# exFAT IP for NVMe reference design manual

Rev2.00 7-May-24

# 1 Introduction

To store data stream on an NVMe SSD, a high-performance storage solution, users must decide between two data formats: raw data format or file system format. In the raw data format, data indices use the physical address of SSD, and users are responsible for managing data structure themselves, especially when dealing with various data types stored on the SSD. Implementing a customized data structure requires dedicated human resources for development and maintenance, especially when integrating with multiple host systems accessing the SSD. However, this approach offers the best performance for data transfer with the SSD.

On the other hand, standard file systems like FAT32 and exFAT provide a comprehensive solution for treating each piece of data as a 'file'. These file systems are well-designed, utilizing file entries to denote file names, types, sizes, and data allocations. FAT32 and exFAT file systems have been widely implemented by various developers, enabling users to integrate standard libraries into their host systems for accessing files on the SSD. However, this convenience comes at the cost of CPU or processor resources required to manage file entries and file data, which can result in constrained write/read performance during file data access.

To overcome the performance constraints associated with using file system, Design Gateway offers IP cores that implements the FAT32 and exFAT file systems using pure hardware logics, called FAT32 IP and exFAT IP. This approach ensures maximum transfer performance for data transfers with SSD, similar to the raw data format. Further information of FAT32 IP can be found from following site.
https://dgway.com/products/IP/NVMe-IP/dg_fat32ip_nvme_data_sheet_en/

Compared to the FAT32 file system, the exFAT file system offers several improvements. Firstly, exFAT supports file size larger than 4 GB and disk capacities exceeding 2 TB. Additionally, it implements name hashing for file names to enhance search functionality and employs checksums in the system data area to boost data reliability. However, exFAT demands much host resources for managing file systems and some legacy systems support only the FAT32 file system, not exFAT.

The exFAT-IP is an IP core designed to directly interface with the NVMe(G3)-IP, an NVMe host controller IP core from Design Gateway. It achieves the same performance as using the raw data format under PCIe Gen3 speed, with read access reaching up to 3300 MB/s. Figure 1-1 illustrates the block diagram of the reference design, showcasing the comparison between the raw data access without the exFAT-IP and the file system access using the exFAT-IP.

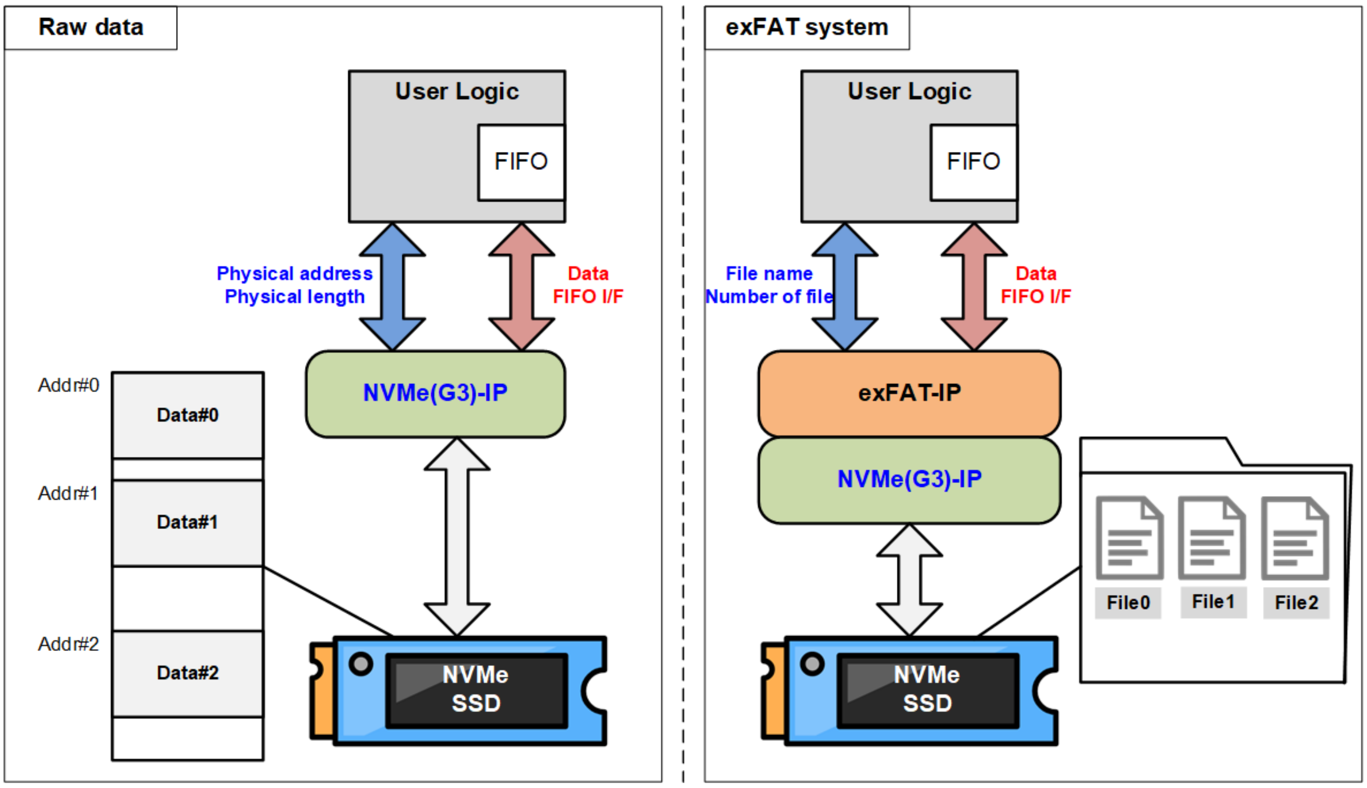

Figure 1-1 Hardware system using NVMe(G3)-IP for raw data and file system

As shown on the left side of Figure 1-1, when utilizing the NVMe(G3)-IP to access an NVMe SSD, the stored data is in raw data format, accessed via physical address. Consequently, the input parameters assigned to the NVMe(G3)-IP consist of physical address and transfer length measured in 512-byte units.

In contrast, on the right side of Figure 1-1, with the exFAT-IP inserted between the user logic and the NVMe(G3)-IP for accessing an NVMe SSD, the data stored on the SSD adopts the exFAT file system format, denoted as File0, File1, and File2. The input parameters designated for the exFAT-IP include file names and the number of files, which are subsequently computed into physical address and 512-byte transfer length supplied to the NVMe(G3)-IP.

Both the raw data system and exFAT file system utilize FIFO interfaces for their data interfaces.

# 2 Hardware overview

The reference design for the exFAT-IP for NVMe builds upon the existing NVMe-IP or NVMeG3-IP reference design by incorporating the exFAT-IP. In this update, the control interface is enhanced from managing physical parameters to handling file parameters. Modifications in the design are highlighted in blue in Figure 2-1.

For more detailed information about the NVMe-IP and NVMeG3-IP reference designs, please refer to the following documents.

NVMe-IP: https://dgway.com/products/IP/NVMe-IP/dg_nvmeip_refdesign_en/
NVMeG3-IP: https://dgway.com/products/IP/NVMe-IP/dg_nvmeg3ip_refdesign_xilinx_en/

Key updates from the NVMe(G3)-IP reference design are as follows.

1) File parameters for the control interface of the exFAT-IP are sourced from the TestGen module.
2) Updates to the LAxi2Reg registers align with the control signals from the test system.
3) CPU firmware is enhanced to accept file parameters from the user, which are then translated into control signals for the hardware via the AXI4-Lite bus. Example parameters include file name, file size, number of files, created date, and created time.

