NVMe to AXI Bridge

Technical Description

Eidetic Communications Inc. 3553 31st NW, Calgary, Alberta T2L 2K7 www.eideticom.com info@eideticom.com

Table of Contents

1 Introduction

This document provides a technical description of the Eideticom NVMe to AXI Bridge. This design is part of the OpenPOWER CAPI SNAP accelerator framework. The NVMe to AXI bridge is used to access data on attached NVMe SSD's. The focus of the document is on the NVMe host subsystem and how to program the NVMe host to transfer data to or from the attached SSD's from or to the onboard DDR memory.

2 System Overview

The NVMe to AXI bridge system is shown below in Figure 1. The system consists of an Eideticom NVMe Host System, Xilinx PCIe Root Complexes, and Xilinx AXI interconnect blocks.

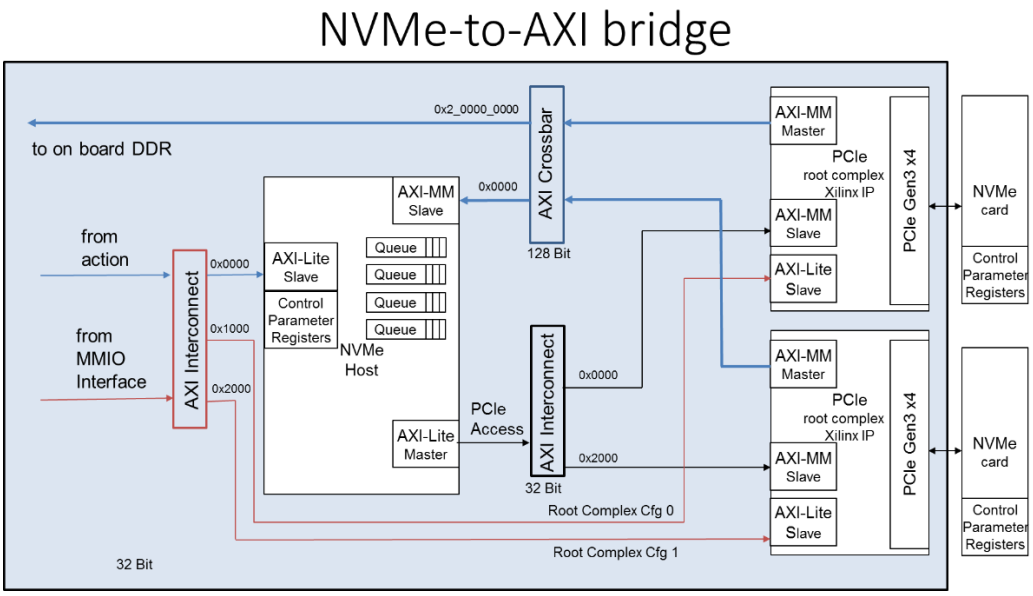

Figure 1: NVMe To AXI Bridge

2.1 Eideticom NVMe Host

The Eideticom NVMe Host system is shown below in Figure 2. This system is responsible for receiving data transfer commands from the CAPI Action Framework and issuing the necessary NVMe requests and receiving the NVMe responses in order to transfer the data to or from the NVMe SSD drives.

The Tx Buffer is used for submission queues and admin command tx data. The Rx Buffer is used for completion queues and for admin command rx data. The completion FSM manages the completion queues and the submission FSM manages the submission queues.

Admin submission queue entries and Tx command data are written directly to the Tx Buffer using the MMIO/Action AXI interface. Admin completion queue entries and Rx command data are read directly from the Rx Buffer using the MMIO/Action AXI interface.

IO submission entries are automatically written to the Tx Buffer by the Submission FSM when a IO command is written using the MMIO/Action AXI interface. IO completion entries are interpreted by the Completion FSM and the completion status is available from the MMIO/Action AXI interface.

