FAT32-IP for NVMe (PCIe Gen4) reference design manual

Rev1.0 13-Nov-23

1 Introduction

In a typical data acquisition system, it needs to select the data format stored in the SSD to be raw data or file system. It is known that using raw data provides better performance due to direct access and sequential data transfer. However, the file system provides better convenience in categorizing data types when multiple data types are stored on the same SSD.

Among the well-known file systems, FAT32 file system is widely supported by various operation systems. Typically, the FAT32 file system is designed using software executed on the CPU. Unfortunately, there is currently no solution available for implementing the file system without CPU involvement.

In scenarios where a CPU-less system requires a file system or high-performance is a critical requirement, the system designers select to develop customized file systems. However, this approach leads to compatibility issues with other systems.

To address the performance and compatibility challenges, as well as using the CPU-less systems, the FAT32-IP has been developed using pure hardwired logic on the FPGA platform. By integrating this IP with the NVMe IP, high-performance data transfer can be achieved under a file system format without utilizing the CPU. However, it is important to note that the FAT32 standard has limitations on file size, supporting up to 4GB, and the maximum partition size, which is of 2 TB. The FAT32-IP uses one partition for accessing an NVMe SSD, so the maximum storage size is 2 TB. If larger file size or storage size is required, an alternative IP operation called exFAT-IP is available. Please refer to our website for more detailed information.

In the FAT32-IP for NVMe (PCIe Gen4) reference design, the IP achieves high-performance data transfer for both write file and read file commands, comparable to raw data access. The results demonstrate a write file speed of 6900 MB/s and a read file speed of 7500 MB/s. Further details of the FAT32-IP reference design can be found in the next topic.

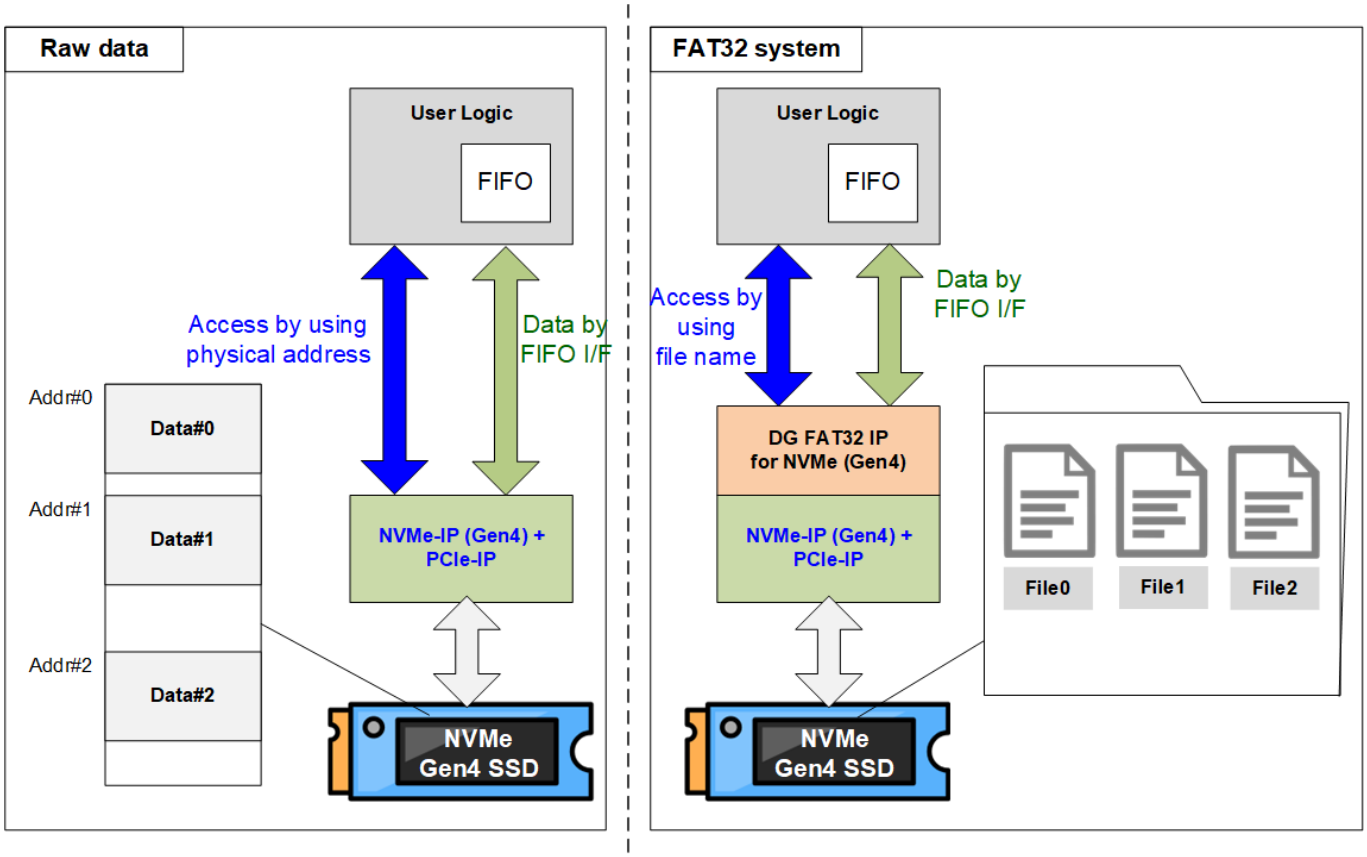

Figure 1-1 Hardware system for raw data and file system

When comparing raw data access via the NVMe-IP user interface with the FAT32 user interface, they use different data indexes. The FAT32 user interface requires inputs in a file parameter format, while the NVMe-IP relies on physical parameters. To specify the start position, the FAT32-IP uses the file name instead of the physical address of the SSD. Similarly, the transfer size is determined by the number of files rather than the transfer length measured in 512-byte unit. However, the data interface for both the NVMe-IP and the FAT32-IP remains similar.

2 Hardware overview

The reference design of FAT32-IP for NVMe (Gen4) has been modified based on the NVMe-IP (Gen4) reference design, incorporating the FAT32-IP. When compared to the standard NVMe-IP (Gen4) design, the key modifications are as follows.

1) The TestGen module has been updated to adapt the control signals from physical parameters to file parameters.
2) The registers within LAxi2Reg have been modified to accommodate the file parameters of FAT32-IP.
3) The CPU firmware has been revised to receive user parameters in file formats.

For further information of the NVMe-IP (Gen4) reference design, please refer to the following link.

