

1394 Kernel Debugging Tips And Tricks

Tom Green
Software Development Lead
Windows Device Experience Group

Session Outline

- What is 1394?
- Why debug with 1394?
- Configuring for debugging with 1394
- How does it work?
- 1394 debugger revisions
- Known issues with 1394 debugging
- Going forward
- Kernel Debugging Over USB 2.0

What Is 1394?

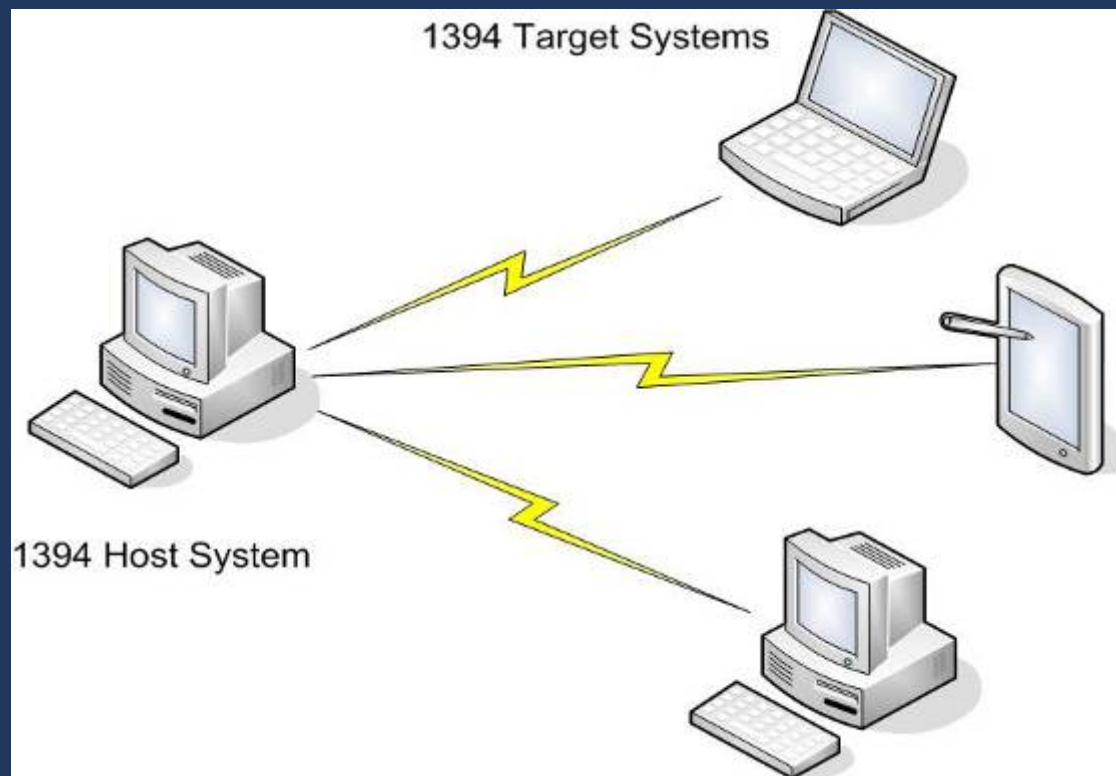
- High-speed serial bus – up to 400Mbps
- Plug-n-Play
- Daisy-chain up to 63 devices (nodes)
- Available as a PCI or PC Card Adapter for systems without 1394 built in
- Two types of 1394 cable connectors
 - 4-pin and 6-pin connector
 - 4-pin to 6-pin and vice-versa works

Why Debug With 1394?

