raNVMe-IP with multiple-user reference design manual

Rev1.1 21-Aug-23

raNVMe-IP with multiple-user reference design manual

Rev1.1 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 many areas by many users, as shown in Figure 1-1. The demo is designed to support up to four users at the same time.

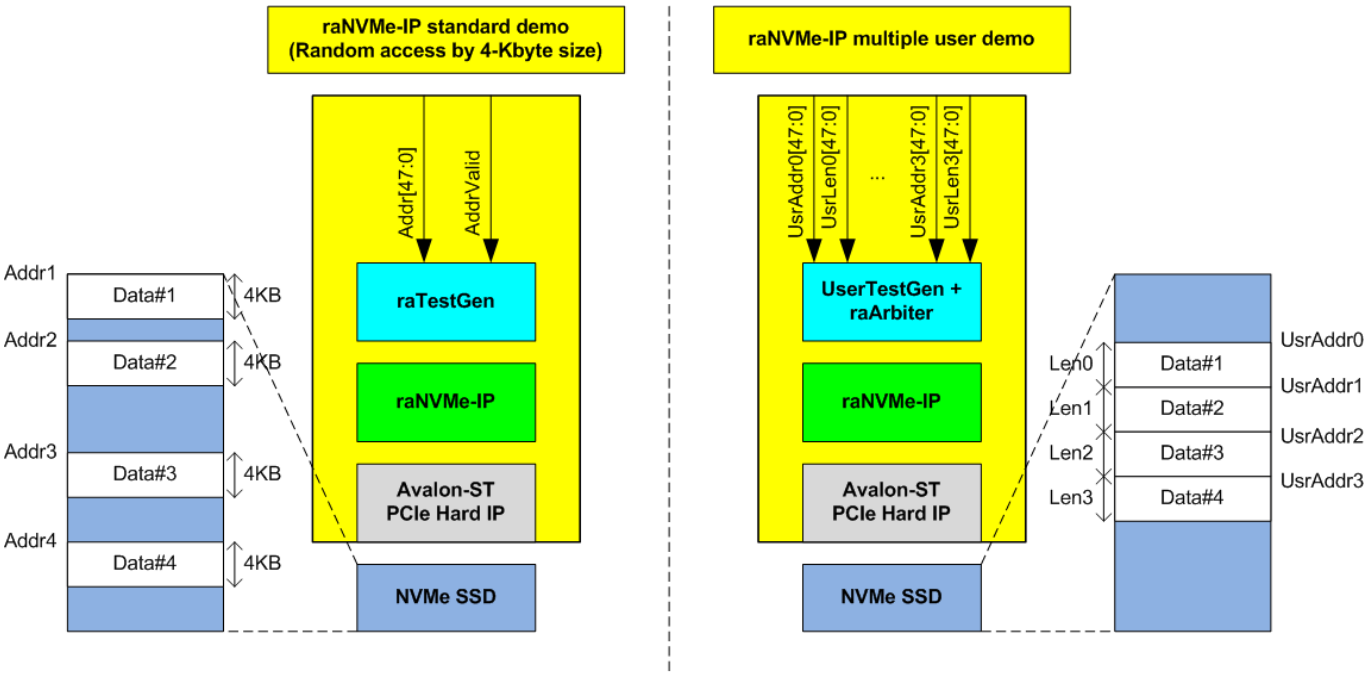

Figure 1-1 Standard demo and multiple-user demo comparison

In multiple-user demo, the start address and transfer length of each user are assigned to different area. The hardware supports to set the address and the length of each user individually while the test firmware receives only two parameters – the first address and total length. After that, the test firmware splits the firmware to four areas and calculates the start address and transfer length for four users. However, the start address and transfer length of each user must be aligned to 4 Kbytes. When transfer length of Write/Read command is more than 4 Kbytes, the hardware generates multiple 4 KB Write/Read commands to raNVMe-IP.

According to raNVMe-IP specification, Write and Read commands are multiple mode commands that allows the user to send up to 32 requests with different address but the same command. If user sends single mode command (Identify, SMART, Shutdown, or Flush), only one command is supported for sending to raNVMe-IP at a time. The example application for using multiple-user demo is shown in Figure 1-2.

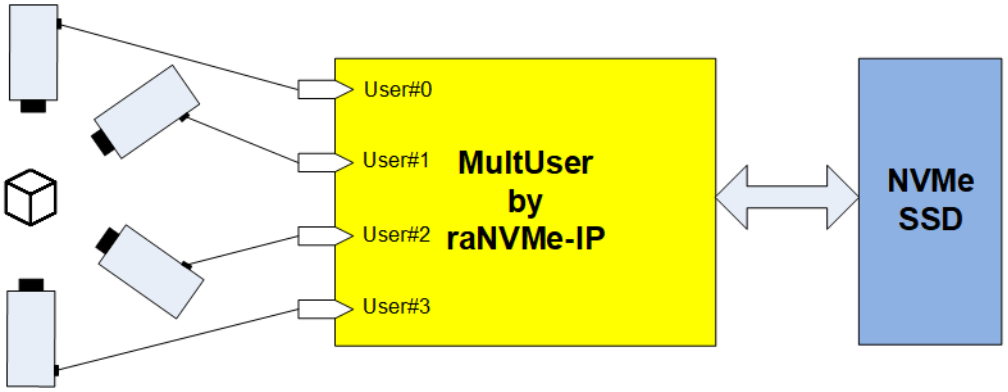

Figure 1-2 Application example

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 be written and read 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 default demo is designed to connect up to four users, the demo can be modified to support more users.

2 Hardware overview

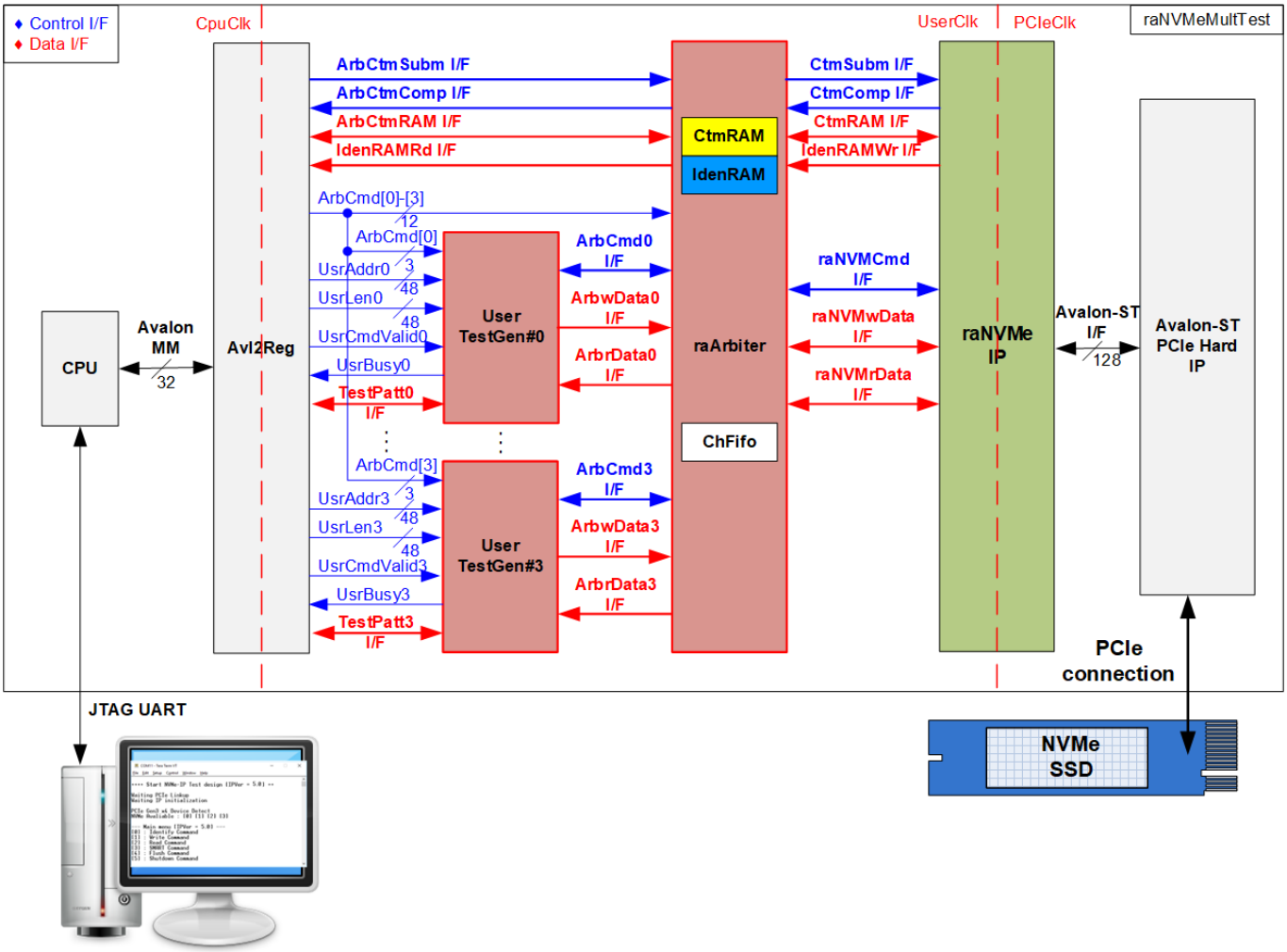

Figure 2-1 raNVMe-IP with multiple user demo hardware

In raNVMe-IP standard demo, one test module connects to the user interface of raNVMe-IP. In multiple-user demo, four test modules (UserTestGen) are integrated to represent that many users can access the same NVMe SSD at the same time by using raArbiter and raNVMe-IP.

According to raNVMe-IP specification, the commands have two modes - single mode and multiple mode. Therefore, raArbiter supports 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 store the order of the command from the active user, ChFifo must be included. raArbiter uses ChFifo to handle the order of the transferred data to be the same as the command order. raArbiter writes the active user number to ChFifo when sending the command request to raNVMe-IP. While ChFifo is read to select the same active user as the command request for transferring data with raNVMe-IP.

When running Write or Read command, each UserTestGen receives the start address and total length from CPU via Avl2Reg. After that, UserTestGen creates many 4KB Write or Read command requests with assigning the address to raArbiter via ArbCmd I/F. The address of each UserTestGen 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. When running other commands, UserTestGen creates one command request to raNVMe-IP.

SMART command and Identify command are single mode commands which needs to transfer the data. Identify and Custom interface of four channels are directly connected between Avl2Reg 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 each channel separately. In the demo, all users should get the same SMART data and Identify data because one SSD is accessed. User can remove or modify the hardware to compatible with the system requirement.