- NVMe-IP (Gen4) with PCIe hard IP

https://dgway.com/products/IP/NVMe-IP/dg_nvmeip_refdesign_g4_en/

- NVMe-IP with PCIe Gen4 Soft IP operating with PCIe PHY IP

https://dgway.com/products/IP/NVMe-IP/dg_nvmeg4ip_refdesign_xilinx_en/

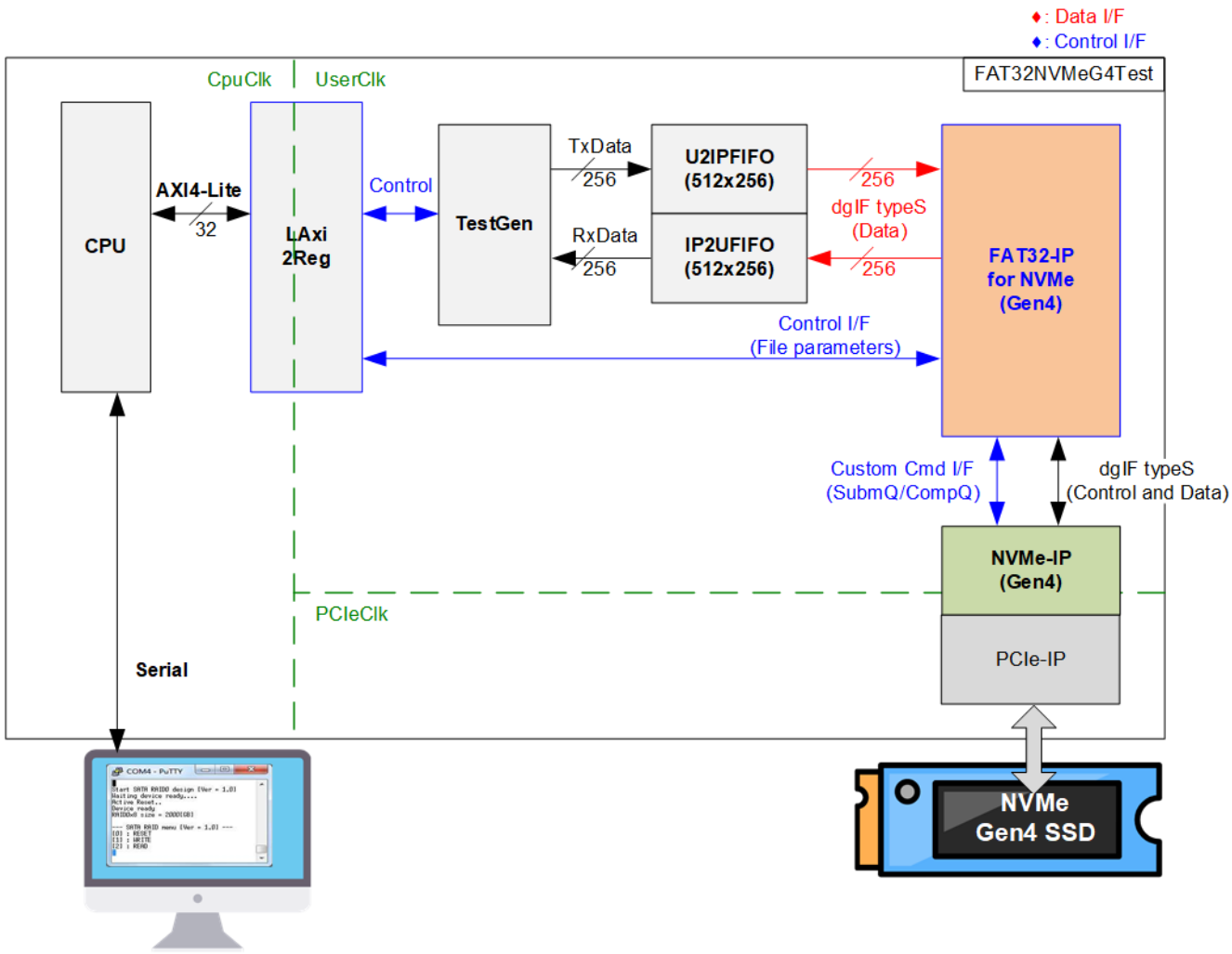

Figure 2-1 FAT32-IP for NVMe (Gen4) demo system

FAT32-IP supports four commands: Format, Write file, Read file, and Shutdown. The transfer performance is displayed as a test result on the Serial console after completing the Write file and Read file commands. Once the Write file command is completed, the user can verify the file on the NVMe SSD by connecting it to other hosts that support the FAT32 system.

The system operates using three clock domains: CpuClk, UserClk, and PCIeClk.

1) CpuClk is the clock domain of the CPU and its peripherals. This clock must provide a stable clock and can differ from other hardware interfaces.
2) UserClk is the clock domain for the user logic, including TestGen and FIFO. It is also fed to FAT32-IP and NVMe-IP (Gen4). According to NVMe-IP (Gen4) datasheet, the frequency of UserClk must be equal to or greater than that of PCIeClk. In this reference design, UserClk is set at 275 MHz.
3) PCIeClk is the clock output from PCIe hard IP to synchronous with data stream of 256-bit AXI4 stream bus. Its frequency is 250 MHz when using 4-lane PCIe Gen4.

Further details of each module within the FAT32-IP reference design are provided below.

2.1 TestGen

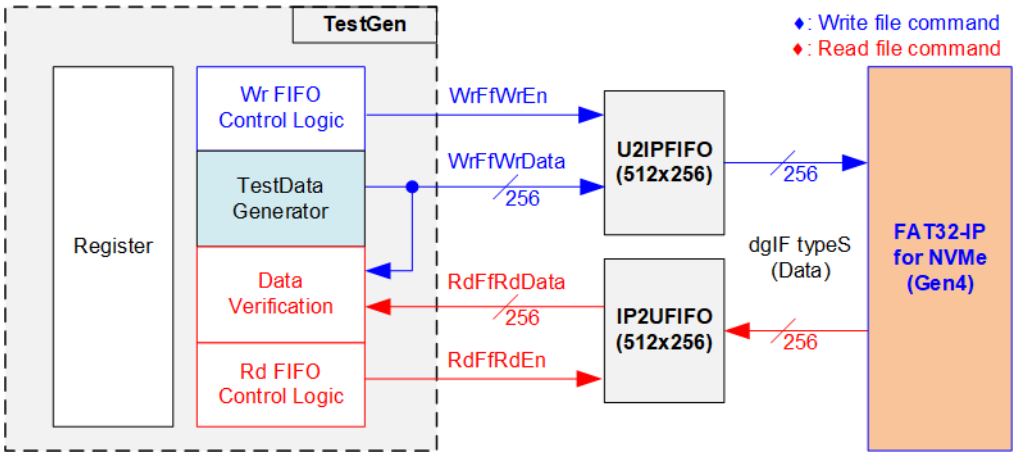

Figure 2-2 TestGen interface

TestGen module handles the data interface of FAT32-IP to transfer data during Write file and Read file commands. In case of a Write file command, TestGen sends 256-bit test data to FAT32-IP via U2IPFIFO. In contrast, for a Read file command, the test data is received from IP2UFIFO for comparison with the expected value, ensuring data accuracy. Data bandwidth of TestGen is set to match that of FAT32-IP by running at the same clock and data bus size. The control logic ensures that the Write or Read enable is always asserted to 1b when the FIFO is ready to write or read data, respectively. This allows FAT32-IP to transfer data through U2IPFIFO and IP2UFIFO without delay, providing the best performance for writing and reading file with the SSD through FAT32-IP.

The Register file in the TestGen receives user-defined test parameters, including file size, file name, the number of files, the command, verification enable, and test pattern selector. Additionally, the internal logic includes a counter to check total transfer size of test data. The detailed hardware logic of TestGen is illustrated in Figure 2-3.

