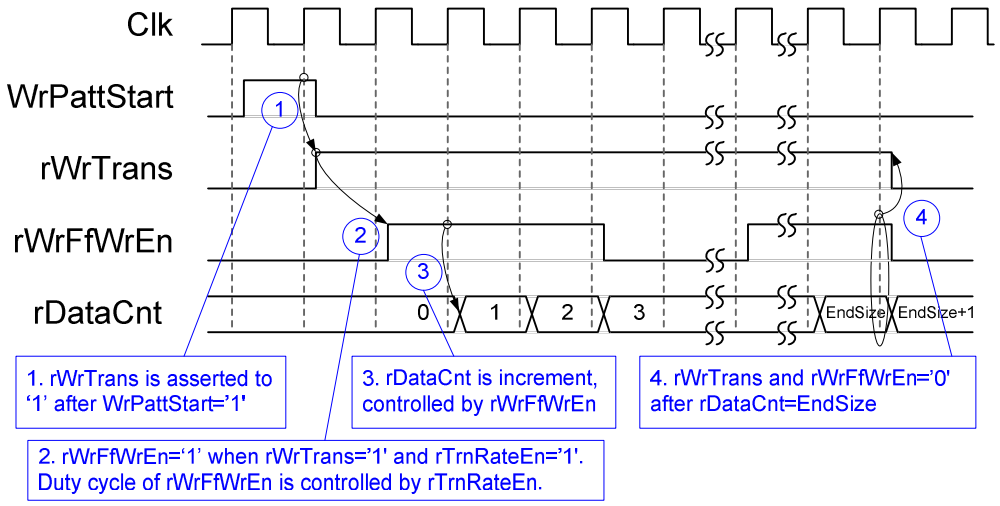

Figure 2-4 rWrTrans Timing diagram

rWrFfWrEn is controlled by rWrTrans and rTrnRateEn. When rWrTrans='1', rWrFfWrEn timing is same as rTrnRateEn to write data to FIFO as constant rate. rDataCnt is up-counter, controlled by rWrFfWrEn during Write command. After last data is transferred, rWrFfWrEn and rWrTrans are de-asserted to '0' to finish write operation.

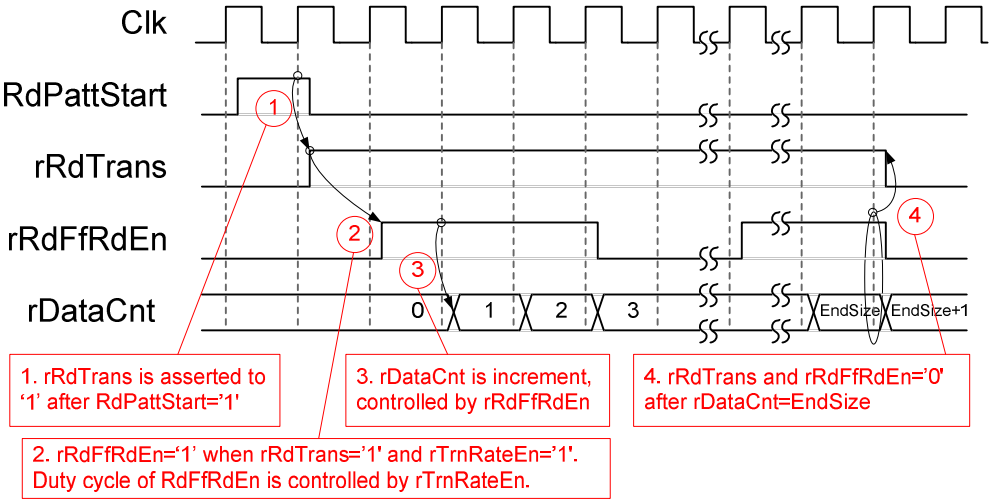

Figure 2-5 rRdTrans Timing diagram

Similar to rWrTrans, rRdTrans timing diagram is shown in Figure 2-5. rRdTrans is asserted to '1' for enabling the logic to read data from FIFO as constant rate. When rRdTrans='1', rRdFfRdEn timing is same as rTrnRateEn to read data from FIFO as constant rate. rDataCnt is the counter to check total data, controlled by rRdFfRdEn during Read command. rRdFfRdEn and rRdTrans are de-asserted to '0' after last data is received.

As shown in upper right side of Figure 2-2, TrnError[1] is asserted to '1' when WrFfFull is equal to '1' during write operation. So, TrnError[1] is used to detect that system cannot operate Write command in this rate. TrnError[2] is asserted to '1' when RdFfEmpty is equal to '1' during reading operation. So, TrnError[2] is used to detect that system cannot operate Read command in this rate.

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 sector, test pattern in this demo is designed as shown in Figure 2-6.

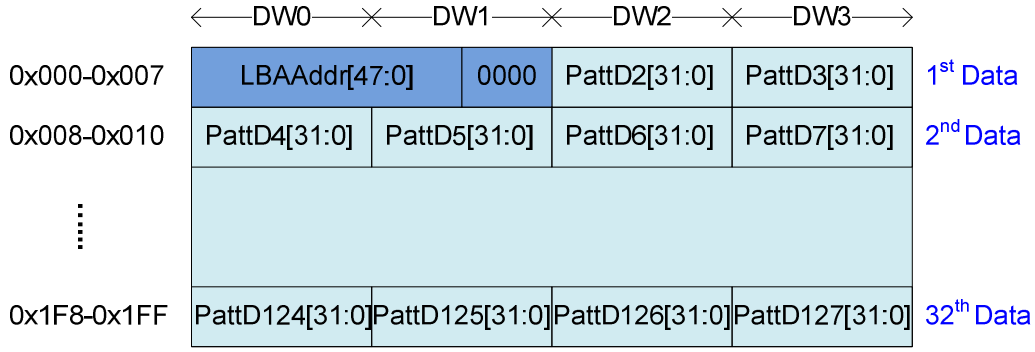

Figure 2-6 Test pattern format in each sector

Test pattern consists of two parts, i.e. 64-bit header in DW#0 - DW#1 of each sector and test data in DW#2 – DW#127. 64-bit header is created by using LBA address value of the data (LBA address is the address in sector unit). As shown in Figure 2-2, TrnAddr is loaded to be start value of rTrnAddr to create 64-bit header value. rTrnAddr is increased after completing to transfer one sector data. rDataCnt and write/read enable signal are monitored to check end of one sector 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 with increment data. To create 32-bit LFSR counter, 128-bit data is calculated by using look-ahead logic to complete four LFSR data in one clock cycle as shown in Figure 2-7.

