raNVMe-IP with multiple user reference design manual

Rev1.0 21-Aug-23

raNVMe-IP with multiple user reference design manual

Rev1.0 21-Aug-23

1 Overview

While raNVMe-IP standard demo is designed to show the performance of random access from the user by using random address, created by LFSR equation, raNVMe-IP with multiple user demo is designed to use the random feature for accessing different area by the different user, as shown in Figure 1-1. The demo is designed to support up to four users at the same time.

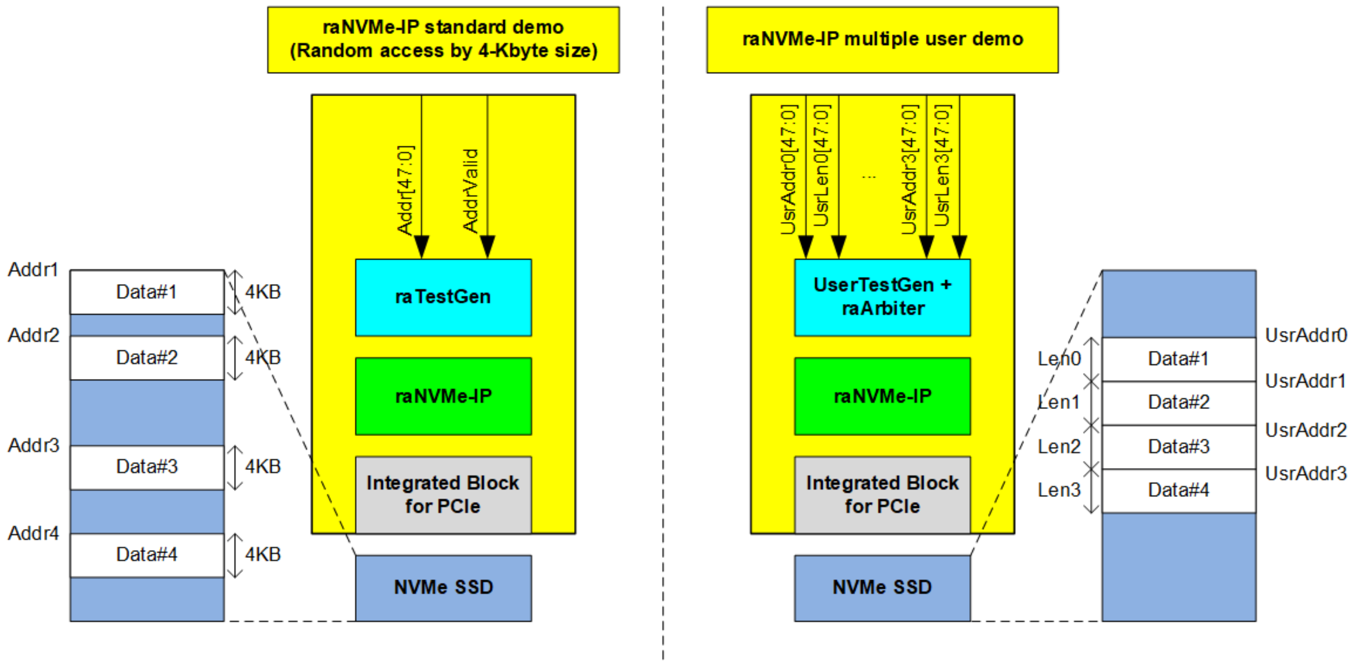

Figure 1-1 Standard demo and multiple user demo comparison

In multiple user demo, the start address of each user and transfer length of each user are different value. For reducing the number of parameters from the user in the test, the start address and the length of each user in Write/Read command is calculated by the firmware and the user sets only start address of the first user and total transfer length. Nevertheless, the user can modify the firmware to set the start address and transfer length of each user independently which are aligned to 4 KB unit.

After getting the parameters of each user, Write/Read command is split into many 4 KB Write/Read commands to raNVMe-IP. According to raNVMe-IP specification, Write and Read command are multiple command type which can be sent many requests at the same time. If the command from each user is different or the command is single command (Identify, SMART, Shutdown or Flush), only one command is sent to raNVMe-IP at a time. The example application for using multiple user demo is shown in Figure 1-2.

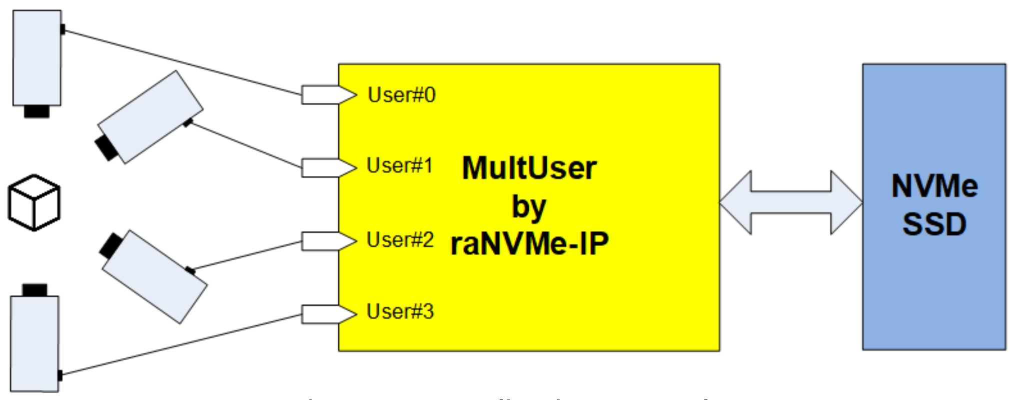

Figure 1-2 Application example

There are some applications which needs to record the data from many sources such as video camera system or multiple sensor system to the same storage. NVMe SSD can write and read data at very high-speed rate, so multiple sources which has lower-speed data rate can be connected for using full-bandwidth of NVMe SSD. Though the demo is designed to connect up to four users, the demo can be modified to support more users.

2 Hardware overview

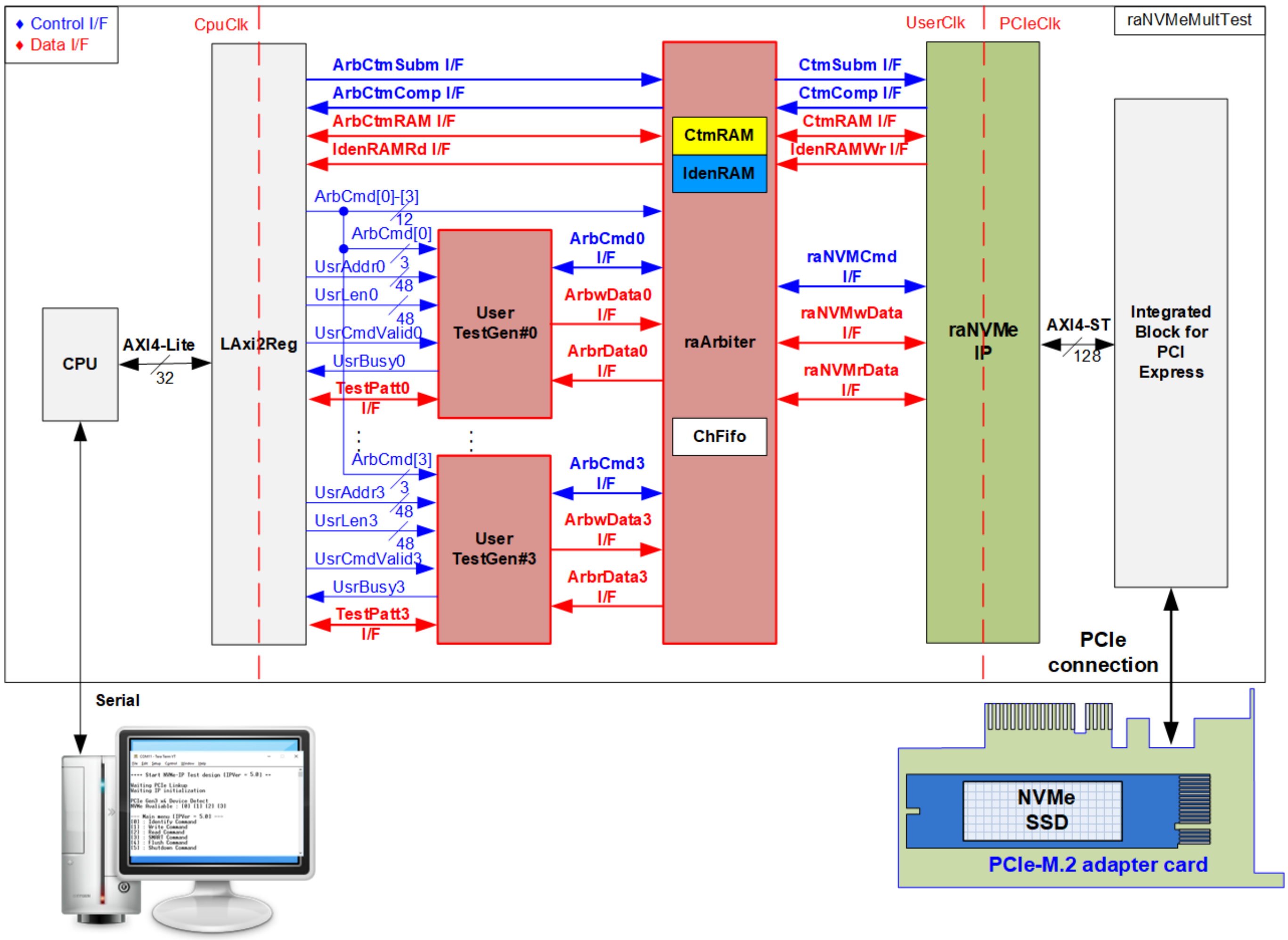

Figure 2-1 raNVMe-IP with multiple user demo hardware

In raNVMe-IP standard demo, one test module, designed as the dummy module of user logic, is connected with raNVMe-IP. In Multiple user demo, four test modules (UserTestGen) are designed to show that raNVMe-IP can support many users accessing the same NVMe SSD at the same time through raArbiter.

According to raNVMe-IP specification, there are two command types, i.e., Single command and Multiple command. raArbiter can operate many commands from many UserTestGen modules when the commands of all channels are all Write command or all Read command. Otherwise, only one command from one UserTestGen module is accepted and forwarded to raNVMe-IP at a time. To handle the order of the active user in command interface to be similar to data interface, ChFifo must be included. The active user of the command interface is written to ChFifo when raArbiter sends the command request to raNVMe-IP. The data interface reads ChFifo for selecting the same active user as command interface for transferring data with raNVMe-IP.

When running Write or Read command, each UserTestGen receives the start address and total length from CPU via LAxi2Reg. After that, UserTestGen creates many 4KB Write or Read command requests with assigning the address to raArbiter via ArbCmd I/F. The address is increased by 4 KB size for sequential access. The data is transferred via ArbwData I/F in Write command or ArbrData I/F in Read command. While UserTestGen creates one command request to raNVMe-IP when running other commands.

SMART command and Identify command are the Single command which needs to transfer the data. Identify and Custom interface of four channels are directly connected between LAxi2Reg and raArbiter. raArbiter selects the active channel and forwards the command to raNVMe-IP. CtmRAM and IdenRAM are included in raArbiter to store SMART data and Identify data of all channels independently. Though all user channels should get the same SMART data and Identify data because the system has one SSD, the design is purposed as the base system for flexibly customized by the user.

Though Multiple user demo can be all designed by pure-hardwire logic, CPU and LAxi2Reg are included for interfacing with user via Serial interface. In test system, it is flexible for checking many test conditions by adjusting command and the parameters on Serial console. Also, the current status, the test progress and the test result can be clearly monitored on Serial console.

There are three clock domains displayed in Figure 2-1, i.e., CpuClk, UserClk and PCIeClk. CpuClk is the clock domain of CPU and its peripherals. This clock must be stable clock which is independent from the other hardware interface. UserClk is the example user clock domain which may be independent clock for running the user interface of raNVMe-IP. According to raNVMe-IP datasheet, clock frequency of UserClk must be more than or equal to PCIeClk. So, this reference design uses 275/280 MHz. PCIeClk is the clock output from PCIe hard IP to synchronous with data stream of 128-bit AXI4 stream bus. When the PCIe hard IP is set to 4-lane PCIe Gen3, PCIeClk frequency is equal to 250 MHz.

More details of the hardware are described as follows.

2.1 UserTestGen

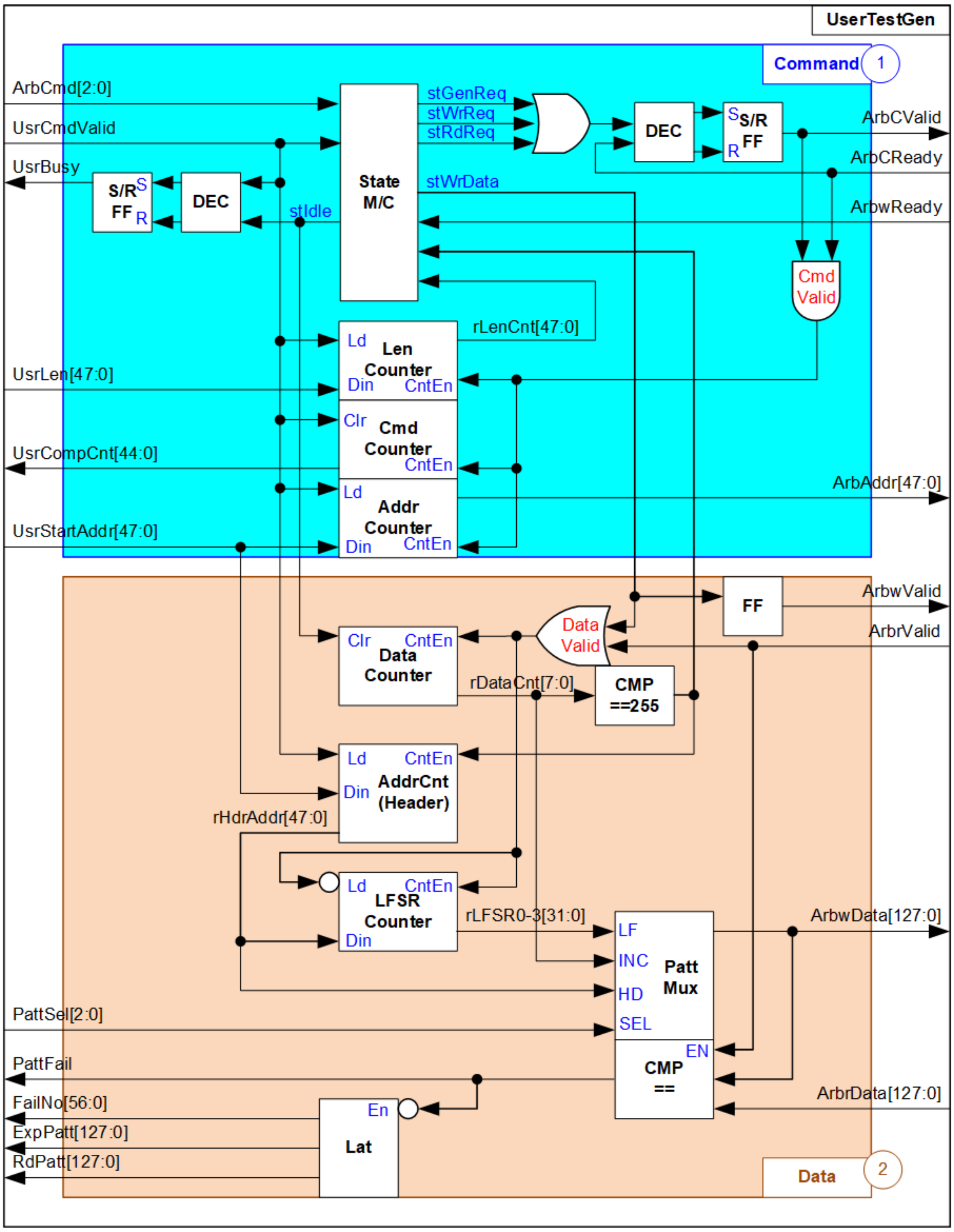

Figure 2-2 UserTestGen block diagram

UserTestGen is the module to send the command request. Also, it transfers the data with the Arbiter when running Write or Read command. In the system, there are four UserTestGen modules to show multiple user function by using raNVMe-IP. The command interface and the data interface of the Arbiter for each user is the same as raNVMe-IP user interface. When running Single command - Identify, SMART, Flush and Shutdown command, one command is run at a time. The data interface of Identify and SMART command it not connected to UserTestGen, but connected to raArbiter. Write and Read command are Multiple command and UserTestGen supports to generate many 4KB Write or Read commands to the Arbiter. The data interface of Write and Read command are connected to UserTestGen module.

