2-ch RAID0 (NVMe-IP for Gen4) reference design manual

Rev1.0 16-Aug-22

1 Introduction

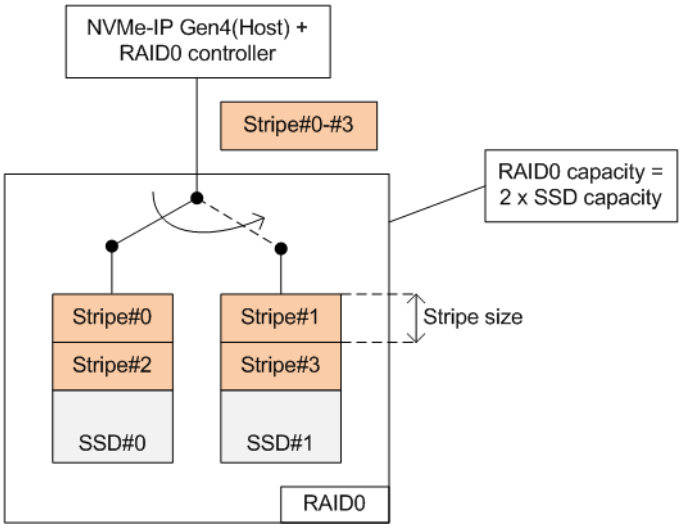


Figure 1-1 RAID0 by 2 SSDs data format

RAID0 system uses multiple storages to extend total storage capacity and increase write/read performance. Assumed that total number of devices connecting in RAID0 system is N, total storage capacity of RAID0 is equal to N times of one device capacity. Write and read performance of RAID0 are almost equal to N times of one device performance.

Data format of RAID0 is shown in Figure 1-1. Data stream of the host side is split into a small stripe for transferring with one SSD at a time. Stripe size is the data size transferring with one SSD before switching to other SSDs. In RAID0 reference design, stripe size is equal to 512-byte.

In this demo, two SSDs are connected in the system. It is recommended to use the same SSD model for all channels to match the characteristic and achieve the best performance. As a result, the total capacity is equal to two times of one SSD and the write/read performance are almost equal to two times of one SSD performance.

The demo uses FIFO implemented by BlockRAM to be the buffer which has smaller size than using DDR. Therefore, the buffer is sometimes not ready to transfer data when SSD pauses data transmission for long time in Write process. Test performance in the demo is average speed, not sustain rate. User can modify RAID0 reference design by increasing the numbers of SSD to achieve the better performance and the bigger capacity. Furthermore, user can add DDR to have larger data buffer in the system for supporting high-speed transferring as sustain rate with some SSDs.

Before running the reference design, it is recommended to read NVMe-IP for Gen4 datasheet and single channel demo from following link.
https://dgway.com/products/IP/NVMe-IP/dg_nvme_ip_data_sheet_g4_en.pdf
https://dgway.com/products/IP/NVMe-IP/dg_nvmeip_refdesign_g4_en.pdf
https://dgway.com/products/IP/NVMe-IP/dg_nvmeg4ip_instruction_en.pdf

2 Hardware overview

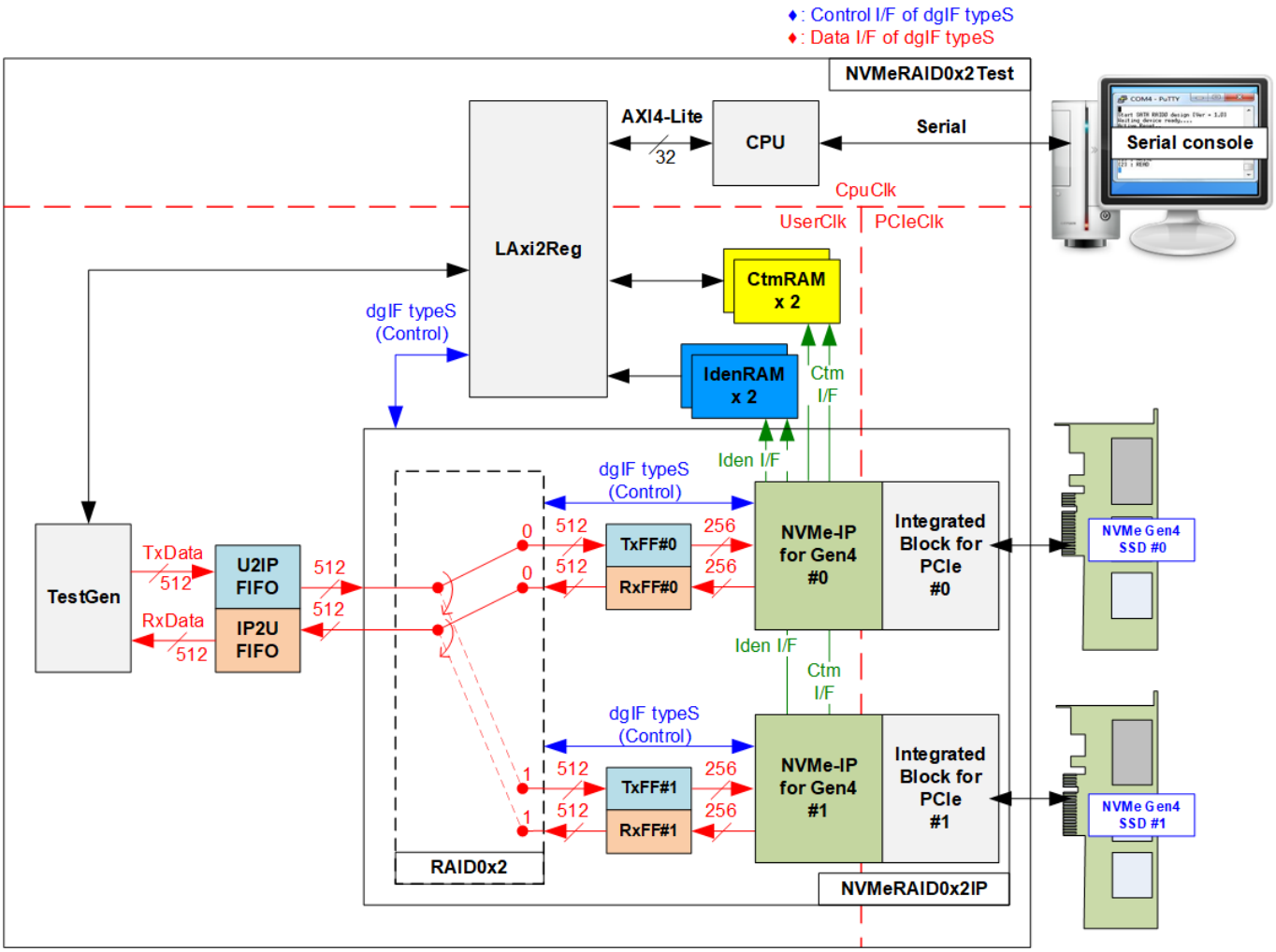


Figure 2-1 RAID0x2 demo system by using NVMe-IP for Gen4

Comparing to the NVMe-IP reference design without RAID0x2, TestGen is modified to support 512-bit data bus, instead of 256-bit data bus. Also, CtmRAM and IdenRAM are extended from one block for one SSD to two blocks for two SSDs. The control interface and the data interface of NVMeRAID0x2IP block are similar to NVMe-IP by using dgIF typeS. In Write command, the data is transferred from the left-hand side to the right-hand side while the data direction is reversed for Read command. CPU and LAxi2Reg are integrated to be user control via Serial console. The user can set the test parameters and monitor the test status via the console.

NVMeRAID0x2IP is the top module of RAID0 block that consists of RAID0x2, FIFOs, two NVMe-IPs, and two PCIe hard IPs. The top RAID0 is designed to have the same user interface as NVMe-IP (dgIF typeS) for connecting with TestGen via two FIFOs – U2IPFIFO and IP2UFIFO.

The CPU firmware implements all commands supported by NVMe-IP. When running SMART command or Identify command, CPU sends the command and the parameters of the command to LAxi2Reg via AXI4-Lite I/F. After that, the command request is created to RAID0x2 and NVMe-IP via Control I/F of dgIF typeS. Finally, the SMART data or Identify controller and namespace data is returned to CtmRAM or IdenRAM, respectively. CPU can decode and display the information of SMART data and Identify data by reading the data from CtmRAM/IdenRAM via AXI4-Lite.

When running Write command, the command is still requested by CPU via Control I/F of dgIF typeS to RAID0x2 and NVMe-IP. While the data of Write command is generated by TestGen module at maximum transfer speed to check the best write performance of the test system. The data is always created and stored to U2IPFIFO when the FIFOs have free space. Next, RAID0x2 block reads the data from U2IPFIFO to store to TxFF#0/TxFF#1, so the data is always available for NVMe-IP to send Write data to NVMe SSD. On the contrary, the data direction is reversed when running Read command. The command is still requested by CPU. NVMe-IP stores the read data from NVMe SSD to RxFF#0/RxFF#1. After that, RAID0x2 forwards the read data from RxFF#0/RxFF#1 to IP2UFIFO. Finally, TestGen reads the data from IP2UFIFO and verify them with the expected value. The verification result (pass or failed) is returned to CPU to be the test result. CPU firmware measures the time usage when running Write command or Read command to calculate the Write/Read performance for displaying as the test result on the console.

For operating with NVMe SSD by Gen4 speed, PCIe hard IP is run at 250 MHz which is the frequency of PCIeClk. According to NVM-IP specification, user clock frequency (UserClk) is recommended to be higher or equal to PCIeClk. However, RAID0 module has the overhead time for switching the active NVMe-IP module. The overhead time is about 12.5%, so it is recommended to set UserClk to be higher or equal to 281.25 MHz (112.5% of 250 MHz). While CPU system is mostly built by using its own clock which is individual and stable. Therefore, LAxi2Reg must integrate clock-crossing domain logic for support different clock domain between CPU system and Test logic.

More details of the hardware are described as follows.

