2-ch RAID0 (muNVMe-IP) reference design manual

Rev1.0 8-Aug-23

2-ch RAID0 (muNVMe-IP) reference design manual

Rev1.0 8-Aug-23

1 Introduction

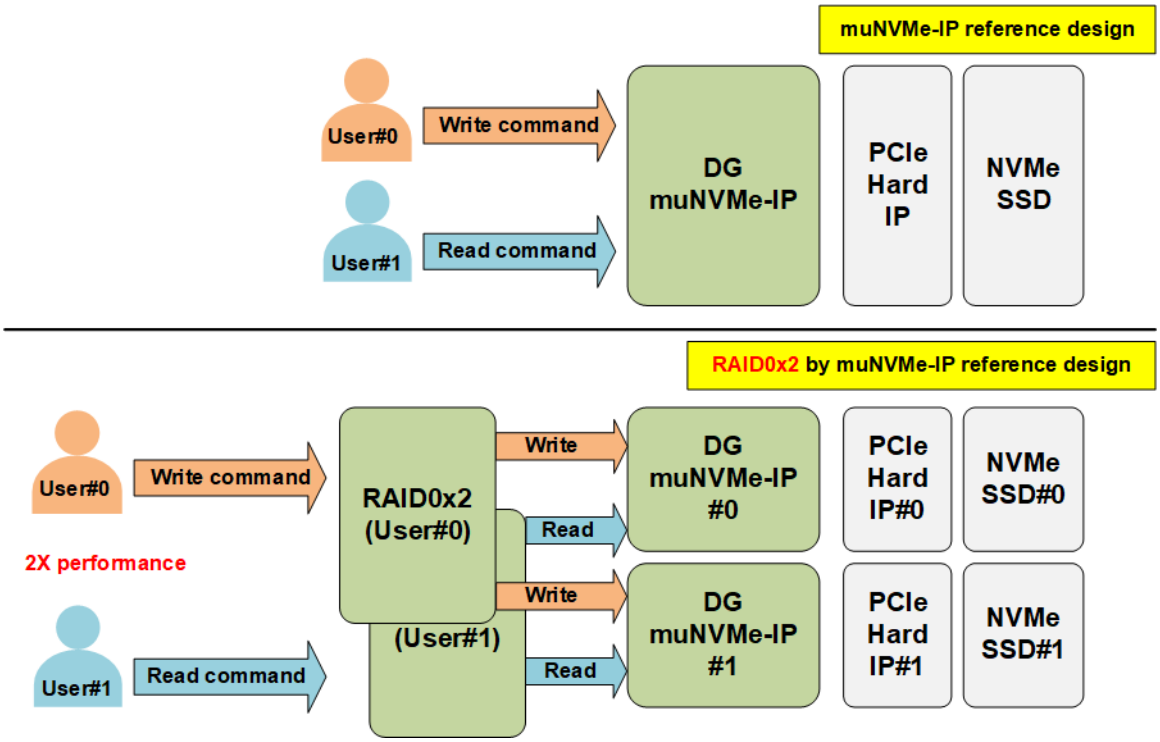

Figure 1-1 RAID0x2 by muNVMe-IP performance

muNVMe-IP has two user interfaces that can send the command to the same NVMe SSD individually. The main application of muNVMe-IP is to allow the user to write and read the data with the SSD at the same time. However, the write/read performance when two commands are run at the same SSD is reduced from the shared SSD. To increase the write/read performance, more NVMe SSDs and muNVMe-IPs must be applied.

As shown in Figure 1-1, RAID0x2 by muNVMe-IP is implemented to increase the write/read performance to two times, comparing to the muNVMe-IP reference design. RAID0x2 modules are designed to split one user command to two muNVMe-IPs. Generally, the transfer size that is sent to each muNMVe-IP from RAID0x2 is a half of the transfer size that is sent by the user. Thus, using two RAID0x2 modules for receiving the command request from two users can increase write/read performance to be two times of the muNVMe-IP reference design. User can modify the design to increase the performance to be four times by using four muNVMe-IPs and two RAID0x4 modules. Using more SSDs in RAID0 system does not only increase transfer performance, but also increase the system storage capacity. To achieve the best performance, it is recommended to use the same SSD model for matching the characteristic while running RAID0 operation.

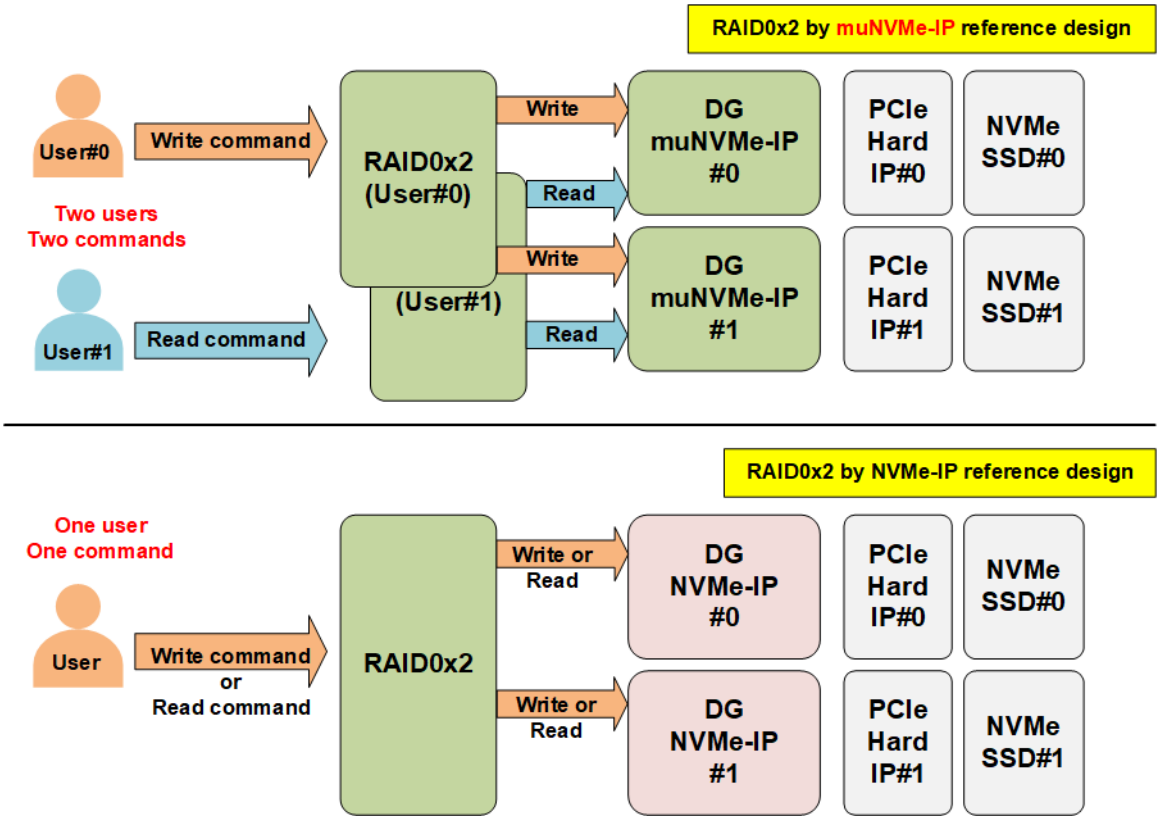

Figure 1-2 RAID0x2 by muNVMe-IP and NVMe-IP comparison

Comparing RAID0x2 by muNVMe-IP to RAID0x2 by NVMe-IP, the different point is the number of user interface. While muNVMe-IP supports two user interfaces for sending write and read command at the same time, NVMe-IP has one user interface to send write command or read command only. If two commands sending at the same time is not necessary, it is recommended to use NVMe-IP for smaller FPGA resource utilization.

Ref: RAID0x2 by NVM-IP reference design document
https://dgway.com/products/IP/NVMe-IP/dg_nvme_raid0x2_refdesign_en.pdf

