Guest OS Backward Compatibilty for FreeBSD Hypervisor

Teaca Ionut - Alexandru, Mihai Carabas
Automatic Control and Computers Faculty
Politehnica University of Bucharest
Emails: ionut.teaca@cti.pub.ro, mihai.carabas@cs.pub.ro

***Index Terms*—FreeBSD, bhyve, hypervisor, device emulation, LPC, ATAPI, CDROM**

I. Introduction

This report presents the main features implemented such as the Redesign, LPC attachment and the emulation of ATAPI devices. We begin with the details of design and configuration and continue with the results showing some driver logs. At last we present the next steps in the development and testing for the ATA/ATAPI emulation.

II. Implementation

A. Redesign

1) ATA Drive: The initial implementation was designed to work with the channel being the data structure that holds the ATA registers set. In this way, the ATA commands are emulated at channel level, the device being selected by the DEV bit from the Device register. With this approach everything worked quite well except when the ATA controller was configured with two different devices such as an ATA disk drive and an ATAPI CDROM. The problem we had to cope was that after a Software Reset command both devices shall reset and put their signature in the ATA registers. As long as there are two identical devices it is not any problem since both devices have the same signature. Obviously the solution is to have different sets of registers for each ATA device. Moreover, we figured out that even the PIO mechanism and the state of the interrupts (enabled / disabled) shall be per drive instead of channel. For this reason we redesigned the channel to take two references to the ATA drives, and each drive to hold that information.

2) PIO: The PIO protocol is a mechanism used to transfer data to or from the ATA device by writing or reading the Data register. The only commands for an ATA disk drive that use PIO are the READ and WRITE commands aimed to get or put data from the disk. Because of this, we have not designed a complicated mechanism, but just handled the transfers when the Data register was read or written. Once that we introduce the ATAPI emulation, we realized there are more commands that use the PIO mechanism to transfer data. Actually, all the ATAPI commands are performed using PIO and even the packet itself is a PIO transfer. So we had to cope with different PIO transfers of different lengths and different logic. Because of this, we designed a general, flexible and easy to use mechanism to create PIO transfers. In order to set up either a read or write PIO transfer, the ata_pio_do_transfer function is called and it is provided the length of the transfer and the callback which is going to be called when the transfer ends. This callback is supposed to implement the logic of any PIO command. Take care that each PIO setup can handle one single transfer in progress, but this is not a problem since the commands for a device are never issued in parallel.

3) Interrupts: There are two main types of interrupts that a device can use to interrupt the CPU: the Level-triggered and Edge-triggered interrupts. The description of these types of interrupts is not the goal of this work, but the relation with the ATA controller shall be presented. The PCI bridge uses the level-triggered interrupts while the LPC bridge uses the edge-triggered interrupts. So, the ATA emulation has to raise both interrupts while working attached under the PCI bus or LPC bus. The interfaces provided by the bhyve library to raise these interrupts are quite different so we need to implement a general mechanism to assert interrupts efficiently no matter under which attachment the ATA runs. The solution is to register different callbacks depending on what attachment is used which will be called whenever the ATA controller wishes to interrupt. This way we do not need to check each time if we run under PCI or LPC so the solution is transparent and efficient.

4) PCI attachment: One difference against the previous work related with the PCI attachment is that from now on we support both ATA channels emulated in the same time. We didn't manage to do it in the last version because of a bug located in the PCI initialization phase solved in the current version. Same as before, each channel can be configured to support one or two drives either ATA or ATAPI drives. In order to configure two ATA channels running under the PCI attachment the configuration string is: *-s 4:0,ata-hd,"0,./diskdev_ata0;1,./diskdev_ata1"*. Practically the semicolon character is used to separate the channels. The parameters that describe one single channel are the same as before: *"X, dev_master, dev_slave"*.

B. LPC attachment

In the last work the ATA emulation was working under the PCI attachment. However, a number of guest operating systems do not have drivers for the PCI attachment and are not able to use this emulation. One of the best supported devices,

especially in older operating systems, is the ATA LPC device. An implementation of a device model for this would allow a larger number of unmodified guest operating systems to run under bhyve.

1) LPC configuration: The LPC bus has two IDE interfaces (primary and secondary), also known as channels. Like the PCI attachment, each channel can support two devices, hance up to 4 ATA/ATAPI devices can be configured. The configuration parameters for the LPC attachment are:
-l ata-hd,X,./DISK_MASTER,./DISK_SLAVE
OR
-l ata-hd,X,./DISK_MASTER
where X is either 0 or 1 depending on what channel is used and is followed by the name of the devices, the first one being the master drive, and the second as the slave drive.

Unlike the PCI attachment where the driver probes the ATA channel on the fly using the PCI bus enumeration, when working under the LPC attachment some hints must be provided so the guest drivers find out the addresses of the IO ports and the IRQ numbers for each ATA channel. Hence the following lines shall be added in the */boot/device.hints* configuration file (see Listing 1):

Listing 1. Boot Device Hints

```
hint.ata.0.at="isa"
hint.ata.0.port="0x1F0"
hint.ata.0.irq="14"
hint.ata.1.at="isa"
hint.ata.1.port="0x170"
hint.ata.1.irq="15"
```

2) Differences between PCI and LPC: Even though the ATA behaviour is the same, and the commands are emulated in the same way no matter what attachment is used, there are some differences between the PCI and LPC, mostly in the initialization phase.

First of all, the PCI bridge uses the level-triggered interrupts while the LPC bridge uses the edge-triggered interrupts so we register different interrupt callbacks that call the "vm_isa_pulse_irq" function provided by the bhyve interface in order to raise interrupts.

