Writing and Debugging EBC Drivers

Michael Kinney
Principal Engineer
Intel

February 27th 2007

Disclaimer

THIS INFORMATION COTNAINED IN THIS DOCUMENT, INCLUDING ANY TEST RESULTS ARE PROVIDED "AS IS" WITH NO WARRANTIES WHATSOEVER, INCLUDING ANY WARRANTY OF MERCHANTABILITY, NONINFRINGEMENT FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION OR SAMPLE.INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT OR BY THE SALE OF INTEL PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT. Intel products are not intended for use in medical, life saving, or life sustaining applications.

Intel retains the right to make changes to its specifications at any time, without notice.

Recipients of this information remain solely responsible for the design, sale and functionality of their products, including any liability arising from product infringement or product warranty.

Intel may make changes to specifications, product roadmaps and product descriptions at any time, without notice.

Intel and the Intel logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

*Other names and brands may be claimed as the property of others.

Agenda

- **A Brief History of EBC**
- EBC Overview
- Designing and Implementing EBC Drivers
- Testing and Debugging EBC Drivers
- EBC Performance Guidelines
- Summary

Motivation and Goals

- Option ROM Cost w/ Multiple Images
  - For EFI 1.02 this was Itanium and IA-32
  - Costs continue to increase as EFI adds CPU architectures
- Design Goals
  - Simple instruction set
    - Lightweight efficient interpreter
  - Share a common call stack
    - Low overhead on calls
  - Share all data structures.
    - No translations required on EBC ⇔ native transitions
  - No library dependencies
  - No C coding restrictions

Options

- JAVA and Forth
  - Rejected due to large libraries
- IA-32 Interpreter
  - Rejected due to the size/complexity of the interpreter
  - Requires updates for new IA-32 instructions
- Remote Procedure Call (RPC) like mechanism
  - PRO: Can handle mixed CPU arch sizes
  - CON: Does not support all C constructs
  - CON: Function call overhead to transpose
  - CON: Difficult to share data structures
    - EFI System Table, Boot Services Table, Protocol Interfaces
  - EFI 1.02 Specification included some support
- EBC Instruction Set with Natural Addressing
  - PRO: Simple instruction set, no library dependencies
  - PRO: Share common stack and data structures
  - CON: Minor C coding restrictions

Agenda

- A Brief History of EBC
- **EBC Overview**
- Designing and Implementing EBC Drivers
- Testing and Debugging EBC Drivers
- EBC Performance Guidelines
- Summary

Natural Addressing

```
typedef struct {
  UINT64  BufferLength;
  VOID    *Buffer;
  UINT16  Checksum;
} MY_STRUCT;
```

	Byte Offset	
Field	32-bit	64-bit
BufferLength	0	0
Buffer	8	8
Checksum	12	16

- All fields are fixed size except INTN, UINTN, and pointers
- Byte Offset = C + N * Size of pointer in bytes
  - BufferLength: Offset = 0 + 0 * sizeof(VOID *) = 0 or 0
  - Buffer: Offset = 8 + 0 * sizeof(VOID *) = 8 or 8
  - Checksum Offset = 8 + 1 * sizeof(VOID *) = 12 or 16
- Encode both C and N into the instruction
  - C and N replace traditional offset field for address modes

Executing EBC Images

- EBC Interpreter
  - Implemented as a UEFI Driver
  - Typically stored in system FLASH (~10 KB compressed)
- Thunks
  - Native code that transfers control to/from EBC functions
  - Translates from native CPU ABI to EBC ABI (stack based)
  - Translates from EBC ABI (stack based) to native CPU ABI
- EBC executables use PE/COFF image format
- EBC executables loaded with EFI Boot Service LoadImage()
  - LoadImage() must support native and EBC images
  - Thunk to image entry point created by LoadImage()
- EBC executables started with EFI Boot Service StartImage()
  - Calls entry point thunk
- Thunks to exported functions created dynamically
  - Startup code contains BREAK instructions to create thunks
  - Function pointer references detected by compiler
    - Assignment or static initialization of protocol functions

EBC Images in PCI Option ROMs

- PCI Bus Driver discovers PCI Option ROMs
- PCI Option ROMs support multiple UEFI Images
  - UEFI Images may be compressed
- UEFI images dispatched by PCI Bus Driver
  - Non-UEFI images, including legacy, are ignored
  - UEFI Drivers dispatch in the order they appear
  - PCI Bus Driver calls LoadImage() and StartImage()
- Bus Specific Driver Override Protocol
  - Produced by PCI Bus Driver
  - Consumed by EFI Boot Service ConnectController()
  - Specifies priority order of Driver Binding Protocols
- Recommendations
  - Legacy Option ROM image first
  - Native UEFI Drivers next
  - EBC UEFI Drivers last
  - Compress driver images