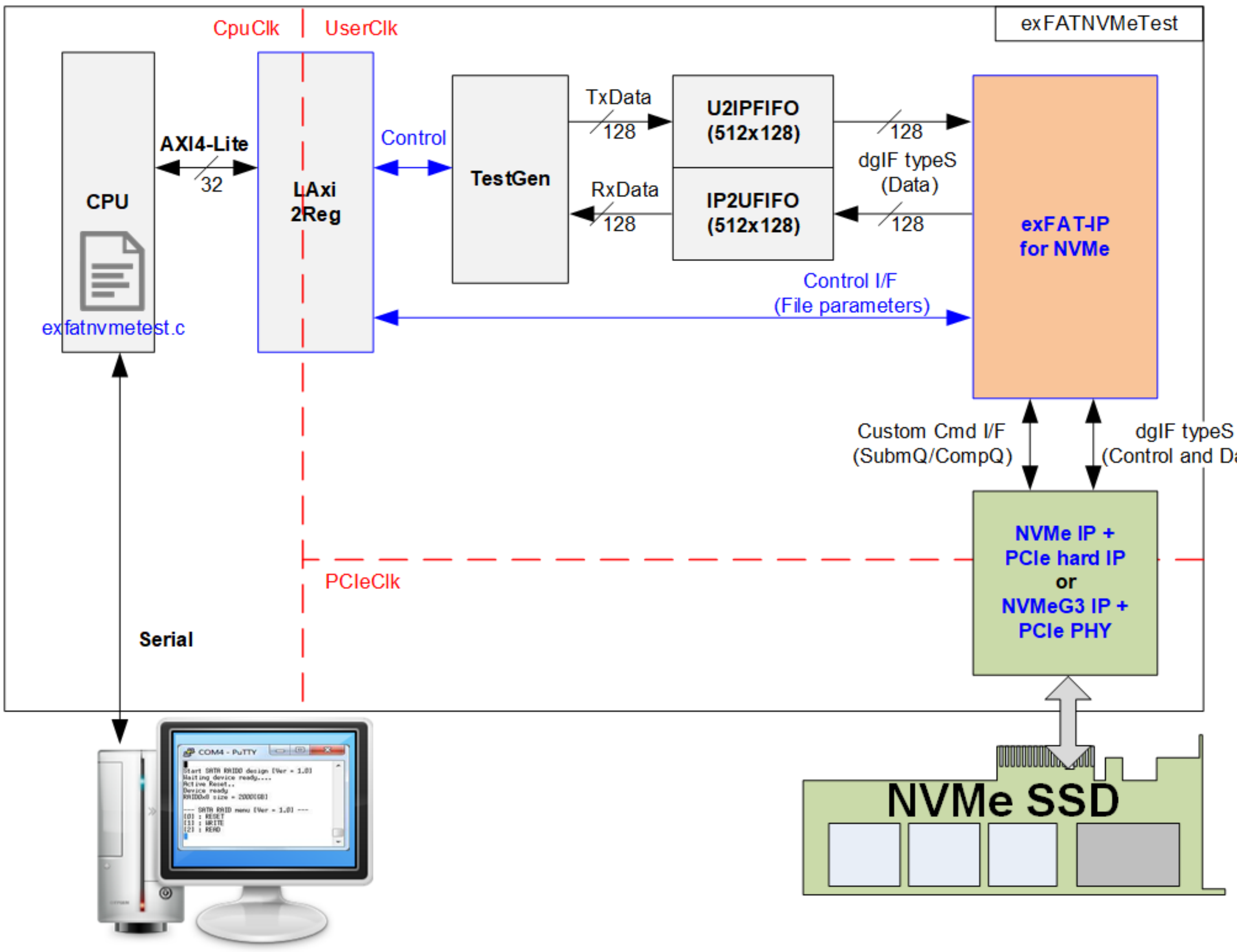

Figure 2-1 exFAT-IP for NVMe reference design

The exFAT-IP supports five commands: Format, Secure Format, Write file, Read file, and Shutdown. Transfer performance is displayed on the Serial console as test results after the execution of Write file or Read file command. Post-Write file operation, users can connect the NVMe SSD to other exFAT-compatible hosts, such as PCs, to read and verify the test data files.

In the reference design, three clock domains are utilized.

1) CpuClk: This is the clock for the CPU and its peripherals. It must be a stable clock, distinct from other hardware interfaces.
2) PCIeClk:
   a) For NVMe-IP: Output from PCIe hard IP, synchronizing with the 128-bit AXI4 stream bus at 250 MHz for 4-lane PCIe Gen3 and 125 MHz for 4-lane PCIe Gen2.
   b) For NVMeG3-IP: Output from PCIe PHY, synchronizing with the 128-bit PIPE interface at 250 MHz for 4-lane PCIe Gen3.
3) UserClk: This clock domain differs from others and serves as the main clock for the user interface of the exFAT-IP, NVMe(G3)-IP, FIFO, and TestGen. According to the NVMe(G3)-IP datasheet, the UserClk frequency must be at least equal to the PCIeClk. In this reference design, UserClk is set at 275/280 MHz for PCIe Gen3 or 200 MHz for PCIe Gen2.

Further details of the hardware implementation in the exFAT-IP for NVMe reference design are elaborated in the subsequent sections.

## 2.1 TestGen

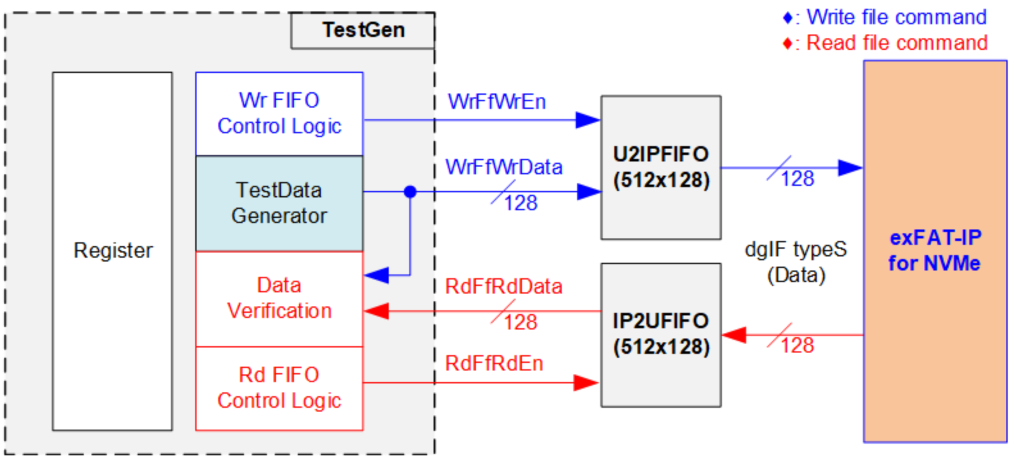

Figure 2-2 TestGen interface

TestGen module manages the data interface for the exFAT-IP, facilitating data transfer in both directions. During a Write file command, TestGen sends 128-bit test data to the exFAT-IP. In contrast, during a Read file command, the test data is received for comparison with the expected value, ensuring data accuracy.

Within this block, the Wr FIFO and Rd FIFO Control Logics assert the write enable and read enable signals when the FIFO is ready to measure the peak performance. The Register file in the TestGen receives various test parameters from the user, including file name, file size, the number of files, transfer direction, verification enable, and test pattern selector. For further details of the hardware logic of TestGen, refer to Figure 2-3.