2.1 TestGen

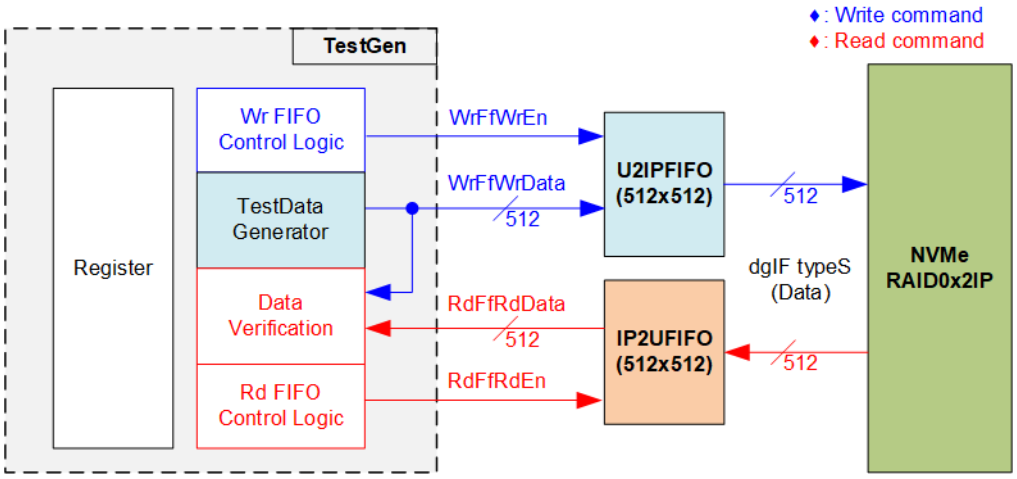


Figure 2-2 TestGen interface

TestGen module is the test logic to send test data to NVMeRAID0x2IP through U2IPFIFO when operating Write command. On the other hand, the test data is applied to be the expected value to verify the read data from NVMeRAID0x2IP through IP2UFIFO when operating Read command. Control logic asserts Write enable or Read enable to '1' when the FIFOs are ready. Data bandwidth of TestGen is matched to NVMeRAID0x2IP by running at the same clock and using the same data bus size. Therefore, U2IPFIFO and IP2UFIFO are always ready for transferring data with NVMeRAID0x2IP in Write and read command. As a result, the test logic shows the best performance to write and read data with the SSD through NVMeRAID0x2IP.

Register file in the TestGen receives test parameters from user, i.e., total transfer size, transfer direction, verification enable, and test pattern. To control transfer size, the counter counts total amount of transferred data. The details of hardware logic within TestGen module are shown in Figure 2-3.

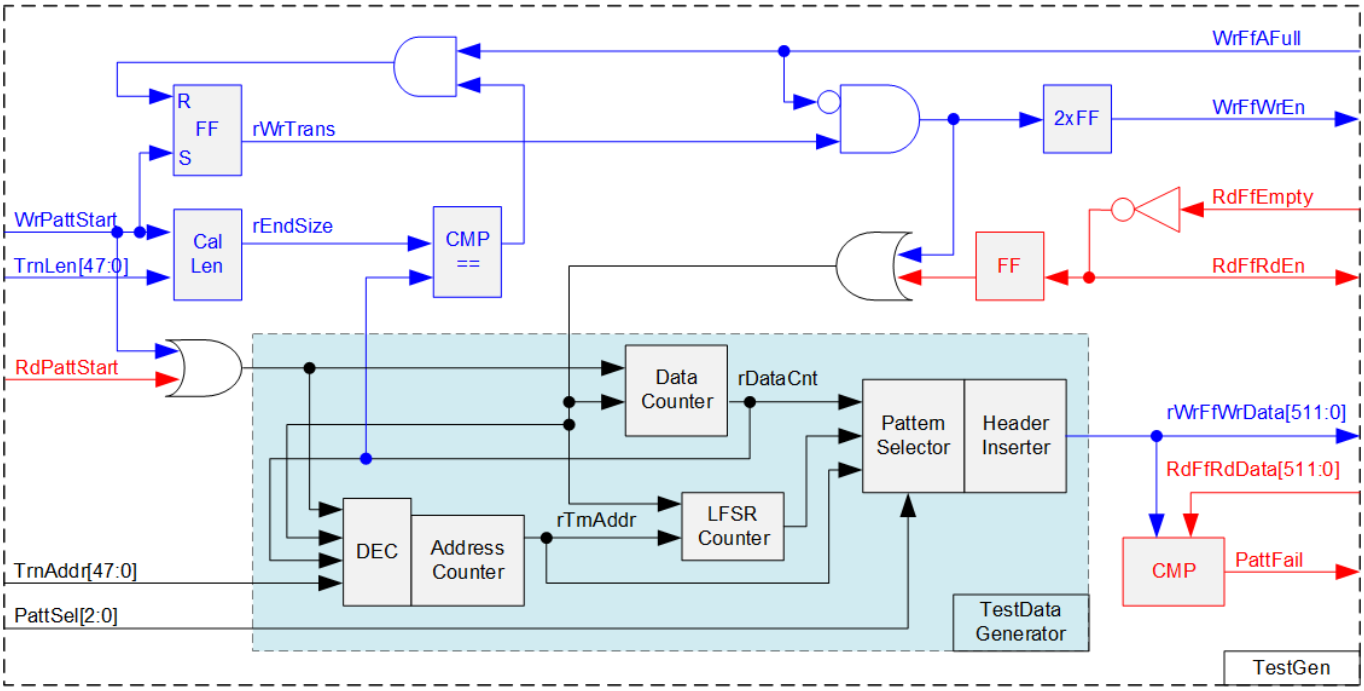


Figure 2-3 TestGen hardware

As shown in the right side of Figure 2-3, flow control signals of FIFO are WrFfAFull and RdFfEmpty. When FIFO is almost full in 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'.

Two counters – Data counter and Address counter are designed in TestGen. Data counter (rDataCnt) counts the amount of transferred data in Write command and Read command. When total amount of transferred data is equal to the end size, set by user, write enable or read enable of FIFO is de-asserted to '0'. Also, the Data counter (rDataCnt) are fed to be the write data for Write command or the expected data for Read command. TestGen supports to generate five patterns of test data, i.e., all-zero, all-one, 32-bit incremental data, 32-bit decremental data, and LFSR counter, selected by Pattern Selector. When creating all-zero or all-one pattern, every bit of data is fixed zero or one, respectively. While other patterns consist of two data parts to create unique test data in every 512-byte data, as shown in Figure 2-4.

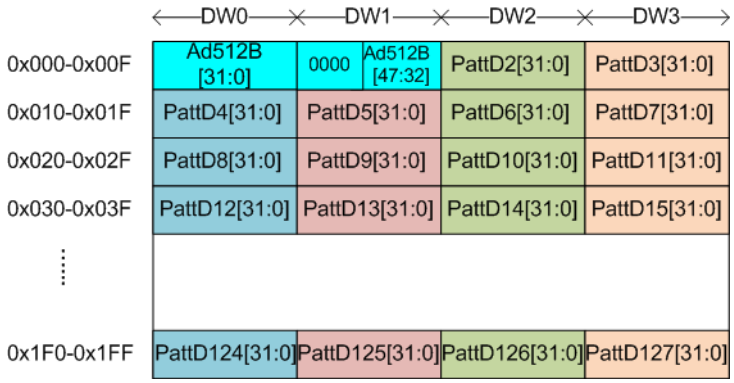


Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

512-byte data consists of 64-bit header in Dword#0 and Dword#1 and the test data in remaining words of 512-byte data (Dword#2 – Dword#127). The header is created by using the address counter (rTrnAddr) which shows the address in 512-byte unit. The initial value of rTrnAddr is set by user and it is increased when finishing transferring 512-byte data. Remaining Dwords (DW#2 – DW#127) depends on pattern selector which may be 32-bit incremental data, 32-bit decremental data, or LFSR counter. 32-bit incremental data is implemented by using rDataCnt. The decremental data can be designed by connecting NOT logic to rDataCnt. The LFSR pattern is designed by using LFSR counter. The equation of LFSR is x^31 + x^21 + x + 1. To implement 512-bit LFSR pattern, the data is split to be two sets of 128-bit data which uses the different start value as shown in Figure 2-5.

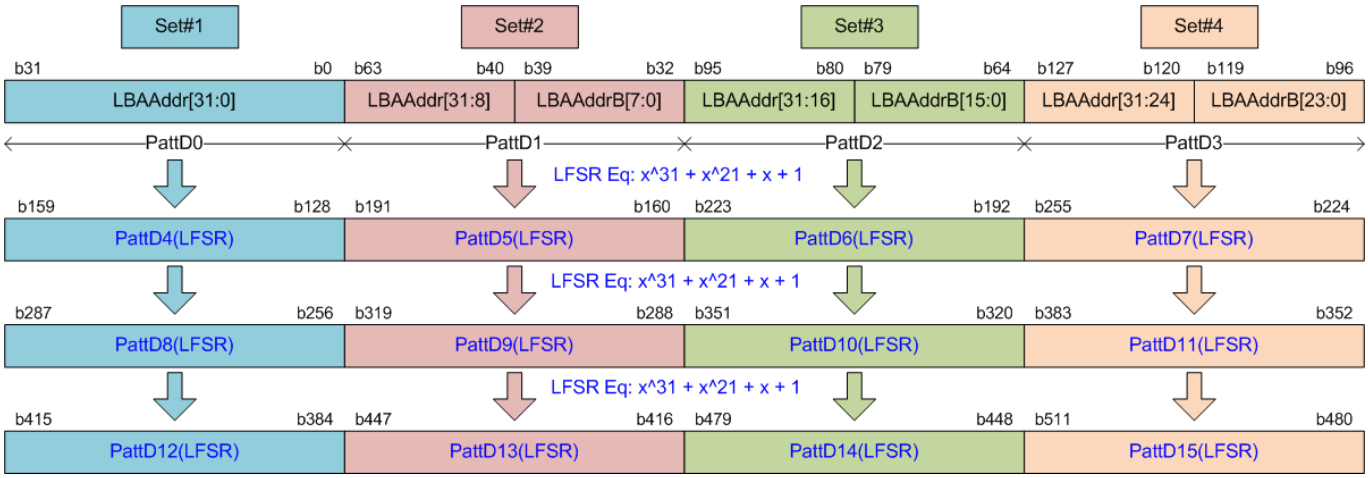


Figure 2-5 512-bit LFSR Pattern in TestGen

By using look-ahead technique, one cycle generates four 32-bit LFSR data or 128-bit data, the same color in Figure 2-5. The start value of each data set consists of 16 bits of LBAAddr (32-bit LBA address) and 16 bits of LBAAddrB (NOT logic of LBA address). 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 timing diagram to write data to FIFO is shown as follows.

