A Review on Advanced Host Controller Interface (AHCI)

Usha V[1], Dr H V Kumaraswamy[2]

[1]Student, Department of TCE, RV College of Engineering, Karnataka, Bengaluru – 560059, India
[2]*Professor & Associate Dean, Department of TCE, RV College of Engineering, Karnataka, Bengaluru – 560059, India*

Abstract - *Integrated Drive Electronics (IDE) was in use to achieve communication between system memory and storage devices, prior to the invention of Advanced Host Controller Interface (AHCI). Since IDE was compatible with older technologies and device and was taking too long to achieve communication with supporting only few number of devices and many more drawbacks AHCI came into existence which can even support IDE. AHCI also provides newer features such as hot plugging, 64 bit addressing, power management, Native command queuing (NCQ) etc. Since AHCI supports NCQ the Hard Disk Drive's (HDD) speed will increase significantly. If Solid State Devices (SSD) is to be used then IDE mode does not support it and one must switch to AHCI.*

Key Words**: Host Controller Interface (AHCI), Integrated Drive Electronics (IDE), Native command queuing (NCQ), Hard Disk Drive (HDD).**

1. INTRODUCTION

AHCI makes use of the concept of scatter or gather list of IDE and this will be result in decreasing CPU or software overhead and also support advanced features of SATA like power management, hot plug, Native command queuing etc. The Communication between system software and devices goes from task file one byte at a time to a command Frame Information Structure which will be in the memory of system which will be fetched by the Host Bus Adopter (HBA). Resulting reduced command setup time, so that host controller could be connected with numerous devices, and software could not communicate directly through task file to a device.

Communication of data to or from the system memory and device occurs via the HBA, which act as a bus master to system memory. HBA fetches or stores the data to system memory without the intervention of CPU and regardless of method of transfer which may be DMA or PIO, there is no accessible data port.

Each and every transfer is performed using DMA. PIO has very little support for errors, for example, before the data transmission, the ending status field of PIO transfer will be given to the HBA during the PIO Setup FIS. If commands like IDENTIFY DEVICE need to be performed then this could be possible only through PIO.

To implement a Serial ATA command queue using the DMA Setup FIS, AHCI defines a standard mechanism. AHCI HBA is built such that individual slots in Command List will be allocated in system's memory for every command. AHCI device driver written place a command into slot which is empty and after this is reflected in AHCI HBA through register access, the HBA shall fetch the command and transfers it. To get the scatter or gather list used in the transfer, the DMA Setup FIS made use of as a reference into the command list. Multiple commands in the list can still be placed by System software whether PIO, ATAPI or DMA, and the AHCI HBA will look into it and transfers it one by one.

This will be achieved by the HBA by simply moving pointer, which is pointing to command list only when DRQ bit, ERR bit, BSY bits are cleared by the device. Advanced Host Controller Interface (AHCI) host devices support 32 ports as 1 to 32. AHCI is a hardware mechanism supporting the system's software to communicate with SATA storage devices like HDD and SSD. AHCI is a PCI class device. AHCI acts as a data mover to or from SATA devices and system's memory. Both None queued commands and queued commands will no t be mixed in the command list for the same device except the Native Command Queuing (NCQ) unload command. AHCI describes a generic area in system memory for a table of entries describing a command list, status and control.

2. AHCI MEMORY STRUCTURE

To achieve communication between AHCI driver and storage devices system memory descriptors which represent the status of transmitted and received FIS as well as pointers for data transfer and HBA registers are used. HBA memory space consists of AHCI registers and is divided as AHCI registers and system memory as shown in Figure 1. AHCI registers are built in part of AHCI HBA which will usually be integrated inside motherboard of the system.

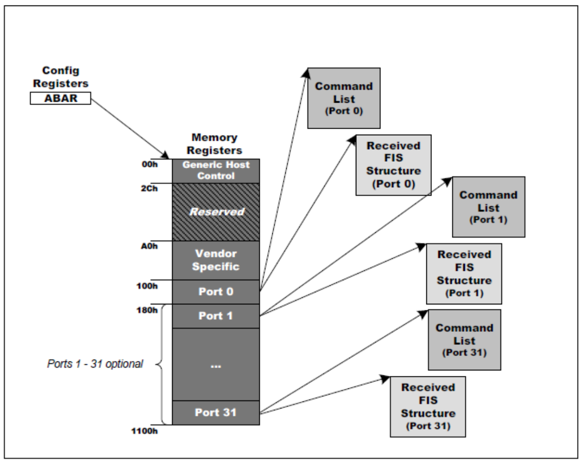

Figure 1: AHCI HBA memory space.

AHCI HBA registers are divided into configuration registers and memory registers. AHCI configuration registers consist of PCI configuration registers and AHCI memory registers consist of Generic Host Control and port registers for corresponding to each 32 ports as depicted in Figure 2. AHCI memory registers are linked by AHCI BAR of AHCI configuration register. Generic Host Control registers indicates all the capabilities. Port control register holds the pointer pointing to command list corresponding to every 32 ports in system memory.

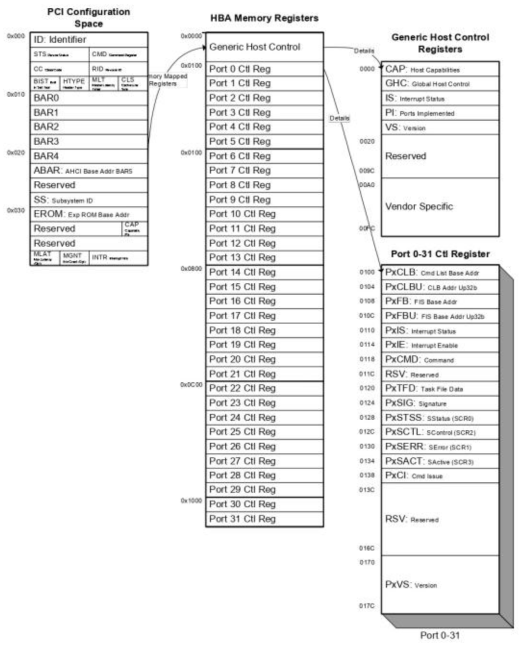

Figure 2: AHCI HBA registers.

