tianocore

UEFI & EDK II TRAINING

EDK II Debugging through UEFI Boot Flow

tianocore.org

LESSON OBJECTIVE

- Debugging commands similar to all debuggers
- Debugging UEFI Platform Initialization Boot Flow

tianocore

DEBUGGING COMMANDS

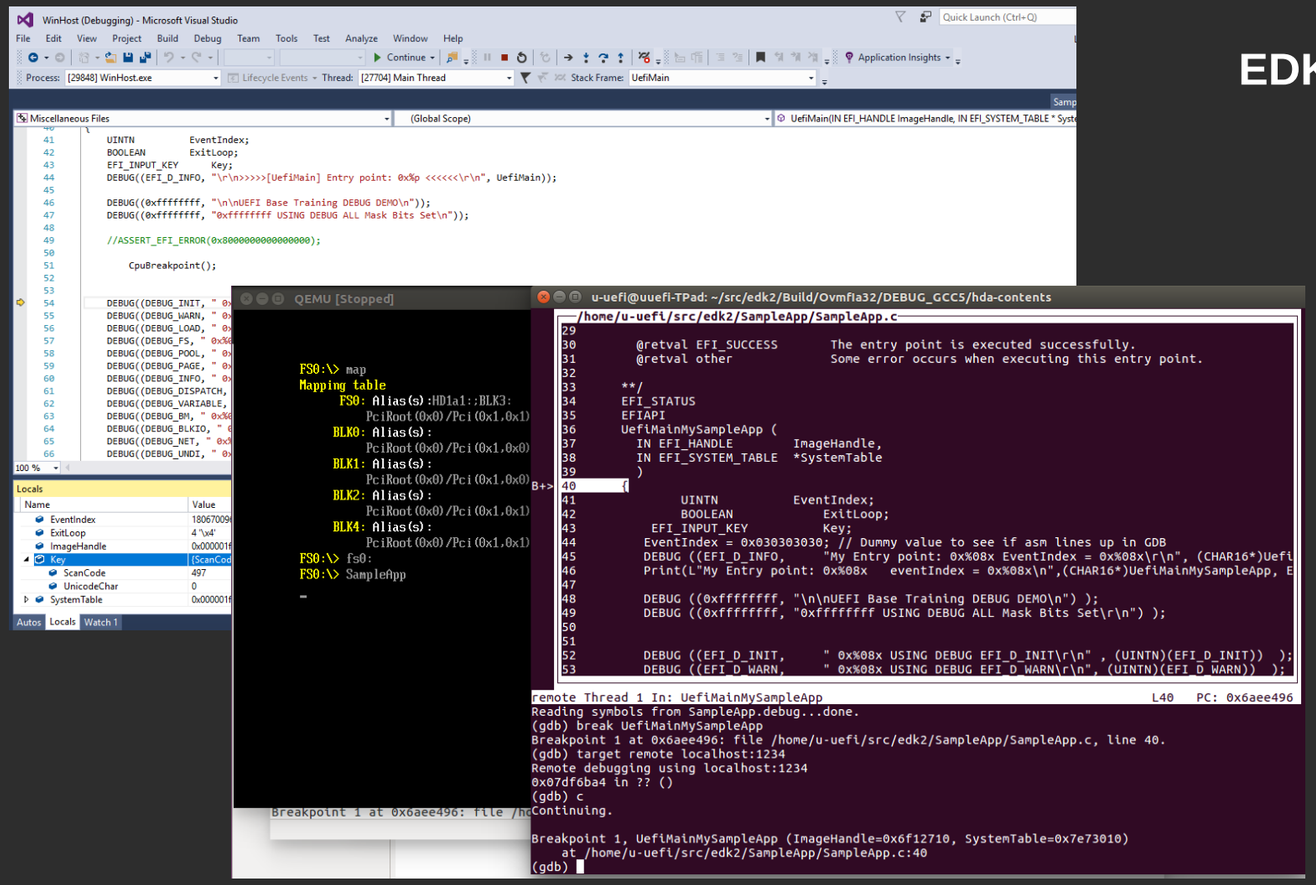
`CpuBreakpoint()` | `CpuDeadLoop()` added to Source

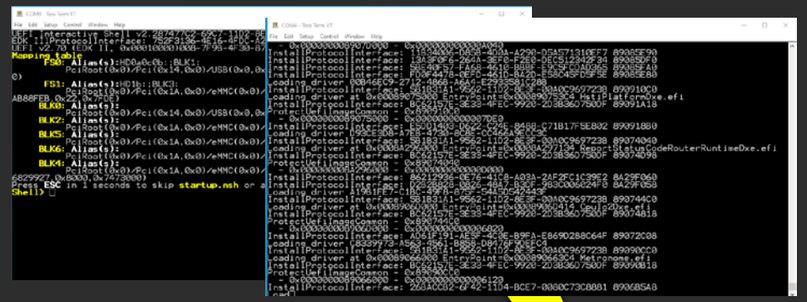
tianocore

UEFI Debug Methods

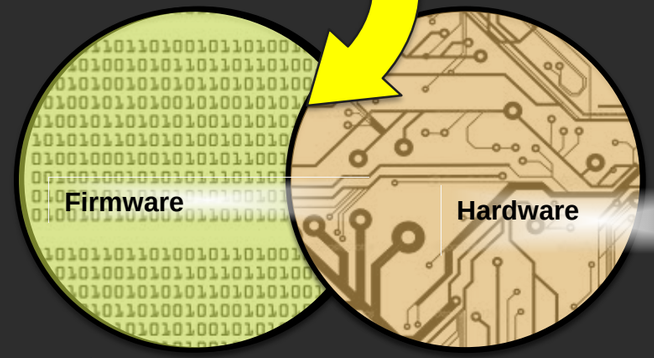
GUI debugger

Software/hardware debuggers

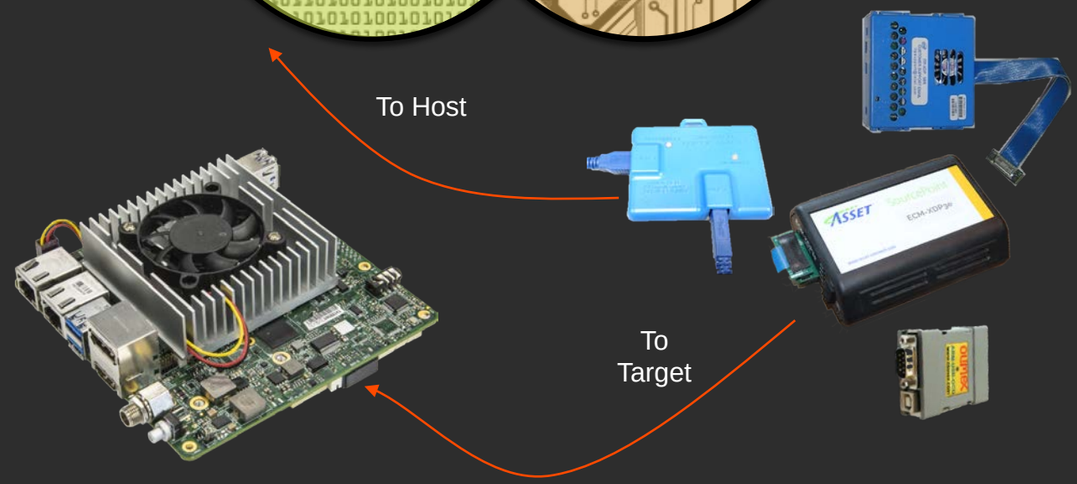




EDK II Debugging



Hardware Debuggers



Source Level Debugging

View call stack

Go

Insert CpuBreakpoint()