2 Hardware overview

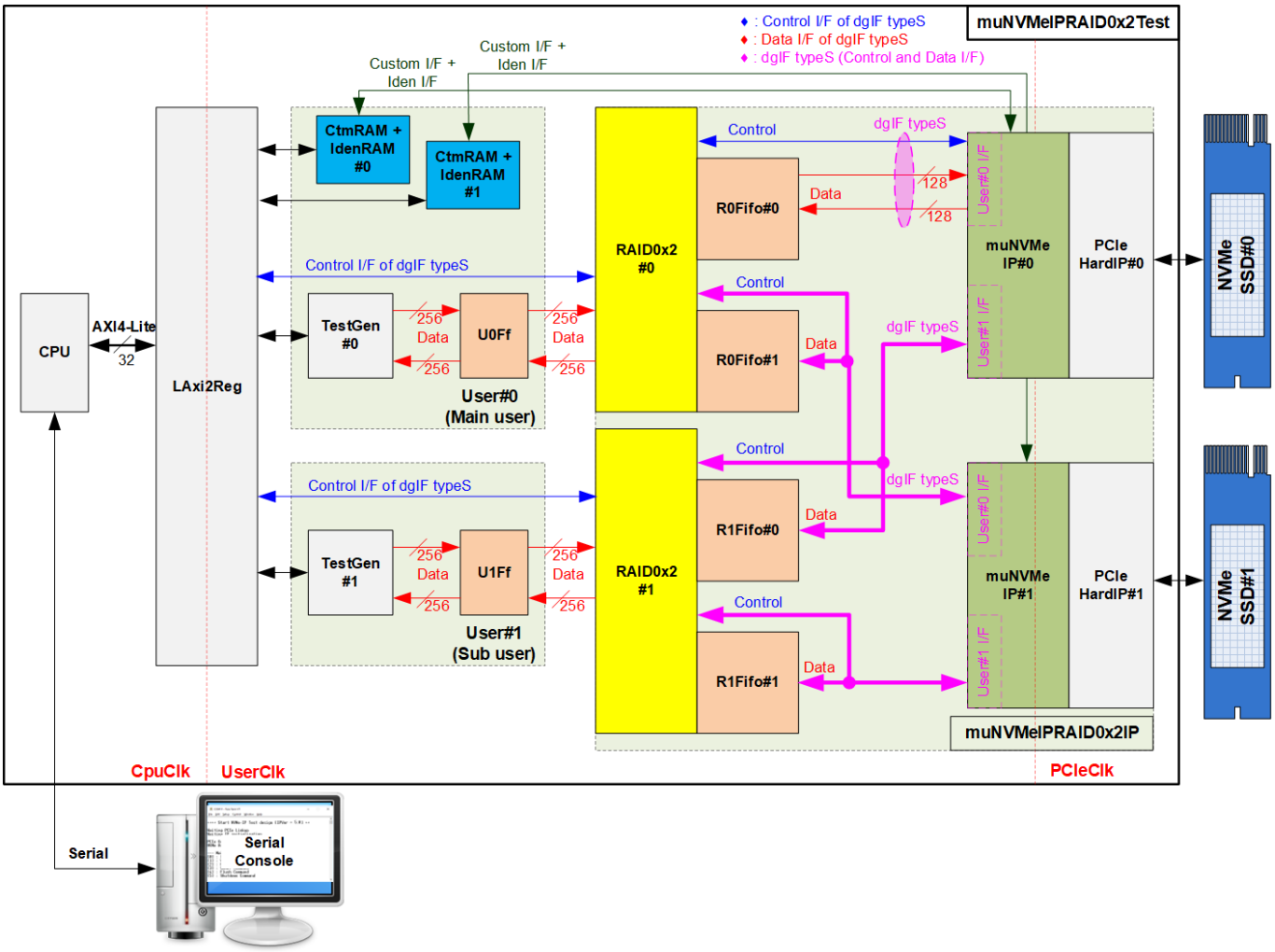

Figure 2-1 muNVMe-IP RAID0x2 demo hardware

Comparing to the muNVMe-IP reference design without RAID0x2, two user modules – User#0 (Main user) and User#1 (Sub user) are still applied, but data bus size is extended to 256 bits, not 128 bits. 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 muNVMeIPRAID0x2IP block are similar to muNVMe-IP by using dgIF typeS. To reduce FIFO resource utilization, the FIFO for storing data transferring between TestGen and RAID0x2 (U0Ff and U1Ff) and between RAID0x2 and muNVMe-IP (R0Fifo#0/#1 and R1Fifo#0/#1) are bi-directional FIFOs that have direction signal to control data transfer direction from left to right or vice versa. In Write command, the data is transferred from left-hand side to 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.

muNVMeIPRAID0x2IP is the top file of RAID0 block that consists of two RAID0x2 modules, FIFOs, two muNVMe-IPs, and two PCIe hard IPs. The top file is designed to have the same user interface as muNVMe-IP (dgIF typeS) for connecting with User#0 and User#1 logic. RAID0x2#0 connects to User#0 I/F of two muNVMe-IPs and uses R0Fifo#0/#1 for transferring data. While RAID0x2#1 connects to User#1 I/F of two muNVMe-IPs and uses R1Fifo#0/#1 for transferring data.

The CPU firmware implements all commands supported by muNVMe-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 muNVMe-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 muNVMe-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 U0Ff/U1Ff when the FIFOs have free space. Next, RAID0x2 block reads the data from U0Ff/U1Ff to store to R0Fifo/R1Fifo, so the data is always available for muNVMe-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. muNVMe-IP stores the read data from NVMe SSD to R0Fifo/R1Fifo. After that, RAID0x2 forwards the read data from R0Fifo/R1Fifo to U0Ff/U1Ff. Finally, TestGen reads the data from U0Ff/U1Ff 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 Gen3 speed, PCIe hard IP is run at 250 MHz which is the frequency of PCIeClk. According to muNVM-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 muNVMe-IP module. The overhead time is about 6.25%, so it is recommended to set UserClk to be higher or equal to 265.625 MHz (106.25% 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

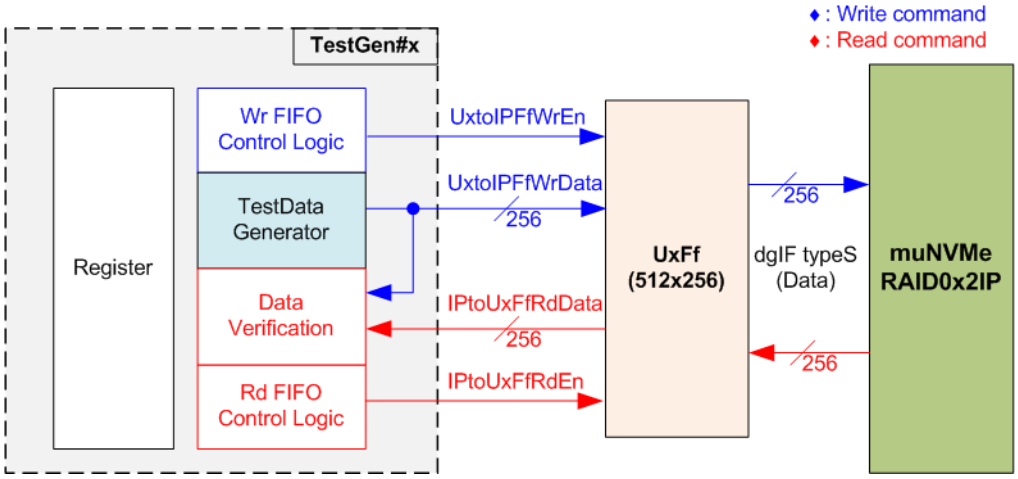

Figure 2-2 TestGen interface

TestGen module is the test logic to send test data to muNVMeRAID0x2 through U#Ff when operating Write command. On the other hand, the test data is applied to be the expected value to verify the read data from muNVMeRAID0x2 through U#Ff 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 muNVMeRAID0x2 by running at the same clock and using the same data bus size. Therefore, U#Ff are always ready for transferring data with muNVMeRAID0x2 in Write and read command. As a result, the test logic shows the best performance to write and read data with the SSD through muNVMeRAID0x2.

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.

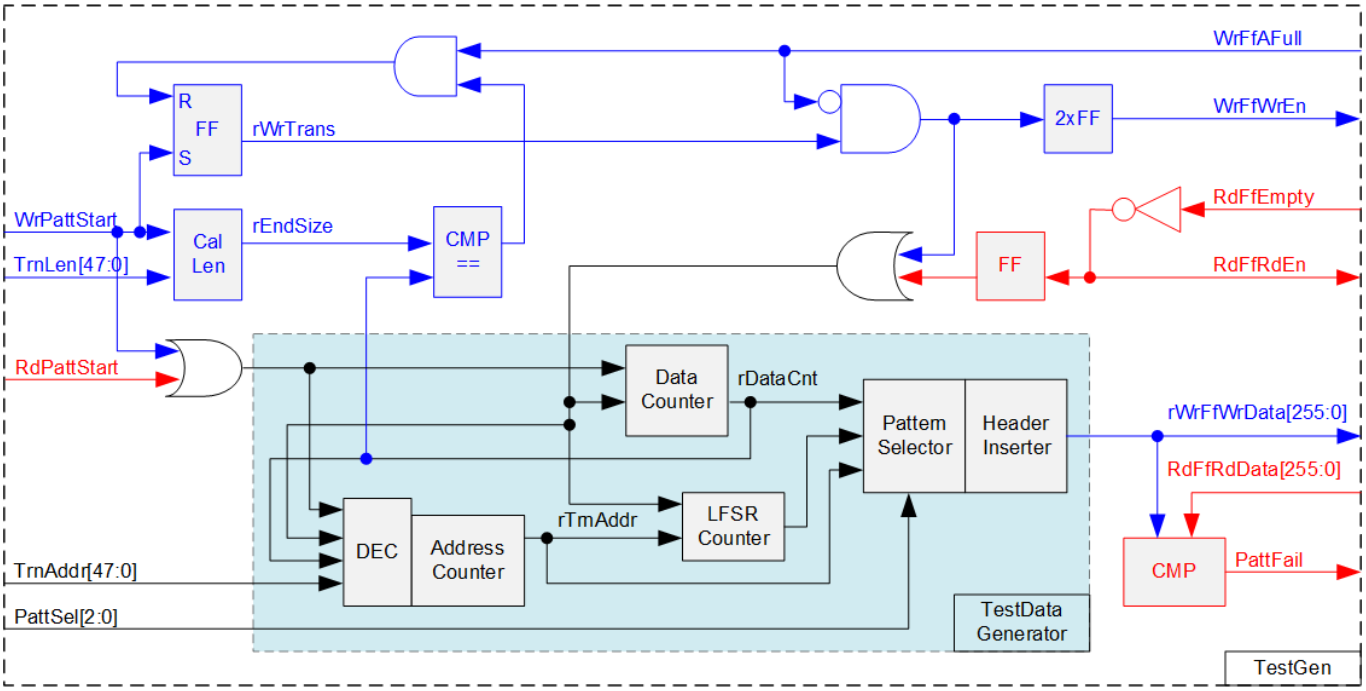

Figure 2-3 TestGen hardware

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.

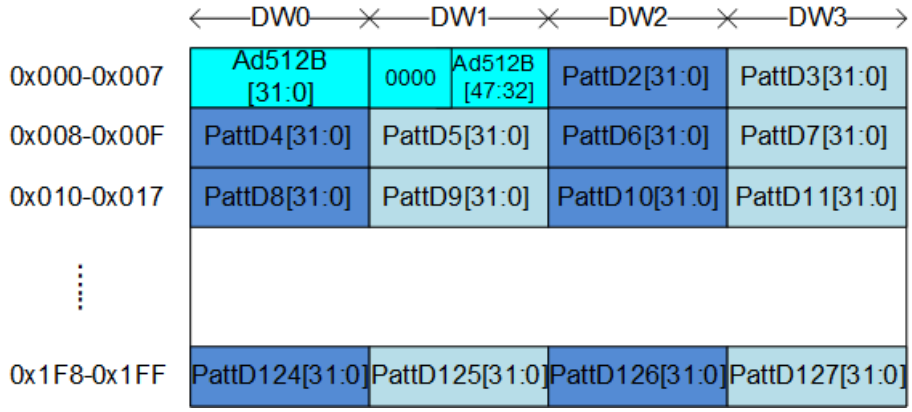

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

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 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 256-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.

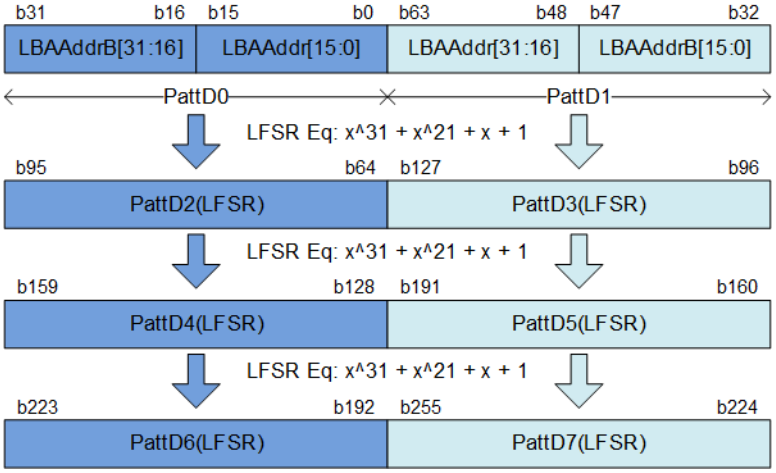

Figure 2-5 256-bit LFSR Pattern in TestGen

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.