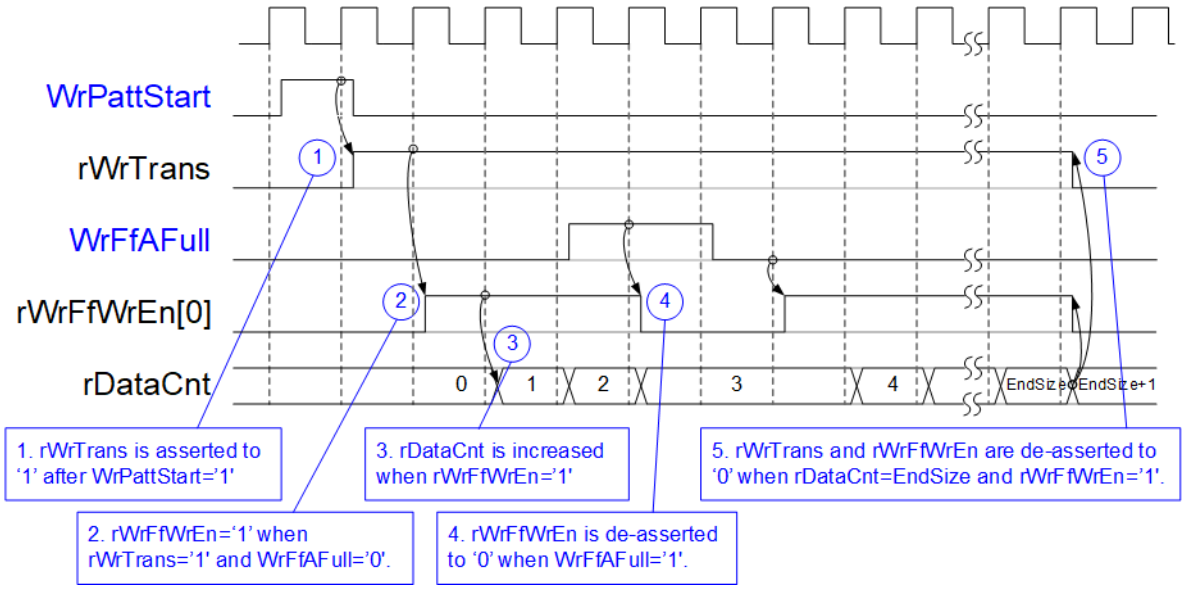


Figure 2-6 Timing diagram of Write operation in TestGen

1) WrPattStart is asserted to '1' for one clock cycle when user sets the register to start write operation. In the next clock, rWrTrans is asserted to '1' to enable the control logic for generating write enable to FIFO.
2) Write enable to FIFO (rWrFfWrEn) is asserted to '1' when two conditions are met. First, rWrTrans must be asserted to '1' to show the Write command is operating. Second, the FIFO must not be full by monitoring WrFfAFull='0'.
3) The write enable is fed to count total amount of data by rDataCnt in the Write command.
4) When FIFO is almost full (WrFfAFull='1'), the write process is paused by de-asserting rWrFfWrEn to '0'.
5) When rDataCnt is equal to the set value (rEndSize), rWrTrans is de-asserted to '0'. Meanwhile, rWrFfWrEn is also de-asserted to '0' to finish generating data.

For read operation, read enable of FIFO is controlled by empty flag of FIFO. Comparing to write enable, the read enable is not stopped by total amount of data and not started by start flag. The read enable is asserted to '1' when FIFO is not empty. The data counter and the address counter are increased when the read enable is asserted to '1' to count total amount of data and generate the header of expect value.

2.2 NVMeRAID0x2IP

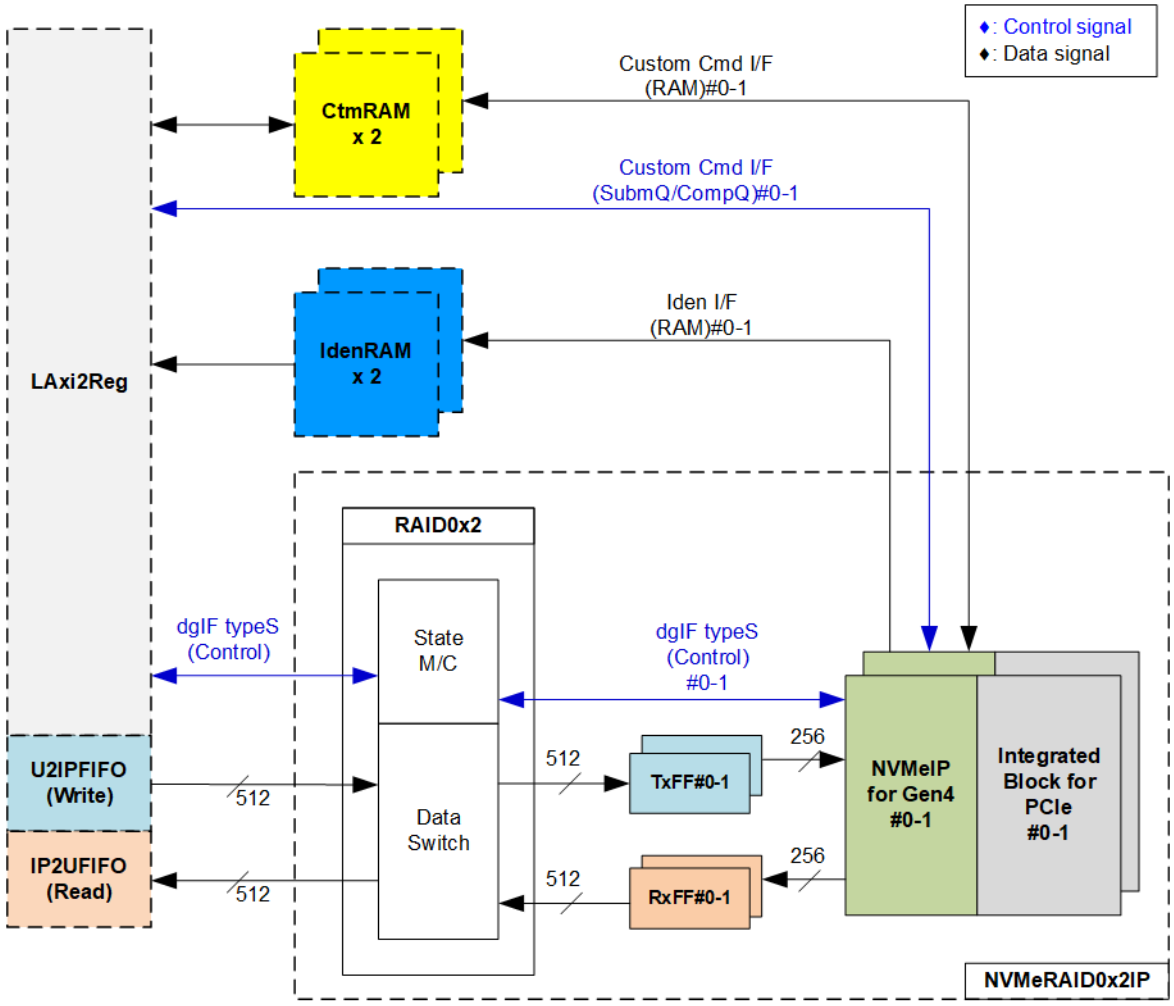


Figure 2-7 NVMeRAID0x2IP hardware

Figure 2-7 shows the connections and the submodules inside NVMeRAID0x2IP. The user interface of NVMeRAID0x2IP consists of control interface and 512-bit data interface while the data interface of NVMe-IP is 256 bits. Therefore, four FIFOs (TxFF#0/#1 and RxFF#0/#1) which are asymmetric FIFOs are connected between RAID0x2 and NVMe-IPs. RAID0x2 decodes the command request from control interface (by LAxi2Reg) and then creates the request with the calculated parameters to control interface of two NVMe-IPs. Also, RAID0x2 includes data switch to connect data interface of TestGen to one of two NVMe-IPs when running Write command or Read command. Therefore, the write data of TestGen is stored to two SSDs as RAID0 operation.

While the data interface of SMART command (Custom I/F) or Identify command (Iden I/F) are not mapped to RAID0x2 module but mapped to CtmRAM and IdenRAM directly. Also, the constant parameters of Custom I/F for running SMART command and Flush command are set by LAxi2Reg directly. More details of each submodule are described as follows.

2.2.1 NVMe-IP for Gen4

NVMe-IP implements NVMe protocol of the host side to access one NVMe SSD directly without PCIe switch connection. NVMe-IP supports six commands, i.e., Write, Read, Identify, Shutdown, SMART, and Flush. NVMe-IP can connect with the Integrated Block for PCI Express (PCIe hard IP) directly. More details of the NVMe-IP are described in the datasheet.
https://dgway.com/products/IP/NVMe-IP/dg_nvme_ip_data_sheet_en.pdf

2.2.2 Integrated Block for PCI Express

This block is the hard IP which is available in some Xilinx FPGAs. The maximum number of SSDs connecting to one FPGA device is limited by the numbers of PCIe hard IP. One NVMe-IP connects to one PCIe hard IP for controlling one NVMe SSD. More details of PCIe hard IP are described in following document.

PG213: UltraScale+ Devices Integrated Block for PCI Express
https://www.xilinx.com/products/intellectual-property/pcie4-ultrascale-plus.html#documentation

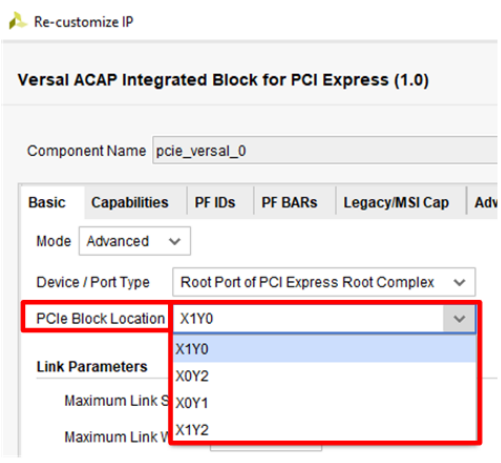
PG343: Versal ACAP Integrated Block for PCI Express
https://www.xilinx.com/products/intellectual-property/pcie-versal.html#documentation

The PCIe hard IP is created by using IP wizard. It is recommended for user to select “PCIe Block Location” which is closed to the transceiver pin that connects to the SSD. To connect two SSDs, two hard IPs with different location assignment are generated in the reference design. Please see more details about the location of PCIe hard IP and transceiver from following document.

UG575: UltraScale and UltraScale+ FPGAs Packaging and Pinouts
https://www.xilinx.com/support/documentation/user_guides/ug575-ultrascale-pkg-pinout.pdf

AM013: Versal ACAP Packaging and Pinouts
https://www.xilinx.com/support/documentation/architecture-manuals/am013-versal-pkg-pinout.pdf

The example of PCIe hard IP location on XCVC1902-VSVA2197 is shown in Figure 2-8.



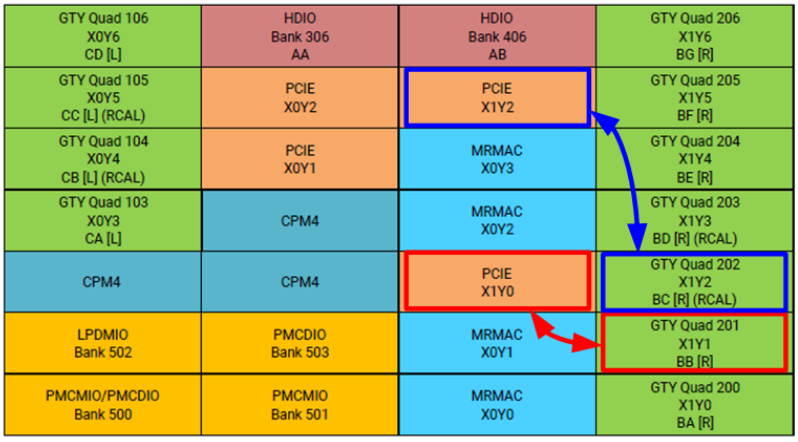


Figure 2-8 PCIe Hard IP Pin location

2.2.3 Dual port RAM

Two dual port RAMs, CtmRAM and IdenRAM, store data from Identify command and SMART command, respectively. IdenRAM is simple dual-port RAM which has one read port and one write port. The data size of Identify command is 8 Kbytes, so IdenRAM size is 8 Kbytes. NVMe-IP and LAxi2Reg have the different data bus size, 256 bits on NVMe-IP but 32 bits on LAxi2Reg. Therefore, IdenRAM is asymmetric RAM that has the different bus size on Write interface and Read interface. Also, NVMe-IP has double word enable to write only 32-bit data in some cases. The RAM setting on Xilinx IP tool supports the write byte enable. The small logic to convert double word enable to be write byte enable is designed as shown in Figure 2-9.

