NVMe-IP reference design manual

Rev1.1 16-Dec-16

1. NVMe

NVM Express (NVMe) defines the interface for the host controller to access solid state drives (SSD) by PCI Express. NVM Express optimizes the process to issue command and completion by using only 2 register writes for command issue/completion cycle. Also, NVMe can support parallel operation by supporting up to 64K commands within single queue. So, performance for both sequential and random access is improved.

In PCIe SSD market, user can find two standards, i.e. AHCI and NVMe. AHCI is the older standard to provide the interface for SATA hard disk drives while NVMe is optimized for non volatile memory like SSD. The comparison between both protocols in more details can be found from "A Comparison of NVMe and AHCI" document.
https://sata-io.org/system/files/member-downloads/NVMe%20and%20AHCI_%20_long_.pdf

The list of NVMe storage devices can be found from http://www.nvmexpress.org/products/.

Generally, user needs to install NVMe driver to access NVMe SSD as shown in Figure 1. Physical connector of NVMe SSD is PCIe type such as M.2 connector. NVMe-IP implements NVMe driver and the task running on CPU by pure-hardware logic. So, CPU is not required to access NVMe SSD when using NVMe-IP in FPGA board.

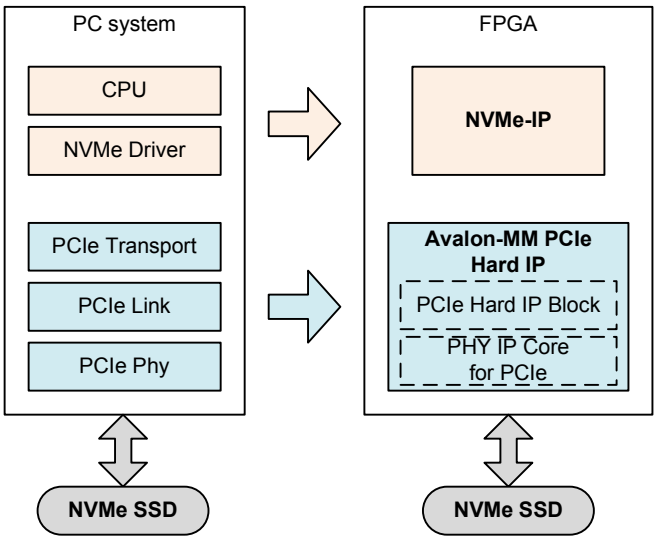

Figure 1 NVMe protocol layer

2. Overview

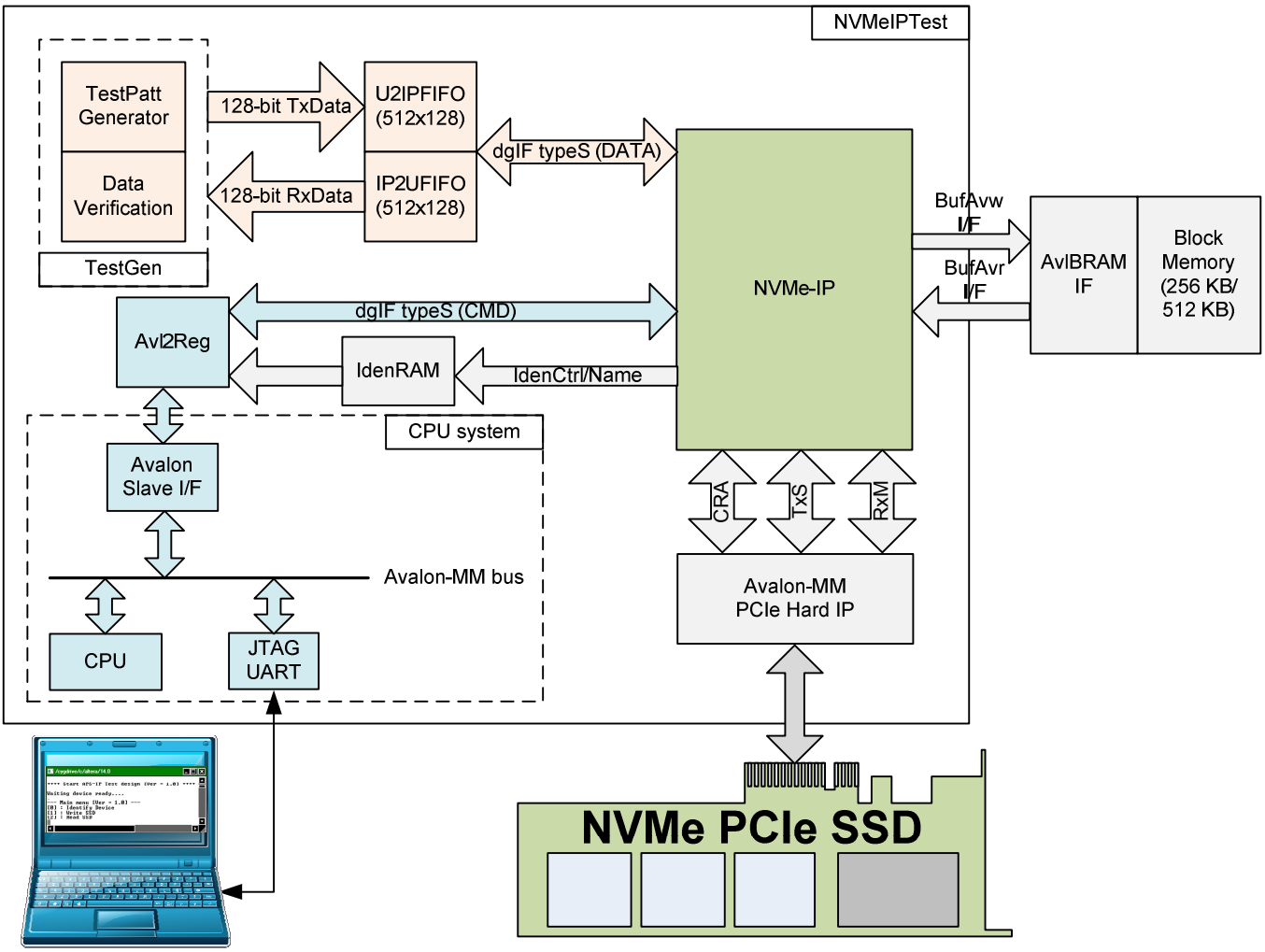

Figure 2 NVMe-IP Demo System

