FAT32-IP for NVMe reference design manual

Rev1.1 14-May-18

1 Introduction

In the hardware system, data stream can be stored to the SSD by using raw data or file system. Using raw data, the data is allocated in the SSD through physical address. If there are many data types in one SSD, the user will need to assign different address for each data group. Without standard, each system defines different data structure to store many data groups in one SSD. It is the problem for the Host to read SSD under many standards from different system.

As a result, file system is created to manage data in the SSD by setting up the table to be an index for data that is written to the SSD. The data is separated into many groups. Each group is called a “file”. For system flexibility, one file has some information to represent itself such as file name, file type, file size, and physical address of data in the file. So, the user can search the data to read from the SSD and identify where the free space in the disk available to write new data.

FAT32 is one of the most popular file systems to use in the storage device. FAT32 File structure is not complicate. The limitation of FAT32 is that maximum partition size is 2 TB and maximum file size is 4 GB.

Generally, when the Host accesses the data in the SSD through file system, the data transfer speed is lower than raw data. To write/read data in file system, it has the overhead time to read the file structure firstly to know the physical address of the data in the file. After that, the Host transfers data with SSD by using physical address. In some cases, data in one file is stored to many areas and the overhead time is increased to check many physical addresses for one file data. More overhead time reduces the performance to access SSD. Nevertheless, using file system makes the system more convenient and more flexible to share data in SSD to other Hosts.

In the reference design, FAT32-IP achieves the advantage of using raw data and file system. The demo shows that FAT32-IP can write and read data with SSD as file system at high performance. The performance of Write file command is about 2100 MB/s while the performance of Read file command is about 3200 MB/s. The hardware structure of FAT32-IP for NVMe-IP is different from the hardware of NVMe-IP design, as shown in Figure 1-1.

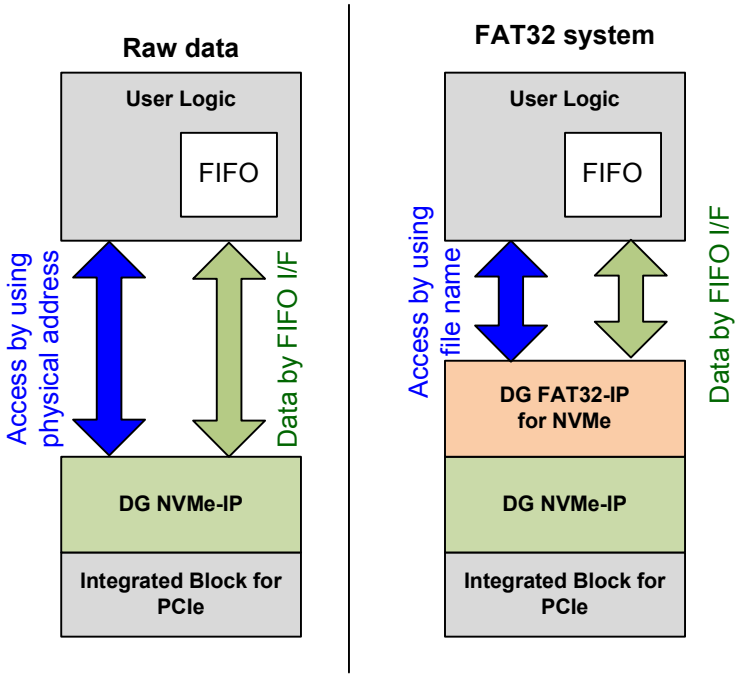

Figure 1-1 Hardware system for raw data and file system

Comparing to raw data system in the left side of Figure 1-1, FAT32 system includes FAT32-IP for NVMe to connect between User Logic and NVMe-IP. User interface is changed from physical level (SSD address and length) to be file level (file name and file size). But the data interface of raw data and FAT32 system is similar (FIFO interface). More details of FAT32-IP for NVMe reference design are described in the next topic.

2 Hardware overview

The reference design of DG FAT32-IP for NVMe is modified from NVMe-IP reference design by including DG FAT32-IP and updating control path to file index instead of physical signals, as shown in blue color of Figure 2-1.

Please see more details of NVMe-IP reference design from following document.
http://www.dgway.com/products/IP/NVMe-IP/dg_nvmeip_refdesign_en.pdf
http://www.dgway.com/products/IP/NVMe-IP/dg_nvmeip_instruction_en.pdf