Note: In multiple user design, the Arbiter must support many user modules. For simple design, there are some limitations of the Arbiter, described as follows.

(1) In Idle status, ArbCReady is de-asserted to '0'. The user logic must assert ArbCValid to send the new command request and then wait until the Arbiter asserts ArbCReady to '1' to accept the request.

(2) When running Write command, the user logic needs to send the command request (ArbCValid='1') before sending the data (ArbwValid='1'). The Arbiter must store the order of active user from command interface and send to the user interface to select the active user for sending the data in the same order. After that, ArbwReady is asserted to '1' by Arbiter to accept the Write data from the selected user.

As shown in Figure 2-2, UserTestGen logic design can be divided to Command block (1) and Data block (2).

Command

The command request is controlled by the state machine which has eight states, as shown in Figure 2-3.

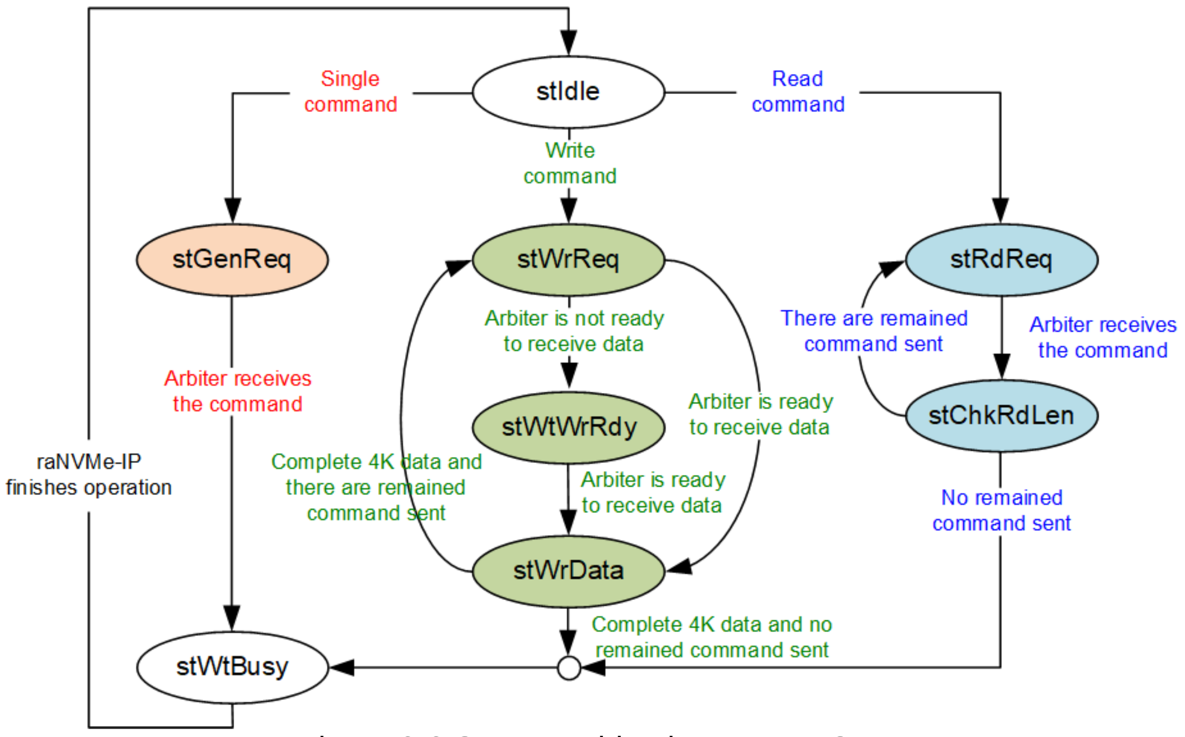
stIdle
Single command
Read command
Write command
stGenReq
stWrReq
stRdReq
Arbiter is not ready to receive data
There are remained command sent
Arbiter receives the command
Arbiter receives the command
stWtWrRdy
Arbiter is ready to receive data
stChkRdLen
raNVMe-IP finishes operation
Complete 4K data and there are remained command sent
Arbiter is ready to receive data
No remained command sent
stWrData
Complete 4K data and no remained command sent
stWtBusy

Figure 2-3 State machine in UserTestGen

(1) stIdle: This state is designed to wait until the user sends the command request by asserting UsrCmdValid to '1'. After that, it changes to stGenReq, WrReq or stRdReq when the command value is Single command, Write command (UsrCmd=010b) or Read command (UsrCmd=011b) respectively.
(2) stGenReq: This state is applied to send the command request to the Arbiter by asserting ArbCValid to '1'. After the Arbiter accepts the command request (ArbCReady='1'), it changes to stWtBusy.
(3) stWtBusy: This state is designed to wait until the Arbiter completes the command operation by de-asserting ArbBusy to '0'. After finishing, the state changes to stIdle.
(4) stWrReq: This state is applied to send the write command request to the Arbiter by asserting ArbCValid to '1'. After the Arbiter accepts the command request by asserting ArbCReady to '1', the state changes to the next state. The next state is stWrData if the Arbiter is ready to receive the data (ArbwReady='1'). Otherwise, it changes to stWtWrRdy for waiting the Arbiter ready for receiving the data.
(5) stWtWrRdy: This state is designed to wait until the Arbiter is ready to receive the data by asserting ArbwReady to '1'. After that, it changes to stWrData to start sending data.
(6) stWrData: This state is stayed for 256 cycles to send 4KB Write data to the Arbiter. After finishing the operation, it changes to stWtBusy if there is no remained command for sending the request. Otherwise, it changes to stWrReq to send the next Write command request.
(7) stRdReq: This state is designed to send the read command request by asserting ArbCValid to '1'. After the Arbiter accepts the command request by asserting ArbCReady to '1', the state changes to stChkRdLen.
(8) stChkRdLen: The state checks if there is remained the command request for sending to the Arbiter. If all command is not sent, it changes to stRdReq to send the next command. Otherwise, it changes to stWtBusy to wait until the Arbiter completes the operation.

As shown in Block (1), ArbCValid is asserted when the state is equal to stGenReq, stWrReq or stRdReq for sending Single command, Write command or Read command respectively. It is de-asserted to '0' after the Arbiter accepts the command. UsrBusy is asserted to '1' when the new command from the user is received and de-asserted to '0' after the state returns to stIdle.

In Block (1), there are three counters for counting transfer length (Len Counter), address (Addr Counter) and completed command (Cmd Counter). Len Counter is down-counter to check remained transfer length for transferring with the Arbiter. Addr Counter is up-counter to send the next 4 KB address after finishing each command for sequential access. Cmd counter is up-counter to show total completed commands to the user for checking the progress of Write/Read operation.

Data
As shown in Block (2), the control signal for transferring the data is the valid signal which is asserted to '1' when the data is valid. 4 KB data, the data size of one Write/Read command, is continuously transferred between UserTestGen and Arbiter by asserting the valid signal for 256 clock cycles.

The data counter is the up-counter to count total cycles for asserting ArbwValid to '1' when running Write command. Also, it is applied to count total received data when running Read command. Therefore, the valid signals, ArbwValid and ArbrValid, are applied to be the counter enable of Data Counter. The output of the data counter, rDataCnt, is also applied to be a part of the test data for sending and verifying process.

Test data is created to be Write data, ArbwData, when running Write command or expected data for comparing with received data, ArbrData, when running Read command. The 4KB data for one Write/Read command consists of 64-bit header data and the test pattern, selected by PattSel.

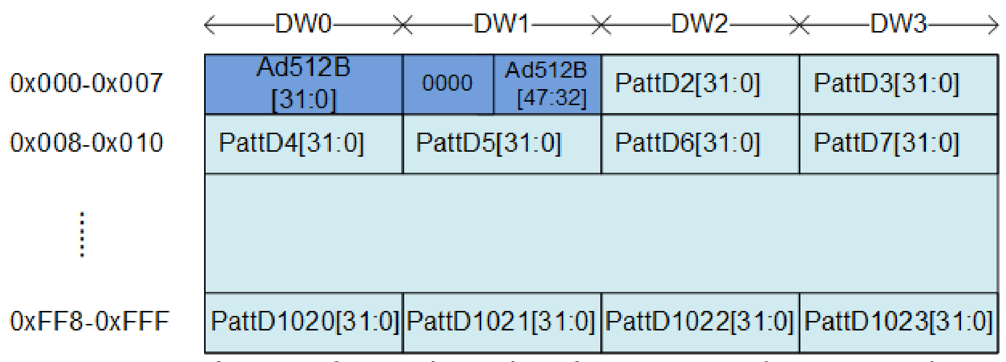

Figure 2-4 Test pattern format of 4096-byte data for Increment/Decrement/LFSR pattern

As shown in Figure 2-4, 4KB data consists of 64-bit header in DW#0 and DW#1 and the test data in DW#2 – DW#1023. 64-bit header is assigned by a physical address of SSD which stores this 4KB data. The header is designed by AddrCnt (Header). It is up-counter, similar to Addr Counter in Command block (1), but the value is increased after finishing transferring each 4KB data. While the remaining data is the test pattern which may be 32-bit incremental data, 32-bit decremental data or 32-bit LFSR counter, selected by PattSel. The 32-bit incremental data is designed by combining current address, rHdrAddr, with the lower bit of data counter, rDataCnt[7:0]. The decremental data is designed by using NOT logic to the incremental data. The equation of 32-bit LFSR data is x^31 + x^21 + x + 1. Four 32-bit LFSR data must be generated within one cycle to create 128-bit data. Therefore, the logic uses look-ahead style to generate four LFSR data in the same clock.

In addition, the user can select test pattern to be all zero or all one data to show the best performance of some SSDs which has data compression in SSD controller. When the pattern is all zero or all one, there is no 64-bit header inserted to 4 KB data.

When running Read command, PattFail is asserted to '1' if the received data, ArbrData, is not equal to the expected data. Also, the signals to show information of the first failure data, i.e., failure data position (FailNo), expected data (ExpPatt) and received data (RdPatt) are latched for user reading.

Figure 2-5 shows timing diagram of UserTestGen when the user sends Single command - Identify, SMART, Flush and Shutdown.

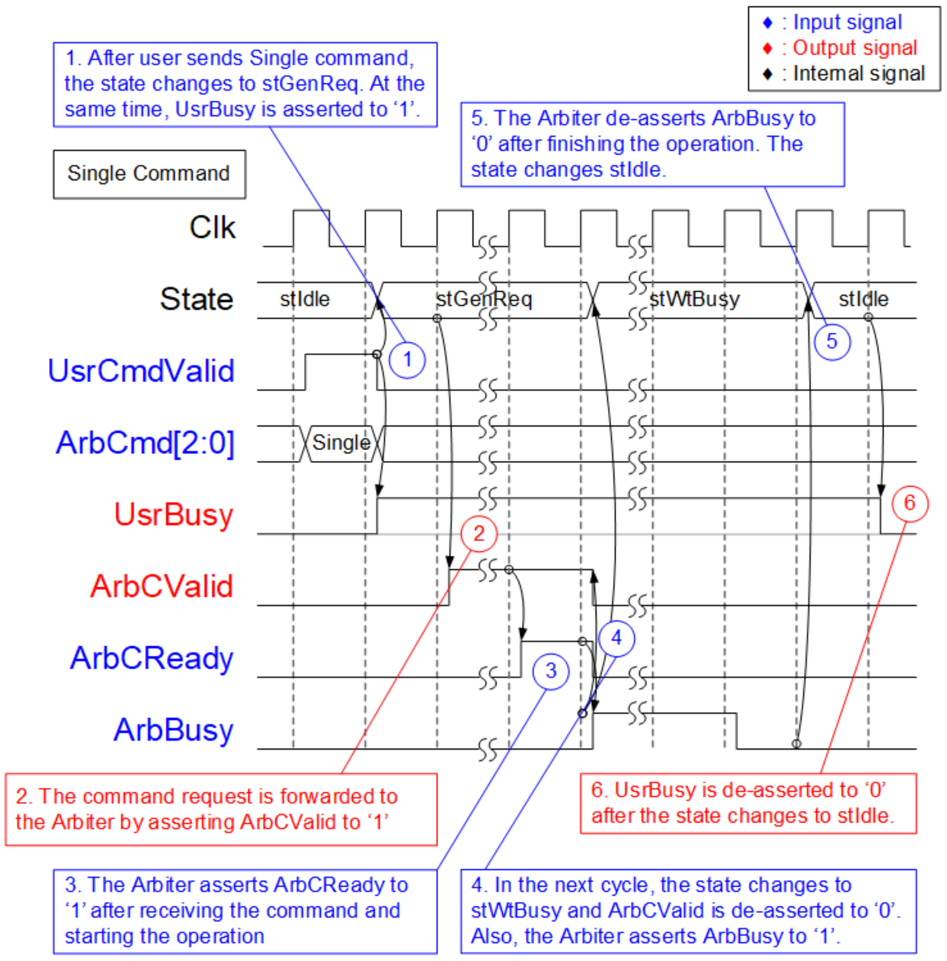
♦ : Input signal
♦ : Output signal
♦ : Internal signal
1. After user sends Single command, the state changes to stGenReq. At the same time, UsrBusy is asserted to '1'.
5. The Arbiter de-asserts ArbBusy to '0' after finishing the operation. The state changes stIdle.
Single Command
Clk
State
stIdle
stGenReq
stWtBusy
stIdle
UsrCmdValid
ArbCmd[2:0]
Single
UsrBusy
ArbCValid
ArbCReady
ArbBusy
2. The command request is forwarded to the Arbiter by asserting ArbCValid to '1'
6. UsrBusy is de-asserted to '0' after the state changes to stIdle.
3. The Arbiter asserts ArbCReady to '1' after receiving the command and starting the operation
4. In the next cycle, the state changes to stWtBusy and ArbCValid is de-asserted to '0'. Also, the Arbiter asserts ArbBusy to '1'.

Figure 2-5 Timing diagram of UserTestGen when running Single command

(1) Before sending the new command, UsrBusy must be equal to '0' which is found when the state is in stIdle. The user asserts UsrCmdValid to '1' with assigning the command on ArbCmd. After that, the state changes to stGenReq.
(2) In stGenReq, ArbCValid is asserted to '1' to send the single command request to the Arbiter. The signals must hold the value until the Arbiter accepts the command.
(3) When the Arbiter is ready to operate the command, it asserts ArbCReady to '1'. After that, the Arbiter runs the operation of the command.
(4) In the next cycle, the state changes to stWtBusy to wait until the Arbiter completes the operation. During operating, the Arbiter asserts ArbBusy '1'.
(5) After the command is operated completely, ArbBusy is de-asserted to '0'. In the next cycle, the state changes to stIdle.
(6) UsrBusy is de-asserted to '0' to show Idle status and the user can send the new command.

Figure 2-6 shows timing diagram of UserTestGen when the user sends Write command.

