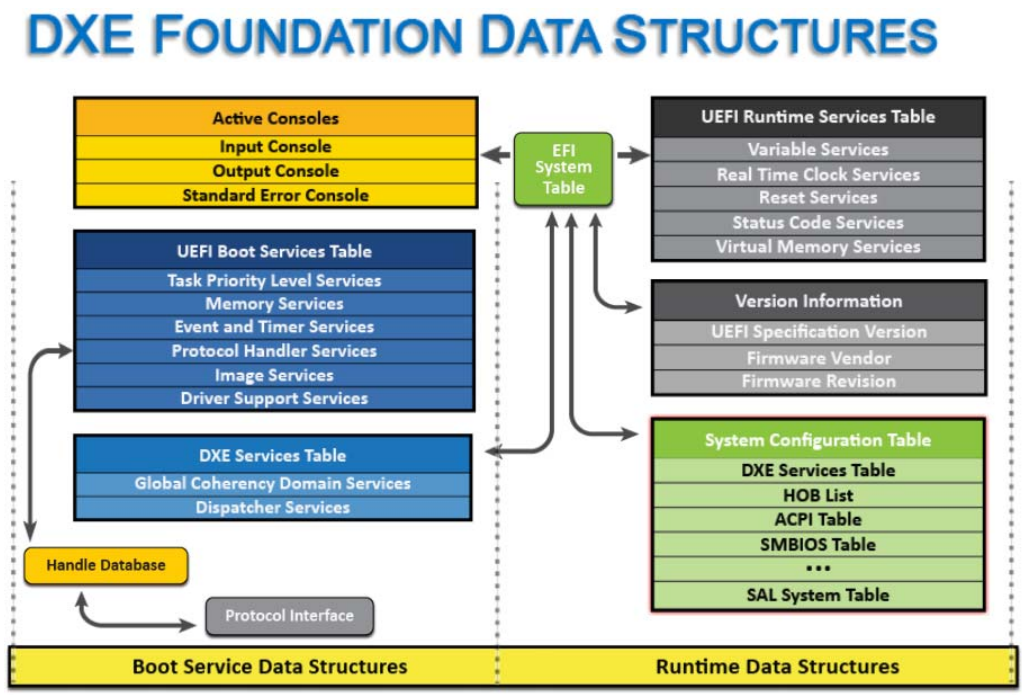
Lesson_1_PEI_and_Course_Intro

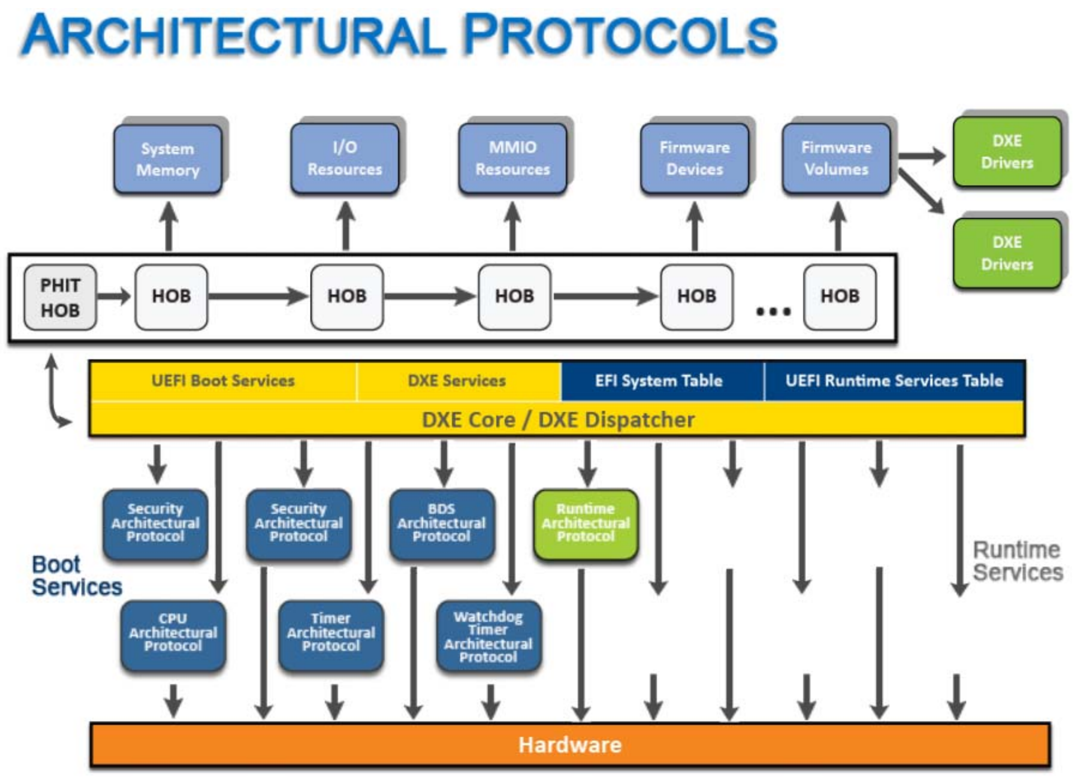
- framework->pi (platform initialization)
  - PI is power on to UEFI
- get to C compiler
  - "Intel architectural support"
    - cache as RAM
    - a.k.a. no evict mode
- PEI
  - discover boot mode
  - init main memory
  - launch DXE core
  - platform information-->DXE
- PEI components
  - dispatcher (core/PEI/PEIMAIN)
  - PEI services
    - function used by PEI-Ms
- PI spec defines firmware volumes
- PEI-M "DXE IPL"-->trasitions to DXE