- Off-the-shelf technology
 - Any 1394 controller supporting OHCI 1.0 works
- Legacy Free
 - Currently the only legacy free debug solution
- Easily extensible for multiple target systems
 - As easy as plugging in another 1394 cable
 - Theoretical maximum of 62 target systems
- Minimal code running on target system
 - Physical memory of target system is exposed on 1394
 - Host client handles all asynchronous data transfers to target system
- It's fast
 - Full system memory dump (128MB):
 - Serial takes over 3 hours (baud rate set to 115200bps)
 - 1394 takes approximately 15 seconds

Hardware Configuration

- Example 1394 Debug Topology



Target Software Configuration

- Standard installation for X86 systems:
 - Open a CMD window
 - `attrib -s -h -r c:\boot.ini`
 - `notepad c:\boot.ini`
 - Add the information shown in blue:

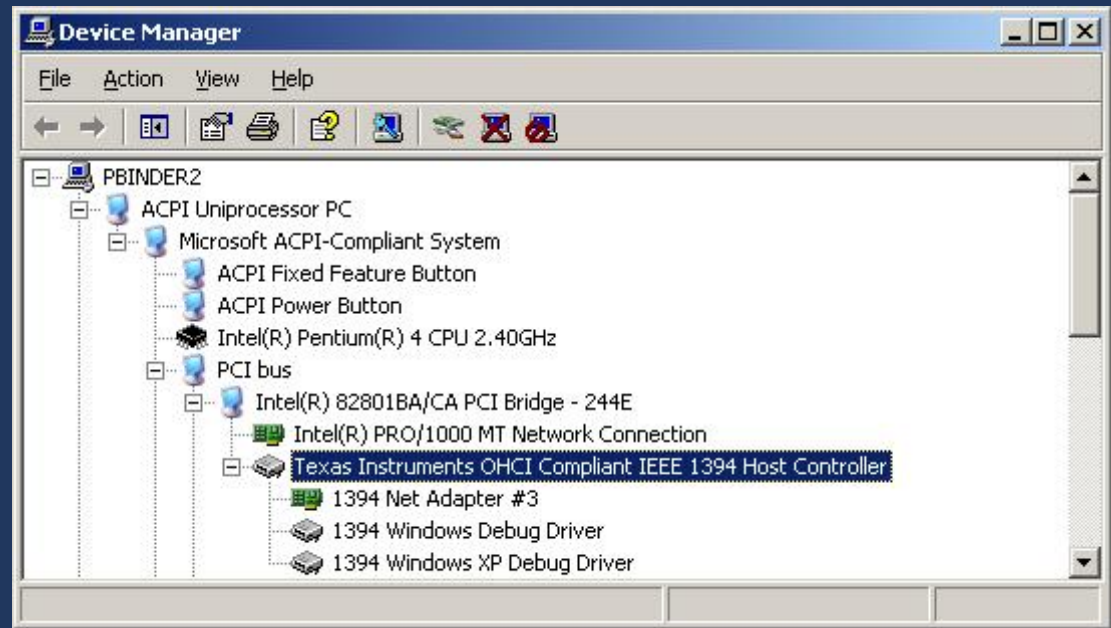
```
[boot loader]
timeout=30
default=multi(0)disk(0)rdisk(0)partition(1)\WINNT
[Operating Systems]
multi(0)disk(0)rdisk(0)partition(1)\WINNT="Windows"
/fastdetect /debug /debugport=1394 /channel=10
```
- Reboot the machine
- For configuring other platforms, refer to the debug documentation

Target Software Configuration

- Additional Tips:

- Disable the 1394 controller from device manager

- Upside: Greatly improves reliability of 1394 debug
- Downside: Loss of some power management functionality
- Make sure it's the 1394 host controller and not the 1394 net adapter:



Host Software Configuration

- Install the Debugging Tools for Windows
- The first time you try to start the debugger, two 1394 virtual drivers will be installed on your system that allow the debugger to communicate with the target systems
 - 1394DBG1.SYS – Windows XP (build 2600) only
 - 1394DBG2.SYS – Windows XP SP1 and future revisions of Windows
- From Device Manager:



- Administrator privileges are required for driver installation

Host Software Configuration With KD

- Open a CMD window and from the debug directory type the following:
 - set _NT_DEBUG_BUS=1394
 - set _NT_DEBUG_1394_CHANNEL=10 ; channel number
 - Select a unique channel number, between 0 and 62, that will be used to identify the target system
 - kd.exe -v
- First time launching kd causes the debug drivers to load. The following message is displayed:



```
Command Prompt
C:\debuggers>set _NT_DEBUG_BUS=1394
C:\debuggers>set _NT_DEBUG_1394_CHANNEL=10
C:\debuggers>kd -v

Microsoft (R) Windows Debugger Version 6.3.0003.3
Copyright (c) Microsoft Corporation. All rights reserved.

Using 1394 for debugging
Failed to open 1394 channel 10
If this is the first time KD was run, this is why this failed.
Virtual 1394 Debugger Driver Installation will now be attempted
Kernel debugger failed initialization, Win32 error 2
"The system cannot find the file specified."
Debuggee initialization failed, Win32 error 2
"The system cannot find the file specified."

C:\debuggers>_
```

Host Software Configuration With KD

- Re-launch `kd.exe -v` and boot target system
- Connection is established:



```
C:\debuggers>kd -v -b

Microsoft (R) Windows Debugger Version 6.3.0003.3
Copyright (c) Microsoft Corporation. All rights reserved.

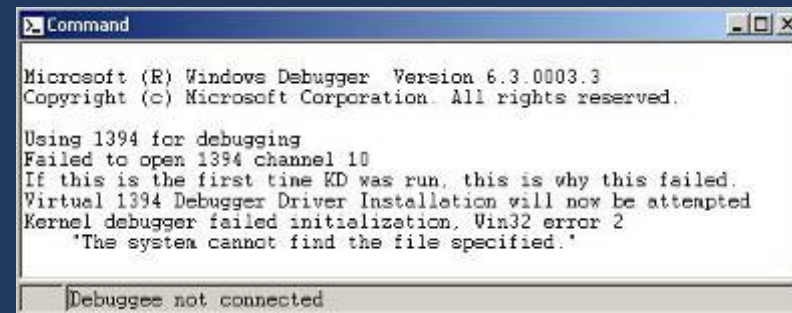
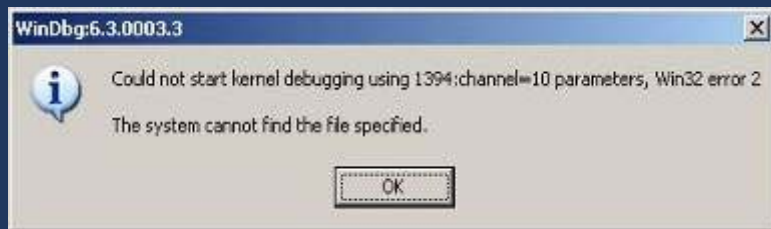
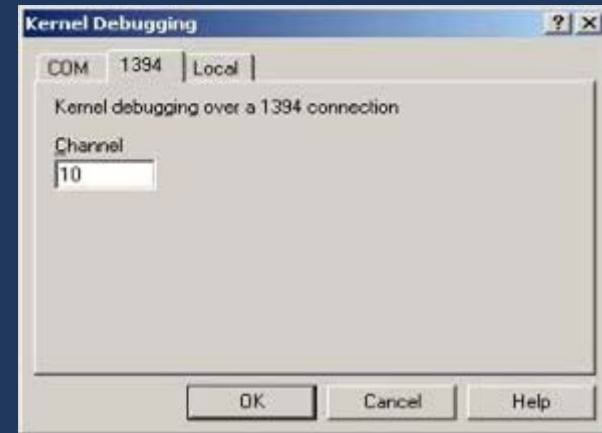
Using 1394 for debugging
Opened \\.\DBG1394_INSTANCE10
Waiting to reconnect...
Connected to Windows XP 2600 x86 compatible target, ptr64 FALSE
Kernel Debugger connection established. (Initial Breakpoint requested)
Symbol search path is: *** Invalid ***
*****
* Symbol loading may be unreliable without a symbol search path.          *
* Use .symfix to have the debugger choose a symbol path.                  *
* After setting your symbol path, use .reload to refresh symbol locations. *
*****
Executable search path is:

<Text Removed>

krnl.exe -
nt!DbgBreakPointWithStatus+0x4:
8051ac9c cc          int     3
kd> _
```