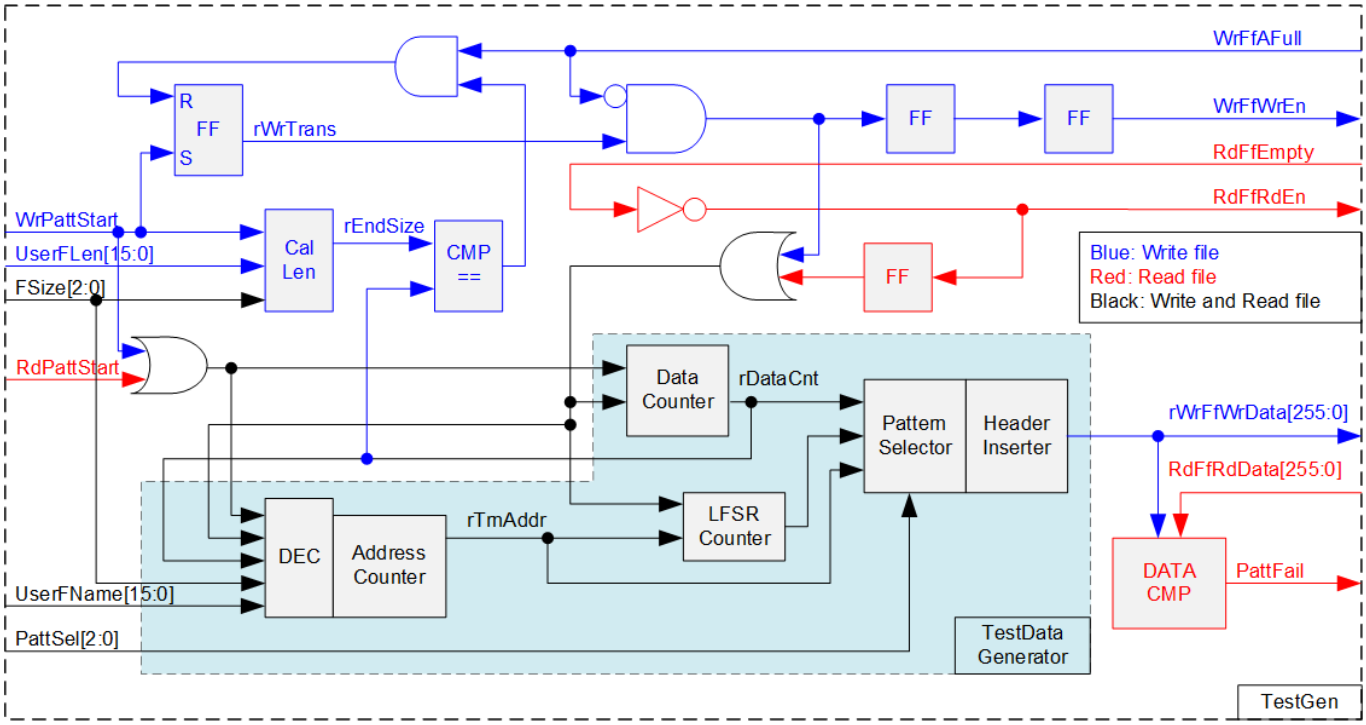

Figure 2-3 TestGen hardware

At the right side of Figure 2-3, the flow control signals of FIFO, WrFfAFull and RdFfEmpty, are used. During write operation, when FIFO is almost full (WrFfAFull=1b), WrFfWrEn is de-asserted to 0b to pause data sending to the FIFO. On the other hand, when there is data in the FIFO during read operation (RdFfEmpty=0b), the logic reads data from the FIFO to compare with the expected data by asserting RdFfRdEn to 1b.

The left side of Figure 2-3 shows the logic designed to count transfer size. Once the total data count is equal to the end size, determined by multiplying the UserFLen value with the decoded file size obtained from the FSize value, the write enable or read enable of the FIFO is de-asserted to 0b. The lower side of Figure 2-3 shows the details to generate test data for writing to the FIFO or verifying data from the FIFO. There are five test patterns available: all-zero, all-one, 32-bit incremental data, 32-bit decremental data, and LFSR, selected by Pattern Selector. When creating an all-zero or all-one pattern, each bit of data is fixed at zero or one, respectively. While other patterns are designed by separating the data into two parts to create unique test data in every 512-byte data, as shown in Figure 2-4.

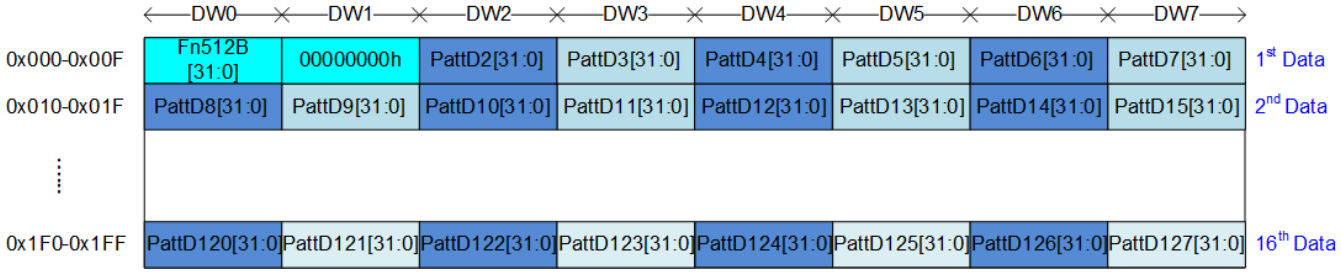

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

In each 512-byte data, the first two double words (Dword#0 and Dword#1) contain a 64-bit header, while the remaining Dwords (Dword#2 – Dword#127) hold the actual test data. The header is generated using the address in 512-byte unit, managed by Address counter block. The initial value of the address counter is calculated by multiplying the UserFName value with the decoded file size obtained from the FSize value. Once the transfer of each 512-byte data is completed, the address counter is incremented. The remaining Dwords (DW#2 – DW#127) depends on the pattern selector, which may be 32-bit incremental data, 32-bit decremental data, or LFSR. The 32-bit incremental data is designed using Data counter, while the decremental data can be designed by connecting NOT logic to incremental data. The LFSR pattern is designed using the LFSR counter. The equation of LFSR is x^31 + x^21 + x + 1.

To implement 256-bit LFSR pattern, the data is split to be two sets of 128-bit data with assigning different initial value. Each 128-bit data uses look-ahead technique to calculate four 32-bit LFSR data in one clock cycle. As shown Figure 2-5, the initial value of LFSR is designed by mixing a part of 512-byte address (LBAAddr) with a part of NOT logic of 512-byte address (LBAAddrB).