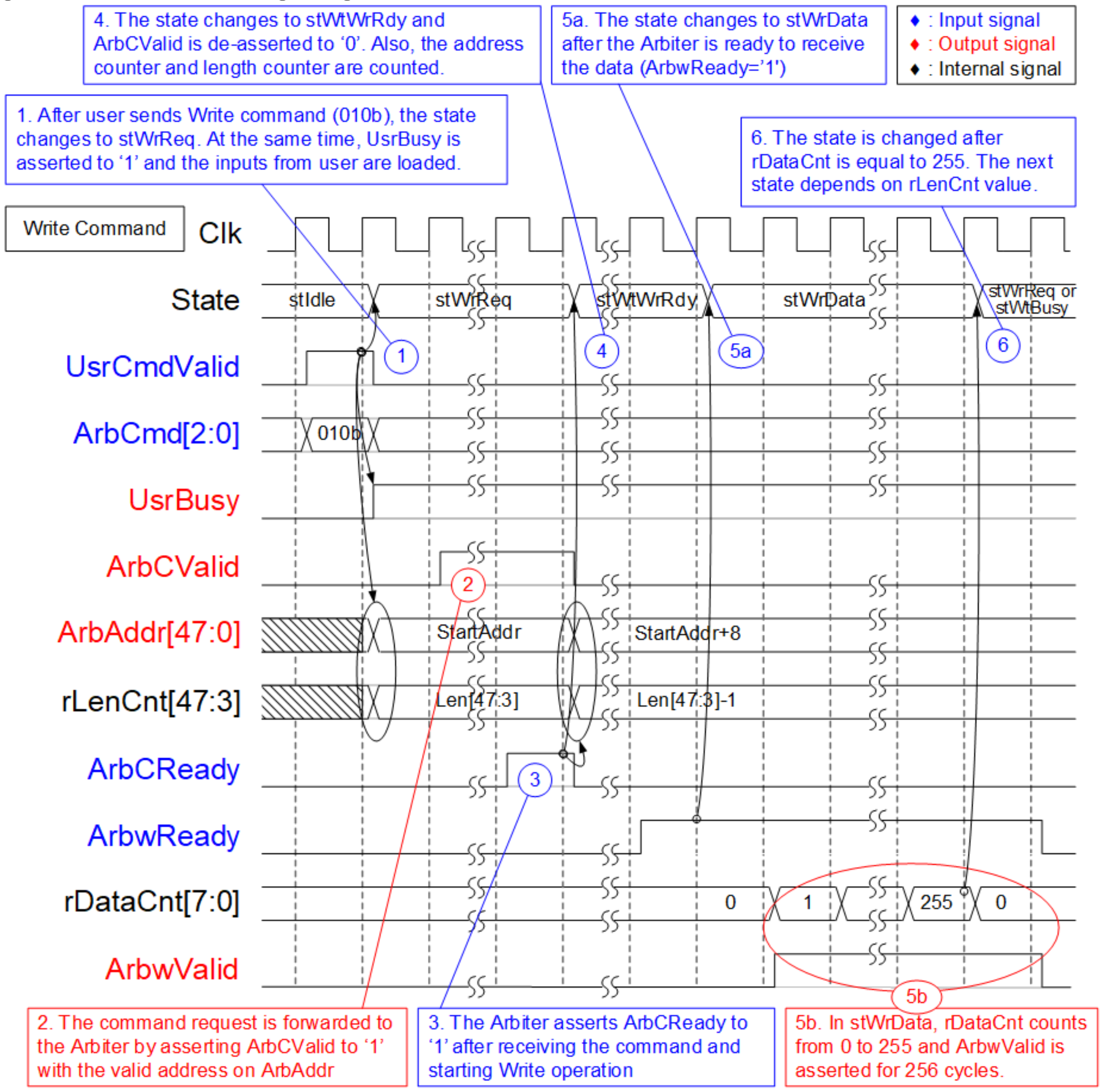
4. The state changes to stWtWrRdy and ArbCValid is de-asserted to '0'. Also, the address counter and length counter are counted.
5a. The state changes to stWrData after the Arbiter is ready to receive the data (ArbwReady='1')
♦ : Input signal
♦ : Output signal
♦ : Internal signal
1. After user sends Write command (010b), the state changes to stWrReq. At the same time, UsrBusy is asserted to '1' and the inputs from user are loaded.
6. The state is changed after rDataCnt is equal to 255. The next state depends on rLenCnt value.
Write Command
Clk
State
stIdle
stWrReq
stWtWrRdy
stWrData
stWrReq or stWtBusy
UsrCmdValid
ArbCmd[2:0]
010b
UsrBusy
ArbCValid
ArbAddr[47:0]
StartAddr
StartAddr+8
rLenCnt[47:3]
Len[47:3]
Len[47:3]-1
ArbCReady
ArbwReady
rDataCnt[7:0]
0
1
255
0
ArbwValid
1
2
3
4
5a
5b
6
2. The command request is forwarded to the Arbiter by asserting ArbCValid to '1' with the valid address on ArbAddr
3. The Arbiter asserts ArbCReady to '1' after receiving the command and starting Write operation
5b. In stWrData, rDataCnt counts from 0 to 255 and ArbwValid is asserted for 256 cycles.

Figure 2-6 Timing diagram of UserTestGen when running Write command

(1) When the Write command (ArbCmd=010b) is sent by the user, the state changes to stWrReq. Also, UsrBusy is asserted to '1' and the inputs from user, i.e., UsrStartAddr and UsrLen are loaded to be ArbAddr and rLenCnt respectively.
(2) In stWrReq, the write command is requested to the Arbiter by asserting ArbCValid to '1' with the valid ArbAddr. After that, the state waits until the request is accepted by the Arbiter.
(3) The Arbiter is ready for processing the Write command from the user logic by asserting ArbCReady to '1'.
(4) The state changes to stWtWrRdy to wait until the Arbiter is ready to receive the data by asserting ArbwReady to '1'. Also, the address (ArbAddr) and the remained length (rLenCnt) are counted for the next command.
(5) The state changes to stWrData for sending 256 data or 4 KB data. rDataCnt is counted from 0 to 255 in this state. The state is changed when rDataCnt is equal to 255.
(6) If there is remained data which must be transferred, the state changes to stWrReq to send the next Write command request. Otherwise, the state changes to stWtBusy to wait until the Write command is finished (ArbBusy='0'), similar to Single command.

Figure 2-7 shows timing diagram of UserTestGen when the user sends Read command.

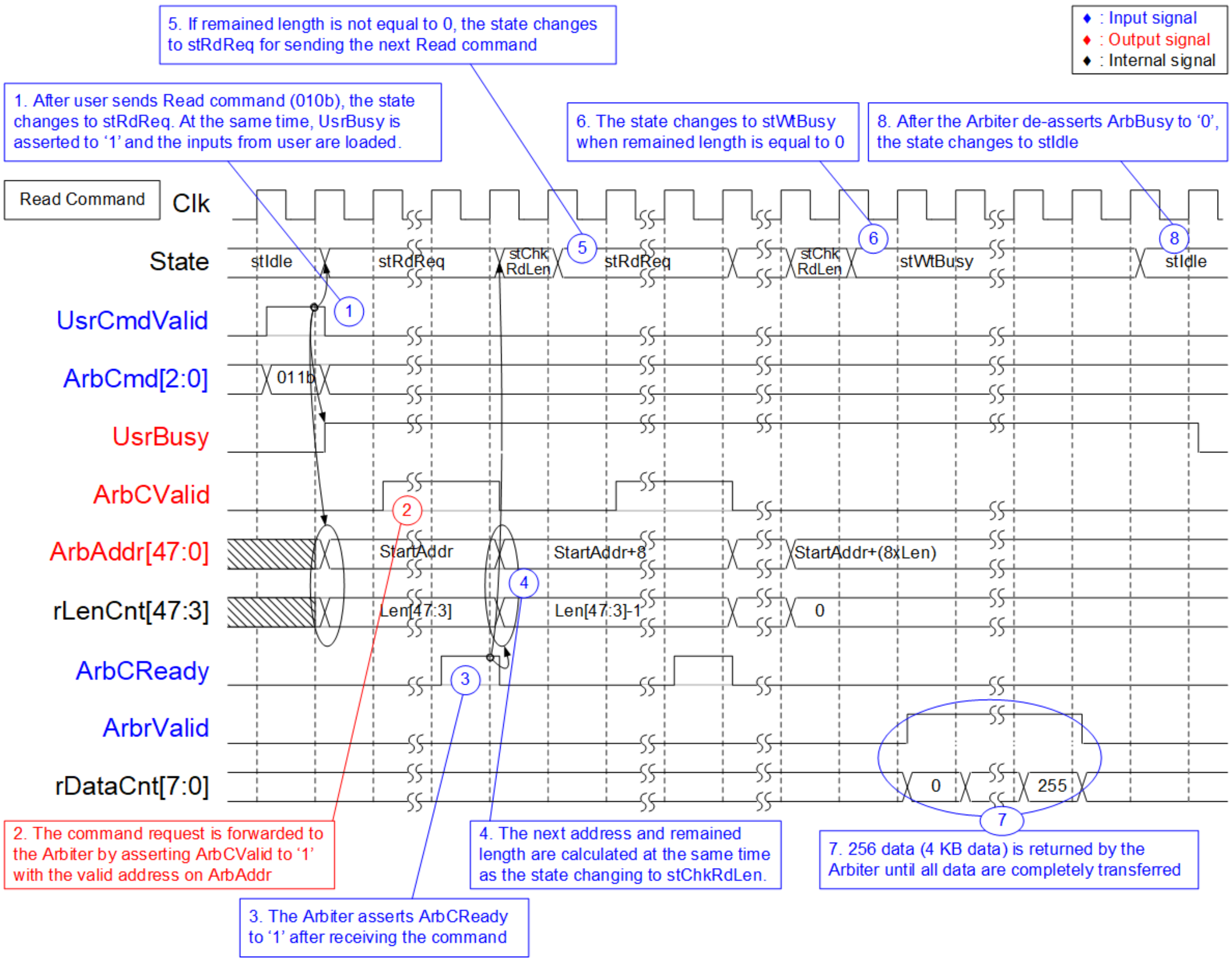
♦ : Input signal
♦ : Output signal
♦ : Internal signal
Read Command
Clk
State
UsrCmdValid
ArbCmd[2:0]
UsrBusy
ArbCValid
ArbAddr[47:0]
rLenCnt[47:3]
ArbCReady
ArbrValid
rDataCnt[7:0]
stIdle
stRdReq
stChk RdLen
stWtBusy
011b
StartAddr
StartAddr+8
StartAddr+(8xLen)
Len[47:3]
Len[47:3]-1
0
255
1. After user sends Read command (010b), the state changes to stRdReq. At the same time, UsrBusy is asserted to '1' and the inputs from user are loaded.
2. The command request is forwarded to the Arbiter by asserting ArbCValid to '1' with the valid address on ArbAddr
3. The Arbiter asserts ArbCReady to '1' after receiving the command
4. The next address and remained length are calculated at the same time as the state changing to stChkRdLen.
5. If remained length is not equal to 0, the state changes to stRdReq for sending the next Read command
6. The state changes to stWtBusy when remained length is equal to 0
7. 256 data (4 KB data) is returned by the Arbiter until all data are completely transferred
8. After the Arbiter de-asserts ArbBusy to '0', the state changes to stIdle

Figure 2-7 Timing diagram of UserTestGen when running Read command

(1) When the Read command (ArbCmd=011b) is sent by the user, the state changes to stRdReq. Also, UsrBusy is asserted to '1' and the inputs from user, i.e., UsrStartAddr and UsrLen are loaded to be ArbAddr and rLenCnt respectively.
(2) In stRdReq, the read command is requested to the Arbiter by asserting ArbCValid to '1' with the valid ArbAddr. After that, the state waits until the request is accepted by the Arbiter.
(3) The Arbiter is ready for processing the Read command from the user logic by asserting ArbCReady to '1'.
(4) The state changes to stChkRdLen to check remained length, rLenCnt. The next state is stRdReq or stWtBusy. At the same time, the address (ArbAddr) and the remained length (rLenCnt) are counted for the next command.
(5) If remained length is not equal to 0, the state changes to stRdReq to send the next Read command.
(6) If remained length is equal to 0, the state changes to stWtBusy to wait until the Arbiter completes the operation and de-asserts ArbBusy to '0'.
(7) 256 data or 4 KB data is returned from the Arbiter after receiving each Read command. After the last 4 KB data is returned, ArbBusy is de-asserted to '0'.
(8) The state returns to stIdle and UsrBusy is de-asserted to '0' in the next cycle.

2.2 raArbiter

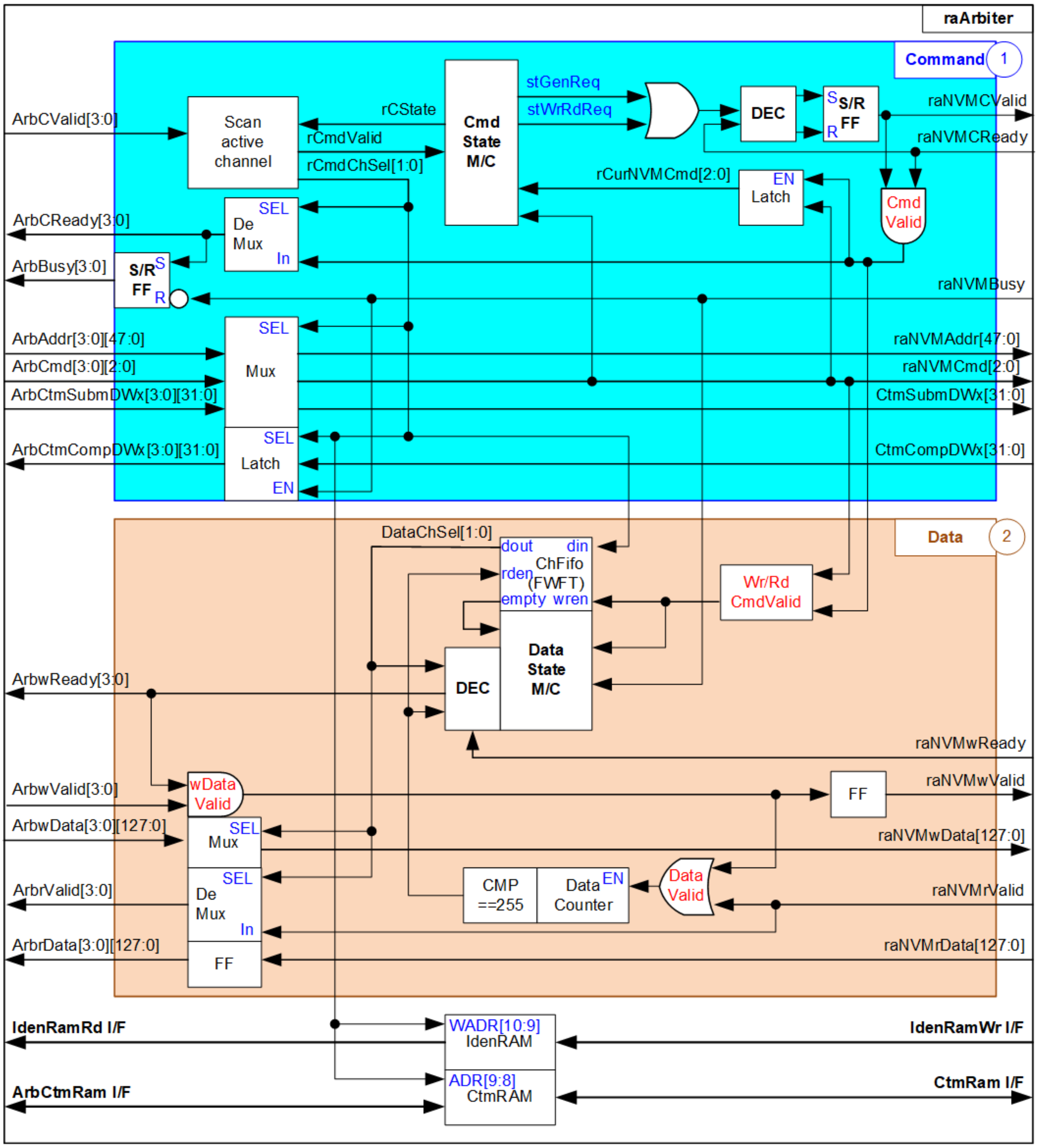

Figure 2-8 raArbiter block diagram

raArbiter is the module to handle four users for accessing the same SSD via one raNVMe-IP. Similar to UserTestGen, the logic inside raArbiter is split into two parts – Command and Data. Besides, it includes IdenRAM and CtmRAM to store the data output from Identify and SMART command for all users. For simple architecture of the arbiter, the user must send the command request by asserting ArbCValid to '1' before starting transferring the data. ChFIFO is designed to store the user number that the Arbiter accepts the command request when the command is Write or Read command. The data block reads the user number from ChFIFO to select the active user for transferring data of Write or Read command with raNVMe-IP. Consequently, the order for transferring data in Write or Read command is similar to the order of the command request. The details of Command block (1) and Data block (2) are described as follows.

