



UEFI Driver Development Training Testing and Debugging

Fang Hua
UEFI Development
Intel

Agenda

- EFI Shell Testing
- Debugging code statements
- COM cable usage
- American Arium usage
- Testing methodologies



EFI Shell usage

- The simplest way to test UEFI Drivers
 - Pros
 - No hardware to install
 - Can control driver selection process
 - Some additional tools exist (i.e. DrvCfg)
 - Cons
 - No source level debugging
 - Extra code required for some (i.e. `printf()`)
 - In NT32 emulator Windows will interfere.
 - Hints
 - The '-b' command line parameter stops each page and waits



Testing Main Driver Functions

- DH
- load / unload
- connect / disconnect / reconnect
- DrvCfg / DrvDiag



Testing Driver Functions

Driver.c

ComponentName.c

GetDriverName()

GetControllerName()

DriverConfiguration.c

DriverDiagnostics.c

```
Shell> dh -d DriverHandle
```

```
Shell> Drivers
```



Testing Driver Functions

Driver.c

ComponentName.c

`GetDriverName()`

`GetControllerName()`

DriverConfiguration.c

DriverDiagnostics.c

```
Shell> dh -d DeviceHandle
```

```
Shell> Devices
```

```
Shell> DevTree
```



DH

- Shell> DH
 - Displays a list of handles allocated in the system with each device's handle identifier (i.e. 61)
- Shell> DH -d [handle]
 - Displays details
 - **Verify**
 - **ComponentName**
 - **functionality**
 - Image Info
 - Protocols consumed

```
800 600
56: Image (IdeBus) DriverBinding ComponentName
57: Image (Uhci) DriverBinding ComponentName
58: Image (Umdl) DriverBinding
59: Image (ScsiBus) DriverBinding ComponentName
5A: Image (ScsiDisk) DriverBinding ComponentName
5B: Image (UsbBot) DriverBinding ComponentName
5C: Image (UsbBus) DriverBinding ComponentName
5D: Image (UsbChid) DriverBinding ComponentName
5E: Image (UsbChid) DriverBinding
5F: Image (UsbKb) DriverBinding ComponentName
60: Image (UsbMassStorage) DriverBinding ComponentName
61: Image (UsbMouse) DriverBinding ComponentName
62: Image (VLC)
63: DriverBinding ComponentName
64: Image (Dhcp4)
65: DriverBinding ComponentName
66: Image (SNP)
67: DriverBinding ComponentName
68: Image (Fat) DriverBinding ComponentName
69:
6A:
6B: Image (Shell) ShellInt
6C:
Shell> _
```

```
800 600
SystemTable...: 4C1EF90
DeviceHandle...: 4C16C90
FilePath.....: 2D2E62AA-9ECF-43B7-8219-94E7FC713DFE
PbFileName....: c:\efi\edk-dev-snapshot-20060612\edk\Sample\Platform\Nt32\B
uild\Nt32\UsbMouse.pdb
ImageBase.....: 3C170000 - 3C1D0000
ImageSize.....: 6000
CodeType.....: BS_code
DataType.....: BS_data
DriverBinding (51E4010)
ComponentName (51E418C)
4CB82451-C207-405B-9694-99E013251341 (03C7D018)
Shell> dh -d 61
61: Image (UsbMouse) DriverBinding ComponentName
Driver Name : Usb Mouse Driver
Image Name : 2D2E62AA-9ECF-43B7-8219-94E7FC713DFE
Driver Version : 00000010
Driver Type : <UNKNOWN>
Configuration : NO
Diagnostics : NO
Managing : <NONE>
Shell> _
```



Testing Driver Functions

Driver.c

ComponentName.c

DriverConfiguration.c

DriverDiagnostics.c

DriverEntryPoint()
Unload()
NotifyExitBootServices()
NotifySetVirtualAddressMap()
Supported()
Start()
Stop()
Produced Protocol Functions

```
Shell> Load -nc DriverName.efi
```

- Other drivers may connect, but not that one



Testing Driver Functions

Driver.c

ComponentName.c

DriverConfiguration.c

DriverDiagnostics.c

DriverEntryPoint()
Unload()
NotifyExitBootServices()
NotifySetVirtualAddressMap()
Supported()
Start()
Stop()
Produced Protocol Functions

Shell> Load *DriverName.efi*



Testing Driver Functions


Driver.c

ComponentName.c

DriverConfiguration.c

DriverDiagnostics.c

DriverEntryPoint()
Unload()
NotifyExitBootServices()
NotifySetVirtualAddressMap()
Supported()
Start()
Stop()
Produced Protocol Functions



