UEFI Driver Development Training Adding Code Lab

Kevin Li
UEFI Development
Intel

Agenda

- Private Data Structures
- Producing protocols
- Consuming protocols

Private Data Structures (PDS)

- Each instance of a driver needs an individual data structure.
  - Allocate from EFI memory services
  - Contains
    - Signature
    - Handle (s) to controller or child
    - Consumed protocol interfaces
    - Produced protocol interfaces
    - Private data fields and services
  - A containing record macro, CR() macro for each of the produced protocols is used to find to the data structure.

See §7 of EFI Driver Writer's Guide.
See §8 of UEFI Driver Writer's Guide Prelim

Disk I/O example

```
#define DISK_IO_PRIVATE_DATA_SIGNATURE EFI_SIGNATURE_32
  ('d','s','k','I')
typedef struct {
  UINTN Signature;
  EFI_DISK_IO_PROTOCOL DiskIo;
  EFI_BLOCK_IO_PROTOCOL *BlockIo;
  UINT32 BlockSize;
} DISK_IO_PRIVATE_DATA;
#define DISK_IO_PRIVATE_DATA_FROM_THIS(a) \
CR (a, DISK_IO_PRIVATE_DATA, DiskIo,
  DISK_IO_PRIVATE_DATA_SIGNATURE)
```

- The CR() entry allows for calls to the produced protocol to find the private data structure when required.

PDS memory

- You allocate the memory in the initialize function.
  - gBS->AllocatePool() + gBS->SetMem()
  - EfiLibAllocateZeroPool() better
  - Page memory is not recommended
- If your function fails you need to free the memory
- When your driver is closing (unload() is called) the memory needs to be freed also.
  - gBS->FreePool()

Accessing PDS

- When a function that you produce gets called it may need to use your CR() macro to get to the PDS

```
DiskIoReadDisk (
  IN EFI_DISK_IO_PROTOCOL *This
)
{
  DISK_IO_PRIVATE_DATA *Private;
  Private = DISK_IO_PRIVATE_DATA_FROM_THIS (This);
…
}
```

- Now the value of Private is set to the PDS that is for that instance of the driver.

SCSI Example

```
typedef struct {
  UINTN Signature;
  EFI_HANDLE Handle;
  EFI_SCSI_PASS_THRU_PROTOCOL ScsiPassThru;
  EFI_DEVICE_PATH_PROTOCOL *DevicePath;
  EFI_SCSI_PASS_THRU_MODE ScsiPassThruMode;
  UINT32 Pun;
  UINT64 Lun;
  UINT8 ScsiDeviceType;
  UINT8 ScsiVersion;
  BOOLEAN RemovableDevice;
} SCSI_DEVICE_CONTEXT;
```

Command Prompt:

- Setup the command prompt
  - Open a Visual Studio Command Prompt
  - Cd \efi\edk (or wherever you unzipped the EDK)
  - Set edk_source=c:\efi\edk
  - Cd sample\platform\nt32\build
  - For MSVS 8.0 Edit Config.env
    USE_VC8 = YES
  - nmake (make sure it all works)

Start with blank template

- Get the BlankDrv template from the CD under code\BlankDrv\version 0
- Copy it into the EDK\sample\bus\pci\
- Update the nt32.DSC file (look at DSC file in Version 0 directory for #DriverTraining)

  (EDK\Sample\Platform\nt32\build\nt32.DSC)

Add a PDS to the driver

- Add a PDS:
  1. Header file (EDK\Sample\Bus\PCI\BlankDrv\Blankdrv.h)
     1. Add some members to the struct yourself.
     2. Uncomment CR() and the extern pointer to the device
  2. Implementation file (EDK\Sample\Bus\PCI\BlankDrv\Blankdrv.c)
     1. Add a pointer to match the extern
     2. Allocate memory Hint: use EfiLibAllocateZeroPool(<size>) and save it to the global variable you added.

- Compile
- You can see an example at “Version 1” of BlankDrv on CD.

Test it!