Multiple-user demo can be all designed by pure-hardwire logic, but CPU and Avl2Reg are included for user interface via JTAG UART. When running the test, user can adjust the test parameters on the console for running many test conditions. Also, the console is applied to display the current status, the test progress, and the test result.

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 is stable clock and independent from the other hardware interface. UserClk is the example clock domain of user logic for interface with raNVMe-IP. According to raNVMe-IP datasheet, clock frequency of UserClk must be more than or equal to PCIeClk. This reference design uses 275 MHz. Finally, PCIeClk is the clock output from PCIe hard IP to synchronous with data stream of 128-bit Avalon-ST interface. 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

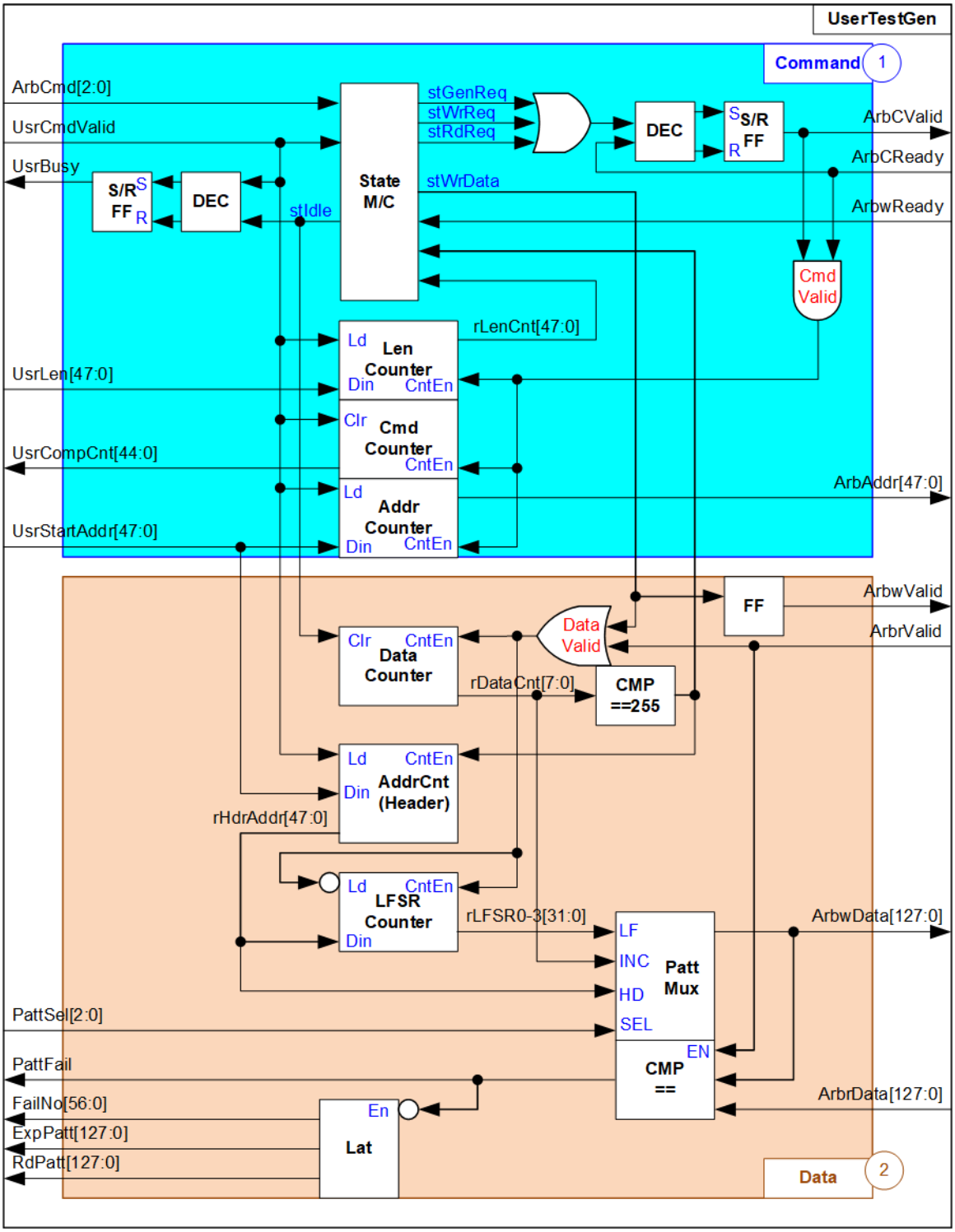

Figure 2-2 UserTestGen block diagram

UserTestGen is the module to send the command request. Also, it transfers the data with the Arbiter when running Write or Read command. The demo system consists of four UserTestGen modules to run multiple-user feature by using one raNVMe-IP. Four user interfaces of the Arbiter which consist of command interface and the data interface are similar to raNVMe-IP user interface. When running single mode command - Identify, SMART, Flush, and Shutdown command, one command is run at a time. The data interface of Identify and SMART command is not connected to UserTestGen, but connected to the Arbiter. Write and Read command are multi-mode commands 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) *ArbCReady is designed as ACK signal. 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 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.

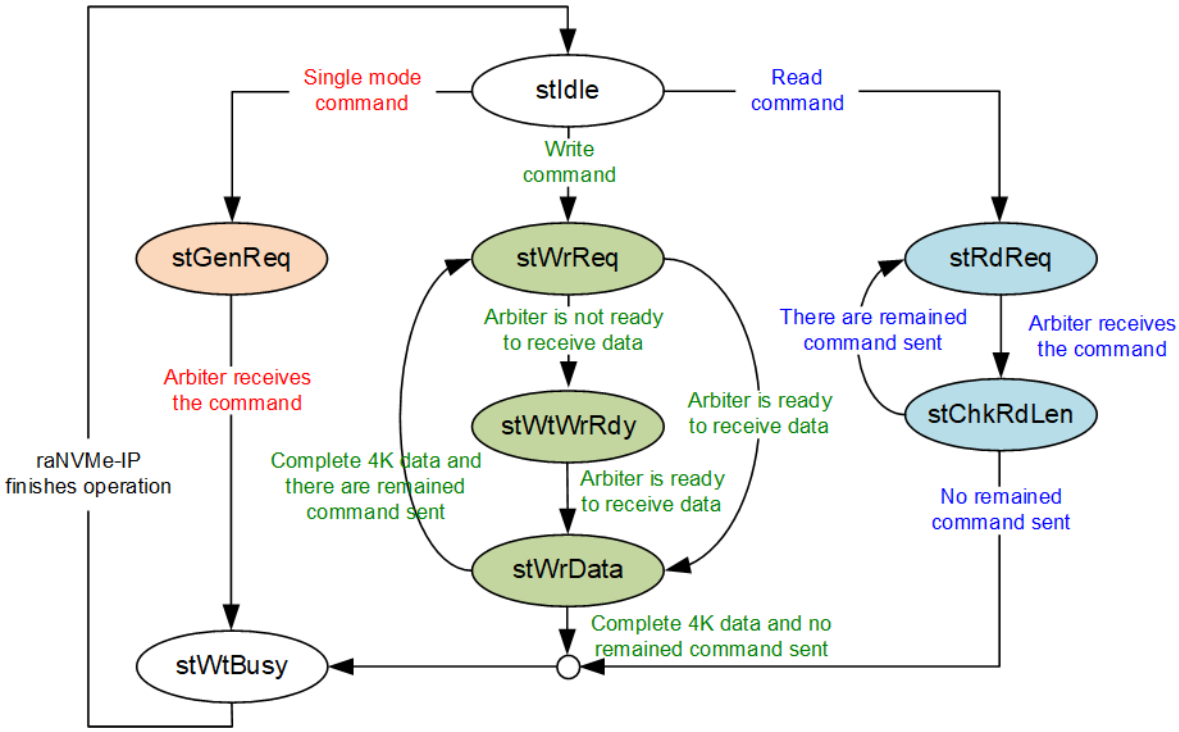

Figure 2-3 State machine in UserTestGen

1) stIdle: This state waits 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 mode command, write command (UsrCmd=010b), or read command (UsrCmd=011b), respectively.
2) stGenReq: This state sends 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 waits until the Arbiter completes the command operation by de-asserting ArbBusy to '0'. After finishing, the state returns to stIdle.
4) stWrReq: This state sends 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 – stWtWrRdy or stWrData. The next state is stWrData if the Arbiter is ready to receive the data (ArbwReady='1'). Otherwise, it changes to stWtWrRdy to wait until the Arbiter is ready for receiving the data.
5) stWtWrRdy: This state waits 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 processed for 256 cycles to send 4KB Write data (256x128-bit) to the Arbiter. After finishing the operation, it changes to stWtBusy if there is no remained command for sending the request. Otherwise, it returns to stWrReq to send the next Write command request.
7) stRdReq: This state sends 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 returns 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. 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 number of 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' for sending each 128-bit data. 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 number of cycles for asserting ArbwValid to '1' when running Write command. Also, it is applied to count the amount of 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.

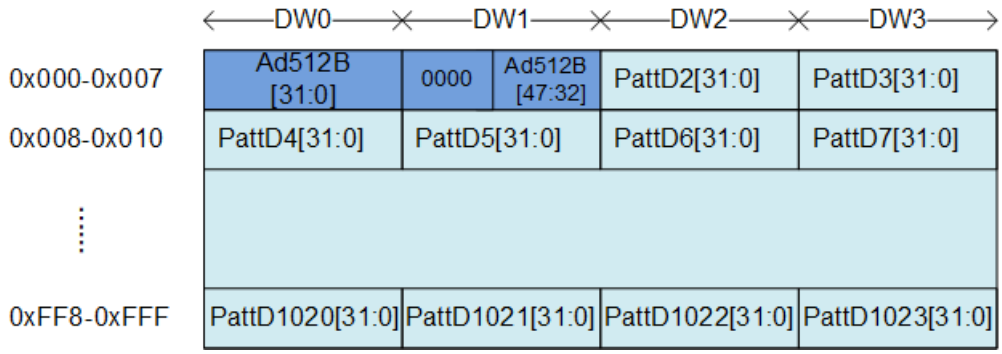

Figure 2-4 Test pattern format of 4096-byte data for Increment/Decrement/LFSR pattern

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 mode command - Identify, SMART, Flush, and Shutdown.

