tianocore

UEFI & EDK II TRAINING

Intel® UEFI Development Kit Debugger (UDK Debugger)

tianocore.org

LESSON OBJECTIVE

- Identify the Intel® UEFI Development Kit Debugger host and target basic configuration and components
- Access the debugger tools
- Make changes to the target firmware
- Launch the debug application
- Use debug commands
- Debugging PI’s phases

UEFI DEBUGGER OVERVIEW

Intel® UEFI Development Kit Debugger Tool

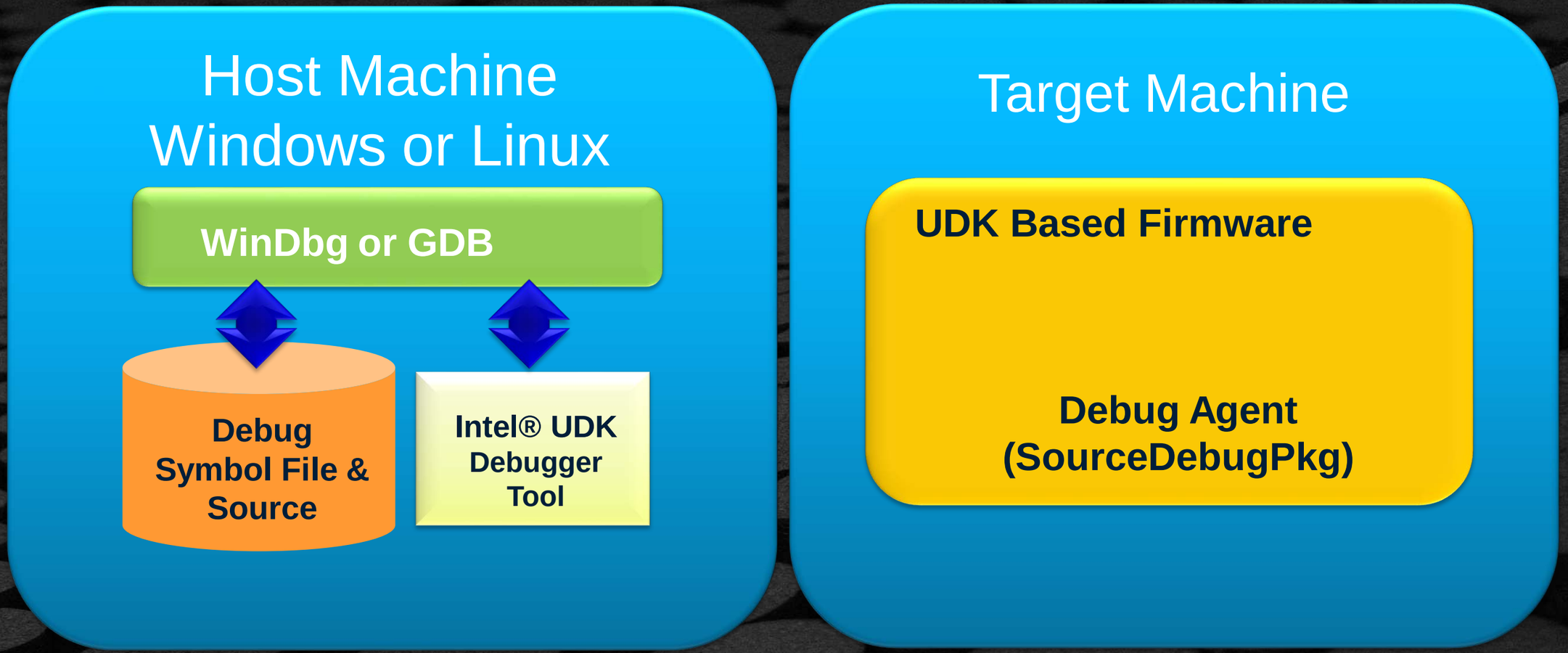
tianocore

Intel® UEFI Development Kit Debugger Tool

- Source level debug of UEFI firmware, drivers & OpROM
- Low-cost alternative to ITP/JTAG debug
- Host-to-target connect via COM port or USB debug port
- Open source based on existing software debuggers for Windows & Linux
- User Manual PDF

