Notes and Items Planned for the Next Edition

This document is a list of subjects that I plan to add to a future edition of the book, *USB: The Universal Serial Bus*, found at http://www.fysnet.net/the_universal_serial_bus.htm

These notes are added so that you may benefit from the information now instead of waiting for the next edition. I will add information as time and interest allows.

Thank you for your support,
Ben

EHCI and Rate-Matching Hubs

Since the ECHI is a high-speed only controller, manufacturers had to include at least one full/low speed controller embedded on the chip. This used more resources as well as requiring the host to support three controller types, the EHCI and then one of the earlier controller types, either UHCI or OHCI.

On newer hardware, Rate-Matching Hubs (RMH) were added. Rather than including one or more full/low speed controllers with the EHCI controller, a standard USB 2.0 Discrete USB hub is included and connected to Port 0 of the EHCI controller. The hub now does all the work of changing the low- and full-speed traffic to high-speed traffic and visa-versa.

When the RMH is enabled, it will appear to the host as an external hub that is connected to Port 0 of the controller. Host software no longer needs to support the UHCI or OHCI interfaces as long as it supports external hubs.

When the RMH is enabled, the EHCI will report having only two ports, the first used for the RMH and the second as the debug port which is muxed (connected) to the second port of the RMH.

If an EHCI controller card is attached and it supports the RMH, the controller will only handle high-speed devices and may or may not have more than two ports. When RMH is disabled, the controller may have as many as 15 ports, though when RMH is enabled, it will have 1, 2, or 3. The first will always be attached to the RMH, the second, if present, will be the debug port, and the third, if present, will be the KVM/USB-R port.

Enabling the RMH is not USB defined, it is controller defined and the manufacture of the card will specify how it is enabled. For an example, the Intel® 5 and 3400 series chipsets use the MISCCTL - Miscellaneous Control Register to enable the RMH. See section 10.1.67 of that chipset's specification.

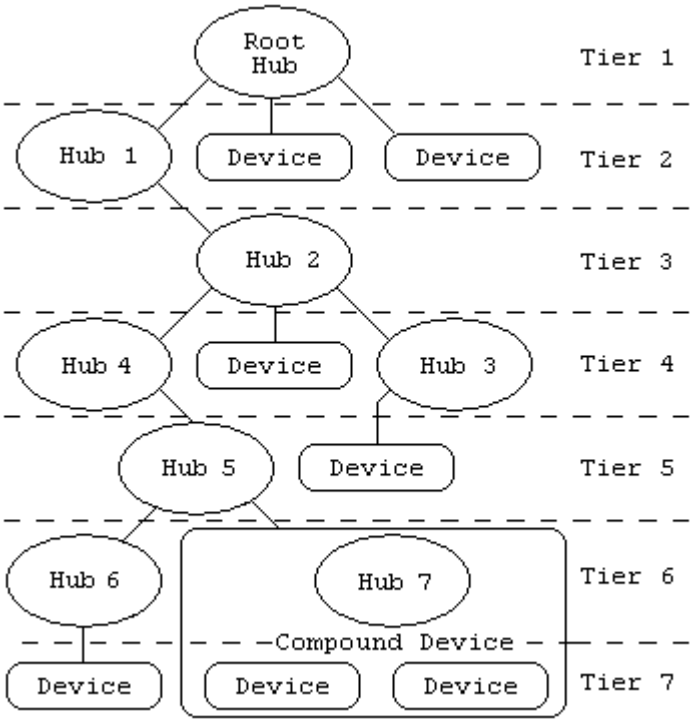
Maximum Hub and Tier

Due to the timing restrains of the USB, a maximum of seven tiers (levels) is allowed per Root Hub port. In other words, you may have no more than a depth of five external hubs attached to any given Root Hub port with a device attached to the fifth hub as the seventh tier.

You may have more than five external hubs attached, so long as you do not have more than five consecutively connected.

The figure below shows an example of a seven-tier set up.

Figure 19-2: Seven Tiers



See Section 4.1.1 in the USB 2.0 Specification for more information.

OHCI's Short Packets and the Toggle bit

You can either place the toggle bit in the Endpoint Descriptor and have the controller update it for you, or you can place the toggle bit in the first TD and then each consecutive TD must advance this toggle bit.