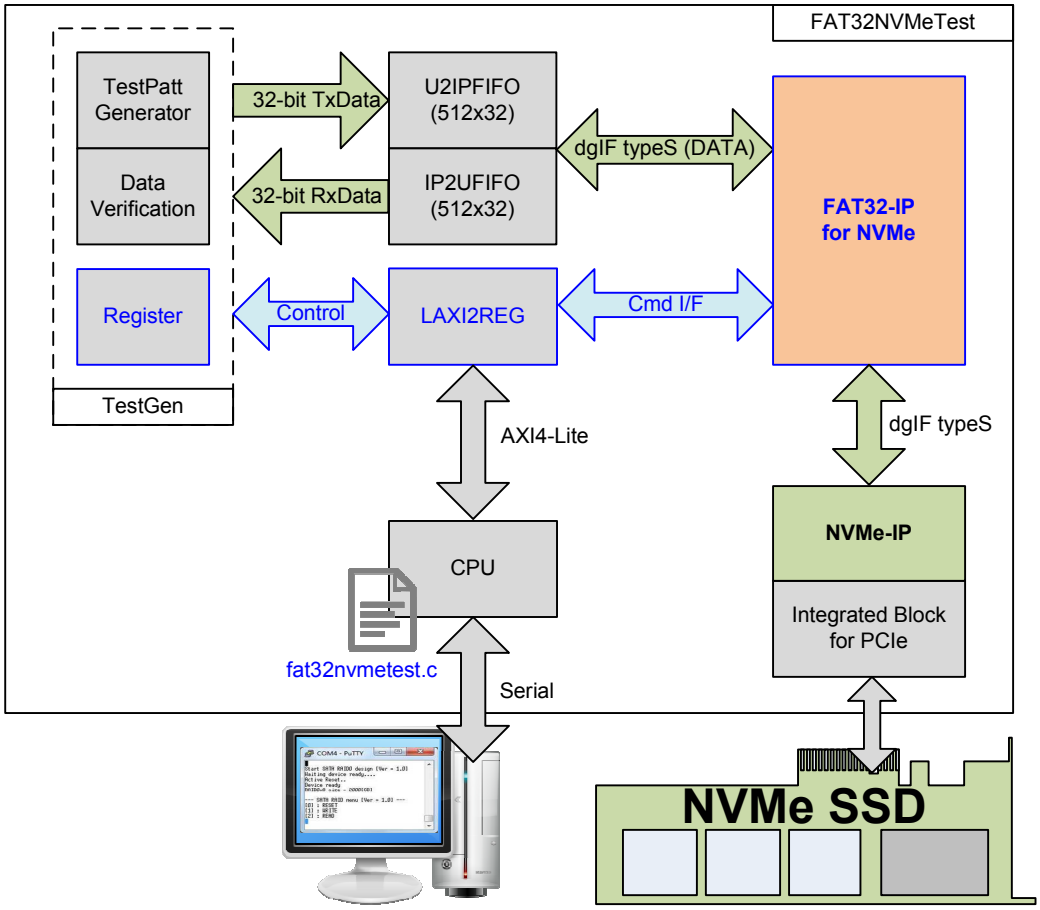

Figure 2-1 FAT32-IP for NVMe demo system

Comparing to NVMe-IP reference design, control signals of user interface are changed from physical level to file level. Registers inside TestGen and LAXI2REG module are modified to store file parameters, i.e. file name, file length, file size, and total file. The parameters are received from user through Serial console. CPU firmware supports three commands on the Main menu, i.e. Format, Write file, and Read file. As a test result, transfer performance is displayed on Serial console after complete to write file or read file operation. Also, SSD could be plugged to other Hosts such as PC for checking test file and data in the test file by using standard tools.

Data path in the design is same as NVMe IP reference design. More details of the hardware in FAT32-IP for NVMe demo design are described as follows.

2.1 TestGen

This module is designed to generate Test pattern to WrFf in Write file command or reads data from RdFf to verify in Read file command at the fastest speed to check system performance. The details of hardware inside TestGen are shown in Figure 2-2.