Agenda

- A Brief History of EBC
- EBC Overview
- **Designing and Implementing EBC Drivers**
- Testing and Debugging EBC Drivers
- EBC Performance Guidelines
- Summary

When to use EBC

- Add-in Video Adapters
- Add-in Disk Controllers
- Not used for NICs (UNDI)
  - UNDI is runtime which must be native.
- Reduce driver image footprint
  - Adapters supporting multiple CPU types
    - IA-32 and IPF
    - IA-32 and X64
    - X64 and IPF
    - IA-32, X64, and IPF
- Reduce adapter SKUs

EBC Development Checklist

- Implement and Test Native Driver
- EBC Development Environments
- EBC Target Environments
- Driver Design Steps
- Driver Implementation Steps
- Portability Considerations

EBC Development Environments

- EDK on TianoCore.org
  - https://edk.tianocore.org/files/documents/16/313/Edk-Dev-Snapshot-20061228.zip
  - Config.env: EFI_GENERATE_INTERMEDIATE_FILE = YES
- Intel® C Compiler for EFI Byte Code Version 1.2 Build 20040123
  - Common Flags:
    - /W3 /WX /FAcs /Fa
  - http://www3.intel.com/cd/software/products/asmo-na/eng/compilers/efibc/219678.htm
- Microsoft* Linker Version 7.10.3077 and above
  - Common Flags:
    - /MACHINE:EBC /OPT:REF /ENTRY:EfiStart
    - /SUBSYSTEM:EFI_BOOT_SERVICE_DRIVER
    - EbcLib.lib
  - Microsoft* Visual Studio .NET 2003
  - Microsoft* Visual Studio 2005
  - Windows* DDK 3790.1830

EBC Target Environments

- UEFI Compliant Platforms
- EDK – DUET Platform
  - Boots UEFI environment on legacy platform
- EDK – NT32 Platform
  - UEFI Emulation environment for Windows
  - Not useful for drivers that touch hardware

Driver Implementation Steps

- Create Driver Directory
- Design Private Context Data Structure
- Add Source Files to Driver Directory
- Add .INF File to Driver Directory
- Add .INF file to .DSC file in Build Directory
- Run nmake to build driver

DEMO: Build EBC SampleDriver
DEMO: Build EBC HelloWorld

Portability Considerations

- Do Not Assume Max Number of Children
- Do Not Use Fixed Memory Addresses
- Do Not Use Assembly
- Do Not Use Floating Point Arithmetic
- Some Minor EBC Porting Considerations
- Bus Drivers Should Support Producing 1 Child at a time if possible (improves boot performance)

Common EBC Source Porting Issues

- EfiMain() and EfiStart() are reserved words
- Function Declarations
  - Must match Function Prototype if present
    - All parameter types and return types
- Pre-Init Data Structures
  - Function pointer fields must match declaration
  - Data fields can not reference sizeof()
  - EFI_STATUS indirectly references sizeof() for EBC
- case statement can not reference sizeof()
  - EFI_STATUS indirectly references sizeof() for EBC

DEMO: PortDemo1 PortDemo2

Common EBC Execution Issues

- Incorrect result of op between variable and immediate data
  - Workaround: Type convert immediate data to UINTN
- Incorrect result of arithmetic calculations
  - INTN and UINT8
  - INTN and UINT16
  - INTN and UINT32
  - UINTN and INT64
  - Workaround: Type convert fixed size to natural
- Incorrect CMP instruction generation
  - Workaround: Not an issue if UEFI base types are used

DEMO: PortDemo3

Agenda

- A Brief History of EBC
- EBC Overview
- Designing and Implementing EBC Drivers
- **Testing and Debugging EBC Drivers**
- EBC Performance Guidelines
- Summary

Testing Recommendations

- UEFI Self Certification Tests (SCTs)
- 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, Itanium® Processor Family, x64, EBC

Debug Methods

- DEBUG()/ASSERT() Macros
- POST Card
- UART Serial Port
- VGA Display
- EBC Debugger