tianocore

Intel® UEFI Development Kit Debugger Tool



Source Level Debugger for UEFI

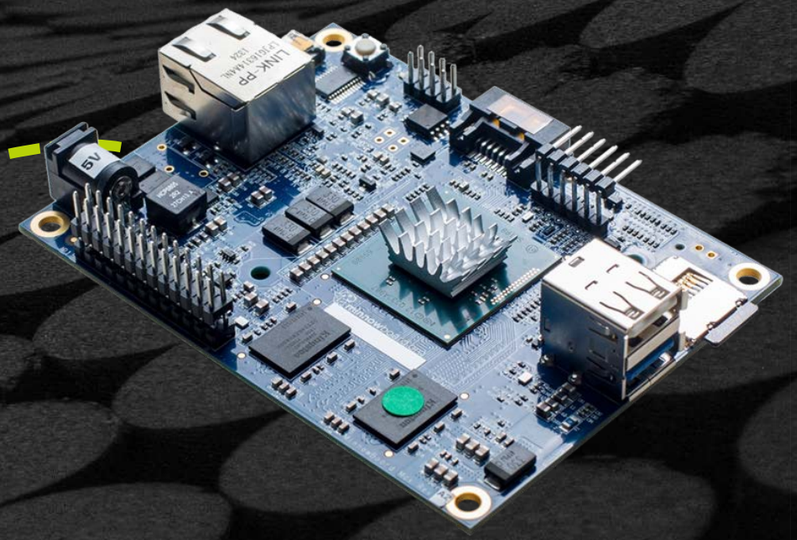
Host & Target Debug Setup

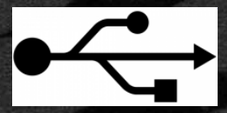
Host

Null Modem Cable or

USB 2.0 Debug Cable or USB 3.0

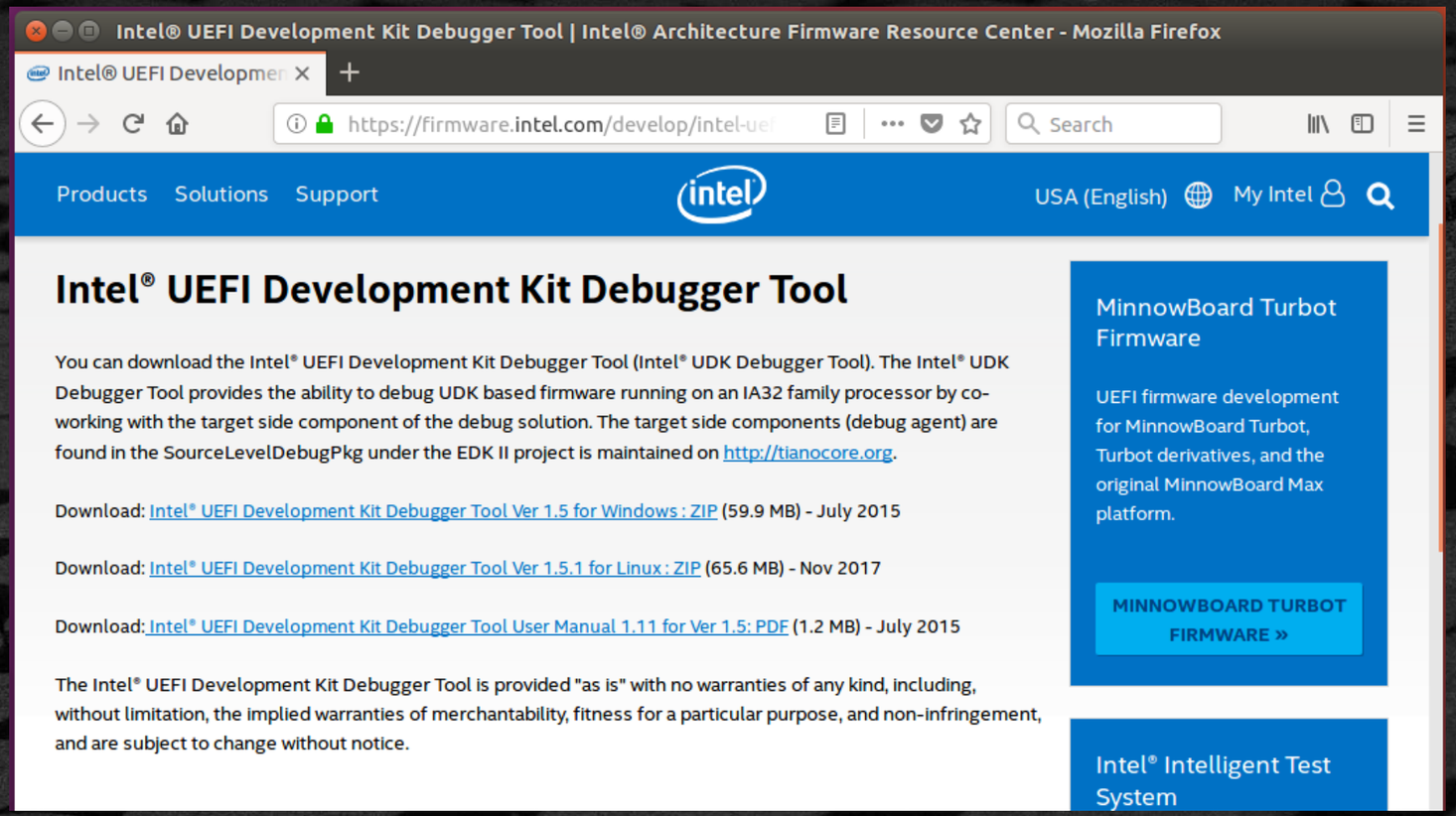
Target





Distribution

Download application: http:/firmware.intel.com - Develop - Tools



Target source: `SourceLevelDebugPkg` at TianoCore.org

Host Configuration Requirements

Microsoft Windows

- XP with Service Pack 3 and Windows 7 and Windows 10
- Debug Tool (WINDBG) x86, version 6.11.0001.404
- Intel UDK Debugger Tool
- WinDBG Extensions in **edk2.dll**
  http://msdl.microsoft.com/download/symbols/debuggers/dbg_x86_6.11.1.404.msi

- Details in backup

Host Configuration Requirements

Linux

- Ubuntu 16.04 LTS client (x64 build)- validated and examples shown
- GNU Debugger (GDB) - with Expat library
- Intel UDK Debugger Tool 1.5.1

Host Configuration Requirements-GDB

Check for the configuration of GDB that is installed

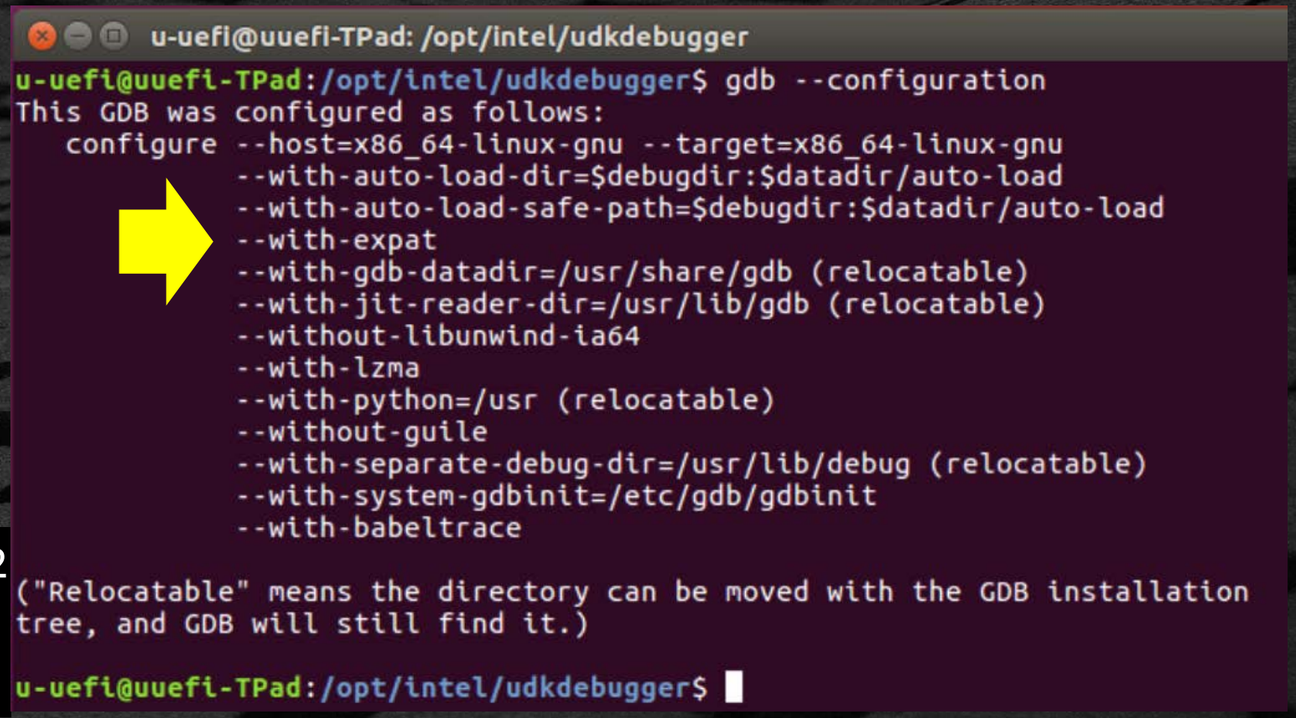
```
bash$ gdb –configuration
```

Install gdb if not installed

```
bash$ sudo apt-get update
bash$ sudo apt-get install gdb
```

Download gdb source and compile with Expat library if there is **no** "--with-expat" as on the screen shot here

```
bash$ ./configure --target=x86_64-w64-mingw32
--with-expat
bash$ make
```