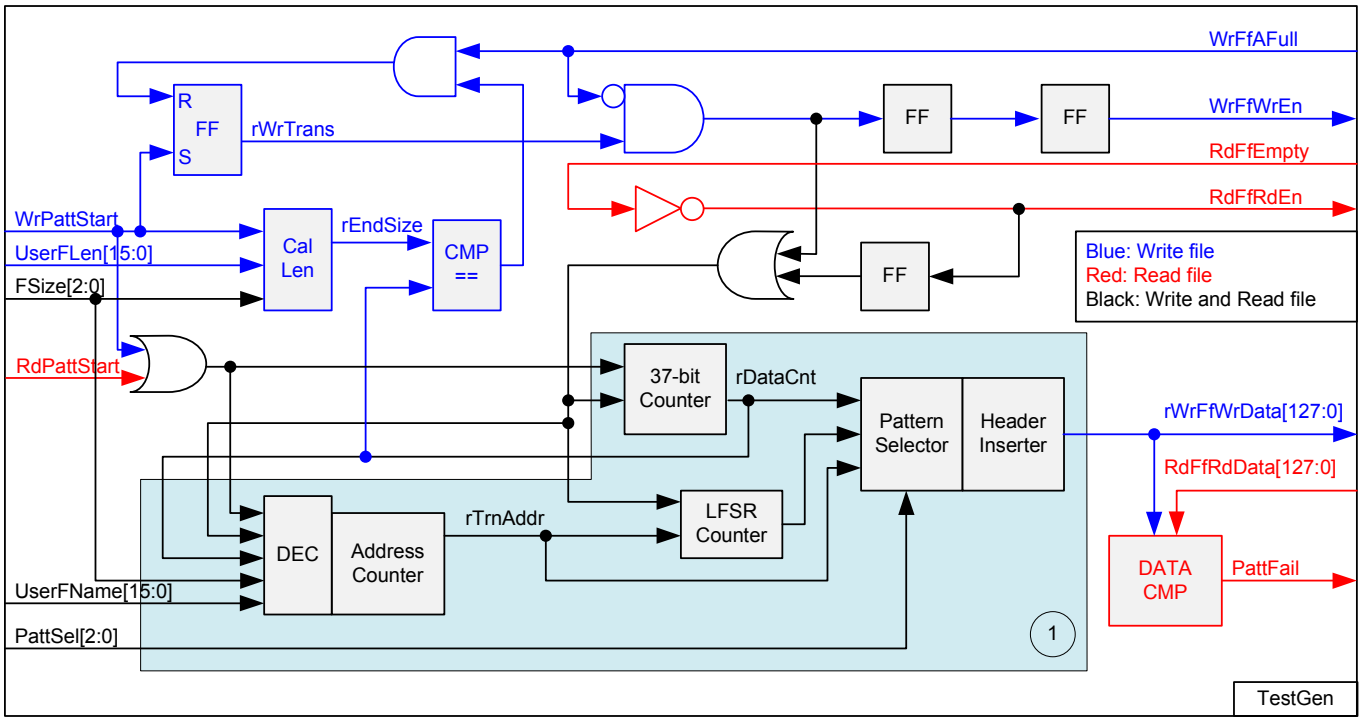

Figure 2-2 TestGen hardware

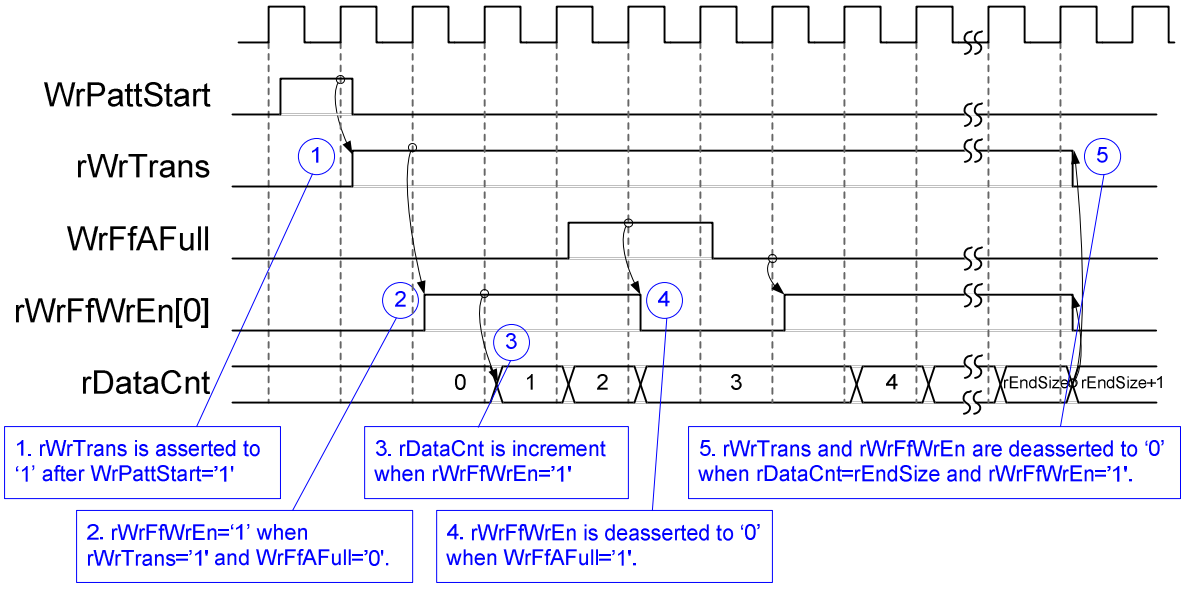

Figure 2-3 Timing diagram of Write operation in TestGen

To start Write file operation, rWrTrans is asserted to '1' when WrPattStart from LAxi2Reg is asserted to '1' for one clock cycle. After that, rWrFfWrEn[0] is asserted to '1' to send test data to WrFf when rWrTrans='1' and WrFfAFull='0'. If WrFfAFull='1', rWrFfWrEn[0] will be de-asserted to '0' to pause data transferring. rDataCnt is data counter to check total transfer size, increased by rWrFfWrEn[0]. When total data are transferred completely (rDataCnt=rEndSize), rWrTrans and rWrFfWrEn[0] are de-asserted to '0' to stop data transferring. rEndSize is total transfer size in sector unit which is calculated by UserFLen x File size (File size is decoded from FSize signal).

For Read file operation, RdFfRdEn signal is designed by connecting NOT logic to RdFfEmpty. rDataCnt is increased when RdFfRdEn is asserted to '1'. In Read operation, rDataCnt is used to generate test pattern for verifying with the received data (RdFfRdData).