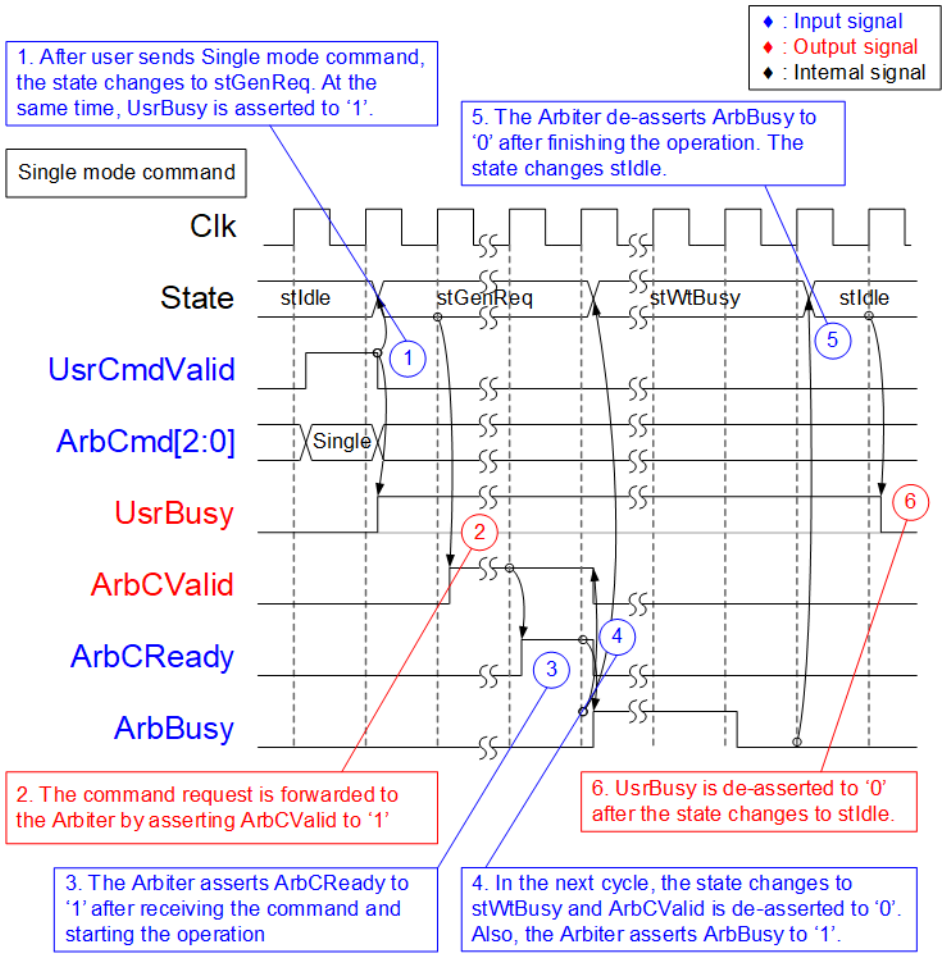

Figure 2-5 Timing diagram of UserTestGen when running single mode command

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' along with the command on ArbCmd. After that, the state changes to stGenReq.
2) In stGenReq, ArbCValid is asserted to '1' to send the single mode 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 starts operating the command.
4) In the next cycle, the state changes to stWtBusy and waits until the Arbiter completes the operation. During operating, the Arbiter asserts ArbBusy to '1'.
5) After the command is operated completely, ArbBusy is de-asserted to '0'. In the next cycle, the state returns 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.

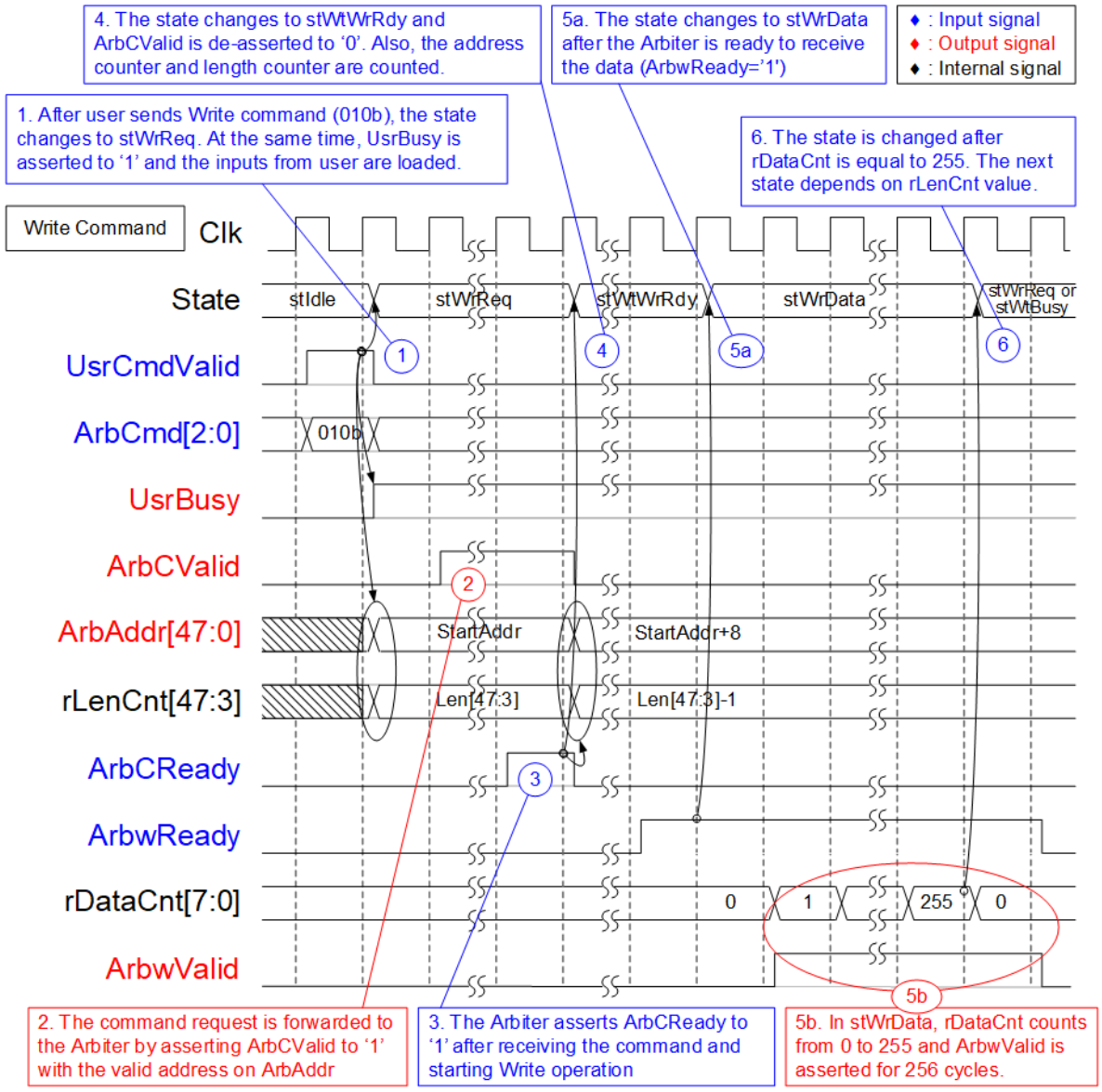

Figure 2-6 Timing diagram of UserTestGen when running Write command

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 - 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 accepts 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 to be the parameters of the next command.
5) The state changes to stWrData for sending 256 data or 4 KB data. rDataCnt is counted from 0 to 255. The state is changed when rDataCnt is equal to 255.
6) If there is remained data which must be transferred, the state returns 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 mode command.

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

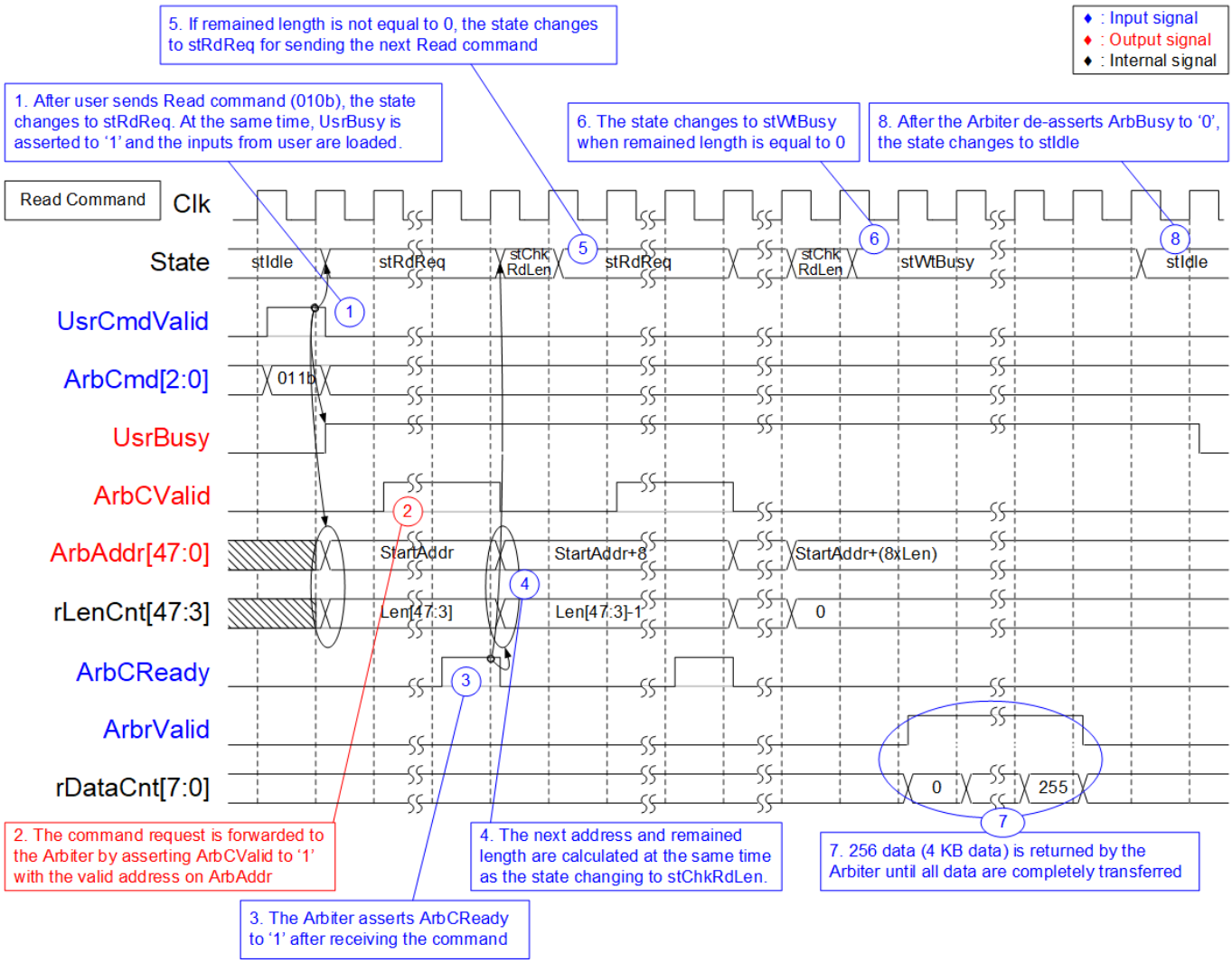

Figure 2-7 Timing diagram of UserTestGen when running Read command

1) When the Read command (ArbCmd=011b) is requested by the user, the state changes to stRdReq. Also, UsrBusy is asserted to '1' and the inputs from user - 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' along with ArbAddr. After that, the state waits until the request is accepted by the Arbiter.
3) The Arbiter accepts 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 can be stRdReq or stWtBusy depending on rLenCnt value. At the same time, the address (ArbAddr) and the remained length (rLenCnt) are counted to be the parameters of the next command.
5) If remained length is not equal to 0, the state returns 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. After that, ArbBusy is de-asserted to '0'.
7) 256 data or 4 KB read data is returned from the Arbiter after receiving one Read command. When 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