Shell> Unload *DriverHandle*



Testing Driver Functions

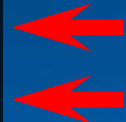
Driver.c

ComponentName.c

DriverConfiguration.c

DriverDiagnostics.c

```
DriverEntryPoint()  
Unload()  
NotifyExitBootServices()  
NotifySetVirtualAddressMap()  
Supported()  
Start()  
Stop()  
Produced Protocol Functions
```



```
Shell> Connect DeviceHandle DriverHandle
```



Testing Driver Functions


Driver.c

ComponentName.c

DriverConfiguration.c

DriverDiagnostics.c

DriverEntryPoint()
Unload()
NotifyExitBootServices()
NotifySetVirtualAddressMap()
Supported()
Start()
Stop()
Produced Protocol Functions



Shell> Disconnect *DeviceHandle*



Testing Driver Functions

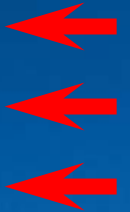
Driver.c

ComponentName.c

DriverConfiguration.c

DriverDiagnostics.c

DriverEntryPoint()
Unload()
NotifyExitBootServices()
NotifySetVirtualAddressMap()
Supported()
Start()
Stop()
Produced Protocol Functions



```
Shell> Reconnect DeviceHandle
```

```
Shell> Reconnect -r
```



Testing Driver Functions

Driver.c

ComponentName.c

DriverConfiguration.c

DriverDiagnostics.c

DriverEntryPoint()
Unload()
NotifyExitBootServices()
NotifySetVirtualAddressMap()
Supported()
Start()
Stop()
Produced Protocol Functions

Install and Boot an UEFI Compliant Operating System



Testing Driver Functions

Driver.c

ComponentName.c

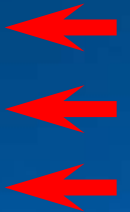
DriverConfiguration.c

DriverDiagnostics.c

SetOptions()

OptionsValid()

ForceDefaults()



```
Shell> DrvCfg -s DriverHandle DeviceHandle
Shell> DrvCfg -v DriverHandle DeviceHandle
Shell> DrvCfg -f DriverHandle DeviceHandle
Shell> DrvCfg -s DrvHndl DevHndl ChildHndl
```



Testing Driver Functions

Driver.c

ComponentName.c

DriverConfiguration.c

DriverDiagnostics.c

RunDiagnostics()



```
Shell> DrvDiag -s DriverHandle DeviceHandle
Shell> DrvDiag -e DriverHandle DeviceHandle
Shell> DrvDiag -m DriverHandle DeviceHandle
```



Testing Driver Functions

Driver.c

ComponentName.c

DriverConfiguration.c

DriverDiagnostics.c

DriverEntryPoint()
Unload()
NotifyExitBootServices()
NotifySetVirtualAddressMap()
Supported()
Start()
Stop()
Produced Protocol Functions



Depends on the protocol



Testing for leaks

- memmap
 - Displays a map of all memory in the system.
 - Can see what is allocated.

```
800 600

BS_data      0000000045FF000-0000000045FFFFF 0000000000000001 000000000000000F
BS_code      000000004600000-00000000460EFFF 000000000000000F 000000000000000F
RT_code      00000000460F000-000000004613FFF 0000000000000005 800000000000000F
BS_data      000000004614000-000000004630FFF 000000000000001D 000000000000000F
RT_data      000000004631000-000000004632FFF 0000000000000002 800000000000000F
BS_data      000000004633000-000000004C19FFF 00000000000005E7 000000000000000F
RT_data      000000004C1A000-000000004C1AFFF 0000000000000001 800000000000000F
BS_data      000000004C1B000-000000004C1DFFF 0000000000000003 000000000000000F
RT_data      000000004C1E000-000000004C1EFFF 0000000000000001 800000000000000F
BS_data      000000004C1F000-000000004C7EFFF 0000000000000060 000000000000000F
RT_data      000000004C7F000-000000004CFEFFF 0000000000000080 800000000000000F
BS_data      000000004CF0000-000000004D0FFFF 0000000000000011 000000000000000F
MemMapIO     0000000000B80000-0000000000B8BFFF 000000000000000C 8000000000000000

LoaderCode:   118 Pages (483,328)
LoaderData:   20 Pages (81,920)
BS_code :     449 Pages (1,839,104)
BS_data :     4,145 Pages (16,977,920)
RT_code :     51 Pages (208,896)
RT_data :     172 Pages (704,512)
available :   11,429 Pages (46,813,184)
MemMapIO :    12 Pages (49,152)
Total Memory: 64 MB (67,108,864) Bytes

Shell> _
```