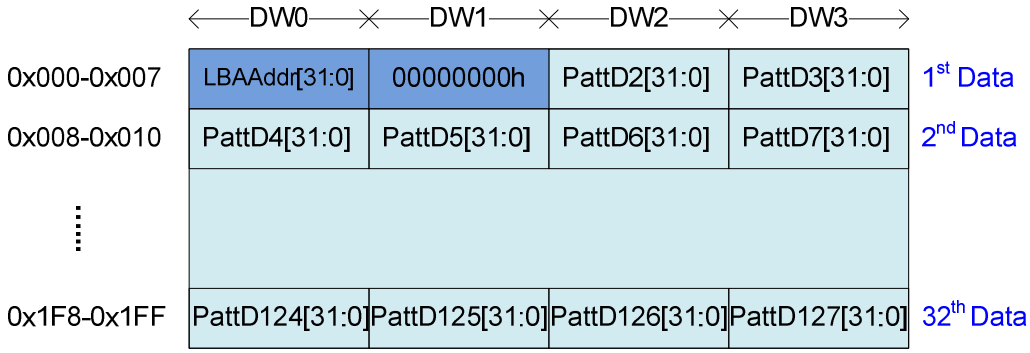

Figure 2-4 Test pattern format in each sector

Block no.1 in lower side of Figure 2-2 shows the logic for generating test pattern in TestGen module. To create unique test data for each sector, test pattern is designed as shown in Figure 2-4.

Test pattern consists of two parts, i.e. 64-bit header in word#0 and word#1 of each sector and test data in word#2 – word#127. 64-bit header is created by using LBA address value of the data (LBA address is the address in sector unit). As shown in Figure 2-2, UserFName and FSize are calculated to be initial value of rTrnAddr (Initial value = UserFName x File size). rTrnAddr is applied to be the header at DW0 of each sector and increased after completing to transfer one sector data. rDataCnt and write/read enable signal are monitored to check end of sector transferring.

TestGen supports to generate five patterns, i.e. 32-bit increment, 32-bit decrement, all 0, all 1, and 32-bit LFSR. 32-bit increment is generated by using rTrnAddr and rDataCnt. Decrement pattern is designed by using NOT logic with increment data. To create 32-bit LFSR counter, 128-bit data is calculated by using look-ahead logic to complete four LFSR data in one clock cycle as shown in Figure 2-5.

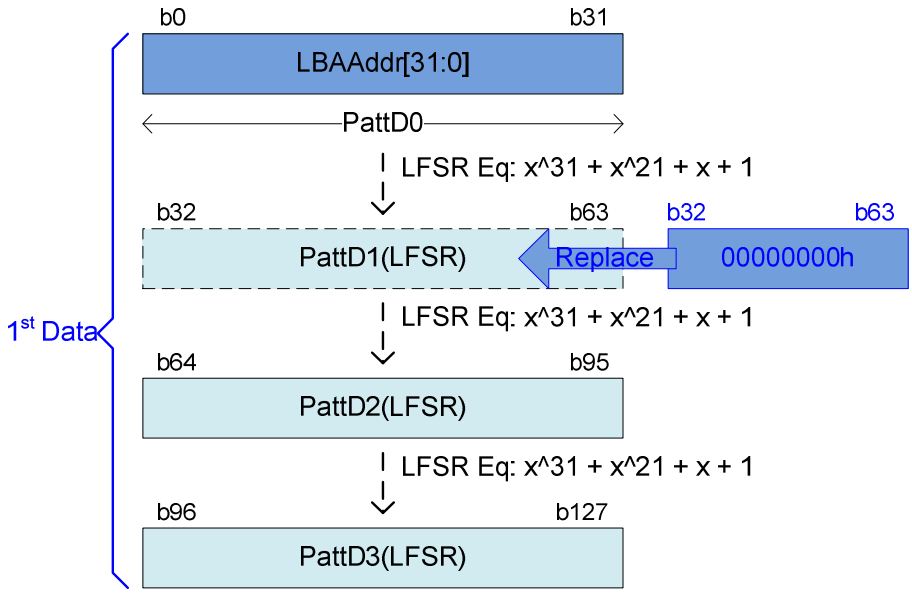

Figure 2-5 LFSR pattern in TestGen

Start value of LFSR is designed by using 32 lower bit of LBA Address. PattD1 is used to calculate the next LFSR data only, but finally it is replaced by 32 upper bit of LBA Address (all 0 value) to complete 64-bit header in each sector.

3-bit PattSel signal are used to select one of five test patterns. Header Inserter inserts 64-bit header to be the 1st data of each sector. After that, test data from pattern counter is transferred to be rWrFfWrData. In Read command, rWrFfWrData is used to be expected value to compare with read data from FIFO (RdFfRdData). PattFail is asserted to '1' when data verification is failed.