View and edit local/global variables

Set breakpoint

Step into/over routines

Go till

View disassembled code

View/edit general purpose register values

CpuBreakpoint Vs. CpuDeadLoop

CpuBreakpoint

When using a Software debugger:

- Visual Studio
- GDB (ovmf with qemu)
- Intel® UDK Debugger
- Debug agent –`SourceLevelDebugPkg`

CpuDeadLoop

When using a Hardware debugger:

- In-Target Probe (ITP)
- Intel® SVT DCI cable
- Intel® SVT Closed Chassis Adapter (CCA)
- Intel® Simics® Simulator
- other 3rd Party Hardware (i.e. Lauterbach w/ JTAG)

The functions `CpuBreakpoint()` and `CpuDeadLoop()` are part of the EDK II Base Libraries and can be compiled with any UEFI or PI Module at any phase of the boot flow (SEC, PEI, DXE, BDS, TSL)

Special DCI Breakpoint with HW Debugger

`CpuIceBreakpoint`

The Intel Architecture has a special op-code for a breakpoint: `int1`
Better than a `CpuDeadLoop()` since it halts the processor. Better trace information
Downside:

- Requires a Hardware Debugger with DCI capabilities to intercept the int1 op code
- There is no “C” equivalent – needs to be assembly code

Example of this code is the downloaded Lab Material :

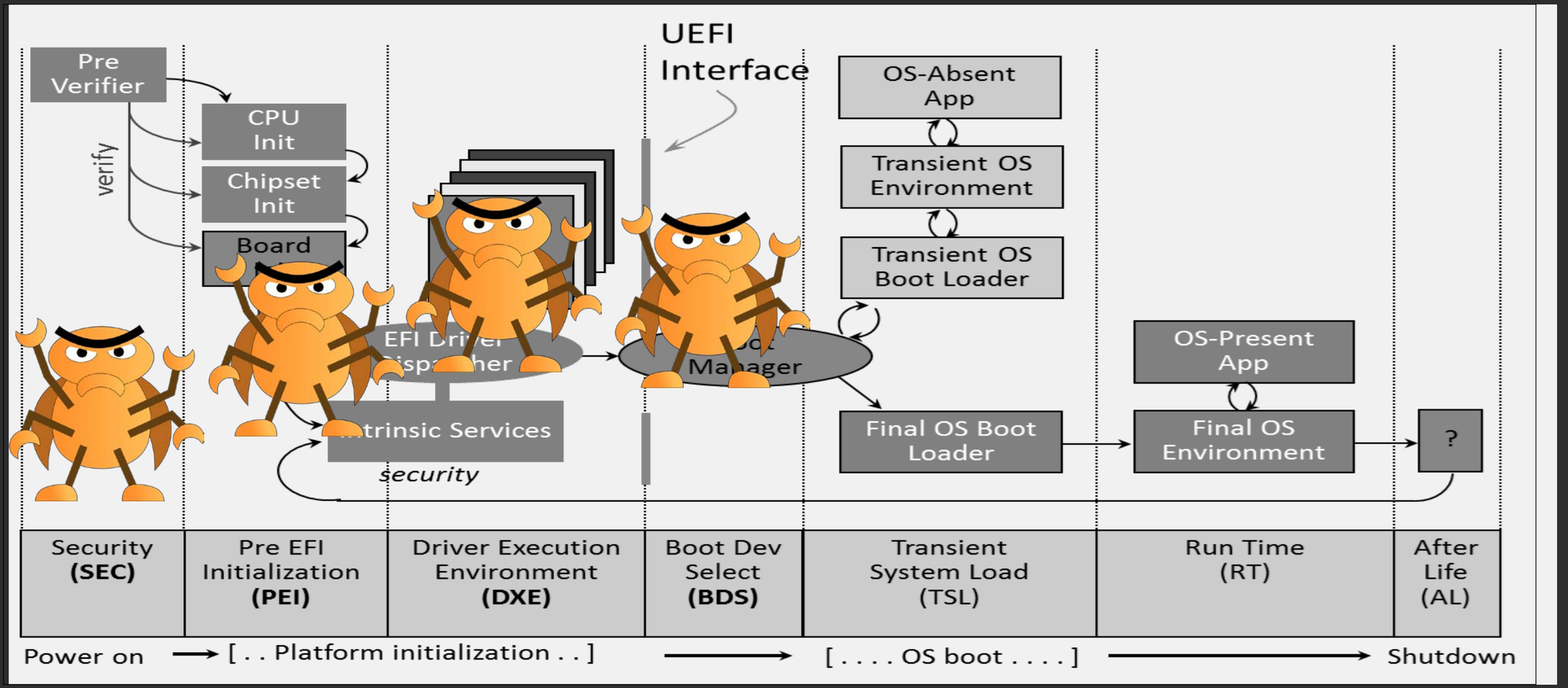
`. . . /LabSampleCode/CpuIceBreakpoint_Code`