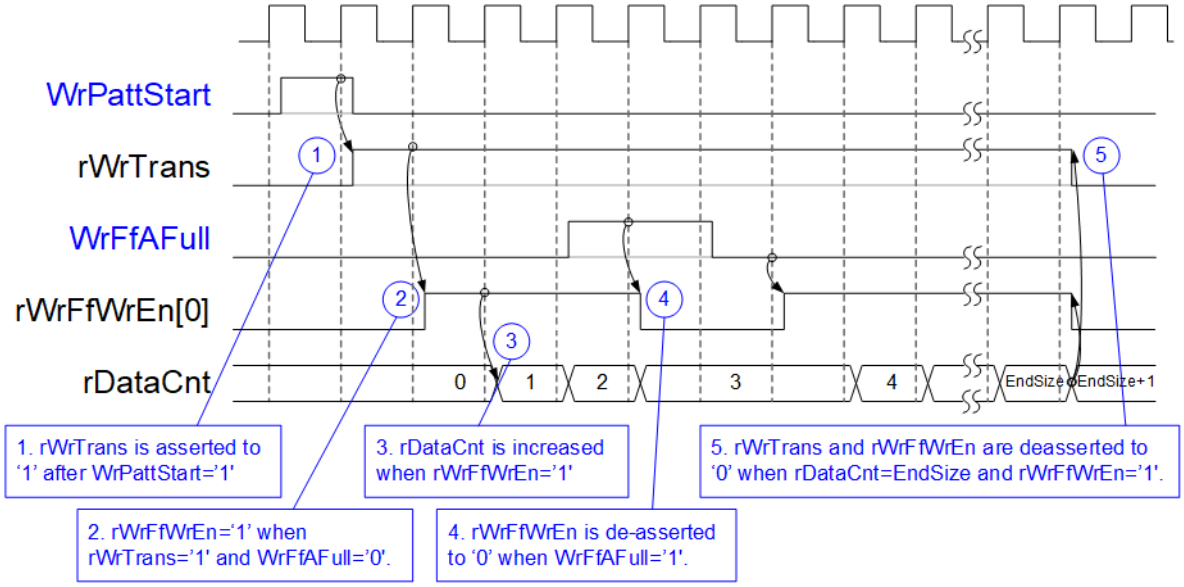

Figure 2-6 Timing diagram of Write operation in TestGen

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 expected value.

2.2 UFf (Ff2D512x256)

UFf (U0Ff, U1Ff) is the bi-directional FIFO that connects between TestGen and RAID0x2 module for each user interface. It is a special FIFO that is designed by using true-dual port RAM to have two write interfaces and two read interfaces. However, there is only one write interface and one read interface while operating Write command or Read command. Data width of the RAM is 256 bits while the RAM depth is 512 words. There is a special control signal, named “Dir”. When Dir is equal to ‘0’, the transfer direction of the FIFO is A to B (Write by A and Read by B). On the contrary, the transfer direction is B to A when Dir is equal to ‘1’.

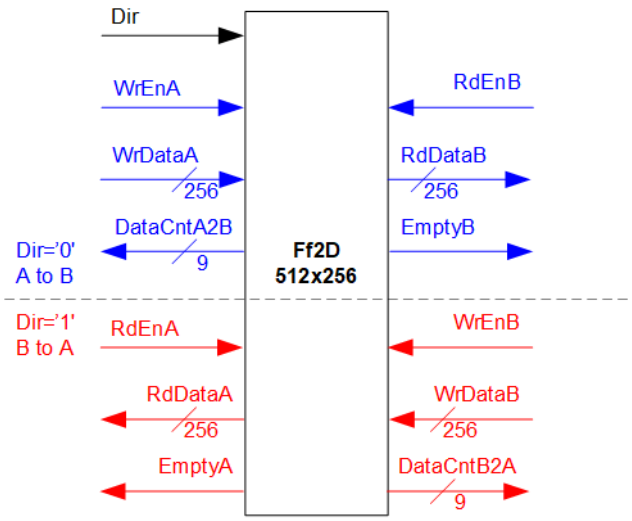

Figure 2-7 Ff2D512x256 module

Bi-directional FIFO has only one memory. It is recommended for user to read all data in FIFO until FIFO is empty before switching transfer direction of FIFO. Timing diagram of FIFO is similar to general FIFO. The read data (RdDataA/B) is valid in the next clock after asserting read enable (RdEnA/B) to ‘1’. While write data (WrDataA/B) must be valid at the same clock as asserting write enable (WrEnA/B) to ‘1’ to write each data to FIFO. DataCntA2B/B2A can be applied to check the amount of data in FIFO.

2.3 muNVMeRAID0x2IP

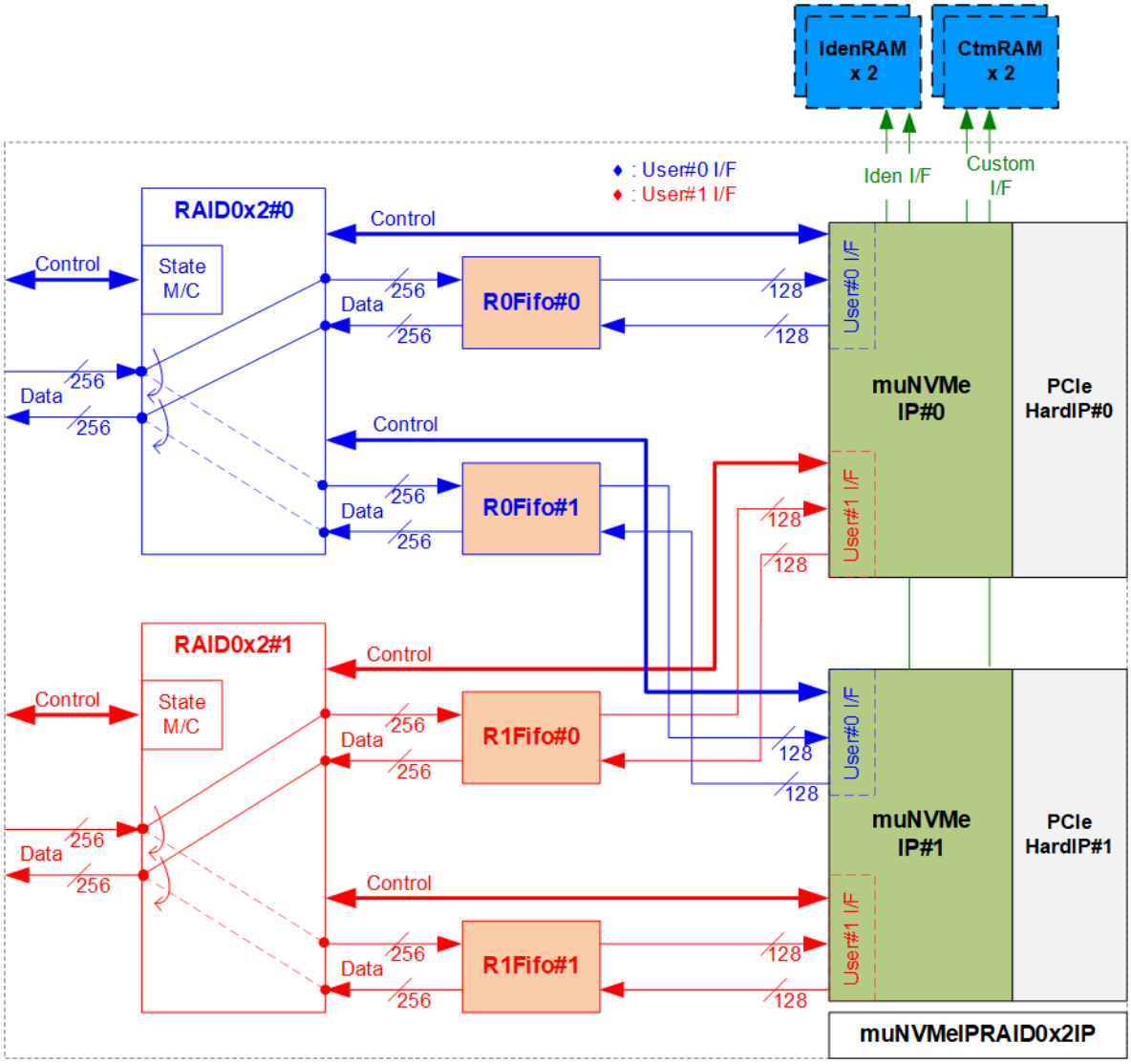

Figure 2-8 muNVMeRAID0x2IP

Figure 2-8 shows the connections and the submodules inside muNVMeRAID0x2IP. The user interface of muNVMeRAID0x2 consists of control interface and 256-bit data interface while the data interface of muNVMe-IP is 128 bits. Therefore, four bi-directional FIFOs which are asymmetric FIFO are integrated between RAID0x2 and muNVMe-IP. RAID0x2 decodes the command request from control interface (by LAxi2Reg) and then creates the request with the calculated parameters to control interface of two muNVMe-IPs. Also, RAID0x2 includes data switch to connect data interface of TestGen to one of two muNVMe-IPs when running Write command or Read command. The connection of RAID0x2#0 is mapped to User#0 I/F of two muNVMe-IPs whil the connection of RAID0x2#1 is mapped to User#1 I/F of two muNVMe-IPs. Therefore, the write data of TestGen#0 and TestGen#1 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.

2.3.1 muNVMe-IP

The muNVMe-IP implements NVMe protocol of the host side to access one NVMe SSD directly without PCIe switch connection. The muNVMe-IP supports two users. The first user (Main user) supports six commands, i.e., Write, Read, Identify, Shutdown, SMART, and Flush. The second user (Sub user) supports two commands - Write and Read. muNVMe-IP can connect to the PCIe hard IP directly. More details of muNVMe-IP are described in datasheet. https://dgway.com/products/IP/NVMe-IP/dg_munvme_ip_data_sheet_en.pdf

2.3.2 Integrated Block for PCIe

This block is the hard IP which is available in some Xilinx FPGAs. It implements Physical, Data Link, and Transaction Layers of PCIe protocol. The maximum number of SSDs connecting to one FPGA device is limited by the numbers of PCIe hard IP. One muNVMe-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

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.

UG1075: Zynq UltraScale+ Device Packaging and Pinouts.

The example of PCIe hard IP location on XCZU7-FFVC1156 is shown in Figure 2-9.