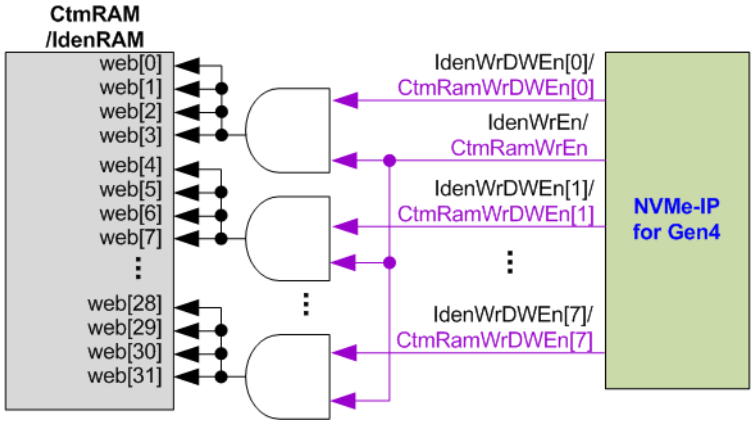


Figure 2-9 Byte write enable conversion logic

Bit[0] of WrDWEn with WrEn signal are the inputs to AND logic. The output of AND logic is fed to bit[3:0] of IdenRAM byte write enable. Bit[1], [2], …, [7] of WrDWEn are applied to be bit[7:4], [11:8], …, [31:28] of IdenRAM write byte enable, respectively.

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

2.2.4 FIFO

Two FIFOs are applied to interface one NVMe-IP with RAID0x2, TxFF and RxFF. Both FIFOs are asymmetric FIFO and has 32 Kbyte size. TxFF converts 512-bit data (RAID0x2 data bus size) to 256-bit data (NVMe-IP for Gen4 data bus size) while RxFF converts 256-bit data to 512-bit data.

2.2.5 RAID0x2

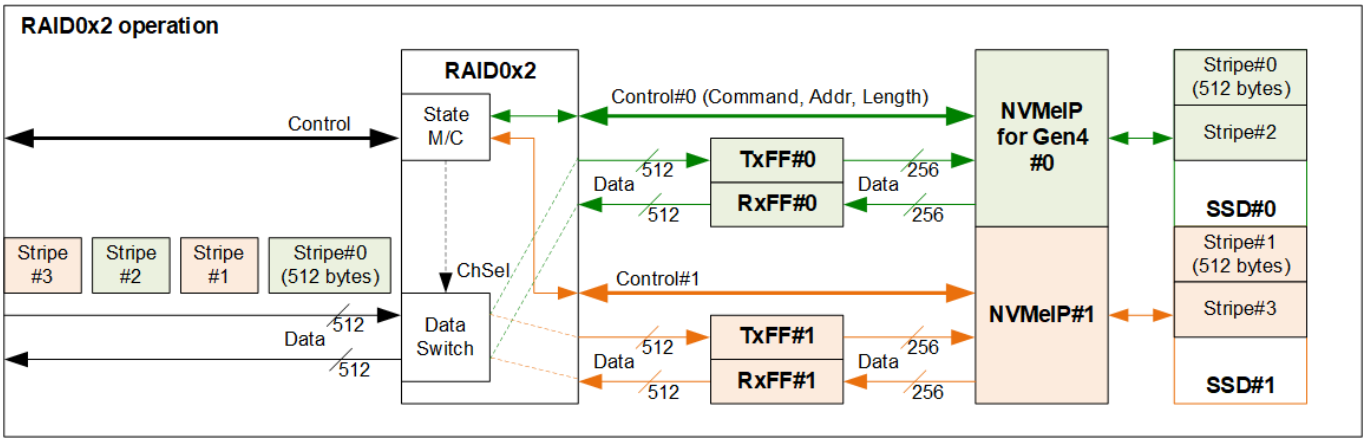


Figure 2-10 RAID0x2 operation

RAID0x2 consists of state machine and data switch for handling control interface and data interface. After receiving new command request from LAxi2Reg, state machine calculates the address and transfer length of each NVMe-IP by decoding user parameter inputs (set by LAxi2Reg). Next, state machine generates command request with the valid address and length to NVMe-IPs. Typically, the length to each NVMe-IP is about (the length from user / 2).

Figure 2-10 shows the details of the data flow for RAID0x2 operation. The stripe size is equal to 512 bytes. The data input from user (TestGen) is split to 512-byte unit. After that, RAID0x2 calculates the first active NVMe-IP and then transfers the first 512-byte data to it. Figure 2-10 shows the example of Write command when the first active NVMe-IP is Ch#0, so Stripe#0 is stored to SSD#0 via NVMe-IP#0 and TxFF#0. Next, the active channel is toggled to Ch#1. The next stripe (Stripe#1) is stored to SSD#1 via NVMe-IP#1 and TxFF#1. The active channel is toggled until the last data is transferred.

The first active channel is determined by using LSB bit of the start address which is the address of 512-byte data block. When LSB of the start address is equal to '0', the first active channel is Ch#0. Thus, the first active channel is Ch#1 for LSB='1'.

There are many pipeline registers inside data switch logic. Therefore, there is overhead time for switching the active channel before transferring 512-byte data. In the reference design, the overhead time is about 1 clock cycle for each 512-byte transferring (512 bytes uses 8 clock cycles of 512-bit data). As a result, the overhead time is about 12.5% (1 cycle/8 cycles). To compensate the overhead time, clock frequency of RAID0x2 must be set to 12.5% higher than NVMe based clock. NVMe based clock for PCIe Gen4 is equal to 250 MHz. The clock frequency after compensating overhead time is at least 281.25 MHz (112.5% of 250 MHz). In the reference design, 285 MHz is applied, so Write/Read performance of RAID0 operation is almost equal to two times of one NVMe SSD performance.

The user interface of RAID0x2 is shown in Table 2-1. Control and data interface are designed to compatible to dgIF typeS format. Please see more details of dgIF typeS format from NVMe-IP datasheet.

Table 2-1 Signal description of RAID0x2 (only User interface)

Signal	Dir	Description
Control I/F of dgIF typeS		
RstB	In	Synchronous reset signal. Active low. De-assert to '1' when Clk signal is stable
Clk	In	System clock for running RAID0x2 and NVMe-IP. It is recommended to use the frequency more than 281.25 MHz for PCIe Gen4 to achieve the best performance. The minimum requirement of Clk signal is the same as NVMe-IP (more than or equal to 250 MHz).
UserCmd[1:0]	In	User Command (000b: Identify, 001b: Shutdown, 010b: Write, 011b: Read, 100b: SMART, 110b: Flush, 101b/111b: Reserved)
UserAddr[47:0]	In	Start address to write or read RAID0 in 512-byte unit. It is recommended to set UserAddr[3:0]=0000b to align 8 Kbyte which is page size for two SSDs (one SSD page size is 4 Kbyte). Otherwise, write and read performance of some SSD models are reduced from 4Kbyte unaligned address.
UserLen[47:0]	In	Total transfer size to write/read SSD in 512 byte unit. Valid from 1 to (LBASize-UserAddr).
UserReq	In	Assert to '1' to send the new command request and de-assert to '0' after RAID0 starts the operation by asserting UserBusy to '1'. This signal can be asserted to '1' when RAID0 is Idle (UserBusy='0'). Command parameter (UserCmd, UserAddr, UserLen, and CtmSubmDW0-DW15) must be valid and stable during UserReq='1'. UserAddr and UserLen are inputs for Write/Read command while CtmSubmDW0-DW15 are inputs for SMART/Flush command.
UserBusy	Out	Asserted to '1' when RAID0 is busy. New request must not be sent (UserReq to '1') when RAID0 is busy (UserBusy='1').
LBASize[47:0]	Out	Total capacity of two SSDs in 512-byte unit. This value is valid after finishing Identify command. It is recommended to run Identify command as the first command. Default value is 0. *Note: LBASize of RAID0 is calculated from the SSD#0 capacity x 2.*
LBAMode	Out	LBA unit size of RAID0. This signal is valid after finishing Identify command. Default value is 0. '0': LBA size = 512 byte, '1': LBA size = 4 Kbyte. RAID0 uses 512-byte stripe size, so 4 Kbyte LBA size is not supported.
UserError	Out	Error flag. Assert to '1' when some bits of UserErrorType are not equal to 0. The flag can be cleared to '0' by asserting RstB to '0'.
UserErrorType[0-1][31:0]	Out	Error status, directly mapped from UserErrorType in each NVMe-IP. [0]-IP#0, [1]-IP#1.
Data I/F of dgIF typeS		
UserFifoWrCnt[15:0]	In	Write data counter of Receive FIFO. Used to check full status of FIFO. When full status is detected, the returned data transmission from Read command may be paused. If the size of FIFO data count is less than 16-bit, please fill '1' to remained upper bit.
UserFifoWrEn	Out	Asserted to '1' to write data to Receive FIFO while running Read command.
UserFifoWrData[511:0]	Out	Write data bus of Receive FIFO. Valid when UserFifoWrEn='1'.
UserFifoRdCnt[15:0]	In	Read data counter of Transmit FIFO. Used to check data size stored in FIFO. The transmitted data packet for Write command may be paused when the counter shows empty status. If the size of FIFO data count is less than 16-bit, please fill '0' to remained upper bit.
UserFifoEmpty	In	The signal is unused.
UserFifoRdEn	Out	Asserted to '1' to read data from Transmit FIFO while running Write command.
UserFifoRdData[511:0]	In	Read data returned from Transmit FIFO. Valid in the next clock after UserFifoRdEn is asserted to '1'.

Timing diagram of RAID0x2 module when running Write command is shown as follows.