The reference design integrates NVMe-IP with simple logic to write and read data with NVMe PCIe SSD at high-speed rate. CPU is additional designed for user interface through JTAG UART. AvlBRAMIF is designed to convert Avalon bus interface from NVMe-IP to Block Memory interface which is applied to be data buffer in the design.

For simple test, user can input the parameters such as start address, transfer size, and command to NiosII command shell. The logic will decode all inputs and convert to be input value for NVMe-IP. When the operation is completed, CPU will check time usage and then calculate to be write/read performance of the SSD. To interface with CPU bus, Avl2Reg module is used to decode the address and data from the bus to be command interface of dgIF typeS. Data interface of dgIF typeS are connected to external FIFO and transferred to data buffer (Block Memory) through AvlBRAMIF. NVMe-IP can connect to two buffer sizes, i.e. 256 Kbyte (Mode=1) or 512 Kbyte (Mode=2). 512 Kbyte can achieve the better transfer performance. Test data for write test and for data verification are generated by TestGen module. IdenCtrl/IdenName data are transferred to IdenRAM, so CPU can decode SSD model name by using Identify data.

NVMe-IP and TestGen modules run in the same clock domain which is source from internal PLL of Avalon-MM PCIe Hard IP. This clock is equal to 125 MHz when interfacing with PCIe Gen2 SSD or 250 MHz when interfacing with PCIe Gen3 SSD.