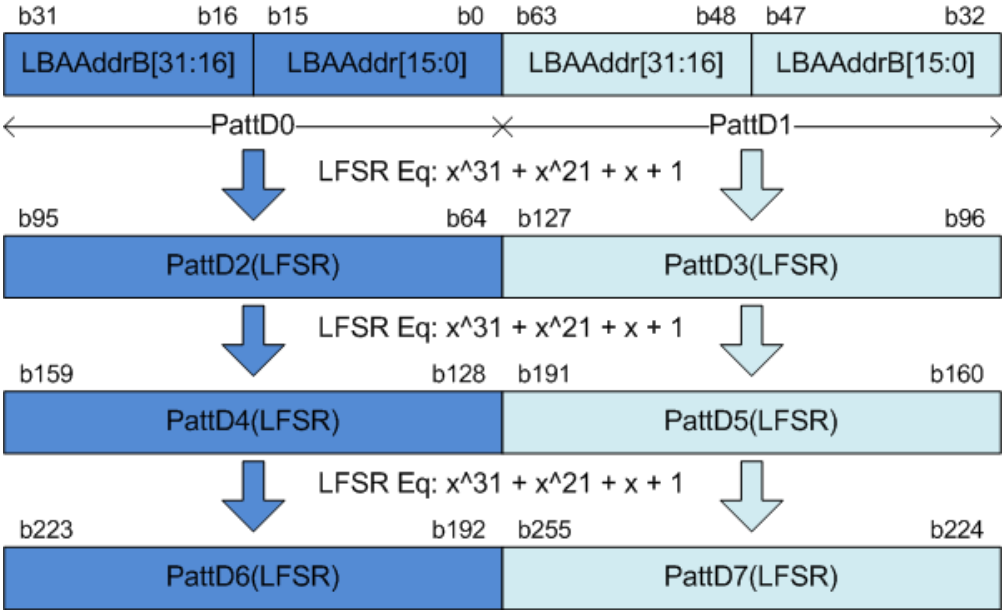

Figure 2-5 256-bit LFSR Pattern in TestGen

Test data is fed to be write data to the FIFO (rWrFfWrData) or to compare with the read data from FIFO (RdFfRdData). PattFail flag is asserted to 1b when data verification fails. The example of timing diagram to write data to FIFO is shown below.

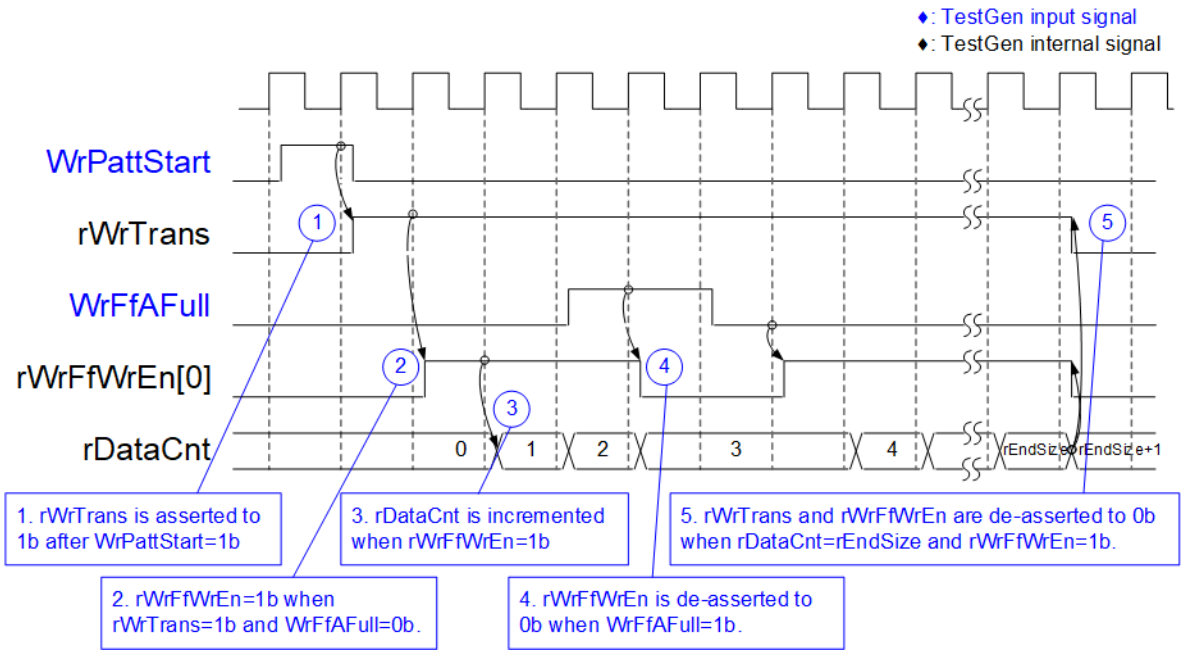

Figure 2-6 Timing diagram of Write operation in TestGen

1) The write operation is initiated by setting WrPattStart signal to 1b for one clock cycle, which is followed by the assertion of rWrTrans to enable the control logic for generating write enable to FIFO.
2) If two conditions are met (rWrTrans is asserted to 1b during the write operation and the FIFO is not full, indicated by WrFfAFull=0b), the write enable (rWrFfWrEn) to FIFO is asserted to 1b.
3) The write enable is fed back to the counter to count the total amount of data in the write operation.
4) If FIFO is almost full (WrFfAFull=1b), the write process is paused by de-asserting rWrFfWrEn to 0b.
5) The write operation is finished when the total data count is equal to the set value. At this point, both rWrTrans and rWrFfWrEn are de-asserted to 0b.

For read transfer, the read enable of FIFO is controlled by the empty flag of FIFO. Unlike the write enable, the read enable signal is not stopped by total data count and not started by start flag. When the read enable is asserted to 1b, the data counter and the address counter are increased for counting the total amount of data and generating the header of expected value, respectively.

2.2 FAT32

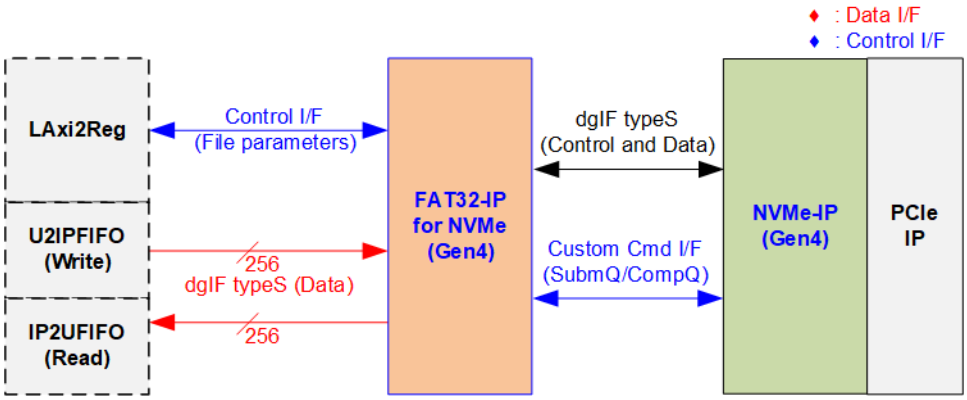

Figure 2-7 FAT32 hardware

In the reference design, the FAT32-IP's user interface consists of a control interface and a data interface. The control interface receives File parameters from LAxi2Reg while the data interface is 256-bit FIFO interface for connecting with U2IPFIFO and IP2UFIFO.

2.2.1 FAT32-IP for NVMe (Gen4)

FAT32-IP for NVMe (Gen4) implements the FAT32 file system by the hardwired logic to allow the user to access an NVMe SSD based on FAT32 file system. It is the add-on module operating with NVMe-IP and PCIe hard IP. Further information about FAT32-IP for NVMe is described in the datasheet, available on our website.
https://dgway.com/products/IP/NVMe-IP/dg_fat32ip_nvmeg4_data_sheet_en/

2.2.2 NVMe-IP (Gen4)

The NVMe-IP (Gen4) implements the NVMe protocol on the host side to direct access an NVMe SSD without PCIe switch connection. It supports six commands: Write, Read, Identify, Shutdown, SMART, and Flush. Design Gateway offers two NVMe-IP (Gen4) options, depending on whether you are using the hard IP or the PCIe soft IP.