- Go back to the EFI command prompt
  - System.cmd
  - Nmake run
- in the EFI Shell
  - map
  - fsnt0:
  - load blankdrv.efi
  - drivers (look at the bottom)
  - dh (look at the bottom)
  - dh <number> (~7A)

Produce a Protocol

- Add the include to your file for the protocol
- Add an instance of the protocol to your PDS
- Publish the Protocol (Start())
- Have functions available for consumers
- Unpublish the protocol (Stop())

InstallProtocolInterface() or InstallMultipleProtocolInterfaces() or EfiLibInstallAllDriverProtocols()

- Provide
  - Handle (or function can create one)
  - GUID (for each one)
  - Interface pointer (for each one)
- If successful the protocol(s) are published.

- EfiLibInstallAllDriverProtocols is limited to Driver Model protocols, but has fewer parameters to pass.

UnInstallProtocolInterface() or UnInstallMultipleProtocolInterfaces()

- Provide
  - Handle (or function can create one)
  - GUID (for each one)
  - Interface pointer (for each one)
- If successful the protocol(s) are unpoublished.

Produce a Protocol Part 1

- Open BlankDrv
- Produce DriverBinding Protocol:
  1. Header File
     1. #Include header reference
     2. Update PDS
        1. Add an object of protocol type
     3. Update CR Macro with protocol object (3rd parameter)
     4. Add function prototypes
        1. Get function information from EDK in Edk\Foundation\Efi\Protocol\DriverBinding\DriverBinding.h
        2. Get function information from the UEFI spec.
        3. You can also cut/paste from another driver

Produce a Protocol Part 2

- Still in BlankDrv
  2. Implementation file
     1. Initialize protocol object in PDS initialization
     2. Publish Protocol
     3. Make function bodies for protocol functions
- Compile - Nmake

- See “Version 2” of BlankDrv on CD for Example

Test it!

- Go back to the EFI command prompt
  - Nmake run
- in the EFI Shell
  - map
  - fsnt0:
  - load blankdrv.efi
  - drivers (look at the bottom)
  - dh (look at the bottom)
  - dh <number> (~7A)
  - Do you see “DriverBinding” beside your image?

Consuming Protocols

- Add the include to your file for the protocol
- Add a pointer to the protocol interface to your PDS
- Open the Protocol
- Call into the Protocol
- Close the Protocol

OpenProtocol()

- Opens a Protocol Interface
- Protocol Usage Tracked in Handle Database
- Agents Tracked with 3 Parameters
  - Image Handle, Device Handle, Attributes

CloseProtocol()