User can download NVMe-IP datasheet and send request to evaluate the IP from our website, http://www.dgway.com/NVMe-IP_A_E.html.
The real transfer performance in the demo depends on each NVMe PCIe SSD characteristic.

3. CPU and Peripheral

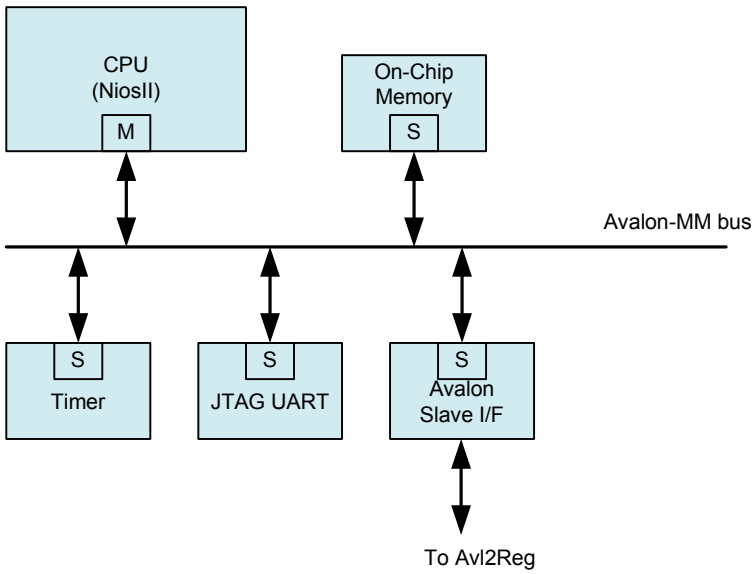

Figure 3 CPU system in reference design

In reference design, CPU peripherals consist of JTAG UART for user interface, Timer for performance measurement, and Memory for CPU firmware. Avalon Slave I/F is connected to Avalon-MM bus for CPU controlling/monitoring test system. More details about memory map for CPU to access Avalon-MM bus are follows.

Table 1 Register Map

Address Rd/Wr	Register Name (Label in the “nvmeiptest.c”)	Description
BA+0x00 Wr	User Address (Low) Reg (USRADRL_REG)	[31:0]: Input to be start sector address (UserAddr[31:0] for dgIF typeS)
BA+0x04 Wr	User Address (High) Reg (USRADRH_REG)	[15:0]: Input to be start sector address (UserAddr[47:32] for dgIF typeS)
BA+0x08 Wr	User Length (Low) Reg (USRLENL_REG)	[31:0]: Input to be transfer length in sector unit (UserLen[31:0] for dgIF typeS)
BA+0x0C Wr	User Length (High) Reg (USRLENH_REG)	[15:0]: Input to be transfer length in sector unit (UserLen[47:32] for dgIF typeS)
BA+0x10 Wr	User Command Reg (USRCMD_REG)	[1:0]: Input to be user command (UserCmd for dgIF typeS) “00”-Identify device, “10”-Write SSD, “11”-Read SSD When this register is written, the design will generate command request to NVMe-IP to start new command operation.
BA+0x14 Wr	Test Pattern Reg (PATTSEL_REG)	[1:0]: Test pattern select “00”-Increment, “01”-Decrement, “10”-All 0, “11”-All 1
BA+0x18 Wr	User Reset Reg (USRRST_REG)	[0]: ‘1’-Reset test system, ‘0’-Release reset