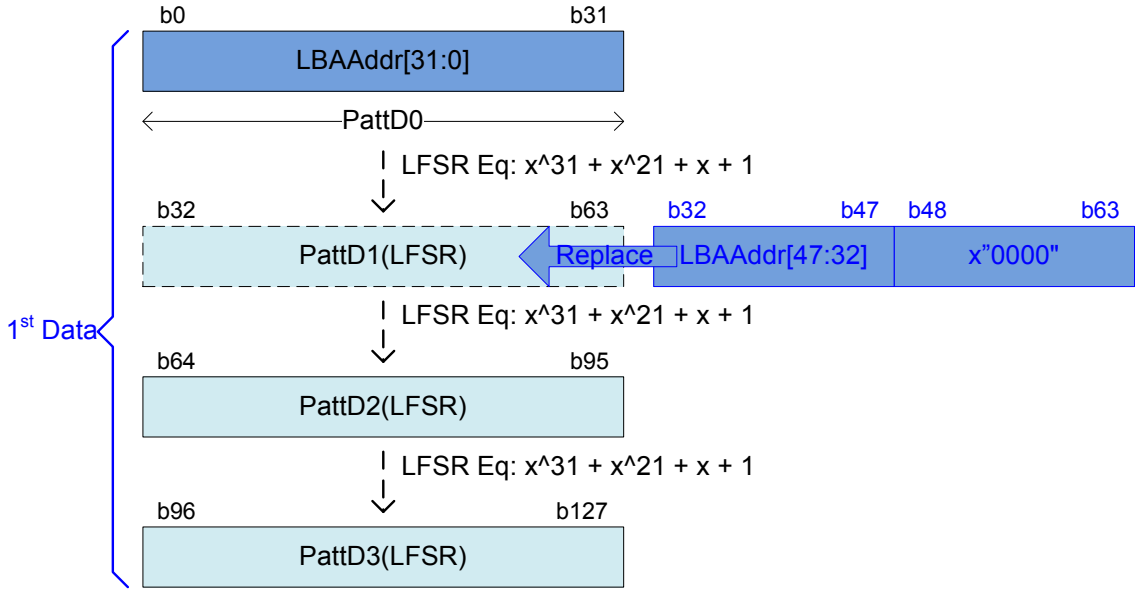

Figure 2-7 LFSR pattern in TestGen

Start value of LFSR is designed by using 32 lower bit of LBA Address. PattD1 is used to calculate the next LFSR data only, but finally it is replaced by 16 upper bit of LBA Address to complete 64-bit header in each sector.

3-bit PattSel signal are used to select one of five test patterns. Header Inserter logic inserts 64-bit header to be the 1st data of each sector. After that, test data from pattern counter is transferred to be rWrFfWrData. In Read command, rWrFfWrData is used to be expected value to compare with read data from FIFO (RdFfRdData). TrnError[0] is asserted to '1' when data verification is failed.

2.2 NVMe

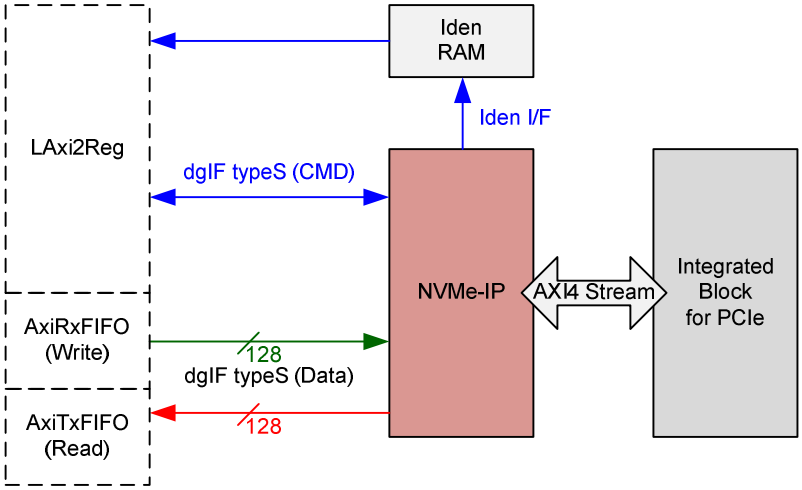

Figure 2-8 RAID hardware

As shown in Figure 2-8, 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. Data bus size of NVMe-IP is 128-bit and connects with AxiRxFIFO and AxiTxFIFO. Identify data of NVMe-IP is stored in IdenRAM. Another side of IdenRAM is connected to LAxi2Reg to allow user reading Identify data of NVMe SSD.

2.2.1 NVMe-IP

NVMe-IP 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 Integrated Block for PCIe which is Hard IP in Xilinx device. More details of NVMe-IP are described in datasheet.
http://www.dgway.com/products/IP/NVMe-IP/dg_nvme_ip_data_sheet_en.pdf

2.2.2 Integrated Block for PCIe

This is Hard IP in Xilinx device which implements the lower layer of PCIe protocol. The interface of the IP is different for each FPGA model. Please see more details from Xilinx website.
https://www.xilinx.com/products/technology/pci-express.html

2.3 DdrCtrl

This module is designed to interface with DDR memory which is the data buffer to support transferring data as sustain rate in this system. MIG is Xilinx IPcore to control external memory such as DDR4. The IP includes clock generator to generate many clocks for interfacing with DDR. This reference design also uses clock generator inside MIG to generate CpuClk (100 MHz) for running CPU and its peripherals. AxiClk is clock output from MIG to be user interface of DDR controller. AxiClk frequency is 300 MHz for running DDR4 at 1200 MHz.

AxiMtWr and AxiMtRd are designed to decode the command request from LAxi2Reg. Data bus of AxiMtWr and AxiMtRd for transferring data with MIG is AXI4 interface. AxiMtWr is connected to Write port of AXI4 bus for transferring data from AxiTxFIFO to MIG. AxiMtRd is connected to Read port of AXI4 bus for transferring data from MIG to AxiRxFIFO. AxiMtWr and AxiMtRd run in AxiClk domain which is very high frequency, so the logic is designed by using pipeline registers to forward data between MIG and AxiTxFIFO/AxiRxFIFO.