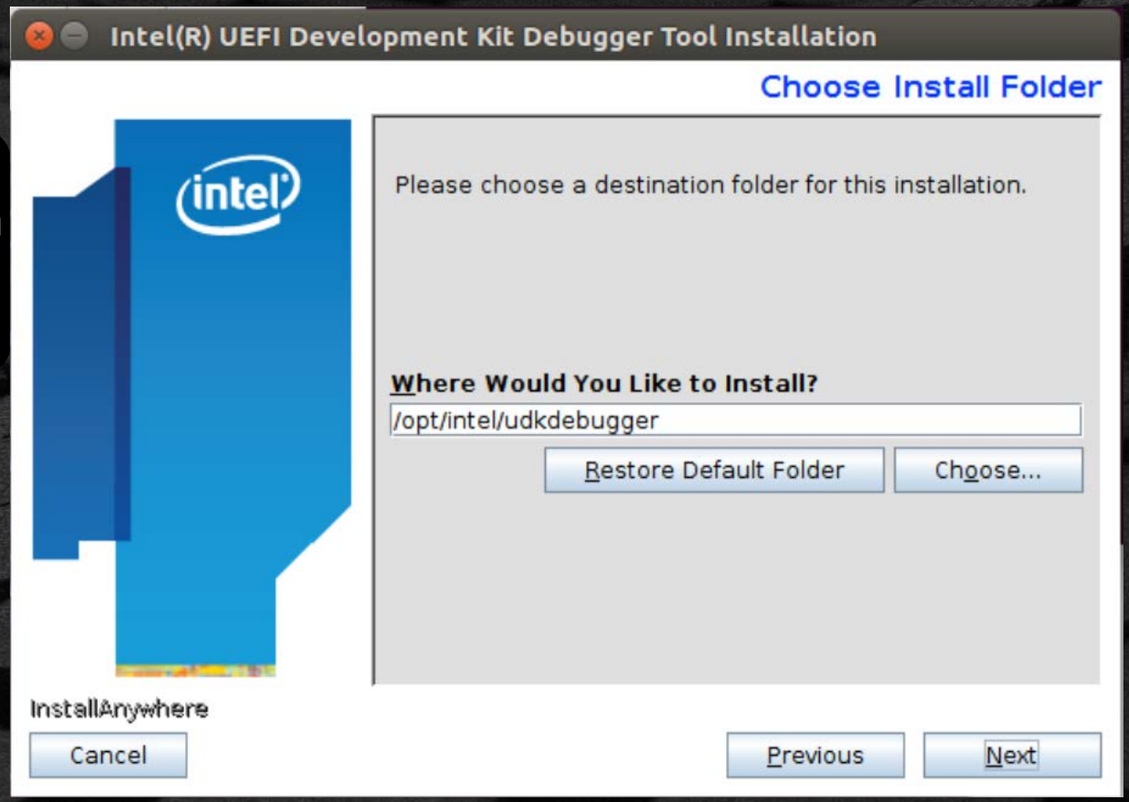


Install UDK Debugger - Linux

Download the Linux tool from : http://firmware.intel.com

Extract the .zip file to a temp directory

```
bash$ cd <temp-directory>
bash$ sudo chmod +x UDK_Debugger_Tool_v1_5_1.bin
  // run the installer
bash$ sudo ./UDK_Debugger_Tool_v1_5_1.bin
```



The tool will be installed to /opt/intel/udkdebugger by default

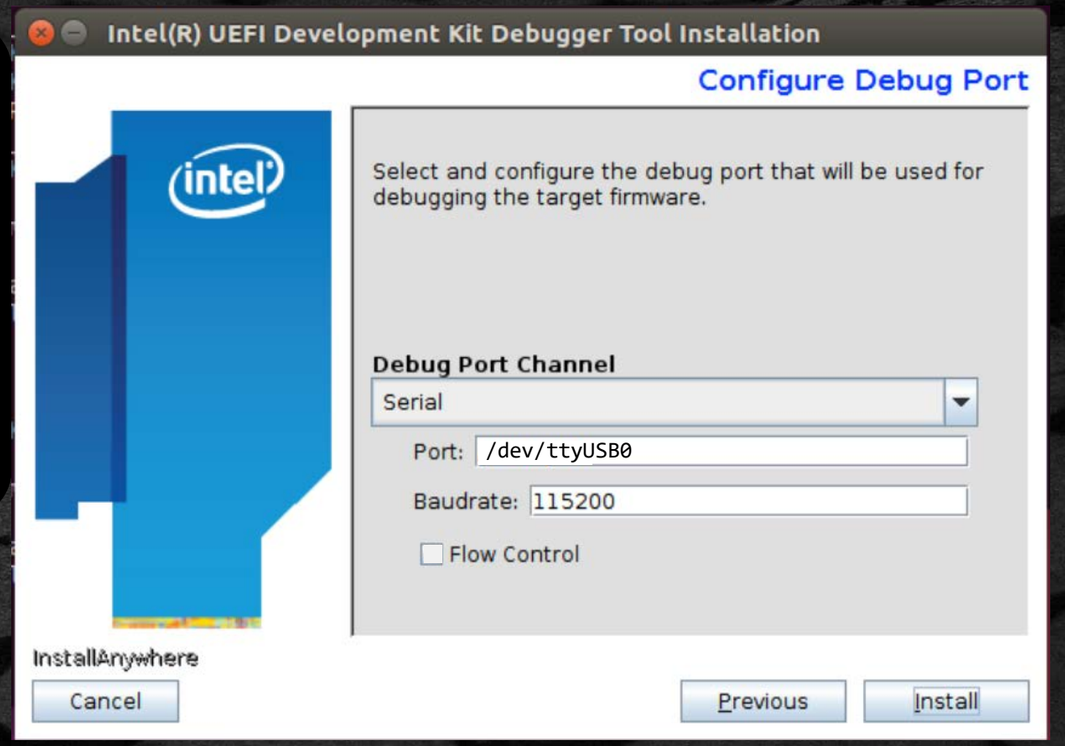
Configuration file: /etc/udkdebugger.conf

Install UDK Debugger - Linux

Menu to configure the tool for the port

Configure Debug Port Menu

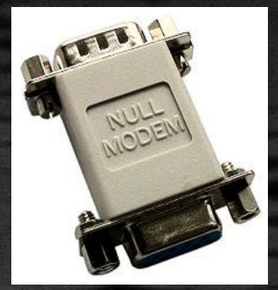
```
// Debug Port Channel
   Serial or USB
// Port:  using FTDI USB Serial use `bash$ dmesg`
// to check
       /dev/ttyUSB0
// Baudrate:
   115200
// Flow control
   none
```



Configuration file: /etc/udkdebugger.conf

Debug Cable Options

Serial Null Modem



- Target must support standard RS-232 COM port
- Host can support standard RS-232 or USB COM port
- Supported by Windows & Linux debug versions

USB 2.0 Debug

- EHCI debug descriptor (using NET20DC adapter or AMI Debug Rx device)
- Target must support USB 2.0 EHCI debug port