2.2 FAT32

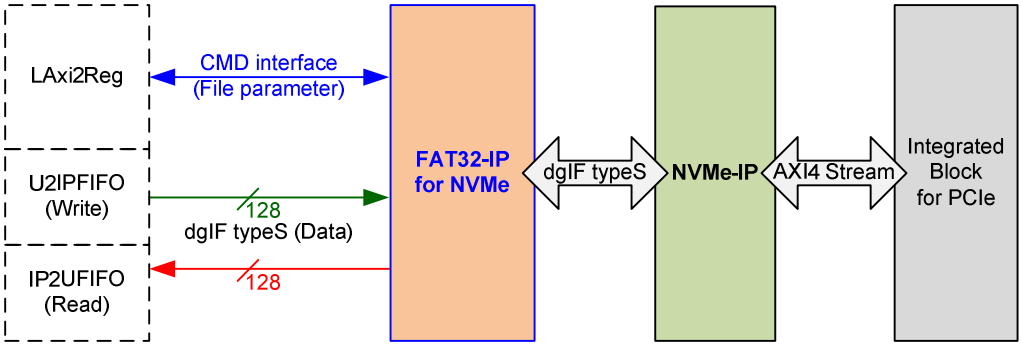

Figure 2-6 FAT32 hardware

As shown in Figure 2-6, user interface of FAT32-IP is split into two groups. CMD interface is connected to LAxi2Reg to receive file parameter from user through Serial console. Data bus size is 128-bit and connects with U2IPFIFO and IP2UFIFO. Another side of FAT32-IP is connected to NVMe-IP.

2.2.1 FAT32-IP for NVMe

FAT32-IP implements the logic to handle data in SSD following FAT32 file system. FAT32-IP is designed to connect with NVMe-IP by using 128-bit data bus. More details of FAT32-IP for NVMe are described in datasheet.
http://www.dgway.com/products/IP/NVMe-IP/dg_fat32ip_nvme_data_sheet_en.pdf

2.2.2 NVMe-IP

NVMe-IP implements NVMe protocol of Host side to access NVMe SSD. User interface is simple designed by using dgIF typeS format. NVMe-IP is designed to connect with Integrated Block for PCIe which is Hard IP in Xilinx device. More details of NVMe-IP are described in datasheet.
http://www.dgway.com/products/IP/NVMe-IP/dg_nvme_ip_data_sheet_en.pdf

2.2.3 Integrated Block for PCIe

This is Hard IP in Xilinx device which implements the lower layer of PCIe protocol. The interface of the IP is different for each FPGA model. Please see more details from Xilinx website.
https://www.xilinx.com/products/technology/pci-express.html

2.3 CPU and Peripherals

The hardware is connected to CPU through AXI4-Lite bus, similar to other CPU peripherals. The hardware registers are mapped to CPU memory address, as shown in Table 2-1. The control and status registers for CPU access are designed in LAxi2Reg.

LAxi2Reg connects to many hardwares in the system such as TestGen, FAT32-IP to get the control and status signals of each module. As shown in Figure 2-7, there are two clock domains applied in this block, i.e. CpuClk which is used to interface with AXI4-Lite bus of CPU and UserClk which is user clock domain for TestGen and NVMe-IP.

AsyncAxiReg includes asynchronous circuit between CpuClk and UserClk. More details of each hardware are described as follows.