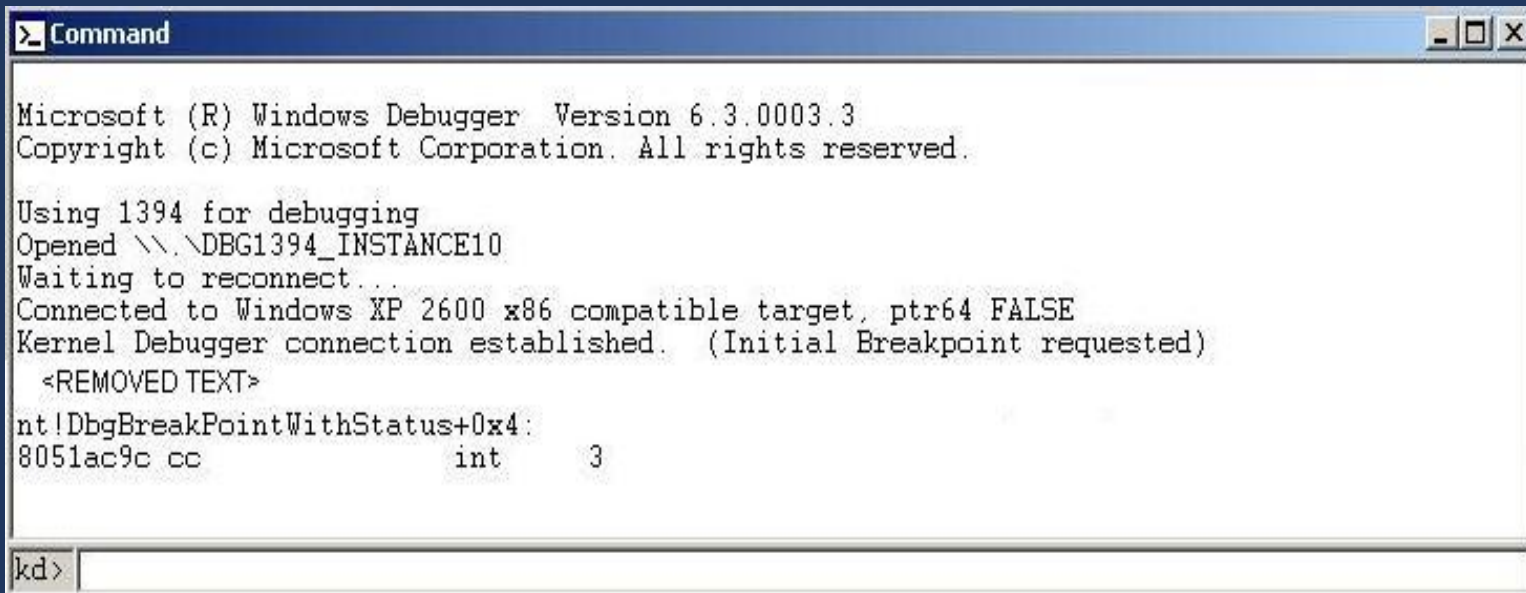
Host Software Configuration With WinDBG

- Launch WinDBG
 - Select File->Kernel Debug (Ctrl-K)
 - Select 1394 property page
 - Enter unique channel number for target system
 - If not already installed, this launches the installation of debug drivers:



Host Software Configuration With WinDBG

- Repeat Kernel Debug connection:
 - File->Kernel Debug (Ctrl-K)
 - Specify channel on 1394 property page
- Connection is established:



```
Command

Microsoft (R) Windows Debugger Version 6.3.0003.3
Copyright (c) Microsoft Corporation. All rights reserved.

Using 1394 for debugging
Opened \\.\DBG1394_INSTANCE10
Waiting to reconnect...
Connected to Windows XP 2600 x86 compatible target, ptr64 FALSE
Kernel Debugger connection established. (Initial Breakpoint requested)
<REMOVED TEXT>
nt!DbgBreakPointWithStatus+0x4:
8051ac9c cc          int     3

kd>
```

How Does It Work?