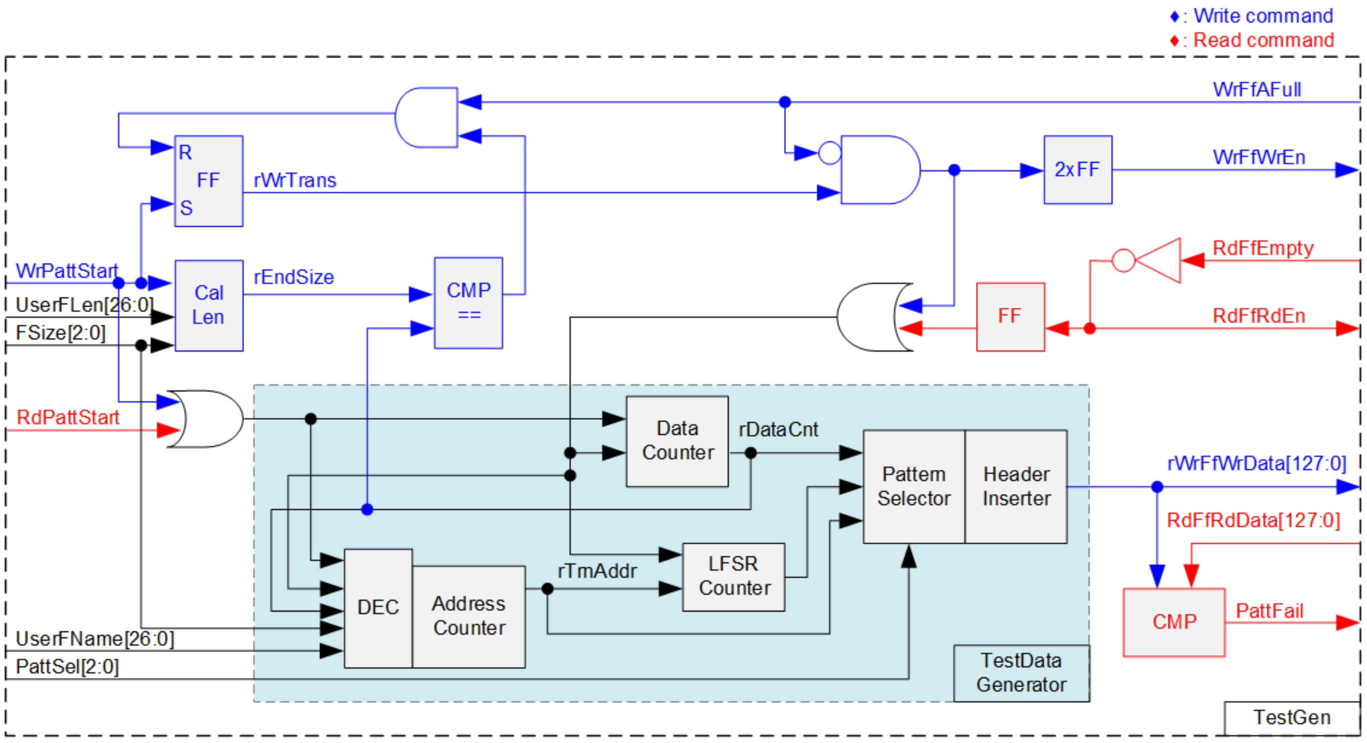

Figure 2-3 TestGen hardware

The primary flow control signal for the Write file command is WrFfAFull, while for the Read file command, it is RdFfEmpty. In the case of a Write file command, WrFfAFull is de-asserted to 0b when there is free space in the Write FIFO beyond a certain threshold. When WrFfAFull=0b, WrFfWrEn is asserted to 1b to send Write data to the FIFO. For a Read file command, RdFfEmpty is de-asserted to 0b when there is available data in the Read FIFO. When RdFfEmpty=0b, RdFfRdEn is asserted to 1b to read data from the FIFO.

The user can configure the following test parameters: the number of files (UserFLen), the first file name (UserFName), file size (FSize), and test pattern selector (PattSel). UserFLen and FSize are used to determine the end position (rEndSize) for comparison with the Data counter. Once all data is completely transferred, WrFfWrEn and RdFfRdEn are set to 0b.

The “TestData Generator” subblock is responsible for generating the test data (WrData) that will be transmitted to the exFAT-IP during the Write file command. Each 512-byte data segment comprises a 64-bit header data and a test pattern, selected by the PattSel parameter.

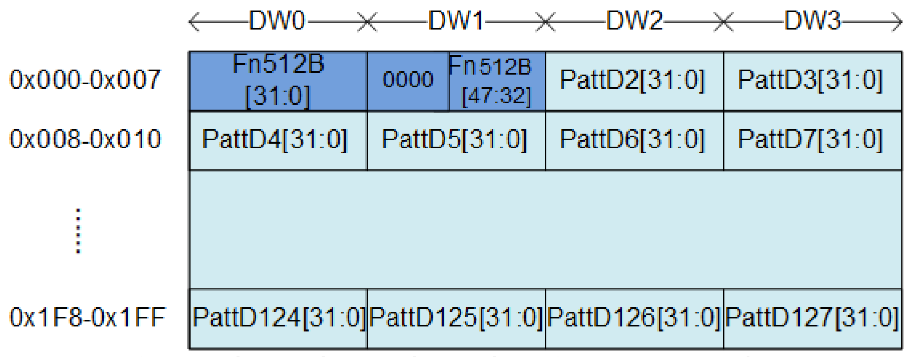

Figure 2-4 Test pattern format in each 512-byte data for Increment/Decrement/LFSR pattern

As shown in Figure 2-4, the 64-bit header at DW#0 (Dword#0) and DW#1 is generated by combining the 48-bit signal of rTrnAddr, computed from UserFName and FSize, with a zero value. The remaining data (DW#2 – DW#1023) represents the test pattern, which can be chosen from three different formats: 32-bit incremental data, 32-bit decremental data, and 32-bit LFSR counter. The 32-bit incremental data is derived from the output of the Data Counter. The decremental data is obtained by applying the logical NOT operation to the incremental data. The LFSR data is generated using the Fibonacci LFSR algorithm, following the equation x^31 + x^21 + x + 1. The 128-bit LFSR pattern employs a look-ahead technique to calculate four 32-bit LFSR data in one clock cycle.

64-bit header is created by calculating the unique value for each 512-byte data, controlled by Address counter inside TestData Generator. The initial value of the address counter is calculated by UserFName x File size, decoded from FSize. After that, the address counter is increased when finishing transferring 512-byte data.

In case of the all-zero and all-one patterns, a 64-bit header is not included within the 512-byte data. These patterns are often used to assess the optimal Write/Read performance of certain SSDs.

The generated test data serves as the Write data for the FIFO (rWrFfWrData) during the Write file command, or it is used as the expected data for verification against the read data obtained from the FIFO (RdFfRdData) during the Read file command. In the event of a verification failure, the failure flag (PattFail) is asserted to 1b. The timing diagram for writing data to the FIFO during the Write file command is shown in Figure 2-5.