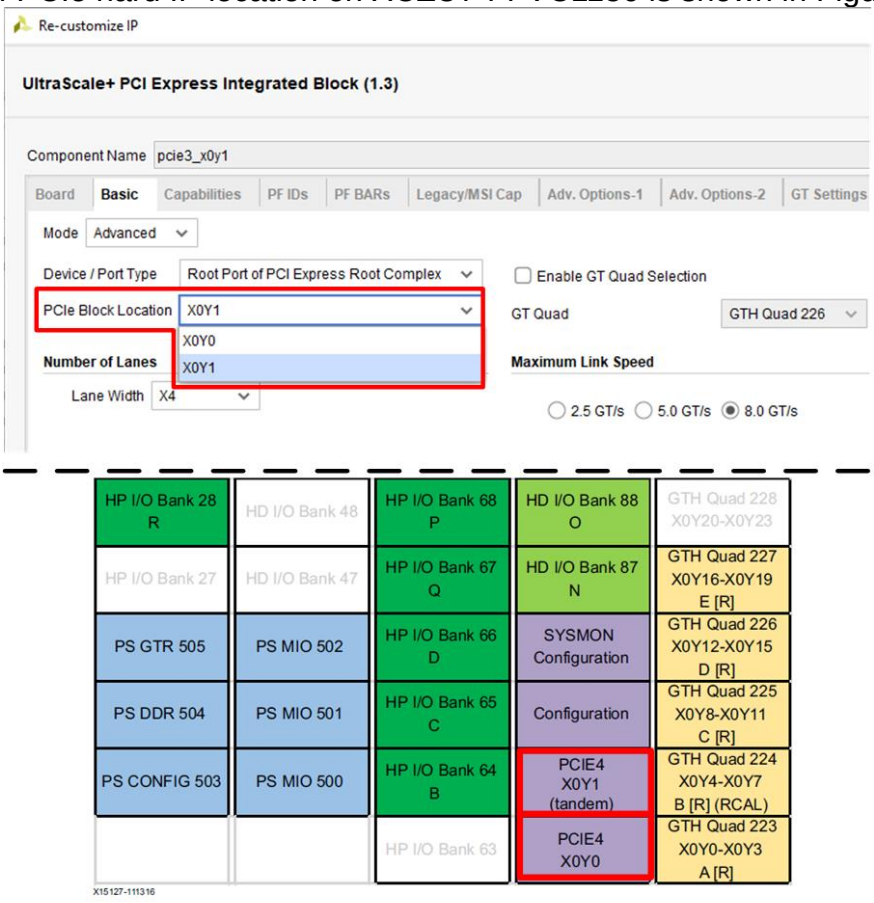

Figure 1-17: **XCZU7 Banks in FFVC1156 Package and XQZU7 Banks in FFRC1156 Package**

Figure 2-9 PCIe Hard IP Pin location

2.3.3 Dual port RAM

Two dual port RAMs, CtmRAM and IdenRAM, store data from Identify command and SMART command, respectively. IdenRAM has 8-Kbyte size to store 8-Kbyte data, output from Identify command. muNVMe-IP and LAxi2Reg have the different data bus size, 128 bits on muNVMe-IP but 32 bits on LAxi2Reg. Therefore, IdenRAM is asymmetric RAM that has the different bus size on Write interface and Read interface. Also, muNVMe-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 2-10.

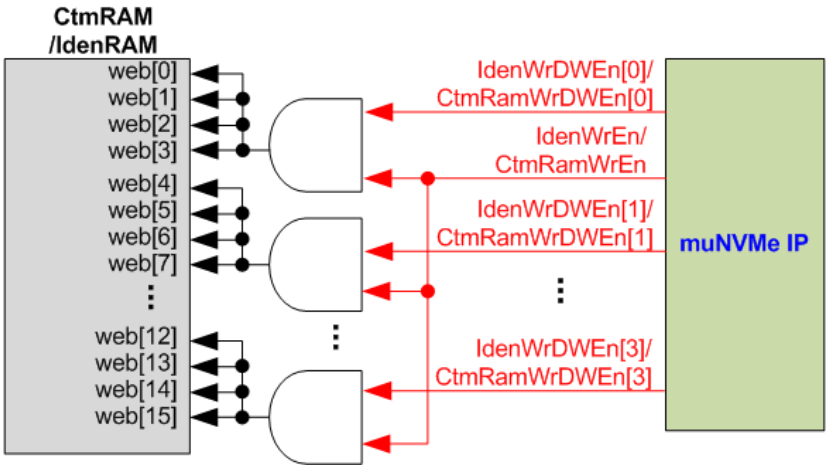

Figure 2-10 Byte write enable conversion logic

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], and [3] of WrDWEn are applied to be bit[7:4], [11:8], and [15:12] of IdenRAM write byte enable, respectively.

Comparing with IdenRAM, CtmRAM is implemented by true dual-port RAM 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. The data size returned from SMART command is 512 bytes.

2.3.4 RFf (Ff2D512x256to128)

RFf#0 (R0Ff#0, R1Ff#0) and RFf#1 (R0Ff#1, R1Ff#1) are the bi-directional FIFO that connects between RAID0x2 and muNVMe-IP for each user interface. There are four FIFOs for four connections. This FIFO is asymmetric FIFO. The data width of A-side which connects to RAID0x2 is 256 bits while the data width of B-side which connects to muNVMe-IP is 128 bits. Similar to UFf, this FIFO is implemented by using true-dual port RAM to have two write interfaces and two read interfaces. “Dir” signal is applied to select the transfer direction of the FIFO. When Dir=’0’, the transfer direction is A to B (Write by A and Read by B). When Dir=’1’, the transfer direction is B to A (Write by B and Read by A).

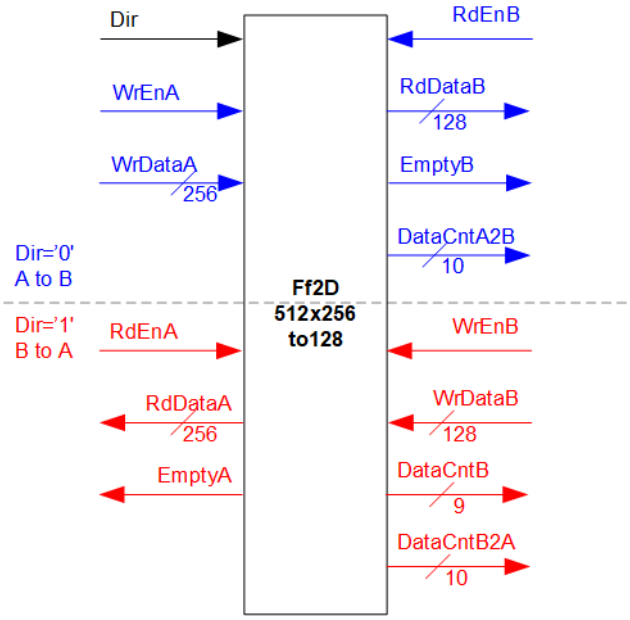

Figure 2-11 Ff2D512x256to128 module

Inside the FIFO, there is only one memory. The user should read all data from the FIFO until it is empty before switching transfer direction by changing “Dir” signal. To write the FIFO, Write data (WrDataA/B) must be valid at the same clock as asserting write enable (WrEnA/B) to ‘1’ to write one data to FIFO. To read the data, the read data (RdDataA/B) is valid in the next clock after asserting read enable (RdEnA/B) to ‘1’. The data counter (DataCntA2B/B2A) of the FIFO shows the amount of data in 128-bit unit. To check the amount of 256-bit data, please ignore the LSB bit of the data counter.

2.3.5 RAID0x2

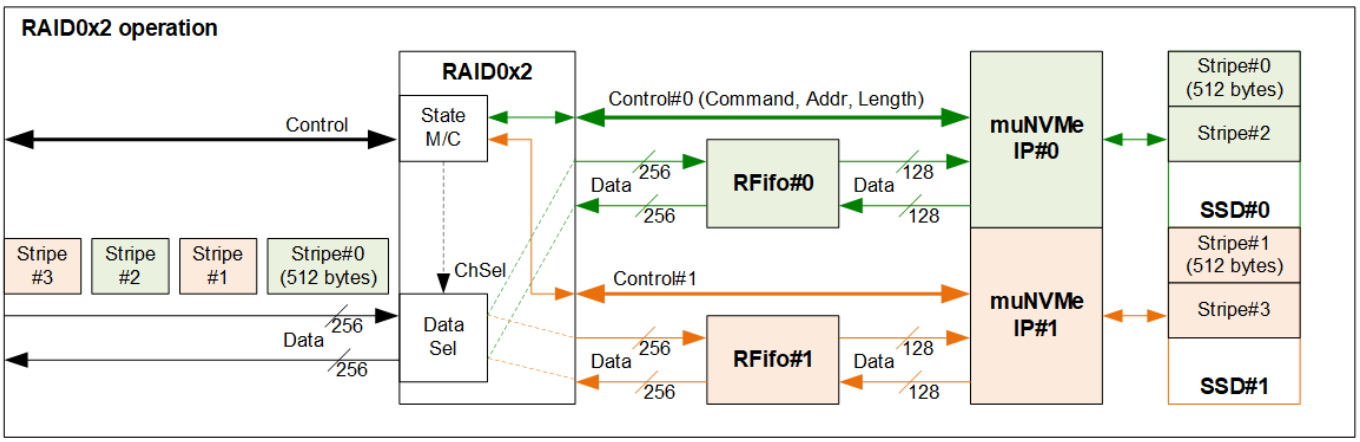

Figure 2-12 RAID0x2 operation

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 muNVMe-IP by decoding user parameter inputs (set by LAxi2Reg). Next, state machine generates command request with the valid address and length to muNVMe-IPs. Typically, the length to each muNVMe-IP is equal to (the length from user / 2).

Figure 2-12 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 muNVMe-IP and then transfers the first 512-byte data to it. Figure 2-12 shows the example when the first active muNVMe-IP is Ch#0, so Stripe#0 is stored to SSD#0 via muNVMe-IP#0 and RFifo#0. Next, the active channel is toggled to Ch#1. The next stripe (Stripe#1) is stored to SSD#1 via muNVMe-IP#1 and RFifo#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 16 clock cycles of 256-bit data). As a result, the overhead time is about 6.25% (1 cycle/16 cycles). To compensate the overhead time, clock frequency of RAID0x2 must be set to 6.25% higher than NVMe based clock. NVMe based clock for PCIe Gen3 is equal to 250 MHz. The clock frequency after compensating overhead time is at least 265.625 MHz (106.25% of 250 MHz). In the reference design, 280 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 muNVMe-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 262.625 MHz for PCIe Gen3 to achieve the best performance. The minimum requirement of Clk signal is the same as muNVMe-IP (more than or equal to 250 MHz).
UserCmd[2:0]	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 when 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').
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 muNVMe-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 bits, please fill '1' to remained upper bit.
UserFifoWrEn	Out	Asserted to '1' to write data to Receive FIFO while running Read command.
UserFifoWrData[255:0]	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 bits, 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[255:0]	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.