- Closes a Protocol Interface
- Removes Entry from Handle Database

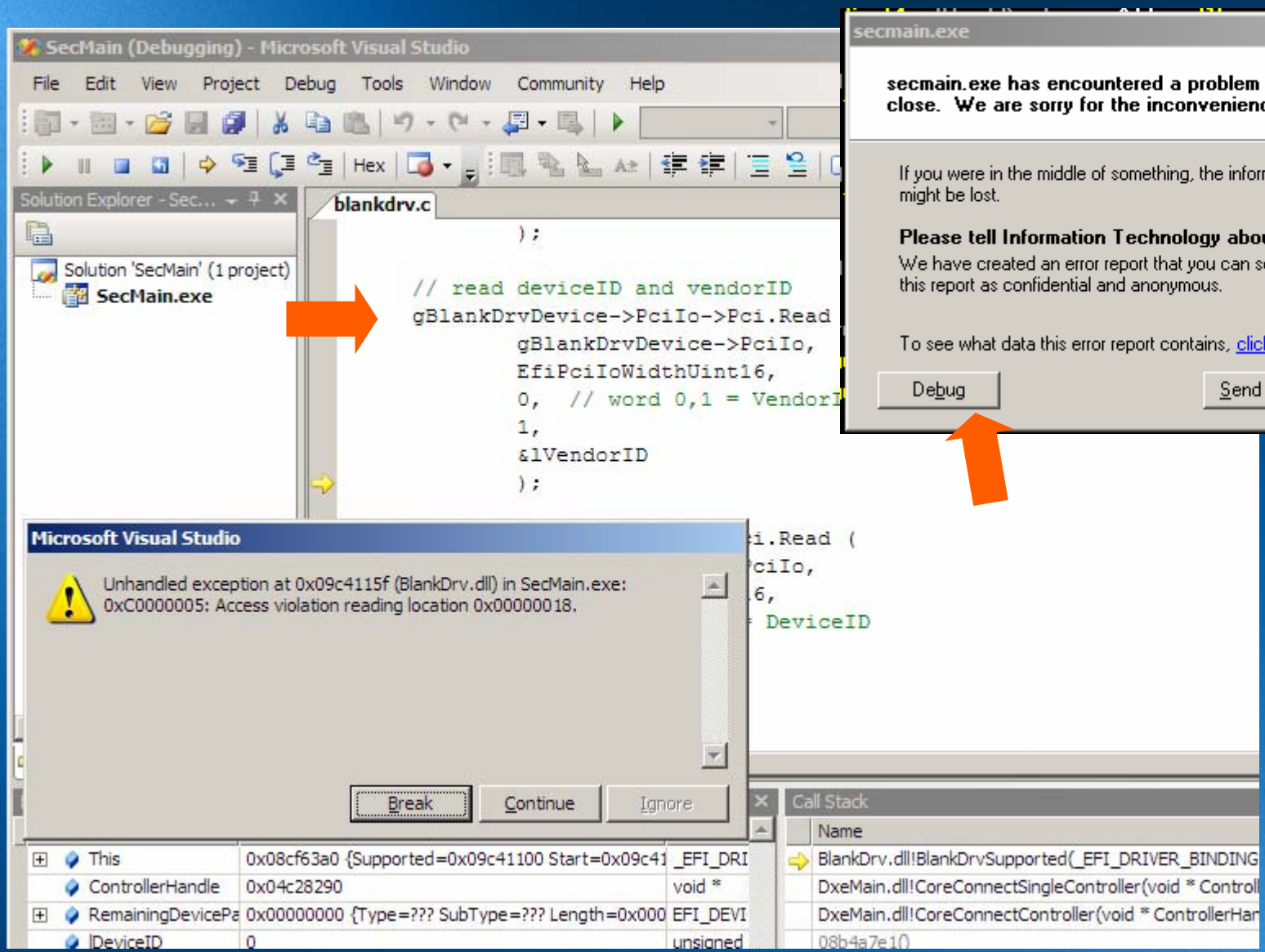
Consume a Protocol

- Open BlankDrv
- Consume PCI I/O:
  1. Header File:
     1. #Include header for protocol (PciIo)
     2. Add pointer in PDS (EFI_PCI_IO_PROTOCOL*)
  2. Implementation File
     1. In a DriverBinding-Supported:
        1. Open Protocol (use CR macro to get pointer)
        2. Call into Protocol (Get DeviceID / VendorID)
        3. Close Protocol
- Compile
- See “Version 3” of BlankDrv on CD

Test it!

- Go back to the EFI command prompt
  - Nmake run
- in the EFI Shell
  - map
  - fsnt0:
  - load blankdrv.efi
  - Verify where is crashes by going into the debugger.
  - Does the NT32 emulator crash when you try to access PCI configuration space? Welcome to the limit of the emulation environment.