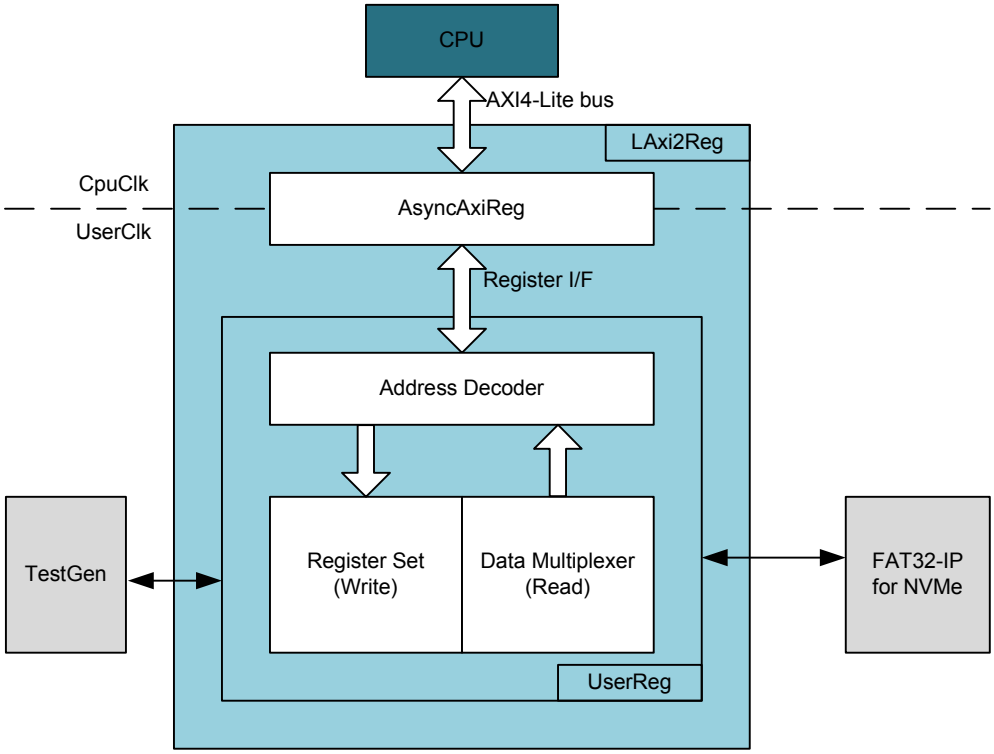

Figure 2-7 CPU and peripherals hardware

2.3.1 AsyncAxiReg

This module is designed to convert the signal interface of AXI4-Lite to be register interface. Also, it supports to convert clock domain from CpuClk to be UserClk domain. Timing diagram of register interface is shown in Figure 2-8.

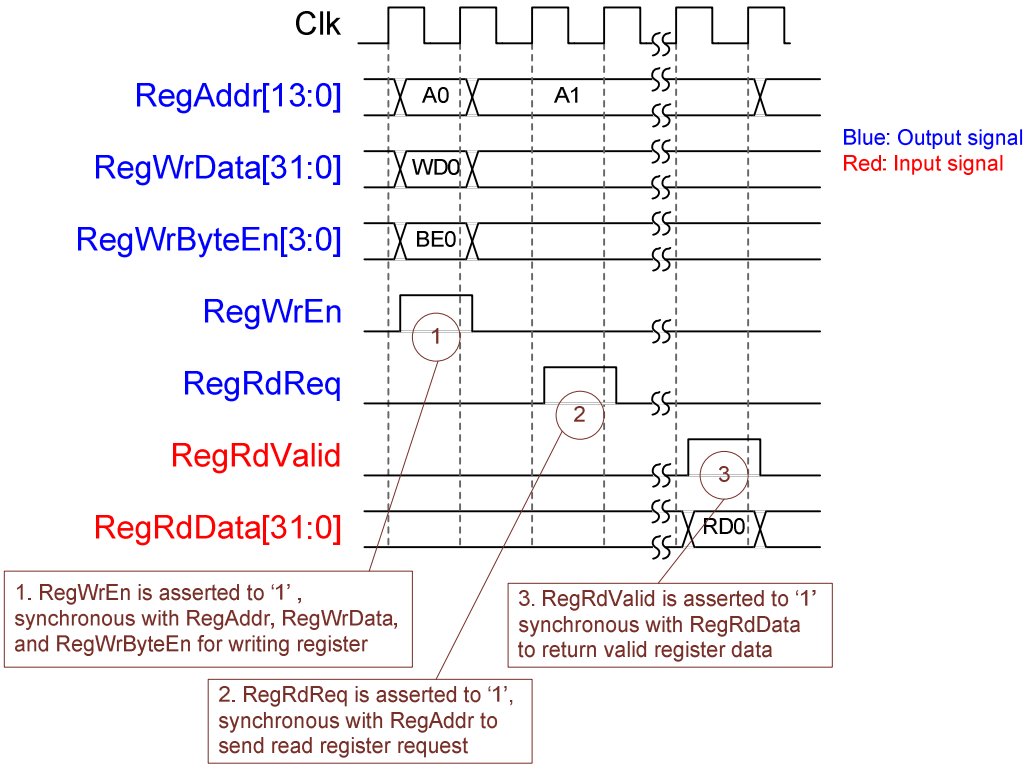

Figure 2-8 Register interface timing diagram

To write register, timing diagram is same as 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 byte enable of this access: bit[0] is write enable for RegWrData[7:0], bit[1] is used for RegWrData[15:8], …, and bit[3] is used for RegWrData[31:24]).

To read register, AsyncAxiReg asserts RegRdReq='1' with the valid value of RegAddr (the register address in 32-bit unit). After that, the module waits until RegRdValid is asserted to '1' to get the read data through RegRdData signal. During read access, RegAddr holds the same value until RegRdValid is asserted to '1'.

2.3.2 UserReg

The details of UserReg module is shown in Figure 2-7. After RegWrEn or RegRdReq is asserted to '1' by AsyncAxiReg to request write or read register access, RegAddr is read by Address decoder to select the active register. For write access, RegWrData signal is loaded to be the new value for the requested register. In this module, RegWrByteEn is not used, so CPU firmware needs to access the hardware register by using 32-bit pointer only.

For read request, there are many status signals for CPU access such as TestGen, FAT32-IP. So, data multiplexer with pipelines register are designed to select the read data to return to CPU following RegAddr value. RegRdValid is designed by using two D Flip-flops, input by RegRdReq signal. So, the read access has two clock cycles latency, measured by the delay time from RegRdReq to RegRdValid.

Memory map of control and status signals inside UserReg module is shown in Table 2-1.