Lesson_2_DXE

- second phase of PI
- protocols and drivers are implemented
- EFI tables are produced
- PI Spec Vol 2 = 2 DXE CIS
- DXE Foundation
  - comparable to PEI Core
  - initialize platform (i.e. chipset)
  - dispatch DXE drivers
    - DXE drivers are unique in that they can define dependencies
    - "use logic-based grammar to determine the driver's dispatch order"
  - dispatch UEFI drivers
    - no dependency rules, other than the fact that all boot and runtime svcs will be initialized
  - load boot manager
    - transfers control to BDS phase
  - HOB List dependent
    - HOBs transferred from PEI
  - No Hardware Specifics
    - DXE uses Architecture Protocols (APs) which are DXE drivers that abstract access to the hardware
  - No Hard-Coded Addresses
- DXE Components
  - drivers
    - init chipset, processor, components
    - provide APs that abstract DXE core from the platform
  - foundation
    - main executable binary responsible for dispatching drivers and producing a set of boot services, runtime services, and DXE services
  - architectural protocols -- hardware abstraction
  - dispatcher -- searches for and executes drivers in correct order
  - EFI system table
    - contains pointers to all UEFI service tables, configuration tables, handle databases, and console devices
- PEI to DXE
  - PEI init -> DXE IPL -> DXE foundation -> FW volumes -> DXE drivers
  - HOBs are only thing passed to DXE foundation
- Events
  - Signal Events
  - Timer Events
  - Wait Events
  - see PI spec Vol 2 and Vol 4
- EFI System Table
  - DXE dispatcher invokes DXE drivers
  - pointer to EFI system table is passed to every DXE driver
  - EFI System Table exposes services
    - UEFI boot services table - available only before ExitBootServices called
    - UEFI runtime services table - available before and after ExitBootServices
    - System Configuration table - stays in memory after ExitBootServices, contains info on industry specs (?)

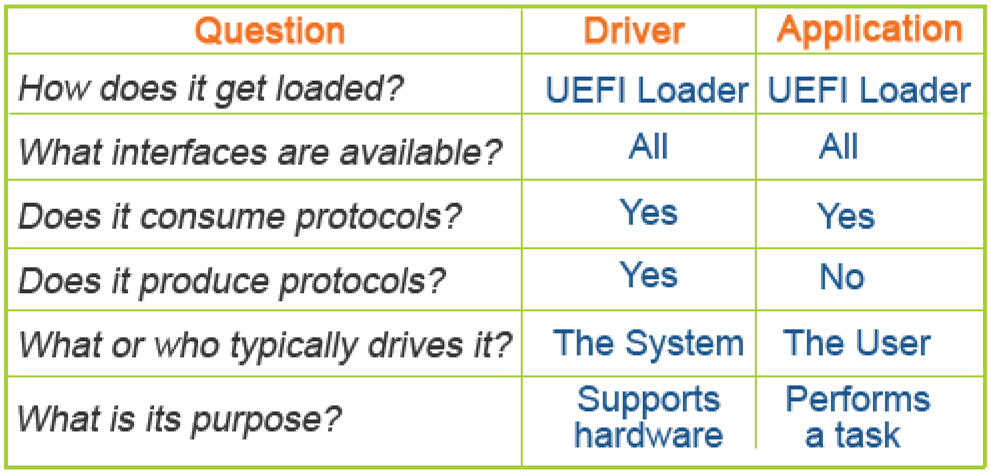
- 

    - ^ EFI System Table has pointers to all these data structures
    - UEFI Spec Section 4
- DXE Foundation code flow
    - single threaded -- BSP
    - one interrupt -- timer tick is the only software interrupt. devices are polled
    - uses events instead of multiple interrupts
- DXE Main
    - inits the DXE Foundation
    - consumes the HOB list
    - builds the EFI System Table, builds the Runtime Services table, builds the Boot Services table
    - transfers control to the DXE dispatcher
    - EDK II -- MdeModulePkg/Core/Dxe/DxeMain/DxeMain.c --> DxeMain()
- Architectural Protocols
    - produced by DXE drivers
    - isolate platform-specific hardware
    - support boot and runtime services
    - AP dependencies
        - dependency grammar
        - RegistryProtocolNotify() to notify when AP gets loaded
        - a priori list
    - PI spec Vol 2, Section 12


ARCHITECTURAL PROTOCOLS
System Memory
I/O Resources
MMIO Resources
Firmware Devices
Firmware Volumes
DXE Drivers
DXE Drivers
PHIT HOB
HOB
HOB
HOB
HOB
HOB
UEFI Boot Services
DXE Services
EFI System Table
UEFI Runtime Services Table
DXE Core / DXE Dispatcher
Security Architectural Protocol
Security Architectural Protocol
BDS Architectural Protocol
Runtime Architectural Protocol
Boot Services
Runtime Services
CPU Architectural Protocol
Timer Architectural Protocol
Watchdog Timer Architectural Protocol
Hardware