Debug


SecMain (Debugging) - Microsoft Visual Studio
File Edit View Project Debug Tools Window Community Help
Solution Explorer - Sec...
Solution 'SecMain' (1 project)
SecMain.exe
blankdrv.c
// read deviceID and vendorID
gBlankDrvDevice->PciIo->Pci.Read
gBlankDrvDevice->PciIo,
EfiPciIoWidthUint16,
0, // word 0,1 = Vendor
1,
&lVendorID
);
DeviceID
Microsoft Visual Studio
Unhandled exception at 0x09c4115f (BlankDrv.dll) in SecMain.exe:
0xC0000005: Access violation reading location 0x00000018.
Break
Continue
Ignore
This
0x08cf63a0 {Supported=0x09c41100 Start=0x09c41
_EFI_DRI
ControllerHandle
0x04c28290
void *
RemainingDevicePa
0x00000000 {Type=??? SubType=??? Length=0x000
EFI_DEVI
DeviceID
0
unsigned
Call Stack
Name
BlankDrv.dll!BlankDrvSupported(_EFI_DRIVER_BINDING
DxeMain.dll!CoreConnectSingleController(void * Control
DxeMain.dll!CoreConnectController(void * ControllerHar
08b4a7e10
secmain.exe
secmain.exe has encountered a problem and needs to close. We are sorry for the inconvenience.
If you were in the middle of something, the information you were working on might be lost.
Please tell Information Technology about this problem.
We have created an error report that you can send to us. We will treat this report as confidential and anonymous.
To see what data this error report contains, click here.
Debug
Send Error Report
Don't Send

Comment out PCI Calls

- Open BlankDrv
  1. Implementation File:
     - Comment out PCI calls
- Compile
  - Nmake
- See “Version 4” of BlankDrv on CD
- Nmake run
- in the EFI Shell
  - map
  - fsnt0:
  - load blankdrv.efi
  - Verify the driver is loaded successfully

Add a Component Name Protocol and Unload Protocol

- Open BlankDrv
- Create a ComponentName.c file
  1. Add a EFI_UNICODE_STRING_TABLE
  2. Add the function for the BlankDrv ComponentNameGetDriverName
- Edit the BlankDrv.h
  1. Add consumed Protocol for EFI_PROTOCOL_DEFINITION (LoadedImage)
  2. Add Produced protocol for EFI_PROTOCOL_DEFINITION (ComponentName)
  3. To the BLANKDRV_DEVICE type definition structure add: EFI_COMPONENT_NAME_PROTOCOL ComponentNameInterface;
  4. Add Component Name Functions for getting the Drivers name and getting the Controller name

Add a Component Name Protocol and Unload Protocol (Cont.)

- Edit the BlankDrv.c file
  1. Add the unload function prototype to the "//Misc. functions local to this module" section
  2. Add a local pointer for the unload function of type EFI_LOADED_IMAGE_PROTOCOL
  3. Initialize the GetDriverName, the GetControllerName and the SupportedLanguages to the Component name protocol elements in the global BlankDrv driver
  4. Add the function to Install unload handler where the LoadedImageInterface->Unload gets assigned the same local pointer as in Step 2
  5. Where the Publishing of the UEFI driver model protocols is, Add the Component name protocol and remove the "NULL"
  6. Add the Unload function that is registered in the LoadImage protocol
     1. This will call the UninstallMultipleProtocolInterfaces with the Driver Binding and the Component Name protocols used for the BlankDrv Driver example

Add a Component Name Protocol and Unload Protocol (Cont.)

- Edit the Make.inf file
  1. Add the ComponentName.c to the [sources.common]
- Compile – Nmake
- See “Version 5” of BlankDrv on CD
  - Nmake Run
- in the EFI Shell
  - map
  - fsnt0:
  - load blankdrv.efi
  - Verify the driver is loaded successfully
  - Check out “dh” and “drivers”
  - Check out “unload”

Test it

```
 76: Image(\/blankdrv.efi) DriverBinding ComponentName

f8:\> _
```

Command >Dh

```
5D 0000000A ? - -  -  - Generic USB Mass Storage Driver    UsbMassStorage
5E 0000000A ? - -  -  - Usb Mouse Driver                   UsbMouse
5F 0000000A ? - -  -  - FAT File System Driver             Fat
76 00000001 ? - -  -  - UEFI BlankDrv Driver               \/blankdrv.efi

f8:\> _
```