Address Rd/Wr	Register Name (Label in the “nvmeiptest.c”)	Description
BA+0x100 Rd	User Status Reg (USRSTS_REG)	[0]: UserBusy of dgIF typeS (‘0’: Idle, ‘1’: Busy) [1]: UserError of dgIF typeS (‘0’: Normal, ‘1’: Error) [2]: Data verification fail (‘0’: Normal, ‘1’: Error) [4:3]: PCIe speed from IP (“00”: No linkup, “01”: PCIe Gen1, “10”: PCIe Gen2, “11”: PCIe Gen3)
BA+0x104 Rd	Total disk size (Low) Reg (LBASIZEL_REG)	[31:0]: Total capacity of SSD in sector unit (LBASize[31:0] from dgIF typeS)
BA+0x108 Rd	Total disk size (High) Reg (LBASIZEH_REG)	[15:0]: Total capacity of SSD in sector unit (LBASize[47:32] from dgIF typeS)
BA+0x10C Rd	User Error Type Reg (USRERRTYPE_REG)	[31:0]: User error status (UserErrorType[31:0] from dgIF typeS)
BA+0x114 Rd	Completion Status Reg (COMPSTS_REG)	[15:0]: Status from Admin completion (AdmCompStatus[15:0] from NVMe-IP) [31:16]: Status from IO completion (IOCompStatus[15:0] from NVMe-IP)
BA+0x120 Rd	Data Failure Address(Low) Reg (RDFAILNOL_REG)	[31:0]: Latch value of failure address[31:0] in byte unit from read command
BA+0x124 Rd	Data Failure Address(High) Reg (RDFAILNOH_REG)	[24:0]: Latch value of failure address [56:32] in byte unit from read command
BA+0x130 Rd	Expected value Word0 Reg (EXPPATW0_REG)	[31:0]: Latch value of expected data [31:0] from read command
BA+0x134 Rd	Expected value Word1 Reg (EXPPATW1_REG)	[31:0]: Latch value of expected data [63:32] from read command
BA+0x138 Rd	Expected value Word2 Reg (EXPPATW2_REG)	[31:0]: Latch value of expected data [95:64] from read command
BA+0x13C Rd	Expected value Word3 Reg (EXPPATW3_REG)	[31:0]: Latch value of expected data [127:96] from read command
BA+0x140 Rd	Read value Word0 Reg (RDPATW0_REG)	[31:0]: Latch value of read data [31:0] from read command
BA+0x144 Rd	Read value Word1 Reg (RDPATW1_REG)	[31:0]: Latch value of read data [63:32] from read command
BA+0x148 Rd	Read value Word2 Reg (RDPATW2_REG)	[31:0]: Latch value of read data [95:64] from read command
BA+0x14C Rd	Read value Word3 Reg (RDPATW3_REG)	[31:0]: Latch value of read data [127:96] from read command
BA+0x150 Rd	Current test byte (Low) Reg (CURTESTSIZEL_REG)	[31:0]: Current test data size of TestGen module in byte unit (bit[31:0])
BA+0x154 Rd	Current test byte (High) Reg (CURTESTSIZEH_REG)	[24:0]: Current test data size of TestGen module in byte unit (bit[56:32])
BA+0x2000 – 0x2FFF	Identify Controller Data (IDENCTRL_REG)	4Kbyte Identify Controller Data Structure
BA+0x3000 – 0x3FFF	Identify Namespace Data (IDENNAME_REG)	4Kbyte Identify Namespace Data Structure

After initialization complete, CPU firmware in the demo will be in idle state to wait user command input through NiosII command shell. The command can be Identify device, write, or read command. The sequence of each command is follows.

For Identify device command,

1) Set USRCMD_REG=”00”. Test logic will generate command and request to NVMe-IP. Busy flag (USRSTS_REG[0]) will change from ‘0’ to ‘1’.
2) CPU will wait until command complete or any error found by monitoring USRSTS_REG value. Bit[0] will be cleared to ‘0’ when command is completed. Bit[1] will be asserted to ‘1’ when any error is detected. If any error is detected, error message will be displayed.
3) To be test result, SSD model name decoded from IDENCTRL_REG and SSD capacity read from LBASIZEL/H_REG are displayed to the command shell.