Command
As shown in Command block of Figure 2-8, the key signal of raArbiter is rCmdChSel, the output of Scan active channel block. This block scans the command request, ArbCValid, from each channel and selects the active user, assigned to rCmdChSel, with asserting the command request (rCmdValid) to Cmd State machine. If all users send the request at the same time, the active user will change from 0 to 3. It uses 2-bit counter inside Scan active channel block for rotating the priority of each user. After that, the inputs from the active user – Address (ArbAddr), Command (ArbCmd) and Custom parameters (ArbCtmSubmDW) are forwarded to raNVMe-IP. raNVMCValid is asserted to '1' by Cmd State machine and then raNVMe-IP accepts the request by asserting raNVMCReady to '1'. Also, raNVMCReady is forwarded to be ArbCReady of the active channel. At the same time, ArbBusy of the active channel is asserted to '1' until raNVMe-IP completes all command operation by de-asserting raNVMBusy to '0'.

The details of Cmd State machine are shown in Figure 2-9.

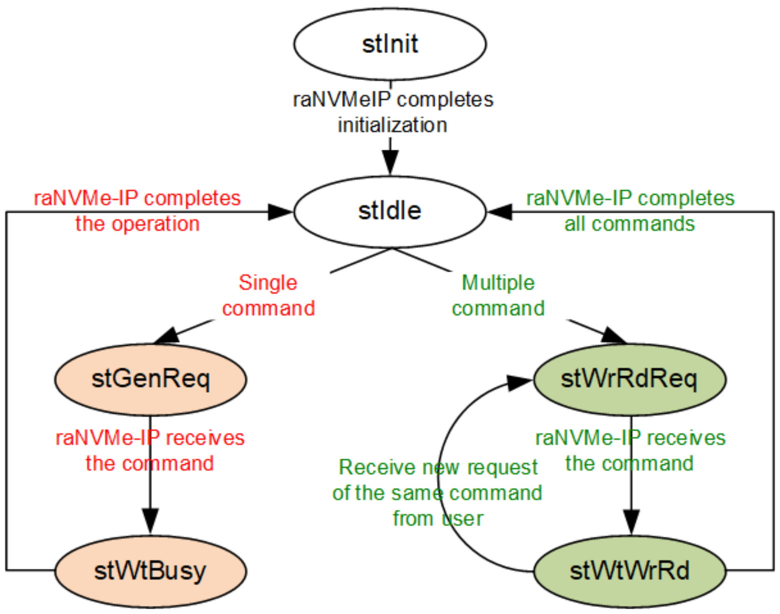
stInit
raNVMeIP completes initialization
stIdle
raNVMe-IP completes the operation
raNVMe-IP completes all commands
Single command
Multiple command
stGenReq
stWrRdReq
raNVMe-IP receives the command
Receive new request of the same command from user
raNVMe-IP receives the command
stWtBusy
stWtWrRd

Figure 2-9 State machine of raArbiter's command block

There are six state machines for controlling the command request sent to raNVMe-IP.

(1) stInit: This state is the first state after power-on the system. It waits until raNVMe-IP completes the initialization process (raNVMBusy='0'). After that, it changes to stIdle.

(2) stIdle: This state waits until Scan active channel block detects the new command request from some users and asserts rCmdValid to '1'. After that, the command value, raNVMCmd, is read. If the command is Single command, the next state is stGenReq. Otherwise, the next state is stWrRdReq.

(3) stGenReq: This state is designed to generate the command request to raNVMe-IP by asserting raNVMCValid to '1'. After raNVMe-IP accepts the request by asserting raNVMCReady to '1', it changes to the next state, stWtBusy.

(4) stWtBusy: This state waits until raNVMe-IP completes to operate the command, monitored by raNVMBusy='0'. After finishing the operation, it goes back to stIdle for receiving the next command from user.

(5) stWrRdReq: This state is designed to generate the Write/Read command request to raNVMe-IP by asserting raNVMCValid to '1'. After raNVMe-IP accepts the request by asserting raNVMCReady to '1', it changes to the next state, stWtWrRd.

(6) stWtWrRd: In multiple mode, if new command is requested from the user and the command is the same command, it changes to stWrRdReq to send more request to raNVMe-IP. Otherwise, it waits until raNVMe-IP completes to operate all commands and de-asserts raNVMBusy to '0'. After that, the state goes back to stIdle. Consequently, if the new command requested by the user is different command, the new command is processed after the state goes back to stIdle.

Timing diagram of Command interface when running Single command and Multiple command are shown in Figure 2-10 and Figure 2-11 respectively.

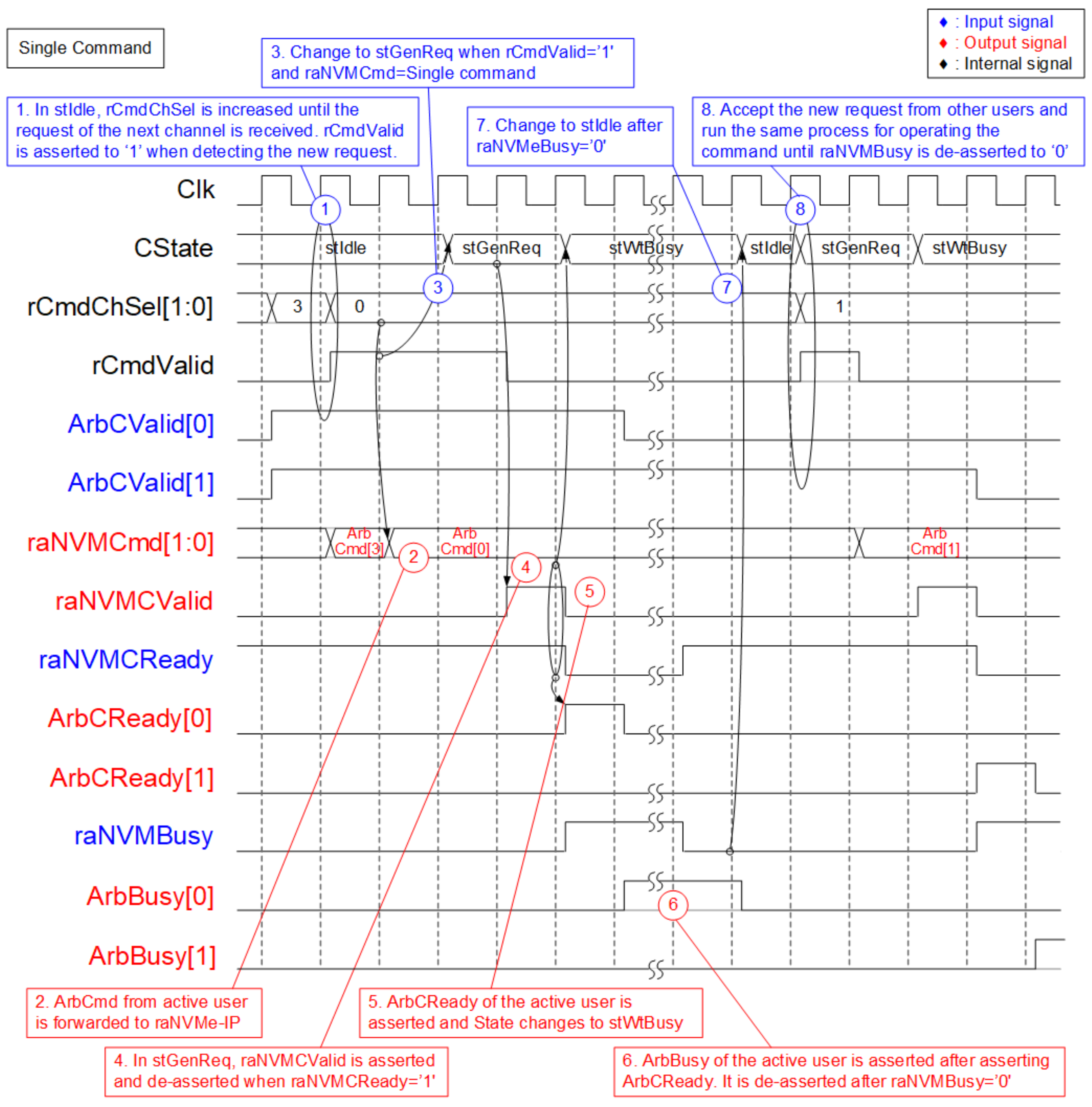
Single Command
♦ : Input signal
♦ : Output signal
♦ : Internal signal
1. In stIdle, rCmdChSel is increased until the request of the next channel is received. rCmdValid is asserted to '1' when detecting the new request.
3. Change to stGenReq when rCmdValid='1' and raNVMCmd=Single command
7. Change to stIdle after raNVMeBusy='0'
8. Accept the new request from other users and run the same process for operating the command until raNVMBusy is de-asserted to '0'
Clk
CState
stIdle
stGenReq
stWtBusy
rCmdChSel[1:0]
3
0
1
rCmdValid
ArbCValid[0]
ArbCValid[1]
raNVMCmd[1:0]
Arb Cmd[3]
Arb Cmd[0]
Arb Cmd[1]
raNVMCValid
raNVMCReady
ArbCReady[0]
ArbCReady[1]
raNVMBusy
ArbBusy[0]
ArbBusy[1]
2. ArbCmd from active user is forwarded to raNVMe-IP
4. In stGenReq, raNVMCValid is asserted and de-asserted when raNVMCReady='1'
5. ArbCReady of the active user is asserted and State changes to stWtBusy
6. ArbBusy of the active user is asserted after asserting ArbCReady. It is de-asserted after raNVMBusy='0'

Figure 2-10 Timing diagram of raArbiter's Command I/F when running Single command

(1) In stIdle, rCmdChSel is increased to scan ArbCValid of each channel until detecting some bits are asserted. In Figure 2-10, when ArbCValid[0]='1' and rCmdChSel=3 (changes to 0 in the next cycle), it will accept the request of channel#0 and start the operation. After that, rCmdChSel changes to 0 and holds the value until the command is completely processed. At the same time, rCmdValid is asserted to '1' when the request of some channels is accepted.

(2) raNVMCmd and other parameters such as raNVMAddr receive the inputs from the active channel, controlled by rCmdChSel.

(3) Two clock cycles after rCmdValid asserted to '1', the state reads raNVMCmd to check the command type. If the command is Single command, it changes to stGenReq.

(4) raNVMCValid is asserted to '1' to send the command request to raNVMe-IP. The signal is latched to '1' until raNVMe-IP accepts the request by asserting raNVMCReady to '1'.

(5) ArbCReady of the active user is asserted to '1' for one cycle to be acknowledge signal. At the same time, the state changes to stWtBusy.

(6) ArbBusy of the active user is asserted to '1' during operating the command. It is de-asserted to '0' after raNVMe-IP completes the operation (raNVMBusy='0').

(7) After raNVMBusy is de-asserted to '0', the state changes to stIdle for receiving the new command request from the user.

(8) Repeat step (1) for scanning the next channel. In Figure 2-10, user#1 sends the request when rCmdChSel=0. Consequently, rCmdChSel changes to 1 and rCmdValid is asserted to '1' for starting operating command from channel#1. After that, repeat step (2) – (7) if the command of user#1 is Single command.

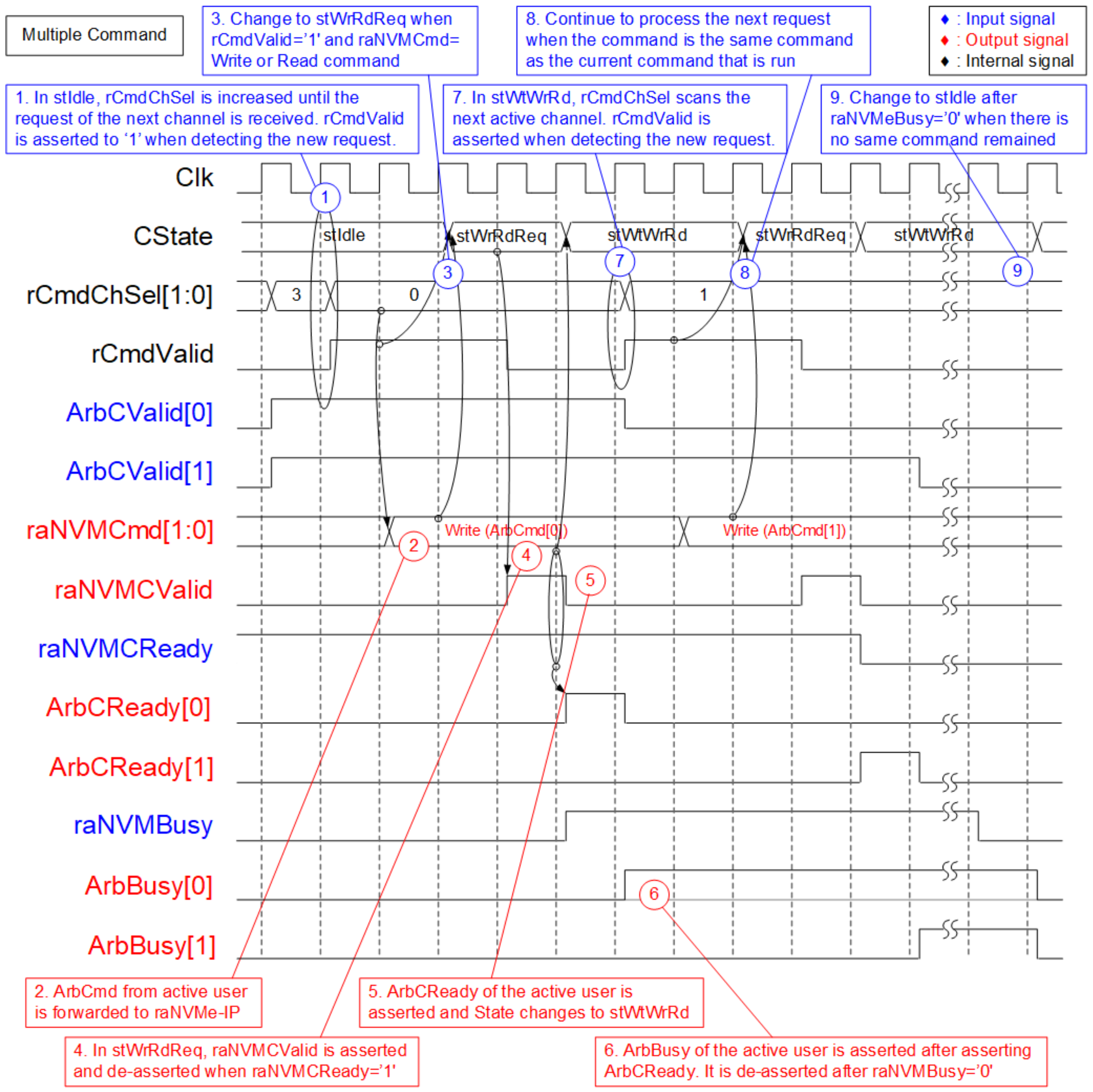
Multiple Command
3. Change to stWrRdReq when rCmdValid='1' and raNVMCmd= Write or Read command
8. Continue to process the next request when the command is the same command as the current command that is run
♦ : Input signal
♦ : Output signal
♦ : Internal signal
1. In stIdle, rCmdChSel is increased until the request of the next channel is received. rCmdValid is asserted to '1' when detecting the new request.
7. In stWtWrRd, rCmdChSel scans the next active channel. rCmdValid is asserted when detecting the new request.
9. Change to stIdle after raNVMeBusy='0' when there is no same command remained
Clk
CState
stIdle
stWrRdReq
stWtWrRd
stWrRdReq
stWtWrRd
rCmdChSel[1:0]
3
0
1
rCmdValid
ArbCValid[0]
ArbCValid[1]
raNVMCmd[1:0]
Write (ArbCmd[0])
Write (ArbCmd[1])
raNVMCValid
raNVMCReady
ArbCReady[0]
ArbCReady[1]
raNVMBusy
ArbBusy[0]
ArbBusy[1]
1
2
3
4
5
6
7
8
9
2. ArbCmd from active user is forwarded to raNVMe-IP
5. ArbCReady of the active user is asserted and State changes to stWtWrRd
4. In stWrRdReq, raNVMCValid is asserted and de-asserted when raNVMCReady='1'
6. ArbBusy of the active user is asserted after asserting ArbCReady. It is de-asserted after raNVMBusy='0'

Figure 2-11 Timing diagram of raArbiter's Command I/F when running Multiple command

Step (1) and (2) are similar to Single command mode.