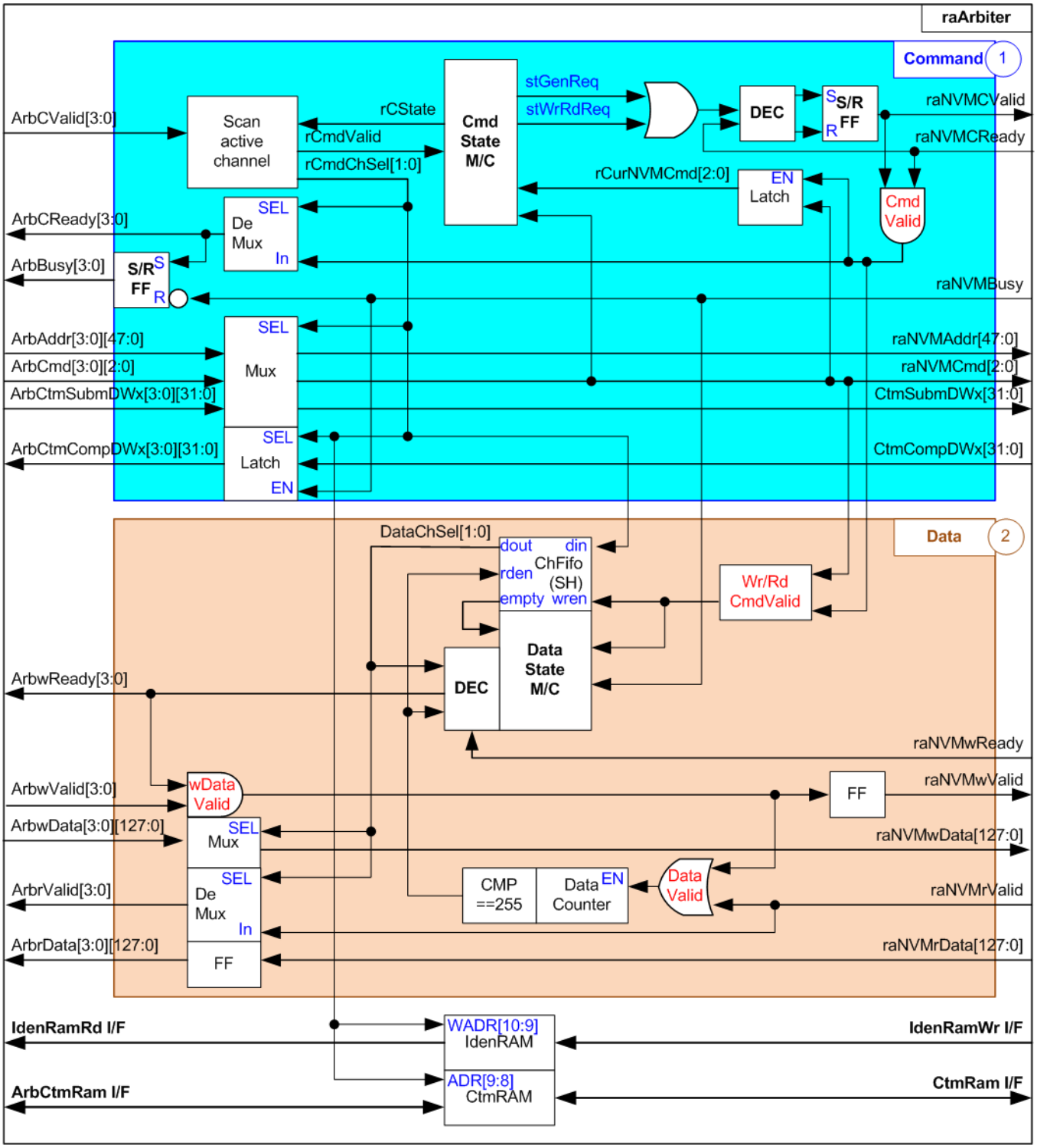

Figure 2-8 raArbiter block diagram

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. raArbiter includes IdenRAM and CtmRAM to store the data returned from Identify command and SMART command that are sent by every user. For simple architecture of the Arbiter, the user must send the command request by asserting ArbCValid to '1' before starting transferring the data. ChFIFO stores 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 command 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. Also, the command request (rCmdValid) is asserted to Cmd State machine. If all users send the request at the same time, the sequence of active user will be 0 -> 1 -> 2 -> 3. 2-bit counter inside Scan active channel block is applied to change 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.

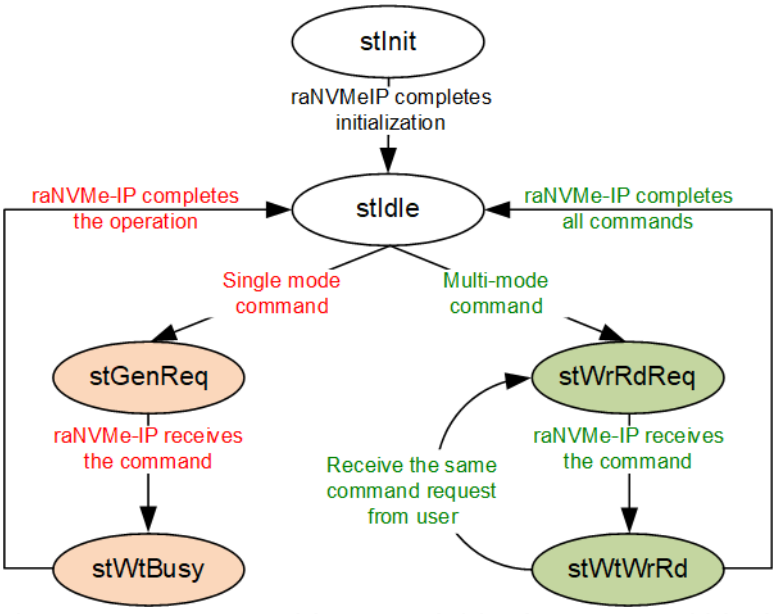

Figure 2-9 State machine of raArbiter's command block

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 then asserts rCmdValid to '1'. After that, the command value (raNVMCmd) is read. If the command is single mode command, the next state is stGenReq. Otherwise, the next state is stWrRdReq.
3) stGenReq: This state generates 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 returns to stIdle for receiving the next command from user.
5) stWrRdReq: This state generates 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. If the new command is different, it needs to wait until raNVMe-IP finishes all operations, monitored by raNVMBusy='0'. After that, the state returns to stIdle to read the new command.

Timing diagram of Command interface when running single mode command and multi-mode command are shown in Figure 2-10 and Figure 2-11, respectively.

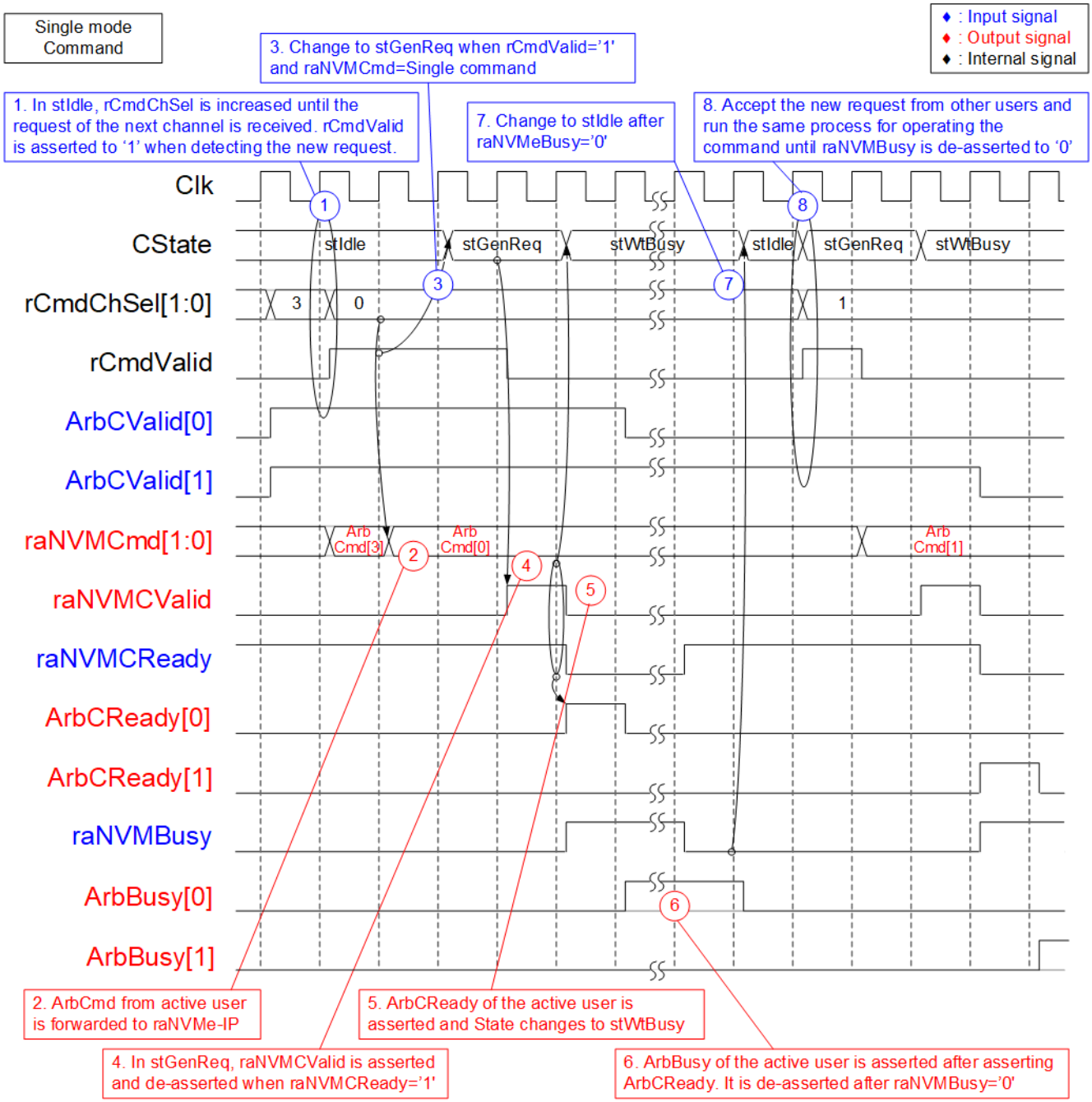

Figure 2-10 Timing diagram of raArbiter's Command I/F when running single mode command

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 load the value 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 mode 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 returns 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 mode command.