For write/read command,

1) Receive start address, transfer length, and test pattern value from user through command shell. If any input is invalid, the operation will be cancelled.
2) Get all inputs and set the value to USRADRL/H_REG, USRLENL/H_REG, and USRCMD_REG (USRCMD_REG=”10” for write transfer, and “11” for read transfer).
3) Similar to step 2) in Identify device command. But USRSTS_REG[2] will be also monitored for read command to confirm that read data is correct.
4) During running command, current transfer size will be displayed every second. Finally, test performance will be displayed on the command shell when command is completed.

4. AvlBRAMIF

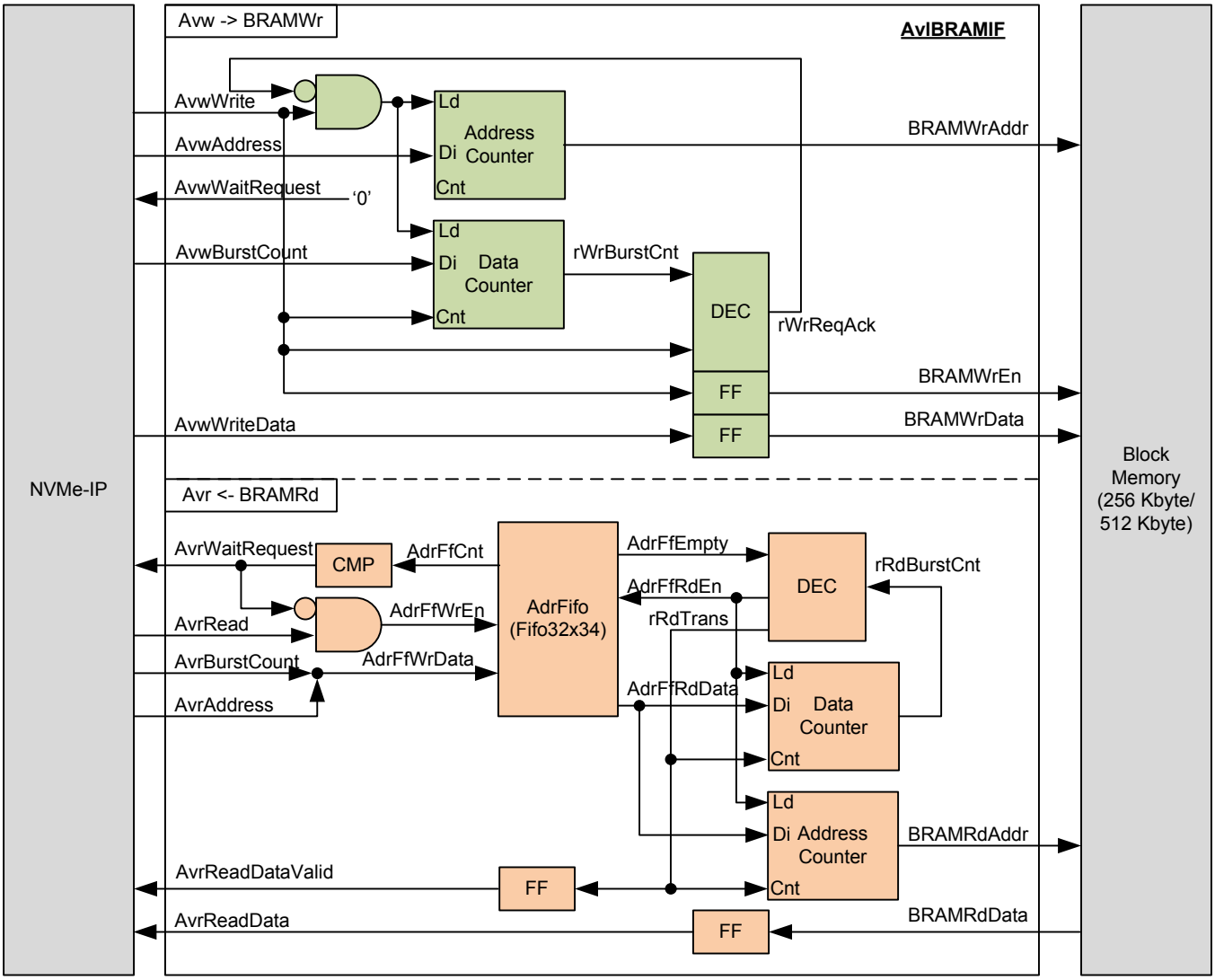

Figure 4 Logic design in AvlBRAMIF

Logic design in AvlBRMAIF can be split into two parts, i.e. Write part which controls data transfer from Avw to Block Memory, and Read port which controls data transfer from Block Memory to Avr.

For write transfer, AvwWaitRequest is always set to '0', so new data from NVMe-IP can be sent to Block Memory without waiting state. Following Avalon standard, Start write address (AvwAddress) and transfer length (AvwBurstCount) are loaded to internal counter at the 1st clock of each burst transfer. AvwWrite is used to be counter enable for both Address and data counter. Address counter is designed to be count-up to generate next write address of Block memory during burst transfer. Data counter is count-down logic to check the end position of