tianocore

DEBUGGING THRU BOOT FLOW

Add Breakpoints to the Compiled BIOS / Firmware Source Code

Debugging the Boot Phases



PEI & DXE Code Modules

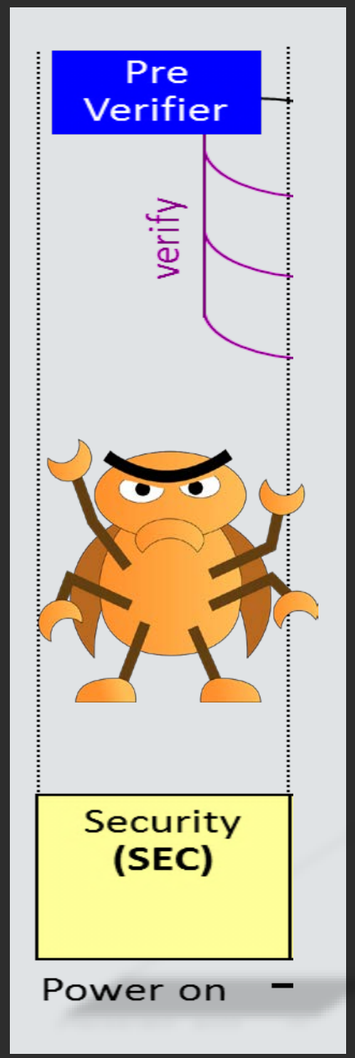
\Edk2\OvmfPkg\OvmfPkgX64.fdf

```
# SEC FV
INF  OvmfPkg/Sec/SecMain.inf
INF  RuleOverride=RESET_VECTOR OvmfPkg/ResetVector/ResetVector.inf
# - - - - - - - - - -
#  PEI FV  Phase modules
#
INF  MdeModulePkg/Core/Pei/PeiMain.inf
INF  MdeModulePkg/Universal/PCD/Pei/Pcd.inf
INF  MdeModulePkg/Universal/ReportStatusCodeRouter/Pei/ReportStatusCodeRouterPei.inf
INF  MdeModulePkg/Universal/StatusCodeHandler/Pei/StatusCodeHandlerPei.inf
INF  OvmfPkg/PlatformPei/PlatformPei.inf
INF  MdeModulePkg/Core/DxeIplPeim/DxeIpl.inf
#- - - - - - - - - -
# DXE FV  Phase modules
#
INF  MdeModulePkg/Core/Dxe/DxeMain.inf
INF  MdeModulePkg/Universal/ReportStatusCodeRouter/RuntimeDxe/ReportStatusCodeRouterRuntimeDxe.inf
INF  MdeModulePkg/Universal/StatusCodeHandler/RuntimeDxe/StatusCodeHandlerRuntimeDxe.inf
INF  MdeModulePkg/Universal/PCD/Dxe/Pcd.inf
INF  MdeModulePkg/Core/RuntimeDxe/RuntimeDxe.inf
INF  MdeModulePkg/Universal/SecurityStubDxe/SecurityStubDxe.inf
INF  MdeModulePkg/Universal/EbcDxe/EbcDxe.inf
INF  OvmfPkg/8259InterruptControllerDxe/8259.inf
INF  UefiCpuPkg/CpuIo2Dxe/CpuIo2Dxe.inf
INF  UefiCpuPkg/CpuDxe/CpuDxe.inf
INF  OvmfPkg/8254TimerDxe/8254Timer.inf
INF  OvmfPkg/IncompatiblePciDeviceSupportDxe/IncompatiblePciDeviceSupport.inf
INF  OvmfPkg/PciHotPlugInitDxe/PciHotPlugInit.inf
INF  MdeModulePkg/Bus/Pci/PciHostBridgeDxe/PciHostBridgeDxe.inf
INF  MdeModulePkg/Bus/Pci/PciBusDxe/PciBusDxe.inf
INF  MdeModulePkg/Universal/ResetSystemRuntimeDxe/ResetSystemRuntimeDxe.inf
# . . .
# BDS Phase modules - part of DXE FV
#
INF  MdeModulePkg/Universal/BdsDxe/BdsDxe.inf
INF  MdeModulePkg/Application/UiApp/UiApp.inf
# . . .
```

Reset Vector @ `0xfffffff0`

Check code root order from Platform .fdf

Debugging the Boot Phases - SEC



Debugging Sec Phase

Hardware debugger capable only

- Break at the Reset Vector
- Check temporary memory – CAR NEM
- Enable the “C” Code
- Transfer control to PEI

SEC flow in code level - OVMF

OvmfPkg/ResetVector/ResetVector.inf

\Edk2\OvmfPkg\ResetVector\ResetVector.nasmb

```
    ;
    ; Search for the Boot Firmware Volume (BFV)
    ;
    OneTimeCall Flat32SearchForBfvBase

    ;
    ; EBP - Start of BFV
    ;

    ;
    ; Search for the SEC entry point
    ;
    OneTimeCall Flat32SearchForSecEntryPoint

    ;
    ; ESI - SEC Core entry point
    ; EBP - Start of BFV
    ;

%ifdef ARCH_IA32

    ;
    ; Restore initial EAX value into the EAX register
    ;
    mov     eax, esp

    ;
    ; Jump to the 32-bit SEC entry point
    ;
    jmp     esi
```

OvmfPkg/Sec/SecMain.inf

Main.asm

Edk2\OvmfPkg\ResetVector\Ia16\ResetVectorVtf0.asm

```
; VTF-0 means that the VTF (Volume Top File) code does not require
; any fixups.
;
vtfSignature:
    DB      'V', 'T', 'F', 0

ALIGN   16

resetVector:
;
; Reset Vector
;
; This is where the processor will begin execution
;
; In IA32 we follow the standard reset vector flow. While in X64, Td guest
; may be supported. Td guest requires the startup mode to be 32-bit
; protected mode but the legacy VM startup mode is 16-bit real mode.
; To make NASM generate such shared entry code that behaves correctly in
; both 16-bit and 32-bit mode, more BITS directives are added.
;
%ifdef ARCH_IA32
    nop
    nop
    jmp     EarlyBspInitReal16

%else

    mov     eax, cr0
    test    al, 1
    jz      .Real
BITS 32
    jmp     Main32
BITS 16
.Real:
    jmp     EarlyBspInitReal16

%endif

ALIGN   16

fourGigabytes:
```

Int16.asm

Main.asm

SEC flow in code level - OVMF

Main.asm

OvmfPkg/Sec/SecMain.inf

\Edk2\Edk2\OvmfPkg\Sec\SecMain.c

```
VOID
EFIAPI
SecCoreStartupWithStack (
  IN EFI_FIRMWARE_VOLUME_HEADER       *BootFv,
  IN VOID                             *TopOfCurrentStack
  )
{
  EFI_SEC_PEI_HAND_OFF        SecCoreData;
  SEC_IDT_TABLE               IdtTableInStack;
  IA32_DESCRIPTOR             IdtDescriptor;
  UINT32                      Index;
  volatile UINT8              *Table;

  //
  // To ensure SMM can't be compromised on S3 resume, we must force re-init of
  // the BaseExtractGuidedSectionLib. Since this is before library contructors
  // are called, we must use a loop rather than SetMem.
  //
  Table = (UINT8*)(UINTN)FixedPcdGet64 (PcdGuidedExtractHandlerTableAddress);
  for (Index = 0;
       Index < FixedPcdGet32 (PcdGuidedExtractHandlerTableSize);
       ++Index) {
    Table[Index] = 0;
  }
```

```
VOID
EFIAPI
SecStartupPhase2(
  IN VOID                     *Context
  )
{
  (*PeiCoreEntryPoint) (SecCoreData, (EFI_PEI_PPI_DESCRIPTOR *)&mPrivateDispatchTable);

  //
  // If we get here then the PEI Core returned, which is not recoverable.
  //
  ASSERT (FALSE);
```