Table 2-1 Register Map

Address Rd/Wr	Register Name (Label in the “fat32nvmetest.c”)	Description
BA+0x00 Wr	User File Name Reg (USERFNAME_REG)	[15:0]: Input to be UserFName of FAT32-IP for NVMe
BA+0x04 Wr	User File Length Reg (USERFLEN_REG)	[15:0]: Input to be UserFLen of FAT32-IP for NVMe
BA+0x08 Wr	File Size Reg (FSIZE_REG)	[2:0]: Input to be FSize of FAT32-IP for NVMe
BA+0x0C Wr	File Time Reg (DATETIME_REG)	[4:0]: Input to be FTimeS of FAT32-IP for NVMe [10:5]: Input to be FTimeM of FAT32-IP for NVMe [15:11]: Input to be FTimeH of FAT32-IP for NVMe [20:16]: Input to be FDateD of FAT32-IP for NVMe [24:21]: Input to be FDateM of FAT32-IP for NVMe [31:25]: Input to be FDateY of FAT32-IP for NVMe
BA+0x10 Wr	User Command Reg (USERCMD_REG)	[1:0]: Input to be UserCmd of FAT32-IP for NVMe When this register is written, the design generates UserReq (command request) to FAT32-IP for NVMe to start new command operation.
BA+0x14 Wr	Pattern Select Reg (PATTSEL_REG)	[2:0]: Test pattern select “000”-Increment, “001”-Decrement, “010”-All 0, “011”-All 1, “100”-LFSR
BA+0x100 Rd	User Status Reg (USRSTS_REG)	[0]: Mapped to UserBusy of FAT32-IP for NVMe [1]: Mapped to UserError of FAT32-IP for NVMe [2]: Data verification fail (‘0’: Normal, ‘1’: Error)
BA+0x104 Rd	Total file capacity Reg (TOTALFCAP_REG)	[15:0]: Mapped to TotalFCap of FAT32-IP for NVMe
BA+0x108 Rd	FAT32-IP Test pin (Low) Reg (FATTESTPINL_REG)	[31:0]: Mapped to TestPin[31:0] of FAT32-IP for NVMe
BA+0x10C Rd	FAT32-IP Test pin (High) Reg (FATTESTPINH_REG)	[31:0]: Mapped to TestPin[63:32] of FAT32-IP for NVMe
BA+0x110 Rd	User Error Type Reg (USRERRTYPE_REG)	[31:0]: Mapped to UserErrorType of FAT32-IP for NVMe
BA+0x114 Rd	NVMe Completion Status Reg (COMPSTS_REG)	[15:0]: Mapped to AdmCompStatus[15:0] of NVMe-IP [31:16]: Mapped to IOCompStatus[15:0] of NVMe-IP
BA+0x118 Rd	NVMe CAP Reg (NVMCAP_REG)	[31:0]: Mapped to NVMeCAPReg[31:0] of NVMe-IP
BA+0x11C Rd	NVMe-IP Test pin Reg (NVMTESTPIN_REG)	[31:0]: Mapped to TestPin[31:0] of NVMe-IP
BA+0x200 Rd	Expected value Word0 Reg (EXPPATW0_REG)	[31:0]: Expected data [31:0] of failure address.
BA+0x204 Rd	Expected value Word1 Reg (EXPPATW1_REG)	[31:0]: Expected data [63:32] of failure address.
BA+0x208 Rd	Expected value Word2 Reg (EXPPATW2_REG)	[31:0]: Expected data [95:64] of failure address.
BA+0x20C Rd	Expected value Word3 Reg (EXPPATW3_REG)	[31:0]: Expected data [127:96] of failure address.

Address Rd/Wr	Register Name (Label in the “fat32nvmetest.c”)	Description
BA+0x210 Rd	Read value Word0 Reg (RDPATW0_REG)	[31:0]: Read data [31:0] of failure address.
BA+0x214 Rd	Read value Word1 Reg (RDPATW1_REG)	[31:0]: Read data [63:32] of failure address.
BA+0x218 Rd	Read value Word2 Reg (RDPATW2_REG)	[31:0]: Read data [95:64] of failure address.
BA+0x21C Rd	Read value Word3 Reg (RDPATW3_REG)	[31:0]: Read data [127:96] of failure address.
BA+0x220 Rd	Failure Byte Address Reg (FAILADDR_REG)	[31:0]: Failure data address in the file
BA+0x224 Rd	Failure File Name Reg (FAILFNAME_REG)	[15:0]: Failure file name
BA+0x228 Rd	Current test byte (Low) Reg (CURTESTSIZEL_REG)	[31:0]: Current test data size of TestGen module in byte unit (bit[31:0])
BA+0x22C Rd	Current test byte (High) Reg (CURTESTSIZEH_REG)	[8:0]: Current test data size of TestGen module in byte unit (bit[40:32])

3 CPU Firmware

After system boot-up, CPU initializes its peripherals such as UART and Timer. Next, CPU waits until FAT32-IP completes initialization process (USRSTS_REG[0]='0'). After that, maximum file to store in the SSD (reading from TOTALFCCAP_REG) is displayed on the console. This value is calculated by setting file size = 32 MB (FSIZE_REG="000"). User can run "Change file size" menu to change file size and maximum file to store in the SSD.