■ Target systems

- On boot, HAL attempts to find and initialize the PCI or PC Card device, in this case, the 1394 host controller
- KD1394.DLL configures system for 1394 debug
 - Initialize 1394 controller
 - Expose debug support in 1394 config ROM
 - Enable 1394 physical access to system memory
 - Map KD packets to physical memory for processing

■ Host systems

- Probe bus for target systems by reading in config rom data
 - If target is found, retrieve channel number and physical memory address
 - If kd client is launched for specified channel, then initiate the debug connection and start transport of KD packets

1394 Debug Revisions

- Windows XP – Build 2600
 - First release of 1394 debug, based on isochronous data transfer
 - Great proof of concept release, but many implementation issues
 - Supports maximum of 4 target systems
- Windows XP Service Pack 1
 - New 1394 debug transport protocol, based on async data transfer
 - Improved reliability
 - Supports bus maximum of 62 target systems
- Windows Server 2003
 - Incremental improvements
 - Better initialization of 1394 host controllers for debug support
- Windows XP Service Pack 2
 - Auto disable of host controller

Known Issues

- Controller Initialization
 - HAL doesn't always initialize the 1394 (PCI) host controller
 - If HKLM\System\CurrentControlSet\Services\PCI\Debug doesn't exist, then the controller failed to initialize
- Coexistence with the core 1394 driver stack
 - Both KD1394.DLL and the core 1394 driver stack access the controller exclusive of each other
- Power Management
 - KD1394.DLL depends on core 1394 driver stack for power state change notifications
 - Disabling 1394 driver stack prevents power state notification to KD1394.DLL

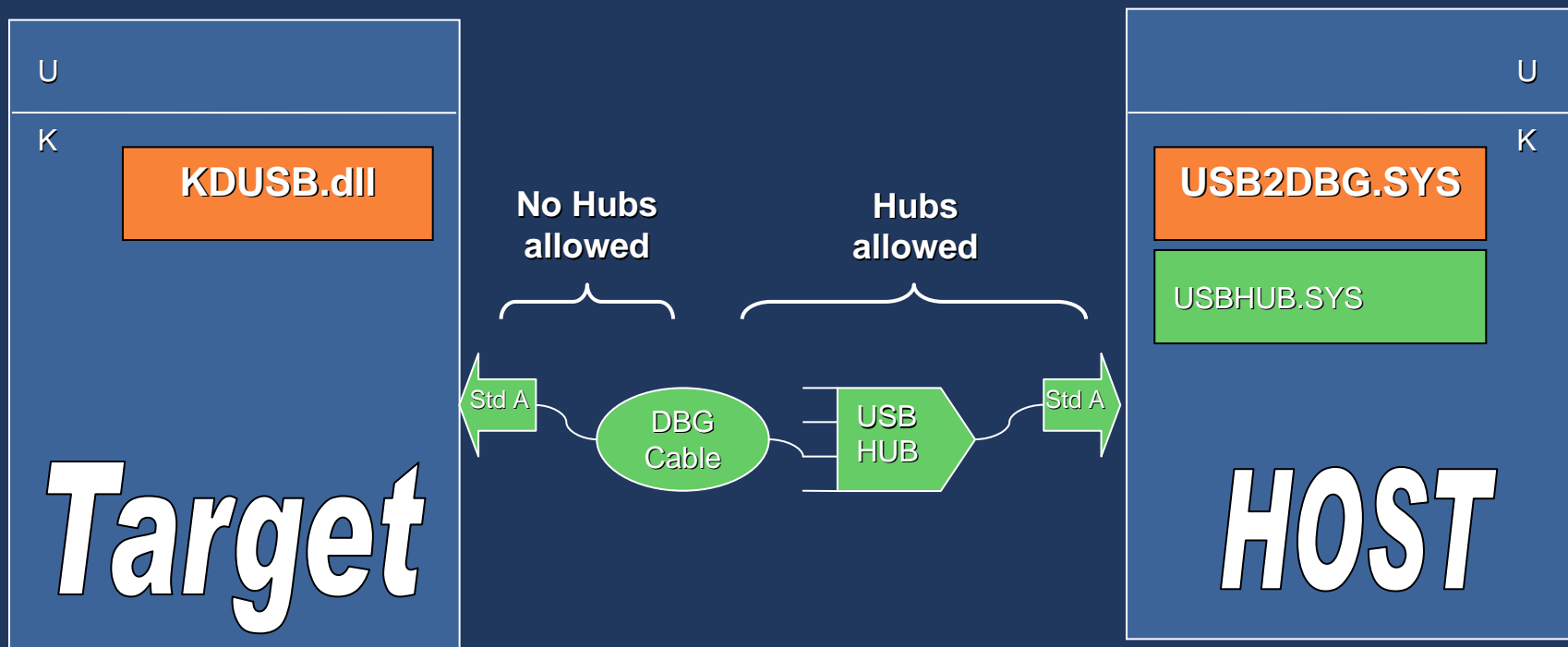
Microsoft Future Plans For 1394 Debugging