HBA Memory space is located in system memory and is reserved for AHCI. Each Port Control Register points to two sections of System Memory that is Base address of corresponding Command List and FIS. Command List consists of a Command Table. Command table contain command which is to be issued and PRD table which contains the scatter or gather list for the data transfer. Different field for every DMA Setup FIS, Set Device Bits FIS, PIO Setup FIS, D2H Register FIS and unknown FIS are reserved in Received FIS Structure as shown in figure 3.

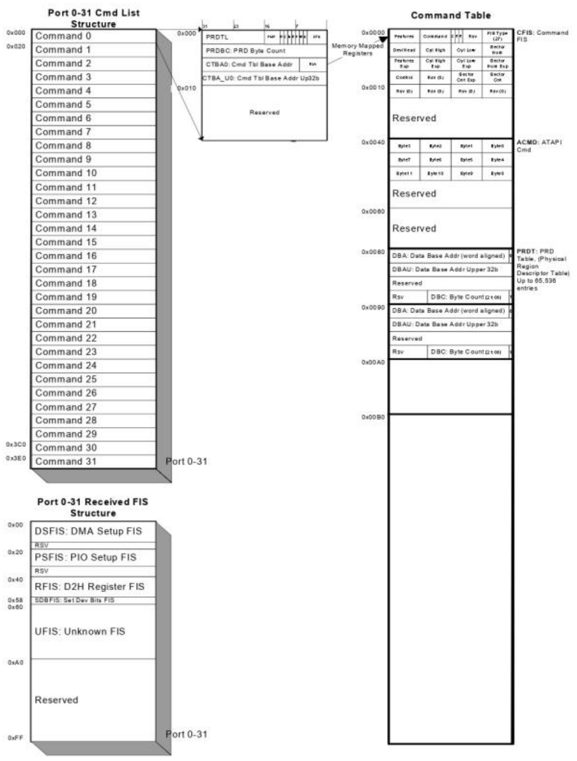

Figure 3: AHCI System memory.

3. DATA TRANSFER IN AHCI

In AHCI, it is assumed that all operations are usually DMA and hence each port has DMA engine. AHCI Driver software is responsible to place the commands in the command list declared this happens when command need to be post and if there is a free slot in command list of corresponding port.

3.1 ATA DMA writes:

Write bit is set to 1 whenever write operation need to happen, and ATAPI bit must not be set since it is ATA command. Read or write operation starts from idle state. First the command is fetched and transmits it, since it is DMA write the received FIS will be placed in DMA setup FIS as in Figure 3. Then the received FIS will be accepted and go to another DMA transfer until the required number of bytes has been written completely.

3.2 ATAPI DMA writes:

Since here ATAPI command is used ATAPI bit will be set to 1. Also write bit will also be set to 1, similar steps will be used that is the command will be fetched and transmitted but it will be in PIO setup FIS. Then the FIS will be accepted by AHCI HBA and ATAPI command will be placed PIO setup now again this will be accepted by AHCI HBA and the data will be transmitted using DMA transmit and will continue to do so until all required number of bytes have been successfully written to device.

3.3 ATA DMA read:

Write bit is set to 0 whenever read operation need to happen, and ATAPI bit must not be set since it is ATA command. Initial steps would be similar to ATA DMA write that the command will be fetched and transmits it this will be accepted by AHCI HBA and this will perform DMA receive operation would take place and will repeat it until the required number of bytes has been read from devices.

3.4 ATAPI DMA read:

Since here ATAPI command is used ATAPI bit will be set to 1 but write bit will also be set to 0, similar steps will be used that is the command will be fetched and transmitted but it will be in PIO setup FIS. Then the FIS will be accepted by AHCI HBA and ATAPI command will be placed PIO setup now again this will be accepted by AHCI HBA and the data will be transmitted using DMA receive operation would take place and will repeat it until the required number of bytes has been read from devices.

4. CONCLUSION

A review on AHCI architecture reveals that use of AHCI mode enhances the speed of accessibility with Hard disk drives and also by little amount in case of Solid state device since AHCI mode supports Native command queuing, power management. Also gives a brief idea of how AHCI memory space is organized and how the DMA read write operation would be performed in both ATA and ATAPI devices.

REFERENCES

[1] D. Cotroneo, L. De Simone, F. Fucci and R. Natella, “MoIO: Run-time monitoring for I/O protocol violations in storage device drivers,” 2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE), Gaithersbury, MD, 2015, pp. 472-483.

[2] https://www.ieee.org/content/dam/ieeeorg/ieee/web/org/about/corporate/ieee-industry-advisoryboard/digital-storage-memory-technology.pdf

[3] https://www.intel.in/content/www/in/en/io/serial-ata/serial-ataahci-spec-rev1-3-1.html

[4] D. C. Turicu, O. Creţ and L. Văcariu, "High performance serial ATA Gen3 controllers on FPGA devices," 2017 International Conference on Field Programmable Technology (ICFPT), Melbourne, VIC, 2017, pp. 32-39.

[5] H. M. Ashour, "Challenges in serial protocols Verification on an emulation environment (SATA as an example)," 2016 11th International Design & Test Symposium (IDT), Hammamet, 2016, pp. 93-97.