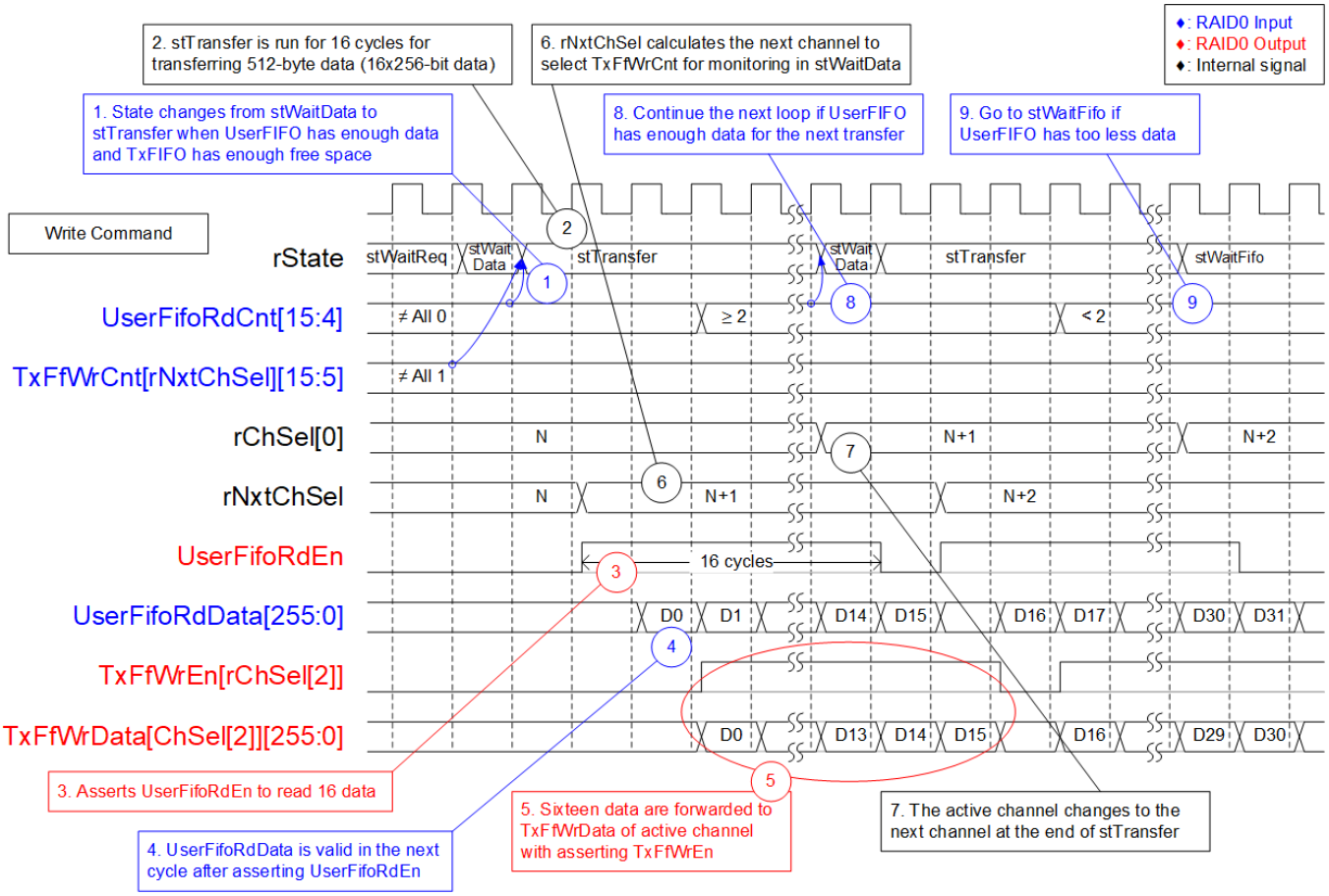

Figure 2-13 RAID0x2 timing diagram in Write command

When user sends Write command to RAID0, data is forwarded from UserFIFO (UFf) to TxFIFO[0] (RFf#0) or TxFIFO[1] (RFf#1). One TxFIFO is active to transfer 512-byte data at a time. After that, the active channel of muNVMe-IP 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 UserFIFO 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 16 clock cycles.
3) UserFifoRdEn is asserted when the state is stTransfer, so it is asserted to '1' for 16 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.

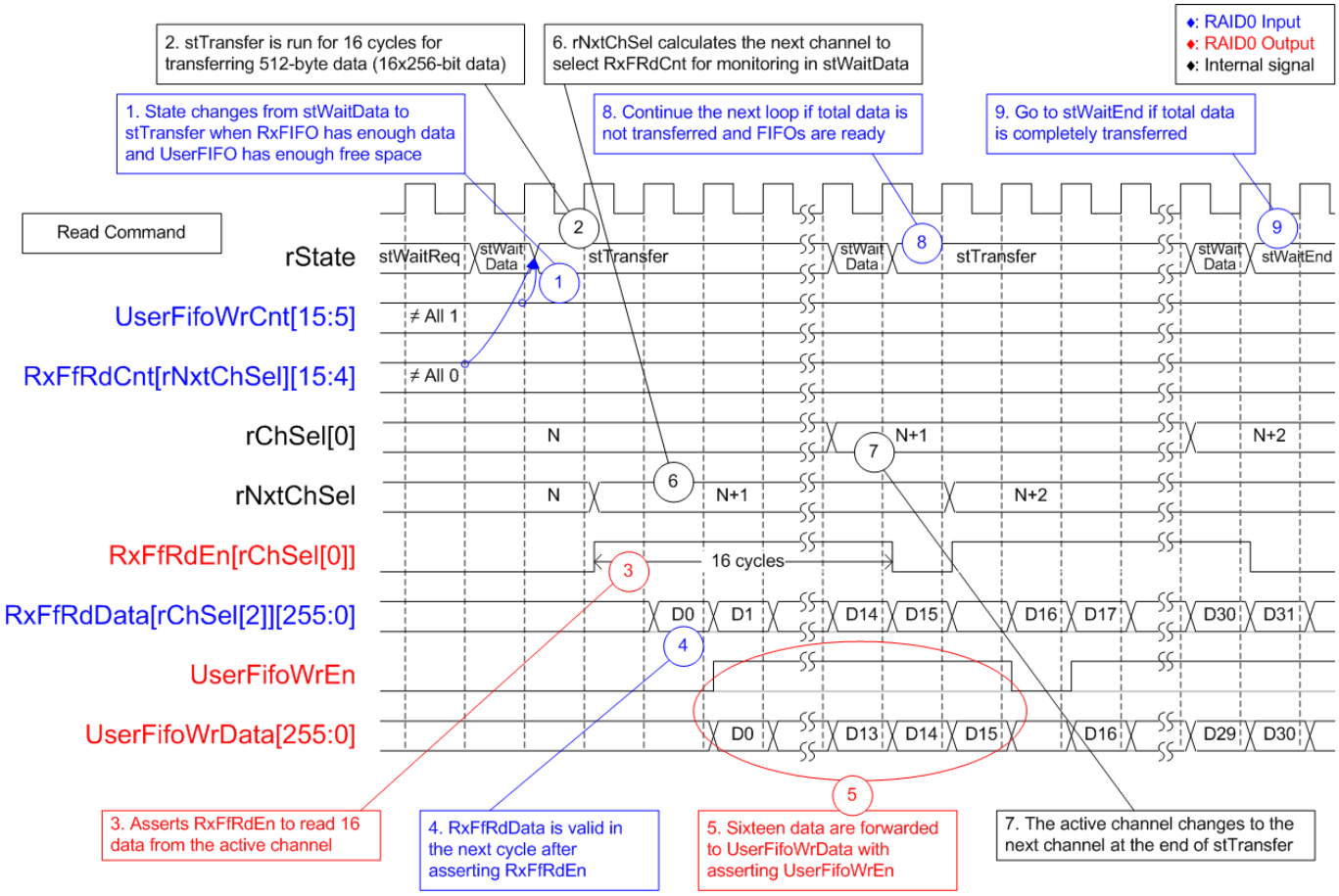

Figure 2-14 RAID0x2 timing diagram in Read command

When user sends Read command to RAID0, data is forwarded from RxFIFO[0] (RFf#0) or RxFIFO[1] (RFf#1) to UserFIFO (UFf). 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 UserFIFO 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 16 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 16 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 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) 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 muNVMe-IPs finish the operation by de-asserting Busy signal to '0'.

2.4 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 muNVMe-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. So, 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 2-15.

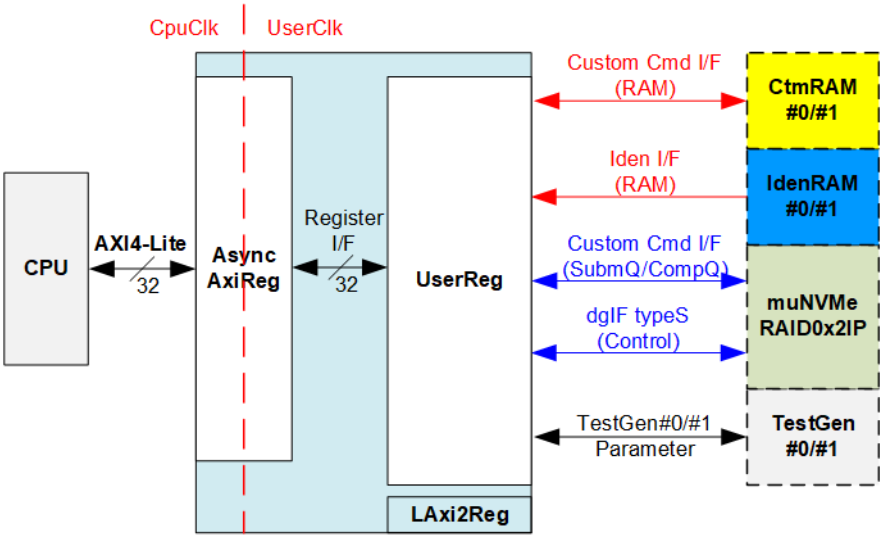

Figure 2-15 CPU and peripherals hardware

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, muNVMeRAID0x2IP, and TestGen. More details of AsyncAxiReg and UserReg are described as follows.