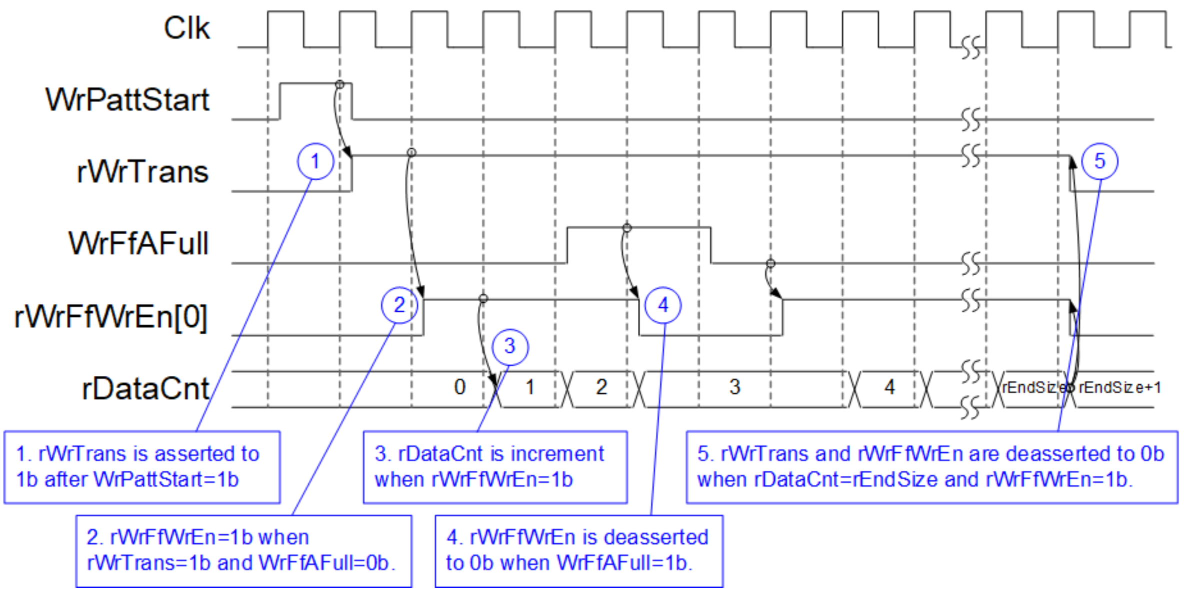
Clk
WrPattStart
rWrTrans
WrFfAFull
rWrFfWrEn[0]
rDataCnt
0
1
2
3
4
rEndSize
rEndSize+1
1. rWrTrans is asserted to 1b after WrPattStart=1b
2. rWrFfWrEn=1b when rWrTrans=1b and WrFfAFull=0b.
3. rDataCnt is increment when rWrFfWrEn=1b
4. rWrFfWrEn is deasserted to 0b when WrFfAFull=1b.
5. rWrTrans and rWrFfWrEn are deasserted to 0b when rDataCnt=rEndSize and rWrFfWrEn=1b.

Figure 2-5 Timing diagram of Write operation in TestGen

1) WrPattStart is set to 1b for one clock cycle when the user sets the register to start file operation. On the subsequent clock, rWrTrans is asserted to 1b to enable the control logic for generating write enable signals to the FIFO.
2) The write enable to FIFO (rWrFfWrEn) is asserted to 1b under two conditions: rWrTrans must be set to 1b indicating that the write operation is active, and the FIFO must not be full, which is indicated by WrFfAFull=0b.
3) The write enable signal also functions as a counter enable to count the total data transferred during the write operation.
4) If the FIFO is nearly full (WrFfAFull=1b), the write process is paused by de-asserting rWrFfWrEn to 0b.
5) When the total data count matches the predetermined value, rWrTrans is de-asserted to 0b. Concurrently, rWrFfWrEn is also de-asserted to 0b to halt data generation.

The read enable of the FIFO is controlled by its empty flag. Unlike the write enable, the read enable signal is continuous and is not controlled by a total count or initiated by a start flag. When read enable is asserted to 1b, both the data counter and the address counter increment concurrently. This process aids in counting total data and generating the header for the expected value.

## 2.2 exFAT

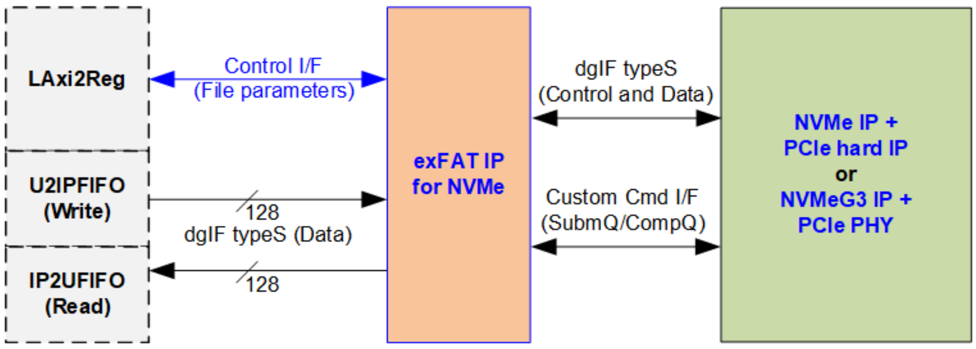

Figure 2-6 exFAT hardware

An exFAT hardware set contains three submodules: the exFAT-IP, the NVMe(G3)-IP, and the PCIe-IP. The user logic is designed to connect with the exFAT-IP, which provides two interface types: the Control interface and the Data interface. In this reference design, the Control interface is connected to LAxi2Reg, while the Data interface is connected to the FIFOs. Further details of each submodule are described in this section.

### 2.2.1 exFAT-IP for NVMe

The exFAT-IP is an extension module of the NVMe(G3)-IP, an IP core from Design Gateway, designed to store and retrieve data on the NVMe SSD using the exFAT file system instead of the raw data format, while achieving the same performance as the raw data format. Further details of the exFAT-IP can be found in the datasheet available on our website.
https://dgway.com/products/IP/NVMe-IP/dg_exfatip_nvme_data_sheet_en/

### 2.2.2 NVMe(G3)-IP

The NVMe(G3)-IP facilitates the NVMe protocol on the host side, allowing direct access to an NVMe SSD without using a PCIe switch. The user interface is designed using dgIF typeS format. Both NVMe-IP and NVMeG3-IP share identical NVMe features, differing only in the implementation at the lower layer boundary of PCIe.

Low-level interface of NVMe-IP is designed to connect with the Integrated Block for PCIe, which is a Hard IP in AMD Xilinx devices. Further details on NVMe-IP are available in the datasheet:
https://dgway.com/products/IP/NVMe-IP/dg_nvme_ip_data_sheet_en/

Low-level interface of NVMeG3-IP is designed to connect with PCIe PHY, an AMD Xilinx IP Core, providing a solution for devices lacking PCIe hard IP. More information on NVMeG3-IP can be found in the datasheet.
https://dgway.com/products/IP/NVMe-IP/dg_nvmeg3_ip_data_sheet_xilinx_en/

### 2.2.3 PCIe

**Integrated Block for PCIe**

This hard IP in AMD Xilinx devices implements the Physical, Data Link, and Transaction Layers of the PCIe specification. Detailed documentation is available in AMD Xilinx documents.
PG054: 7 Series FPGAs Integrated Block for PCI Express
PG023: Virtex-7 FPGA Gen3 Integrated Block for PCI Express
PG156: UltraScale Devices Gen3 Integrated Block for PCI Express
PG213: UltraScale+ Devices Integrated Block for PCI Express

**PCIe PHY IP**

This module, an AMD Xilinx IP Core, implements the Physical Layer of the PCIe specification. The user interface follows the PHY Interface for PCI Express (PIPE) standards. When paired with NVMeG3-IP, the PCIe PHY is set to a lone width of x4 and a link speed of 8.0 GT/s. Further details can be found in the “PG239: PCI Express PHY” document.
https://www.xilinx.com/support/documentation/ip_documentation/pcie_phy/v1_0/pg239-pcie-phy.pdf

## 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 the 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 2-7, is designed to connect the CPU system via the AXI4-Lite interface, in compliance with the standard.