The first option is the NVMe-IP designed to work with PCIe Gen4 hard IP. This IP is available for FPGAs that include integrated PCIe Gen4 hard IP. More details of this IP are described in datasheet, available on our website.
https://dgway.com/products/IP/NVMe-IP/dg_nvme_ip_data_sheet_g4_en/

The alternative solution is provided for FPGAs that lack PCIe Gen4 hard IP integration, the NVMe-IP with PCIe Gen4 soft IP. Additional information about this IP can be found from the following document link.
https://dgway.com/products/IP/NVMe-IP/dg_nvmeg4_ip_data_sheet_xilinx_en/

2.2.3 PCIe-IP

There are two distinct types of PCIe-IP options available for connection with each NVMe-IP (Gen4). The first one is the Integrated Block for PCIe for integration with the NVMe-IP (Gen4). Further details about the PCIe hard IP is available in Xilinx's documentation.

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

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

The other option is the PCIe PHY, intended for use with the NVMe-IP integrating with PCIe Gen4 soft IP. For more information, please explore Xilinx's documentation by visiting the following link.
https://docs.xilinx.com/r/en-US/pg239-pcie-phy

2.3 CPU and Peripherals

The CPU system uses a 32-bit AXI4-Lite bus as the interface to access peripherals such as the Timer and UART. The system also integrates an additional peripheral to access FAT32-IP test logic by assigning a unique base address and address range. To support CPU read and write operations, the hardware logic must comply with the AXI4-Lite bus standard. LAxi2Reg module, as shown in Figure 2-8, is designed to connect the CPU system via the AXI4-Lite interface, in compliance with the standard.

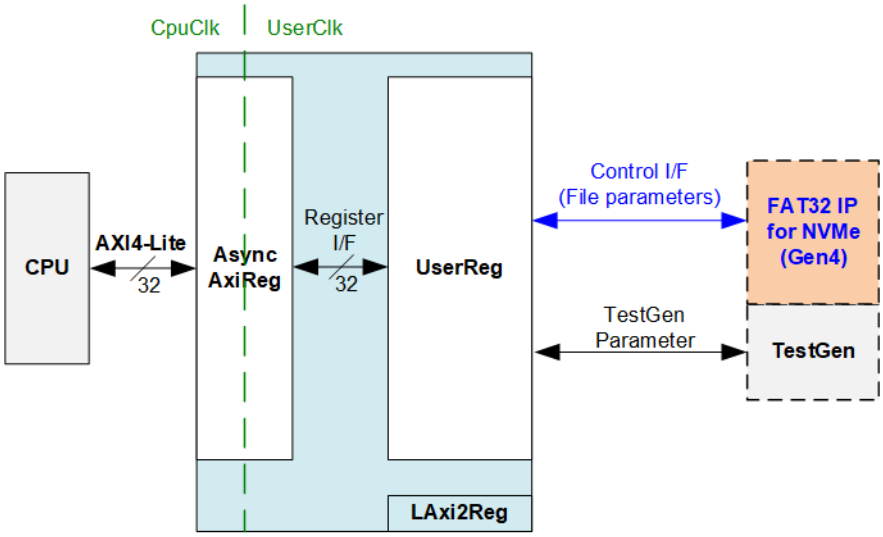

Figure 2-8 CPU and peripherals hardware

LAxi2Reg consists of AsyncAxiReg and UserReg. AsyncAxiReg converts AXI4-Lite signals into a simple Register interface with a 32-bit data bus size, similar to the AXI4-Lite data bus size. It also includes asynchronous logic to handle clock domain crossing between the CpuClk and UserClk domains.

UserReg includes the register file of the parameters and the status signals of other modules in the test system, including the FAT32-IP and TestGen. More details of AsyncAxiReg and UserReg are explained below.

2.3.1 AsyncAxiReg

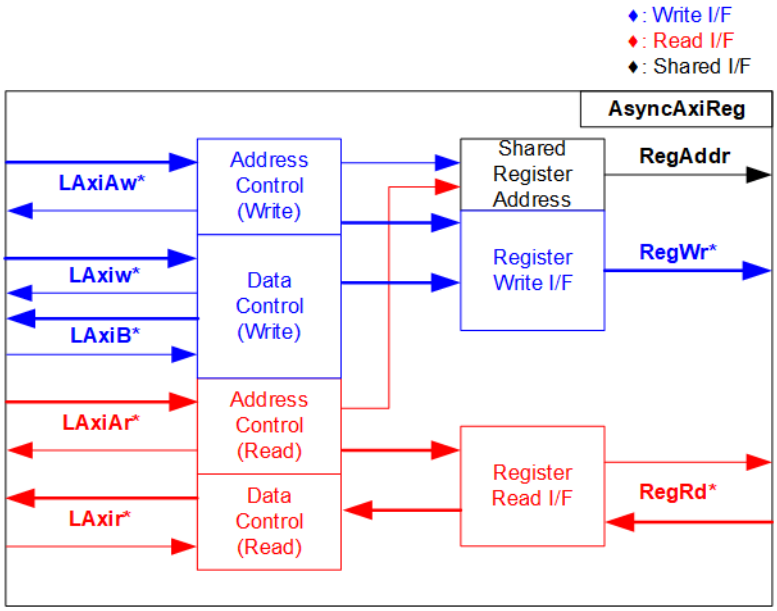

Figure 2-9 AsyncAxiReg Interface

The signal on AXI4-Lite bus interface can be grouped 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://github.com/Architech-Silica/Designing-a-Custom-AXI-Slave-Peripheral/blob/master/designing_a_custom_axi_slave_rev1.pdf

According to AXI4-Lite standard, the write channel and read channel operate independently for both control and data interfaces. Therefore, the logic within AsyncAxiReg to interface with AXI4-Lite bus is divided 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 2-9. The Write control I/F and Write data I/F of the AXI4-Lite bus are latched and transferred to become the Write register interface with clock domain crossing registers. Similarly, the Read control I/F of AXI4-Lite bus is latched and transferred to the Read register interface, while Read data is returned from Register interface to AXI4-Lite via clock domain crossing registers. In the Register interface, RegAddr is a shared signal for write and read access, so it loads the value from LAxiAw for write access or LAxiAr for read access.

The Register interface is compatible with single-port RAM interface for write transaction. The read transaction of the Register interface has been slightly modified from RAM interface by adding the RdReq and RdValid signals to control read latency time. The address of Register interface is shared for both write and read transactions, so user cannot write and read the register at the same time. The timing diagram of the Register interface is shown in Figure 2-10.