- DXE Dispatcher
  - loads and dispatches DXE and UEFI drivers
  - /MdeModulePkg/Core/Dxe/Image/Image.c --> CoreStartImage()
  - PI Spec Vol 2, Section 10.11
- DXE Driver Types
  - Early DXE Phase Drivers
    - execute first in the DXE phase, contain DEPEX to define dispatch order, produce APs, basic services, and initialization
  - UEFI Drivers
    - do not interact with HW, access to console and boot devices, abstract bus controllers
  - The last driver DXE Dispatcher calls is the Boot Device Selection (BDS) driver; BDS establishes consoles (keybd and video) and processing UEFI Boot Options. Can also go back and ask DXE Dispatcher to load other drivers, if necessary
- SMM Services
  - PI Spec, Vol 4, Section 1.2
  - Upon SMI, processor executes from a known, pre-defined starting vector
  - SMM code resides in special location (SMRAM) and locked after SMM code initialized
  - SMM Services and DXE Foundation services provide the same functionality, but in different locations. SMM in SMRAM.
  - DXE inits the SMM IPL --> the SMM Constructor acts like a mini-DXE dispatcher

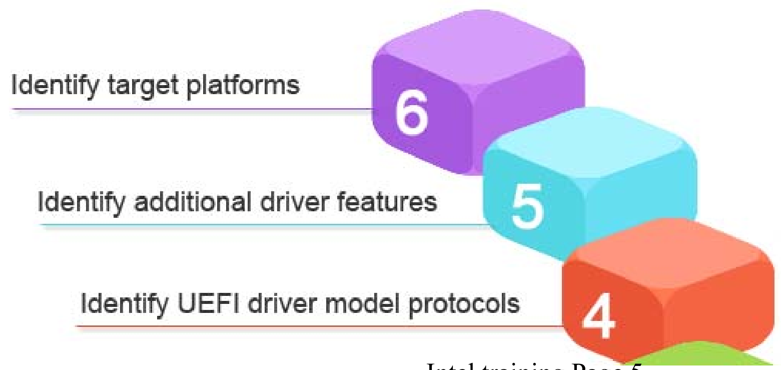
Lesson_3_Drivers

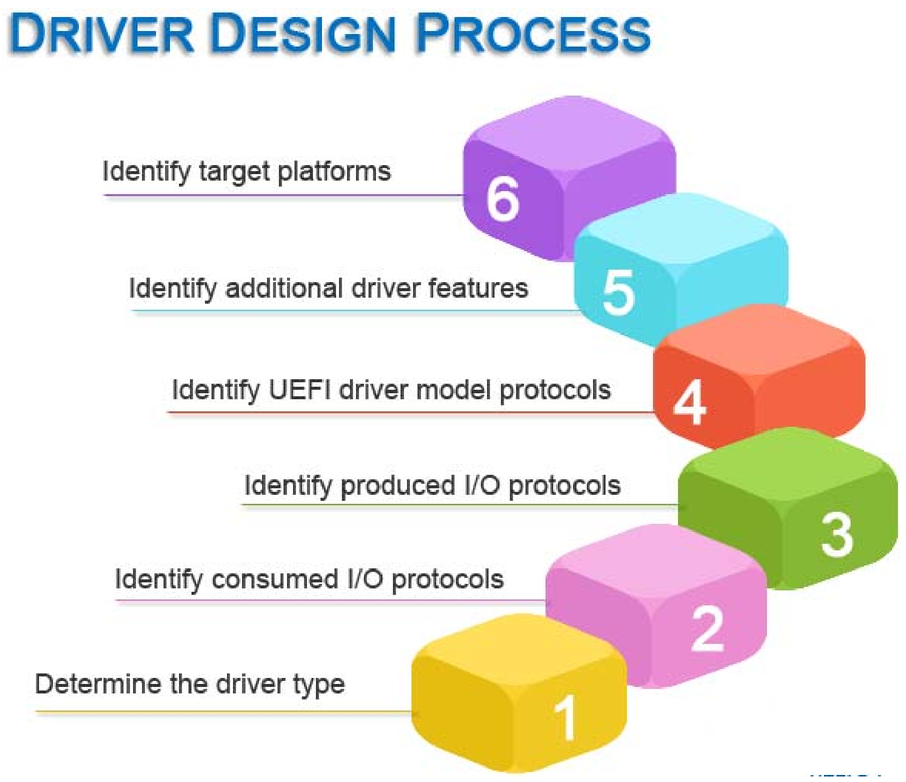
- UEFI Driver Writers Guide
  - like a cookbook: get recipes for what you're trying to accomplish and mix them together
- UEFI Drivers
  - support complex bus hierarchies
  - independed of hardware storage devices, including flash
  - establishes driver binding protocol
  - extended firmware
  - portable across platforms
- UEFI Driver contents
  - entry point
    - establishes protocols (w/hooks)
    - exits
    - driver re-loaded by DXE dispatcher when hook invoked
  - published functions
    - e.g. binding protocol (Supported, Start, Stop)
  - consumed functions
  - data structures
- driver vs. application
- 