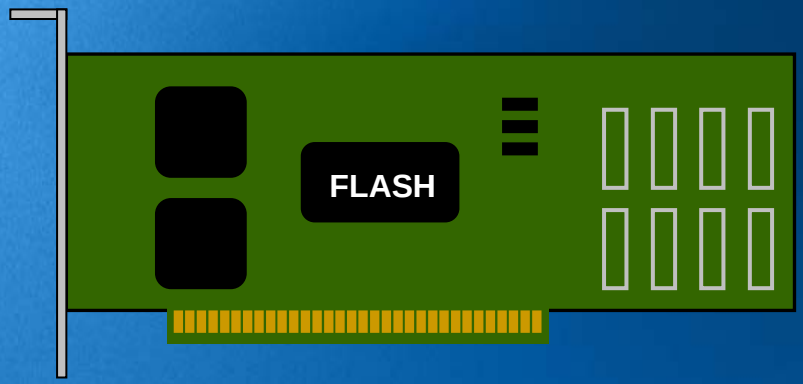
Command >Drivers

```
f8:\> dh -d 76
 76: Image(\/blankdrv.efi) DriverBinding ComponentName
     Driver Name     : UEFI BlankDrv Driver
     Image Name      : \/blankdrv.efi
     Driver Version  : 00000001
     Driver Type     : <UNKNOWN>
     Configuration   : NO
     Diagnostics     : NO
     Managing        : <NONE>


f8:\> unload 76
 76: Image(\/blankdrv.efi) DriverBinding ComponentName
Unload driver image (y/n)? y
unload: Success

f8:\> _
```

Command >Unload


intel®
Leap ahead™

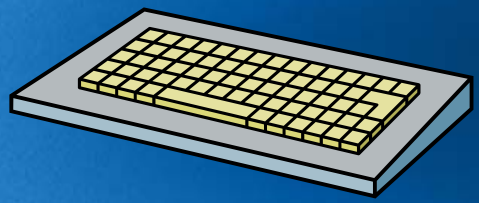
What Protocols are Consumed?



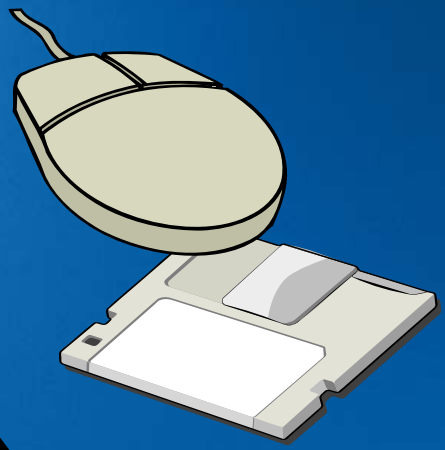
- PCI Adapters
  - –PCI I/O Protocol
  - –Device Path Protocol

- USB Peripherals
  - –USB I/O Protocol
  - –Device Path Protocol

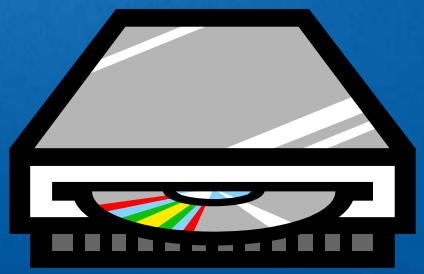
What I/O Protocols are Produced?



