ATA Driver Report

Name: Ashwin Chidambaram, SID:
Ujjwal Sai, SID:
Haoxuan Zhang, SID: 3173595

EECS 446 Spring 2020
Final Report

1. Section 1

Text Example
A new line!

A new line!

1.1 subsection 1

Test Example
A new line!

A new line!
indent 1 cm

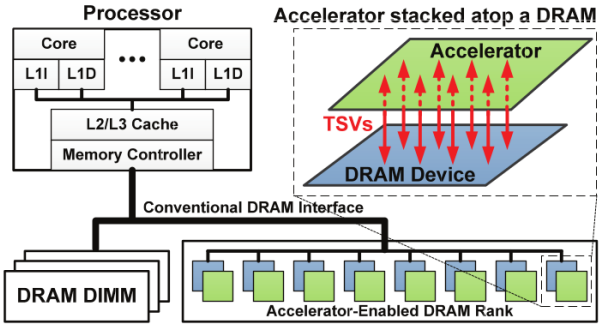

Figure 1: This is caption

Reference Firgure 1
Reference 1: [1]

2. Background

3. ATA DMA

4. ATA Interrupt

5. PCI support

Our previous work for DMA and interrupt handling are only supporting legacy mode in which at most 4 drives could be detected at a well known location. The code of DMA and interrupt handling driver only supporting legacy mode is in the branch of `ata_interrupts` in our Github repository. In our another branch `ATA_PCI`, we tried to impliment PCI support with DMA and polling and it works correctly with legacy device testing. Because we didn't find a way to attach arbitrary number of IDE controllers to PCI bus, the driver is not tested with ata device located not in legacy location. This section, we will interpret the PCI mechnism and our work on PCI support for the ata driver.

5.1 PCI Configuration Space

Figure 2 shows the content of PCI configuration space.

register	offset	bits 31-24	bits 23-16	bits 15-8	bits 7-0
00	00	Device ID		Vendor ID	
01	04	Status		Command	
02	08	Class code	Subclass	Prog IF	Revision ID
03	0C	BIST	Header type	Latency Timer	Cache Line Size
04	10	Base address #0 (BAR0)			
05	14	Base address #1 (BAR1)			
06	18	Base address #2 (BAR2)			
07	1C	Base address #3 (BAR3)			
08	20	Base address #4 (BAR4)			
09	24	Base address #5 (BAR5)			
0A	28	Cardbus CIS Pointer			
0B	2C	Subsystem ID		Subsystem Vendor ID	
0C	30	Expansion ROM base address			
0D	34	Reserved			Capabilities Pointer
0E	38	Reserved			
0F	3C	Max latency	Min Grant	Interrupt PIN	Interrupt Line

Figure 2: PCI Configuration Space[4]

The PCI configuration space has the I/O port mapped address at a well known places. It contains basic information of the device and the base address register which will be used to configure the base address of the device port registers.

According to OSdev PCI page[4], the Class Code shown in Figure 2 contains class code, sub class code and program interface information which we would use to identify whether the device is

an IDE device or not. The next subsection will illustrate the PCI native mode and how we obtain the base address of the device registers.

5.2 PCI native mode

Each ATA channel will have 8 I/O registers and 3 Control registers. The first step to utilize these registers is to find the location of the registers. Table 1 shows the I/O port register address offset relative to the base address and the information is referenced from OSdev ATA PIO page[3].

Offset from I/O base(Direction)	Register
0	Data register
1(Read)	Error Register
1(Write)	Feature Register
2	Sector Count Register
3	LAB low
4	LBA mid
5	LBA high
6	Head register
7 (Read)	Status Register
7 (Write)	Command Register

Table 1: ATA I/O registers

ATA device also has 3 control register including Alternate Status Register, Device Control Register and Drive Address Register. We didn't use these registers in our driver except reading Alternate Status Register for delay.

Unlike the legacy mode where all the registers will be located at fixed and well known places, base addresses will be determined by Base Address Register. According to IDE controller programmer mannel[2], in primary channel, The I/O port base will be determined by BAR0, and the control port base will be determined by BAR1. In secondary channel, I/O port base will be determined by BAR2 while the control port base will be determined by BAR3.

To support DMA, we also need to get control of the bus master. The base address of the bus master is the value in BAR4 or BAR4 + 8 for primary channel or secondary channel. After obtaining the base address, we could use the address to access the I/O port mapped registers in the IDE controller including the ata drive registers and the bus master registers.

5.3 Process of recognizing Device and Our Work on PCI To recognize the device, we would:

a) Loop through PCI bus and devices to find devices having class code and subclass code to be 0x1 which means IDE device

b) Allocate space for the IDE controller which contains 4 drive status

c) Initialize members in drive status structure according to the channel, device ID and Base Address Register.

d) Write to PCI configuration space command with 0x04 to enable Bus master

e) Device detection and identification for each drive in the controller in the same way as the previous APIO ata driver.

After detection and initializing the address information in the status structure, we would be able to utilize the information collected by the status structure to conduct read and write.

Our work on PCI is in the ATA_PCI branch ata_pci.c and the description of our works is as follows:

a) We could scan the PCI devices and register the device with name of ata(bus number)-(device number)-(channel)-(id)

b) We initialize register addresses according to the base address registers.

c) Because legacy mode ata devices are still attached to PCI bus, we tested our PCI support with legacy location and the testing result is still correct.

d) We havent confirmed a way to add IDE controller attached with drives to the PCI bus so that we could test the driver with arbitrary device addresses.

6. Result

References

[1] Reference Here

[2] Intel IDE Controller Programmers Reference Manual, November 2001, Revision 1.0

[3] OSDev, ATA PIO Mode, `https://wiki.osdev.org/ATA_PIO_Mode`, Retrieved on June 10

[4] OSDev, PCI `https://wiki.osdev.org/PCI`, Retrieved on June 10