CPU firmware supports four menus. Menu 1-3 is designed to run three commands following USRCMD_REG value, i.e. "00" for Format command, "10" for Write file command, and "11" for Read file command. Menu 4 is designed to change file size in the SSD. More details of the sequence in each operation are described as follows.

3.1 Format

The sequence of the firmware when user selects Format menu is below.

1) Set USERCMD_REG="00" to format the SSD. FAT32 IP busy flag (USRSTS_REG[0]) changes from '0' to '1' after operating Format command.
2) CPU waits until the operation is completed or the error is found by monitoring USRSTS_REG value. Bit[0] is de-asserted to '0' when command is completed. Bit[1] is asserted to '1' when some errors are detected. In case of error condition, there is error message displayed on the console. If the command is completed, maximum file storing in the SSD (TOTALFCAP_REG) will be displayed on the console.

3.2 Write file/Read file command

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

1) Receive file name, total files, and test pattern through Serial console. If some inputs are invalid, the operation will be cancelled. In case of Write file operation, user can change created date and time of file by updating DATETIME_REG.
2) Get all inputs and set the value to USERFNAME_REG, USERFLEN_REG, PATTSEL_REG, and USRCMD_REG (USRCMD_REG="10" for Write file and "11" for Read file).
3) CPU waits until the operation is completed or some errors (except verification error) are found by monitoring USRSTS_REG[2:0].
   If USRSTS_REG[2] is asserted to '1', verification error message will be displayed. After that, CPU still runs until end of operation or user inputs any key to cancel operation.
4) During running command, current transfer size reading from CURTESTSIZE_REG is displayed every second. Finally, test performance is displayed on Serial console when command is completed.

3.3 Change file size

The sequence of the firmware when user selects Change file size is below.

1) Receive new file size value from user through Serial console. If the input is valid and UserBusy='0', the new value will be set to FSIZE_REG.
2) CPU displays maximum file storing in the SSD by reading TOTALFCAP_REG. Warning message is displayed to recommend user to format the SSD after changing file size.

4 Example Test Result

The example test result when running demo system by using 512 GB Samsung 960 Pro is shown in Figure 4-1.

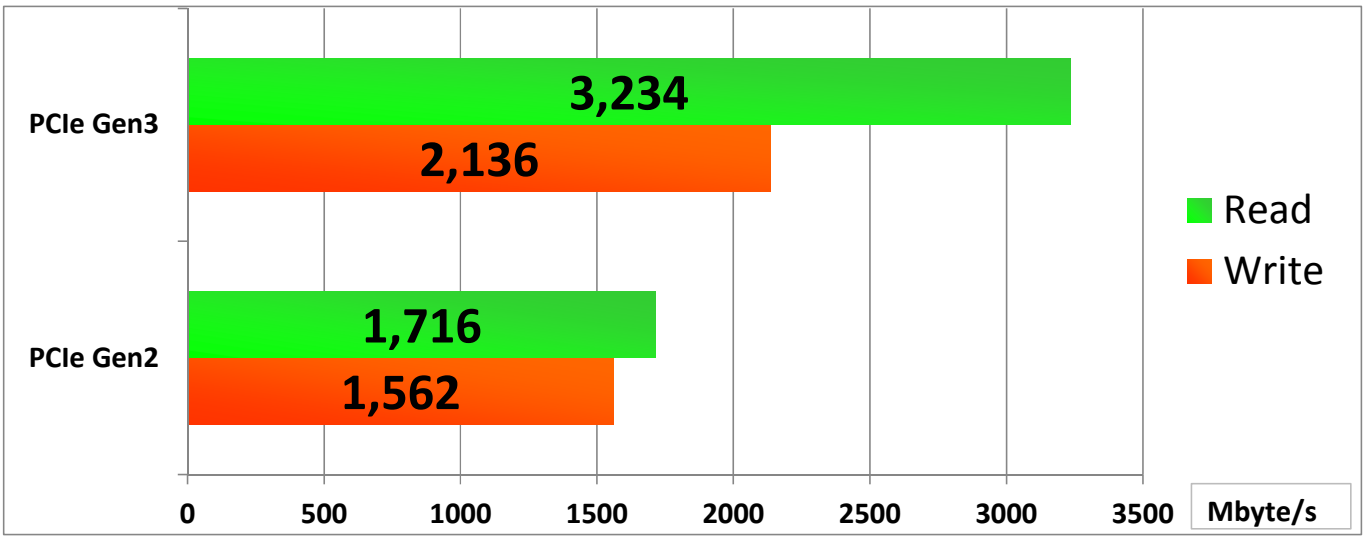

Figure 4-1 Test Performance of FAT32-IP demo for NVMe by using Samsung 960 Pro SSD

By using PCIe Gen3 on KCU105 board, write performance is about 2136 Mbyte/sec and read performance is about 3234 Mbyte/sec. Using PCIe Gen2 on ZC706 board, write performance is about 1562 Mbyte/sec and read performance is about 1716 Mbyte/sec.

5 Revision History

Revision	Date	Description
1.0	13-Mar-18	Initial version release
1.1	14-May-18	Update details of reference design