- Simple Input Protocol

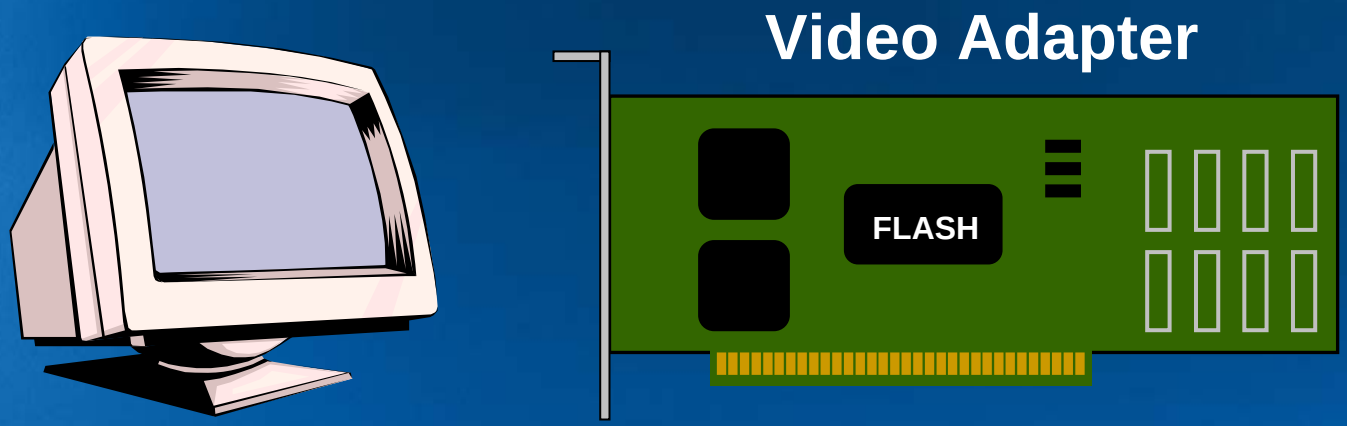


- Simple Pointer Protocol



- Block I/O Protocol

What I/O Protocols are Produced?

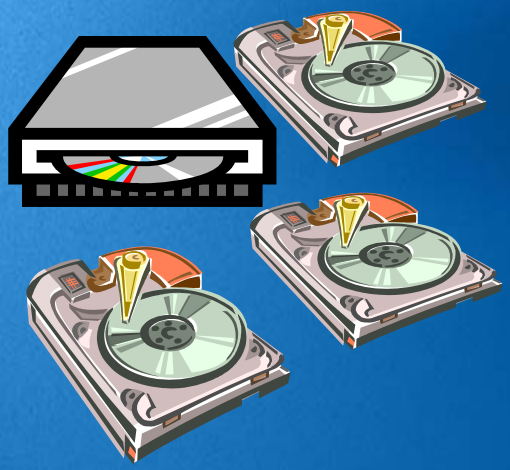


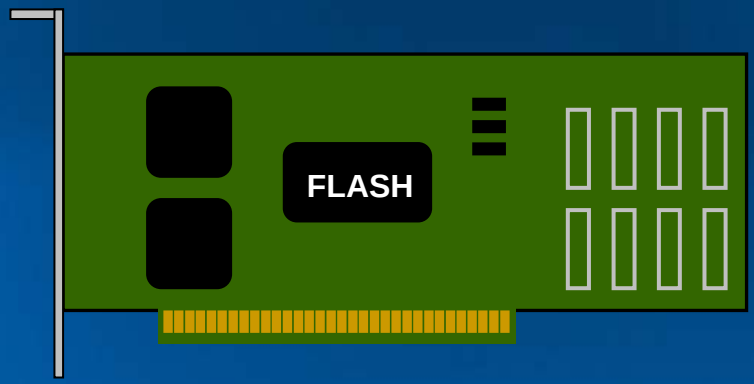
- UGA Draw Protocol and UGA I/O Protocol

or

- Simple Text Output Protocol

What I/O Protocols are Produced?