\Edk2\Edk2\OvmfPkg\Sec\Ia32\SecEntry.nasm

```
global ASM_PFX( ModuleEntryPoint)
ASM_PFX( ModuleEntryPoint):

    ;
    ; Fill the temporary RAM with the initial stack value.
    ; The loop below will seed the heap as well, but that's harmless.
    ;
    mov     eax, FixedPcdGet32 (PcdInitValueInTempStack)       ; dword to store
    mov     edi, FixedPcdGet32 (PcdOvmfSecPeiTempRamBase)      ; base address,
                                                               ;   relative to
                                                               ;   ES
    mov     ecx, FixedPcdGet32 (PcdOvmfSecPeiTempRamSize) / 4 ; dword count
    cld                                                        ; store from base
                                                               ;   up
    rep stosd

    ;
    ; Load temporary RAM stack based on PCDs
    ;
    %define SEC_TOP_OF_STACK (FixedPcdGet32 (PcdOvmfSecPeiTempRamBase) + \
                              FixedPcdGet32 (PcdOvmfSecPeiTempRamSize))
    mov     eax, SEC_TOP_OF_STACK
    mov     esp, eax
    nop

    ;
    ; Setup parameters and call SecCoreStartupWithStack
    ;   [esp]   return address for call
    ;   [esp+4] BootFirmwareVolumePtr
    ;   [esp+8] TopOfCurrentStack
    ;
    push    eax
    push    ebp
    call    ASM_PFX(SecCoreStartupWithStack)
```

SEC flow in code level - HW

UefiCpuPkg/SecCore/SecCore.inf

edk2-Platforms\Platform\Intel\QuarkPlatformPkg\Library\PlatformSecLib\PlatformSecLib.c

The entry point function is
_ModuleEntryPoint in PlatformSecLib

One example/Instance \edk2-Platforms\Platform\Intel\QuarkPlatformPkg\Library\PlatformSecLib\PlatformSecLib.inf -Flat32.asm

```
VOID
EFIAPI
PlatformSecLibStartup (
  VOID
  )
{
  //
  // Process all library constructor functions linked to SecCore.
  // This function must be called before any library functions are called
  //
  ProcessLibraryConstructorList ();

  //
  // Set write back cache attribute for SPI FLASH
  //
  MtrrSetMemoryAttribute (
    PcdGet32 (PcdFlashAreaBaseAddress),
    PcdGet32 (PcdFlashAreaSize),
    CacheWriteBack
    );

  //
  // Set write back cache attribute for 512KB Embedded SRAM
  //
  MtrrSetMemoryAttribute (
    PcdGet32 (PcdEsramStage1Base),
    SIZE_512KB,
    CacheWriteBack
    );

  //
  // Pass control to SecCore module passing in the size of the temporary RAM in
  // Embedded SRAM, the base address of the temporary RAM in Embedded SRAM, and
  // the base address of the boot firmware volume.  The top 32KB of the 512 KB
  // embedded SRAM are used as temporary RAM.
  //
  SecStartup (
    SIZE_32KB,
    PcdGet32 (PcdEsramStage1Base) + SIZE_512KB - SIZE_32KB,
    (VOID *)(UINTN)PcdGet32 (PcdFlashFvRecoveryBase)
    );
}
```

```
_ModuleEntryPoint       PROC C PUBLIC
```

```
ProtectedModeEntryPoint PROC NEAR C PUBLIC

  JMP32  stackless_EarlyPlatformInit

  ;
  ; Set up stack pointer
  ;
  mov     esp, PcdGet32(PcdEsramStage1Base)
  mov     esi, QUARK_ESRAM_MEM_SIZE_BYTES
  add     esp, esi                          ; ESP = top of stack (stack grows downwards)

  ;
  ; Store the the BIST value in EBP
  ;
  mov     ebp, 00h          ; No processor BIST on Quark
```

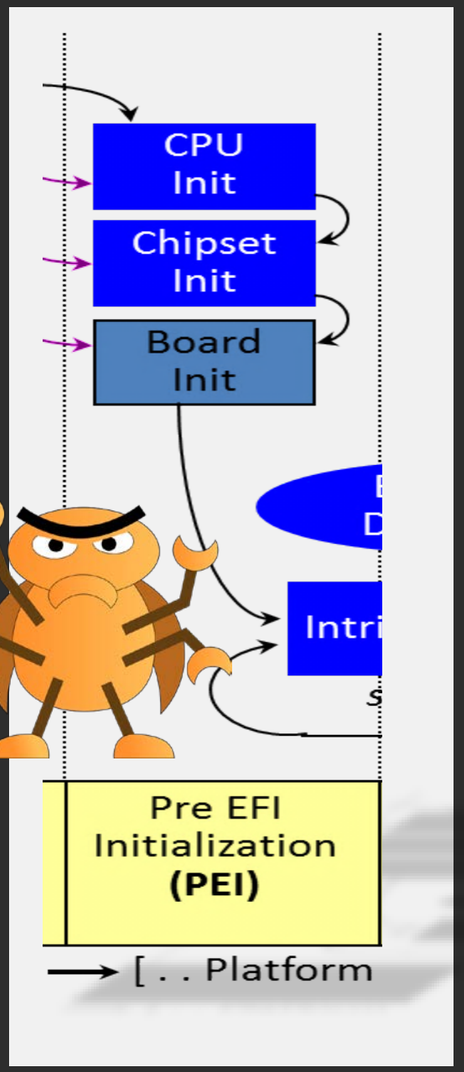
```
PushBist:
  push    ebp
  loop    PushBist

  ;
  ; Pass Control into the PEI Core
  ;
  call PlatformSecLibStartup
```

\Edk2\Edk2\UefiCpuPkg\SecCore\SecMain.c

Hint: SEC Entrypoint on real HW - FSP path:
\Edk2\IntelFsp2Pkg\FspSecCore\Ia32\FspApiEntryT.nasm

Debugging the Boot Phases - PEI



- Use debugger prior to PEI Main
- Check proper execution of PEI drivers
- Execute basic chipset & Memory init.
- Check memory availability
- Complete flash accessibility
- Execute recovery driver
- Detect DXE IPL

PEI core flow in code level

MdeModulePkg\Core\Pei\PeiMain.inf

MdeModulePkg/Core/Pei/PeiMain/PeiMain.c

```
VOID
EFIAPI
PeiCore (
  IN CONST EFI_SEC_PEI_HAND_OFF        *SecCoreDataPtr,
  IN CONST EFI_PEI_PPI_DESCRIPTOR      *PpiList,
  IN VOID                              *Data
  )
{
  SetPeiServicesTablePointer ((CONST EFI_PEI_SERVICES **)&PrivateData.Ps);
  InitializeMemoryServices   (&PrivateData, SecCoreData, OldCoreData);
  InitializeSecurityServices (&PrivateData.Ps, OldCoreData);
  InitializeDispatcherData   (&PrivateData,    OldCoreData, SecCoreData);
  InitializeImageServices    (&PrivateData,    OldCoreData);
…
  Status = PeiServicesInstallPpi (&mMemoryDiscoveredPpi);
…
  PeiDispatcher (SecCoreData, &PrivateData);
…
}
```

PEI Phase: Trace Each PEIM

There is a loop function in :