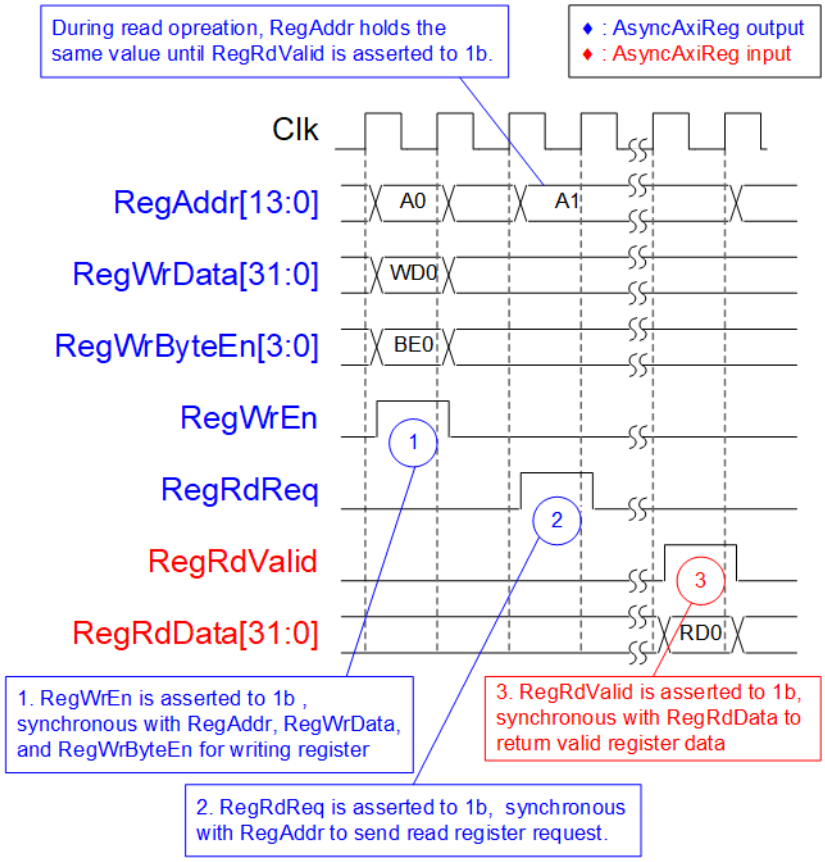

Figure 2-10 Register interface timing diagram

1) Timing diagram to write register is similar to that of a single-port RAM. The RegWrEn signal is set to 1b, along with a valid RegAddr (Register address in 32-bit units), RegWrData (write data for the register), and RegWrByteEn (write byte enable). The byte enable consists of four bits that indicate the validity of the byte data. For example, bit[0], [1], [2], and [3] are set to 1b when RegWrData[7:0], [15:8], [23:16], and [31:24] are valid, respectively.
2) To read register, AsyncAxiReg sets the RegRdReq signal to 1b with a valid value for RegAddr. The 32-bit data is returned after the read request is received. The slave detects the RegRdReq signal being set to start the read transaction. In the read operation, the address value (RegAddr) remains unchanged until RegRdValid is set to 1b. The address can then be used to select the returned data using multiple layers of multiplexers.
3) The slave returns the read data on RegRdData bus by setting the RegRdValid signal to 1b. After that, AsyncAxiReg forwards the read value to the LAxir* interface.

2.3.2 UserReg

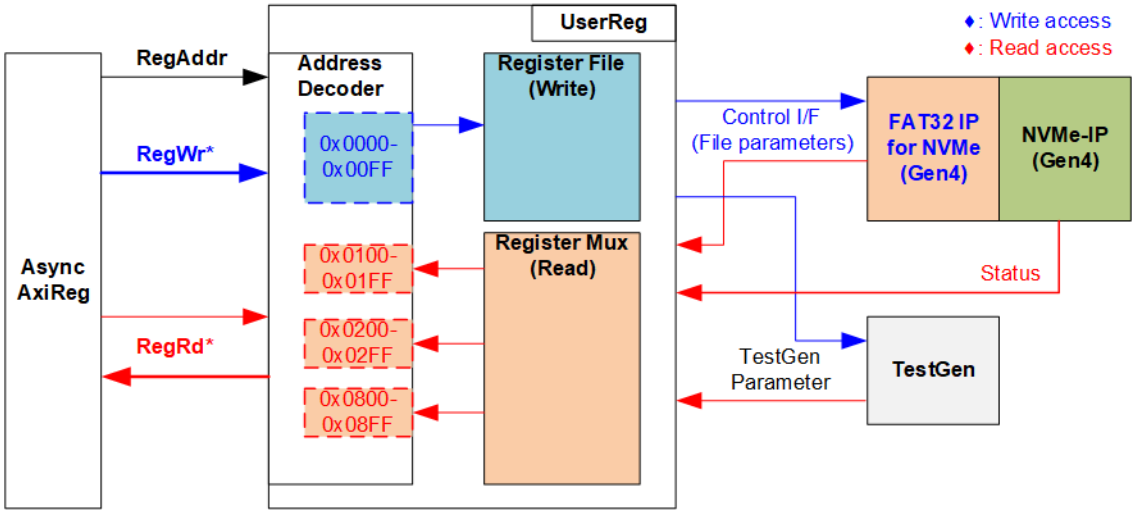

Figure 2-11 UserReg Interface

The UserReg module consists of an Address decoder, a Register File, and a Register Mux. The Address decoder decodes the address requested by AsyncAxiReg and selects the active register for either write or read transactions. The assigned address range in UserReg is divided into six areas, as shown in Figure 2-11.

1) 0x0000 – 0x00FF: Mapped to set the command with the parameters of FAT32-IP and TestGen. This area is write-access only.
2) 0x0100 – 0x01FF: Mapped to read the status signals of FAT32-IP and NVMe-IP. This area is read-access only.
3) 0x0200 – 0x02FF: Mapped to read the status signals of TestGen module. This area is read-access only.
4) 0x0800 – 0x08FF: Mapped to read IP information of FAT32-IP and NVMe-IP. This area is read-access only.

The Address decoder decodes the upper bits of RegAddr to select the active hardware (FAT32-IP, NVMe-IP, or TestGen). The Register File within UserReg has a 32-bit bus size, so the write byte enable (RegWrByteEn) is not used in the test system and the CPU uses 32-bit pointer to set the hardware register.

For reading a register, multi-level multiplexers (mux) select the data to return to CPU by using the address. The lower bits of RegAddr are fed to the submodule to select the active data from each submodule. While the upper bits are used in UserReg to select the returned data from each submodule. The total latency time of read data is equal to two clock cycles, and RegRdValid is created by RegRdReq by asserting two D Flip-flops. More details of the address mapping within the UserReg module are shown in Table 2-1