(3) Two clock cycles after rCmdValid asserted to ‘1’, the state reads raNVMCmd to check the command type. If the command is Multiple command, it changes to stWrRdReq. In Figure 2-11, assume that the command is Write command.

(4) raNVMCValid is asserted to ‘1’ to send the command request to raNVMe-IP. The signal is latched to ‘1’ until raNVMe-IP accepts the request by asserting raNVMCReady to ‘1’.

(5) ArbCReady of the active user is asserted to ‘1’ for one cycle to be acknowledge signal. At the same time, the state changes to stWtWrRd.

(6) ArbBusy of the active user is asserted to ‘1’ during operating the command. It is de-asserted to ‘0’ after raNVMe-IP completes the operation (raNVMBusy=’0’).

(7) During waiting the command operating, rCmdChSel is increased to scan the new command request. If the new request is detected, rCmdValid is asserted to ‘1’.

(8) It changes to stWrRdReq for operating the next command if the new command from the new active user is the same command as the current command. As shown in Figure 2-11, the new command from user#1 is Write command which is the same as current command from user#0. The new command is requested to raNVMe-IP by asserting raNVMCValid to ‘1’ with forwarding the parameters from the new user.

(9) If there is no new command which is the same command (Write command), the state waits until raNVMBusy is de-asserted to ‘0’. After that, it changes to stIdle to operate the new command from user.

Data

As shown in Data block of Figure 2-8, rCmdChSel from Command block is stored to ChFIFO. When data interface of the user and raNVMe-IP are ready for transferring, Data State machine reads data from ChFIFO, DataChSel. DataChSel defines the active user of data path for transferring with raNVMe-IP. ChFIFO is FWFT FIFO, so DataChSel is valid to read when the FIFO is not empty. Data State machine asserts read enable after DataChSel is completely used.

Data counter is designed to count total number of write data or read data when running Write command or Read command. During transferring data between the active user and raNVMe-IP, DataChSel must not change the value. Consequently, the data from active user, ArbwData, is forwarded to raNVMwData in Write command. On the contrary, the read data valid of the active user is asserted by raNVMrValid (read data valid of raNVMe-IP) in Read command. 256 data (4Kbyte) are transferred for each data request.

The details of Data State machine are shown in Figure 2-12.

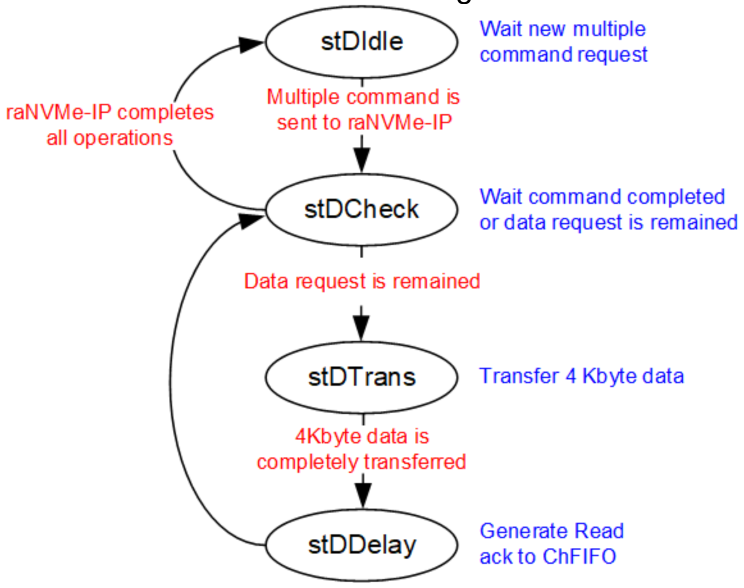
stDIdle
Wait new multiple command request
raNVMe-IP completes all operations
Multiple command is sent to raNVMe-IP
stDCheck
Wait command completed or data request is remained
Data request is remained
stDTrans
Transfer 4 Kbyte data
4Kbyte data is completely transferred
stDDelay
Generate Read ack to ChFIFO

Figure 2-12 State machine of raArbiter's data block

Four state machines are designed for transferring data between the active user and raNVMe-IP when operating Multiple command (Write or Read command). The order for transferring the data is stored in ChFIFO, written by Command block to have the same order as the command request.

(1) stDIdle: This state waits until the multiple command is sent to raNVMe-IP by asserting raNVMCValid to '1'. raNVMCmd must be Multiple command. After that, it changes to next state, stDCheck.
(2) stDCheck: This state is designed to check if there is remained data request in ChFIFO. If ChFIFO is not empty, it changes to the next state, stDTrans. Otherwise, it stays in this state until raNVMe-IP completes the operation of all commands and de-asserts raNVMBusy to '0'. After that, it will change to stDIdle.
(3) stDTrans: This state is designed to wait until 4Kbyte data is completely transferred between the active user and raNVMe-IP. rDataCnt, the output of data counter, is monitored to read total number of transferred data. After completing 4Kbyte data transferring by running this state for 256 cycles via 128-bit bus, it changes to sDDelay.
(4) stDDelay: This state is designed to assert Read ack of ChFIFO after completing to use the read data of ChFIFO, DataChSel. After that, it changes to stDCheck to check remaining data in ChFIFO.

Timing diagram of Data interface when running Write command and Read command are shown in Figure 2-13 and Figure 2-14 respectively.

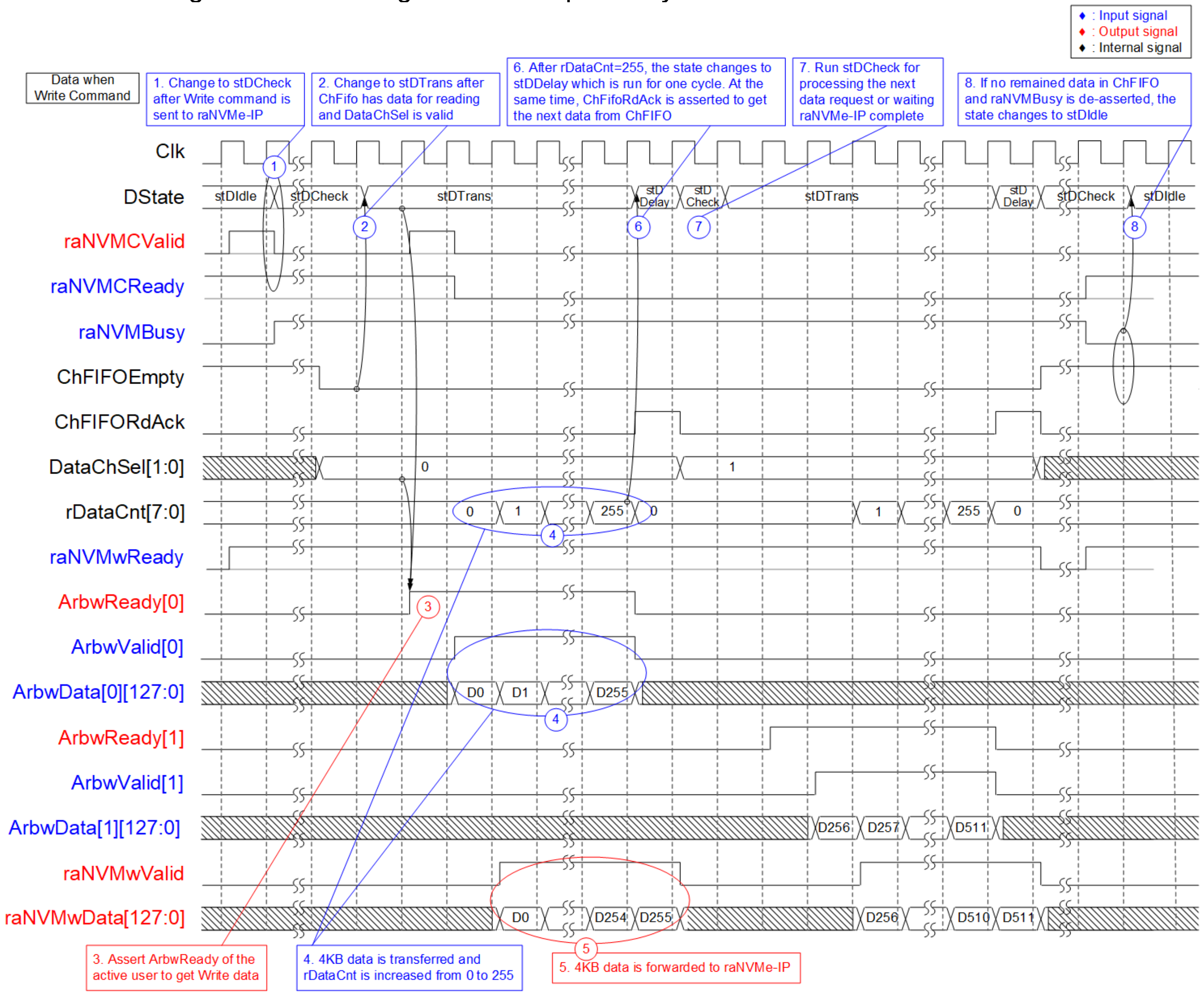
♦ : Input signal
♦ : Output signal
♦ : Internal signal
Data when Write Command
1. Change to stDCheck after Write command is sent to raNVMe-IP
2. Change to stDTrans after ChFifo has data for reading and DataChSel is valid
6. After rDataCnt=255, the state changes to stDDelay which is run for one cycle. At the same time, ChFifoRdAck is asserted to get the next data from ChFIFO
7. Run stDCheck for processing the next data request or waiting raNVMe-IP complete
8. If no remained data in ChFIFO and raNVMBusy is de-asserted, the state changes to stDIdle
Clk
DState
raNVMCValid
raNVMCReady
raNVMBusy
ChFIFOEmpty
ChFIFORdAck
DataChSel[1:0]
rDataCnt[7:0]
raNVMwReady
ArbwReady[0]
ArbwValid[0]
ArbwData[0][127:0]
ArbwReady[1]
ArbwValid[1]
ArbwData[1][127:0]
raNVMwValid
raNVMwData[127:0]
stDIdle
stDCheck
stDTrans
stD Delay
stD Check
D0
D1
D255
D256
D257
D511
D254
D510
3. Assert ArbwReady of the active user to get Write data
4. 4KB data is transferred and rDataCnt is increased from 0 to 255
5. 4KB data is forwarded to raNVMe-IP

Figure 2-13 Timing diagram of raArbiter's Data I/F when running Write command

(1) In stDIdle, it waits until Multiple command is sent to raNVMe-IP. In Figure 2-13, when raNVMCValid and raNVMCReady are asserted to '1' with setting raNVMCCmd to be Write command, the state changes to stDCheck to start transferring data from the user to raNVMe-IP. At the same time as sending Write command to raNVMe-IP, Command block writes the active user number to ChFIFO. Assume that user#0 and user#1 send Write command respectively.

(2) After ChFIFO is written, ChFIFOEmpty is de-asserted to '0' and the read data of ChFIFO, DataChSel, is valid. Next, the state changes to stDTrans.

(3) ArbwReady of the active channel (user#0), selected by DataChSel, is asserted to get 4KB write data from the active user.

(4) Next, the active user sends 4KB write data (ArbwData) with asserting ArbwValid for 256 cycles. rDataCnt is increased from 0 to 255 for counting total number of transferred data.

(5) The write data from user is forwarded to raNVMwData via data multiplexer, controlled by DataChSel. At the same time, raNVMwValid is asserted to '1' to be valid signal of write data.

(6) After completing 256 data, rDataCnt=255, the state changes to stDDelay which is run for one cycle.

(7) In stDDelay, ChFIFORdAck is asserted to '1' after completing to use the current read data. The next data from ChFIFO will be available if the data is available in ChFIFO. Next state is stDCheck for checking ChFIFO status. If ChFIFO has remained data (ChFIFOEmpty='0'), repeat step (2) to transfer the data of the next data request. Otherwise, it waits until raNVMe-IP completes the operation and goes to step (8).

(8) If no remained data in ChFIFO and raNVMBusy is de-asserted to '0', it returns to stDIdle to complete the operation.