Usually, for Control endpoints, you place the toggle in the TD's since the first one is always clear and the last one, the Status packet, is always set. However, for bulk and interrupt endpoints, you may want to place the toggle in the Endpoint Descriptor and let the controller keep track of it.

However, what happens when a short packet it detected?

First, let's focus on what the controller does when a short packet is detected.

When a short packet is detected, meaning that on a packet where you requested more than the device sent, yet it was a successful transfer, the controller will do one of two things depending on the bufferRounding bit in the TD of the short packet.

If this bit is set, the controller will mark the TD as a successful transfer, however, the controller will continue on to the next TD pointed to by the NextTD field. Continuing to the next TD may not be what you planned on.

If this bit is clear, the controller will halt the Endpoint Descriptor by setting the Halted bit, then move on to the next Endpoint Descriptor, but it will mark the TD as a Data Underrun error occurred.

If a successful transfer of all bytes requested occur, the controller will mark the TD as NoError. With this in mind, it is suggested that you clear the bufferRounding bit and watch for the Underrun error as well.

How does this all effect the Toggle bit? For an example, during a Bulk transfer, you might request a 128-byte Sense Media request. If the device has a 64-byte max packet size, you have created two TD's for the endpoint. The first TD might have a toggle bit of 0, while the second TD having a toggle bit of 1. With this in mind, the next time you send a TD to this endpoint, you would use a toggle bit of 0 again.

However, what if the device only returned a 40-byte response? Assuming you have the bufferRounding bit clear, the second TD was never executed, and the device will be expecting a toggle bit of 1 on the next packet.

Therefore, for Bulk and Interrupt transfers, it is suggested that you use the toggleCarry bit in the Endpoint Descriptor for all transfers with that endpoint. After the controller retires the TD, it will update the toggleCarry bit in the Endpoint Descriptor with the value of the next packets toggle value. If you use the same Endpoint Descriptor for all transfers for

this endpoint, assuming no stalls, you don't have to worry about the toggle bit, let the controller do it for you.

Remember that after clearing the stall, the device will be expecting a toggle bit of zero for the next requested transfer for the endpoint.

Resetting UHCI Controllers

In my tests, I have found that not all UHCI controllers follow the reset requirements specified within the USB specification. For example, my early reset code followed the USB specification and did the following:

- Set reset bit
- Delay for *USB_TDRSTR* milliseconds
- Clear reset and Connect Status Change bit
- Delay for *USB_TRSTRCY* milliseconds
- Set Enable bit
- Wait for Enable bit to become set
- Clear Enable/Disable Change bit

Even though all low-speed devices successfully reset and then enumerated with this code, some full-speed, including higher speed devices dropping down to full-speed, would not reset successfully, but would return a Stall and/or Time-out error at the first time it saw a packet from the Host.

Come to find out, there are a few issues with some UHCI controllers, namely the VIA UHCI Host Controller with a model number of VT83C572 or VT82C586 as mentioned before. This controller, as well as a few others, must not be in reset when you clear the Connect Status Change (CSC) bit, as well as this CSC bit must be clear, before you try to set the Enable bit.

As for the UHCI port register, if you did not follow these specific rules, the port might show enabled, but the enable was not successful and the attached full-speed device would not enumerate.

Also, some devices needed a 50µs delay after setting the Enable bit and before clearing the Enable/Disable Change bit, as well as a 50ms delay after clearing this bit and before you start to enumerate the device.

After much research and trial and error, I have come up with the following technique that seems to work for all devices I have tried it with, both low- and full-speed as well as higher speed devices dropping down to full-speed.

- Set reset bit
- Delay for *USB_TDRSTR* milliseconds
- Clear reset bit writing a zero to the Connect Status Change and Enable bits
- Delay for 250 microseconds (see note)
- Set Enable bit and clear Connect Status Change bit at the same time
- Delay for 50 microseconds
- Clear Enable/Disable Change bit
- Delay for 50 milliseconds before sending any packets

Please note that the Reset Recovery Time (USB_TRSTRCY) is not used. I only allow 250µs for this time instead of the specification required 10ms of delay. Some devices did not like to longer delay between the time of the reset and the CSC bit and Enable bit transitions.

As stated throughout this book, not all devices and controllers follow the specification exactly and have their own quirks that you must deal with. It is trial and error to see which technique will work with most, and hopefully all, combination of controllers and devices.