each transfer. rWrReqAck is designed to assert to '1' only the 1st clock of each burst, so it is used to be load enable for both Address and Data counter. BRAMWrEn and BRAMWrData are fed directly from AvwWrite and AvwWriteData.

For read transfer, AdrFifo is included to store start address and transfer length of each request from NVMe-IP. If FIFO is not full, AvrWaitRequest will be equal to ‘0’ to receive the next request from NVMe-IP. So, NVMe-IP can send many read requests without waiting data strobe of previous request. FIFO depth is equal to 32 and implemented by the logic, not Block Memory. FIFO counter is monitored to check almost full condition which will assert AvrWaitRequest to ‘1’ to hold the new request.

For data transfer in read request, it will go to data transfer state if new request is available in AdrFifo (AdrFfEmpty=’0’). AdrFfRdEn will be asserted for one clock to read address and size of one request, and then rRdTrans will change to ‘1’ to start data transfer. The address of the request will be loaded to Address counter while transfer length will be loaded to Data counter. Similar to write transfer, Address counter is count-up mode for read address generating and Data counter is count-down mode for data counting to check end of burst position. rRdTrans is de-asserted to ‘0’ at the end of burst transfer. So, rRdTrans is the main control signal which is used to be counter enable and ReadDataValid. BRAMRdData is directly forwarded to be AvrReadData.