Table 2-1 Register Map

Address Rd/Wr	Register Name (Label in the “fat32nvmeg4test.c”)	Description
0x0000 – 0x00FF: Control signals of FAT32-IP and TestGen (Write access only)		
BA+0x000	User File Name Reg (USRFNAME_INTREG)	[15:0]: Input to be UserFName of FAT32-IP for NVMe (Gen4)
BA+0x004	User File Length Reg (USRFLEN_INTREG)	[15:0]: Input to be UserFLen of FAT32-IP for NVMe (Gen4)
BA+0x008	File Size Reg (USRFSIZE_INTREG)	[2:0]: Input to be FSize of FAT32-IP for NVMe (Gen4). During the operation, this value must not be modified. If it needs to update this value, it needs to execute Format command.
BA+0x00C	Created Date and Time Reg (DATETIME_INTREG)	[4:0]: Input to be FTimeS of FAT32-IP for NVMe (Gen4) [10:5]: Input to be FTimeM of FAT32-IP for NVMe (Gen4) [15:11]: Input to be FTimeH of FAT32-IP for NVMe (Gen4) [20:16]: Input to be FDateD of FAT32-IP for NVMe (Gen4) [24:21]: Input to be FDateM of FAT32-IP for NVMe (Gen4) [31:25]: Input to be FDateY of FAT32-IP for NVMe (Gen4)
BA+0x010	User Command Reg (USRCMD_INTREG)	[1:0]: Input to be UserCmd of FAT32-IP for NVMe (Gen4) When this register is written, UserReq of FAT32-IP for NVMe (Gen4) is asserted. The command execution is initiated.
BA+0x014	Pattern Selector Reg (PATTSEL_INTREG)	[2:0]: Select test pattern 000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1, 100b-LFSR.
0x0100 – 0x01FF: Status signals of FAT32-IP (Read access only)		
BA+0x100	User Status Reg (USRSTS_INTREG)	[0]: Mapped to UserBusy of FAT32-IP for NVMe (Gen4) [1]: Mapped to UserError of FAT32-IP for NVMe (Gen4) [2]: Data verification fail (0b: Normal, 1b: Error)
BA+0x104	Total file capacity Reg (TOTALFCAP_INTREG)	[15:0]: Mapped to TotalFCap[15:0] of FAT32-IP for NVMe (Gen4)
BA+0x108	User Error Type Reg (USRERRTYPE_INTREG)	[31:0]: Mapped to UserErrorType[31:0] of FAT32-IP for NVMe (Gen4)
BA+0x110	FAT32 IP Test pin (Low) Reg (TESTPINL_INTREG)	[31:0]: Mapped to TestPin[31:0] of FAT32-IP for NVMe (Gen4)
BA+0x114	FAT32 IP Test pin (High) Reg (TESTPINH_INTREG)	[31:0]: Mapped to TestPin[63:32] of FAT32-IP for NVMe (Gen4)
BA+0x140	Completion Status Reg (COMPSTS_INTREG)	Completion Status from NVMe-IP (Gen4) [15:0]: Admin completion Status (AdmCompStatus[15:0]) [31:16]: I/O completion Status (IOCompStatus[15:0])
BA+0x144	NVMe CAP Reg (NVMCAP_INTREG)	[31:0]: NVMeCAPReg[31:0] output from NVMe-IP (Gen4)
BA+0x150	NVMe Test pin Reg (NVMTESTPIN_INTREG)	[31:0]: Mapped to TestPin[31:0] of NVMe-IP (Gen4)
BA+0x160 – BA+0x16F	MAC Test pin 0-3 Reg (MACTESTPIN0-3_INTREG)	Mapped to MACTestPin[127:0] of NVMeG4-IP (NVMe IP with PCIe Gen4 Soft IP), not available for NVMe-IP with PCIe Hard IP 0x160: bit[31:0], 0x164: bit[63:32], 0x168: bit[95:64], 0x16C: bit[127:96]

Address Rd/Wr	Register Name (Label in the “fat32nvmeg4test.c”)	Description
0x0200 – 0x02FF: Status signals of TestGen (Read access only)		
BA+0x200 – BA+0x21F	Expected value Word0-7 Reg (EXPPATW0-W7_INTREG)	The 256-bit expected data of the 1st failure when executing a Read file command. 0x0200: Bit[31:0], 0x0204[31:0]: Bit[63:32], …, 0x021C[31:0]: Bit[255:224]
BA+0x240 – BA+0x25F	Read value Word0-7 Reg (RDPATW0-W7_INTREG)	The 256-bit read data of the 1st failure when executing a Read file command. 0x0210: Bit[31:0], 0x0214[31:0]: Bit[63:32], …, 0x025C[31:0]: Bit[255:224]
BA+0x280	Failure Byte Address Reg (FAILADDR_INTREG)	[31:0]: Bit[31:0] of the byte index in the file where the 1st failure data is found during Read file command
BA+0x284	Failure File Name Reg (FAILFNAME_INTREG)	[15:0]: Filename which encounters the 1st failure data
BA+0x288	Current test byte (Low) Reg (CURTESTSIZEL_INTREG)	[31:0]: Bit[31:0] of the current test data size in TestGen module
BA+0x28C	Current test byte (High) Reg (CURTESTSIZEH_INTREG)	[8:0]: Bit[40:32] of the current test data size in TestGen module
Other interfaces (Read access only)		
BA+0x800	FAT32-IP Version Reg (FAT32NVMVER_INTREG)	[31:0]: FAT32-IP version number, mapped to IPVersion [31:0] of FAT32-IP for NVMe (Gen4)
BA+0x804	NVMe-IP Version Reg (NVMEVER_INTREG)	[31:0]: NVMe-IP version number, mapped to IPVersion [31:0] of NVMe-IP (Gen4)

3 CPU Firmware

3.1 Test firmware (fat32nvmeg4test.c)

Upon system boot-up, the CPU executes the initialization sequence as outlined below.

1) The CPU initializes its peripherals such as UART and Timer.
2) The CPU awaits the completion of the initialization process by FAT32-IP, indicated by the USRSTS_INTREG[0]=0b.
3) The CPU asks the user to input whether to format the disk or not. If the user selects to format the disk, the next step involves executing the Format command menu, as described in section 3.1.1. Otherwise, the CPU proceeds with the following steps.
   i) The CPU requests the user to configure the file size parameter.
   ii) If an input value is invalid, the CPU continues to request the new parameter until a valid value is provided.
   iii) Once a valid input is received, the new value is set to USRFSIZE_INTREG.
   iv) The console displays the value of the file size and the maximum number of files, and continues to step 4).

4) The consoled displays the main menu, offering four commands for testing purposes: Format command (section 3.1.1), Write file command (section 3.1.2), Read file command (section 3.1.2), and Shutdown command (section 3.1.3). Further operation details of each command are provided below.

3.1.1 Format

When the Format menu is selected, the CPU firmware follows the following sequence.

1) The CPU displays the current file size value on the console and asks the user to keep it or change it. If the user chooses to change file size, a menu to select the file size is displayed, and the CPU awaits the user's input for the new file size value.
2) The CPU assigns the file size value to the USRFSIZE_INTREG register.
3) The CPU sets USRCMD_INTREG=00b to initiate the Format command. As a result, FAT32-IP then changes to busy state, indicated by the change of USRSTS_INTREG[0] from 0b to 1b.
4) The CPU monitors USRSTS_INTREG[1:0] to determine the completion status or identify any errors during the operation.
   - Bit[0] is de-asserted to 0b when the command is completed.
   - Bit[1] is asserted to 1b if any errors are detected. In the event of an error condition, an error message is displayed on the console and the operation is halted.

5) Once the command execution is completed, the maximum number of files that can be stored in the disk, read from TOTALFCAP_INTREG register, is displayed on the console.

3.1.2 Write file/Read file

When the Write file or Read file menu is selected, the CPU firmware follows the following sequence.