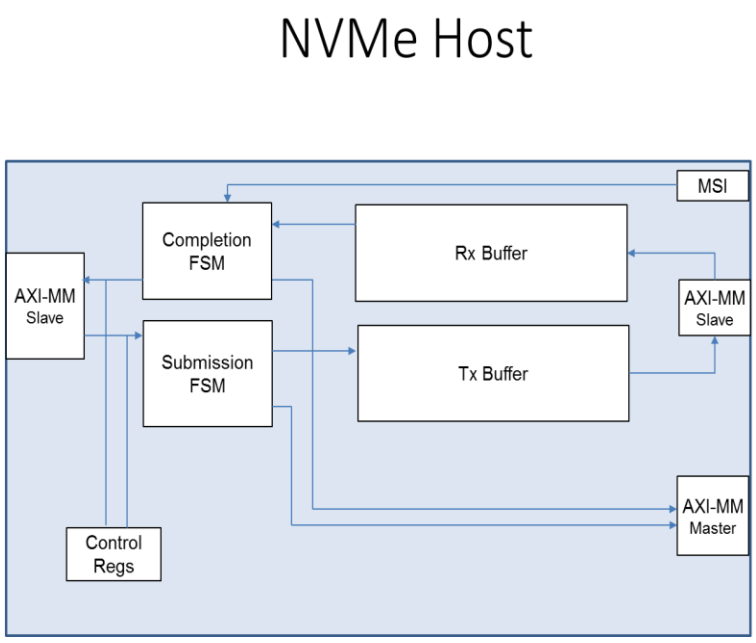

Figure 2: NVMe Host System

2.2 Buffer memory layout

The buffer memory layouts are listed in the tables below. Each of the Tx submission queue entries is 64 bytes in length. Each of the Rx completion queue entries is 16 bytes in length. The PCIE virtual address should be the value used to initialize the queue addresses in the NVME SSD. The PCIE virtual addresses will be automatically mapped to the correct physical address by the NVME host logic.

Table 1 Tx Buffer Layout

Start Address	PCIE Virtual Address	Memory Region
0x6f00	0x28000000	Admin Tx Data (4 kB)
0x3880	0x20000000	IO SQ SSD1 (218 entries)
0x3780	0x18000000	Admin SQ SSD1 (4 entries)
0x100	0x10000000	IO SQ SSD0 (218 entries)
0x0	0x08000000	Admin SQ SSD0 (4 entries)

Table 2 Rx Buffer Layout

Start Address	PCIE Virtual Address	Memory Region
0x1bc0	0x50000000	Admin Rx Data (8 kB)
0xe20	0x48000000	CO SQ SSD1 (218 entries)
0xde0	0x40000000	Admin CQ SSD1 (4 entries)
0x40	0x38000000	IO CQ SSD0 (218 entries)
0x0	0x30000000	Admin CQ SSD0 (4 entries)

2.3 AXI Interconnect Memory Mapping

The MMIO/Action AXI addresses are mapped to locations below.

Address	Memory Region
0x0000	NVMe Host
0x1000	PCIe Config Root Complex 0
0x2000	PCIe Config Root Complex 1

The NVMe Host to PCIe AXI addresses are mapped to the locations below. These locations are accessed using the NVMe Host ADMIN_PCIE_ADDR and HOST_PCIE_DATA registers.

Address	Memory Region
0x0000	PCIe Root Complex 0 PCI Space
0x2000	PCIe Root Complex 1 PCI Space

The PCIe to AXI addresses are mapped to the locations below. These locations are accessed by the PCIE SSD's based on the NVMe command they are processing.

Address	Memory Region
0x0000_0000_0000_0000	NVMe Rx Buffer
0x0000_0002_0000_0000	DDR Access

2.4 NVMe Host Address Space

The NVMe Host address space is mapped as below.

Address	Memory Region
0x00	NVMe Host Action Registers
0x80	NVMe Host Admin Registers
0x100	NVMe Host Buffer Data (HOST_BUFFER_DATA)
0x104	NVMe Host PCIe Data (HOST_PCIE_DATA)

2.4.1 NVMe Host Action Registers

These registers are used by the action for sending commands to read and write data, and for requesting the status of issued commands.

Table 3 Action Write Registers

Offset	Register	Description
0x0	DPTR_LOW	Transfer data pointer low 32 bits
0x4	DPTR_HIGH	Transfer data pointer high 32 bits
0x8	LBA_LOW	SSD LBA low 32 bits
0xC	LBA_HIGH	SSD LBA high 32 bits
0x10	LBA_NUM	Number of LBA blocks in transfer
0x14	COMMAND	Command Register. Writing to this location starts the data transfer.

Table 4 Action Read Registers

Offset	Register	Description
0x0	STATUS	Command completion status for all 16 action fifos and submission queue full flags.
0x4	TRACK_0	Action ID[0] status fifo
0xN*4 + 4	TRACK_N	Action ID[N] status fifo
0x40	TRACK_15	Action ID[15] status fifo

2.4.1.1 Action Command Register

Writing to the action command register will start the data transfer. The bits of this register are defined below.

Bits	Assignment	Meaning
3:0	CMD_TYPE	0 == Read SSD Command 1 == Write SSD Command 2 == Admin Command
7:4	CMD_QUEUE_ID	0 == SSD0 Admin Q 1 == SSD0 IO Q 2 == SSD1 Admin Q 3 == SSD1 IO Q
11:8	CMD_ACTION_ID	Action ID for command
31:12	RESERVED	

2.4.1.2 Action Status Register

Bits	Assignment	Meaning
3:0	Submission Queue Full	Bit 0 – 1 == SSD0 Admin Q Full Bit 1 – 1 == SSD0 IO Q Full Bit 2 – 1 == SSD1 Admin Q Full Bit 3 – 1 == SSD1 IO Q Full
16	Action 0 FIFO Status	0 == Action 0 FIFO empty 1 == Action 0 FIFO has data
16+N	Action N FIFO Status	0 == Action N FIFO empty 1 == Action N FIFO has data
31	Action 15 FIFO Status	0 == Action 15 FIFO empty 1 == Action 15 FIFO has data