MdeModulePkg/Core/Pei/Dispatcher/Dispatcher.c

Add CpuBreakpoint(); before launching each PEIM

```
VOID
PeiDispatcher (
  IN CONST EFI_SEC_PEI_HAND_OFF  *SecCoreData,
  IN PEI_CORE_INSTANCE           *Private
  )
{ // ...
        // Call the PEIM entry point
        //
        PeimEntryPoint = (EFI_PEIM_ENTRY_POINT2)(UINTN)EntryPoint;
        PERF_START (PeimFileHandle, "PEIM", NULL, 0);
// Add a call to CpuBreakpoint();  approx. line 1460
        CpuBreakpoint();
        PeimEntryPoint(PeimFileHandle, (const EFI_PEI_SERVICES **) &Private->Ps);
```

PEI services

```
EFI_PEI_SERVICES  gPs = {
…
  PeiInstallPpi,
  PeiLocatePpi,
  PeiGetBootMode,
  PeiSetBootMode,

  PeiGetHobList,

  PeiInstallPeiMemory,
  PeiAllocatePages,
  PeiAllocatePool,
  (EFI_PEI_COPY_MEM)CopyMem,
  (EFI_PEI_SET_MEM)SetMem,

  PeiReportStatusCode,
  PeiResetSystem,

  …
,
  PeiFreePages,
};
```

MdePkg\Include\Pi\PiPeiCis.h

Call PEI services using PeiServices ptr

```
Status = (*PeiServices)->GetBootMode(
                                 PeiServices,
                                 &BootMode
                                 );
```

To call PEI services viaPeiServicesLib

```
Status = PeiServicesGetBootMode(
                                 &BootMode
                                 );
```

Example for PEI drivers

```
Loading PEIM 9B3ADA4F-AE56-4C24-8DEA-F03B7558AE50
Loading PEIM at 0x0000082BFC0 EntryPoint=0x0000082F40A PcdPeim.efi
…
Loading PEIM A3610442-E69F-4DF3-82CA-2360C4031A23
Loading PEIM at 0x00000831140 EntryPoint=0x00000832594 ReportStatusCodeRouterPei.efi
…
Loading PEIM at 0x00000833140 EntryPoint=0x00000834404 StatusCodeHandlerPei.efi
Loading PEIM 222C386D-5ABC-4FB4-B124-FBB82488ACF4
Loading PEIM at 0x000008350C0 EntryPoint=0x0000083A74E PlatformPei.efi
…
PeiInstallPeiMemory MemoryBegin 0x3F36000, MemoryLength 0x4042000
// RELOCATION HAPPEN HERE
Loading PEIM at 0x00007EE8000 EntryPoint=0x00007EF0DAC PeiCore.efi
…
Loading PEIM at 0x00007EE2000 EntryPoint=0x00007EE544A PcdPeim.efi
…
Loading PEIM at 0x00007EDD000 EntryPoint=0x00007EE023A DxeIpl.efi
…
Loading PEIM at 0x00007ED9000 EntryPoint=0x00007EDB523 S3Resume2Pei.efi
..
Loading PEIM at 0x00007ECD000 EntryPoint=0x00007ED563E CpuMpPei.efi
```

Check for transition from PEI to DXE

Critical point before calling DXE in:

MdeModulePkg/Core/Pei/PeiMain.c

Add CpuBreakpoint(); before entering DxeIpl

```
VOID
EFIAPI
PeiCore (
  IN CONST EFI_SEC_PEI_HAND_OFF        *SecCoreDataPtr,
  IN CONST EFI_PEI_PPI_DESCRIPTOR      *PpiList,
  IN VOID                              *Data
  )
{ // ...
  // Enter DxeIpl to load Dxe core.
  //
  DEBUG ((DEBUG_INFO, "DXE IPL Entry\n"));
// Add a call to CpuBreakpoint();  approx. line 512
  CpuBreakpoint();
  Status = TempPtr.DxeIpl->Entry (
                             TempPtr.DxeIpl,
                             &PrivateData.Ps,
                             PrivateData.HobList
```

Check for transition from DxeIpl to DXE

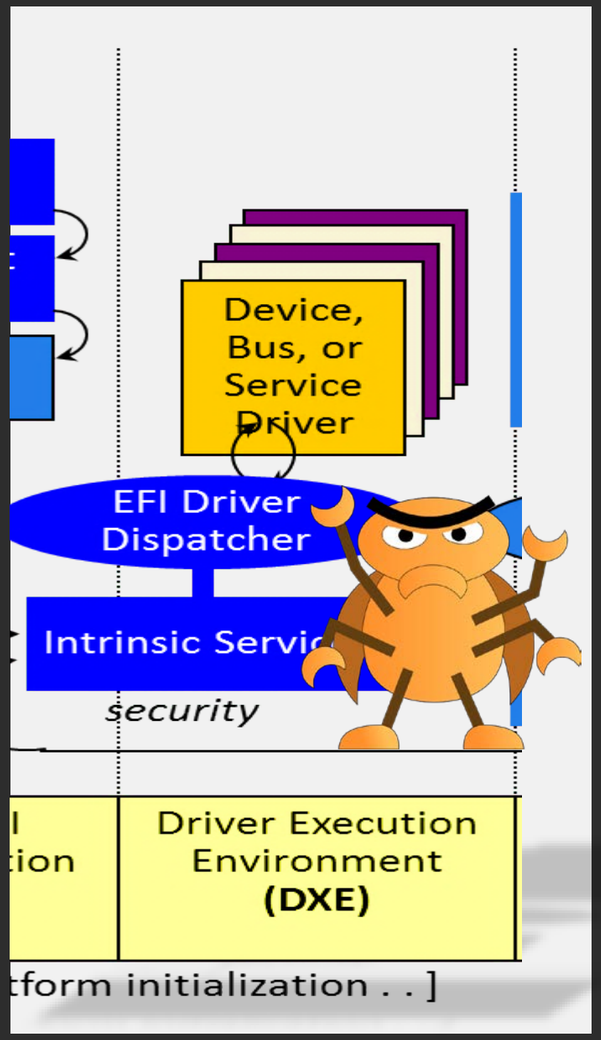
Critical point before calling DXE Core in:

MdeModulePkg/Core/DxeIplPeim/DxeLoad.c

Before entering Dxe Core (Notice also this is a standalone module - DxeIpl.efi)

```
EFI_STATUS
EFIAPI
DxeLoadCore (
  IN CONST EFI_DXE_IPL_PPI *This,
  IN EFI_PEI_SERVICES      **PeiServices,
  IN EFI_PEI_HOB_POINTERS  HobList
  )
{ // ...
  // Transfer control to the DXE Core
  // The hand off state is simply a pointer to the HOB list
  //
// Add a call to CpuBreakpoint();  approx. line 448
  CpuBreakpoint();
  HandOffToDxeCore (DxeCoreEntryPoint, HobList);
  //
  // If we get here, then the DXE Core returned.  This is an error
```