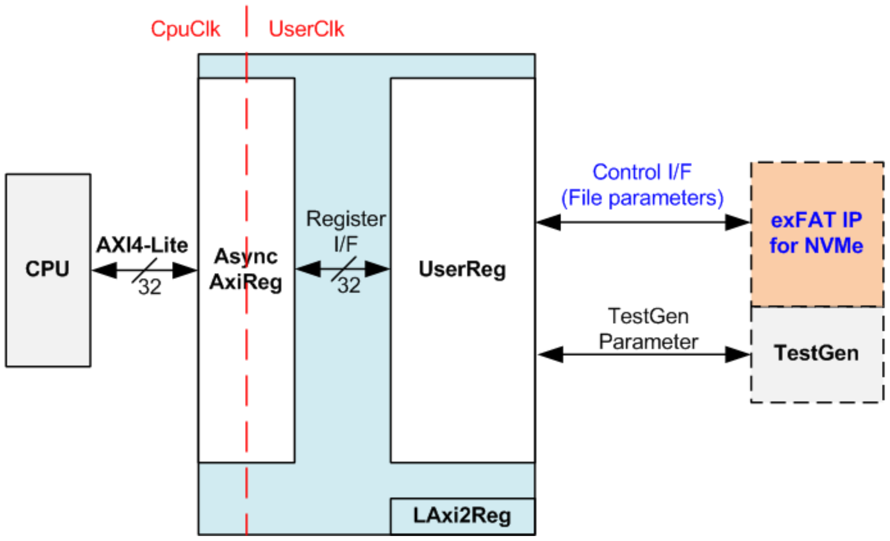

Figure 2-7 CPU and peripherals hardware

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

## 2.3.1 AsyncAxiReg

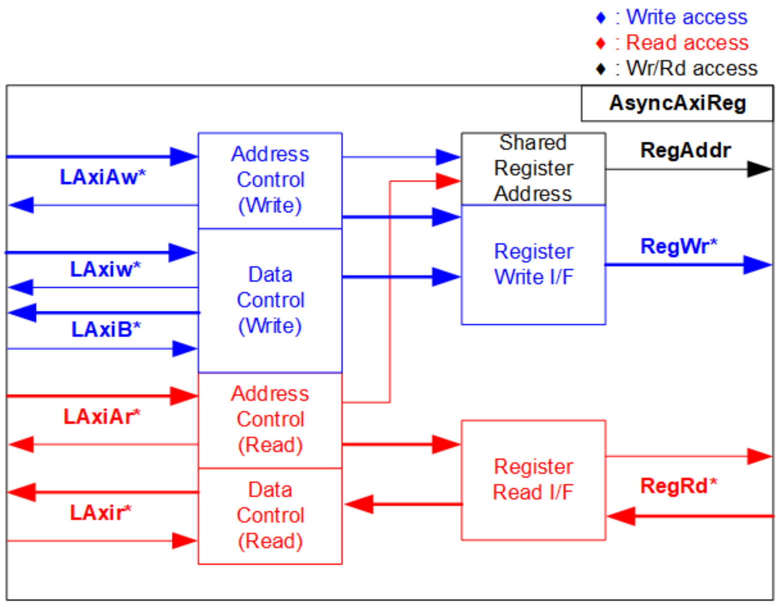

Figure 2-8 AsyncAxiReg Interface

The AXI4-Lite bus interface signals are categorized into five groups: LAxiAw* (Write address channel), LAxiw* (Write data channel), LAxiB* (Write response channel), LAxiAr* (Read address channel), and LAxir* (Read data channel). More information on creating custom logic for the AXI4-Lite bus can be found in the following document.
https://github.com/Architech-Silica/Designing-a-Custom-AXI-Slave-Peripheral/blob/master/designing_a_custom_axi_slave_rev1.pdf

According to the AXI4-Lite standard, the write channel and read channel operate independently for both control and data interfaces. Therefore, the logic in the AsyncAxiReg module to interface with the AXI4-Lite bus is divided into four groups: Write control logic, Write data logic, Read control logic, and Read data logic, as shown on the left side of Figure 2-8. 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 the AXI4-Lite bus is latched and transferred to the Read register interface, while Read data is returned from the Register interface to the AXI4-Lite bus via clock domain crossing registers. In the Register interface, RegAddr is a shared signal for write and read access, loading the value from LAxiAw for write access or LAxiAr for read access.

The Register interface is compatible with a single-port RAM interface for write transaction. However, the read transaction of the Register interface has been slightly modified from the RAM interface by adding the RdReq and RdValid signals to control read latency time. Since the address of the Register interface is shared for both write and read transactions, the user cannot write and read the register simultaneously. The timing diagram of the Register interface is shown in Figure 2-9.

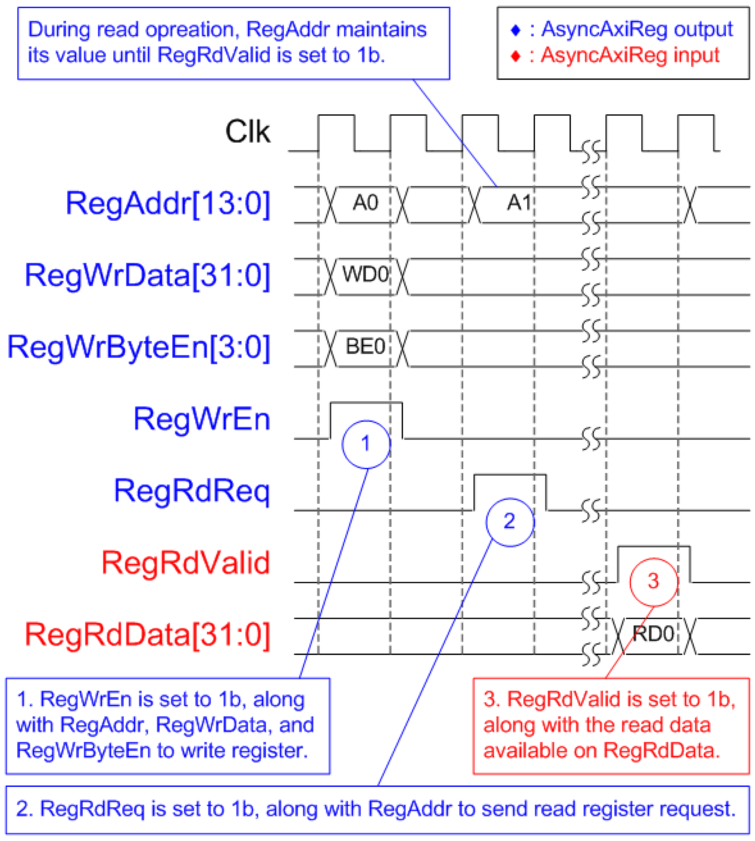
During read opreation, RegAddr maintains its value until RegRdValid is set to 1b.
♦ : AsyncAxiReg output
♦ : AsyncAxiReg input
Clk
RegAddr[13:0]
A0
A1
RegWrData[31:0]
WD0
RegWrByteEn[3:0]
BE0
RegWrEn
RegRdReq
RegRdValid
RegRdData[31:0]
RD0
1
2
3
1. RegWrEn is set to 1b, along with RegAddr, RegWrData, and RegWrByteEn to write register.
3. RegRdValid is set to 1b, along with the read data available on RegRdData.
2. RegRdReq is set to 1b, along with RegAddr to send read register request.

Figure 2-9 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

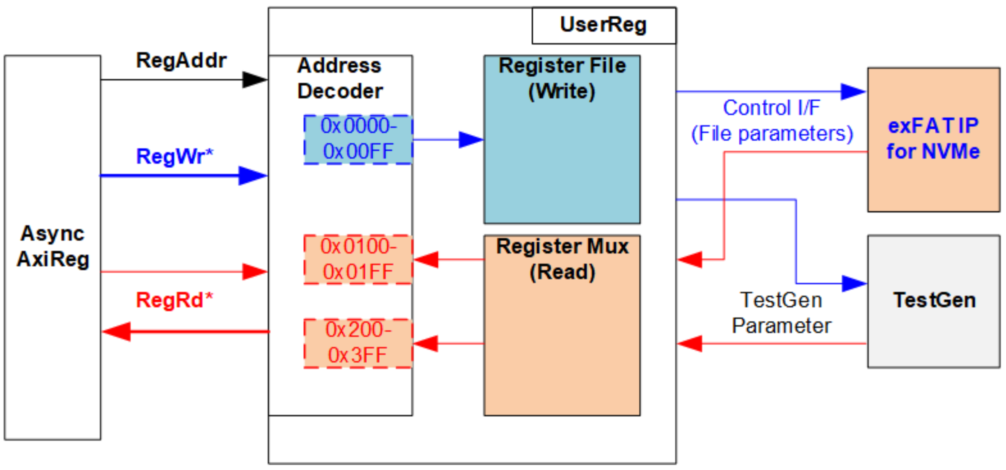

Figure 2-10 UserReg Interface

The UserReg module consists of an Address decoder, a Register File, and a Register Mux. The Address decoder interprets the address requested by AsyncAxiReg and selects the active register for either write or read transactions. The address range assigned in UserReg is divided into three areas, as illustrated in Figure 2-10.