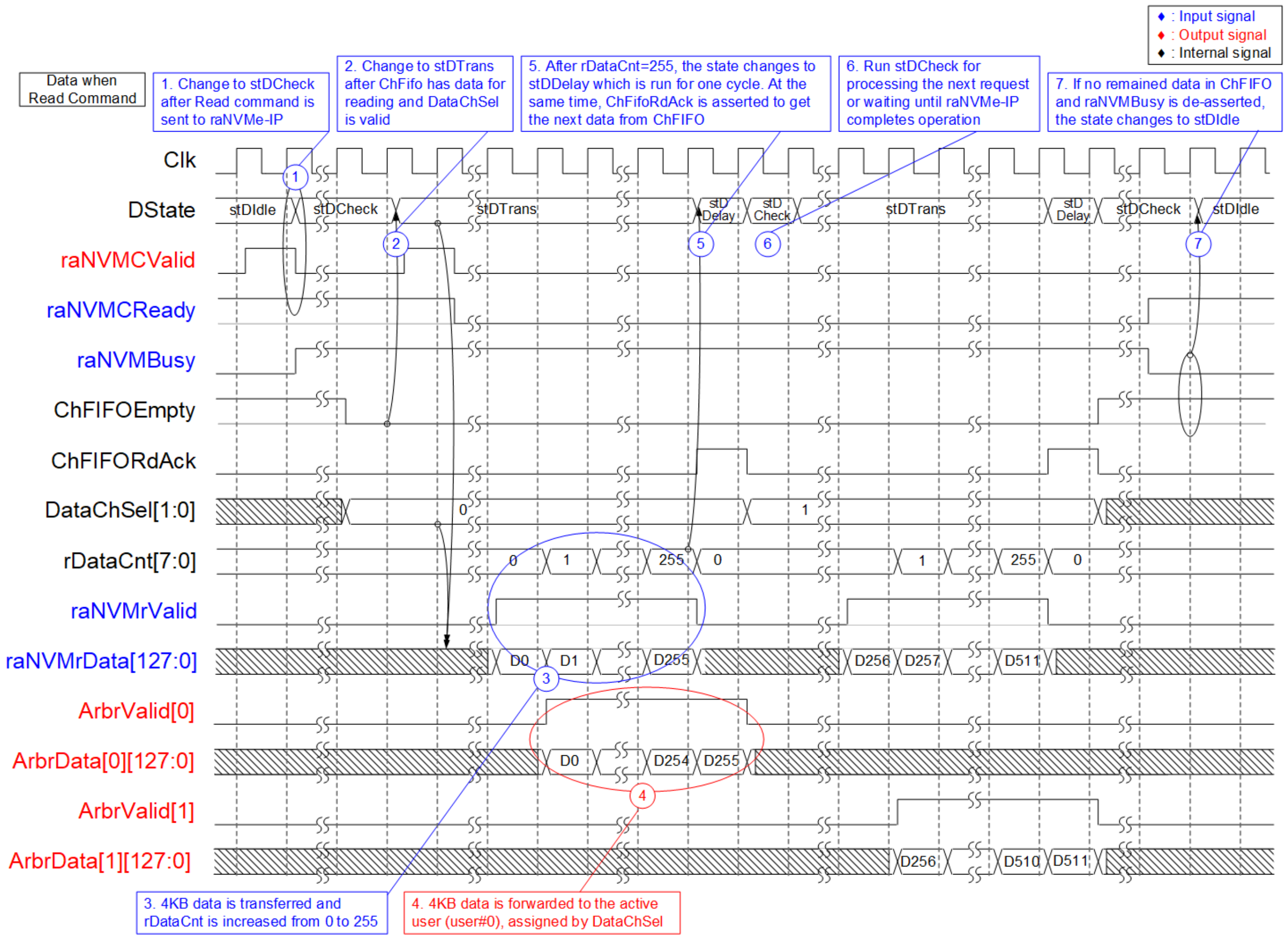
♦ : Input signal
♦ : Output signal
♦ : Internal signal
Data when Read Command
1. Change to stDCheck after Read command is sent to raNVMe-IP
2. Change to stDTrans after ChFifo has data for reading and DataChSel is valid
5. After rDataCnt=255, the state changes to stDDelay which is run for one cycle. At the same time, ChFifoRdAck is asserted to get the next data from ChFIFO
6. Run stDCheck for processing the next request or waiting until raNVMe-IP completes operation
7. If no remained data in ChFIFO and raNVMBusy is de-asserted, the state changes to stDIdle
Clk
DState
stDIdle
stDCheck
stDTrans
stD Delay
stD Check
stDTrans
stD Delay
stDCheck
stDIdle
raNVMCValid
raNVMCReady
raNVMBusy
ChFIFOEmpty
ChFIFORdAck
DataChSel[1:0]
0
1
rDataCnt[7:0]
0
1
255
0
1
255
0
raNVMrValid
raNVMrData[127:0]
D0
D1
D255
D256
D257
D511
ArbrValid[0]
ArbrData[0][127:0]
D0
D254
D255
ArbrValid[1]
ArbrData[1][127:0]
D256
D510
D511
3. 4KB data is transferred and rDataCnt is increased from 0 to 255
4. 4KB data is forwarded to the active user (user#0), assigned by DataChSel

Figure 2-14 Timing diagram of raArbiter's Data I/F when running Read command

Timing diagram of data interface when running Read command is almost similar to Write command.

Step (1) and (2) are similar to Write command.

(3) raNVMe-IP returns 4KB data by asserting raNVMrValid to '1' for 256 cycles with the valid data on raNVMrData. During transferring, rDataCnt is increased from 0 to 255.

(4) The read data from raNVMe-IP is forwarded to the active user, defined by DataChSel. raNVMrValid and raNVMrData are forwarded to the active user to be ArbrValid and ArbrData respectively. Consequently, ArbrValid of the active user (user#0) is asserted to '1' for 256 cycles with the valid data on ArbrData.

(5) After receiving the last data (rDataCnt=255), the state changes to stDDelay.

(6) Similar to step(7) of Write command, ChFIFORdAck is asserted to '1' in stDDelay and then returns to stDCheck in the next cycle. Step (2) – (6) are repeated if ChFIFO is not empty. Otherwise, continue to step (7) when raNVMe-IP completes the operation, raNVMBusy='0'.

(7) After raNVMBusy is de-asserted to '0', the state returns to stDIdle for processing the next command.

IdenRAM and CtmRAM

Two dual port RAMs, CtmRAM and IdenRAM, store the returned data from SMART command and Identify command respectively. Each RAM is split into four parts for storing data of usre#0 – user#3. rCmdChSel is applied to be 2-upper bits of the address for accessing CtmRAM and IdenRAM.

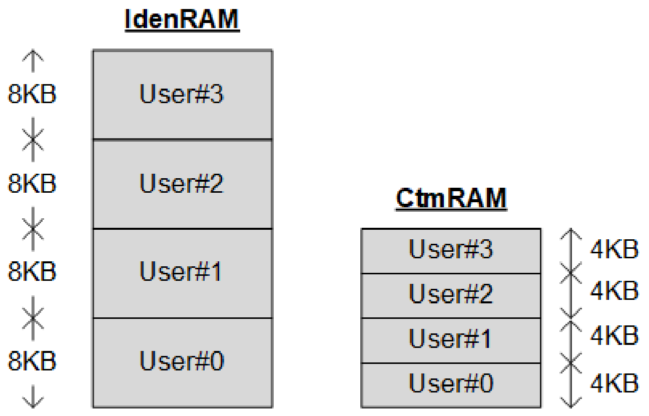

Figure 2-15 IdenRAM and CtmRAM

Identify command returns 8K-byte data, so IdenRAM has 32K-byte size to store data of four users. SMART command returns 512-byte data, but 4K-byte size of CtmRAM is reserved for each user. CtmRAM size in the reference design is finally equal to 16 Kbytes.

raNVMe-IP and LAxi2Reg have the different data bus size, 128-bit on raNVMe-IP but 32-bit on LAxi2Reg. Consequently, IdenRAM and CtmRAM have the different bus size for connecting with two modules. Also, raNVMe-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, so the small logic to convert double word enable to be write byte enable is designed as shown in Figure 2-16.

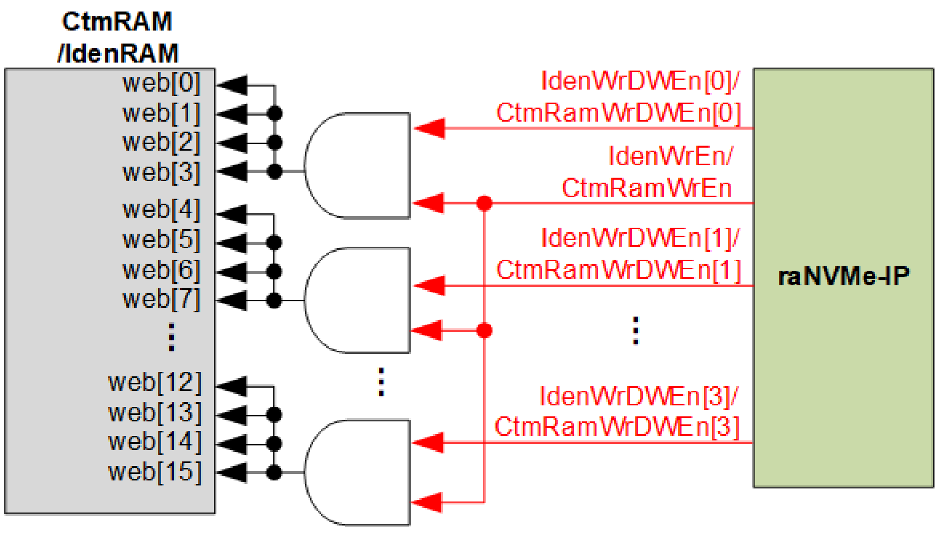

Figure 2-16 Byte write enable conversion logic

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.

2.3 NVMe

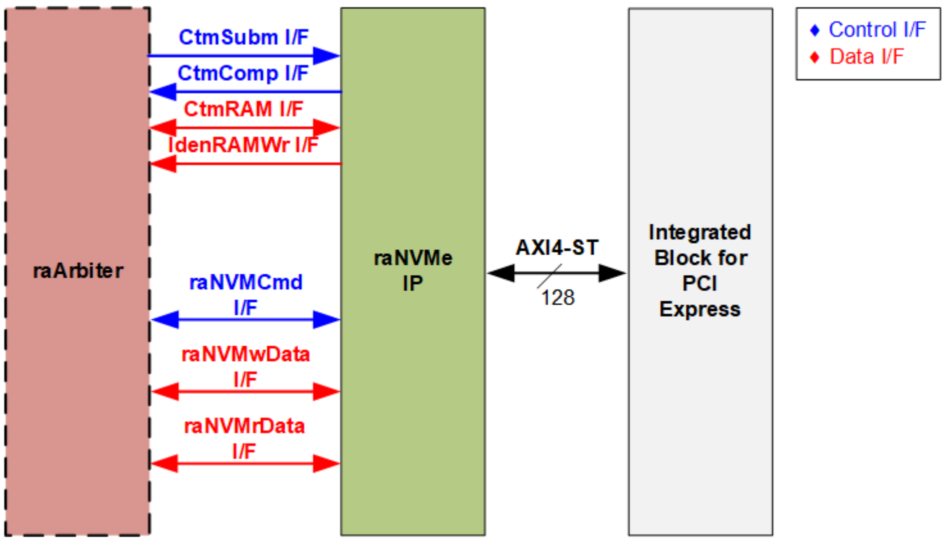

Figure 2-17 NVMe hardware

As shown in Figure 2-17, the user interface of raNVMe-IP is connected to raArbiter while another side of raNVMe-IP is connected to PCIe Hard IP. The user interface of raNVMe-IP consists of control interface and data interface. The control interface receives command and the parameters from the user while data interface transfers the data when the command needs data transferring.

raNVMCmd interface is the command interface for requesting the command which has six commands – Identify, SMART, Flush, Shutdown, Write and Read. CtmSubm I/F and CtmComp I/F are applied for setting parameters and returning status when running custom commands – Flush and SMART command.

There are four commands which has data transferring and each command transfers data via its own interface.

- CtmRAM I/F: Transfers SMART data to CtmRAM when running SMART command.
- IdenRAM I/F: Transfers Identify data to IdenRAM when running Identify command.
- raNVMwData I/F: Transfers Write data from raArbiter when running Write command.
- raNVMrData I/F: Transfers Read data from raNVMe-IP when running Read command.

Though each command uses the different interface for transferring the data, every data interface has the same data bus size, 128-bit data.

2.3.1 raNVMe-IP

The raNVMe-IP implements NVMe protocol of the host side to access one NVMe SSD. 32 Write or Read commands with random addressing can be sent to raNVMe-IP. More details of raNVMe-IP are described in datasheet.
https://dgway.com/products/IP/NVMe-IP/dg_ranvme_ip_data_sheet_xilinx.pdf

2.3.2 Integrated Block for PCIe

This block is the hard IP in Xilinx device which implements Physical, Data Link and Transaction Layers of PCIe specification. More details are described in Xilinx document.
PG156: UltraScale Devices Gen3 Integrated Block for PCI Express
PG213: UltraScale+ Devices Integrated Block for PCI Express

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 for running with multiple users by raNVMe-IP is connected with CPU as a peripheral on 32-bit AXI4-Lite bus. Consequently, CPU can set test parameters and monitor test status. CPU assigns the different base address to each peripheral for accessing one peripheral at a time. Also, the address range of each peripheral can be defined independently.

In the reference design, the CPU system is built with one additional peripheral to access the test logic. The base address and the range for accessing the test logic are defined in the CPU system. So, the hardware logic must be designed to support AXI4-Lite bus standard for CPU writing and reading. LAxi2Reg module is designed to connect with the CPU system as shown in Figure 2-18.

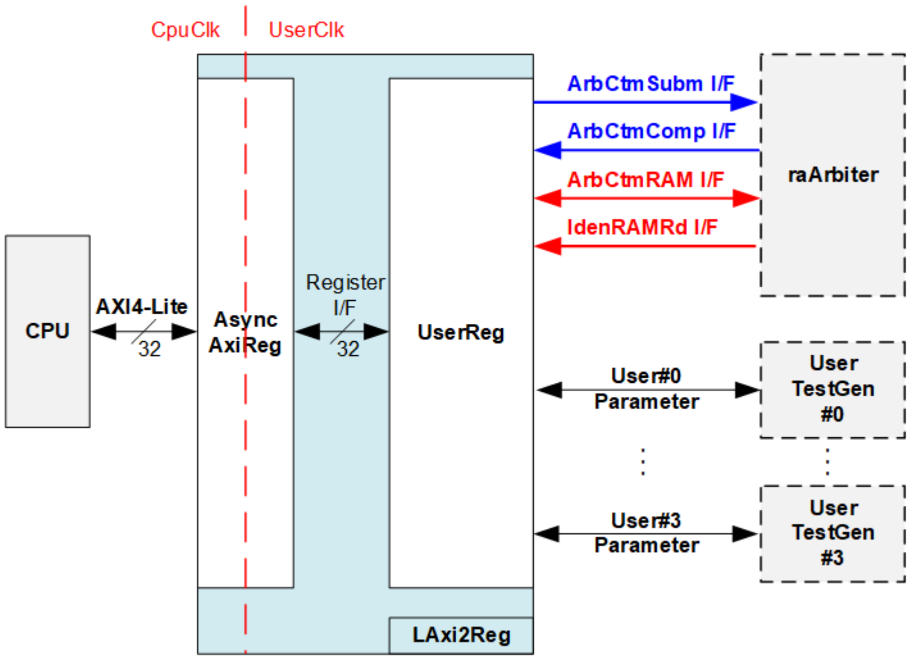

Figure 2-18 CPU and peripherals hardware

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. Additionally, AsyncAxiReg includes asynchronous logic to support clock crossing between CpuClk and UserClk domain.

UserReg includes the register files of the parameters and the status signals of other modules in the Test system. More details of AsyncAxiReg and UserReg are described as follows.

2.4.1 AsyncAxiReg

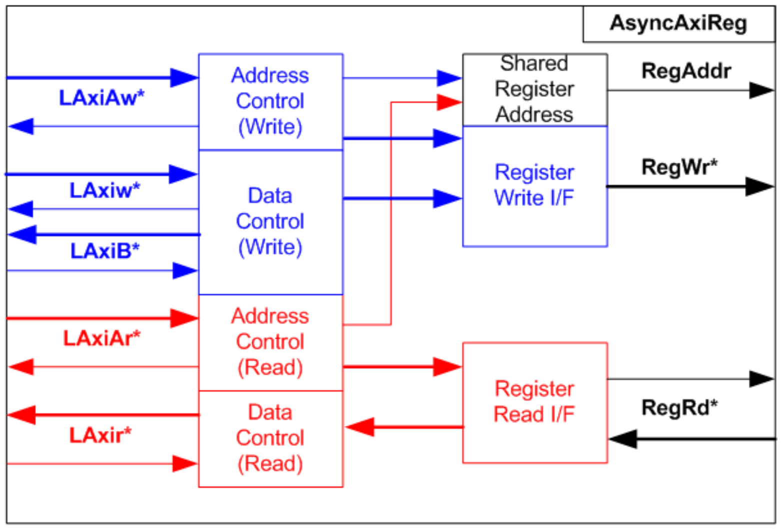

Figure 2-19 AsyncAxiReg Interface

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 2-19. Write control I/F and Write data I/F of AXI4-Lite bus are latched and transferred to be Write register interface with clock-crossing registers. Similarly, Read control I/F of AXI4-Lite bus are latched and transferred to be Read register interface while Read data is returned from Register interface to AXI4-Lite through clock-crossing registers. In Register interface, RegAddr is shared signal for write and read access, so it loads the value 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 2-20.