2.4.1 AsyncAxiReg

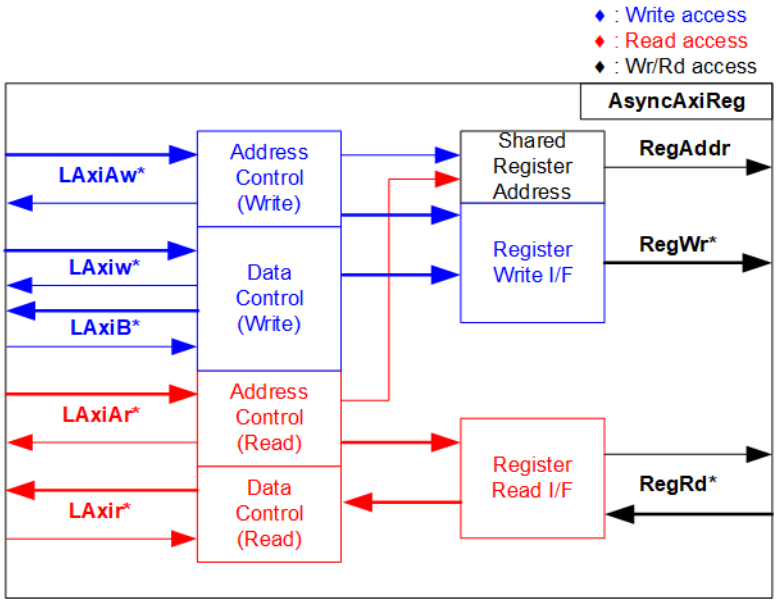

Figure 2-16 AsyncAxiReg Interface

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. 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 2-16. 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, so user cannot write and read the register at the same time. The timing diagram of the register interface is shown in Figure 2-17.

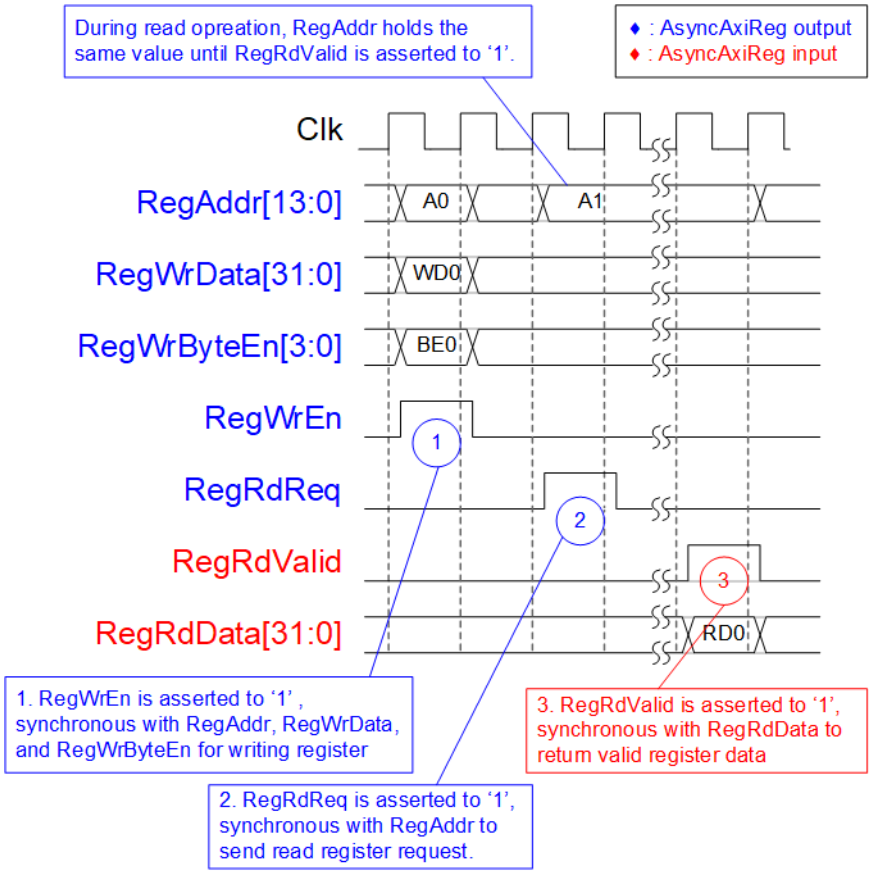

Figure 2-17 Register interface timing diagram

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

2.4.2 UserReg

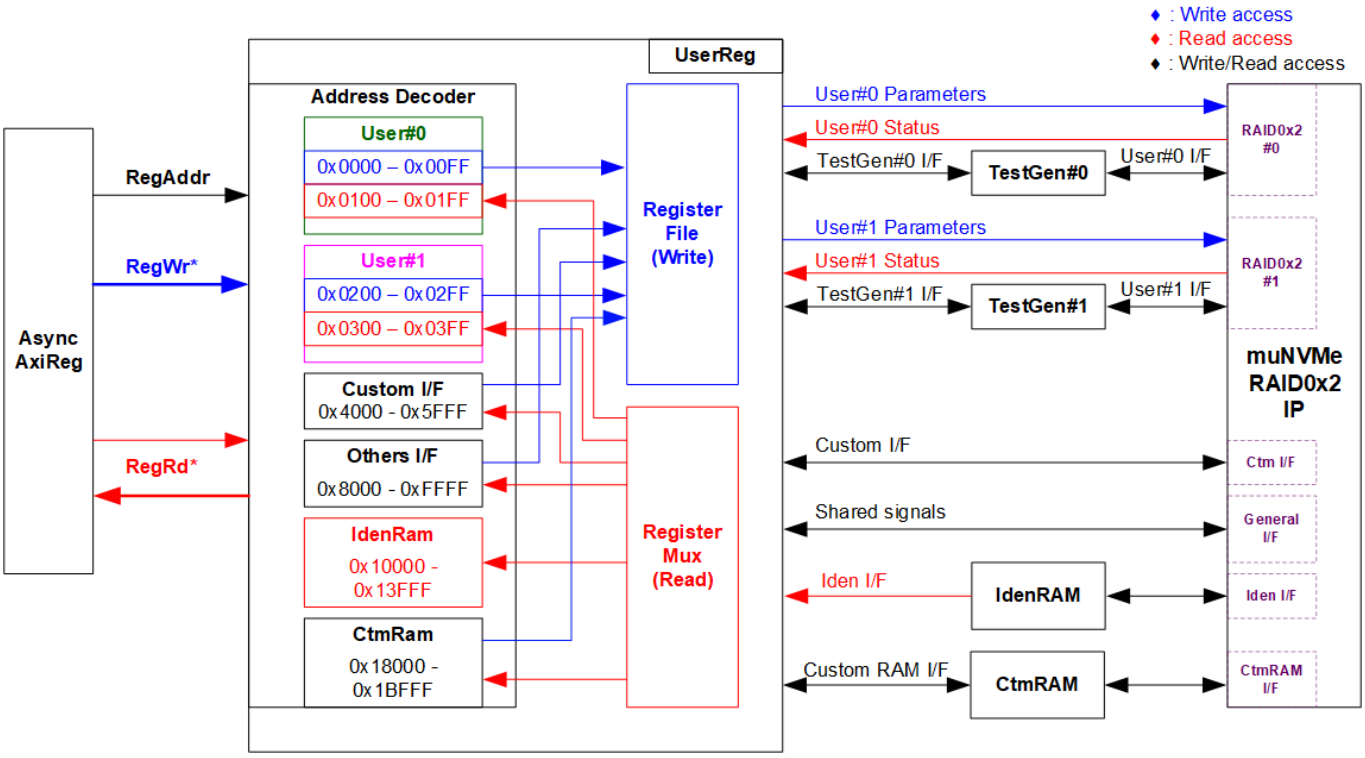

Figure 2-18 UserReg Interface

The logic inside UserReg consists of Address decoder, RegFile, and RegMux. The address decoder decodes the address which is requested from AsyncAxiReg and then selects the active register for write or read transaction. The address range assigned in UserReg is split into six areas, as shown in Figure 2-18.

1) 0x0000 – 0x01FF: mapped to RAID0x2#0 and TestGen#0
2) 0x0200 – 0x03FF: mapped to RAID0x2#1 and TestGen#1
3) 0x4000 – 0x5FFF: mapped to Custom command interface
4) 0x8000 – 0xFFFF: mapped to other interfaces such as shared parameters for all Users, PCIe status, and IP version.
5) 0x10000 – 0x13FFF: mapped to read data from IdenRAM. This area is read-access only.
6) 0x18000 – 0x1BFFF: mapped to write or read data with custom command RAM interface. This area supports write-access and read-access but the demo shows only read access by running SMART command.

Address decoder decodes the upper bit of RegAddr for selecting the active hardware that is RAID0x2, TestGen, IdenRAM, or CtmRAM. 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 multiplexers select 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 defintion