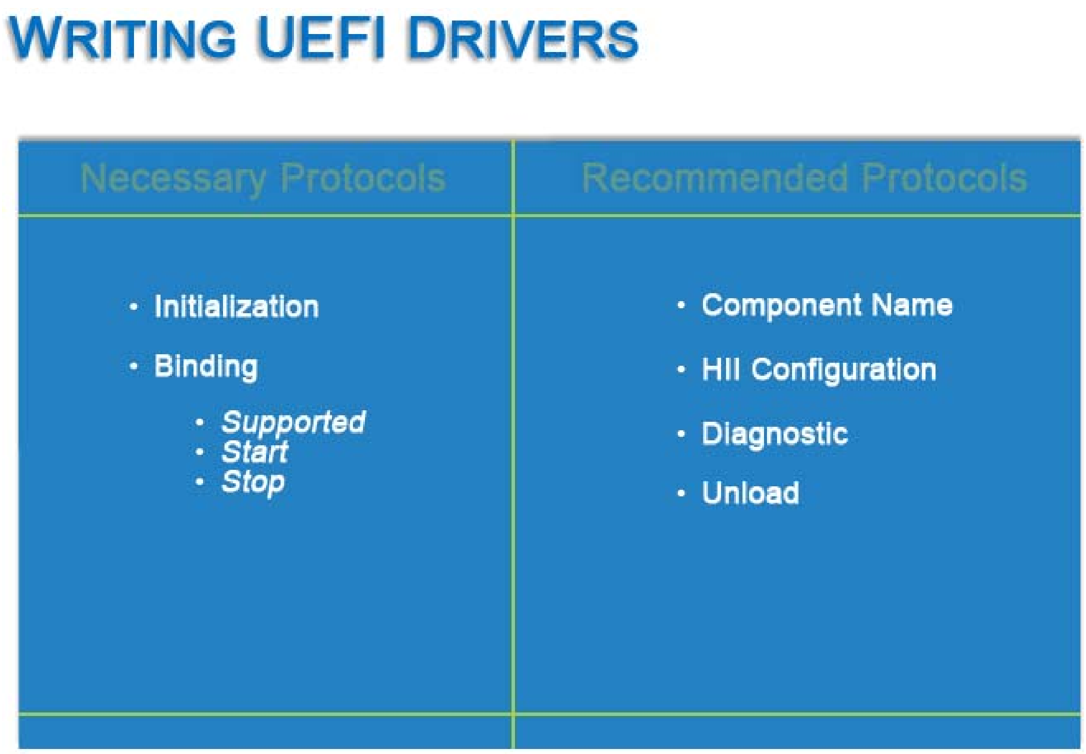
Question	Driver	Application
How does it get loaded?	UEFI Loader	UEFI Loader
What interfaces are available?	All	All
Does it consume protocols?	Yes	Yes
Does it produce protocols?	Yes	No
What or who typically drives it?	The System	The User
What is its purpose?	Supports hardware	Performs a task

- UEFI Protocol
  - block of function pointers and data structures
  - a protocol is an interface
  - produced by a UEFI driver

DRIVER DESIGN PROCESS

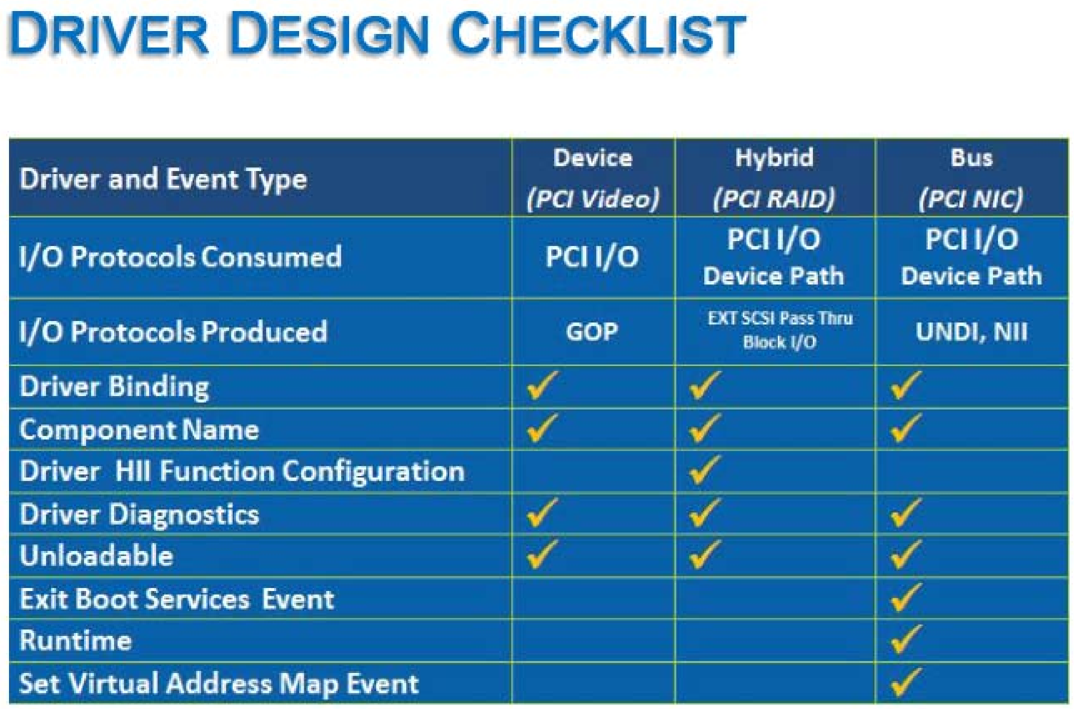
- 
Identify target platforms
6
Identify additional driver features
5
Identify UEFI driver model protocols
4