USB 3.0

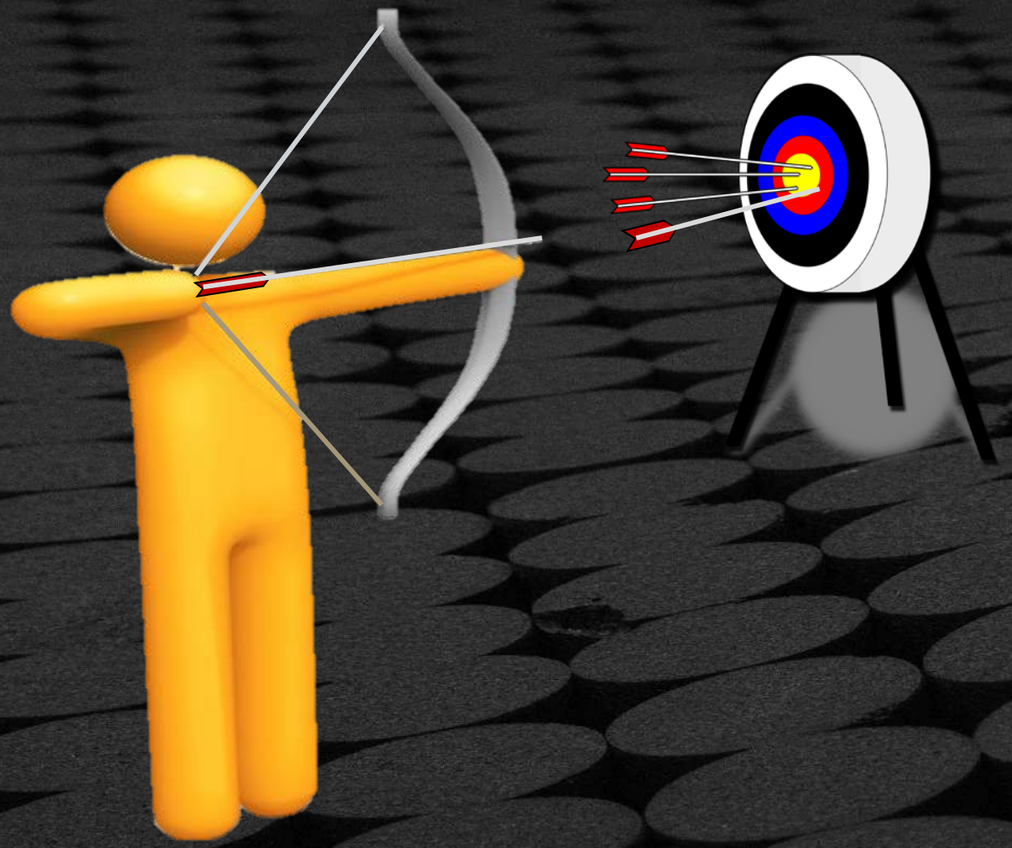
- Release 1.5

CHANGES TO TARGET

Required changes needed to be built with the target platform

Changes to Target Firmware

Goal: Minimize changes to target firmware



Add call to new library class (**`DebugAgentLib`**) In SEC, DXE Main, and SMM CPU Mod.

Or if you don't want to add one

A **NULL** implementation of **`DebugAgentLib`** is checked into open source

Updates to DSC

Libraries

SourceLevelDebugPkg Lib Instance

```
[LibraryClasess]  General
```

```
PeCoffExtraActionLib
```

➢ `PeCoffExtraActionLibDebug.inf`

```
DebugCommunicationLib
```

➢ `DebugCommunicationLibSerialPort.inf`

or

➢ `DebugCommunicationLibUsb.inf`

```
[LibraryClasses.IA32] SEC / PEI
```

```
DebugAgentLib
```

➢ `SecPeiDebugAgentLib.inf`

```
[LibraryClasses.X64] DXE
```

```
DebugAgentLib
```

➢ `DxeDebugAgentLib.inf`

```
[LibraryClasses.X64.DXE_SMM_DRIVER] SMM
```

```
DebugAgentLib
```

➢ `SmmDebugAgentLib.inf`

Updates to DSC

Libraries

SourceLevelDebugPkg Lib Instance

```
[LibraryClasess]  General
```

```
PeCoffExtraActionLib
```
➤ `PeCoffExtraActionLibDebug.inf`

```
DebugCommunicationLib
```
COM1 or USB

➤ `DebugCommunicationLibSerialPort.inf`

or

➤ `DebugCommunicationLibUsb.inf`

```
[LibraryClasses.IA32] SEC / PEI
```

```
DebugAgentLib
```
➤ `SecPeiDebugAgentLib.inf`

```
[LibraryClasses.X64] DXE
```

```
DebugAgentLib
```
➤ `DxeDebugAgentLib.inf`

```
[LibraryClasses.X64.DXE_SMM_DRIVER] SMM
```

```
DebugAgentLib
```
➤ `SmmDebugAgentLib.inf`

Updates to DSC for USB 3.0

Libraries

SourceLevelDebugPkg Lib Instance

[LibraryClasess] *General*

PeCoffExtraActionLib → **PeCoffExtraActionLibDebug.inf**

[LibraryClasses.IA32] SEC/*PEI*

DebugCommunicationLib → **DebugCommunicationLibUsb3Pei.inf**

DebugAgentLib → **SecPeiDebugAgentLib.inf**

[LibraryClasses.X64] *DXE*

DebugCommunicationLib → **DebugCommunicationLibUsb3Dxe.inf**

DebugAgentLib → **DxeDebugAgentLib.inf**

[LibraryClasses.X64.DXE_SMM_DRIVER] *SMM*

DebugCommunicationLib → **DebugCommunicationLibUsb3Dxe.inf**

DebugAgentLib → **SmmDebugAgentLib.inf**

Updates to FDF

Update for the Firmware Volume FVMAIN

- this is so there is no conflict with the terminal console driver

```
[FV.FVMAIN]
. . .
#  DXE Phase modules
. . .
# Comment out module for Terminal driver
# INF  IntelFrameworkModulePkg/Bus/Isa/IsaSerialDxe/IsaSerialDxe.inf
```

Updates to INF – default with”Debug” builds

```
//. . .
[BuildOptions]
  MSFT:*_*_*_CC_FLAGS = /Od /Oy-
```

CONFIGURE COM PORT (PCD) (TARGET)

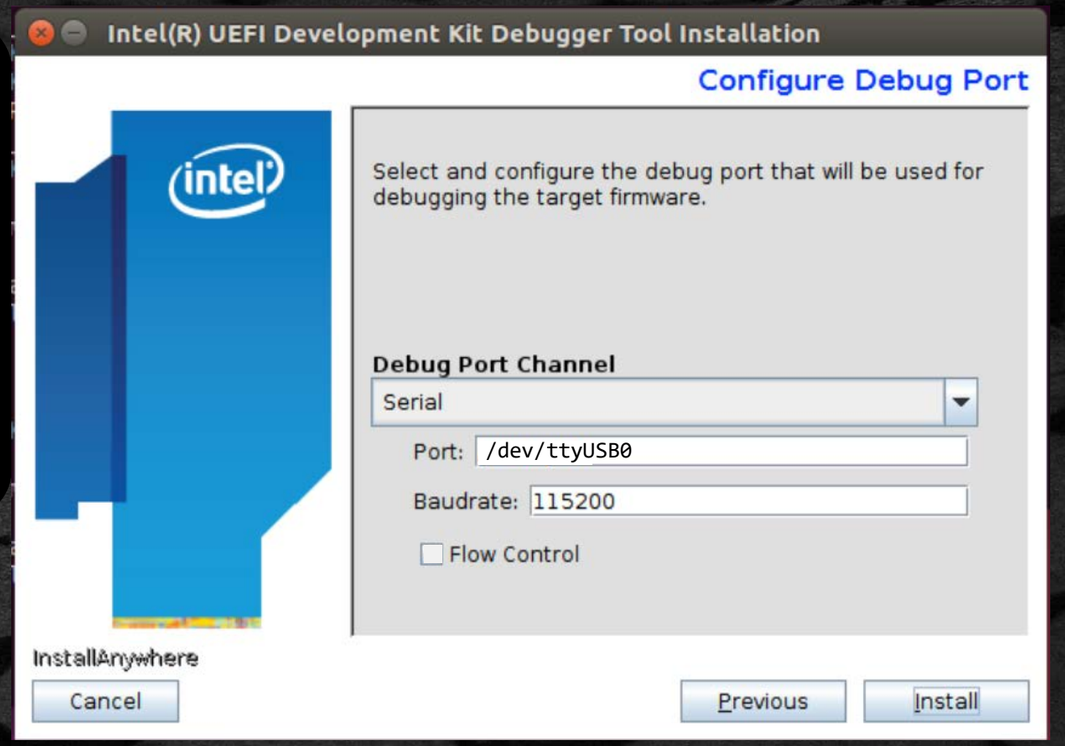
- Configure target to use COM port via PCD
- Ensure COM port not used by other project modules/features
- COM 1 is the default on target
- Simple “ASCII Print” though COM port is allowed
- Non-NULL DebugAgent library instance must be used