Address Rd/Wr	Register Name (Label in the “munvmeraid0g3test.c”)	Description
0x0000 – 0x01FF: Signal Interface of RAID0x2#0 and TestGen#0		
0x0000 – 0x00FF: Control signals of User#0 and TestGen#0 (Write access only)		
BA+0x0000	User#0 Address (Low) Reg (U0ADRL_INTREG)	[31:0]: Input to be bit[31:0] of User#0 start address as 512-byte unit (U0Addr[31:0] of RAID0x2#0)
BA+0x0004	User#0 Address (High) Reg (U0ADRH_INTREG)	[15:0]: Input to be bit[47:32] of User#0 start address as 512-byte unit (U0Addr[47:32] of RAID0x2#0)
BA+0x0008	User#0 Length (Low) Reg (U0LENL_INTREG)	[31:0]: Input to be bit[31:0] of User#0 transfer length as 512-byte unit (U0Len[31:0] of RAID0x2#0)
BA+0x000C	User#0 Length (High) Reg (U0LENH_INTREG)	[15:0]: Input to be bit[47:32] of User#0 transfer length as 512-byte unit (U0Len[47:32] of RAID0x2#0)
BA+0x0010	User#0 Command Reg (U0CMD_INTREG)	[2:0]: Input to be User#0 command (UserCmd of RAID0x2#0) 000b: Identify, 001b: Shutdown, 010b: Write SSD, 011b: Read SSD, 100b: SMART, 110b: Flush, 101b/111b: Reserved When this register is written, the command request is sent to RAID0x2#0 to start the operation.
BA+0x0014	User#0 Test Pattern Reg (U0PATTSEL_INTREG)	[2:0]: Select test pattern of TestGen#0 000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1, 100b-LFSR
0x0100 – 0x01FF: Status signals of RAID0#0 and TestGen#0 (Read access only)		
BA+0x0100	User#0 Status Reg (U0STS_INTREG)	[0]: User#0 Busy of RAID0x2#0 (‘0’: Idle, ‘1’: Busy) [1]: U0Error of RAID0x2#0 (‘0’: Normal, ‘1’: Error) [2]: Data verification fail in TestGen#0 (‘0’: Normal, ‘1’: Error)
BA+0x0110- BA+0x0117	User#0 Error Type Reg (U0ERRTYPECH0-1_INTREG)	0x0110[31:0]: Mapped to U0ErrorType[31:0] of muNVMeIP#0 0x0114[31:0]: Mapped to U0ErrorType[31:0] of muNVMeIP#1
BA+0x0120- BA+0x0127	User#0 Completion Status Reg (U0COMPSTSCH0-1_INTREG)	0x0120[15:0]: Mapped to U0AdmCompStatus[15:0] of muNVMeIP#0 0x0120[31:16]: Mapped to U0IOCompStatus[15:0] of muNVMeIP#0 0x0124[15:0]: Mapped to U0AdmCompStatus[15:0] of muNVMeIP#1 0x0124[31:16]: Mapped to U0IOCompStatus[15:0] of muNVMeIP#1
BA+0x0130- BA+0x0137	User#0 Test pin (Low) Reg (U0TESTPINLCH0-1_INTREG)	0x0130[31:0]: Mapped to U0TestPin[31:0] of muNVMeIP#0 0x0134[31:0]: Mapped to U0TestPin[31:0] of muNVMeIP#1
BA+0x0140- BA+0x0147	User#0 Test pin (High) Reg (U0TESTPINHCH0-1_INTREG)	0x0140[15:0]: Mapped to U0TestPin[47:32] of muNVMeIP#0 0x0144[15:0]: Mapped to U0TestPin[47:32] of muNVMeIP#1
BA+0x0170	User#0 Data Failure Address(Low) Reg (U0RDFAILNOL_INTREG)	[31:0]: Bit[31:0] of the byte address of the 1st failure data in TestGen#0 when operating Read command
BA+0x0174	User#0 Data Failure Address(High) Reg (U0RDFAILNOH_INTREG)	[24:0]: Bit[56:32] of the byte address of the 1st failure data in TestGen#0 when operating Read command
BA+0x0178	User#0 Current test byte (Low) Reg (U0CURTESTSIZEL_INTREG)	[31:0]: Bit[31:0] of the current test data size in TestGen#0 module
BA+0x017C	User#0 Current test byte (High) Reg (U0CURTESTSIZEH_INTREG)	[24:0]: Bit[56:32] of the current test data size of TestGen#0 module
BA+0x0180- BA+0x019F	User#0 Expected value Word0-7 Reg (U0EXPPATW0-W7_INTREG)	128-bit of the expected data at the 1st failure data in TestGen#0 when operating Read command 0x0180: Bit[31:0], 0x0184: Bit[63:32], …, 0x019C: Bit[255:224]
BA+0x01C0- BA+0x01DF	User#0 Read value Word0-7 Reg (U0RDPATW0-W7_INTREG)	128-bit of the read data at the 1st failure data in TestGen#0 when operating Read command 0x01C0: Bit[31:0], 0x01C4: Bit[63:32], …, 0x01DC: Bit[255:224]

Address Rd/Wr	Register Name (Label in the “munvmeraid0g3test.c”)	Description
0x0200 – 0x03FF: Signal Interface of RAID0#1 (muNVMe-IP) and TestGen#1		
0x0200 – 0x02FF: Control signals of User#1 and TestGen#1 (Write access only)		
BA+0x0200- BA+0x0217	Control signals of User#1/TestGen#1 (U1ADRL_INTREG – U1PATTSEL_INTREG)	Similar to 0x0000 – 0x0017 which are the registers of RAID0x2#0 and TestGen#0, these are mapped to RAID0x2#1 and TestGen#1 instead. However, U1CMD_INTREG can be assigned by two values for two commands only – 010b: Write SSD and 011b: Read SSD.
0x0300 – 0x03FF: Status signals of User#1 and TestGen#1 (Read access only)		
BA+0x0300- BA+0x03FF	Status signals of User#1/TestGen#1 (U1STS_INTREG – U0RDPATW7_INTREG)	Similar to 0x0100 – 0x01FF which are the registers of RAID0x2#0 and TestGen#0, these are mapped to RAID0x2#1 and TestGen#1 instead. However, U1COMPSTSCH0-1_INTREG[31:16] is applied to map to U1IOCompStatus[15:0] of muNVMe-IP while bit[15:0] is reserved. U1TESTPINLCH0-1_INTREG[15:0] is mapped to U1TestPin[15:0] while U1TESTPINLCH0-1_INTREG[31:16] and U1TESTPINH_INTREG are reserved.
0x4000 – 0x5FFF: Custom Command Interface		
BA+0x4000- BA+0x407F Wr	Custom Submission Queue Reg (CTMSUBMQCH0/1_STRUCT)	Submission queue entry of SMART and Flush command. 0x4000-0x403F: Input to be CtmSubmDW0-DW15 of muNVMeIP#0. 0x4000: DW0, 0x4004: DW1, …, 0x403C: DW15 0x4040-0x407F: Input to be CtmSubmDW0-DW15 of muNVMeIP#1. 0x4040: DW0, 0x4044: DW1, …, 0x407C: DW15
BA+0x5000- BA+0x501F Rd	Custom Completion Queue Reg (CTMCOMPQCH0/1_STRUCT)	0x5000-0x500F: CtmCompDW0-DW3 output from muNVMeIP#0. 0x5000: DW0, 0x5004: DW1, …, 0x500C: DW3 0x5010-0x501F: CtmCompDW0-DW3 output from muNVMeIP#1. 0x5010: DW0, 0x5014: DW1, …, 0x501C: DW3
0x8000 – 0xFFFF: Other Interfaces		
BA+0x8000 Wr	NVMe Timeout Reg (NVMTIMEOUT_INTREG)	[31:0]: Mapped to TimeOutSet[31:0] of muNVMeIP#0-1
BA+0x8100- BA+0x8107 Rd	PCIe Status Reg (PCIESTSCH0-1_INTREG)	0x8100: Mapped to PCIe hard IP status of channel#0 0x8104: Mapped to PCIe hard IP status of channel#1 [0]: PCIe linkup status from PCIe hard IP (‘0’: No linkup, ’1’: linkup) [3:2]: PCIe link speed from PCIe hard IP (00b: Not linkup, 01b: PCIe Gen1, 10b: PCIe Gen2, 11b: PCIe Gen3) [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 Integrated Block for PCIe datasheet
BA+0x8110- BA+0x8117 Rd	NVMe CAP Reg (NVMCAPCH0-1_INTREG)	0x8110[31:0]: Mapped to NVMeCAPReg[31:0] of muNVMeIP#0 0x8114[31:0]: Mapped to NVMeCAPReg[31:0] of muNVMeIP#1
BA+0x8120 Rd	Total disk size (Low) Reg (LBASIZEL_INTREG)	[31:0]: Mapped to LBASize[31:0] of RAID0x2#0 to show RAID0 capacity
BA+0x8124 Rd	Total disk size (High) Reg (LBASIZEH_INTREG)	[15:0]: Mapped to LBASize[47:32] of RAID0x2#0 to show RAID0 capacity [31]: Mapped to LBAMode of RAID0x2
BA+0x8200 Rd	IP Version Reg (IPVERSION_INTREG)	[31:0]: Mapped to IPVersion[31:0] of muNVMe-IP

Address Rd/Wr	Register Name (Label in the “munvmeraid0g3test.c”)	Description
0x10000 – 0x13FFF: Identify RAM (Read access)		
BA+0x10000- BA+0x13FFF	Identify Data (IDENCTRLCH0-1_CHARREG/ IDENNAMECH0-1_CHARREG)	0x10000-0x10FFF: 4Kbyte Identify controller data of muNVMeIP#0 0x11000-0x11FFF: 4Kbyte Identify namespace data of muNVMeIP#0 0x12000-0x12FFF: 4Kbyte Identify controller data of muNVMeIP#1 0x13000-0x13FFF: 4Kbyte Identify namespace data of muNVMeIP#1
0x18000 – 0x1BFFF: Custom RAM (Write/Read access)		
BA+0x18000- BA+0x1BFFF Wr/Rd	Custom command Ram (CTMRAMCH0-1_CHARREG)	0x18000-0x19FFF: 8-Kbyte Custom RAM of muNVMeIP#0, 0x1A000-0x1BFFF: 8-Kbyte Custom RAM of muNVMeIP#1 8Kbyte CtmRAM interface is applied to store 512-byte data output from SMART Command.

3 CPU Firmware

3.1 Test firmware (munvmeraid0g3test.c)

After system boot-up, CPU starts the initialization sequence as follows.

1) CPU initializes UART and Timer parameters.
2) CPU waits until PCIe connection links up (PCIESTSCH0/1_INTREG[0]='1').
3) CPU waits until muNVMeRAID0x2 completes initialization process (U0/1STS_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 PCIESTSCH0/1_INTREG[7:2].
5) CPU displays the main menu. There are five menus for running six commands with RAID0x2, i.e., Identify, Write, Read, SMART, Flush, and Shutdown.

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

3.1.1 Identify Command

This command can be requested by User#0 I/F. The sequence of the firmware when user selects Identify command is below.

1) Set U0CMD_INTREG=000b to send Identify command request on User#0 I/F of muNVMeRAID0x2. After that, busy flag of User#0 I/F (U0STS_INTREG[0]) changes from '0' to '1'.
2) CPU waits until the operation is completed or some errors are found by monitoring U0STS_INTREG[1:0].

   Bit[0] is de-asserted to '0' after finishing operating the command. Next, the data from Identify command of two muNVMe-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 U0ERRTYPECH0/1_INTREG[31:0]. Finally, the process is stopped.

3) After busy flag (U0STS_INTREG[0]) is de-asserted to '0', CPU displays some information decoded from IdenRAM (IDENCTRLCH0/1_CHARREG) such as SSD model name. Also, RAID0x2 capacity (LBASIZEL/H_INTREG) from muNVMeRAID0x2 output is read and displayed on the console. Finally, CPU checks LBA Size of RAID0x2 (LBASIZEH_REG[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

Write and Read command can be requested by User#0 I/F and User#1 I/F. The sequence of the firmware when user selects Write/Read command is below.