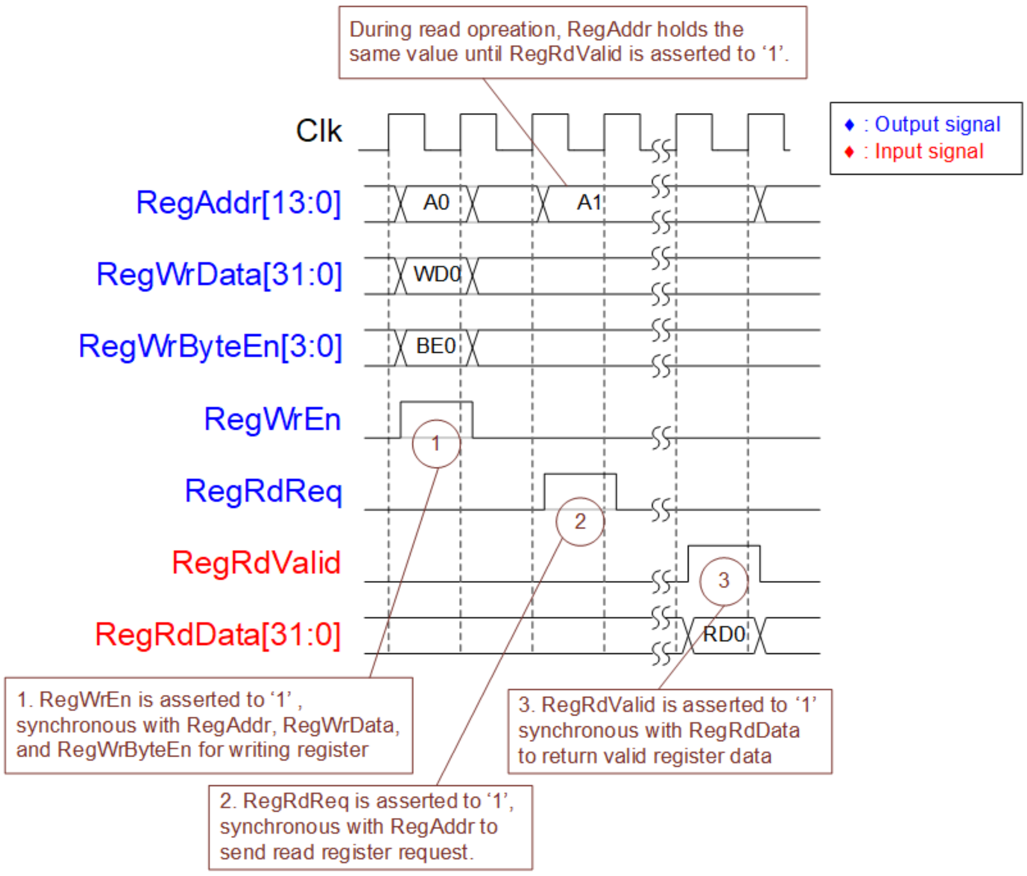
During read opreation, RegAddr holds the same value until RegRdValid is asserted to '1'.
Clk
RegAddr[13:0]
A0
A1
RegWrData[31:0]
WD0
RegWrByteEn[3:0]
BE0
RegWrEn
RegRdReq
RegRdValid
RegRdData[31:0]
RD0
♦ : Output signal
♦ : Input signal
1
2
3
1. RegWrEn is asserted to '1' , synchronous with RegAddr,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

Figure 2-20 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 must monitor RegRdReq signal to start the read transaction. During read operation, the address value (RegAddr) does not change the value until RegRdValid is asserted to '1'. So, the address can be used for selecting the returned data by using multiple-layer 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

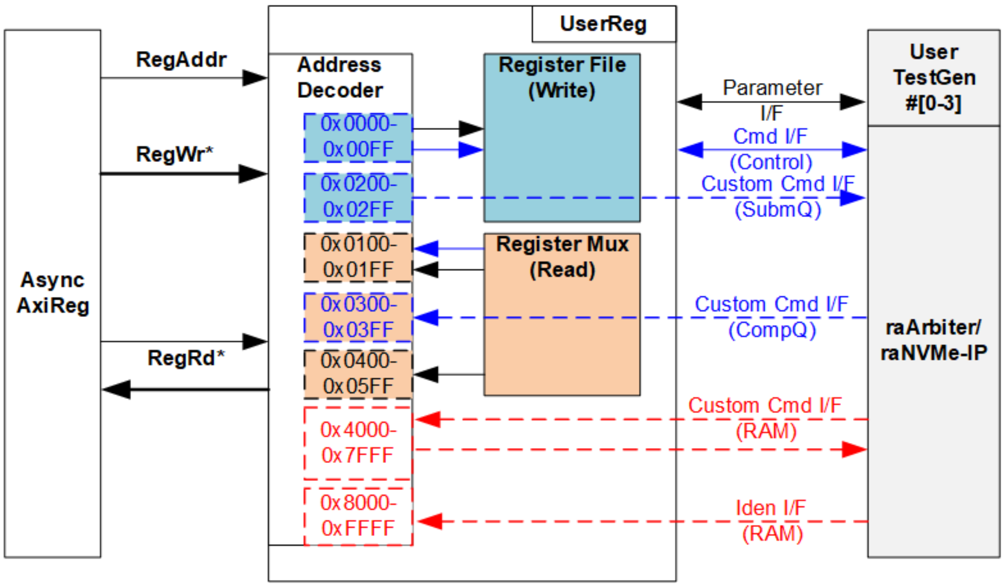

Figure 2-21 UserReg Interface

The address range to map to UserReg is split into seven areas, as shown in Figure 2-21.

1) 0x0000 – 0x00FF: mapped to set the command with the parameters of all UserTestGen and some parameters of raNVMe-IP. This area is write-access only.
2) 0x0200 – 0x02FF: mapped to set the parameters of custom command for all users. This area is write-access only.
3) 0x0100 – 0x01FF: mapped to read the status signals of raNVMe-IP and all UserTestGen. This area is read-access only.
4) 0x0300 – 0x03FF: mapped to read the status of custom command for all users. This area is read-access only.
5) 0x0400 – 0x05FF: mapped to read the verification details and completed count of all UserTestGen. This area is read-access only.
6) 0x4000 – 0x7FFF: mapped to write or read data with custom command RAM for all users. This area supports write and read access. The demo shows only read access for running SMART command.
7) 0x8000 – 0xFFFF: mapped to read data from IdenRAM for all users. This area is read-access only.

Address decoder decodes the upper bit of RegAddr for selecting the active hardware. The register file inside UserReg is 32-bit bus size, so write byte enable (RegWrByteEn) is not used. To write hardware registers, the CPU must use 32-bit pointer to place 32-bit valid value on the write data bus.

To read register, three-step multiplexer is designed to select the read data within each address area. The lower bit of RegAddr is applied in each Register area to select the data. Next, the address decoder uses the upper bit to select the read data from each area for returning to CPU. Totally, the latency of read data is equal to three clock cycles, so RegRdValid is created by RegRdReq with asserting three D Flip-flops. More details of the address mapping within UserReg module are shown in Table 2-1.

Table 2-1 Register Map

Address	Register Name (Label in the “ranvmemultest.c”)	Description
0x0000 – 0x00FF: Test parameters of UserTestGen and raNVMe-IP (Write access only)		
BA+0x0000	User#0 Start address (Low) Reg (USR0ADRL_REG)	[31:0]: Input to be bit[31:0] of start address of User#0 as 512-byte unit for using in Write or Read command, UsrStartAddr[31:0]
BA+0x0004	User#0 Start address (High) Reg (USR0ADRH_REG)	[15:0]: Input to be bit[47:32] of start address of User#0 as 512-byte unit for using in Write or Read command, UsrStartAddr[47:32]
BA+0x0008	User#0 Transfer length (Low) Reg (USR0LENL_REG)	[31:0]: Input to be bit[31:0] of transfer length of User#0 as 512-byte unit for using in Write or Read command, UsrLen[31:0]
BA+0x000C	User#0 Transfer length (High) Reg (USR0LENH_REG)	[15:0]: Input to be bit[47:32] of transfer length of User#0 as 512-byte unit for using in Write or Read command, UsrLen[47:32]
BA+0x0010	User#0 Command Reg (USR0CMD_REG)	[2:0]: Input to be command of User#0, ArbCmd[2:0] “000”: Identify, “001”: Shutdown, “010”: Write SSD, “011”: Read SSD, “100”: SMART, “110”: Flush, “101”/”111”: Reserved When this register is written command, the new command request (UsrCmdValid) is asserted to User#0.
BA+0x0014	User#0 Test pattern Reg (U0PATTSEL_REG)	[2:0]: Select test pattern of User#0. “000”-Increment, “001”-Decrement, “010”-All 0, “011”-All 1, “100”-LFSR. [3]: Verification enable. ‘0’ -No verification, ‘1’-Enable verification.
BA+0x0020 – BA+0x0037	User#1 Test parameters	Similar to 0x0000 – 0x0017 which is User#0 Test parameters, defined to be USR1ADRL/H_REG, USR1LENL/H_REG and USR1CMD_REG
BA+0x0040 – BA+0x0057	User#2 Test parameters	Similar to 0x0000 – 0x0017 which is User#0 Test parameters, defined to be USR2ADRL/H_REG, USR2LENL/H_REG and USR2CMD_REG
BA+0x0060 – BA+0x0077	User#3 Test parameters	Similar to 0x0000 – 0x0017 which is User#0 Test parameters, defined to be USR3ADRL/H_REG, USR3LENL/H_REG and USR3CMD_REG
BA+0x0080	NVMe Timeout Reg (NVMTIMEOUT_REG)	[31:0]: Timeout value of raNVMe-IP (TimeOutSet[31:0] of raNVMe-IP)

Address	Register Name (Label in the "ranvmemultest.c")	Description
0x0100 – 0x01FF: Status signals of raNVMe-IP and UserTestGen (Read access only)		
BA+0x0100	User Status Reg (USRSTS_REG)	[0]: Mapped to raNVMBusy of raNVMe-IP. '0': IP is Idle, '1': IP is busy. [1]: Mapped to raNVMError of raNVMe-IP. '0': No error, '1': Error is found. [4], [5], [6] and [7]: Data verification fail of UserTestGen#0, #1, #2 and #3 respectively. '0': Normal, '1': Error. [8], [9], [10] and [11]: Busy flag of UserTestGen#0, #1, #2 and #3 respectively. '0': User is Idle, '1': User is busy.
BA+0x0104	Total disk size (Low) Reg (LBASIZEL_REG)	[31:0]: Mapped to LBASize[31:0] of raNVMe-IP
BA+0x0108	Total disk size (High) Reg (LBASIZEH_REG)	[15:0]: Mapped to LBASize[47:32] of raNVMe-IP
BA+0x010C	User Error Type Reg (USRERRTYPE_REG)	[31:0]: Mapped to UserErrorType[31:0] of raNVMe-IP to show error status
BA+0x0110	PCIe Status Reg (PCISTS_REG)	[0]: PCIe linkup status from PCIe hard IP ('0': No linkup, '1': linkup) [3:2]: PCIe link speed from PCIe hard IP ("00": Not linkup, "01": PCIe Gen1, "10": PCIe Gen2, "11": PCIe Gen3) [7:4]: PCIe link width status from PCIe hard IP ("0001": 1-lane, "0010": 2-lane, "0100": 4-lane, "1000": 8-lane) [13:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in Integrated Block for PCIe datasheet
BA+0x0114	NVMe CAP Reg (NVMCAP_REG)	[31:0]: Mapped to NVMeCAPReg[31:0] of raNVMe-IP
BA+0x0118	Admin Completion Status Reg (ADMCOMPSTS_REG)	[15:0]: Mapped to AdmCompStatus[15:0] of raNVMe-IP to show status of Admin completion
BA+0x011C	IO Completion Status Reg (IOCOMPSTS_REG)	[31:0]: Mapped to IOCompStatus[15:0] of raNVMe-IP to show status of I/O completion.
BA+0x0120	NVMe IP Test pin Reg (NVMTESTPIN_REG))	[31:0]: Mapped to TestPin[31:0] of raNVMe-IP
0x0200 – 0x03FF: Custom command interface		
BA+0x0200 – BA+0x02FF Wr	Custom Submission Queue Reg (CTMSUBMQ_REG)	[31:0]: Submission queue entry of SMART and Flush command which are set for User#0 – User#3, ArbCtmSubmDW0-DW15. 0x200: DW0, 0x204: DW1, …, 0x23C: DW15 for User#0 0x240: DW0, 0x244: DW1, …, 0x27C: DW15 for User#1 0x280: DW0, 0x284: DW1, …, 0x2BC: DW15 for User#2 0x2C0: DW0, 0x2C4: DW1, …, 0x2FC: DW15 for User#3
BA+0x0300 – BA+0x033F Rd	Custom Completion Queue Reg (CTMCOMPQ_REG)	[31:0]: Completion queue entry of SMART and Flush command which are returned for User#0 – User#3, ArbCtmCompDW0-DW3. 0x300: DW0, 0x304: DW1, …, 0x30C: DW3 for User#0 0x310: DW0, 0x314: DW1, …, 0x31C: DW3 for User#1 0x320: DW0, 0x324: DW1, …, 0x32C: DW3 for User#2 0x330: DW0, 0x334: DW1, …, 0x33C: DW3 for User#3

Address Rd/Wr	Register Name (Label in the "ranvmemultest.c")	Description
0x0400 – 0x05FF: Verification details and completed count of UserTestGen (Read access only)		
BA+0x0400	User#0 Expected value Word0 Reg (U0EXPPATW0_REG)	[31:0]: Bit[31:0] of the expected data of User#0 at the 1st failure data in Read command
BA+0x0404	User#0 Expected value Word1 Reg (U0EXPPATW1_REG)	[31:0]: Bit[63:32] of the expected data of User#0 at the 1st failure data in Read command
BA+0x0408	User#0 Expected value Word2 Reg (U0EXPPATW2_REG)	[31:0]: Bit[95:64] of the expected data of User#0 at the 1st failure data in Read command
BA+0x040C	User#0 Expected value Word3 Reg (U0EXPPATW3_REG)	[31:0]: Bit[127:96] of the expected data of User#0 at the 1st failure data in Read command
BA+0x0410	User#0 Read value Word0 Reg (U0RDPATW0_REG)	[31:0]: Bit[31:0] of the read data of User#0 at the 1st failure data in Read command
BA+0x0414	User#0 Read value Word1 Reg (U0RDPATW1_REG)	[31:0]: Bit[63:32] of the read data of User#0 at the 1st failure data in Read command
BA+0x0418	User#0 Read value Word2 Reg (U0RDPATW2_REG)	[31:0]: Bit[95:64] of the read data of User#0 at the 1st failure data in Read command
BA+0x041C	User#0 Read value Word3 Reg (U0RDPATW3_REG)	[31:0]: Bit[127:96] of the read data of User#0 at the 1st failure data in Read command
BA+0x0420	User#0 Data Failure Address (Low) Reg (U0RDFAILNOL_REG)	[31:0]: Bit[31:0] of the byte address of User#0 at the 1st failure data in Read command
BA+0x0424	User#0 Data Failure Address (High) Reg (U0RDFAILNOH_REG)	[24:0]: Bit[56:32] of the byte address of User#0 at the 1st failure data in Read command
BA+0x0428	User#0 Completed Count (Low) Reg (U0CMDCMPCNTL_REG)	[31:0]: Bit[31:0] of the completed command count in User#0, UsrCompCnt
BA+0x042C	User#0 Completed Count (High) Reg (U0CMDCMPCNTH_REG)	[12:0]: Bit[44:32] of the completed command count in User#0, UsrCompCnt
BA+0x0480 – BA+0x04AF	User#1 Verification details	Similar to 0x0400 – 0x042F which is User#0 Verification details, defined to U1EXPPATW0-W3_REG, U1RDPATW0-W3_REG, U1RDFAILNOL/H_REG and U1CMDCMPCNTL/H_REG
BA+0x0500- BA+0x052F	User#2 Verification details	Similar to 0x0400 – 0x042F which is User#0 Verification details, defined to U2EXPPATW0-W3_REG, U2RDPATW0-W3_REG, U2RDFAILNOL/H_REG and U2CMDCMPCNTL/H_REG
BA+0x0580- BA+0x05AF	User#3 Verification details	Similar to 0x0400 – 0x042F which is User#0 Verification details, defined to U3EXPPATW0-W3_REG, U3RDPATW0-W3_REG, U3RDFAILNOL/H_REG and U3CMDCMPCNTL/H_REG
IP Version and RAM		
BA+0x0800 Rd	IP Version Reg (IPVERSION_REG)	[31:0]: Mapped to IPVersion[31:0] of raNVMe-IP
BA+0x4000 – BA+0x7FFF Wr/Rd	Custom command RAM (CTMRAM_REG)	Connect to 16K byte CtmRAM interface for storing SMART data of User#0-User#3, 4K byte for each user. 0x4000-0x4FFF: Custom RAM area for User#0 0x5000-0x5FFF: Custom RAM area for User#1 0x6000-0x6FFF: Custom RAM area for User#2 0x7000-0x7FFF: Custom RAM area for User#3
BA+0x8000 – BA+0xFFFF Rd	Identify RAM (IDENCTRL_REG)	0x8000-0x8FFF: 4Kbyte Identify Controller Data for User#0 0x9000-0x9FFF: 4Kbyte Identify Namespace Data for User#0 0xA000-0xAFFF: 4Kbyte Identify Controller Data for User#1 0xB000-0xBFFF: 4Kbyte Identify Namespace Data for User#1 0xC000-0xCFFF: 4Kbyte Identify Controller Data for User#2 0xD000-0xDFFF: 4Kbyte Identify Namespace Data for User#2 0xE000-0xEFFF: 4Kbyte Identify Controller Data for User#3 0xF000-0xFFFF: 4Kbyte Identify Namespace Data for User#3

3 CPU Firmware

3.1 Test firmware (ranvmeiptest.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 (PCISTS_REG[0]='1').
3) CPU waits until raNVMe-IP completes initialization process (USRSTS_REG[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 PCISTS_REG[7:2].
5) CPU displays the main menu. There are six menus for running six commands of raNVMe-IP, i.e., Identify, Write, Read, SMART, Flush and Shutdown.