Checking For Load / Unload Leaks

- gBS->AllocatePool() / gBS->FreePool()
- gBS->AllocatePages() / gBS->FreePages()
- gBS->InstallMultipleProtocolInterfaces()
- gBS->UninstallMultipleProtocolInterfaces()

```
Shell> Memmap
Shell> Dh
Shell> Load DriverName.efi
Shell> Memmap
Shell> Dh
Shell> Unload DriverHandle
Shell> Memmap
Shell> Dh
```



Checking For Memory Leaks

- gBS->AllocatePool() / gBS->FreePool()
- gBS->AllocatePages() / gBS->FreePages()

```
Shell> Memmap
Shell> Connect DeviceHandle DriverHandle
Shell> Memmap
Shell> Disconnect DeviceHandle DriverHandle
Shell> Memmap
Shell> Reconnect DeviceHandle
Shell> Memmap
```



Checking For Handle Leaks

- gBS->InstallMultipleProtocolInterfaces()
- gBS->UninstallMultipleProtocolInterfaces()

```
Shell> dh
Shell> Connect DeviceHandle DriverHandle
Shell> dh
Shell> Disconnect DeviceHandle DriverHandle
Shell> dh
Shell> Reconnect DeviceHandle
Shell> dh
```



Checking For Protocol Leaks

- gBS->OpenProtocol() / gBS->CloseProtocol()

```
Shell> OpenInfo DeviceHandle
Shell> Connect DeviceHandle DriverHandle
Shell> OpenInfo DeviceHandle
Shell> Disconnect DeviceHandle DriverHandle
Shell> OpenInfo DeviceHandle
Shell> Reconnect DeviceHandle
Shell> OpenInfo DeviceHandle
```



Debugging code statements

- ASSERT()
- ASSERT_EFI_STATUS()
- CR()
- EFI_BREAKPOINT()
- DEBUG()
- Setting the error level



ASSERT()

- Standard ASSERT call.
- Evaluates the argument.
 - If false print filename and line number and halts



ASSERT_EFI_STATUS

- Same as before but evaluates argument against EFI_STATUS instead of true / false.
 - If argument is not EFI_SUCCESS prints filename and line number and halts.



CR()

- Used to verify data types
- CR (Record, Type, Field, Signature)
 - Calls ASSERT(FALSE) if Data Structure Signature does not match



EFI_BREAKPOINT

- Generates a CPU break point instruction
- Requires source level debugging capability



DEBUG

- Allows for printing out of strings depending on error level.
 - `DEBUG(ErrorLevel, String, ...)`
 - Prints string if error level is active.



Setting the error level in console

```
Shell> Err ErrorLevel
```

- some standard error levels:

–EFI_D_ERROR	0x80000000
–EFI_D_INIT	0x00000001
–EFI_D_WARN	0x00000002
–EFI_D_INFO	0x00000040
–EFI_D_BLKIO	0x00001000
–EFI_D_UNDI	0x00010000