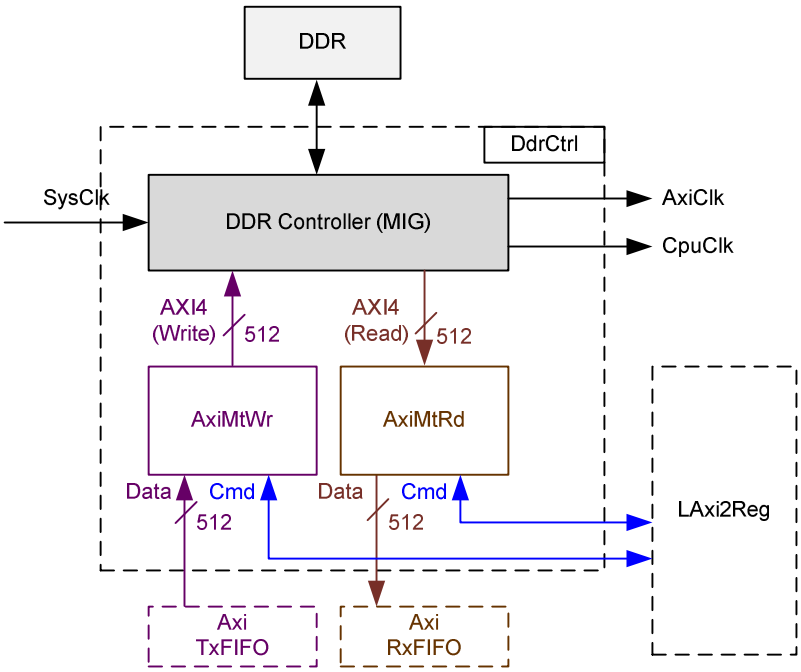

Figure 2-9 DdrCtrl hardware

2.3.1 AxiMtWr

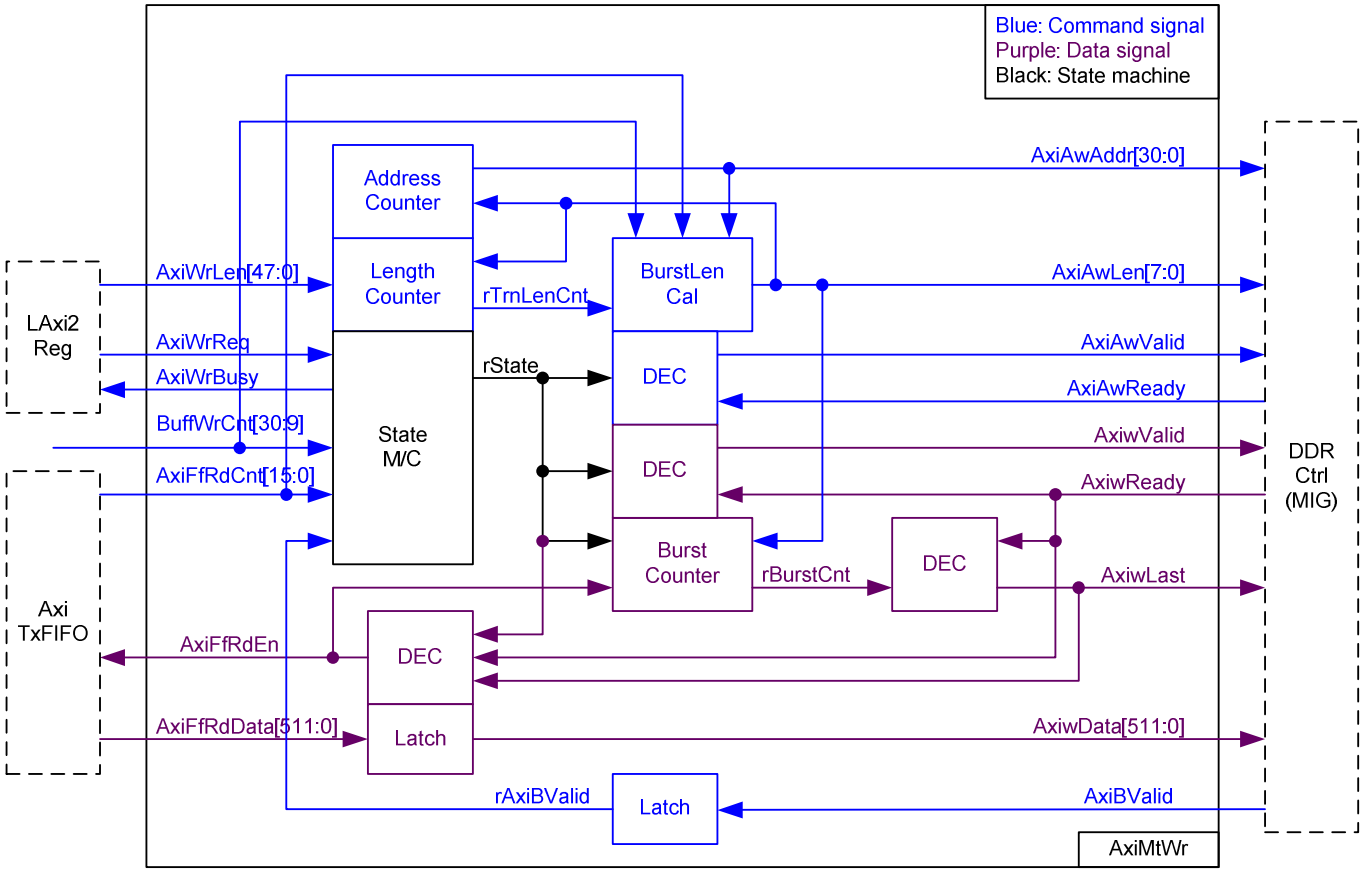

Figure 2-10 AxiMtWr hardware

As shown in Figure 2-10, state machine is designed in AxiMtWr to control the sequence for write transferring through AXI4 bus interface. The signals of AXI4 are separated into two groups, i.e. command and data signals. Command signals are shown in blue color at upper side and data signals are shown in purple color at lower side.

The sequence of write transaction by using AXI4 interface is as follows.
1) Send address and address control signals (AxiAwAddr, AxiAwLen, AxiAwValid)
2) Send data and data control signals (AxiwData, AxiwValid)
3) Wait response (AxiBValid).

State machine is designed following AXI4 interface sequence. After new write request (AxiWrReq) is detected, state machine changes to command request state. At the same time, Length counter loads total transfer size from AxiWrLen and Address counter is reset to 0. Next, state machine reads data counter of AxiTxFIFO (AxiFfRdCnt) and waits until numbers of data in AxiTxFIFO is enough for one burst size. Also, State machine reads data counter of DDR (BuffWrCnt) to confirm that DDR is not full. If AxiTxFIFO and DDR are ready, state machine will generate write request to DDR (AxiAwValid) with the valid value of DDR address (AxiAwAddr) and length (AxiAwLen). After MIG asserts AxiAwValid to '1' to send acknowledge that current command is received, state machine changes to data transfer state.

To transfer data from AxiTxFIFO to DDR, AxiFfRdEn is asserted to '1' to read data from FIFO. Next, the logic asserts AxiwValid='1' to write data to DDR. Data from AxiTxFIFO is forwarded to latch register before forwarding to DDR (AxiwData). Two step pipeline registers in data path are used to solve timing constraint problem. rBurstCnt is designed to count data in the burst. Data transfer state is completed after rBurstCnt shows that current data is the last data in the burst and asserts AxiwLast to '1'. Next, state machine changes to waiting response state.

AxiBValid is asserted to '1' after DDR completes to receive all data in one burst. After that, state machine checks the remaining transfer size by reading rTrnLenCnt. rTrnLenCnt loads total counter from AxiWrLen when starting the new transfer. rTrnLenCnt decreases the value after complete each burst transfer. If total data still not be transferred, state machine will change to command request state to start new transfer. If total data is complete, state machine will change to Idle.

From above sequence, address, data, and response in one burst are transferred sequentially by using one state machine, not pipelined transfer for simple design. To reduce the effect of overhead time from address request and waiting response, the maximum burst size could be extended to 4kByte size. BurstLen Cal block reads FIFO/DDR status, current address alignment, and remaining transfer size to calculate burst size for the next loop. Four sizes can be set, i.e. 512 byte, 1 kByte, 2 kByte, and 4 kByte. Using bigger burst size reduces the effect of overhead time and transfer performance is increased. Burst size could be set to 4 kByte when following conditions are met.