Install UDK Debugger - Linux

Menu to configure the tool for the port

Configure Debug Port Menu

```
// Debug Port Channel
   Serial or USB
// Port:  using FTDI USB Serial use `bash$ dmesg`
// to check
       /dev/ttyUSB0
// Baudrate:
   115200
// Flow control
   none
```



Configuration file: /etc/udkdebugger.conf

DEBUGGING USING GDB

Source Level Debug Features

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

Go to Windows WinGDB Examples

Launching UDK Debugger- Linux

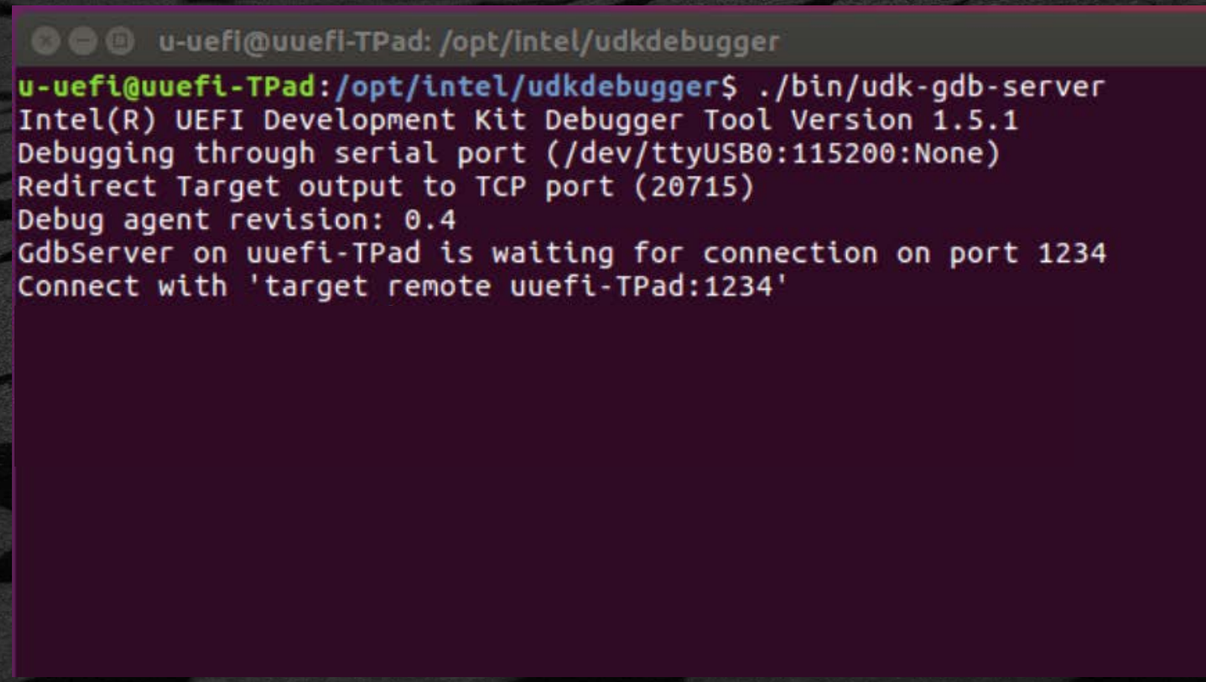
Example showing Ubuntu 16.04 LTS with GDB

Need to open 3 Terminal prompt windows
First Terminal(1) is the UDK debugger server

```
bash$ cd opt/intel/udkdebugger
bash$ ./bin/udk-gdb-server
```

Power on the Target and wait 2-3 seconds

Terminal (1)



Launching UDK Debugger- Linux

Example showing Ubuntu 16.04 LTS with GDB

Open a second Terminal(2) for GDB

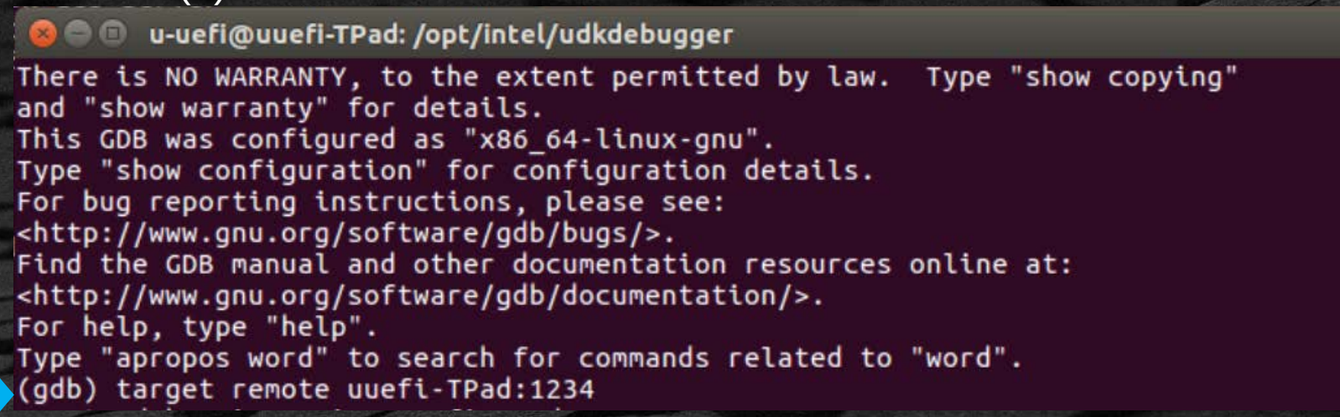
```
bash$ cd opt/intel/udkdebugger
bash$ gdb
```