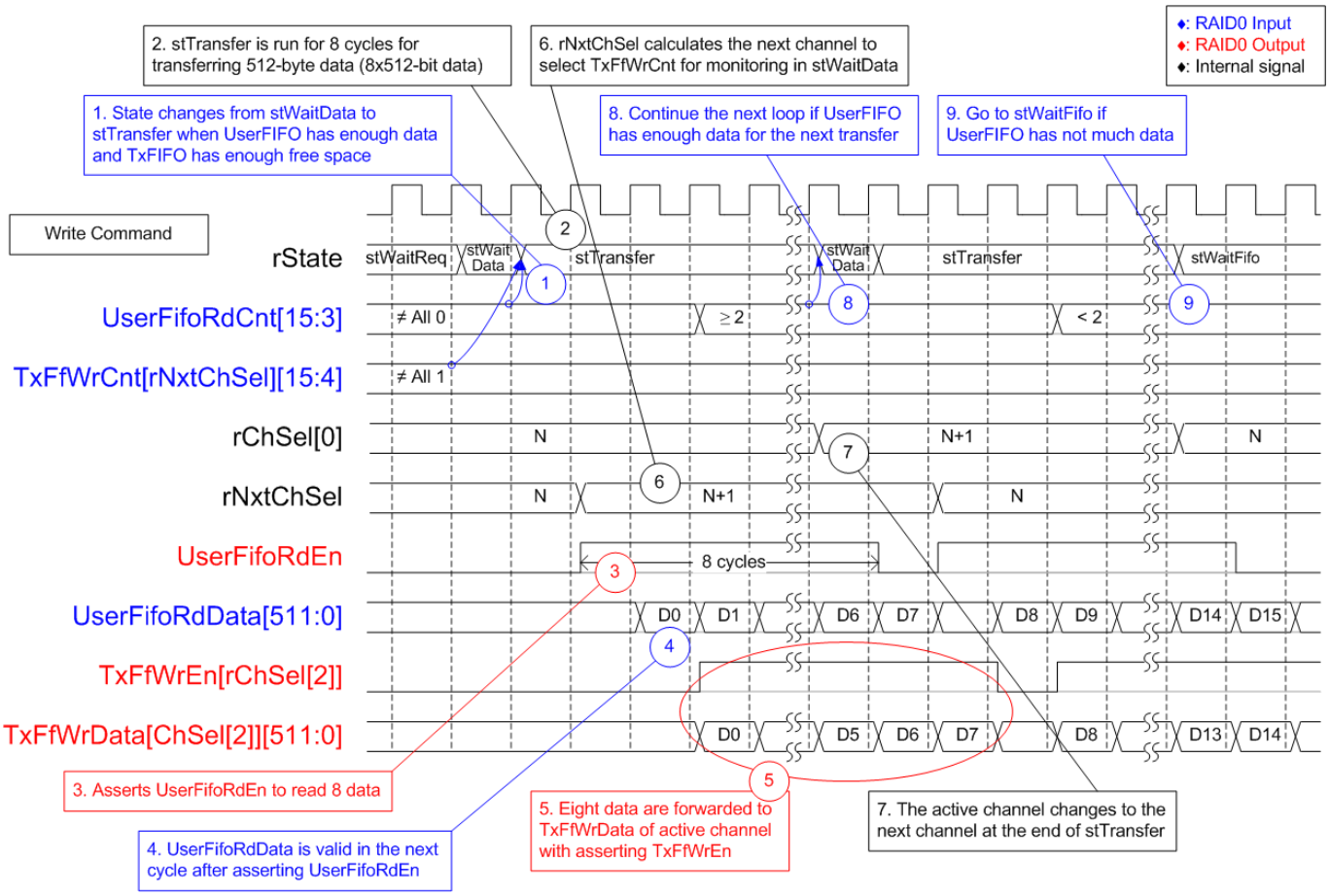


Figure 2-11 RAID0x2 timing diagram in Write command

When user sends Write command to RAID0, data is forwarded from UserFIFO (U2IPFIFO) to TxFIFO#0 (TxFF#0) or TxFIFO#1 (TxFF#1). One TxFIFO is active to transfer 512-byte data at a time. After that, the active channel of NVMe is switched to the next channel for transferring data, following RAID0 behavior.

1) stWaitData is the core state of RAID0x2 module. First, it checks the remained transfer size. The operation is finished when the remained transfer size is equal to 0. Otherwise, UserFifoRdCnt and TxFfWrCnt are monitored to confirm that at least 512-byte data is stored in U2IPFIFO and TxFIFO has at least 1024-byte free space. If FIFOs are ready, the write operation is started.

   Note:

   i) TxFfWrCnt of two channels are fed to multiplexer to select the active channel. Therefore, it has one clock latency, comparing to UserFifoRdCnt.

   ii) TxFfWrCnt is controlled by rNxtChSel which shows the next active channel for running in "stTransfer". After starting the first transfer loop, rNxtChSel is increased for scanning free space size of the next active channel FIFO.

2) State machines changes to stTransfer to start forwarding the write data from user logic to TxFIFO. This state is run for 8 clock cycles.
3) UserFifoRdEn is asserted when the state is stTransfer, so it is asserted to '1' for 8 clock cycles to read 512-byte data from UserFIFO.
4) Read data (UserFifoRdData) is valid in the next cycle after asserting Read enable (UserFifoRdEn).
5) The data is forwarded to TxFIFO of the active channel, selected by rChSel[2] which is two-clock latency signal of rChSel[0].
   Note: rChSel[0] shows the active channel for transferring data in stTransfer state.
6) When running in stTransfer state, rNxtChSel calculates the next active channel from rChSel[0]. rNxtChSel is applied to select the active channel for reading TxFfWrCnt of the next transfer in stWaitData.
7) The active channel for transferring data (rChSel[0]) is increased after finishing 512-byte data transferring in stTransfer state.
8) To reduce the overhead time for running the next transfer loop, UserFifoRdCnt is monitored in stTransfer state. If read counter shows at least 2x512-byte data is stored in FIFO, the new transfer loop will be started in the next cycle by changing to stWaitData and then returning to step 1. Otherwise, the next state is stWaitFifo, described in step 9.
9) stWaitFIFO is designed to wait until the current data transferring is done and UserFifoRdCnt is completely updated for monitoring. After waiting for three clock cycles, the state changes to stWaitData to transfer the next 512-byte data or finish the operation.

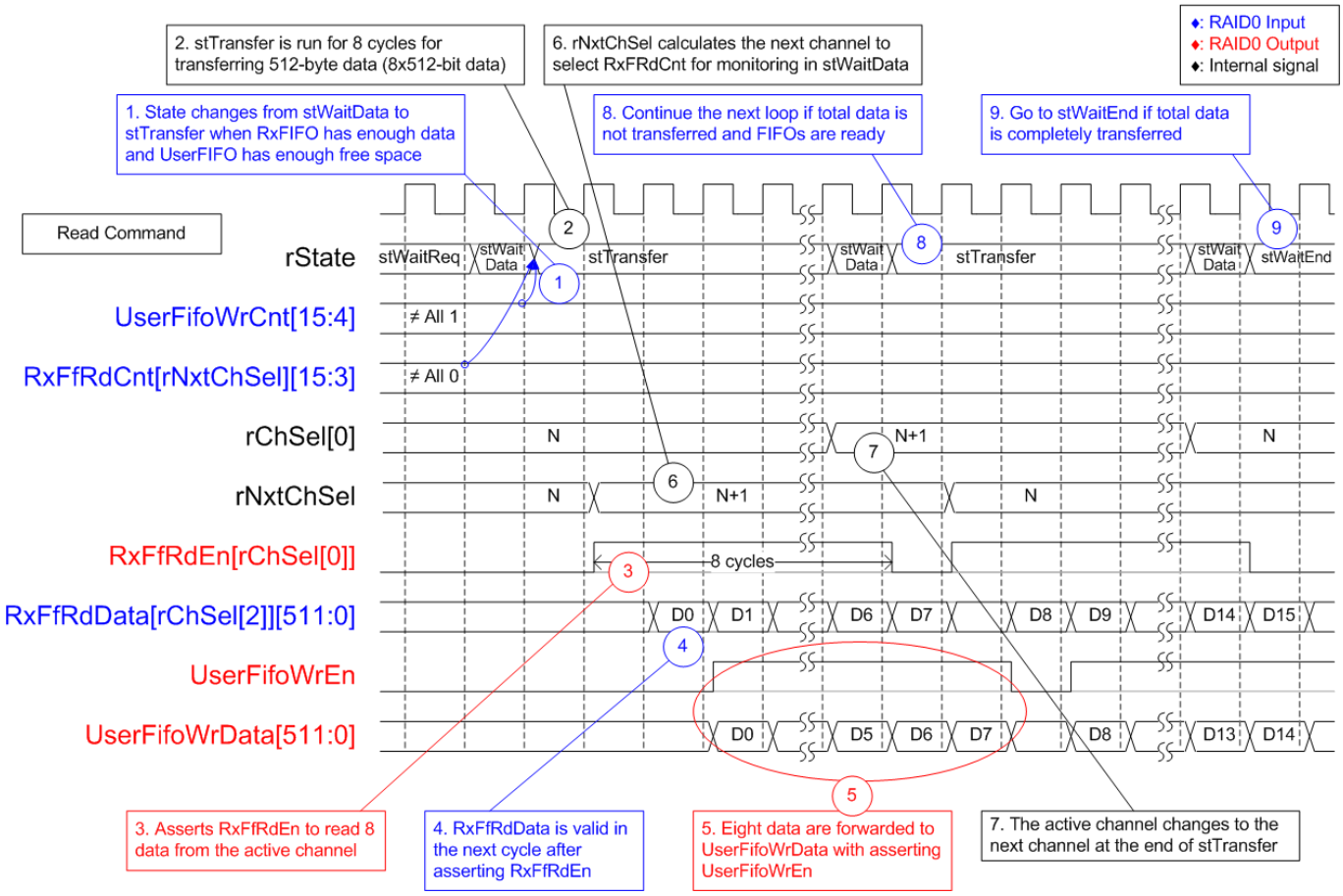


Figure 2-12 Raid0x2 timing diagram in Read command

When user sends Read command to RAID0, data is forwarded from RxFIFO#0 (RxFF#0) or RxFIFO#1 (RxFF#1) to IP2UFIFO. Similar to Write command, one RxFIFO is active to transfer 512-byte at a time. The active channel is switched to the next channel after finishing 512-byte data transferring, following RAID0 behavior.

1) stWaitData is the state to check the remained transfer length and FIFO status. In read command, RxFfRdCnt and UserFifoWrCnt are monitored to confirm that at least 512-byte data is stored in RxFIFO and IP2UFIFO has at least 1024-byte free space. The read operation is started when both FIFO status are ready and there is remained transfer length.

   Note:

   i) RxFfRdCnt of two channels are fed to multiplexer to select the active channel. Therefore, it has one clock latency, comparing to UserFifoWrCnt.

   ii) RxFfRdCnt is controlled by rNxtChSel which is the next active channel for running in "stTransfer".