- 
DRIVER DESIGN PROCESS
Identify target platforms
6
Identify additional driver features
5
Identify UEFI driver model protocols
4
Identify produced I/O protocols
3
Identify consumed I/O protocols
2
Determine the driver type
1
- UEFI Driver Types
  - Images = Drivers and Applications
    - OS Loader is just a UEFI Application that calls ExitBootServices and never returns
  - Service Drivers
    - do not manage HW
    - provides services to other drivers
    - do not support driver binding protocol
  - Initializing Drivers
    - make contact with HW
    - perform one-time initialization operations
    - do not create handles
    - do not produce protocols
    - unload when done
  - Root Bridge Drivers
    - manage part of core chipset
    - direct contact w/HW
    - create one or more root bridge handles
    - produces root bridge I/O protocols
  - UEFI Drivers
    - bus drivers
    - hybrid drivers
    - device drivers

- 

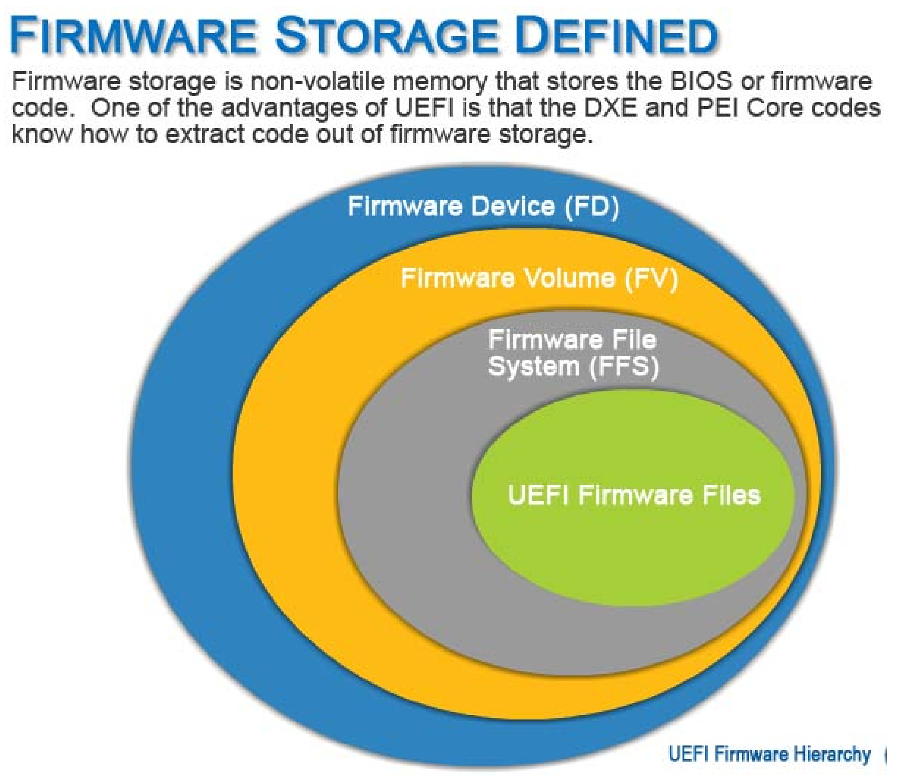
WRITING UEFI DRIVERS

Necessary Protocols	Recommended Protocols
• Initialization	• Component Name
• Binding	• HII Configuration
• *Supported*	• Diagnostic
• *Start*	• Unload
• *Stop*	

- Driver guidelines
  - don't touch hardware in driver entry
  - keep Supported() small and simple
  - move complex I/O into Start() and Stop()
  - Start() and Stop() should mirror each other
- 

DRIVER DESIGN CHECKLIST

Driver and Event Type	Device *(PCI Video)*	Hybrid *(PCI RAID)*	Bus *(PCI NIC)*
I/O Protocols Consumed	PCI I/O	PCI I/O Device Path	PCI I/O Device Path
I/O Protocols Produced	GOP	EXT SCSI Pass Thru Block I/O	UNDI, NII
Driver Binding	✓	✓	✓
Component Name	✓	✓	✓
Driver HII Function Configuration		✓	
Driver Diagnostics	✓	✓	✓
Unloadable	✓	✓	✓
Exit Boot Services Event			✓
Runtime			✓
Set Virtual Address Map Event			✓

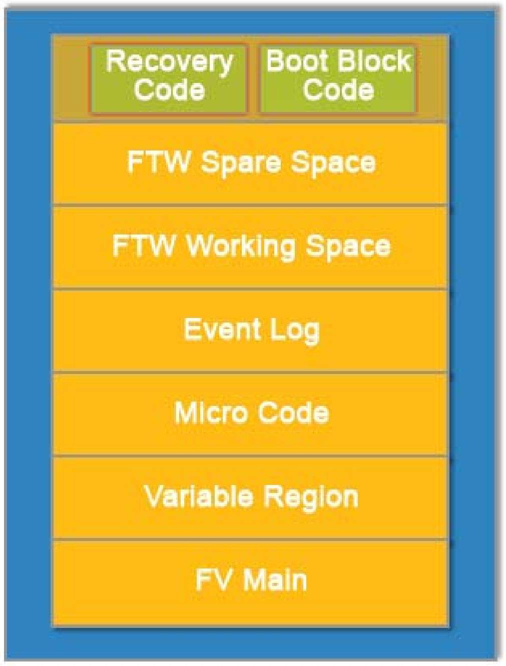
- 

  FIRMWARE STORAGE DEFINED

  Firmware storage is non-volatile memory that stores the BIOS or firmware code. One of the advantages of UEFI is that the DXE and PEI Core codes know how to extract code out of firmware storage.