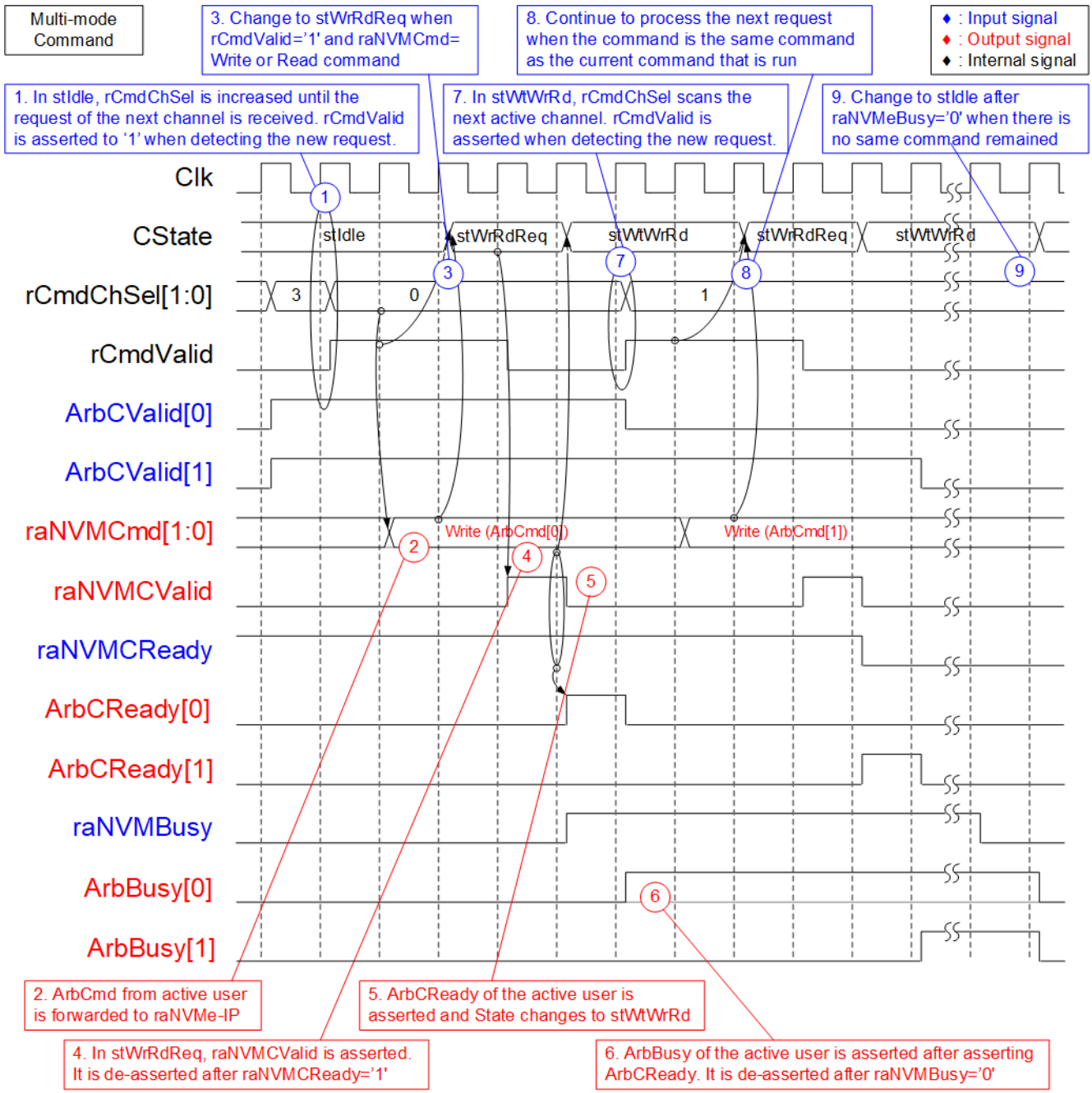

Figure 2-11 Timing diagram of raArbiter's Command I/F when running multi-mode command

Step 1) and 2) are similar to single mode command.

3) Two clock cycles after rCmdValid asserted to ‘1’, the state reads raNVMCmd to check the command type. If the command is multi-mode 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) The state 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 user data interface and raNVMe-IP are ready for transferring, Data State machine reads data (DataChSel) from ChFIFO. DataChSel defines the active user that transfers the data with raNVMe-IP. ChFIFO is Show-Ahead FIFO. Therefore, the read data (DataChSel) is valid when the FIFO is not empty. While Data State machine asserts read enable to get the next active channel after finishing the current user command.

Data counter counts total amount 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.

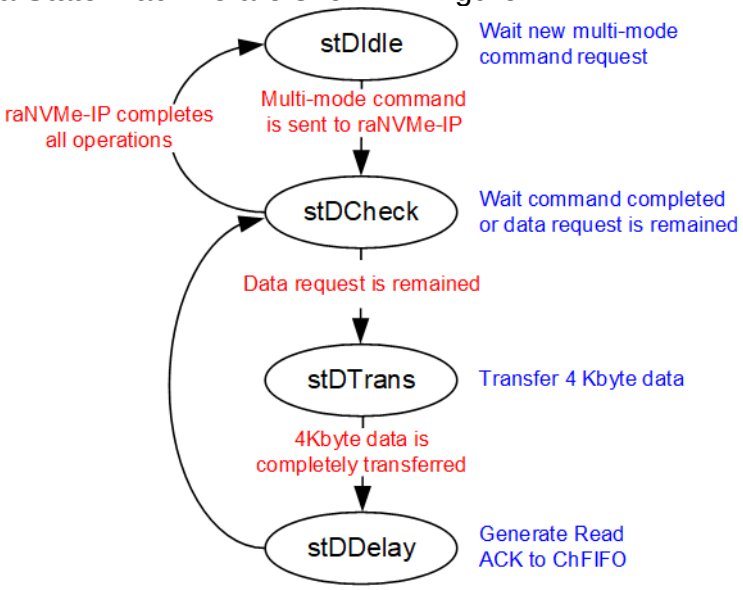

Figure 2-12 State machine of raArbiter's data block

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

1) stDIdle: This state waits until the multi-mode command is sent to raNVMe-IP by asserting raNVMCValid to '1'. After that, it changes to next state, stDCheck.
2) stDCheck: This state checks if there is remained data request in ChFIFO. If ChFIFO is not empty, it changes to the next state - stDTrans. Otherwise, it waits until raNVMe-IP completes all operations and de-asserts raNVMBusy to '0' before returning to stDIdle.
3) stDTrans: This state waits until 4Kbyte data is completely transferred between the active user and raNVMe-IP. rDataCnt, the output of data counter, is monitored to read total amount of transferred data. After finishing transferring 4Kbyte data (256 cycles of 128-bit data), it changes to stDDelay.
4) stDDelay: This state asserts Read ACK to ChFIFO after finishing the operation to get the next DataChSel. After that, it returns 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.