Attach to the UDK debugger

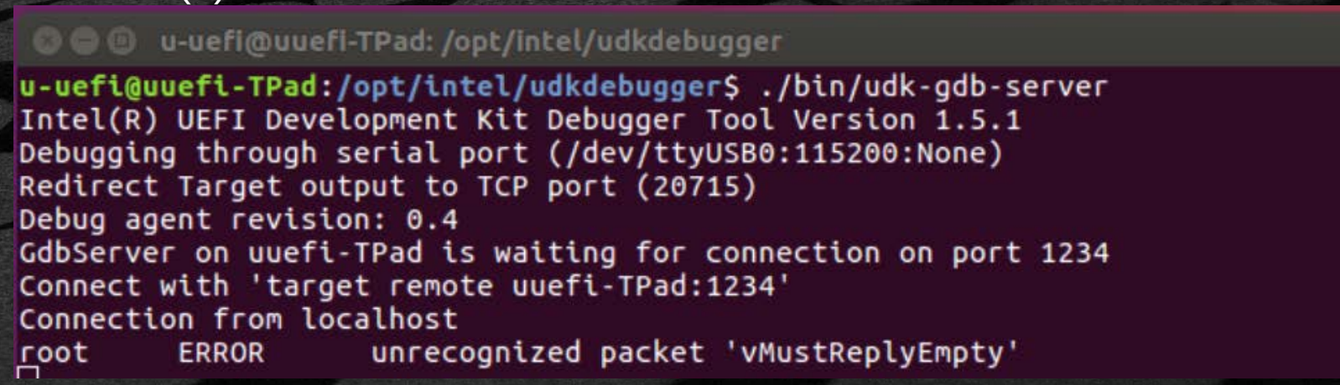
```
(gdb) target remote <HOST>:1234
```

Terminal(1) will show "Connection from localhost" message

Terminal (2)



Terminal (1)



Launching UDK Debugger- Linux

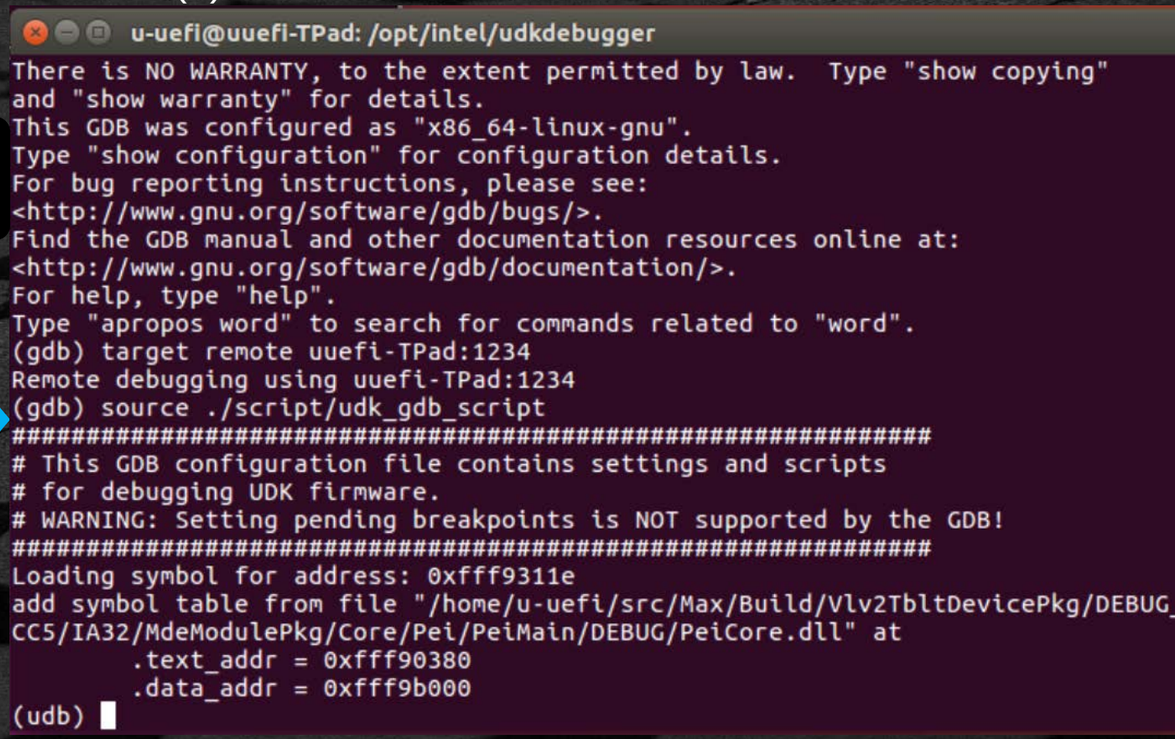
Example showing Ubuntu 16.04 LTS with GDB

Open the udk scripts in GDB – Terminal(2)

```
(gdb) source ./script/udk_gdb_script
```

Symbols will show for PeiCore, also notice the prompt changes from "(gdb)" to "(udb)"

Terminal (2)

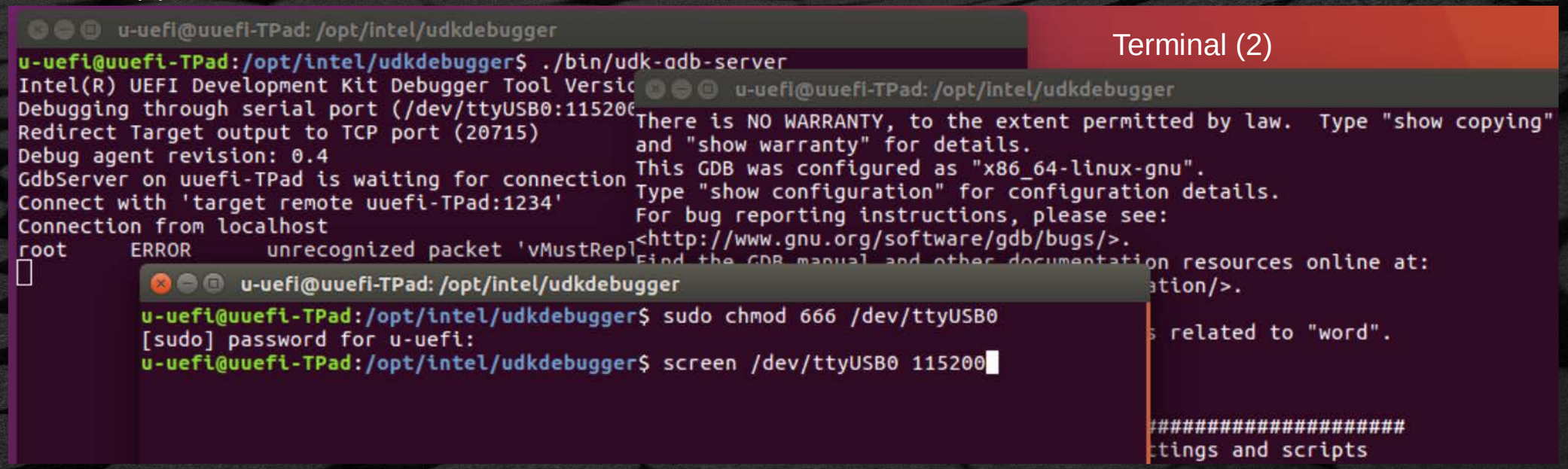


Launching UDK Debugger- Linux

Optional - open a 3rd Terminal(3) with a terminal program

- Example showing "screen" terminal program

Terminal (1)



Terminal (3)