5. Avl2Reg

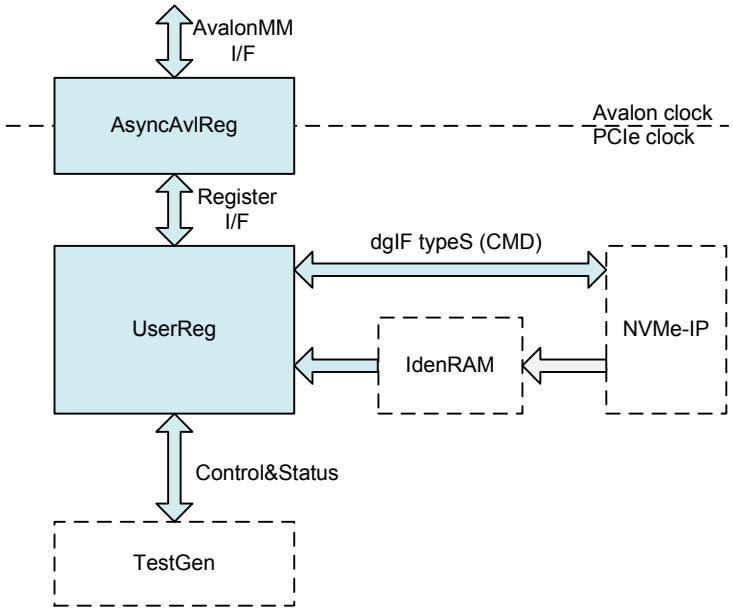

Figure 5 Avl2Reg interface

This module consists of two submodules, i.e. AsyncAvlReg and UserReg. AsyncAvlReg is designed to convert Avalon interface to be register interface and convert clock domain from Avalon clock to user clock system. UserReg module is designed to decode write/read address which is mapped following Table 1. Transfer parameters such as transfer direction, size, and address from user will be converted to be command interface of dgIF typeS for NVMe-IP and converted to be control signal for TestGen module. During transferring, CPU can read the register to check NVMe-IP status, TestGen result, and Identify device data.

6. TestGen

In this module, there are two operations, i.e. generating test data to WrFf port when user selects write command, or verifying received data from RdFf port when user selects read command. The details of logic design inside this module are displayed in Figure 6.

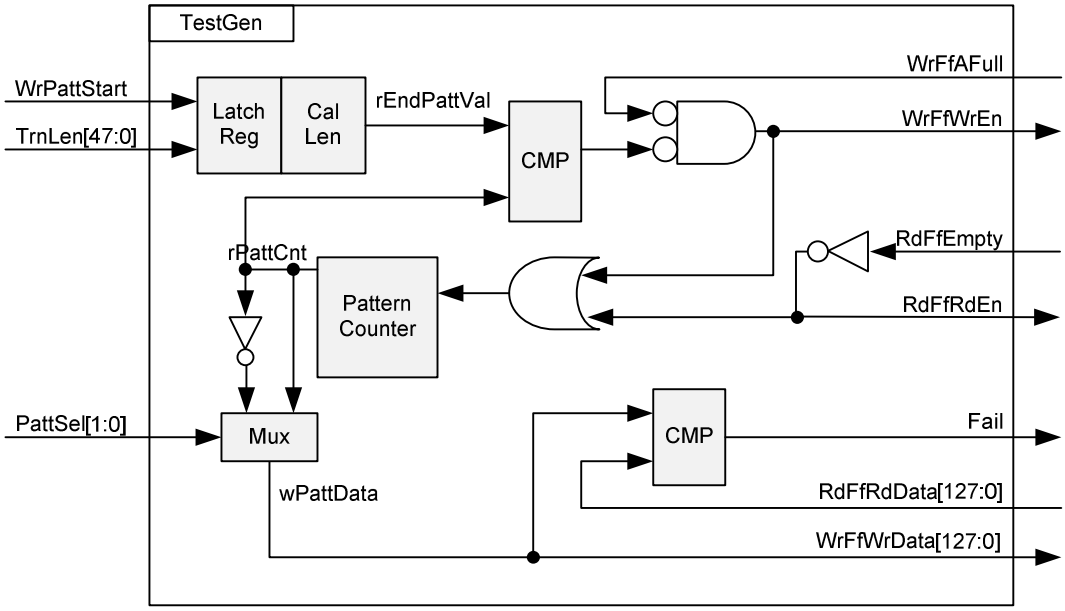

Figure 6 Logic design in TestGen

For write transfer, test pattern will be generated from Pattern counter module after WrPattStart is asserted. WrFfAFull is monitored to confirm that WrFf still have free space to store new test data. Test data pattern will be fed to WrFf when FIFO is available and stopped when total transfer size is equal to set value from user. TrnLen is the input to show total transfer size in sector unit and used to calculate the end value of test data pattern for stopping data generating. Four test patterns can be selected through PattSel input, i.e. increment, decrement, all 0, and all 1 value.

For read transfer, read enable of RdFf is asserted when FIFO has available data, monitored by RdFfEmpty signal. Test data generator is used to generate expected data to compare and verify RdFfRdData value. If data is mismatched, Fail flag will be asserted.

7. Revision History

Revision	Date	Description
1.0	5-Aug-16	Initial Release
1.1	16-Dec-16	Change buffer from DDR to Block Memory