XinUSB: A USB Driver for Xinu

Anna Whitley under mentorship of Dr. Dennis Brylow, MSCS, REU Summer 2011

Problem

This project is part an ongoing effort to create a USB host controller driver for the Xinu OS to run on the Linksys E2100L routers. The difficulty has been that information on how to write a host controller driver is incredibly scarce. Most resources assume thorough knowledge of the USB System, are not platform-specific, are questionably-documented code, or are for writing device drivers, not system drivers.

Relevancy

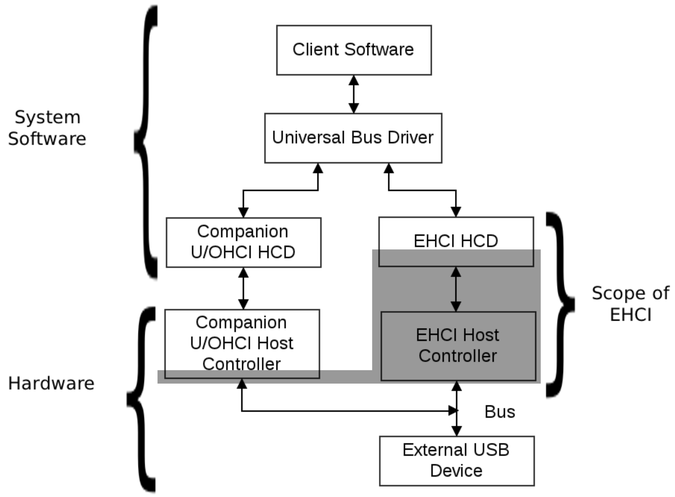
USB devices have become ubiquitous in the world of computing. Being able to use the USB ports on these routers would open up the possibility of using many new devices with Xinu, e.g.:

- Mice
- External flash drives
- Kinect
- Lights
- USB-to-serial, giving them a second serial port

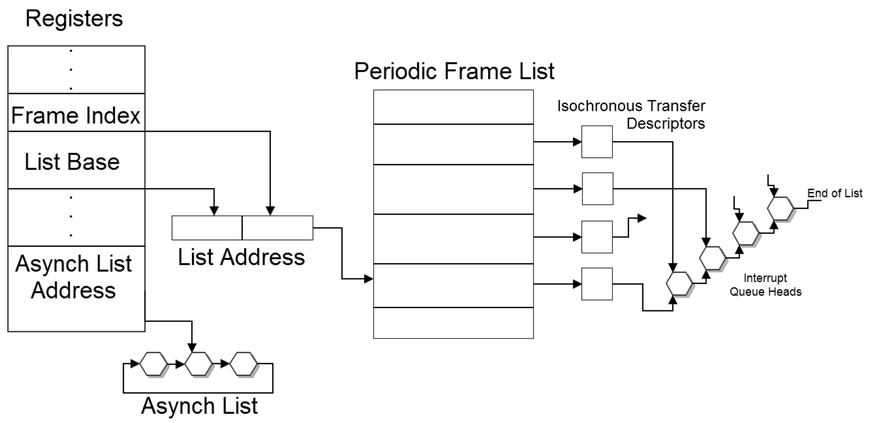
Useful Acronyms

- HCD – Host Controller Driver, the software that controls the host controller (the hardware).
- EHCI – Extended Host Controller Interface, interface that supports the USB 2.0 protocol.
- OHCI – Open Host Controller Interface, interface that supports the USB 1.1 protocol.

The USB System



The USB HCD Protocol



References

[1] Xinu: http://xinu.mscs.mu.edu
[2] Extended Host Controller Interface Specification for Universal Serial Bus revision 1.0
[3] Universal Serial Bus Specification Revision 2.0
[4] OpenWRT: https://openwrt.org/

Progress

- Got OpenWRT flashed to a router with help of Kyle Persohn.
- Found OpenWRT's USB initialization code after weeks spent sorting through millions of lines of code.
- Confirmed location of host controller's registers.
- Confirmed IRQ channel 3 for USB.
- Found EHCI and USB specification documentation which confirmed location and described desired behavior of USB registers.

```
Welcome to the wonderful world of Xinu!
xsh$ test
identRegs
val @ loc: 0 @ 0xBB000000
val @ loc: 0 @ 0xBB000004
val @ loc: 0 @ 0xBB000008
val @ loc: 0 @ 0xBB00000C
val @ loc: 0 @ 0xBB000010
val @ loc: 0 @ 0xBB000014
val @ loc: 0 @ 0xBB000100
val @ loc: 0 @ 0xBB000104
val @ loc: 0 @ 0xBB000108
val @ loc: 0 @ 0xBB000120
val @ loc: 0 @ 0xBB000124
val @ loc: 0 @ 0xBB000134
val @ loc: 0 @ 0xBB000140
val @ loc: 0 @ 0xBB000144
val @ loc: 0 @ 0xBB000160
val @ loc: 0 @ 0xBB000180
val @ loc: 0 @ 0xBB000184
val @ loc: 0 @ 0xBB0001A4
val @ loc: 0 @ 0xBB0001C0
ctrlRegs
```

```
Welcome to the wonderful world of Xinu!
xsh$ test
identRegs
val @ loc: 42FA05 @ 0xBB000000
val @ loc: 22 @ 0xBB000004
val @ loc: 10020001 @ 0xBB000008
val @ loc: D @ 0xBB00000C
val @ loc: 80060908 @ 0xBB000010
val @ loc: 608 @ 0xBB000014
val @ loc: 1000040 @ 0xBB000100
val @ loc: 10011 @ 0xBB000104
val @ loc: 6 @ 0xBB000108
val @ loc: 1 @ 0xBB000120
val @ loc: 186 @ 0xBB000124
val @ loc: 3000 @ 0xBB000134
val @ loc: 80000 @ 0xBB000140
val @ loc: 80 @ 0xBB000144
val @ loc: 808 @ 0xBB000160
val @ loc: 1 @ 0xBB000180
val @ loc: C000004 @ 0xBB000184
val @ loc: 202F20 @ 0xBB0001A4
val @ loc: 800080 @ 0xBB0001C0
ctrlRegs
val @ loc: 32 @ 0xB8030004

[registers which correctly initialize to 0
and unrelated registers omitted for brevity]
```

Above Left: Register values before discovering initialization.
Above Right: Register values after discovering and implementing initialization.
Left: Visual representation of the EHCI data structures that keep track of data transfer scheduling.

Future Work

After completion of the first 3 steps, the USB driver will have core functionality:

- Figure out and correct the status register's incorrect behavior.
- Point registers to already-created data structures (see diagram to the left).
- Once EHCI is working, explore and implement Universal Bus Driver.
- Once that is working, explore OHCI.