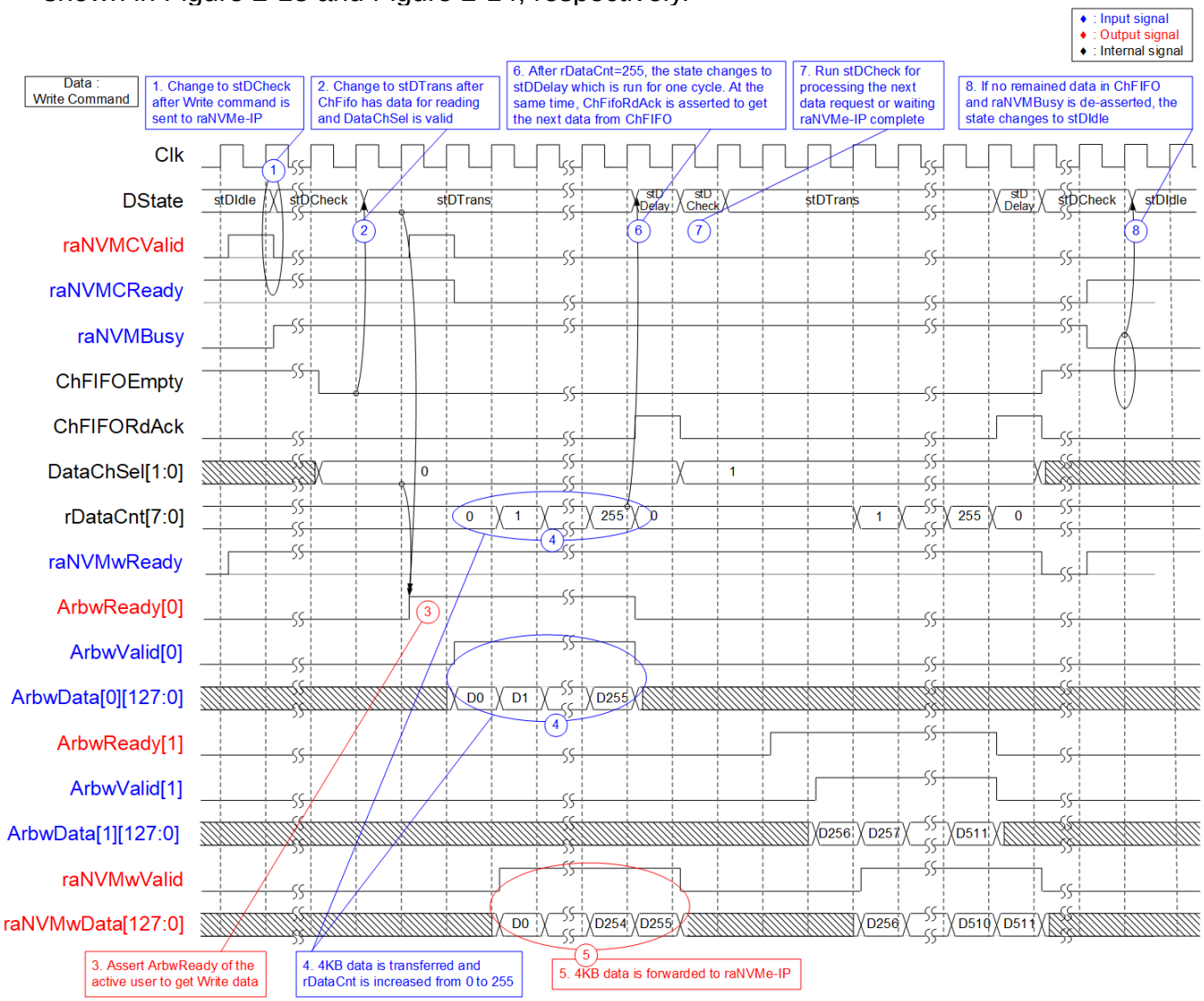

Figure 2-13 Timing diagram of raArbiter's Data I/F when running Write command

1) In stDIdle, it waits until multi-mode 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 sends Write command and then user#1 sends Write command.
2) After ChFIFO is written, ChFIFOEmpty is de-asserted to '0'. Therefore, DataChSel which is the read data of ChFIFO 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 amount 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 transferring the final data (rDataCnt=255), the state changes to stDDelay which is run for one cycle.
7) In stDDelay, ChFIFORdAck is asserted to '1' to flush the current read data and get the next data. 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 active user. Otherwise, it waits until raNVMe-IP completes all operations and continues to step 8).
8) If no remained data in ChFIFO and raNVMBusy is de-asserted to '0', it returns to stDIdle.

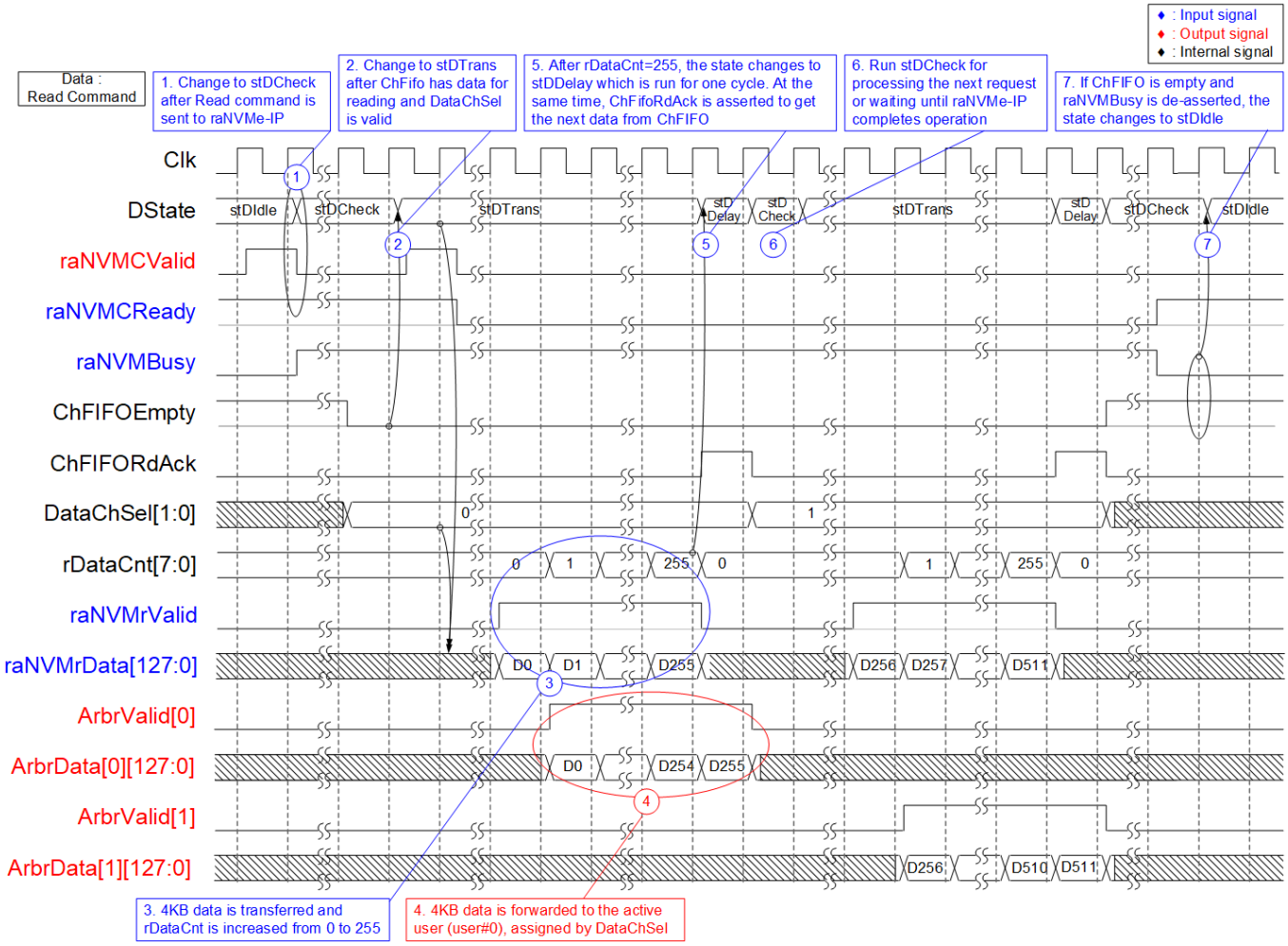

Figure 2-14 Timing diagram of raArbiter's Data I/F when running Read command

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 along with the 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 final 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 the next step 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 of 2-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 user#0 – user#3. rCmdChSel is applied to be 2-upper bits of the address for accessing CtmRAM and IdenRAM.

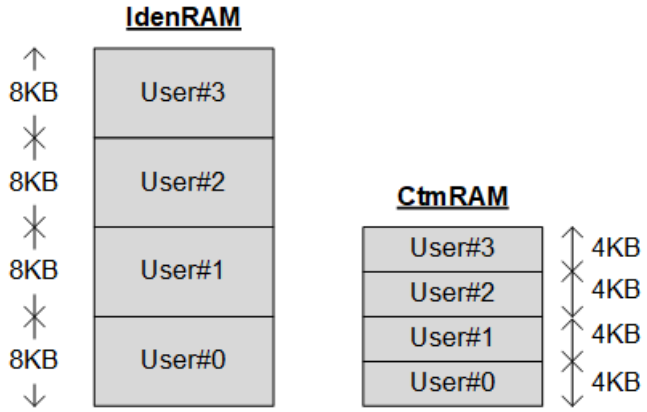

Figure 2-15 IdenRAM and CtmRAM

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. Therefore, CtmRAM size in the reference design is finally equal to 16 Kbytes.

raNVMe-IP and Avl2Reg have the different data bus size, 128-bit on raNVMe-IP but 32-bit on Avl2Reg. Consequently, two user interfaces of IdenARM and CtmRAM have the different bus size for connecting with raNVMe-IP and Avl2Reg. Also, raNVMe-IP has double word enable to write only 32-bit data in some cases. The RAM setting on IP catalog of QuartusII supports the write byte enable, so one of double word enable is extended to be 4-bit write byte enable as shown in Figure 2-16.

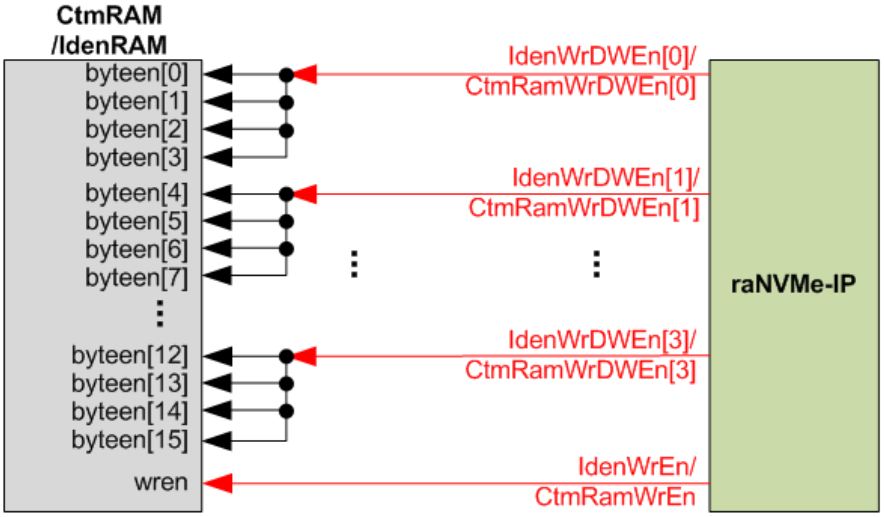

Figure 2-16 Byte write enable conversion logic

Bit[0], [1], [2], and [3] of WrDWEn are fed to be bit[3:0], bit[7:4], bit[11:8], and bit[15:12] of IdenRAM write byte enable, respectively.

Comparing with IdenRAM, CtmRAM is implemented by two-port RAM which has two read ports and two write ports with byte write enable. The data width of both interfaces generated by IP catalog is 128-bit. Double-word enable is extended to be 4-bit write byte enable, similar to IdenRAM. Two write/read port RAM is used for supporting the customized custom command that needs the data input. To support SMART command, using two-port RAM which has one write port and one read port is enough.