Debugging the Boot Phases - DXE



- Search for cyclic dependency check
- Trace ASSERTs caused during DXE execution
- Debug individual DXE drivers
- Check for architectural protocol failure
- Ensure BDS entry call

DXE core flow in code level

MdeModulePkg/Core/Dxe/DxeMain.inf

MdeModulePkg/Core/Dxe/DxeMain/DxeMain.c

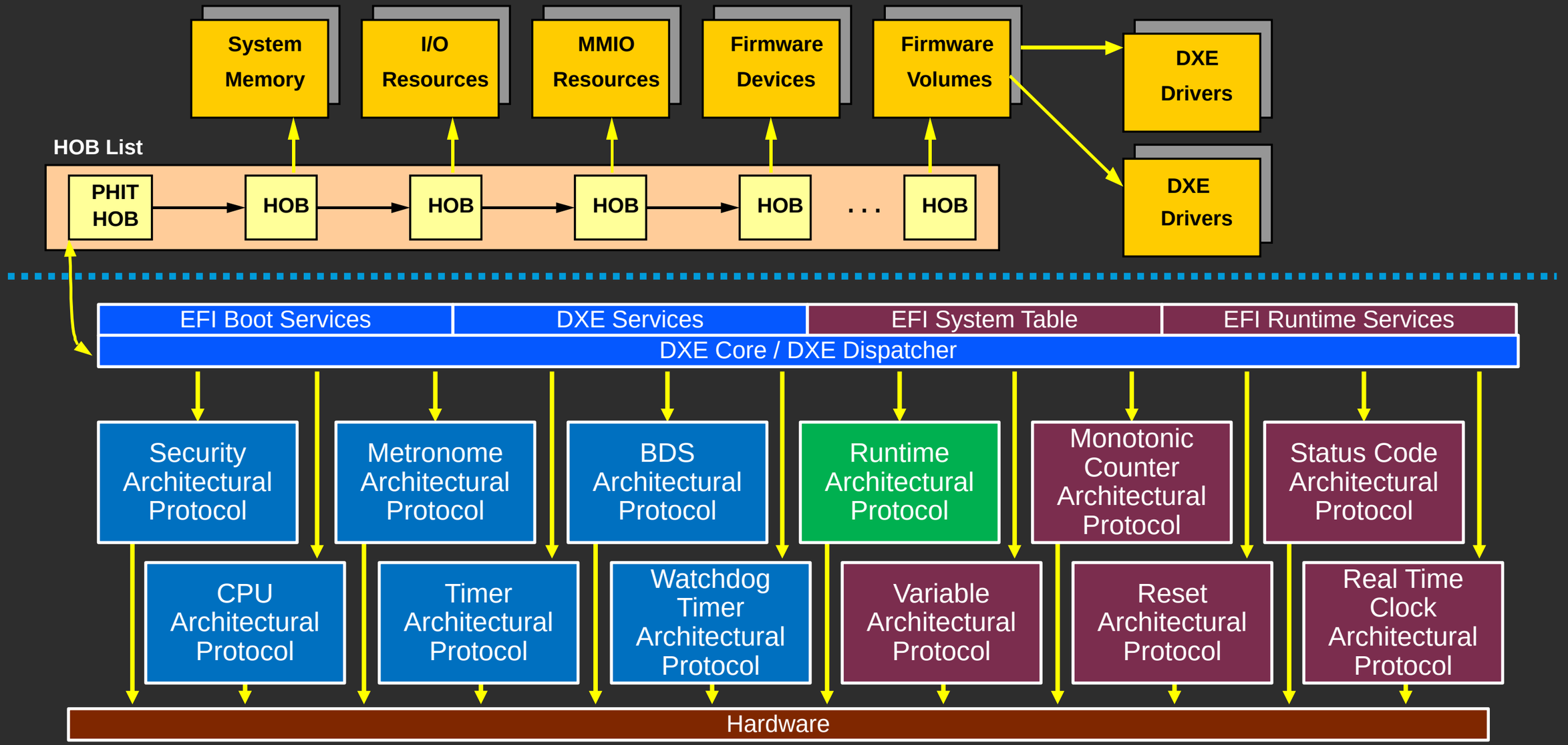
```
VOID
EFIAPI
DxeMain (
  IN  VOID *HobStart
  )
{…
   CoreInitializeMemoryServices (&HobStart, &MemoryBaseAddress, &MemoryLength);
   gDxeCoreST->RuntimeServices = gDxeCoreRT;
 …
   FwVolBlockDriverInit (gDxeCoreImageHandle, gDxeCoreST);
 …
   CoreDispatcher ();

   Status = CoreAllEfiServicesAvailable ();
   gBds->Entry (gBds);
};
```

HOB list to transfer info from PEI to DXE

EFI_RESOURCE_SYSTEM_MEMORY HOB from PEI

DXE Core Block Diagram



See § 12 of PI Spec Vol. 2

DXE Arch protocols in code

What are Architectural Protocols?

- Typically functions that isolate platform specific hardware (e.g., real-time clock)
- Provide support for boot services and runtime services

Architectural Protocols GUID	Code location	Usage
gEfiBdsArchProtocolGuid	Edk2\MdeModulePkg\Universal\BdsDxe\BdsDxe.inf	gBds->Entry()
gEfiStatusCodeRuntimeProtocolGuid	Edk2\MdeModulePkg\Universal\ReportStatusCodeRouter\RuntimeDxe	ReportStatusCode()
gEfiRuntimeArchProtocolGuid	Edk2\MdeModulePkg\Core\RuntimeDxe	gRuntime->EventHead gRuntime->ImageHead
gEfiCapsuleArchProtocolGuid	Edk2\MdeModulePkg\Universal\CapsuleRuntimeDxe	gRT->UpdateCapsule() gRT->QueryCapsuleCapabilities()
gEfiMonotonicCounterArchProtocolGuid	Edk2\MdeModulePkg\Universal\MonotonicCounterRuntimeDxe	gBS->GetNextMonotonicCount() gRT->GetNextHighMonotonicCount()
gEfiRealTimeClockArchProtocolGuid	Edk2\MdeModulePkg\Universal\PcatRealTimeClockRuntimeDxe	gRT->GetTime() gRT->SetTime()
gEfiSecurityArchProtocolGuid	Edk2\MdeModulePkg\Universal\SecurityStubDxe	gSecurity2->FileAuthentication()
gEfiVariableArchProtocolGuid	Edk2\MdeModulePkg\Universal\Variable	gRT->SetVariable()
gEfiVariableWriteArchProtocolGuid	Edk2\MdeModulePkg\Universal\Variable\RuntimeDxe	gRT->GetVariable()
gEfiWatchdogTimerArchProtocolGuid	Edk2\MdeModulePkg\Universal\WatchdogTimerDxe	gTimer->SetTimerPeriod() gTimer->GetTimerPeriod()
gEfiMetronomeArchProtocolGuid	Edk2\MdeModulePkg\Universal\MetronomeDxe	gMetronome->WaitForTick()
gEfiResetArchProtocolGuid	Edk2\MdeModulePkg\Universal\ResetSystemRuntimeDxe	gRT->ResetSystem