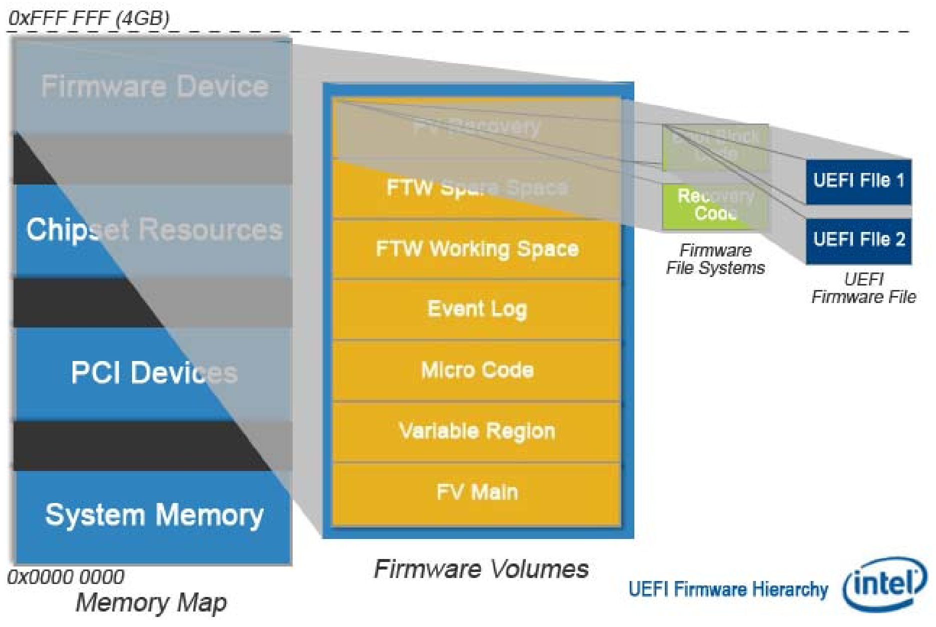
- Firmware File
  - code and data in firmware volume
  - attributes
    - name, type, alignment, size
  - smallest and least modular code
  - UEFI build tools create the firmware files
- Firmware File System
  - describes the organization of files and free space within the Firmware Volume
  - each FFS has a unique GUID
  - based on FAT32
  - UEFI build tools create the firmware file systems
- Firmware Volume
  - logical firmware device -- logical representation of a physical storage device
  - a flash part, or group of flash parts; even over the network
  - later part of DXE can use firmware protocol to access other (off system) FVs
  - UEFI build tools create the firmware volume
- Firmware Device
  - non-volatile storage component
  - divided into various areas like code, data, fault tolerant area