1) 0x0000 – 0x00FF: Allocated for setting the test parameters of exFAT-IP and TestGen. This area is designated for write access only.
2) 0x0100 – 0x01FF: Dedicated to reading the status of exFAT-IP. This area is restricted to read access only.
3) 0x0200 – 0x02FF: Used for reading the status of TestGen. This area is also read access only.

The Address decoder uses the upper bits of RegAddr to determine the correct hardware (exFAT-IP or TestGen) for interaction. Within UserReg, the Register File operates with a 32-bit bus width, rendering the write byte enable (RegWrByteEn) unused in the test system. The CPU employs a 32-bit pointer for setting hardware registers.

For reading a register, multi-level multiplexers (mux) select the output data bases on the address. The lower bits of RegAddr are utilized within the submodule to choose the active data inside it, while the upper bits are used by UserReg to select the correct submodule from which to retrieve the read data. The total latency time for reading data is two clock cycles, with RegRdValid being created by asserting two D Flip-flops upon a RegRdReq. Additional details regarding the address mapping within the UserReg module are provided in Table 2-1.

Table 2-1 Register Map

| Address<br>Rd/Wr | Register Name<br>(Label in the "exfatnvmetest.c") | Description |
|---|---|---|
| **0x0000 – 0x00FF: Control signals of exFAT-IP and TestGen (Write access only)** | | |
| BA+0x0000 | User File Name Reg<br>(USRFNAME_INTREG) | [26:0]: Input to be UserFName of exFAT-IP for NVMe, the first file name to execute commands. |
| BA+0x0004 | User File Length Reg<br>(USRFLEN_INTREG) | [26:0]: Input to be UserFLen of exFAT-IP for NVMe, the total number of files requested in this command. |
| BA+0x0008 | File Size Reg<br>(USRFSIZE_INTREG) | [2:0]: Input to be FSize of exFAT-IP for NVMe, configured file size. |
| BA+0x000C | Created Date and Time Reg<br>(DATETIME_INTREG) | [4:0]: Input to be FTimeS of exFAT-IP for NVMe<br>[10:5]: Input to be FTimeM of exFAT-IP for NVMe<br>[15:11]: Input to be FTimeH of exFAT-IP for NVMe<br>[20:16]: Input to be FDateD of exFAT-IP for NVMe<br>[24:21]: Input to be FDateM of exFAT-IP for NVMe<br>[31:25]: Input to be FDateY of exFAT-IP for NVMe |
| BA+0x0010 | User Command Reg<br>(USRCMD_INTREG) | [2:0]: Input to be UserCmd of exFAT-IP for NVMe<br>When this register is written, the design asserts UserReq=1b (command request) to exFAT-IP for NVMe to start the operation. |
| BA+0x0014 | Pattern Select Reg<br>(PATTSEL_INTREG) | [2:0]: Select test data pattern<br>000b-Increment, 001b-Decrement, 010b-All 0, 011b-All 1,100b-LFSR |
| BA+0x0020 | Timeout Reg<br>(TIMEOUT_INTREG) | [31:0]: Mapped to TimeOutSet[31:0] of exFAT-IP, timeout value for waiting for a response from SSD. |
| **0x0100 – 0x01FF: Status signals of exFAT-IP (Read access only)** | | |
| BA+0x0100 | User Status Reg<br>(USRSTS_INTREG) | [0]: Mapped to UserBusy of exFAT-IP for NVMe, busy status.<br>[1]: Mapped to UserError of exFAT-IP for NVMe, error flag.<br>[2]: Data verification fail in TestGen (0b: Normal, 1b: Error) |
| BA+0x0104 | Total file capacity Reg<br>(TOTALFCAP_INTREG) | [26:0]: Mapped to TotalFCap[26:0] of exFAT-IP for NVMe, maximum number of files for this SSD. |
| BA+0x0108 | User Error Type Reg<br>(USRERRTYPE_INTREG) | [31:0]: Mapped to UserErrorType[31:0] of exFAT-IP for NVMe, indicating error status of exFAT-IP. |
| BA+0x010C | exFAT IP Test pin (Low) Reg<br>(TESTPINL_INTREG) | [31:0]: Mapped to TestPin[31:0] of exFAT-IP for NVMe, reserved for internal use. |
| BA+0x0110 | exFAT IP Test pin (High) Reg<br>(TESTPINH_INTREG) | [31:0]: Mapped to TestPin[63:32] of exFAT-IP for NVMe, reserved for internal use. |
| BA+0x0114 | Directory capacity Reg<br>(DIRCAP_INTREG) | [19:0]: Mapped to DirCap[19:0] of exFAT-IP for NVMe, maximum number of files for each directory. |
| BA+0x0118 | File Size in the disk Reg<br>(DFSIZE_INTREG) | [2:0]: Mapped to DiskFsize of exFAT-IP for NVMe, file size currently used in this SSD. |
| BA+0x011C | Total file in the disk Reg<br>(DFNUM_INTREG) | [26:0]: Mapped to DiskFnum of exFAT-IP for NVMe, total count of files in the SSD. |
| BA+0x0120 | NVMe LBA Size (Low) Reg<br>(NVMLBASIZEL_INTREG) | [31:0]: Mapped to LBASize(bit[31:0]) of NVMe(G3)-IP, total capacity of SSD in 512-byte unit. |
| BA+0x0124 | NVMe LBA Size (High) Reg<br>(NVMLBASIZEH_INTREG) | [15:0]: Mapped to LBASize(bit[47:32]) of NVMe(G3)-IP, total capacity of SSD in 512-byte unit.<br>[30]: Mapped to SuppSecureFmt of exFAT-IP, Secure Format supported flag.<br>[31]: Mapped to LBAMode of NVMe(G3)-IP, indicating LBA unit size of SSD. |
| BA+0x0128 | NVMe Completion Status Reg<br>(NVMCOMPSTS_INTREG) | Completion Status from NVMe(G3)-IP<br>[15:0]: Mapped to AdmCompStatus[15:0], Admin completion Status.<br>[31:16]: Mapped ot IOCompStatus[15:0], I/O completion Status. |
| BA+0x012C | NVMe CAP Reg<br>(NVMCAP_INTREG) | [31:0]: Mapped to NVMeCAPReg[31:0] of NVMe(G3)-IP, capabilities of NVMe. |