1) Remaining transfer size (rTrnLenCnt) is more than or equal to 4 kByte (bit[47:3] is not equal to 0).
2) Current address (AxiAwAddr) is aligned to 4 kByte (bit[11:0] is equal to 0).
3) Data in AxiTxFIFO (AxiFfRdCnt) is more than or equal to 4 kByte (bit[15:6] is not equal to 0).
4) Data in DDR (BuffWrCnt) has at least 8 kByte free space (bit[30:13] is not equal to all 1).

AxiAwLen is the burst size calculated by BurstLen Cal. AxiAwLen is fed to Address counter to calculate the next address (AxiAwAddr) and fed to Length Counter to calculate remaining transfer size (rTrnLenCnt).

The basic operation of AxiMtWr is displayed in timing diagram in Figure 2-11.

Figure 2-11 shows the operation of AxiMtWr when burst size is equal to 8 for transferring 512 byte data.

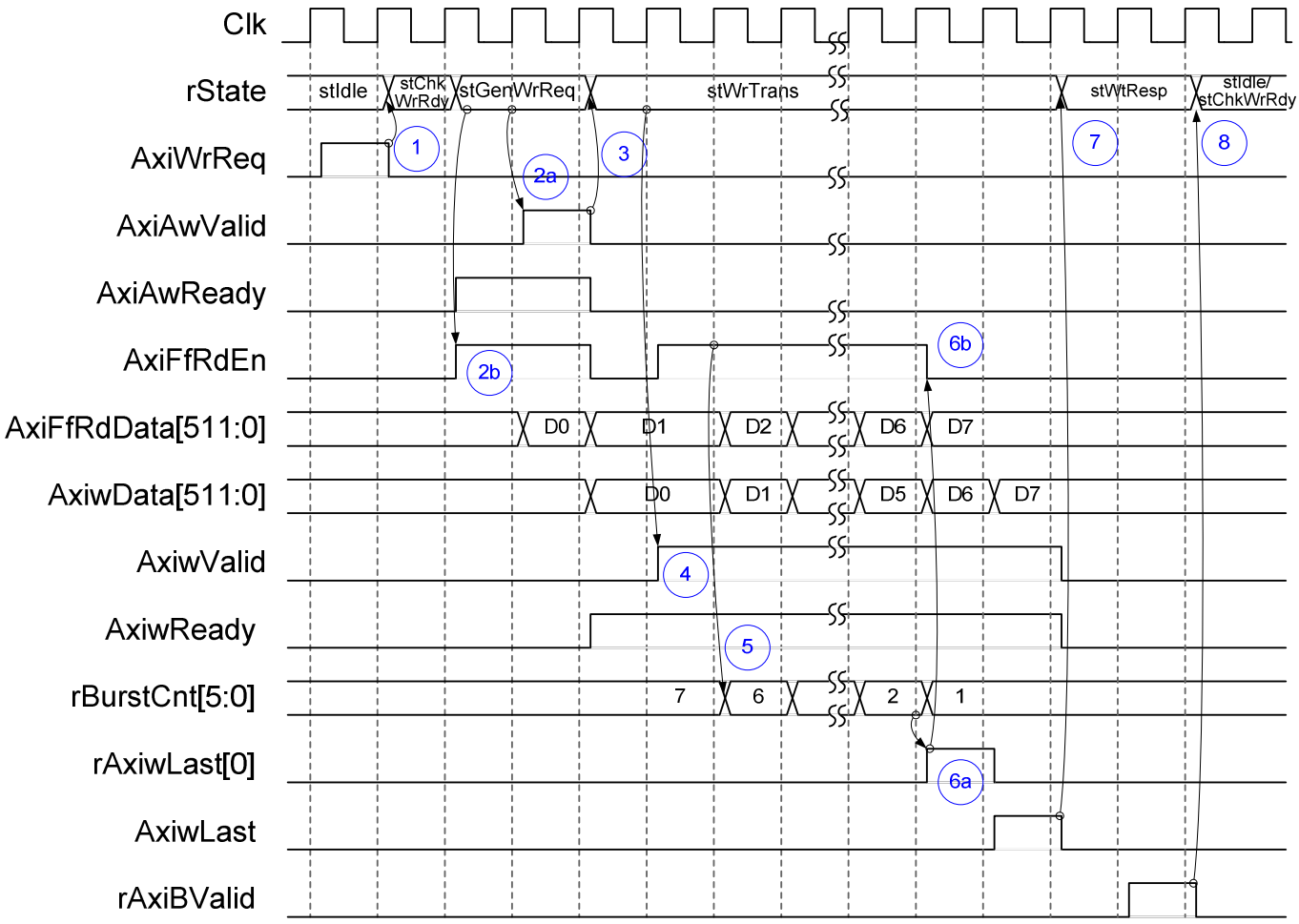

Figure 2-11 AxiMtWr Timing diagram when burst size = 512 byte

After AxiWrReq is asserted to '1', state machine changes to stChkWrRdy to monitor data size in FIFO and DDR. If FIFO and DDR are ready to transfer one sector data, state machine will change to next state. stGenWrReq is designed for two functions, i.e. asserting AxiAwValid='1' for generating write request until AxiAwReady='1' and asserting AxiFfRdEn=1' for two clock cycles to read two data from AxiTxFIFO. After write request is received (AxiAwValid='1' and AxiAwReady='1'), state machine changes to stWrTrans to transfer 8 data.

AxiwValid is asserted to '1' in stWrTrans with the valid value of AxiwData which stores previous data of AxiFfRdData. During transferring, rBurstCnt is decreased by AxiFfRdEn signal. When rBurstCnt=2 and AxiwReady='1', rAxiwLast[0] is asserted to '1' to pause data reading from FIFO (AxiFfRdEn='0'). After that, AxiwLast (the previous status of rAxiwLast[0]) is asserted to '1' with the last data available on the bus to complete data transfer.

State machine changes to stWtResp to wait until AxiBValid asserting to '1'. State machine changes from stWtResp to stIdle when total data are transferred. If remaining size is not equal to 0, state machine will go back to stChkWrRdy to start the next burst.