2) State machines changes to stTransfer to start forwarding 512-byte read data from RxFIFO to user logic by staying in this state for 8 clock cycles.
3) RxFfRdEn of the active channel, selected by rChSel[0], is asserted to '1' when the state is stTransfer. Thus, it is asserted for 8 clock cycles.
4) Read data (RxFfRdData) is valid in the next cycle after asserting Read enable (RxFfRdEn).
5) The data of the active channel, selected by rChSel[2], is forwarded to UserFIFO. *Note: rChSel[2] is ChSel[0] signal with two-clock latency.*
6) When running in stTransfer state, rNxtChSel calculates the next active channel from rChSel[0]. rNxtChSel is applied to select the active channel for reading RxFfRdCnt which is monitored in stWaitData.
7) The active channel for transferring data (rChSel[0]) is increased after finishing 512-byte data transferring in stTransfer state.
8) If there is remained transfer length, the state changes to stTransfer to start new data transferring, similar to step 2. Otherwise, the next state is stWaitEnd, described in step 9.
9) If all data are completely transferred, the state changes to stWaitEnd to wait until all NVMe-IPs finish the operation by de-asserting Busy signal to '0'.

2.3 CPU and Peripherals

32-bit AXI4-Lite bus is applied to be the bus interface for CPU accessing the peripherals such as Timer and UART. The test system of NVMe-IP is connected with CPU as a peripheral on 32-bit AXI4-Lite bus for CPU controlling and monitoring. CPU assigns the different base address and the address range to each peripheral for accessing one peripheral at a time.

In the reference design, the CPU system is built with one additional peripheral to access the test logic. Therefore, the hardware logic must be designed to support AXI4-Lite bus standard for CPU writing and reading. LAxi2Reg module is designed to connect with the CPU system as shown in Figure 2-13.

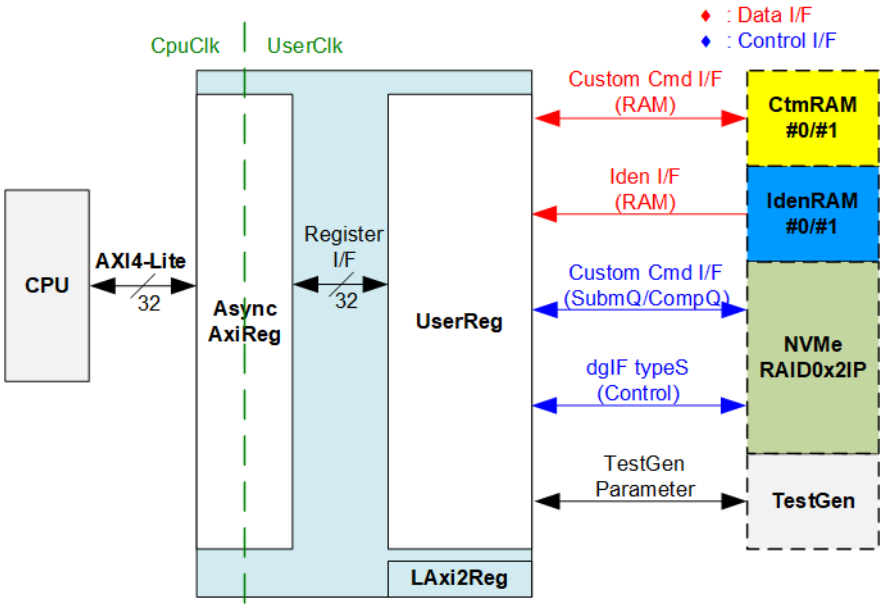


Figure 2-13 CPU and peripherals hardware

LAxi2Reg consists of AsyncAxiReg and UserReg. AsyncAxiReg is designed to convert the AXI4-Lite signals to be the simple register interface which has 32-bit data bus size, similar to AXI4-Lite data bus size. Besides, AsyncAxiReg includes asynchronous logic to support clock domain 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, NVMeRAID0x2IP, and TestGen. More details of AsyncAxiReg and UserReg are described as follows.

2.3.1 AsyncAxiReg

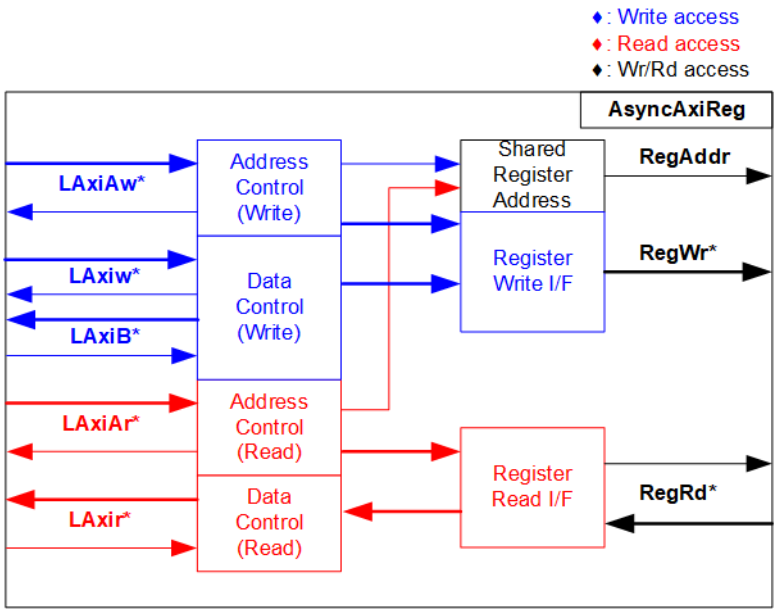


Figure 2-14 AsyncAxiReg Interface

The signal on AXI4-Lite bus interface can be split into five groups, i.e., LAxiAw* (Write address channel), LAxiw* (Write data channel), LAxiB* (Write response channel), LAxiAr* (Read address channel), and LAxir* (Read data channel). More details to build custom logic for AXI4-Lite bus is described in following document.
https://forums.xilinx.com/xlnx/attachments/xlnx/NewUser/34911/1/designing_a_custom_axi_slave_rev1.pdf

According to AXI4-Lite standard, the write channel and the read channel are operated independently. Also, the control and data interface of each channel are run separately. So, the logic inside AsyncAxiReg to interface with AXI4-Lite bus is split into four groups, i.e., Write control logic, Write data logic, Read control logic, and Read data logic as shown in the left side of Figure 2-14. Write control I/F and Write data I/F of AXI4-Lite bus are latched and transferred to be Write register interface with clock domain crossing registers. Similarly, Read control I/F of AXI4-Lite bus are latched and transferred to be Read register interface. While the returned data from Register Read I/F is transferred to AXI4-Lite bus by using clock domain crossing registers. In register interface, RegAddr is shared signal for write and read access. Therefore, it loads the address from LAxiAw for write access or LAxiAr for read access.

The simple register interface is compatible with single-port RAM interface for write transaction. The read transaction of the register interface is slightly modified from RAM interface by adding RdReq and RdValid signals for controlling read latency time. The address of register interface is shared for write and read transaction. Therefore, user cannot write and read the register at the same time. The timing diagram of the register interface is shown in Figure 2-15.

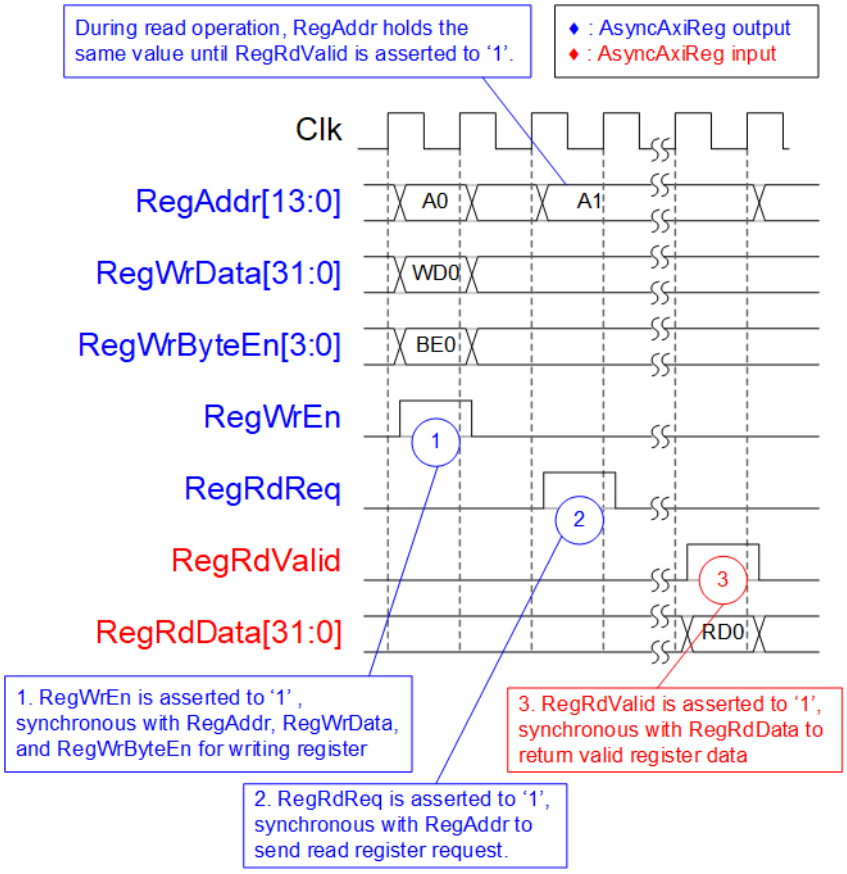


Figure 2-15 Register interface timing diagram

1) To write register, the timing diagram is similar to single-port RAM interface. RegWrEn is asserted to '1' with the valid signal of RegAddr (Register address in 32-bit unit), RegWrData (write data of the register), and RegWrByteEn (the write byte enable). Byte enable has four bits to be 4-byte data enable. Bit[0], [1], [2], and [3] are equal to '1' when RegWrData[7:0], [15:8], [23:16], and [31:24] are valid, respectively.
2) To read register, AsyncAxiReg asserts RegRdReq to '1' with the valid value of RegAddr. 32-bit data must be returned after receiving the read request. The slave detects RegRdReq asserted to start the read transaction. In read operation, the address value (RegAddr) does not change until RegRdValid is asserted to '1'. Therefore, the address can be used for selecting the returned data by using multiple levels of multiplexer.
3) The read data is returned on RegRdData bus by the slave with asserting RegRdValid to '1'. After that, AsyncAxiReg forwards the read value to LAxir* interface.