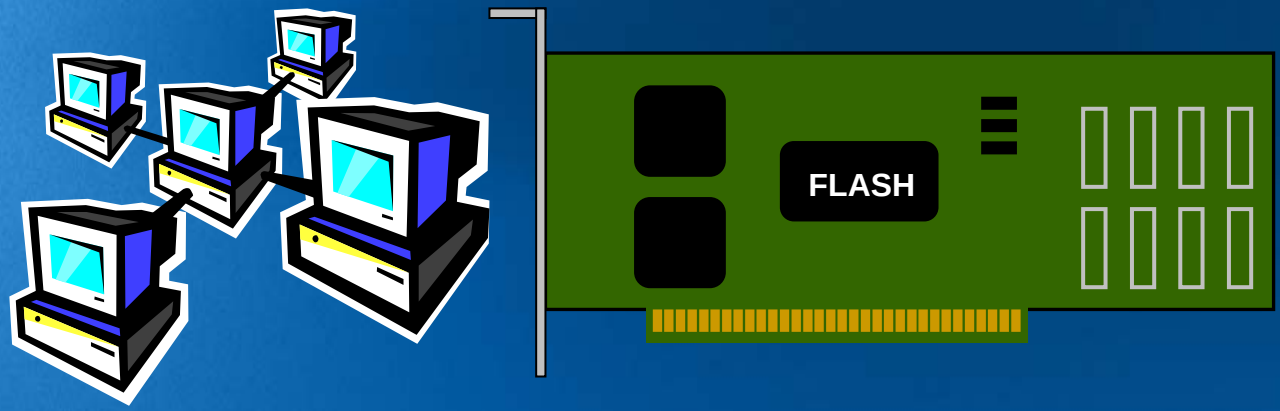
SCSI
SCSI RAID
Fiber Channel

- SCSI Pass Thru Protocol

and

- Block I/O Protocol

What I/O Protocols are Produced?



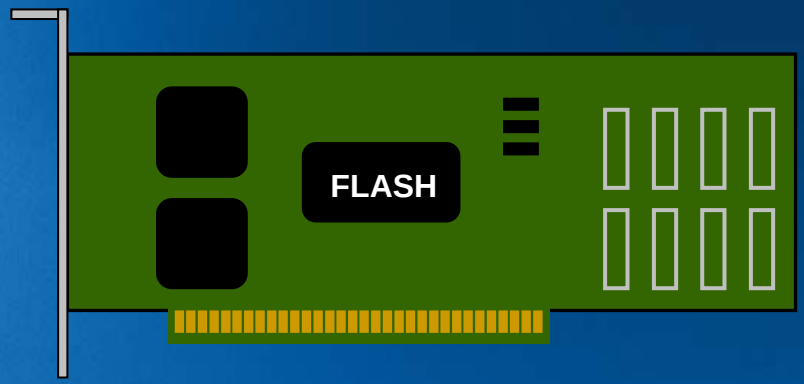
Network Interface Controller (NIC)

- UNDI

and

- Network Interface Identifier Protocol

What I/O Protocols are Produced?

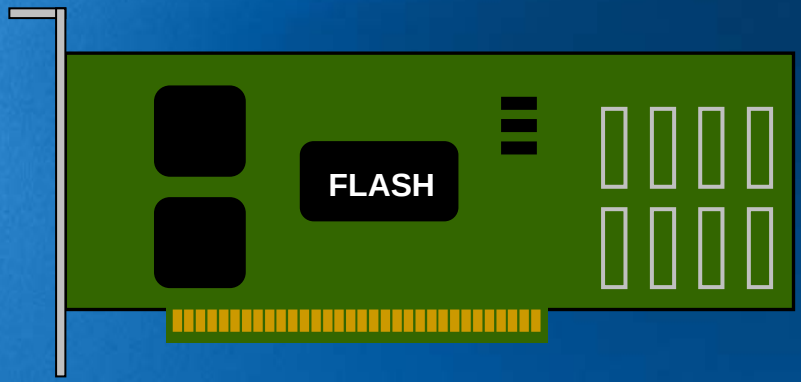


Serial Adapter
- **Single UART**
- **Multi-Port UART**
- **Modem**

Byte Stream Device

- Serial I/O Protocol
- Debug Port Protocol

What I/O Protocols are Produced?



USB Host Controller
- UHCI
- OHCI
- EHCI

- USB Host Controller Protocol

What protocols are produced?

Driver Type	
Keyboard	Simple Input Protocol
Mouse	Simple Pointer Protocol
Floppy	Block I/O Protocol
Video	UGA and UGA Draw or GOP
SCSI, SCSI RAID, FibreChannel	(Extended) SCSI Pass Thru and Block I/O
Network Interface Card	UNDI Protocol and NII
Serial Adapter or Byte Stream Device	Serial I/O Protocol and Debug Port Protocol
USB	USB Host Controller Protocol

What Protocols are Consumed?

Driver Type	Protocols
PCI Device Driver	PCI I/O Protocol Device Path Protocol Simple Text Input Protocol Simple Text Output Protocol
USB Device Driver	USB I/O Protocol Device Path Protocol Simple Text Input Protocol Simple Text Output Protocol


intel®
Leap ahead™