2.3.2 AxiMtRd

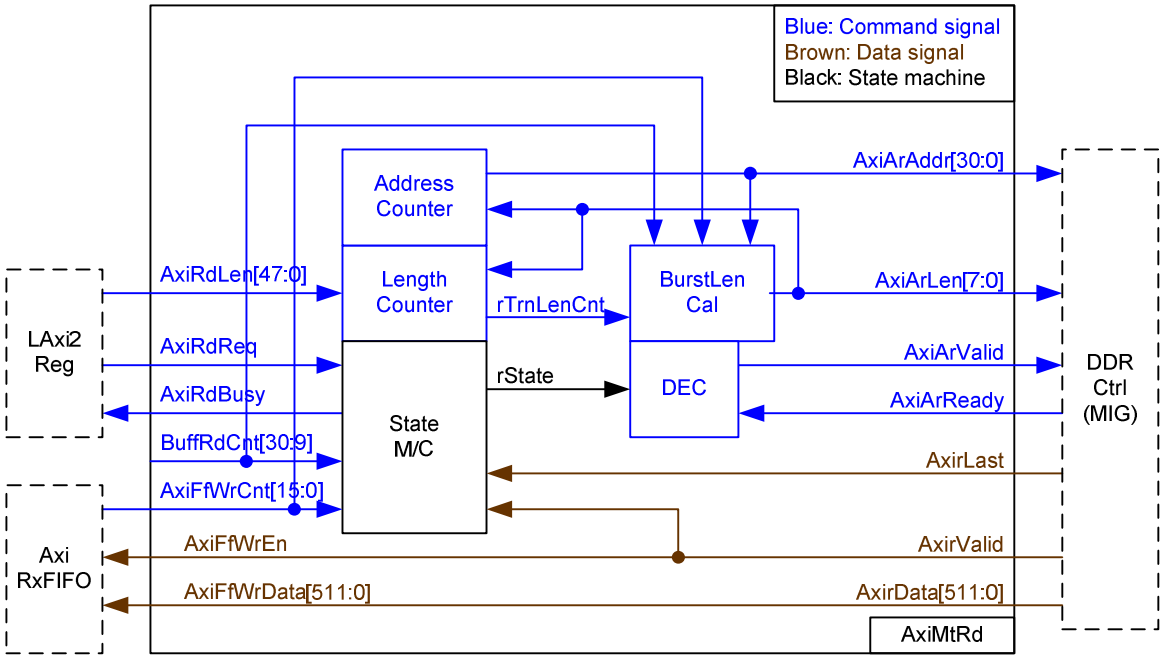

Figure 2-12 AxiMtRd hardware

Comparing to AxiMtWr, the logic inside AxiMtRd is simpler. State machine is designed to control the sequence of command and data following AXI4 protocol. Command signals are shown in blue color at upper side and data signals are shown in brown color at lower side.

After new read request (AxiRdReq) is asserted to '1', state machine changes to command request state. At the same time, total transfer size (AxiRdLen) is loaded to Length counter and Address counter is reset to 0. Before transferring data of each burst size, state machine checks that data available in DDR is enough by monitoring BuffRdCnt and free space of AxiRxFIFO is enough by monitoring AxiFfWrCnt. Then, state machine asserts read request to DDR (AxiArValid='1') with the valid value of DDR address (AxiArAddr) and burst size (AxiArLen). After MIG asserts AxiArValid to '1' to send acknowledge that current command is received, state machine changes to data transfer state.

Data from DDR is forwarded to AxiRxFIFO when AxirValid='1'. AxirReady is fixed to '1' to always receive the data from DDR. AxirLast is asserted to '1' to complete one burst transfer. After that, state machine checks the remaining transfer size by reading rTrnLenCnt. rTrnLenCnt loads total counter from AxiRdLen when starting the new transfer. rTrnLenCnt decreases the value after complete each burst transfer. If total data still not be transferred, state machine will change to command request state to start new transfer. If total data is complete, state machine will change to Idle.

Similar to AxiMtWr, BurstLen Cal block reads FIFO/DDR status, current address alignment, and remaining transfer size to calculate burst size for the next loop. Four sizes can be set, i.e. 512 byte, 1 kByte, 2 kByte, and 4 kByte. Using bigger burst size reduces the effect of overhead time and transfer performance is increased. Burst size could be set to 4 kByte when following conditions are met.

1) Remaining transfer size (rTrnLenCnt) is more than or equal to 4 kByte (bit[47:3] is not equal to 0).
2) Current address (AxiArAddr) is aligned to 4 kByte (bit [11:0] is equal to 0).
3) Data in DDR (BuffRdCnt) is more than or equal to 4 kByte (bit [30:12] is not equal to 0).
4) Data in AxiRxFIFO (AxiFfWrCnt) has at least 8 kByte free space (bit[15:7] is not equal to all 1).

AxiArLen is the burst size calculated by BurstLen Cal. AxiArLen is fed to Address counter to calculate the next address (AxiArAddr) and fed to Length Counter to calculate remaining transfer size (rTrnLenCnt). The basic operation of AxiMtRd is displayed in timing diagram in Figure 2-13.

Figure 2-13 shows the operation of AxiMtRd when burst size is equal to 8 for transferring 512 byte data.

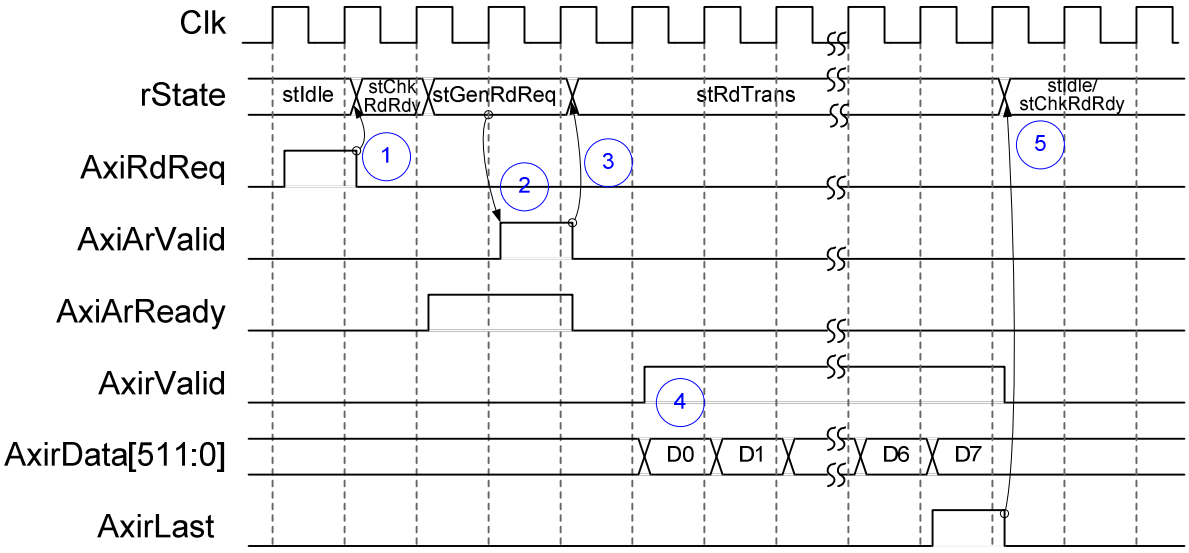

Figure 2-13 AxiMtRd Timing diagram when burst size = 512 byte

After AxiRdReq is asserted to '1', state machine changes to stChkRdRdy to monitor FIFO and DDR status. If FIFO and DDR are ready to transfer one sector data, state machine will change to next state. In stGenRdReq state, AxiArValid is asserted to '1' for DDR read request. After read request is received (AxiArValid='1' and AxiArReady='1'), state machine changes to stRdTrans to wait 8 data sent from DDR. When DDR returns data, AxirValid is asserted to '1' with the valid value of AxirData. AxirValid and AxirData are bypassed to be FIFO write enable (AxiFfWrEn) and FIFO write data (AxiFfWrData). AxirLast is asserted to '1' when transferring the last data. After transferring the last data and remaining size is equal to 0, state machine changes from stRdTrans to stIdle. If remaining size is not zero, state machine will go back to stChkRdRdy to start new burst loop.