Inside Firmware Devices

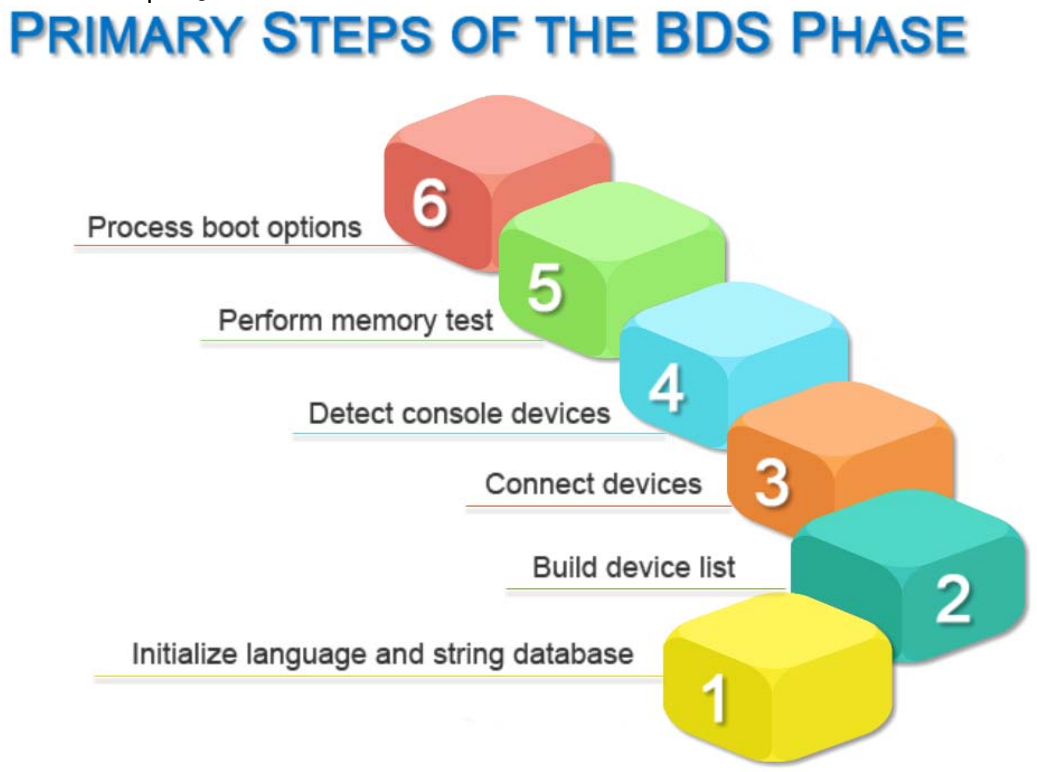
- 

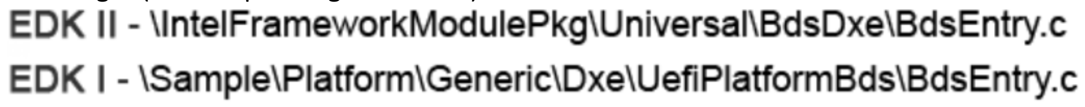
- PEI images are "terse images" as contrasted to DXE
    - the headers are different--much smaller on PEI

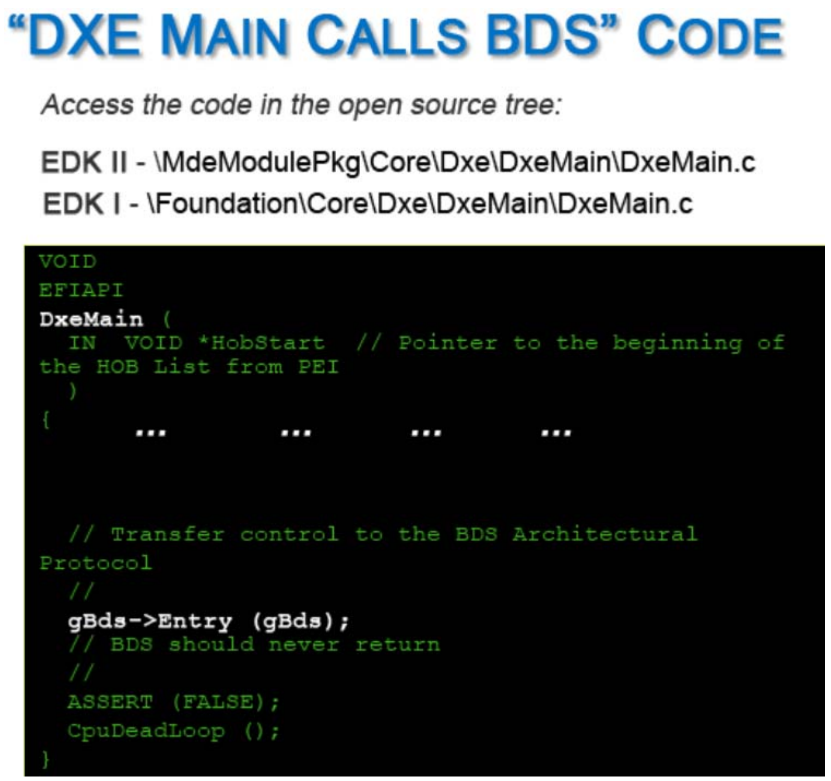
Flash Devices' Physical Memory Map

- 

Lesson_5_BDS_HII

- BDS is the phase after DXE
  - UEFI spec §3
- 

- goals of BDS
  - centralized policy and user interface
  - boot w/minimal driver and user interaction
  - implement menu-based setup
  - central repository for platform boot policy
- if BDS fails to load the desired OS Loader, it will return control to the DXE dispatcher and ask for another boot target (&corresponding OS Loader)
- 
  EDK II - \IntelFrameworkModulePkg\Universal\BdsDxe\BdsEntry.c
  EDK I - \Sample\Platform\Generic\Dxe\UefiPlatformBds\BdsEntry.c
- BDS enumerates all possible boot devices; creates a device path for each
- Boot Manager replaces the "Legacy BIOS Boot Specification"