Debug Macros

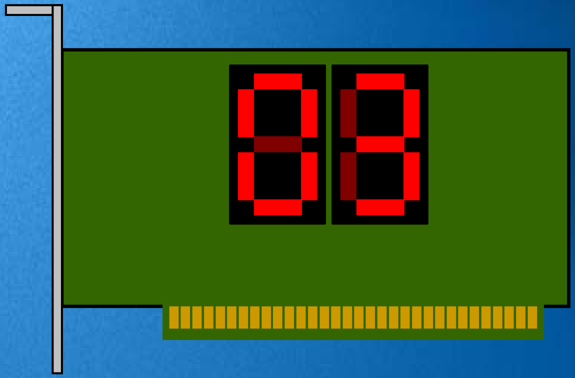
- ASSERT (Expression)
  - If Expression is FALSE, then print file name and line number and halt.
- ASSERT_EFI_ERROR (Status)
  - If Status is not EFI_SUCCESS, then print file name and line number and halt.
- CR (Record, Type, Field, Signature)
  - ASSERT()s if Data Structure Signature does not match
- EFI_BREAKPOINT ()
  - Generate a CPU break point instruction

Debug Macros

- **DEBUG (ErrorLevel, String, …)**
  - Print String if ErrorLevel is active.

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

When DEBUG() is not Available



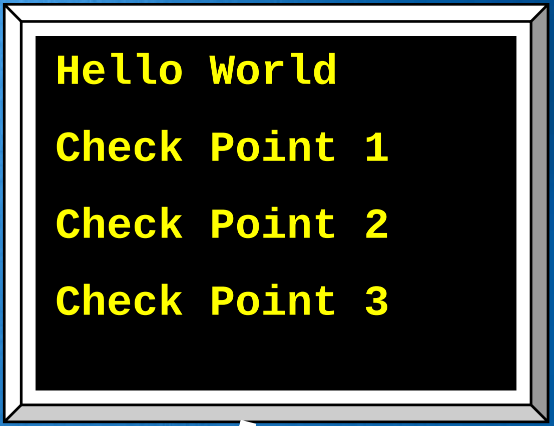
- POST Card (I/O 0x80)
  - PCI Root Bridge I/O Protocol
  - PCI I/O Protocol

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

CAUTION

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

When DEBUG() is not Available



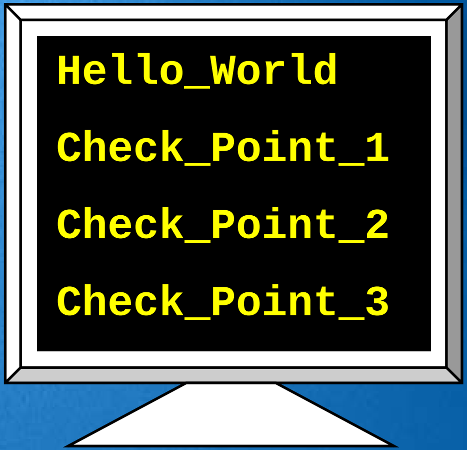
```
Hello World
Check Point 1
Check Point 2
Check Point 3
```

- 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



```
Hello_World
Check_Point_1
Check_Point_2
Check_Point_3
```