Unlike the PCI attachment that uses the BAR registers to address the ATA internals registers, the LPC attachment uses two IO ports for each channel. Each IO port has its address at a known address which is specified in the the */boot/device.hints*. The first IO port is used to address eight io port registers while the second IO port is used to address the alternate register.

Even though the PIO protocol used to transfer data works pretty much the same, the only difference being the LPC attachment uses words of 16 bits to write or read the DATA register unlike the PCI which uses words of 32 bits. This could cause a worse performance for the ATA controller working in the LPC attachment because there are two times more accesses to the DATA register so the overhead is doubled.

Maybe the most important difference is that there is no DMA channel for the ATA controller while running under the LPC attachment. Hence, all the transfer commands use the PIO protocol to read or write data. When running under the PCI attachment, the DMA channel is provided by the PCI adapter.

C. ATAPI Emulation

At the moment, the guest operating system installation boots from a virtio CDROM. Even though the guest is installed on a ATA drive, we need to get rid of the virtio emulation. The solution is to emulate the CDROM as an ATAPI device which is one of the best supported devices, especially in older operating systems. ATAPI stands from ATA Packet Interface which means the ATAPI devices get commands from the host using packets transmitted through ATA commands. The ATAPI commands are represented by the SCSI set, and our goal is to emulate only the subset required by the CDROM emulation.

1) ATAPI configuration: The ATAPI CDROM devices are configured like the ATA drives except that the name of the image file shall have the iso extension. Otherwise the device is considered a regular ATA drive. For example, in order to add an ATAPI CDROM on the ATA channel 0 with the media from release.iso these are the configuration parameters: *-l ata-hd,0,./release.iso*.

2) ATA commands: Even though our goal is to implement the General feature set commands supported by the ATA 6 standard, in order to support ATAPI devices, we had to implement two more ATA commands: ATA_ATAPI_IDENTIFY and ATA_PACKET_CMD.

ATA_ATAPI_IDENTIFY: this ATA command is used by the host in order to get information about the ATAPI device. The host finds out there is an ATAPI device after the Software Reset when each device changes the registers with its signature. If there is an ATAPI device, the host will call the ATA_ATAPI_IDENTIFY to get extra information like the model and serial number and other capabilities.

ATA_PACKET_CMD: this command is issued by the host in order to send a packet command to the ATAPI device. After this command, the host transmits the packet of 12 bytes of data using the PIO protocol to the device. The first byte from the packet represents the op code and it is used to select the ATAPI command.

3) ATAPI commands: Our goal is to implement only a subset of the SCSI commands required by the CDROM device. Each ATAPI command is sent in a packet command of 12 bytes using the PIO protocol. So far, we managed to implement the following commands:

INQUIRY: using this command, the host asks some information such as the vendor, product and revision from the ATAPI device. The device replies the data to the host through a PIO transfer of 36 bytes.

READ_CAPACITY: the host reads the capacity of the media CDROM. The device replies the number of blocks and the block size of its media through a PIO transfer of 8 bytes.

READ_TOC: the host requests the Drive to read data from a Table of Contents and transfer the result back to the Host. Our ATAPI module emulates the media having one single track so the response to this command is composed of Track1 in the data zone.

READ_10: this command is issued by the host in order to read data from the media using the PIO protocol. The command specifies the LBA address of the starting block and the number of blocks to be read. For each 2048 bytes representing the size of the block the ATAPI drive interrupts the host.

PREVENT_ALLOW and TEST_UNIT_READY: we don't do anything special but these commands have been implemented only because the driver issues these commands and the drive must acknowledge them by raising an interrupt.

III. Scenarios and Results

The Listing 2 shows both ATA channels attached under the *isa0* driver. Notice the irq numbers and IO port addresses are the ones specified in the */boot/device.hints* configuration file.

Listing 2. LPC ATA channels

```
ata0: <ATA channel> at port
ata0: 0x1f0-0x1f7,0x3f6 irq 14 on isa0
ata1: <ATA channel> at port
ata1: 0x170-0x177,0x376 irq 15 on isa0
```

In the Listing 3 there is an ATA disk drive inserted in the *ata1* channel which is attached under the LPC bus. You can see there is no difference if the ATA channel is attached under the PCI or LPC bus when the ada driver probes the ATA drive.

Listing 3. ATA Disk Drive under LPC

```
ada1 at ata1 bus 0 scbus2 target 0 lun 0
ada1: <BHYVE ATA IDE DISK 1.0> ATA-6 device
ada1: Serial Number 123456
ada1: 16.700MB/s transfers (PIO4, PIO 65536bytes)
ada1: 8192MB
ada1: (16777216 512 byte sectors: 16H 63S/T 16644C)
ada1: Previously was known as ad2
```

In the Listing 4 there is an ATAPI CDROM inserted in the *ata1* channel which is attached under the LPC bus. The *cd* driver has probed the CDROM device and received some information about the device using the ATAPI commands.

Listing 4. ATAPI CDROM under LPC

```
cd0 at ata1 bus 0 scbus2 target 0 lun 0
cd0: <BHYVE ATAPI IDE CDROM 1.1>
cd0: Removable CD-ROM SCSI-0 device
cd0: Serial Number 123456
cd0: 16.700MB/s
cd0: transfers (PIO4, ATAPI 12bytes, PIO 65534bytes)
cd0: cd present [350001 x 2048 byte records]
```

IV. Conclusion and Further Work

This report presents the relevant features implemented such the LPC attachment and the ATAPI emulation but there is still to develop. The next step is to understand how the data is structured and organized on the CDROM media followed by the implementation of the READ commands from the media. The last step is to test the whole implementation again together with the new features implemented to make sure there are no bugs introduced by the new changes. The final scenario of testing is to install the guest operation system from an ATAPI CDROM image to the ATA disk drive.