2.3 NVMe

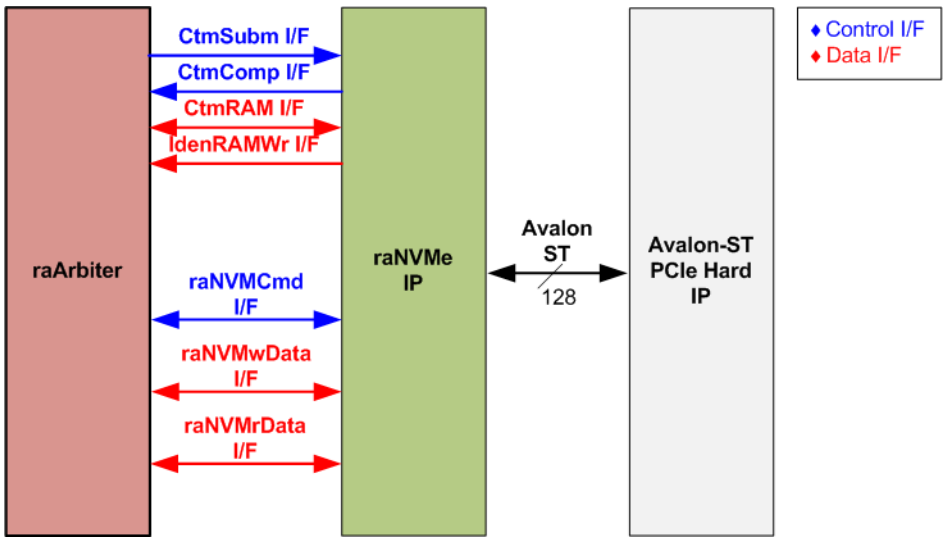

Figure 2-17 NVMe hardware

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 commands 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_ranvmeip_datasheet_intel.pdf

2.3.2 Avalon-ST PCIe Hard IP

This block is the hard IP in Intel FPGA device which implements Physical, Data Link, and Transaction Layers of PCIe specification. More details are described in Intel FPGA document.

Intel Arria10 and Intel Cyclone 10 GX Avalon-ST Interface for PCIe Solutions User Guide
https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_a10_pcie_avst.pdf

2.4 CPU and Peripherals

32-bit Avalon-MM bus is applied to be the bus interface for CPU accessing the peripherals such as Timer and JTAG UART. The test system for running with multiple users by raNVMe-IP is connected with CPU as a peripheral on 32-bit Avalon-MM 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 individually.

In the reference design, the CPU system is built with one additional peripheral to access the test logic. The hardware logic must be designed to support Avalon-MM bus standard for CPU writing and reading. Avl2Reg module is the interface module with the CPU system as shown in Figure 2-18.

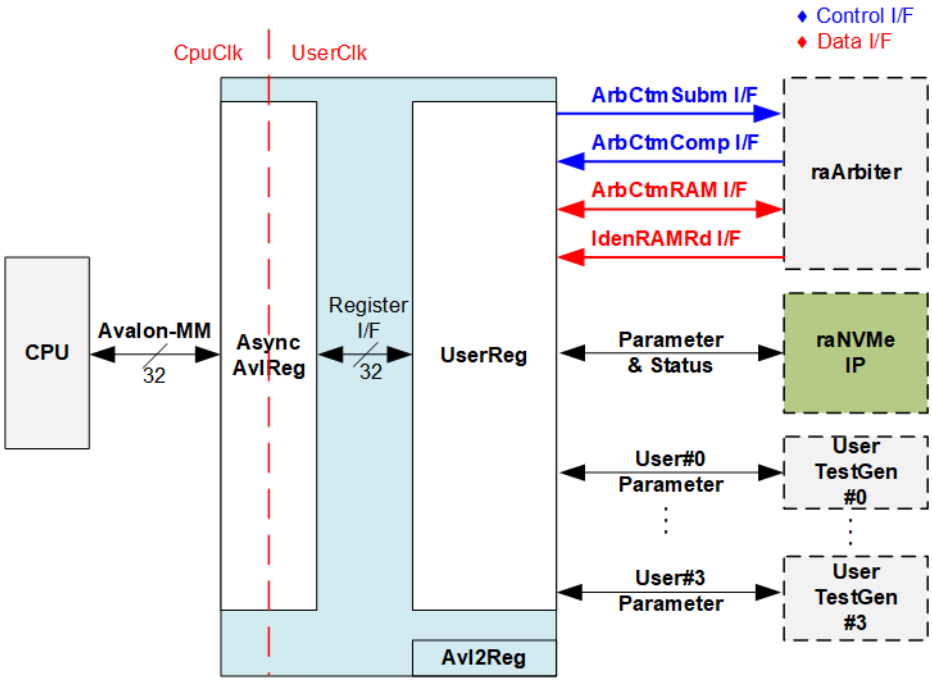

Figure 2-18 CPU and peripherals hardware

Avl2Reg consists of AsyncAvlReg and UserReg. AsyncAvlReg is designed to convert the Avalon-MM signals to be the simple register interface which has 32-bit data bus size, similar to Avalon-MM data bus size. In addition, AsyncAvlReg includes asynchronous logic to support clock domain 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 AsyncAvlReg and UserReg are described as follows.

2.4.1 AsyncAvlReg

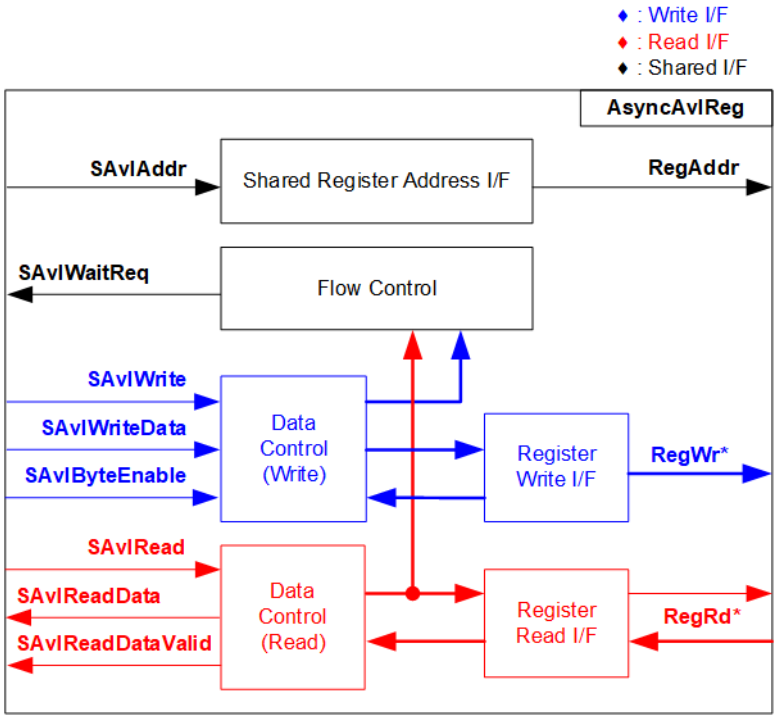

Figure 2-19 AsyncAvlReg Interface

The signal on Avalon-MM bus interface can be split into three groups, i.e., Write channel (blue color), Read channel (red color), and Shared control channel (black color). More details of Avalon-MM interface specification are described in following document https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/manual/mnl_avalon_spec.pdf

According to Avalon-MM specification, one command (write or read) can be operated at a time. The logics inside AsyncAvlReg are split into three groups, i.e., Write control logic, Read control logic, and Flow control logic. Flow control logic controls SAvlWaitReq to hold the next request from Avalon-MM interface if the current request does not finish. Write control and Write data I/F of Avalon-MM bus are latched and transferred to be Write register interface with clock domain crossing registers. Similarly, Read control I/F are latched and transferred to be Read register interface. After that, the returned data from Register Read I/F is transferred to Avalon-MM bus by using clock domain crossing registers. Address I/F of Avalon-MM is latched and transferred to Address register interface as well.

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.

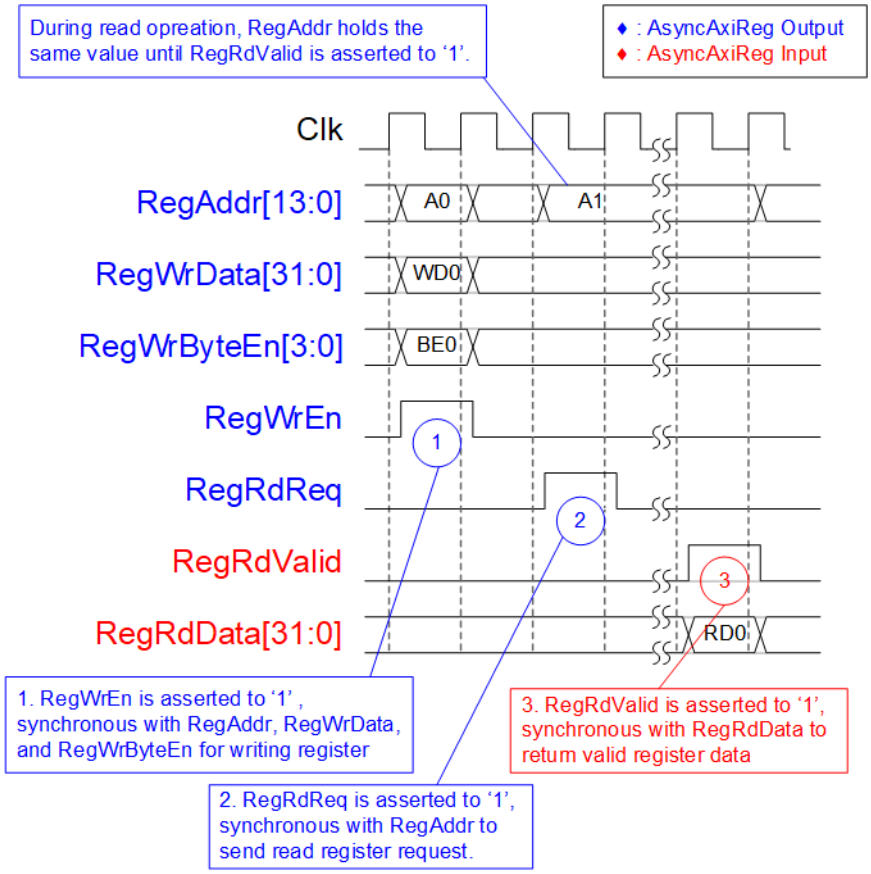

Figure 2-20 Register interface timing diagram

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, AsyncAvlReg asserts RegRdReq to '1' with the valid value of RegAddr. 32-bit data is returned after receiving the read request. The slave detects RegRdReq signal and starts the read transaction. During 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 layers of multiplexer.
3) The read data is returned on RegRdData bus by the slave with asserting RegRdValid to '1'. After that, AsyncAvlReg forwards the read value to SAvlRead interface.