2.4 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. The control and status registers for CPU access are designed in LAxi2Reg.

LAxi2Reg connects to many hardwares in the system such as DdrCtrl, TestGen, NVMe-IP, and IdenRAM to get the control and status signals of each module. As shown in Figure 2-14, there are three clock domains applied in this block, i.e. DdrClk which is used to interface with AXI4 bus of DdrCtrl, CpuClk which is used to interface with AXI4-Lite bus of CPU, and UserClk which is user clock domain for TestGen and NVMe-IP.

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

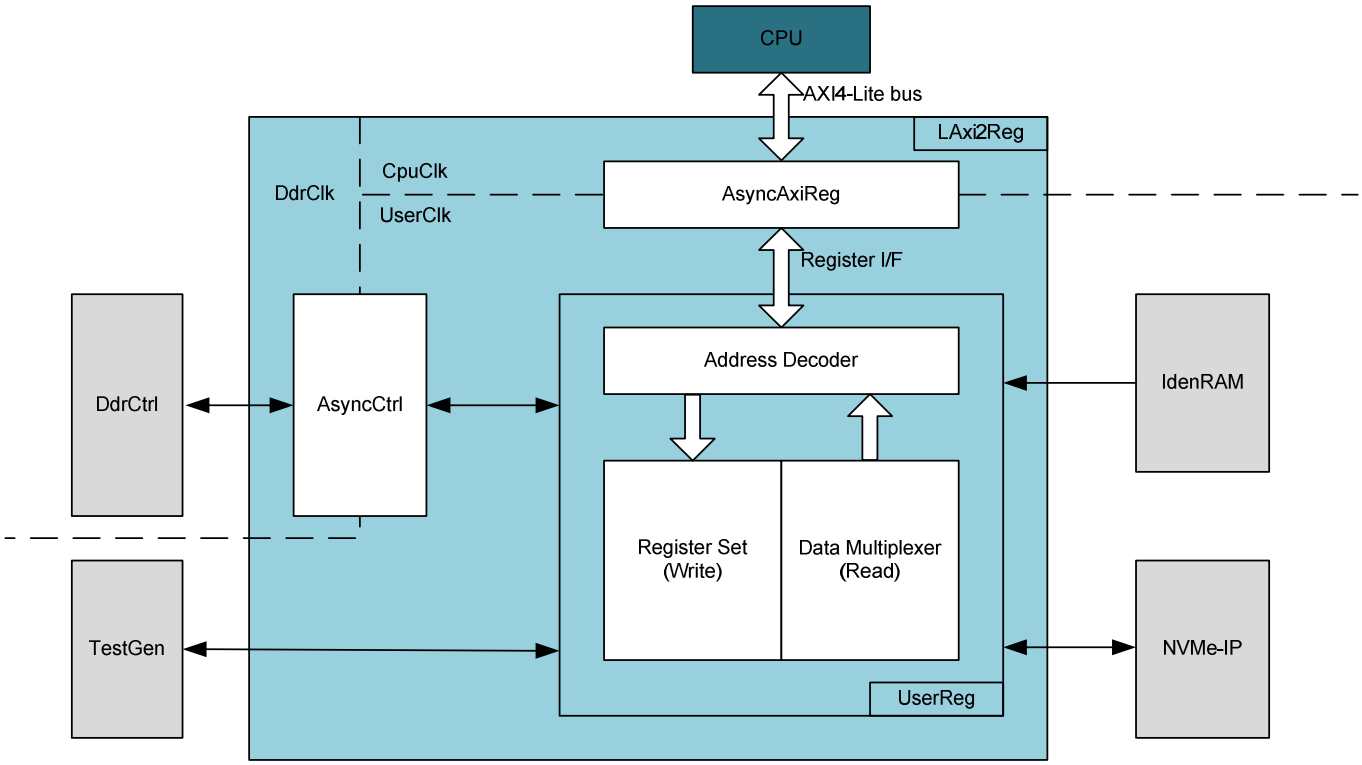

Figure 2-14 CPU and peripherals hardware

2.4.1 AsyncCtrl

This module is designed to be asynchronous circuit of control and status signals between DdrClk and UserClk. Two register types are designed to cross clock domain. If the signal is valid only one clock cycle, latch register will be applied to latch the signal in source clock domain before transferring to destination clock domain. If the signal is stable for long time, simple D Flip-Flip will be applied for transferring signal from source clock domain to destination clock domain.

2.4.2 AsyncAxiReg

This module is designed to convert the signal interface of AXI4-Lite to be register interface. Also, it supports to convert clock domain from CpuClk to be UserClk domain. Timing diagram of register interface is shown in Figure 2-15.

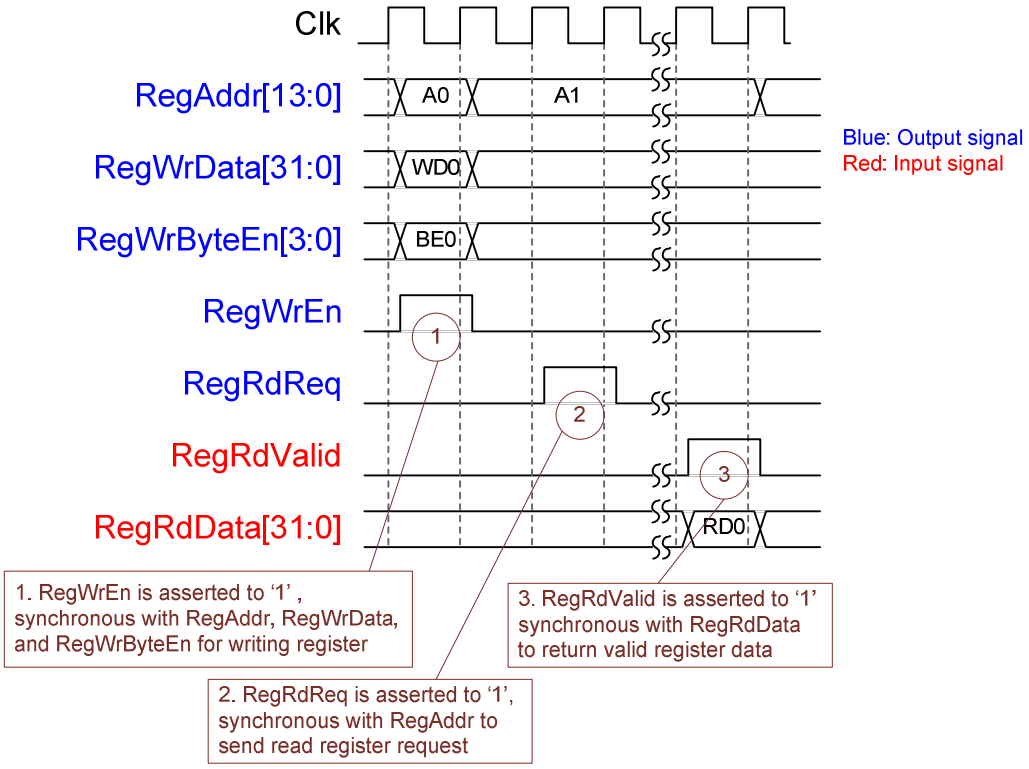

Figure 2-15 Register interface timing diagram

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] for RegWrData[7:0], bit[1] for RegWrData[15:8], …, and bit[3] for RegWrData[31:24]).