UDK Debugger – Setting break points

Example showing Ubuntu 16.04 LTS with GDB

Terminal(2) Breakpoint at PeiDispatcher

```
(udb) b PeiDispatcher
```

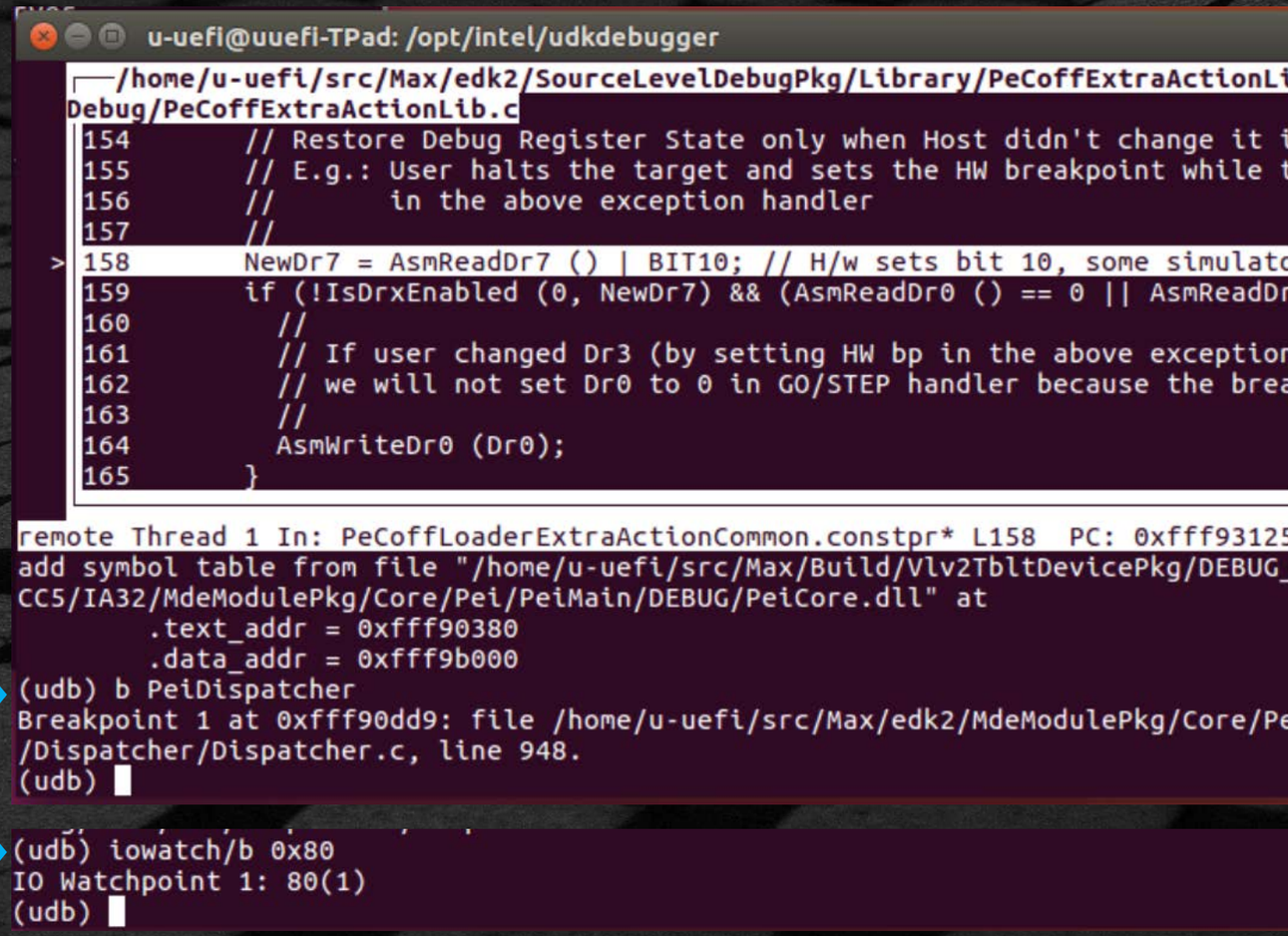
Break at Port 0x80

```
(udb)iowatch/b 0x80
```

Break at absolute address

```
(udb)b *0xfff94a68
```

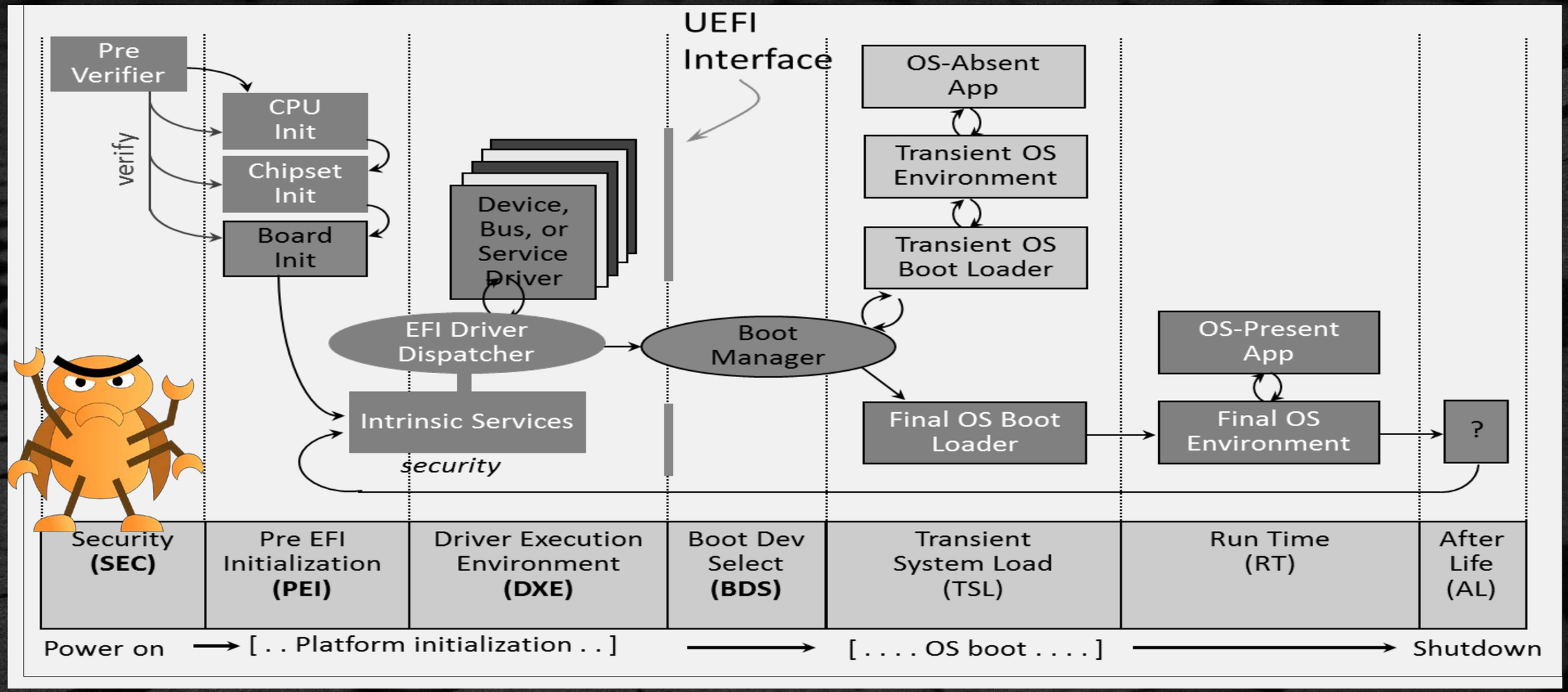
Terminal (2)