- 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);
```

CAUTION

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

EBC Debugger Demo

- Compile with /FAcs and /Fa
  - Generates .COD files with mixed source/asm
- Link with /MAP:mapfile
  - Generate .MAP file of functions in EBC driver
- Config.env
  - EFI_GENERATE_INTERMEDIATE_FILE = YES

DEMO: EBC Debugger

Agenda

- A Brief History of EBC
- EBC Overview
- Designing and Implementing EBC Drivers
- Testing and Debugging EBC Drivers
- **EBC Performance Guidelines**
- Summary

EBC Performance Guidelines

- Do as little work in EBC driver as possible
  - Use EFI Boot Services
  - Use EFI Runtime Services
  - Use Protocols produced by other drivers
- Perform operations at largest size possible

DEMO: BadPerf and GoodPerf

EBC Performance Guidelines

- EFI Boot Services
  - CopyMem(), SetMem()
- PCI I/O Services
  - PollMem() and PollIo()
  - Mem.Read(), Mem.Write(), Io.Read(), Io.Write()
    - Supports Buffer, FIFO, and Fill operations
    - EfiPciIoWidthUintX, EfiPciIoWidthFifoUnitX, EfiPciIoWidthFillUintx
  - Pci.Read() and Pci.Write()
    - Use buffer to perform many PCI cycles at once
  - CopyMem()
    - Video scroll operations when HW engine no available
  - Map(), UnMap()
    - Perform double buffering as required in native code

DEMO: CirrusLogic

Summary

- Use EFI Driver Writer's Guide for UEFI 2.0
  - Draft Version 0.94
- Implement and Test Native Driver First
- Be aware of EBC Source Portability Issues
  - No assembly or floating point support
- Call External Services for Performance
  - UEFI Boot Services
  - UEFI Protocols
- Use EBC Debug Methods and EBC Debugger
- Validate with SCTs, EFI Shell, and OS Install/Boot
- Follow EBC Option ROM Recommendations
  - EBC Images Last
  - Use UEFI Compression to reduce size

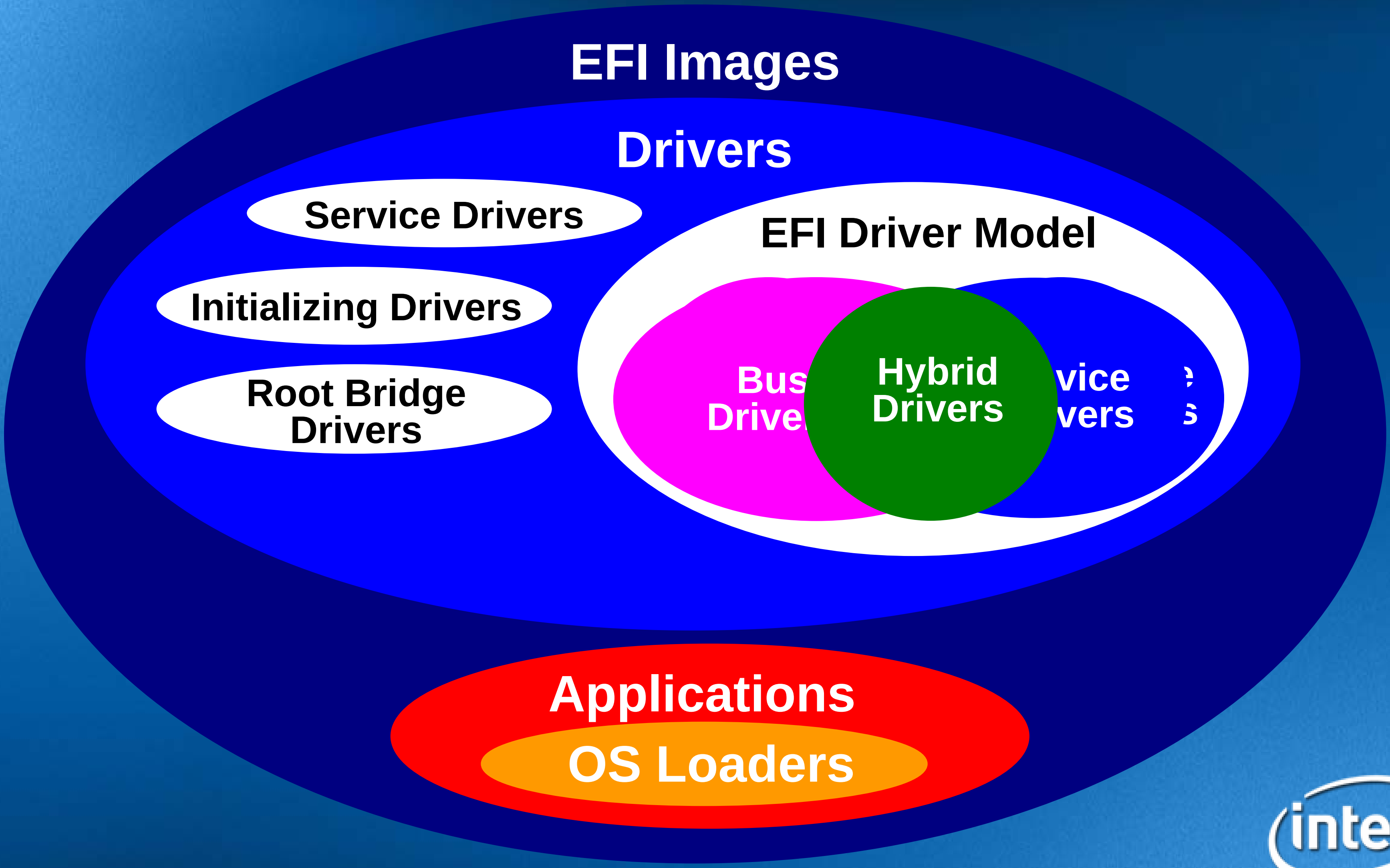
intel®

Leap ahead™

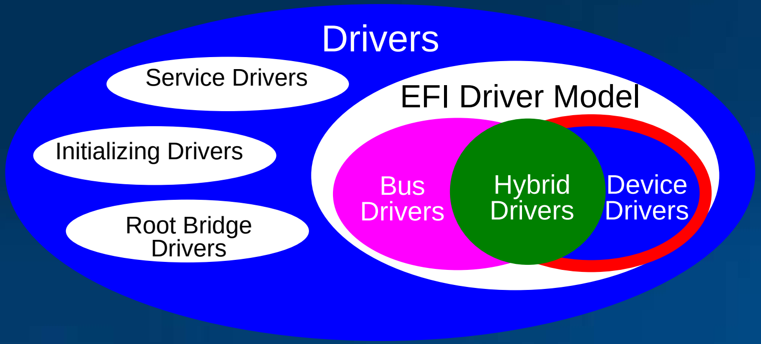
Definitions

- EFI Image
  - **Executable Image in a PE32 Image Format**