To read register, AsyncAxiReg asserts RegRdReq='1' with the valid value of RegAddr (the register address in 32-bit unit). After that, the module waits until RegRdValid is asserted to '1' to get the read data through RegRdData signal. During read access, RegAddr holds the same value until RegRdValid is asserted to '1'.

2.4.3 UserReg

The details of UserReg module is shown in Figure 2-14. After RegWrEn or RegRdReq is asserted to '1' by AsyncAxiReg to request write or read register access, RegAddr is read by Address decoder to select the active register. For write access, RegWrData signal is loaded to be the new value for the requested 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, there are many status signals for CPU access such as DdrCtrl, TestGen, NVMe-IP, and IdenRAM. So, data multiplexer with pipelines register are designed to select the read data to return to CPU following RegAddr value. RegRdValid is designed by using three D Flip-flops, input by RegRdReq signal. So, the read access has three clock cycles latency after RegRdReq is asserted to '1'.

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 ""nvmeipddrterst.c"")	Description
		NVMe-IP
BA+0x000 Wr	User Address (Low) Reg (USRADRL_REG)	[31:0]: Input to be start sector address (UserAddr[31:0] of dgIF typeS for RAID0)
BA+0x004 Wr	User Address (High) Reg (USRADRH_REG)	[15:0]: Input to be start sector address (UserAddr[47:32] of dgIF typeS for RAID0)
BA+0x008 Wr	User Length (Low) Reg (USRLENL_REG)	[31:0]: Input to be transfer length in sector unit (UserLen[31:0] of dgIF typeS for RAID0)
BA+0x00C Wr	User Length (High) Reg (USRLENH_REG)	[15:0]: Input to be transfer length in sector unit (UserLen[47:32] of dgIF typeS for RAID0)
BA+0x010 Wr	User Command Reg (USRCMD_REG)	[1:0]: Input to be user command (UserCmd of dgIF typeS for NVMe-IP). "00"-Identify device, "10"-Write Dev, "11"-Read Dev. When this register is written, the design generates command request to NVMe-IP to start new command operation.
BA+0x100 Rd	Total device size (Low) Reg (LBASIZEL_REG)	[31:0]: Total capacity of SSD in sector unit (LBASize[31:0] of dgIF typeS for NVMe-IP)
BA+0x104 Rd	Total device size (High) Reg (LBASIZEH_REG)	[15:0]: Total capacity of SSD in sector unit (LBASize[47:32] of dgIF typeS for NVMe-IP)
BA+0x108 Rd	User Status Reg (USRSTS_REG)	[0]: UserBusy of dgIF typeS for NVMe-IP ('0': Idle, '1': Busy) [1]: UserError of dgIF typeS for NVMe-IP ('0': Normal, '1': Error)
BA+0x110 Rd	User Error Type Reg (USRERRTYPE_REG)	[31:0]: User error status, directly mapped from UserErrorType[31:0] of NVMe-IP
BA+0x120 Rd	PCIe Status Reg (PCISTS_REG)	[0]: PCIe linkup status ('0': No linkup, '1': linkup) [3:2]: PCIe link speed ("00": No link up, "01": PCIe Gen1, "10": PCIe Gen2, "11": PCIe Gen3) [7:4]: PCIe link width ("0001": 1 lane, "0010": 2 lane, "0100": 4 lane, "1000": 8 lane) [13:8]: Current LTSSM State of PCIe-IP
BA+0x128 Rd	Completion Status Reg (COMPSTS_REG)	[15:0]: Status from Admin completion, directly mapped from AdmCompStatus[15:0] of NVMe-IP [31:16]: Status from IO completion, directly mapped from IOCompStatus[15:0] of NVMe-IP
BA+0x130 Rd	NVMe CAP Reg (NVMCAP_REG)	[31:0]: NVMe capability register output from SSD, directly mapped to NVMeCAPReg[31:0] of NVMe-IP
BA+0x138 Rd	NVMe IP Test pin Reg (NVMTESTPIN_REG)	[31:0]: IP test pin, directly mapped to TestPin[31:0] of NVMe-IP

Address Rd/Wr	Register Name (Label in ""nvmeipddrterst.c"")	Description
		TestGen block
BA+0x200 Wr	Test Pattern Reg (PATTSEL_REG)	[2:0]: Test pattern select "000"-Increment, "001"-Decrement, "010"-All 0, "011"-All 1, "100"-LFSR
BA+0x204 Wr	Transfer Ratio Reg (TRNRATIO_REG)	Set the sustain rate for write or read access of TestGen module. [11:8]: the numerator of the ratio, [3:0]: the denominator of the ratio. Transfer Rate (bit/sec) = (Numerator/Denominator) x UserClk (250MHz) x 128-bit
BA+0x208 Wr	DDR Read threshold (DDRRDTHR_REG)	[11:0]: The threshold value of DDR buffer in MB unit. TestGen module waits until the data in DDR is more than or equal to the threshold value before start reading the 1st data from DDR as sustain rate when user sends Read command (USRCMD_REG="11"). Valid range is 0-4095 MB.
BA+0x300 Rd	TestGen status PATTSTS_REG	[0]: Busy flag of write operation from TestGen. ('0': No operation, '1': Write is operating). [1]: Busy flag of read operation from TestGen ('0': No operation, '1': Read is operation). Note that this flag is asserted after data in DDR is equal or more than threshold value, set by DDRRDTHR_REG. [2]: Busy flag of write operation from DdrCtrl ('0': No operation, '1': Write is operating). [3]: Busy flag of read operation from DdrCtrl ('0': No operation, '1': Write is operating). [4]: Verification Fail ('0': Normal, '1': Data verification is failed) [5]: Error from buffer overflow during Write command ('0': Normal, '1': AxiTxFIFO is full during sustain write) [6]: Error from buffer underflow during Read command ('0': Normal, '1': AxiRxFIFO is empty during sustain read)
BA+0x320 Rd	Expected value Word0 Reg (EXPPATW0_REG)	[31:0]: Latch value of expected data [31:0] in TestGen during Read command
BA+0x324 Rd	Expected value Word1 Reg (EXPPATW1_REG)	[31:0]: Latch value of expected data [63:32] in TestGen during Read command
BA+0x328 Rd	Expected value Word2 Reg (EXPPATW2_REG)	[31:0]: Latch value of expected data [95:64] in TestGen during Read command
BA+0x32C Rd	Expected value Word3 Reg (EXPPATW3_REG)	[31:0]: Latch value of expected data [127:96] in TestGen during Read command
BA+0x340 Rd	Read value Word0 Reg (RDPATW0_REG)	[31:0]: Latch value of read data [31:0] in TestGen during Read command
BA+0x344 Rd	Read value Word1 Reg (RDPATW1_REG)	[31:0]: Latch value of read data [63:32] in TestGen during Read command
BA+0x348 Rd	Read value Word2 Reg (RDPATW2_REG)	[31:0]: Latch value of read data [95:64] in TestGen during Read command
BA+0x34C Rd	Read value Word3 Reg (RDPATW3_REG)	[31:0]: Latch value of read data [127:96] in TestGen during Read command
BA+0x360 Rd	Data Failure Address (Low)Reg (RDFAILNOL_REG)	[31:0]: Latch value of failure address [31:0] in byte unit from Read command
BA+0x364 Rd	Data Failure Address (High)Reg (RDFAILNOH_REG)	[24:0]: Latch value of failure address [56:32] in byte unit from Read command
BA+0x368 Rd	Current test byte (Low) Reg (CURTESTSIZEL_REG)	[31:0]: Current test data size of TestGen module in byte unit (bit[31:0])
BA+0x36C Rd	Current test byte (High) Reg (CURTESTSIZEH_REG)	[24:0]: Current test data size of TestGen module in byte unit (bit[56:32])
BA+0x370 Rd	DDR Write Address Reg (DDRWRADDR_REG)	[31:0]: Current DDR write address in byte unit (bit[8:0] is fixed to 0 to align 512-byte unit)
BA+0x374 Rd	DDR Read Address Reg (DDRRDADDR_REG)	[31:0]: Current DDR read address in byte unit (bit[8:0] is fixed to 0 to align 512-byte unit)