More details for operating each command in CPU firmware are described as follows.

3.1.1 Identify Command

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

1) Receive number of users to operate command, valid from 1-4 users. The operation is cancelled if the input is out-of-range.
2) Receive the first user number for sending the command, valid from User#0 to User#3. If selected user is out-of-range, the operation is cancelled.

   For example, when number of users is 3 and the first user is user#2, the CPU sets the request to user#2, user#3 and user#0 respectively.
3) Set the current user variable by the input from step 2).
4) Set USR(X)CMD_REG ="000" (Identify command) when (X) is current user number. Next, the command request is asserted to UserTestGen and busy flag of the current user (USRSTS_REG[8+(X)] when (x) is current user number) changes from '0' to '1'.
   Note: Bit[8], [9], [10] and [11] of USRSTS_REG are busy flag of User#0, #1, #2 and #3 respectively.
5) CPU waits until the operation is completed or some errors are detected by monitoring USRSTS_REG[1] and USRSTS_REG[8+(X)].

   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 USRERRTYPE_REG[31:0]. Finally, the process is stopped.
   Bit[8+(X)] is de-asserted to '0' after finishing operating the command. After the command is completed, the data from Identify command of raNVMe-IP is stored in IdenRAM and CPU goes to the next step.

6) CPU displays the information which is decoded from IdenRAM (IDENCTRL_REG) such as SSD model name and the information from raNVMe-IP output such as SSD capacity (LBASIZEH/L_REG). The address to access IdenRAM depends on the current user number.
7) Repeat step 4) – 6) for running the operation by the next user number if the current user is not the last user number.

3.1.2 Write/Read Command

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

Step 1) – 2) are similar to step 1) – 2) in Identify command.

3) Receive start address, transfer length and test pattern from Serial console. If some inputs are invalid, the operation is cancelled.
   Note: Start address and transfer length must be aligned to 8.

   The start address and transfer length of each user are calculated by the firmware by following sequence.
   a. Start address value is set to be start address of the first user while the transfer length of the first user is RoundUpAlign8(Transfer length/number of users).
   b. Start address value of the next user is equal to start address of the first user + the length of the first user. Transfer length is RoundUpwithalign8(Remained transfer length/remained number of users).

   For example, when number of users = 3, the first user=2, start address = 0x1000 and transfer length = 0x310. The firmware sets the parameters of each user in following sequence.
   User#2: Start address = 0x1000, Length = RoundupAlign8(0x310/3) = 0x108
   User#3: Start address = 0x1108, Length = RoundupAlign8((0x310-0x108)/2) = 0x108
   User#1: Start address = 0x1210, Length = RoundupAlign8((0x310-0x108-0x108)/1) = 0x100

4) Set the parameters to each user, i.e., USR(X)ADRL/H_REG for start address, USR(X)LENL/H_REG for transfer length and U(X)PATTSEL_REG for test pattern selector.
   Note: (X) is the user number.

5) Set USR(X)CMD_REG[2:0]="010" for Write command or "011" for Read command when (X) is current user number, starting from the first user number to the last user number.
6) CPU waits until the operation is completed or some errors (except verification error) are detected by monitoring USRSTS_REG[1] and USRSTS_REG[11:4]. Display the error message when some bits are asserted to '1'.
   Note: Bit[4], [5], [6] and [7] of USRSTS_REG is data failure flag of User#0, #1, #2 and #3 respectively.

   Bit[1] is asserted when IP error is detected. The process is hanged up when this error is found.
   Bit[7:4] is not equal to 0000b in Read command when data failure is found in some users. The verification error message is displayed. In this condition, CPU is still running until the operation is done or user inputs any keys to cancel operation.
   Bit[11:8] is de-asserted to 0000b when command of all user is completed. After that, CPU goes to the next step.

   During running command, current transfer size of each user is read and displayed on the console every second. It can be calculated from the read value of U(X)CMDCMPCNTL/H_REG with multiplying by 4Kbyte. Also, total current transfer size is calculated from the sum of each user's transfer size and displayed on the console.

7) Calculate and display the test results on the console, i.e., total time usage, total transfer size and transfer speed.

3.1.3 SMART Command

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

Step 1) – 3) are similar to step 1) – 3) in Identify command.

4) Set 16-Dword of Submission queue entry of current user number (CTMSUBMQ_REG) to be SMART command value.
5) Set USR(X)CMD_REG[2:0]="100" (SMART command) when (X) is current user number. Next, the command request is asserted to UserTestGen and busy flag of the current user (USRSTS_REG[8+(X)] when (x) is current user number) changes from '0' to '1'.
6) CPU waits until the operation is completed or some errors are detected by monitoring USRSTS_REG[1] and USRSTS_REG[8+(X)].

   Note: Bit[8], [9], [10] and [11] of USRSTS_REG is busy flag of User#0, #1, #2 and #3 respectively.

   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 USRERRTYPE_REG[31:0]. Finally, the process is stopped.

   Bit[8+(X)] is de-asserted to '0' after finishing operating the command. After the command is completed, the data from SMART command of raNVMe-IP is stored in CtmRAM and CPU goes to the next step.

7) CPU displays the information which is decoded from CtmRAM (CTMRAM_REG) such as Temperature, Total Data Read, Total Data Written, Power On Cycles, Power On Hours and Number of Unsafe Shutdown. The address to access CtmRAM depends on the current user number.
8) Repeat step 4) – 7) for running the operation by the next user number if the current user is not the last user number.

   More details of SMART log are described in NVM Express Specification.
   https://nvmexpress.org/developers/nvme-specification/

3.1.4 Flush Command

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

Step 1) – 3) are similar to step 1) – 3) in Identify command.

4) Set 16-Dword of Submission queue entry of current user number (CTMSUBMQ_REG) to be Flush command value.
5) Set USR(X)CMD_REG[2:0]="110" (Flush command) when (X) is current user number. Next, the command request is asserted to UserTestGen and busy flag of the current user (USRSTS_REG[8+(X)] when (x) is current user number) changes from '0' to '1'.
6) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_REG[1] and USRSTS_REG[8+(X)].

   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 USRERRTYPE_REG[31:0]. Finally, the process is stopped.

   Bit[8+(X)] is de-asserted to '0' after finishing operating the command. After that, CPU goes to the next step.
7) Repeat step 4) – 6) for running the operation by the next user number if the current user is not the last user number.

3.1.5 Shutdown Command

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

1) One SSD is available in the demo, so Shutdown command can be sent once by one user. The console receives the user number for running Shutdown command which is valid from User#0 to User#3. If selected user is out-of-range, the operation is cancelled.
2) Set the current user variable by the input from step 1).
3) Set USR(X)CMD_REG[2:0]="001" (Shutdown command) when (X) is current user number. Next, the command request is asserted to UserTestGen and busy flag of the current user (USRSTS_REG[8+(X)] when (x) is current user number) changes from '0' to '1'
4) CPU waits until the operation is completed or some errors are found by monitoring USRSTS_REG[1] and USRSTS_REG[8+(X)].

   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 USRERRTYPE_REG[31:0]. Finally, the process is stopped.
   Bit[8+(X)] is de-asserted to '0' after finishing operating the command. After that, CPU goes to the next step.

5) After Shutdown command, the SSD and raNVMe-IP change to inactive status. So, the CPU cannot receive the new command. 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 raNVMe-IP
Description	Run SMART command or Flush command of the current user, following in topic 3.1.3 (SMART Command) and 3.1.4 (Flush Command).

int flush_ctmnvm(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the raNVMe-IP
Description	Set Flush command to CTMSUBMQ_REG and call exec_ctm function to operate Flush command for the current user.

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.

int get_user_num(void)	
Parameters	None
Return value	0: Valid input, -1: Invalid input
Description	Receive total number of users and the start user number and verify the value. If the inputs are valid, set the value to global parameters – num_user and start_user. Otherwise, the function returns -1.

void iden_dev(void)	
Parameters	None
Return value	None
Description	Run Identify command of the current user, following in topic 3.1.1 (Identify Command). One user is run when calling this function.

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

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

void show_progress(unsigned int* disp_header)	
Parameters	disp_header: 1-Display info with header for the first runtime 0-Display info without header for the other runtime
Return value	None
Description	For the first runtime, display the header to show the information. Calculate total transfer length of each user by reading U[X]CMDCMPCNTL/H_REG and then display on the console by calling show_size function.

void show_result(void)	
Parameters	None
Return value	None
Description	Print total size by reading U[X]CMDCMPCNT_REG and then calling 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 on MB/s unit and IOPS unit.

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

void show_smart_hex(unsigned char *char_ptr16B)	
Parameters	*char_ptr16B
Return value	None
Description	Display SMART data as hexadecimal unit.

void show_smart_raw(unsigned char *char_ptr16B)	
Parameters	*char_ptr16B
Return value	None
Description	Display SMART data as decimal unit when the input value is less than 4 MB. Otherwise, display overflow message.

void show_smart_unit(unsigned char *char_ptr16B)	
Parameters	*char_ptr16B
Return value	None
Description	Display SMART data as GB or TB unit. When the input value is more than a limit (500 PB), the overflow message is displayed instead.

void show_vererr(int user_no)	
Parameters	user_no: user number for displaying verification error. Valid from 0-3.
Return value	None
Description	Read U[user_no]RDFAILNOL/H_REG (error byte address), U[user_no]EXPPATW0-W3_REG (expected value) and U[user_no]RDPATW0-W3_REG (read value) to display verification error details on the console.

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

int smart_ctmadm(void)	
Parameters	None
Return value	0: No error, -1: Some errors are found in the raNVMe-IP
Description	Set SMART command to CTMSUBMQ_REG and call exec_ctm function to operate SMART command for the current user. Finally, decode and display SMART information on the console.

int wrrd_mult(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 comparison when running demo system by using 280 GB Intel 900P and KCU105 board (PCIe Gen3) by single user and four users are shown in Figure 4-1.

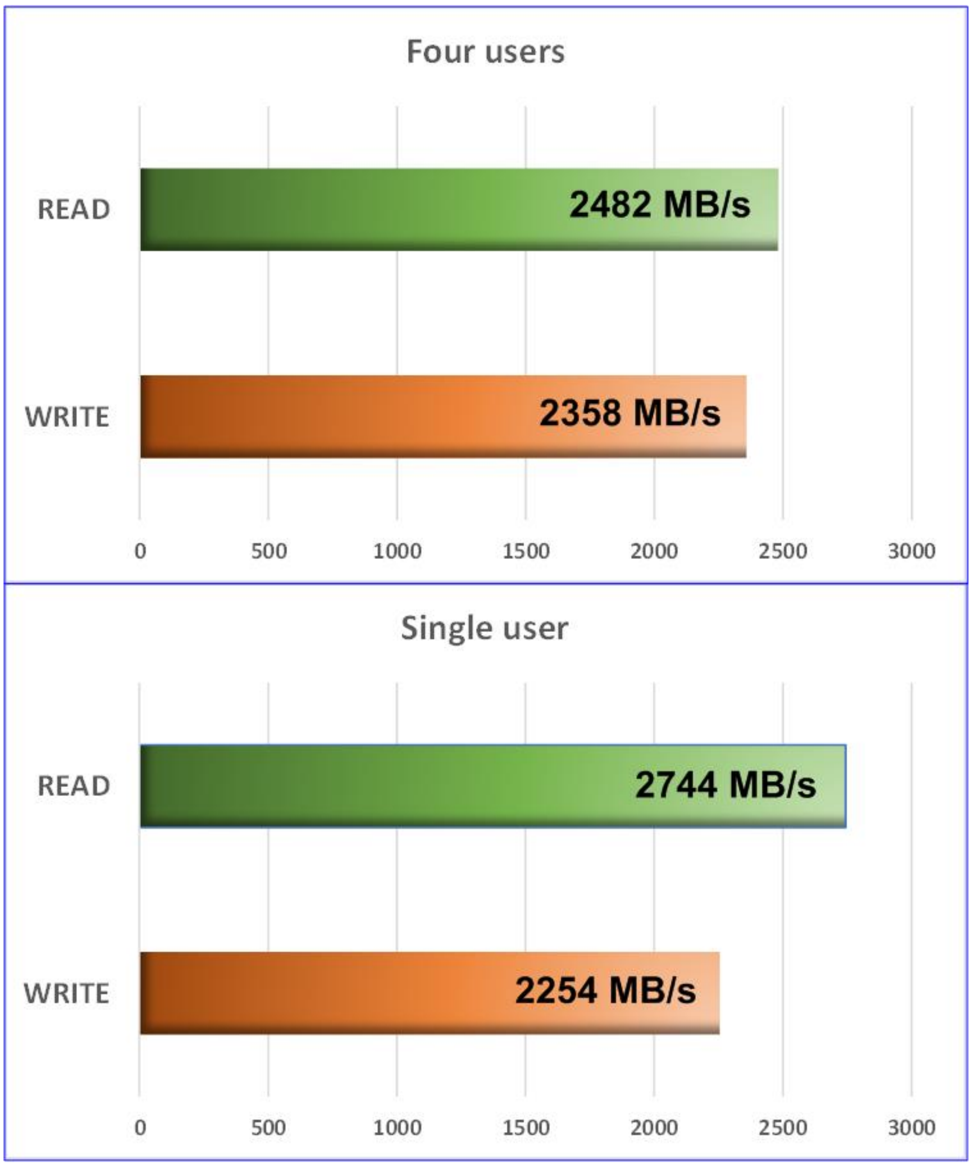

Figure 4-1 Test Performance of raNVMe-IP demo

When running Single user access, Write performance is about 2200 Mbyte/s and Read performance is about 2700 Mbyte/sec. When running four user access, total write performance is about 2,400 MB/s and total read performance is about 2,300 MB/s.

Comparing to two mode result, write performance is not much different while read performance of four user mode is slightly reduced. When running four user mode, the address sent to SSD is not sequential order.

5 Revision History

Revision	Date	Description
1.0	7-Jan-21	Initial Release