COM cable usage

- Setup up
- Messages



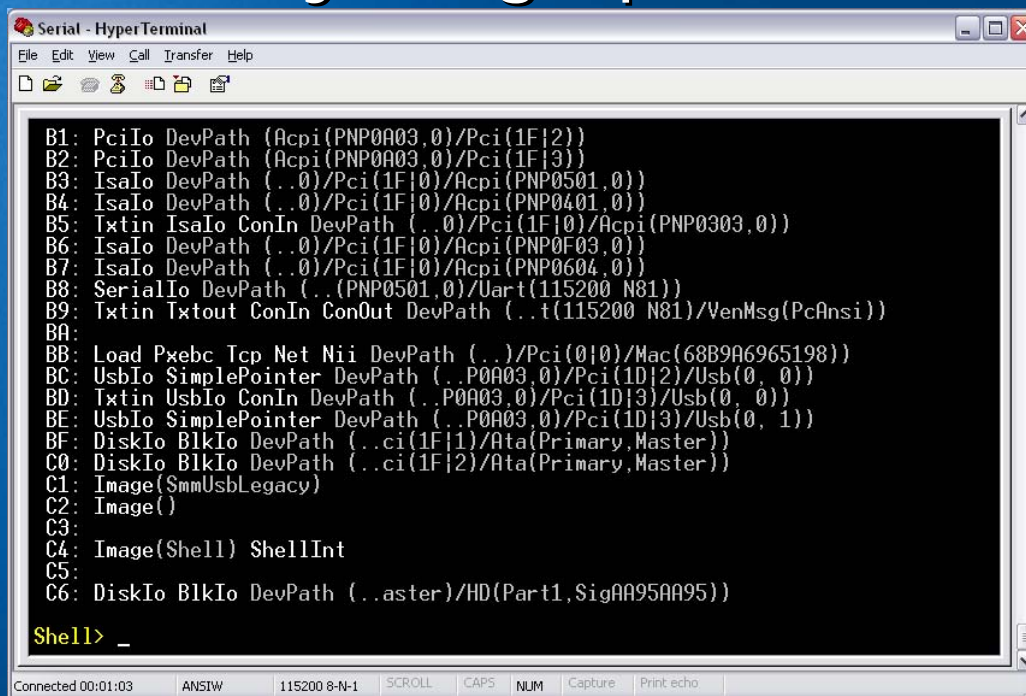
COM setup

- Connect a null modem cable
- Open a connection with these settings:
 - Bits per second: 115200
 - Data bits: 8
 - Parity: None
 - Stop Bits: 1
 - Flow Control: None
- Boot the Framework / UEFI based BIOS



COM messages

- Can run DH or any other command from within serial connection
- Obviously no graphics stuff



```
Serial - HyperTerminal
File Edit View Call Transfer Help
B1: PciIo DevPath (Acpi(PNP0A03,0)/Pci(1F|2))
B2: PciIo DevPath (Acpi(PNP0A03,0)/Pci(1F|3))
B3: IsaIo DevPath (..0)/Pci(1F|0)/Acpi(PNP0501,0))
B4: IsaIo DevPath (..0)/Pci(1F|0)/Acpi(PNP0401,0))
B5: TxtIn IsaIo ConIn DevPath (..0)/Pci(1F|0)/Acpi(PNP0303,0))
B6: IsaIo DevPath (..0)/Pci(1F|0)/Acpi(PNP0F03,0))
B7: IsaIo DevPath (..0)/Pci(1F|0)/Acpi(PNP0604,0))
B8: SerialIo DevPath (..(PNP0501,0)/Uart(115200 N81))
B9: TxtIn TxtOut ConIn ConOut DevPath (..t(115200 N81)/VenMsg(PcAnsi))
BA:
BB: Load PxeBc Tcp Net Nii DevPath (..)/Pci(0|0)/Mac(68B9A6965198))
BC: UsbIo SimplePointer DevPath (..P0A03,0)/Pci(1D|2)/Usb(0, 0))
BD: TxtIn UsbIo ConIn DevPath (..P0A03,0)/Pci(1D|3)/Usb(0, 0))
BE: UsbIo SimplePointer DevPath (..P0A03,0)/Pci(1D|3)/Usb(0, 1))
BF: DiskIo BlkIo DevPath (..ci(1F|1)/Ata(Primary,Master))
C0: DiskIo BlkIo DevPath (..ci(1F|2)/Ata(Primary,Master))
C1: Image(SmmUsbLegacy)
C2: Image()
C3:
C4: Image(Shell) ShellInt
C5:
C6: DiskIo BlkIo DevPath (..aster)/HD(Part1,SigAA95AA95))

Shell> _
```

Connected 00:01:03 ANSIW 115200 8-N-1 SCROLL CAPS NUM Capture Print echo



American Arium usage

- American Arium makes a HW debugger that works with EFI.
 - Enables source level debugging
 - www.Arium.com



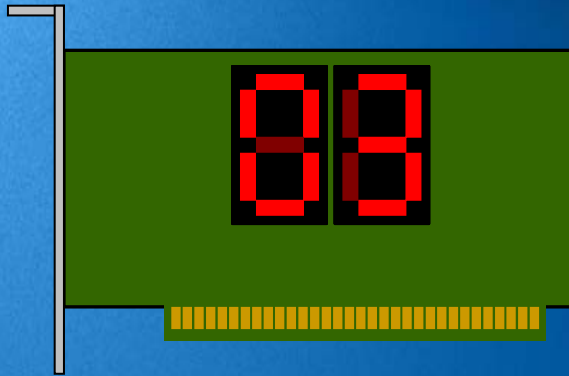
Testing methodologies

- How do I know if my driver works?
 - Can I use my device?
 - Can I boot off my device?
 - Is my driver fully compliant?
 - When am I done?
-
- UEFI IHV SCT





When DEBUG() is not Available



- POST Card (I/O 0x80)
 - PCI Root Bridge I/O Protocol
 - PCI I/O Protocol

```
Value = 0x03;
Status = PciIo->Io.Write (
    PciIo,                                // This
    EfiPciIoWidthUint8,                  // Width
    EFI_PCI_IO_PASS_THROUGH_BAR,         // BAR
    0x80,                                // Offset
    1,                                    // Count
    &Value                                // Buffer
);
```