Address Rd/Wr	Register Name (Label in ""nvmeipddrterst.c"")	Description
Identify RAM		
BA+0x2000 – 0x2FFF	Identify Controller Data (IDENCTRL_REG)	4Kbyte Identify controller data structure
BA+0x3000 – 0x3FFF	Identify Device Command 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 is link up (PCISTS_REG[0]='1') and NVMe-IP is not busy (USRSTS_REG[0]='0'). Finally, Main menu is displayed.

CPU firmware supports three commands, following USRCMD_REG value, i.e. "00" for Identify device, "10" for Write, and "11" for 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 device command is below.

1) Set USRCMD_REG="00" and send request to NVMe-IP. NVMe-IP busy flag (USRSTS_REG[0]) changes from '0' to '1' after receiving new command.
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, error message is displayed on the console. If the command is completed, the data from Identify command will be stored to IdenRAM.
3) CPU reads Identify data from IdenRAMs which are mapped to IDENCTRL_REG address. Then, CPU displays the information such as SSD model name on Serial console. Also, SSD capacity (LBASIZEL/H_REG) is displayed in GB unit on the console.

3.2 Write/Read command

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

1) Receive start address, transfer length, test pattern, sustain rate, DDR threshold value (DDR threshold is the parameter for Read command only) from user 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, TRNRATIO_REG, DDRRDTHR_REG, PATTSEL_REG, and USRCMD_REG (USRCMD_REG="10" for write transfer and "11" for read transfer).
3) CPU waits until the operation is completed or some errors (except verification error) are found by monitoring USRSTS_REG[1:0] and PATTSTS_REG[6:0]. PATTSTS_REG[3:0] is monitored because CPU needs to wait until PattGen module completes to verify all data from DDR in Read command.
   If some bits of PATTSTS_REG[6:5] (buffer overflow/underflow flag) are asserted to '1', the operation will be cancelled with error message displayed. Next, CPU waits for 2 sec (NVMe-IP timeout value set in HDL code) before reading USRSTS_REG[1]. The error from NVMe-IP can be caused to buffer overflow and underflow error. If USRSTS_REG[1] is equal to '1', the buffer error will be caused from NVMe-IP error. In this error condition, Error message from NVMe-IP is displayed on the console.
   If PATTSTS_REG[4] is asserted to '1', verification error message will be displayed. After that, CPU still runs until end of operation or user inputs any key.
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.

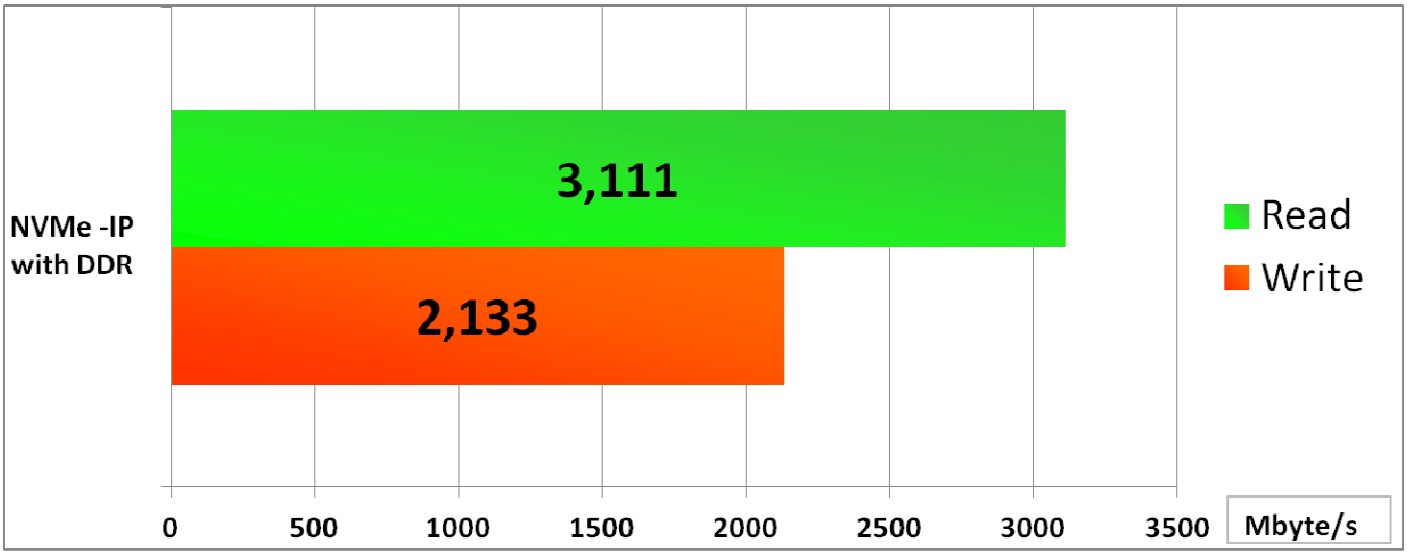

Figure 4-1 Test Performance of NVMe-IP with DDR demo by using Samsung 960 Pro SSD

When running NVMe-IP with DDR demo on KCU105 board, write sustain performance is about 3,100 Mbyte/sec and read sustain performance is about 2,100 Mbyte/sec.

5 Revision History

Revision	Date	Description
1.0	17-Apr-18	Initial version release