| Address<br>Rd/Wr | Register Name<br>(Label in the "exfatnvmetest.c") | Description |
|---|---|---|
| **0x0100 – 0x01FF: Status signals of exFAT-IP (Read access only)** | | |
| BA+0x0130 | NVMe Test pin Reg<br>(NVMTESTPIN_INTREG) | [31:0]: Mapped to TestPin[31:0] of NVMe(G3)-IP |
| BA+0x0138 - BA+0x013F | NVM MAC Test pin Reg<br>(NVMMACTESTPINL/H_ INTREG) | [31:0]: Mapped to MACTestPin of NVMeG3-IP<br>This register.is not applied for NVMe-IP.<br>0x0138: Bits[31:0], 0x013C: Bits[63:32] |
| **0x0200 – 0x02FF: Status signals of TestGen (Read access only)** | | |
| BA+0x0200 - BA+0x20F | Expected value Word0-3 Reg<br>(EXPPATW0-3_INTREG)<br>(EXPPATW3_INTREG) | 128-bit of the expected data at the 1st failure data in TestGen when operating Read file command.<br>0x0200: Bits[31:0], 0x0204: Bits[63:32], …, 0x20C: Bits[127:96] |
| BA+0x0210 - BA+0x21F | Read value Word0-3 Reg<br>(RDPATW0-3_INTREG) | 128-bit of the read data at the 1st failure data in TestGen when operating Read file command.<br>0x0210: Bits[31:0], 0x0214: Bits[63:32], …, 0x21C: Bits[127:96] |
| BA+0x0220 | Failure Byte Address (Low) Reg<br>(FAILADDRL_INTREG) | [31:0]: Bit[31:0] of the byte address in the file at the 1st failure data in TestGen when operating Read file command. |
| BA+0x0224 | Failure Byte Address (High) Reg<br>(FAILADDRH_INTREG) | [6:0]: Bit[38:32] of the byte address in the file at the 1st failure data in TestGen when operating Read file command. |
| BA+0x0228 | Failure File Name Reg<br>(FAILFNAME_INTREG) | [26:0]: Filename of the 1st failure data in TestGen when operating Read file command. |
| BA+0x0230 | Current test byte (Low) Reg<br>(CURTESTSIZEL_INTREG) | [31:0]: Bit[31:0] of the current data transfer size in bytes of TestGen module when operating Write file or Read file command. |
| BA+0x0234 | Current test byte (High) Reg<br>(CURTESTSIZEH_INTREG) | [24:0]: Bit[56:32] of the current data transfer size in bytes of TestGen module when operating Write file or Read file command. |
| **0x0800 – 0xFFFF: Other interfaces** | | |
| BA+0x0800<br>Rd | exFAT-IP Version Reg<br>(EXFATNVMVER_INTREG) | [31:0]: Mapped to IPVersion[31:0] of exFAT-IP, indicating the version of exFAT-IP. |
| BA+0x0804<br>Rd | NVMe-IP Version Reg<br>(NVMVER_INTREG) | [31:0]: Mapped to IPVersion[31:0] of NVMe(G3)-IP, indicating the version of NVMe(G3)-IP. |

# 3 CPU Firmware

## 3.1 Test firmware (exfatnvmetest.c)

Upon system startup, the CPU follows these steps to complete the initialization process.

1) Initialize UART and Timer settings.
2) Wait for the completion of the exFAT-IP initialization process, indicated by USRSTS_INTREG[0]=0b.
3) Read and display the SSD information, including Secure erase support (NVMLBASIZEH_INTREG[30]), maximum number of files in SSD (TOTALFCAP_INTREG), maximum number of files per directory (DIRCAP_INTREG), current file size configuration (DFSIZE_INTREG), and total files stored in SSD (DFNUM_INTREG).
4) Present the format menu, which can be Format and Secure format (only for the SSD that is supported). The user can choose to execute the Format command, Secure Format command, or proceed to the next step without executing the Format command.

   To execute a Format or Secure Format operation, the command value assigned to the exFAT-IP is different, while the remaining steps are similar. Further details of each Format operation are described in the subsequent section.
5) Display the main menu, providing four test options: (Secure Format), Write File, Read File, and Shutdown.

Further details for executing each test options are described below.

### 3.1.1 (Secure) Format

When this test option is selected, the Format operation is executed following the steps below.

1) Display three options on this menu: Format execution, Secure Format execution, and No execution. The user enters the keys to proceed or cancel the Format operation.
2) Prompt the user to set the created date and created time of the empty directories or skip setting and use the same value. Once all inputs are received, determine the value for setting to DATETIME_INTREG and set it.
3) Read disk capacity from NVMLBASIZEL/H_INTREG, calculate supported file sizes, and display results on the console.
4) Ask the user to set the file size. Validate the input and set it to USRFSIZE_INTREG.
5) Set bits[2:0] of USERCMD_INTREG to send (Secure) Format command. The exFAT-IP sets busy status (USRSTS_INTREG[0]) to 1b upon initiating the operation.
6) The CPU waits for the command completion or an error detection by monitoring the value of USRSTS_INTREG[1:0].
   - Bit[0] set to 0b indicates the completion of the command. Following this, proceed to the subsequent step.
   - Bit[1] set to 1b indicates an error detection. Following this, read the error details from USRERRTYPE_INTREG, interpret the read value, and display an error message corresponding to the error type on console.

7) Upon the completion of the command, read the SSD information, and display the read value on the console. The SSD information includes the maximum number of files in SSD (TOTALFCAP_INTREG), the maximum number of files per directory (DIRCAP_INTREG), current file size configuration (DFSIZE_INTREG), and total files stored in SSD (DFNUM_INTREG).

### 3.1.2 Write file/Read file command

The process upon selecting this test option is described below.

1) Select operations: Write file or Read file.
2) Receive additional parameters and configure them into the registers corresponding the selected operation.

   <u>Write file operation</u>

   i) Prompt the user to configure the created date and created time of the new file or use the latest value. Compute the result and set it to DATETIME_INTREG.
   ii) Show the latest write file name and prompt the user to use this value for creating the new files, subsequent to the latest value, or to use other values to overwrite the old files with new data. Validate its value, and configure it to USRFNAME_INTREG.
   iii) Receive additional inputs from the user: number of files and test pattern. Validate these inputs and compute the total transfer size for displaying on the console.
   iv) Configure the user inputs into the corresponding registers, including USRFNAME _INTREG, USRFLEN_INTREG, PATTSEL_INTREG, and USRCMD_INTREG. Following this, the IP initiates the operation.

   <u>Read file operation</u>

   i) Read DFNUM_INTREG to check the number of available files in the SSD, and display the read value to indicate the valid range of the first file name for Read file command.
   ii) Prompt the user to specify the file name, validate its value, and then configure it to USRFNAME_INTREG. After this, follow step iii) – step iv) of Write file operation.

3) The CPU waits for the command completion, detects errors by monitoring the value of USRSTS_INTREG[2:0].
   - Bit[0] set to 0b indicates the completion of the command. Proceed to the subsequent step after this.
   - Bit[1] set to 1b indicates an error detection. Upon this detection, read the error details from USRERRTYPE_INTREG, interpret the read value, and display an error message corresponding to the error type on console.
   - Bit[2] set to 1b indicates an error due to data verification failure. Upon this detection, display the details of the error data on the console. Despite a verification error, the test operation continues until its completion or the reception of operation cancellation.

   During data transfer, read CURTESTSIZEL/H_INTREG to check the current transfer size, and display the result on the console every second to monitor the test progress.

4) Upon the completion of the command, read the test results from registers and internal variables:
   - Retrieve the total transfer size from CURTESTSIZEL/H_INTREG.
   - Retrieve the timer value to compute the total time usage and average transfer speed.
   - Retrieve the file name and number of files to compute the first file name and the last file name with their directories.

### 3.1.3 Shutdown Command

When this test option is selected, the Shutdown operation is executed following these steps.

1) Ask the user to confirm the Shutdown operation. The user enters the keys to proceed or cancel the Shutdown operation.
2) Set bits[2:0] of USERCMD_INTREG to send the Shutdown command. The exFAT-IP sets busy status (USRSTS_INTREG[0]) to 1b upon initiating the operation.
3) The CPU waits for the command completion or an error detection by monitoring the value of USRSTS_INTREG[1:0].
   - Bit[0] set to 0b indicates the completion of the command. Following this, proceed to the subsequent step.
   - Bit[1] set to 1b indicates an error detection. Following this, read the error details from USRERRTYPE_INTREG, interpret the read value, and display an error message corresponding to the error type on the console.

4) Upon the completion of the command, display a message to indicate the SSD becomes inactive. The user cannot send additional command requests to the exFAT-IP. To continue testing, the user must power off and power on the system.