DXE: Trace Each Driver Load

DXE Dispatcher calls to each driver's entry point in:

MdeModulePkg/Core/Dxe/Image/Image.c

Break every time a DXE driver is loaded.

```
EFI_STATUS
EFIAPI
CoreStartImage (
  IN EFI_HANDLE  ImageHandle,
  OUT UINTN      *ExitDataSize,
  OUT CHAR16     **ExitData  OPTIONAL
  )
{ // ...
    //
    // Call the image's entry point
    //
    Image->Started = TRUE;
// Add a call to CpuBreakpoint();  approx. line 1650
    CpuBreakpoint();
    Image->Status = Image->EntryPoint (ImageHandle, Image->Info.SystemTable);
```

Example for DXE Drivers

```
Loading PEIM at 0x00007E9D000 EntryPoint=0x00007EAD476 DxeCore.efi
…
Loading driver at 0x00007596000 EntryPoint=0x0000759E016 DevicePathDxe.efi
Loading driver at 0x00007590000 EntryPoint=0x00007593B5B PcdDxe.efi
Loading driver at 0x00007AE7000 EntryPoint=0x00007AE9E3D FvbServicesRuntimeDxe.efi
…
Loading driver at 0x00007AE1000 EntryPoint=0x00007AE38DB ReportStatusCodeRouterRuntimeDxe.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 75AC898
…
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 758F040
Loading driver at 0x00007ADB000 EntryPoint=0x00007ADD599 RuntimeDxe.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 758FF18
…
InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 758F440
Loading driver at 0x000075A3000 EntryPoint=0x000075A4693 SecurityStubDxe.efi
…
Loading driver at 0x00007ACF000 EntryPoint=0x00007AD14C4 EmuVariableFvbRuntimeDxe.efi
Loading driver at 0x000070C5000 EntryPoint=0x000070D92C1 SetupBrowser.efi
Loading driver at 0x000070F3000 EntryPoint=0x000070F653D SmbiosDxe.efi
Loading driver at 0x000070EC000 EntryPoint=0x000070F0243 QemuFwCfgAcpiPlatform.efi
Loading driver at 0x000070BA000 EntryPoint=0x000070C08EE tftpDynamicCommand.efi
```

Transition from DXE to BDS

DXE call to BDS entry point in:

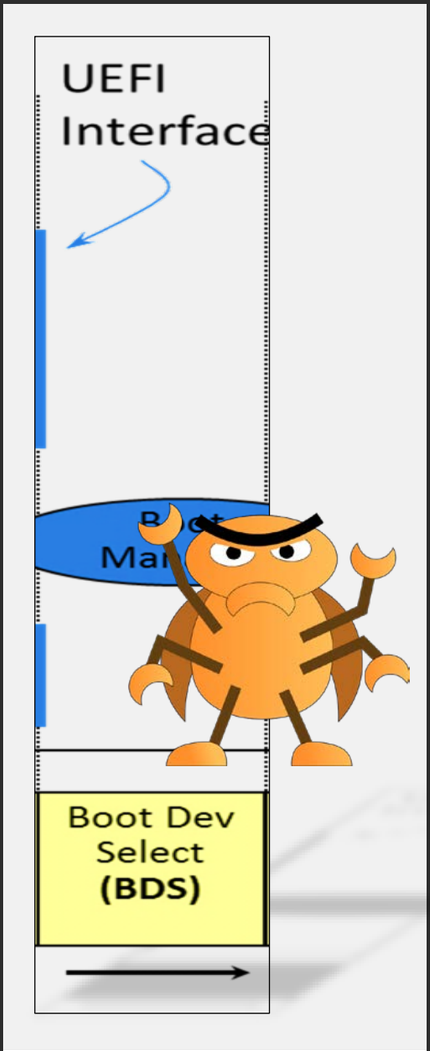
MdeModulePkg/Core/Dxe/DxeMain/DxeMain.c

Add CpuBreakpoint(); to break before BDS.

```
VOID
EFIAPI
DxeMain (
  IN  VOID *HobStart
  )
{ // ...
  // Transfer control to the BDS Architectural Protocol
  //
// Add a call to CpuBreakpoint();  approx. line 550
  CpuBreakpoint();
  gBds->Entry (gBds);

  //
  // BDS should never return
  //
  ASSERT (FALSE);
  CpuDeadLoop ();
```

Debugging the Boot Phases - BDS



- Detect console devices (input and output)
- Check enumeration of all devices' preset
- Detect boot policy
- Ensure BIOS "front page" is loaded
- Get to Boot Options

BDS Entry

Critical point before calling DXE Core in:

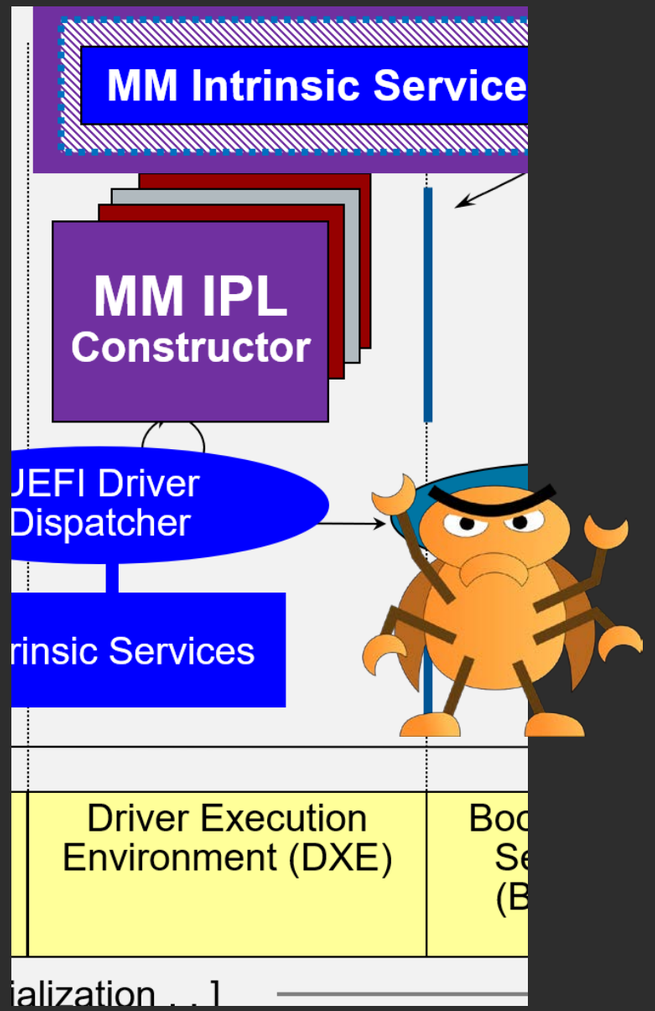
MdeModulePkg\Universal\BdsDxe\BdsEntry.c

```
VOID
EFIAPI
BdsEntry (
  IN EFI_BDS_ARCH_PROTOCOL  *This
  )
{
  BdsFormalizeEfiGlobalVariable ();
...
  InitializeLanguage (TRUE);
...
  EfiBootManagerConnectAllDefaultConsoles ();
  EfiBootManagerBoot (&BootManagerMenu); //start the setup menu
...
  Status = EfiBootManagerVariableToLoadOption
                          (BootNextVariableName, &LoadOption);
  EfiBootManagerBoot (&LoadOption);

}
```