2.3.2 UserReg

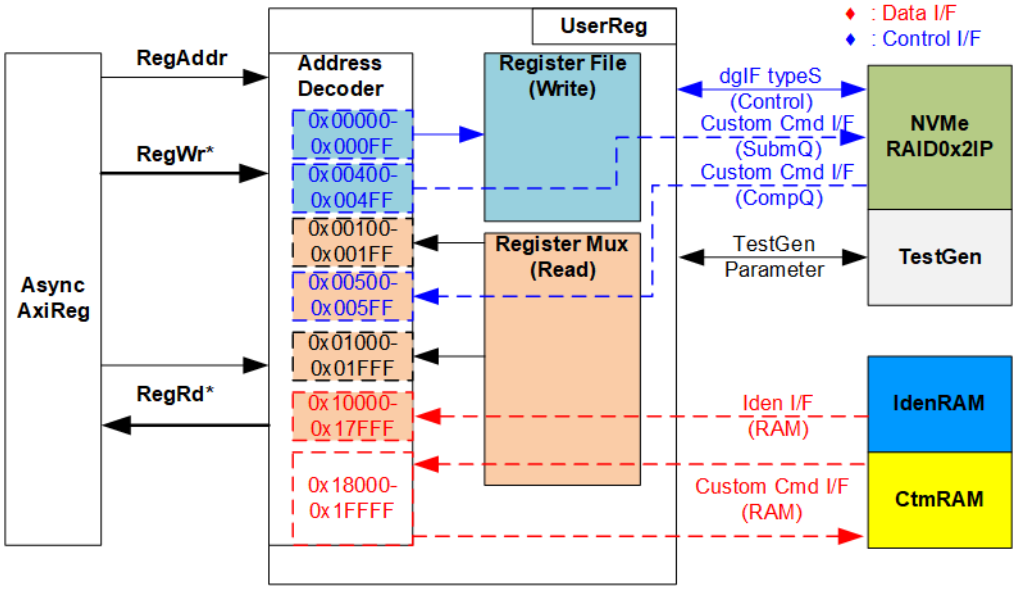


Figure 2-16 UserReg Interface

The address range to map to UserReg is split into six areas, as shown in Figure 2-16.

1) 0x00000 – 0x000FF : mapped to set the command with the parameters of NVMeRAID0x2IP and TestGen. This area is write-access only.
2) 0x00400 – 0x004FF : mapped to set the parameters for custom command interface of NVMe-IP. This area is write-access only.
3) 0x00100 – 0x001FF : mapped to read the status signals of NVMeRAID0x2IP. This area is read-access only.
4) 0x00500 – 0x005FF : mapped to read the status of custom command interface (NVMe-IP). This area is read-access only.
5) 0x01000 – 0x01FFF : mapped to read the status of TestGen. This area is read-access only.
6) 0x10000 – 0x17FFF : mapped to read data from IdenRAM. This area is read-access only.
7) 0x18000 – 0x1FFFF : mapped to write or read data with custom command RAM interface. 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. Therefore, write byte enable (RegWrByteEn) is not applied in the test system and the CPU uses 32-bit pointer to set the hardware register.

To read register, three-step multiplexer selects the data to return to CPU by using the address. The lower bit of RegAddr is fed to the submodule to select the active data from each submodule. While the upper bit is applied in UserReg to select the returned data from each submodule. Totally, the latency time of read data is equal to three clock cycles. Therefore, RegRdValid is created by RegRdReq with asserting three D Flip-flops. More details of the address mapping within UserReg module are shown in Table 2-2.

Table 2-2 Register Map

Address Wr/Rd	Register Name (Label in “nvmeraid0g4test.c”)	Description
0x00000 – 0x000FF: Status signals of NVMeRAID0x2IP and TestGen (Write access only)		
BA+0x00000	User Address (Low) Reg (USRADRL_INTREG)	[31:0]: Input to be bit[31:0] of start address in 512-byte unit (UserAddr[31:0] of RAID0x2)
BA+0x00004	User Address (High) Reg (USRADRH_INTREG)	[15:0]: Input to be bit[47:32] of start address in 512-byte unit (UserAddr[47:32] of RAID0x2)
BA+0x00008	User Length (Low) Reg (USRLENL_INTREG)	[31:0]: Input to be bit[31:0] of transfer length in 512-byte unit (UserLen[31:0] of RAID0x2)
BA+0x0000C	User Length (High) Reg (USRLENH_INTREG)	[15:0]: Input to be bit[47:32] of transfer length in 512-byte unit (UserLen[47:32] of RAID0x2)
BA+0x00010	User Command Reg (USRCMD_INTREG)	[2:0]: Input to be user command (UserCmd of RAID0x2) (000b: Identify, 001b: Shutdown, 010b: Write RAID, 011b: Read RAID, 100b: SMART, 110b: Flush, 101b/111b: Reserved). When this register is written, the command request is sent to RAID0x2 to start the operation.
BA+0x00014	Test Pattern Reg (PATTSEL_INTREG)	[2:0]: Select test pattern 000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1, 100b-LFSR
BA+0x00020	NVMe Timeout Reg (NVMTIMEOUT_INTREG)	[31:0]: Input to be timeout value of all NVMe-IPs (TimeOutSet[31:0] of NVMe-IP)
0x00100 – 0x001FF: Status signals of NVMeRAID0x2IP (Read access only)		
BA+0x00100	User Status Reg (USRSTS_INTREG)	[0]: UserBusy of RAID0x2 (‘0’: Idle, ‘1’: Busy) [1]: UserError of RAID0x2 (‘0’: Normal, ‘1’: Error) [2]: Data verification fail in TestGen (‘0’: Normal, ‘1’: Error)
BA+0x00104	Total device size (Low) Reg (LBASIZEL_INTREG)	[31:0]: Mapped to LBASize[31:0] of RAID0x2 to show RAID0 capacity
BA+0x00108	Total device size (High) Reg (LBASIZEH_INTREG)	[15:0]: Mapped to LBASize[47:32] of RAID0x2 to show RAID0 capacity [31]: Mapped to LBAMode of RAID0
(BA+0x00110)- (BA+0x00117)	User Error Type CH#0-#1 Reg (USRERRTYPE0-1_INTREG)	0x0110: NVMe-IP#0, 0x0114: NVMe-IP#1, [31:0]: Mapped to UserErrorType of NVMe-IP
(BA+0x00120)- (BA+0x00127)	PCIe Status CH#0-#1 Reg (PCIESTS0-1_INTREG)	0x0120: NVMe-IP#0, 0x0124: NVMe-IP#1 [0]: PCIe linkup status from PCIe hard IP (‘0’: No linkup, ’1’: linkup) [3:2]: Two lower bits to show PCIe link speed. MSB is bit[16]. (000b: Not linkup, 001b: PCIe Gen1, 010b: PCIe Gen2, 011b: PCIe Gen3, 111b: PCIe Gen4) [7:4]: PCIe link width status from PCIe hard IP (0001b: 1-lane, 0010b: 2-lane, 0100b: 4-lane, 1000b: 8-lane) [13:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in PCIe hard IP datasheet [16]: The upper-bit to show PCIe link speed of PCIe hard IP. Two lower bits are bit[3:2].
(BA+0x00130)- (BA+0x00137)	Completion Status CH#0-#1 Reg (COMPSTS0-1_INTREG)	0x0130: NVMe-IP#0, 0x0134: NVMe-IP#1, [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+0x00140)- (BA+0x00147)	NVMe CAP CH#0-#1 Reg (NVMCAP0-1_INTREG)	0x0140: NVMe-IP#0, 0x0144: NVMe-IP#1, [31:0]: Mapped to NVMeCAPReg[31:0] of NVMe-IP
(BA+0x00150)- (BA+0x00157)	NVMe IP Test pin CH#0-#1 Reg (NVMTESTPIN0-1_INTREG)	0x0150: NVMe-IP#0, 0x0154: NVMe-IP#1, [31:0]: Mapped to TestPin[31:0] of NVMe-IP.

Address Wr/Rd	Register Name (Label in “nvmeraid0g4test.c”)	Description
0x01000 – 0x01FFF: Status signals of TestGen (Read access only)		
BA+0x01000	Data Failure Address(Low) Reg (RDFAILNOL_INTREG)	[31:0]: Bit[31:0] of the byte address of the 1st failure data in Read command
BA+0x01004	Data Failure Address(High) Reg (RDFAILNOH_INTREG)	[24:0]: Bit[56:32] of the byte address of the 1st failure data in Read command
BA+0x01008	Current test byte (Low) Reg (CURTESTSIZEL_INTREG)	[31:0]: Bit[31:0] of the current test data size in TestGen module
BA+0x0100C	Current test byte (High) Reg (CURTESTSIZEH_INTREG)	[24:0]: Bit[56:32] of the current test data size in TestGen module
(BA+0x01200)- (BA+0x0123F)	Expected value Word0-15 Reg (EXPPATW0-W15_INTREG)	512-bit of the expected data at the 1st failure data in Read command 0x1200: Bit[31:0], 0x1204: Bit[63:32], …, 0x123C: Bit[511:480]
(BA+0x01400)- (BA+0x0143F)	Read value Word0-7 Reg (RDPATW0-W15_INTREG)	512-bit of the read data at the 1st failure data in Read command 0x1400: Bit[31:0], 0x1404: Bit[63:32], …, 0x143C: Bit[511:480]
0x00400 – 0x00FFF: Custom command of NVMeRAID0x2IP		
(BA+0x00400)- (BA+0x0047F) Wr	Custom Submission Queue CH#0–#1 Reg (CTMSUBMQ0-1_STRUCT)	[31:0]: Submission queue entry of SMART and Flush command. Input to be CtmSubmDW0-DW15 of NVMe-IP#0-#1 successively. 0x400: DW0, 0x404: DW1, …, 0x43C: DW15 of NVMe-IP#0 0x440: DW0, 0x444: DW1, …, 0x47C: DW15 of NVMe-IP#1
(BA+0x00500)- (BA+0x0051F) Rd	Custom Completion Queue CH#0–#1 Reg (CTMCOMPQ0-1_STRUCT)	[31:0]: CtmCompDW0-DW3 output from NVMe-IP#0-#1 successively. 0x500: DW0, 0x504: DW1, …, 0x50C: DW3 of NVMe-IP#0 0x510: DW0, 0x514: DW1, …, 0x51C: DW3 of NVMe-IP#1
BA+0x00800 Rd	IP Version Reg (IPVERSION_INTREG)	[31:0]: IP version number (IPVersion[31:0] of NVMe-IP)
0x10000 – 0x1FFFF: Identify RAM and Custom RAM		
(BA+0x10000) – (BA+0x13FFF) Rd	Identify Controller Data CH#0-#1 (IDENCTRL0-1_CHARREG/ IDENNAME0-1_CHARREG)	0x10000-0x10FFF: 4Kbyte Identify controller data of NVMe-IP#0 0x11000-0x11FFF: 4Kbyte Identify namespace data of NVMe-IP#0 0x12000-0x12FFF: 4Kbyte Identify controller data of NVMe-IP#1 0x13000-0x13FFF: 4Kbyte Identify namespace data of NVMe-IP#1
(BA+0x18000) – (BA+0x1BFFF) Wr/Rd	Custom command Ram CH#0-#1 (CTMRAM0-1_CHARREG)	0x18000-0x19FFF: NVMe-IP#0, 0x1A000-0x1BFFF: NVMe-IP#1 8Kbyte CtmRAM interface of NVMe-IP#0-#1. Used to store 512-byte data output from SMART Command.

3 CPU Firmware

3.1 Test firmware (nvmeraid0g4test.c)

After system boot-up, CPU runs following steps to finish the initialization process.