## 3.2 Function list in Test firmware

| void change_ftime(void) | |
|---|---|
| Parameters | None |
| Return value | None |
| Description | Print current created time and date by calling 'show_ftime' function. Afterward, prompt the user to to either use the old value or change it. Verify the user input if a new value is received. Then, set the created date and time to DATETIME_INTREG and the global parameter (DateTime). |

| int format_fat(void) | |
|---|---|
| Parameters | None |
| Return value | 0: User cancels command or command is completed.<br>-1: Receive invalid input or error is found. |
| Description | Execute (Secure) Format command as outlined in section 3.1.1. |

| void get_cursize(unsigned int user, unsigned long long* cursize) | |
|---|---|
| Parameters | None |
| Return value | None |
| Description | Read CURTESTSIZEH/L_INTREG, determine current transfer size, and update result to 'cursize' parameter. |

| int get_param(userin_struct* userin, unsigned int user_cmd) | |
|---|---|
| Parameters | userin: A set of test parameters input from the user, including file name, a number of files, and test pattern.<br>user_cmd: 2-Write file command and 3-Read file command |
| Return value | 0: Valid input, -1: Invalid input |
| Description | Receive test parameters from the user via the console and validate their values. If any input is invalid, return -1 as function result. Otherwise, update the user-defined values to the 'userin' parameter. |

| void show_dir(userin_struct* userin, unsigned int user_cmd) | |
|---|---|
| Parameters | userin: A set of test parameters input from the user, including file name, a number of files, and test pattern.<br>user_cmd: 2-Write file command and 3-Read file command |
| Return value | None |
| Description | Calculate the location of the first and last files, and then display their names along with their directories and the command operation on the console. |

| void show_diskinfo(void) | |
|---|---|
| Parameters | None |
| Return value | None |
| Description | Retrieve the information of the SSD, including file size (via calling 'show_fsize' function), maximum number of files in the SSD (TotalFCap), maximum number of files per directory (DirCap), and total available files in the SSD (DFnum), and display the read values on the console. |

| void show_error(void) | |
|---|---|
| Parameters | None |
| Return value | None |
| Description | Read USRERRTYPE_INTREG, interpret an error type, and display the result on the console. |

| void show_fsize(void) | |
|---|---|
| Parameters | None |
| Return value | None |
| Description | Read the file size parameter (DFsize) and decode it to represent the size in MB or GB units. Then, display the result on the console. |

| void show_ftime(void) | |
|---|---|
| Parameters | None |
| Return value | None |
| Description | Read ‘DateTime’ variable, decode it to date, month, year, hour, minute, and second. Then, display the result on the console. |

| void show_result(unsigned int timeuseh, unsigned int timeusel) | |
|---|---|
| Parameters | timeuseh: The upper 32-bit read value of timer<br>timeusel: The lower 32-bit read value of timer |
| Return value | None |
| Description | Update the total transfer size by calling the ‘get_cursize’ function and display the result on the console using the ‘show_size’ function. Next, display the total time usage using the ‘show_time’ function. Subsequently, calculate the transfer performance and display the result on the console. |

| void show_size(unsigned long long size_input) | |
|---|---|
| Parameters | Data size in bytes |
| Return value | None |
| Description | Print input value in MB, GB, or TB unit. |

| void show_testpin(void) | |
|---|---|
| Parameters | None |
| Return value | None |
| Description | Read TESTPINL/H_INTREG, NVMTESTPIN_INTREG, and NVMMAC TESTPINL/H_INTREG (only for NVMeG3-IP). Then, display these values on the console. |

| void show_time(unsigned int timeuseh, unsigned int timeusel) | |
|---|---|
| Parameters | timeuseh: The upper 32-bit read value of timer<br>timeusel: The lower 32-bit read value of timer |
| Return value | None |
| Description | Read the values of timeuseh and timeusel, and calculate the total time to display on the console in usec, msec, or sec. |

| void show_vererr(void) | |
|---|---|
| Parameters | None |
| Return value | None |
| Description | Read the registers and display the details of verification error on the console, including FAILFNAME_INTREG (error file name), FAILADDRL/H_INTREG (error address), EXPPATW0-3_INTREG (expected value), and RDPATW0-3_INTREG (read value). |

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

| void update_diskparam(void) | |
|---|---|
| Parameters | None |
| Return value | None |
| Description | Read TOTALFCAP_INTREG, DFSIZE_INTREG, and DFNUM_INTREG to update these values to internal parameters (TotalFCap, DFsize, and DFnum). |

| 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.<br>-1: Receive invalid input or error is found. |
| Description | Execute the Write file or Read file command as outlined in section 3.1.2. |

## 4 Example Test Result

Figure 4-1 shows the test results obtained from running the demo system using a 512 GB Samsung 970 Pro SSD.

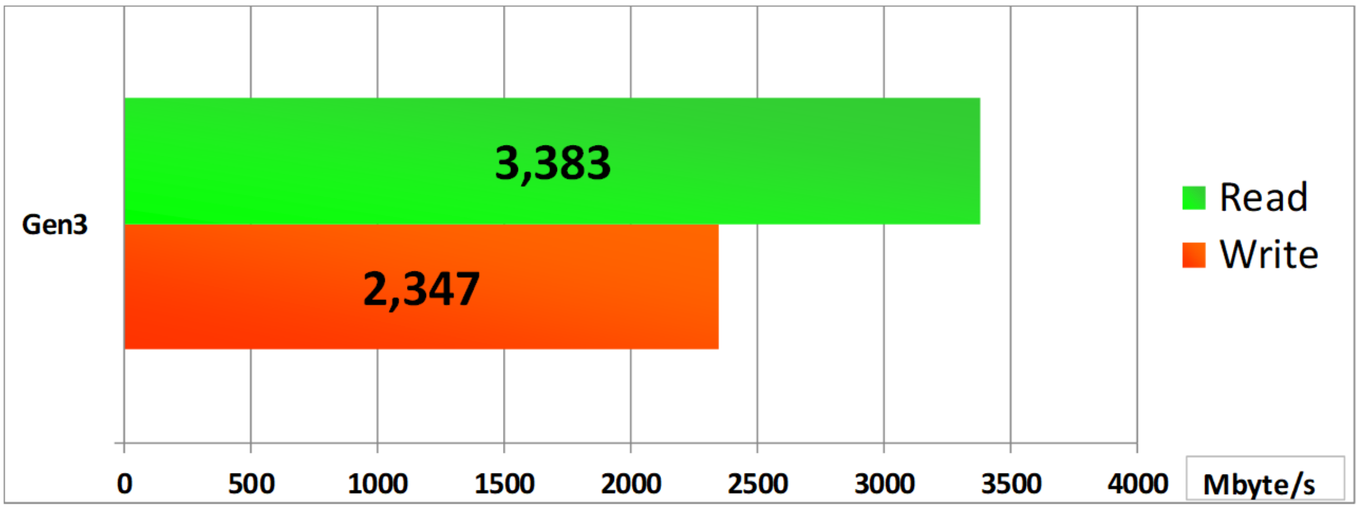

Figure 4-1 Test Performance of exFAT-IP for NVMe demo using Samsung 970 Pro SSD

Utilizing PCIe Gen3 on the KCU105 board, the write performance is approximately 2300 Mbyte/sec, while the read performance achieves about 3300 Mbyte/sec.

## 5 Revision History

| Revision | Date | Description |
|---|---|---|
| 2.00 | 7-May-24 | Add Secure Format feature |
| 1.05 | 14-Mar-22 | Add NVMeG3 IP to reference design |
| 1.04 | 23-Jul-20 | Update CPU firmware |
| 1.03 | 19-Feb-20 | Update TestGen for all zero and all one pattern |
| 1.02 | 14-May-19 | Update Dircap size and the design details |
| 1.01 | 22-Mar-19 | - Add DiskFsize, DiskFnum signal<br>- Add function list |
| 1.00 | 15-Jan-19 | Initial release |