- Invoked after DXE Dispatcher is Complete
- Connects UEFI Drivers as Required
  - Establishes Consoles (Keyboard, Video)
  - Processes UEFI Boot Options (Boots OS)

Boot Phases – SMM



- SMM initial program load (SmmIpl)
- Major SMM core drivers
- One SMM user driver introduction
- SMM BKMs

SMM Code Initialize

```
#
# SMM Initial Program Load (a DXE_RUNTIME_DRIVER)
#
INF MdeModulePkg/Core/PiSmmCore/PiSmmIpl.inf

#
# SMM_CORE - gSmst
#
INF MdeModulePkg/Core/PiSmmCore/PiSmmCore.inf

INF OvmfPkg/CpuHotplugSmm/CpuHotplugSmm.inf
INF UefiCpuPkg/CpuIo2Smm/CpuIo2Smm.inf
INF MdeModulePkg/Universal/LockBox/SmmLockBox/SmmLockBox.inf
INF UefiCpuPkg/PiSmmCpuDxeSmm/PiSmmCpuDxeSmm.inf

#
# Variable driver stack (SMM)
#
INF OvmfPkg/QemuFlashFvbServicesRuntimeDxe/FvbServicesSmm.inf
INF MdeModulePkg/Universal/FaultTolerantWriteDxe/FaultTolerantWriteSmm.inf
INF MdeModulePkg/Universal/Variable/RuntimeDxe/VariableSmm.inf
INF MdeModulePkg/Universal/Variable/RuntimeDxe/VariableSmmRuntimeDxe.inf
```

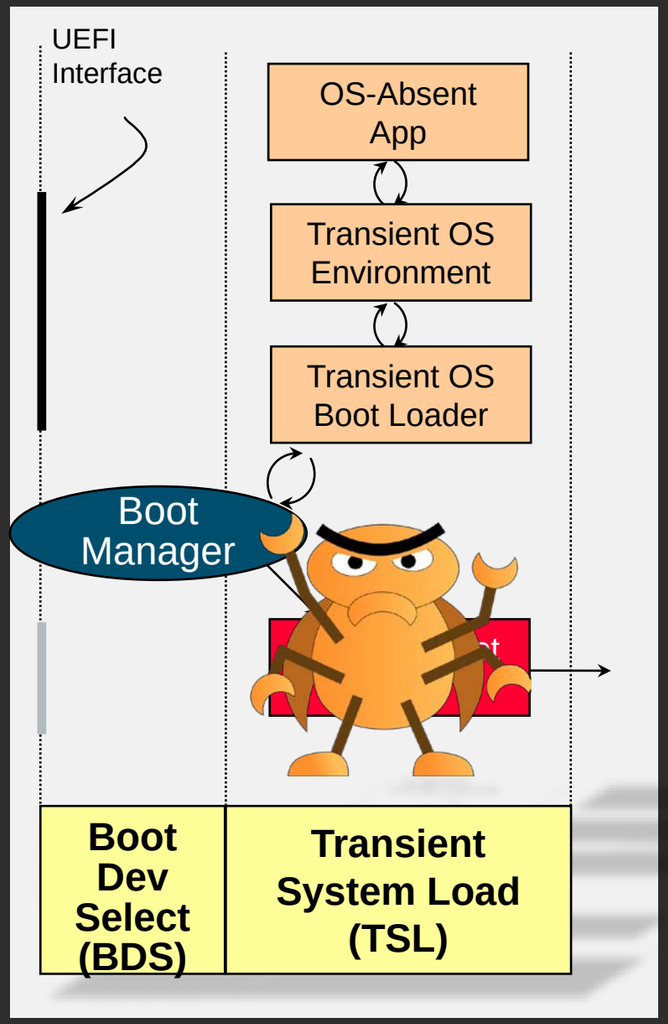
SMM driver example

UefiCpuPkg/CpuIo2Smm/CpuIo2Smm.inf

```
[Defines]
  INF_VERSION                    = 0x00010005
  BASE_NAME                      = CpuIo2Smm
  MODULE_UNI_FILE                = CpuIo2Smm.uni
  FILE_GUID                      = A47EE2D8-F60E-42fd-8E58-7BD65EE4C29B
  MODULE_TYPE                    = DXE_SMM_DRIVER
  VERSION_STRING                 = 1.0
  PI_SPECIFICATION_VERSION       = 0x0001000A
  ENTRY_POINT                    = SmmCpuIo2Initialize
```

- Don’t use non-SMM services like `gBS`, use `gSmst` instead
  e.g., `gSmst->SmmIo.Io.Read` / `gSmst->SmmAllocatePages`
- Don’t use conventional Memory (non SMRAM)
- Can register handler on SMI – `PeriodicSmiEnable()`
- Remember SMRAM is limited ~8M

Debugging the Boot Phases - Pre-Boot



- "C" source debugging
- UEFI Drivers
  - Init
  - Start
  - Supported
- UEFI Shell Applications
  - Entry point
  - Local variables
- `CpuBreakpoint()`

Debug in Pre-Boot – UEFI Shell Application

Add CpuBreakpoint() to SampleApp.c near the entry point

Add SampleApp.inf to the platform .dsc file

```
bash$ cd <edk2 workspace directory>
bash$ . edksetup.sh
bash$ build -m SampleApp/SampleApp.inf
```

Copy the binary SampleApp.efi to USB drive

SampleApp.c(~/src/edk2-ws/edk2/SampleApp) - gedit

Save

```
EFI_STATUS
EFIAPI
UefiMain (
  IN EFI_HANDLE        ImageHandle,
  IN EFI_SYSTEM_TABLE  *SystemTable
  )
{
        UINTN             EventIndex;
        BOOLEAN           ExitLoop;
        EFI_INPUT_KEY     Key;
        DEBUG((0xffffffff, "\n\nUEFI Base Training DEBUG DEMO\n"));
        DEBUG((0xffffffff, "0xffffffff USING DEBUG ALL Mask Bits Se

        CpuBreakpoint();
```

SUMMARY

- Debugging commands similar to all debuggers
- Debugging UEFI Platform Initialization Boot Flow

tianocore

Questions?

Return to Main Training Page

Return to Training Table of contents for next presentation link

tianocore

ACKNOWLEDGEMENTS