1) CPU initializes its peripherals such as UART and Timer.
2) CPU waits until PCIe connection links up (PCIESTS0/1_INTREG[0]='1').
3) CPU waits until NVMeRAID0x2 completes initialization process (USRSTS_INTREG[0]='0'). If some errors are found, the process stops with displaying the error message.
4) CPU displays PCIe link status (the number of PCIe lanes and the PCIe speed) by reading PCIESTS0/1_INTREG[16:2].
5) CPU displays the main menu. There are six menus for running six commands with RAID0x2, i.e., Identify, Write, Read, SMART, Flush, and Shutdown.

More details of each command are described as follows.

3.1.1 Identify command

The step to operate Identify command is described as follows.

1) Set USRCMD_INTREG[2:0]=000b to send Identify command request to NVMeRAID0x2. After that, busy flag of NVMeRAID0x2 (USRSTS_INTREG[0]) changes from '0' to '1'.
2) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

   Bit[0] is de-asserted to '0' after finishing operating the command. Next, the data from Identify command of two NVMe-IPs are stored in IdenRAM.
   Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE0/1_INTREG[31:0]. Finally, the process is stopped.

3) After busy flag (USRSTS_INTREG[0]) is de-asserted to '0', CPU displays some information decoded from IdenRAM (IDENCTRL0/1_INTREG) such as SSD model name. Also, RAID0x2 capacity (LBASIZEL/H_INTREG) from NVMeRAID0x2 output is read and displayed on the console. Finally, CPU checks LBA Size of RAID0x2 (LBASIZEH_INTREG[31]). If LBA Size of RAID0x2 is 4Kbyte which is not supported by RAID0x2, the process is stopped and the error message is displayed.

3.1.2 Write/Read command

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

1) Receive start address, transfer length, and test pattern from Serial console. If some inputs are invalid, the operation is cancelled.
2) Get all inputs and then set to USRADRL/H_INTREG, USRLENL/H_INTREG, and PATTSEL_INTREG.
3) Set USRCMD_INTREG[2:0]=010b for Write command or 011b for Read command. After that, the new command request is sent to NVMeRAID0x2 for running Write command or Read command. Busy flag (USRSTS_INTREG[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_INTREG[2:0].

   Bit[0] is de-asserted to '0' when command is completed.
   Bit[1] is asserted to '1' when some errors are detected. After that, the error message is displayed on the console to show the error details, decoded from USRERRTYPE0/1_INTREG[31:0]. Finally, the process is stopped.
   Bit[2] is asserted to '1' when data verification is failed. The verification error message is displayed on the console to show the error details. However, CPU is still running until the operation is done or user presses any key(s) to cancel operation.

   While running the operation, current transfer size read from CURTESTSIZEL/H_INTREG is displayed every second.

5) After busy flag (USRSTS_INTREG[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 (CTMSUBMQ0/1_STRUCT) to be SMART command value.
2) Set USRCMD_INTREG[2:0]=100b to send SMART command request to NVMeRAID0x2. After that, busy flag (USRSTS_INTREG[0]) changes from '0' to '1'.
3) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

   Bit[0] is de-asserted to '0' when command is completed. After that, the data returned from SMART command is stored to CtmRAM.
   Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE0/1_INTREG[31:0]. Finally, the process is stopped.

4) After busy flag (USRSTS_INTREG[0]) is de-asserted to '0', CPU decodes SMART command information from CtmRAM (CTMRAM0/1_CHARREG), i.e., Remaining Life, Percentage Used, 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 (CTMSUBMQ0/1_STRUCT) to be Flush command value.
2) Set USRCMD_INTREG[2:0]=110b to send Flush command request of NVMeRAID0x2. After that, busy flag (USRSTS_INTREG[0]) changes from '0' to '1'.
3) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

   Bit[0] is de-asserted to '0' when command is completed. After that, CPU returns to the main menu.
   Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE0/1_INTREG[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_INTREG[2:0]=001b to send Shutdown command request of NVMeRAID0x2. After that, busy flag (USRSTS_INTREG[0]) changes from '0' to '1'.
2) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_INTREG[1:0].

   Bit[0] is de-asserted to '0' when command is completed. After that, the CPU goes to the next step.
   Bit[1] is asserted to '1' when some errors are detected. The error message is displayed on the console to show the error details, decoded from USRERRTYPE0/1_INTREG[31:0]. Finally, the process is stopped.

3) After busy flag (USRSTS_INTREG[0]) is de-asserted to '0', all SSDs and all NVMe-IPs change to inactive status. The CPU cannot receive new command from user. The user must power off the test system.

3.2 Function list in Test firmware

int exec_ctm(unsigned int user_cmd)	
Parameters	user_cmd: 4-SMART command, 6-Flush command
Return value	0: No error, -1: Some errors are found in NVMeRAID0x2
Description	Run SMART command or Flush command, following in topic 3.1.3 (SMART Command,) and 3.1.4 (Flush Command).

unsigned long long get_cursize(void)	
Parameters	None
Return value	Read value of CURTESTSIZEH/L_INTREG
Description	Read CURTESTSIZEH/L_INTREG 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).

int setctm_flush(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in NVMeRAID0x2
Description	Set Flush command to CTMSUBMQ0/1_STRUCT and call exec_ctm function to start Flush command.

int setctm_smart(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in NVMeRAID0x2
Description	Set SMART command to CTMSUBMQ0/1_STRUCT and call exec_ctm function to start SMART command. Finally, decode and display SMART information on the console

void show_error(void)	
Parameters	None
Return value	None
Description	Read USRERRTYPE0/1_INTREG, decode the error flag, and display error message following the error flag.

void show_pciestat(void)	
Parameters	None
Return value	None
Description	Read PCIESTS0/1_INTREG until the read value from two read times is stable. After that, display the read value on the console.

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

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

void show_smart_hex16byte(volatile unsigned char *char_ptr)	
Parameters	*char_ptr: Pointer of 16-byte SMART data
Return value	None
Description	Display 16-byte SMART data as hexadecimal unit.

void show_smart_int8byte(volatile unsigned char *char_ptr)	
Parameters	*char_ptr: Pointer of 8-byte SMART data
Return value	None
Description	When the input value is less than 4 billion (32-bit), display 8-byte SMART data as decimal unit. Otherwise, display overflow message.

void show_smart_size8byte(volatile unsigned char *char_ptr)	
Parameters	*char_ptr: Pointer of 8-byte SMART data
Return value	None
Description	Display 8-byte SMART data as GB or TB unit. When the input value is more than limit (500 PB), the overflow message is displayed instead.

void show_vererr(void)	
Parameters	None
Return value	None
Description	Read RDFAILNOL/H_INTREG (error byte address), EXPPATW0-W7_INTREG (expected value), and RDPATW0-W7_INTREG (read value) to display verification error details on the console.

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

int 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 RAID0 demo system by using two 1 TB WD BLACK SN850 SSDs is shown in Figure 4-1.

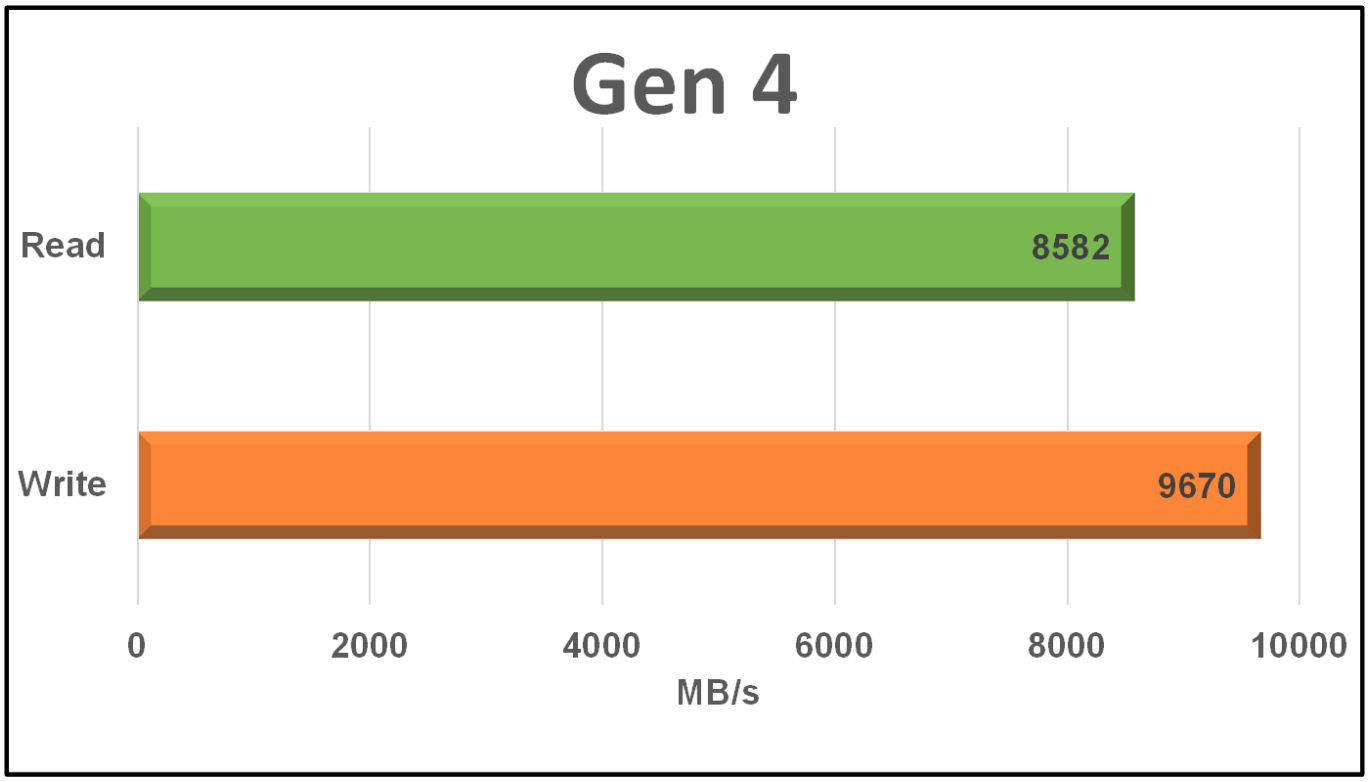


Figure 4-1 Performance of 2-ch RAID0 demo by using two 1TB WD BLACK SN850 SSDs

When running 2-ch RAID0 with 2 NVMe Gen4 SSDs, write performance is about 9,600 Mbyte/sec and read performance is about 8,500 Mbyte/sec. Test pattern is LSFR and transfer size is 64 Gbytes.

Note: By customizing the IP to increase the buffer size to be 1 Mbyte instead of 256 Kbyte, the read performance is increased. By using two 1 TB WD BLACK SN850 SSDs, the read performance is about 12,400 Mbyte/sec.

5 Revision History

Revision	Date	Description
1.0	16-Aug-22	Initial version release