- EFI Driver
  - EFI Image that Typically Manages Physical Devices
  - Many Types are Possible
- Handle
  - Object Containing One or More Protocols
- Protocol
  - Object Containing Functions and Data
- Controller
  - Physical Device that is Managed by an EFI Driver
- Event
  - Object that may be Signaled or Waited Upon
  - Synchronous and Asynchronous Notifications

UEFI Driver Types



Device Driver

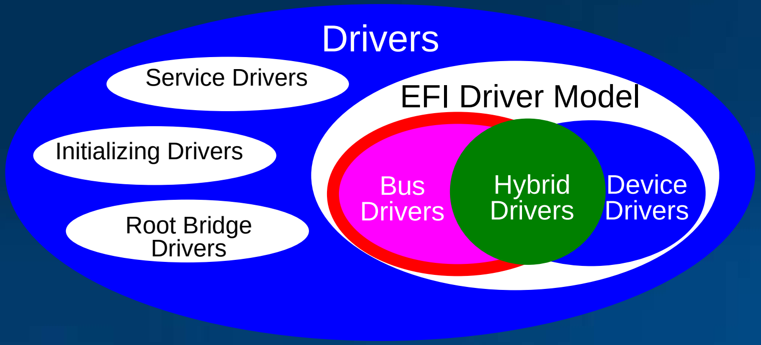


- Manages a Controller or Peripheral Device
- Start() Does Not Create Any Child Handles
- Start() Produces One or More I/O Protocols
  - Installed onto the Device’s Controller Handle

Examples:

PCI Video Adapters
USB Host Controllers
USB Keyboards / USB Mice
PS/2 Keyboards / PS/2 Mice

Bus Driver



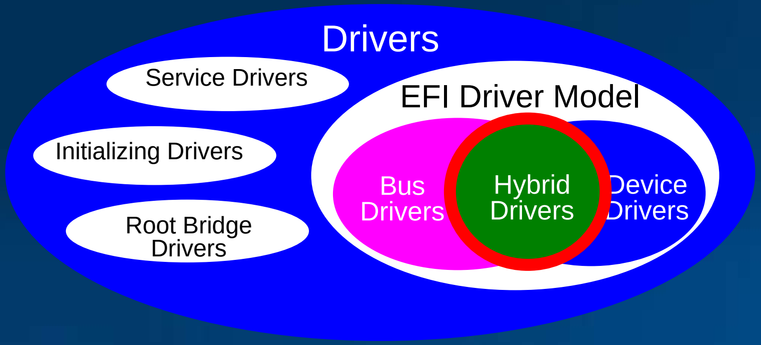
- Manages and Enumerates a Bus Controller
- Start() Creates One or More Child Handles
- Start() Produces Bus Specific I/O Protocols
  - Installed onto the Bus's Child Handles

Examples:

PCI Network Interface Controllers

Serial UART Controllers

Hybrid Driver



- Manages and Enumerates a Bus Controller
- Start() Creates One or More Child Handles
- Start() Produces Bus Specific I/O Protocols
  - Installed onto the Bus's Controller Handle
  - Installed onto Bus's Child Handles

Examples:

PCI SCSI Host Controllers

PCI Fiber Channel Controllers

Driver Design Steps

- Determine Driver Type
- Identify Consumed I/O Protocols
- Identify Produced I/O Protocols
- Identify EFI Driver Model Protocols
- Identify Additional Driver Features
- Identify Target Platforms
  - IA-32
  - Itanium Processor Family
  - EFI Byte Code (EBC)

Driver Design Checklist

	PCI Video	PCI RAID
Driver Type	Device	Hybrid
I/O Protocols Consumed	PCI I/O Device Path	PCI I/O Device Path
I/O Protocols Produced	GOP	SCSI Pass Thru Block I/O
Driver Binding	✓	✓
Component Name	✓	✓
Driver Configuration		✓
Driver Diagnostics	✓	✓
Unloadable	✓	✓
Exit Boot Services Event	sometimes	sometimes
Runtime		
Set Virtual Address Map Event		