- Remove dependency between 1394 debug and core 1394 stack for power state notifications
- Improve HAL PCI and PC Card device initialization
- Add support for 1394 user mode debug
- Auto disable of host controller when used for debugging – shipping in Windows XP Service Pack 2 and Windows Server 2003 Service Pack 1

Kernel Debugging Over USB 2.0

- Design wins
 - Aids mobile system (without 1394/serial port) debugging
 - Provides a backup to debugging 1394/serial
 - Can not debug cardbus issue if laptop has only one cardbus slot consumed by 1394 debugging card
- Hardware requirements
 - EHCI controller must support Kernel Debugging (EHCI Spec V1.0 – Appendix C)
 - Kernel debugging PC-to-PC cable must support debugging specification
- Software/BIOS requirements
 - USB KD driver and DLL needed
 - BIOS might need to identify debug port# (via ACPI Port Properties)
 - BIOS will need to support hi-speed device communication

Kernel Debug Over USB 2.0 – Basic Architecture



- Debug cable must directly connect to specific port on Target
- Multiple debug sessions off a single Host are allowed
- Special protocol being developed to optimize kernel debugging over USB 2.0

Recommendations

- OEMs:
 - Expose the USB 2.0 debug port capable of kernel debugging
 - BIOS must support hi-speed and talk to EHCI
 - Do NOT connect internal devices to USB 2.0 debug port
- BIOS Developers
 - BIOS must support hi-speed and talk to EHCI
- IHVs (developing USB 2.0 Kernel Debug cables)
 - Use the USB Debug device Specification (version 0.9 or later)
 - Ensure that either end of cable can work at full speed while the opposite end is working in hi-speed mode.

Participate in the beta and provide feedback

Call To Action

- Ensure that your device and driver work on ALL 64-bit enable Windows operating systems
- Participate in the Kernel Debugger over USB 2.0 Beta Program

Community Resources

- Community Sites
 - <http://www.microsoft.com/communities/default.mspx>
- List of Newsgroups
 - <http://communities2.microsoft.com/communities/newsgroups/en-us/default.aspx>
- Attend a free chat or webcast
 - <http://www.microsoft.com/communities/chats/default.mspx>
 - <http://www.microsoft.com/seminar/events/webcasts/default.mspx>
- Locate a local user group(s)
 - <http://www.microsoft.com/communities/usergroups/default.mspx>
- Non-Microsoft Community Sites
 - <http://www.microsoft.com/communities/related/default.mspx>

Additional Resources

- Debugging Tools and other information:
 - <http://www.microsoft.com/whdc/ddk/debugging>
- Questions? Comments?
 - E-mail: [windbgfb @ microsoft.com](mailto:windbgfb@microsoft.com)
 - Newsgroup: microsoft.public.windbg
- Related Sessions
 - USB – Tips and Tricks

Question & Answer