2.4.1.3 Action Track Registers

Offset	Register	Description
0x4	TRACK_0	Bit 0 – '1' Action 0 command complete (self-clearing) Bit 1 – '1' Action 0 command returned error (self-clearing) Bits 31:2 -- Reserved
0xN + 4	TRACK_N	Action ID[N] status as above
0x40	TRACK_15	Action ID[15] status as above

2.4.2 Admin Registers

Offset	Name	Assignment
0x0	ADMIN_CONTROL	Bit 0 – Enable NVMe Host Bit 1 – Enable Auto-increment Addressing Bit 2 – Clear Error Status
0x4	ADMIN_STATUS	Bit 0 – NVMe Host Ready Bit 1 – Error Condition Detected Bit 2 – Admin Command to SSD0 Complete

		Bit 3 – Admin Command to SSD1 Complete
0x8	ADMIN_BUFFER_ADDR	Buffer address for accessing buffer data.
0xC	ADMIN_PCIE_ADDR	PCIe address for accessing PCIe space.
0x10	ADMIN_NSID	NVMe Namespace ID to be used for action commands
0x14	ADMIN_ASQ_INDEX	Admin submission queue indexes. Bits 7:0 => SSD0 Admin Submission Queue Index Bits 23:16 => SSD1 Admin Submission Queue Index
0x18	ADMIN_SCRATCH	Admin scratch register. Read and writing has no effect on NVMe functionality.

3 Programming Guide

3.1 Initialization sequence

This sequence needs to be performed after system power on and anytime after any SSD changes are made to the system. After this sequence is complete the NVME to AXI Bridge is ready to receive data transfer commands from the action interface.

1) Wait until the NVMe host ready bit is set in the ADMIN_STATUS register.
2) Set the enable NVMe host bit in the ADMIN_CONTROL register.
3) Set up the PCIe root complexes using the MMIO interface to write to the PCIe Root Complex Config address spaces.
4) Set up the SSD PCIE registers using the NVME host registers ADMIN_PCIE_ADDR and HOST_PCIE_DATA.
5) Write the admin submission queue entry to the Tx Buffer using the NVME host registers ADMIN_BUFFER_ADDR and HOST_BUFFER_DATA.
6) If the command needs to send data, such as a set features command, write the data to the Tx Buffer using the NVMe Host registers ADMIN_BUFFER_ADDR and HOST_BUFFER_DATA.
7) Write to the COMMAND register to start the command.
8) Poll the ADMIN_STATUS register until command is completed.
9) Read the completion queue entry from the Rx Buffer using the NVME host registers ADMIN_BUFFER_ADDR and HOST_BUFFER_DATA.
10) If the command returns data, such as an identify command, read the data from the Rx Buffer using the NVMe Host registers ADMIN_BUFFER_ADDR and HOST_BUFFER_DATA.

3.2 Action Command Sequence

This sequence is used by the action interface to transfer data to or from the attached SSD's from or to the onboard DDR memory. This sequence can be re-issued in parallel by any of the 16 action kernels and the data transfer will happen for all sequences concurrently. A action kernel can issue several command in sequence and the returned status completion will occur in the sequence in which the commands were issued.

1) Ensure that there is space in the IO submission queue by checking the relevant bits in the STATUS registers in the action address space.
2) Program the DPTR_LOW register in the action address space with the low 32 bits of the DDR memory location of the source or destination data.
3) Program the DPTR_HIGH register in the action address space with the value 0x00000002 + bit 32 of the DDR memory location of the source or destination data.
4) Program the LBA_LOW register in the action address space with the low 32 bits of the SSD LBA being accessed.
5) Program the LBA_HIGH register in the action address space with the high 32 bits of the SSD LBA being accessed.
6) Program the LBA_NUM register in the action address space with the number of LBA blocks that will be transferred.
7) Set the CMD_TYPE in the COMMAND register data to reflect whether the access is a transfer to (write) or from (read) the SSD drive.
8) Set the CMD_QUEUE_ID in the COMMAND register data to reflect which attached SSD is to be accessed.
9) Set the CMD_ACTION_ID in the COMMAND register data to the action kernel ID that is issuing the command. This value will be used to determine which status FIFO will signal the command completion.
10) Write the COMMAND data to the COMMAND register. This will start the command.
11) Wait for command completion. There are two options.
    a. Poll the STATUS register in the action address space until the bit location equal to the action kernel id is '1'. Next read the relevant TRACK_N register to get the completion bit (bit 0) and the error status bit (bit 1) for the command.
    b. Or: Poll the relevant TRACK_N register until a '1' is read from the completion bit (bit 0). The error status will be returned in bit 1.