1) Receive 2-user parameters such as command (Write, Read, or Disable), start address, transfer length, and test pattern from Serial console. If some inputs are invalid, the operation is cancelled.
2) Get all inputs and set to U(0/1)ADRL/H_INTREG, U(0/1)LENL/H_INTREG, and U(0/1)PATTSEL_INTREG.
3) Set U(0/1)CMD_INTREG[2:0] = 010b for Write command or 011b for Read command and then the command request is asserted to the user who runs the Write command or Read command. After that, busy flag of the active user (U(0/1)STS_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 U(0/1)STS_INTREG[2:0].

   Bit[0] is de-asserted to '0' when command of User#0/#1 is completed.
   Bit[1] is asserted when error is detected in User#0/#1. After that, error message is displayed on the console to show the error details. Finally, the process is hanged up.
   Bit[2] is asserted when data verification is failed in User#0/#1. After that, the verification error message is displayed. However, CPU is still running until the operation is done or user inputs any key to cancel operation.

   While the command is running, current transfer size of the active user read from U(0/1)CURTESTSIZEL/H_INTREG is displayed every second.

5) After both busy flags (U(0/1)STS_INTREG[0]) are de-asserted to '0', CPU displays the test result of the active user on the console, i.e., total time usage, total transfer size, and transfer speed.

3.1.3 SMART Command

This command can be requested by User#0 I/F. The sequence of the firmware when user selects SMART command is below.

1) Set 16-Dword of Submission queue entry (CTMSUBMQCH0/1_STRUCT) to be SMART command value.
2) Set U0CMD_INTREG[2:0]=100b to send SMART command request on User#0 I/F of muNVMeRAID0x2. After that, busy flag of User#0 I/F (U0STS_INTREG[0]) changes from '0' to '1'.
3) CPU waits until the operation is completed or some errors are found by monitoring U0STS_INTREG[1:0].

   Bit[0] is de-asserted to '0' after finishing operating the command. Next, the data from SMART command is stored in CtmRAM.
   Bit[1] is asserted when some errors are detected. The error message is displayed on the console to show the error details, decoded from U0ERRTYPECH0/1_INTREG[31:0]. Finally, the process is stopped.

4) After busy flag (U0STS_INTREG[0]) is de-asserted to '0', CPU displays some information decoded from CtmRAM (CTMRAMCH0/1_CHARREG) such as 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

This command can be requested by User#0 I/F. The sequence of the firmware when user selects Flush command is below.

1) Set 16-Dword of Submission queue entry (CTMSUBMQCH0/1_STRUCT) to be Flush command value.
2) Set U0CMD_INTREG[2:0]=110b to send Flush command request on User#0 I/F of muNVMeRAID0x2. After that, busy flag of User#0 I/F (U0STS_INTREG[0]) changes from '0' to '1'.
3) CPU waits until the operation is completed or some errors are found by monitoring U0STS_INTREG[1:0].

   Bit[0] is de-asserted to '0' after finishing operating the command. Next, CPU returns to the main menu.
   Bit[1] is asserted when some errors are detected. The error message is displayed on the console to show the error details, decoded from U0ERRTYPECH0/1_INTREG[31:0]. Finally, the process is stopped.

3.1.5 Shutdown Command

This command can be requested by User#0 I/F. The sequence of the firmware when user selects Shutdown command is below.

1) Set U0CMD_INTREG[2:0]=001b to send Shutdown command request on User#0 I/F of muNVMeRAID0x2. After that, busy flag of User#0 I/F (U0STS_INTREG[0]) changes from '0' to '1'.
2) CPU waits until the operation is completed or some errors are found by monitoring U0STS_INTREG[1:0].

   Bit[0] is de-asserted to '0' after finishing operating the command. After that, the CPU goes to the next step.
   Bit[1] is asserted when some errors are detected. The error message is displayed on the console to show the error details, decoded from U0ERRTYPECH0/1_INTREG[31:0]. Finally, the process is stopped.

3) After busy flag (U0STS_INTREG[0]) is de-asserted to '0', all SSDs and all muNVMe-IPs change to inactive status. The CPU cannot receive the new command from user. The user must power off the test system.

3.2 Function list in Test firmware

int exec_ctm(unsigned int user_cmd)	
Parameters	user_cmd: 4-SMART command, 6-Flush command
Return value	0: No error, -1: Some errors are found in the muNVMeRAID0x2
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(unsigned int user)	
Parameters	user: 0-User#0, 1-User#1
Return value	Read value of U0/1CURTESTSIZEH/L_INTREG
Description	Read U0/1CURTESTSIZEH/L_INTREG and return read value as function result.

int get_param(userin_struct* userin)	
Parameters	userin: Four inputs from user, i.e., command, start address, total length in 512-byte unit, and test pattern
Return value	0: Valid input, -1: Invalid input
Description	Receive the input parameters from the user and verify the value. When the input is invalid, the function returns -1. Otherwise, all inputs are updated to userin parameter.

void iden_dev(void)	
Parameters	None
Return value	None
Description	Run Identify command, following in topic 3.1.1 (Identify Command).

void print_space(unsigned long long size_input)	
Parameters	size_input: test size for displaying on the console
Return value	None
Description	Calculate the number of digits and the number of spaces to display size_input with alignment on the console.

int setctm_flush(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the muNVMeRAID0x2
Description	Set Flush command to CTMSUBMQCH0/1_STRUCT and call exec_ctm function to operate Flush command.

int setctm_smart(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the muNVMeRAID0x2
Description	Set SMART command to CTMSUBMQCH0/1_STRUCT and call exec_ctm function to operate SMART command. Finally, decode and display SMART information on the console

void show_error(unsigned int user)	
Parameters	user: 0-User#0, 1-User#1
Return value	None
Description	Read U(0/1)ERRTYPECH0/1_INTREG, decode the error flag, and display error message following the error flag.

void show_pciestat(unsigned int channel)	
Parameters	Channel: 0-Channel#0, 1-Channel#1
Return value	None
Description	Read PCIESTSCH0/1_INTREG until the read value from two read times is stable. After that, display the read value on the console.

void show_result(unsigned int user, unsigned int cmd, unsigned int timeuseh, unsigned int timeusel)	
Parameters	user , cmd, timeuseh, timeusel
Return value	None
Description	Print user channel, command, and total size by calling get_cursize and show_size function. After that, calculate total time usage from timeuseh and timeusel and then 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(unsigned int user)	
Parameters	user: 0-User#0, 1-User#1
Return value	None
Description	Read U0/1RDFAILNOL/H_INTREG (error byte address), U0/1EXPPATW0-W7_INTREG (expected value), and U0/1RDPATW0-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(void)	
Parameters	None
Return value	0: No error, -1: Receive invalid input
Description	Run Write command or Read command, following in topic 3.1.2 (Write/Read Command)

4 Example Test Result

The test results when running demo system by 1-2 users are shown in Figure 4-1. The test environment uses two 280 GB Intel 900P and ZCU106 board (PCIe Gen3).

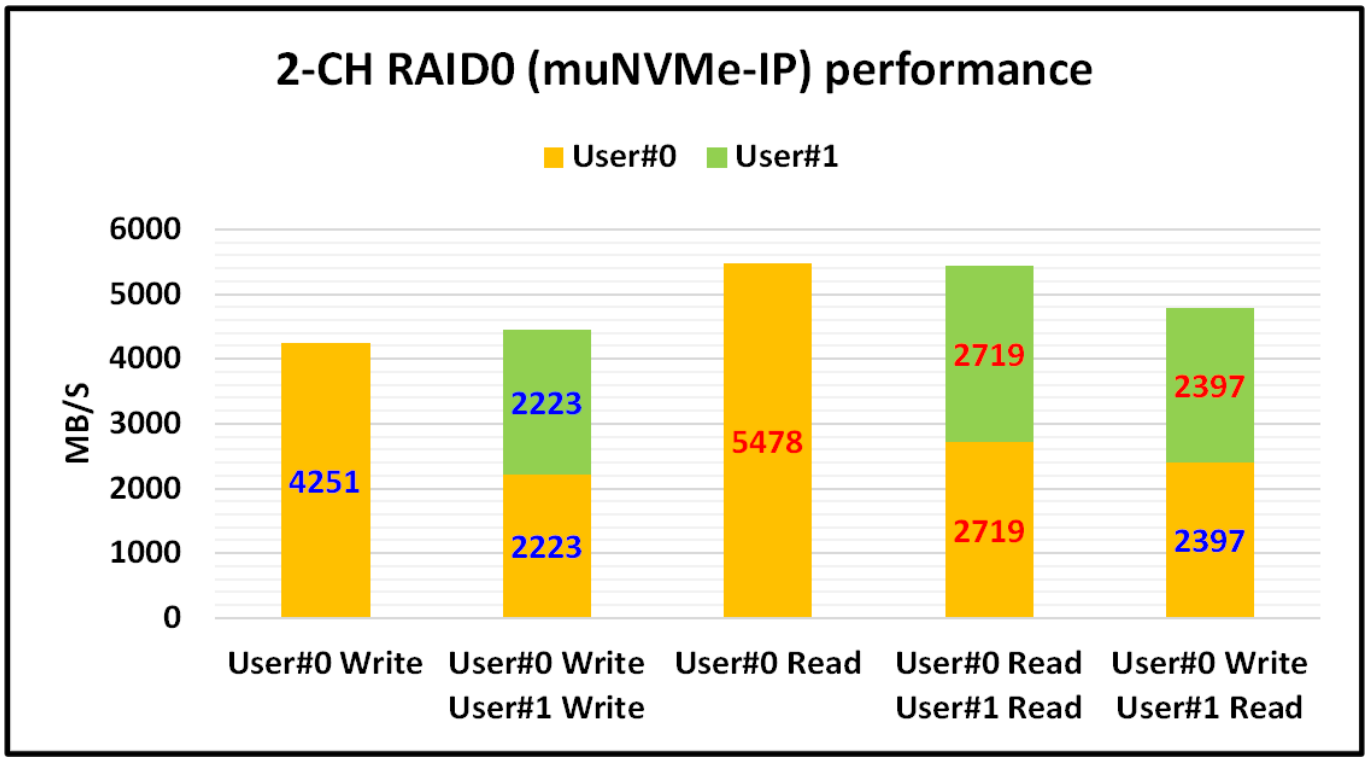

Figure 4-1 Test Performance of RAID0x2 by muNVMe-IP demo

Some SSDs can balance the device loading when Write and Read command are requested at the same time. As shown in Figure 4-1, the result shows the same performance on two users when two users run the same command or different command. Also, total performance of two users when running the same command is almost equal to the performance of that command by one user.

While some SSDs has different characteristic. The load balancing for running Write command and Read command is different. The test result of these SSDs is different when running two users by mixed Write/Read command, comparing to using the same commands.

5 Revision History

Revision	Date	Description
1.0	11-Aug-22	Initial Release