- 

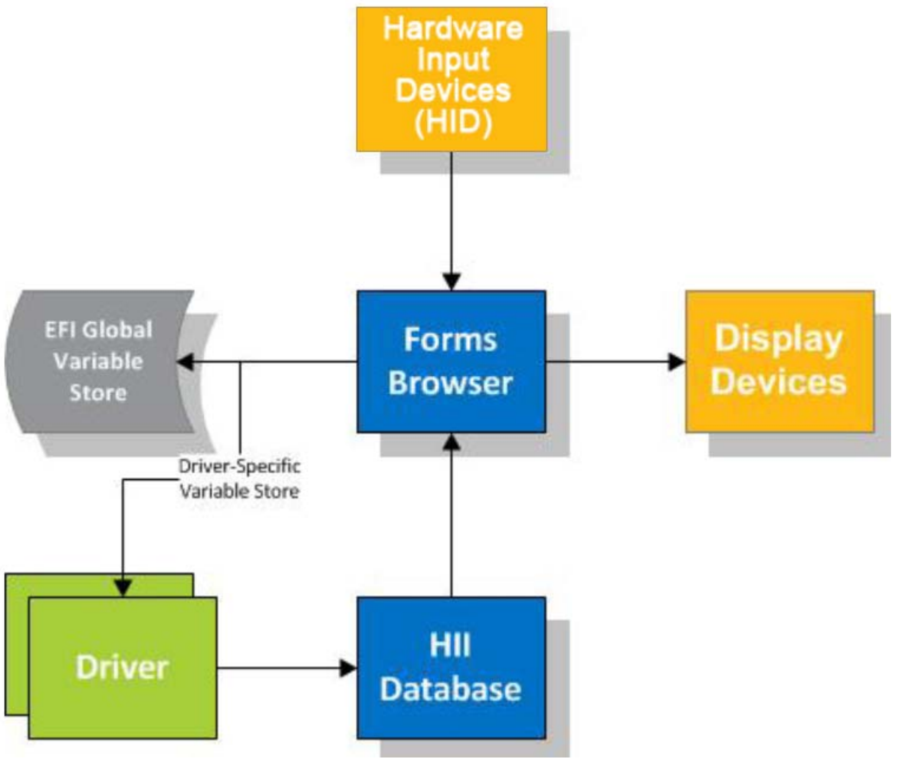
"DXE Main Calls BDS" Code

Access the code in the open source tree:

EDK II - \MdeModulePkg\Core\Dxe\DxeMain\DxeMain.c
EDK I - \Foundation\Core\Dxe\DxeMain\DxeMain.c

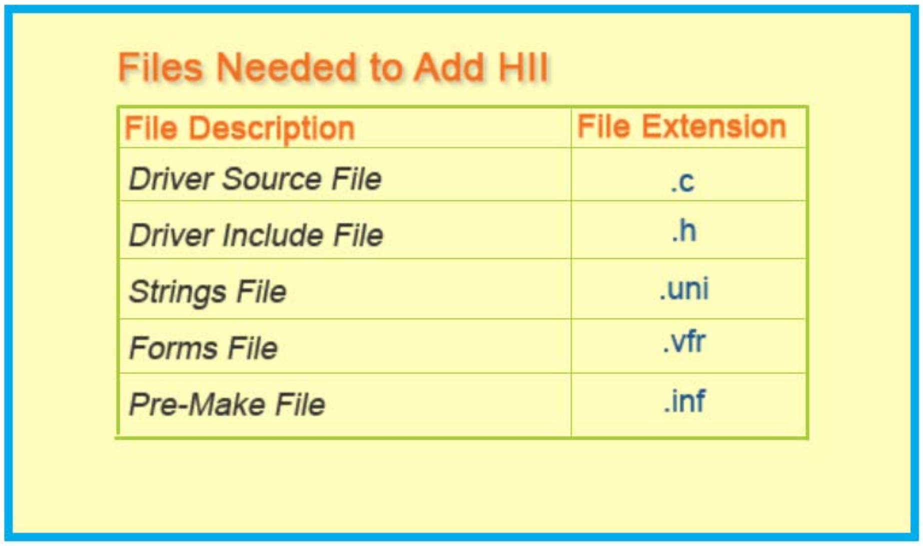
```
VOID
EFIAPI
DxeMain (
  IN  VOID *HobStart  // Pointer to the beginning of
the HOB List from PEI
  )
{
      ...       ...       ...       ...



  // Transfer control to the BDS Architectural
Protocol
  //
  gBds->Entry (gBds);
  // BDS should never return
  //
  ASSERT (FALSE);
  CpuDeadLoop ();
}
```

- ConSplitter-->ConIn-->ConOut-->gfx output
- a boot option is a device path that describes an OS loader file
- Boot Option List global defined variables
  - BootOrder
  - Boot####
  - BootNext
- what does BDS do again?
  - DXE and UEFI drivers load to support hardware
  - global vars (above) like BootOrder define boot sequence
  - inits console devices (for Setup and F12)
  - inits the boot device
  - passes control to os loader
- HII
  - provides localizable text: database, data structure, browser engine
  - motherboard and add-in cards publish to HII database
  - if user selects setup, HII Browser takes over and displays forms and strings from database
  - can use same menu for both option ROMs and Setup
  - interface is global--each driver not have to write their own
- (green=drivers)(gray=NVRAM)(blue=HII)(yellow=user interfaces)

- 
Hardware Input Devices (HID)
EFI Global Variable Store
Forms Browser
Display Devices
Driver-Specific Variable Store
Driver
HII Database
- inside the HII Database is the HII Browser Engine
    - fonts, strings, forms, and images
- forms
    - menus on the user interface
    - similar to an application in that it reads data and interacts w/user
- package
    - data structure that contains fonts/strings/forms from a driver(s) that gets publlished to the HII database
- strings
    - sets of characters, in unicode
    - tokenized--each string represented by a token
    - C code references the token
- fonts
    - UEFI has a standard font
    - you can add additional fonts
- keyboard -- uefi supports multiple keyboard mappings
- VFR -- visual forms representation
    - part of EDKII, not UEFI
    - like HTML
- IFR -- internal forms representation
    - defined by UEFI
    - binary code
    - EDK compiles VFR source in IFR

MINIMUM FILES FOR HII

There are several files you'll need to include in order to add HII to your driver and/or application:

- 

Files Needed to Add HII

File Description	File Extension
Driver Source File	.c
Driver Include File	.h
Strings File	.uni
Forms File	.vfr
Pre-Make File	.inf