2.4.2 UserReg

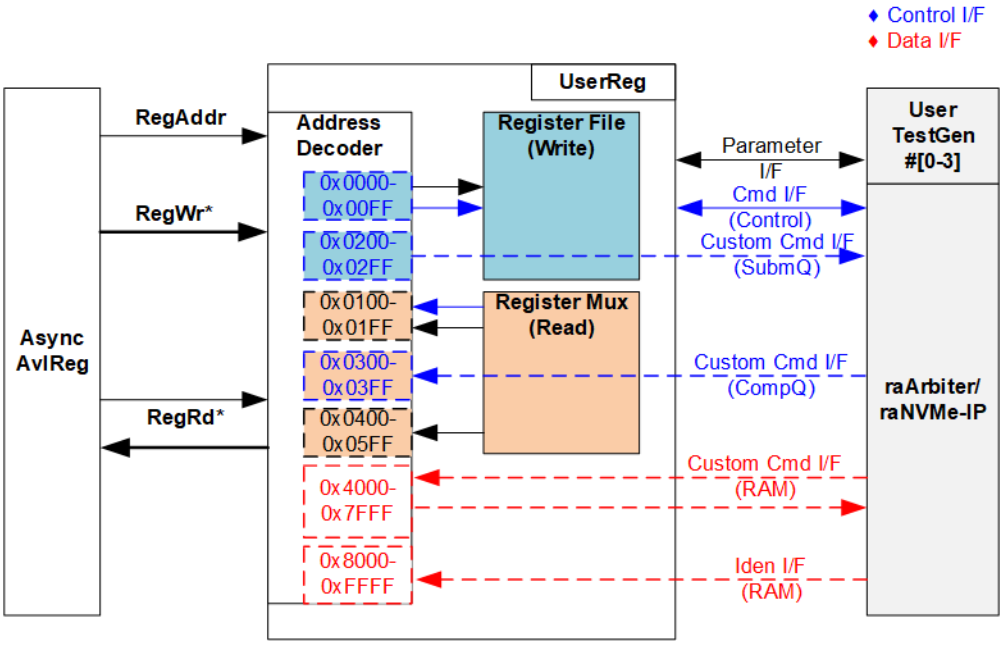

Figure 2-21 UserReg Interface

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 modules. 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 modules. 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 access 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. Therefore, write byte enable (RegWrByteEn) is not applied in the test system and the CPU uses 32-bit pointer to set the hardware register.

To read register, three-step multiplexer 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 active data. Next, the address decoder uses the upper bit to select the read data from active area and returns the data to CPU. Totally, the latency time 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 "ranvmemulttest.c")	Description
0x0000 – 0x00FF: Test parameters of UserTestGen and raNVMe-IP (Write access only)		
BA+0x0000	User#0 Start address (Low) Reg (USR0ADRL_INTREG)	[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_INTREG)	[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_INTREG)	[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_INTREG)	[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_INTREG)	[2:0]: Input to be command of User#0, ArbCmd[0][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_INTREG)	[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_INTREG, USR1LENL/H_INTREG, and USR1CMD_INTREG
BA+0x0040 – BA+0x0057	User#2 Test parameters	Similar to 0x0000 – 0x0017 which is User#0 Test parameters, defined to be USR2ADRL/H_INTREG, USR2LENL/H_INTREG, and USR2CMD_INTREG
BA+0x0060 – BA+0x0077	User#3 Test parameters	Similar to 0x0000 – 0x0017 which is User#0 Test parameters, defined to be USR3ADRL/H_INTREG, USR3LENL/H_INTREG, and USR3CMD_INTREG
BA+0x0080	NVMe Timeout Reg (NVMTIMEOUT_INTREG)	[31:0]: Timeout value of raNVMe-IP (TimeOutSet[31:0] of raNVMe-IP)

Address	Register Name (Label in the "ranvmemulttest.c")	Description
0x0100 – 0x01FF: Status signals of raNVMe-IP and UserTestGen (Read access only)		
BA+0x0100	User Status Reg (USRSTS_INTREG)	[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_INTREG)	[31:0]: Mapped to LBASize[31:0] of raNVMe-IP
BA+0x0108	Total disk size (High) Reg (LBASIZEH_INTREG)	[15:0]: Mapped to LBASize[47:32] of raNVMe-IP
BA+0x010C	User Error Type Reg (USRERRTYPE_INTREG)	[31:0]: Mapped to UserErrorType[31:0] of raNVMe-IP to show error status
BA+0x0110	PCIe Status Reg (PCIESTS_INTREG)	[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) [12:8]: Current LTSSM State of PCIe hard IP. Please see more details of LTSSM value in Avalon-ST PCIe Hard IP datasheet
BA+0x0114	NVMe CAP Reg (NVMCAP_INTREG)	[31:0]: Mapped to NVMeCAPReg[31:0] of raNVMe-IP
BA+0x0118	Admin Completion Status Reg (ADMCOMPSTS_INTREG)	[15:0]: Mapped to AdmCompStatus[15:0] of raNVMe-IP to show status of Admin completion
BA+0x011C	IO Completion Status Reg (IOCOMPSTS_INTREG)	[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_INTREG))	[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_STRUCT)	[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_STRUCT)	[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 “ranvmemulttest.c”)	Description
0x0400 – 0x05FF: Verification details and completed count of UserTestGen (Read access only)		
BA+0x0400 – BA+0x040F	User#0 Expected value Word0-3 Reg (U0EXPPATW0-W3_INTREG)	128-bit expected data of User#0 at the 1st failure data in Read command. 0x0400: Bit[31:0], 0x0404: Bit[63:32], 0x0408: Bit[95:64], 0x040C: Bit[127:96]
BA+0x0410 – BA+0x041F	User#0 Read value Word0-3 Reg (U0RDPATW0-W3_INTREG)	128-bit read data of User#0 at the 1st failure data in Read command. 0x0410: Bit[31:0], 0x0414: Bit[63:32], 0x0418: Bit[95:64], 0x041C: Bit[127:96]
BA+0x0420	User#0 Data Failure Address (Low) Reg (U0RDFAILNOL_INTREG)	[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_INTREG)	[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_INTREG)	[31:0]: Bit[31:0] of the completed command count in User#0, UsrCompCnt
BA+0x042C	User#0 Completed Count (High) Reg (U0CMDCMPCNTH_INTREG)	[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_INTREG, U1RDPATW0-W3_INTREG, U1RDFAILNOL/H_INTREG, and U1CMDCMPCNTL/H_INTREG
BA+0x0500- BA+0x052F	User#2 Verification details	Similar to 0x0400 – 0x042F which is User#0 Verification details, defined to U2EXPPATW0-W3_INTREG, U2RDPATW0-W3_INTREG, U2RDFAILNOL/H_INTREG, and U2CMDCMPCNTL/H_INTREG
BA+0x0580- BA+0x05AF	User#3 Verification details	Similar to 0x0400 – 0x042F which is User#0 Verification details, defined to U3EXPPATW0-W3_INTREG, U3RDPATW0-W3_INTREG, U3RDFAILNOL/H_INTREG, and U3CMDCMPCNTL/H_INTREG
0x8000 – 0xFFFF: IP Version and RAM		
BA+0x0800 Rd	IP Version Reg (IPVERSION_INTREG)	[31:0]: Mapped to IPVersion[31:0] of raNVMe-IP
BA+0x4000 – BA+0x7FFF Wr/Rd	Custom command RAM (CTMRAM_CHARREG)	Mapped 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_CHARREG)	Mapped to 32K byte IdenRAM interface for storing Identify data of User#0-User#3, 8K byte for each user. 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 (ranvmemulttest.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 (PCIESTS_INTREG[0]='1').
3) CPU waits until raNVMe-IP completes initialization process (USRSTS_INTREG[0]='0'). If some errors are found, the process stops with displaying the error message.
4) CPU displays PCIe link status (the number of PCIe lanes and the PCIe speed) by reading PCIESTS_INTREG[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 the 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 the 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_INTREG ="000" (Identify command) when (X) is the current user number. Next, the command request is asserted to UserTestGen and busy flag of the current user (USRSTS_INTREG[8+(X)] when (x) is the current user number) changes from '0' to '1'.
   Note: Bit[8], [9], [10], and [11] of USRSTS_INTREG 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_INTREG[1] and USRSTS_INTREG[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_INTREG[31:0]. Finally, the process is stopped.
   Bit[8+(X)] is de-asserted to '0' after finishing operating the command. Next, 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_CHARREG) such as SSD model name and the information from raNVMe-IP output such as SSD capacity (LBASIZEH/L_INTREG) on the console. 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 the 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.
   i) 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).
   ii) Start address value of the next user is equal to start address of the first user + the transfer length of the first user. Transfer length is RoundUpwithalign8(Remained transfer length/the number of remained users).

   For example, when the 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#0: Start address = 0x1210, Length = RoundupAlign8((0x310-0x108-0x108)/1) = 0x100

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

5) Set USR(X)CMD_INTREG[2:0]=”010” (Write) or “011” (Read) 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_INTREG[1] and USRSTS_INTREG[11:4]. Display the error message when some bits are asserted to ‘1’.
   Note: Bit[4], [5], [6], and [7] of USRSTS_INTREG 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 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 the operation of all users is done. After that, CPU goes to the next step.

   During running command, current amount of transferred data 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_INTREG with multiplying by 4Kbyte. Also, total amount of data is calculated from the sum of the amount of data of each user for displaying 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_STRUCT) to be SMART command value.
5) Set USR(X)CMD_INTREG[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_INTREG[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_INTREG[1] and USRSTS_INTREG[8+(X)].
   Note: Bit[8], [9], [10], and [11] of USRSTS_INTREG 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_INTREG[31:0]. Finally, the process is stopped.
   Bit[8+(X)] is de-asserted to ‘0’ after the operation is done. Next, 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_INTREG) such as Remaining Life, Percentage Used, Temperature, Total Data Read, Total Data Written, Power On Cycles, Power On Hours, and Number of Unsafe Shutdown. The address to access CtmRAM is calculated by 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_STRUCT) to be Flush command value.
5) Set USR(X)CMD_INTREG[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_INTREG[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_INTREG[1] and USRSTS_INTREG[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_INTREG[31:0]. Finally, the process is stopped.
   Bit[8+(X)] is de-asserted to ‘0’ after the operation is done. Next, 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_INTREG[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_INTREG[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_INTREG[1] and USRSTS_INTREG[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_INTREG[31:0]. Finally, the process is stopped.
   Bit[8+(X)] is de-asserted to ‘0’ after the operation is done. Next, CPU goes to the next step.

5) After Shutdown command, the SSD and raNVMe-IP change to inactive status. The CPU cannot receive more commands. 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 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 and validate total number of users and the start user number. 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.

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

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

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

void show_pciestat(void)	
Parameters	None
Return value	None
Description	Read PCIESTS_INTREG 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_INTREG 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_INTREG 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_INTREG (error byte address), U[user_no]EXPPATW0-W3_INTREG (expected value), and U[user_no]RDPATW0-W3_INTREG (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 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 performance comparison when running the demo by using 280 GB Intel 900P on A10GX board (PCIe Gen3) between 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.

From the results of two environments, 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
1.1	31-May-22	Update UserReg and Firmware
1.0	3-Mar-21	Initial Release