May not work on all platforms
May produce unpredictable results
Must be removed from production drivers



When DEBUG() is not Available

Hello World
Check Point 1
Check Point 2
Check Point 3

- UART (COM1 I/O 0x3F8-0x3FF)
- UART (Platform Specific MMIO)
 - PCI Root Bridge I/O Protocol
 - PCI I/O Protocol



```
Status = PciIo->PollIo (PciIo, EfiPciIoWidthUint8,  
                        EFI_PCI_IO_PASS_THROUGH_BAR,  
                        0x3FD, 0x20, 0x20, 1000000, &Lsr);  
Status = PciIo->Io.Write (PciIo, EfiPciIoWidthUint8,  
                          EFI_PCI_IO_PASS_THROUGH_BAR,  
                          0x3F8, 1, &Data);
```

May not work on all platforms
May produce unpredictable results
Must be removed from production drivers



When DEBUG() is not Available



- VGA (MMIO 0xB8000-0xBFFFF)
 - PCI Root Bridge I/O Protocol
 - PCI I/O Protocol

```
VideoAddress  = 0xB8000 + (Row * 80 + Column) * 2;  
VideoCharacter = 0x0700 | Character;  
Status = PciIo->Mem.Write (PciIo, EfiPciIoWidthUint16,  
                             EFI_PCI_IO_PASS_THROUGH_BAR,  
                             VideoAddress, 1, &VideoCharacter);
```



May not work on all platforms
May produce unpredictable results
Must be removed from production drivers



Recommendations

- Test Functions with EFI Shell Commands
- Check for Leaks with EFI Shell Commands
- Install EFI Compliant Operating System
- Boot EFI Compliant Operating System
- Debug Macros Identify Critical Failures
- Use Same Techniques on all CPU Types
 - IA-32, Intel® 64, IA-64, EBC

Many Tools Available to Test and Debug



Testing EFI drivers

- Run drivers shell command
 - Verify driver attributes in database: handle, child controllers, device path generation, bus or device, config/diag protocol
 - Component name protocol should show name of driver, mfg, ver no. etc. to identify driver.
 - Version no. field should be set correctly for the driver



EFI Tests

- Should be done for EBC drivers and/or native drivers (on IA32, Intel® 64 and IA-64).
- Preferred drivers are EBC if they exist.
- Should test on each type of target platform (on IA32, Intel® 64 and IA-64) that the cards/drivers should work on.
- Run EFI IHV SCTs on EFI handles pertaining to the cards implementation (ie USB, Lan, SCSI etc.)
- Run manual tests below first before running EFI IHV SCTs.



Testing EFI drivers

- Do dh -d #handle on driver
 - Verify child controllers managed
 - Type of controller



Testing EFI drivers

- Do connect –r
- Verify with map command file systems/blkio produced by driver (or whatever driver stack the drivers should produce: usb – uhci, usbhub, usbkbd etc. Lan – Undi, snpdx etc.)
- Do disconnect on individual controllers
- Do reconnect –r (many times >100 with .nsh)
 - Disconnect /connect many times to look for resource leaks – memmap command or failures to connect. Available memory should not decrease constantly as each disconnect is run(it ok for avg to be constant but not to increase over many reconnects).
- # handles should not increase everytime reconnect occurs.



Testing EFI storage drivers

- Verify EFI file systems are usable.
 - Copy large files ranging from min block size to max lba blocks on media.
 - Verify copied files on known good system other than EFI (use windows or linux to do verify)
 - Repeat copying many times and verify with original images on known machine
 - Check first and last block (gpt) on partition
 - Do Dblk on blkio driver to check for correctness of data with known files (and see if driver works)



Testing EFI drivers

- Test DRVCFG(setup config) options on driver
 - Verify nvram settings in driver
- Test DRVDIAG options on driver
 - Verify diagnostic tests



Testing EFI drivers

- Run with console redirection on (through serial port)
- Run Drvcfg with console redirection
- Run Drvdiag with console redirection
- Remote head must be able to see/interact and record all console interactions with driver, drvcfg and drvdiag utilities.



Testing EFI drivers

- Lan drivers (non –storage)
 - Test booting to PXE server
 - Transfer large image (entire disk images)
 - Using TCP/IP stack in toolkit



Video UGA drivers

- Load UGA driver for display card
- Exercise setmode for different resolutions
- Run loadbmp and exercise blt to screen with various size pictures. Use known bmp so that all the buffer is used and pixels are verified
- Use shell mode command to change resolutions
- CIs x (0..9) verify different colors.
- Disconnect and reconnect UGA handles with console
- Verify with EFI OS (booting) that OS goes to desired setmode during OS loader.