1) For the Read file command, the CPU proceeds to the next step. However, for the Write file command, the user is asked to set the created date and created time for the next created file. Alternatively, the latest values can be used. The values are then assigned to the DATETIME_INTREG register.
2) The CPU receives inputs from the user, including the start file number, total number of files, and the desired test pattern. If any inputs are found to be invalid, the operation is cancelled. All received inputs are then assigned to the USRFNAME_INTREG, USRFLEN_INTREG and PATTSEL_INTREG registers, respectively.
3) The CPU sends the request of Write file or Read file command by setting the USRCMD_INTREG register to 10b for Write file command or 11b for Read file command.
4) The CPU monitors USRSTS_INTREG[2:0] to check that the operation is completed or any errors (except verification error) occur.

   Bit[0] is de-asserted to 0b when the command is completed.
   Bit[1] is asserted to 1b if any errors are detected. In the event of an error condition, an error message is displayed on the console and the operation is halted.
   Bit[2] is asserted if data verification fails. A verification error message is then displayed. The CPU continues to run until the operation is completed or until the user inputs any key to cancel the operation.

   During the command execution, every second the console displays the current transfer size, read from CURTESTSIZEL/H_INTREG register.

5) Once the busy flag (USRSTS_INTREG[0]) is de-asserted to 0b, the CPU displays the test result on the console, including the total time usage, total transfer size, and transfer speed.

3.1.3 Shutdown

When the Shutdown menu is selected, the CPU firmware follows the following sequence.

1) The CPU displays a confirmation message to ensure the user's intention to proceed with the Shutdown operation. If the user confirms, the CPU proceeds to the next step.
2) The CPU sets USRCMD_INTREG=01b to initiate the Shutdown command. As a result, FAT32-IP then changes to busy state, indicated by the change of USRSTS_INTREG[0] from 0b to 1b.
3) The CPU monitors USRSTS_INTREG[1:0] to determine if the operation is completed or if any errors occur.
   Bit[0] is de-asserted to 0b when the command is completed.
   Bit[1] is asserted to 1b if any errors are detected. In the event of an error condition, an error message is displayed on the console and the operation is halted.

4) After the command is completed, the SSD enters to inactive status. Also, the CPU can no longer receive new commands from the user. The user must power off the test system after completing this command.

3.2 Function list in Test firmware

void change_fsize(void)	
Parameters	None
Return value	None
Description	Execute the change file size operation, as outlined in steps 1) – 2) of section 3.1.1 (Format).

void change_ftime(void)	
Parameters	None
Return value	None
Description	Display the current created time and date utilizing the “show_ftime” function. Next, receive the user input to use the same values or change them. If the inputs are valid, update the created time and date to DATETIME_INTREG and assign them to the global parameter (DateTime).

int format_fat(void)	
Parameters	None
Return value	None
Description	Execute the Format command operation, as outlined in steps 3) – 5) of section 3.1.1 (Format).

unsigned long long get_cursize(void)	
Parameters	None
Return value	Read value of CURTESTSIZEH/L_INTREG
Description	Read the value of CURTESTSIZEH/L_INTREG and return its value as the result of the function.

int get_param(userin_struct* userin)	
Parameters	userin: Three inputs set by user: start file number, number of files, and test pattern
Return value	0: Valid input, -1: Invalid input
Description	Display the range of each input parameter, receive the user inputs, and validate the user input value. If it is invalid, the function will return -1. Otherwise, update all inputs to the userin parameter.

void show_error(void)	
Parameters	None
Return value	None
Description	Read the value of USRERRTYPE_INTREG, decode its, and display the corresponding error message, such as a timeout error, NVMe-IP (Gen4) error, or unsupported LBA size.

void show_fsize(void)	
Parameters	None
Return value	None
Description	Display the current disk information retrieved from global parameters which are file size (FsizeMB) and maximum file in the disk (TotalFCap).

void show_ftime(void)	
Parameters	None
Return value	None
Description	Display the current created date and time retrieved from global parameter (DateTime).

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

void show_size(unsigned long long size_input)	
Parameters	Size in byte unit
Return value	None
Description	Display the input value in Mbyte or Gbyte units.

void show_testpin(void)	
Parameters	None
Return value	None
Description	Display TestPin of FAT32-IP for NVMe-IP (Gen4) on the console to facilitate debugging by reading the value of TESTPINL/H_INTREG, NVMTESTPIN_INTREG, and MACTESTPIN0-3_INTREG (only for NVMe-IP with PCIe Gen4 Soft IP).

void show_vererr(void)	
Parameters	None
Return value	None
Description	To show the details regarding verification errors, print information from the hardware registers, including the first error file name (FAILFNAME_INTREG), the first error address (FAILADDR_INTREG), the expected value (EXPPATW0-7_INTREG), and the error read value (RDPATW0-7_INTREG).

int shutdown_dev(void)	
Parameters	None
Return value	0: Shutdown command is finished. -1: User cancels command or error is found.
Description	Execute Shutdown command, as outlined in section 3.1.3 (Shutdown).

int wrrd_file(unsigned int user_cmd)	
Parameters	Command from user (2: Write file command, 3: Read file command)
Return value	0: Operation is successful. -1: Receive invalid input or error is found.
Description	Execute Write file or Read file command as outlined in section 3.1.2 (Write file/Read file).

4 Example Test Result

The example test result when running demo system by using 1 TB Samsung 990 Pro is shown in Figure 4-1.

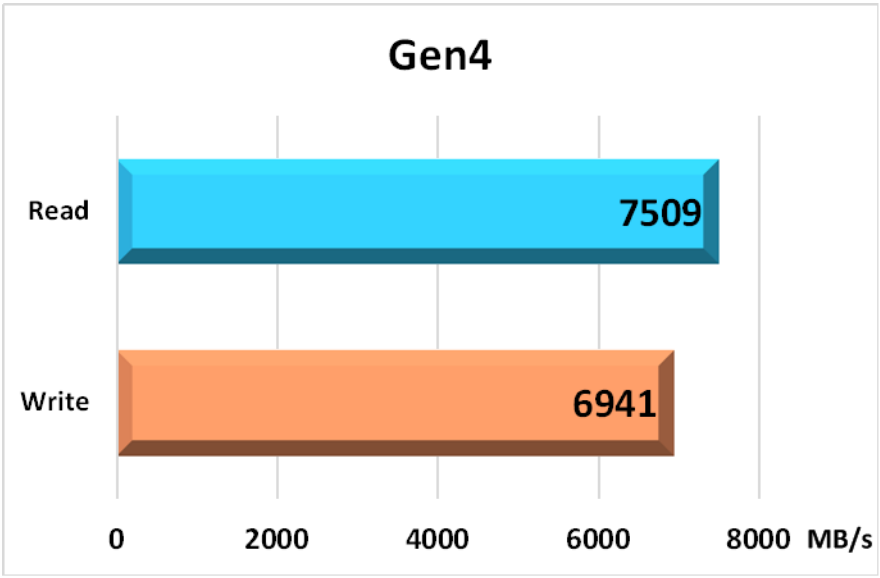

Figure 4-1 Test Performance of FAT32-IP demo for NVMe by using Samsung 990 Pro SSD

By using PCIe Gen4 on ZCU102 board, write performance is about 6,900 Mbyte/sec and read performance is about 7,500 Mbyte/sec.

5 Revision History

Revision	Date	Description
1.0	13-Nov-23	Initial version release