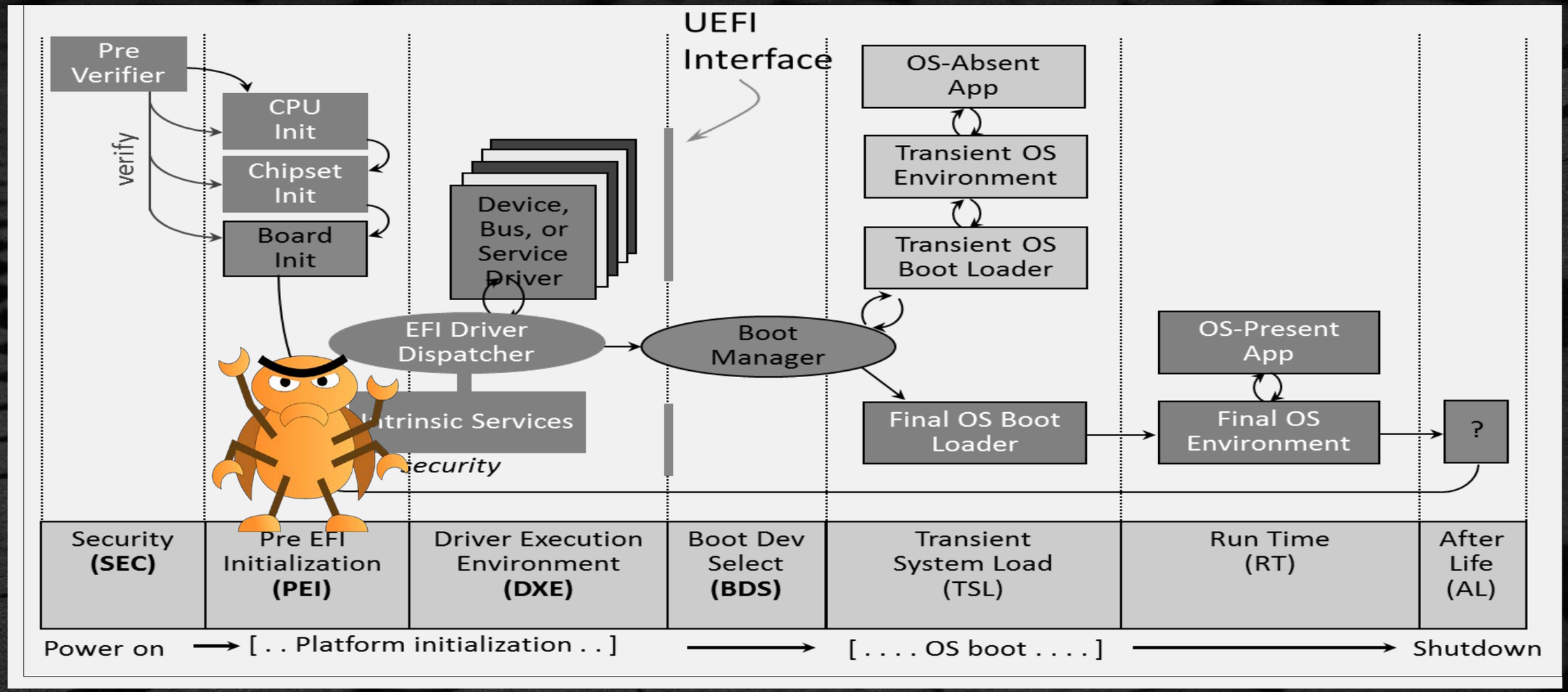
tianocore

DEBUGGING THRU BOOT FLOW

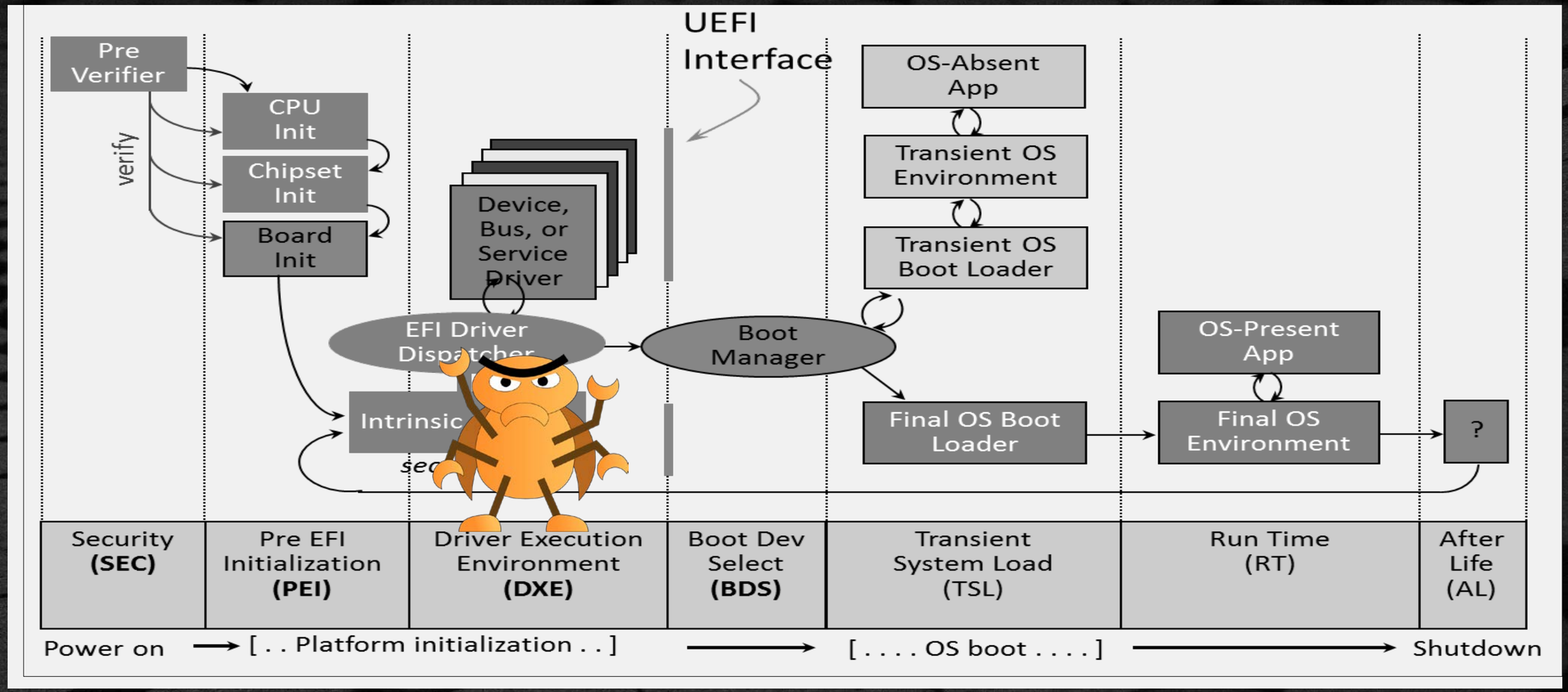
Debugging the Boot Phases



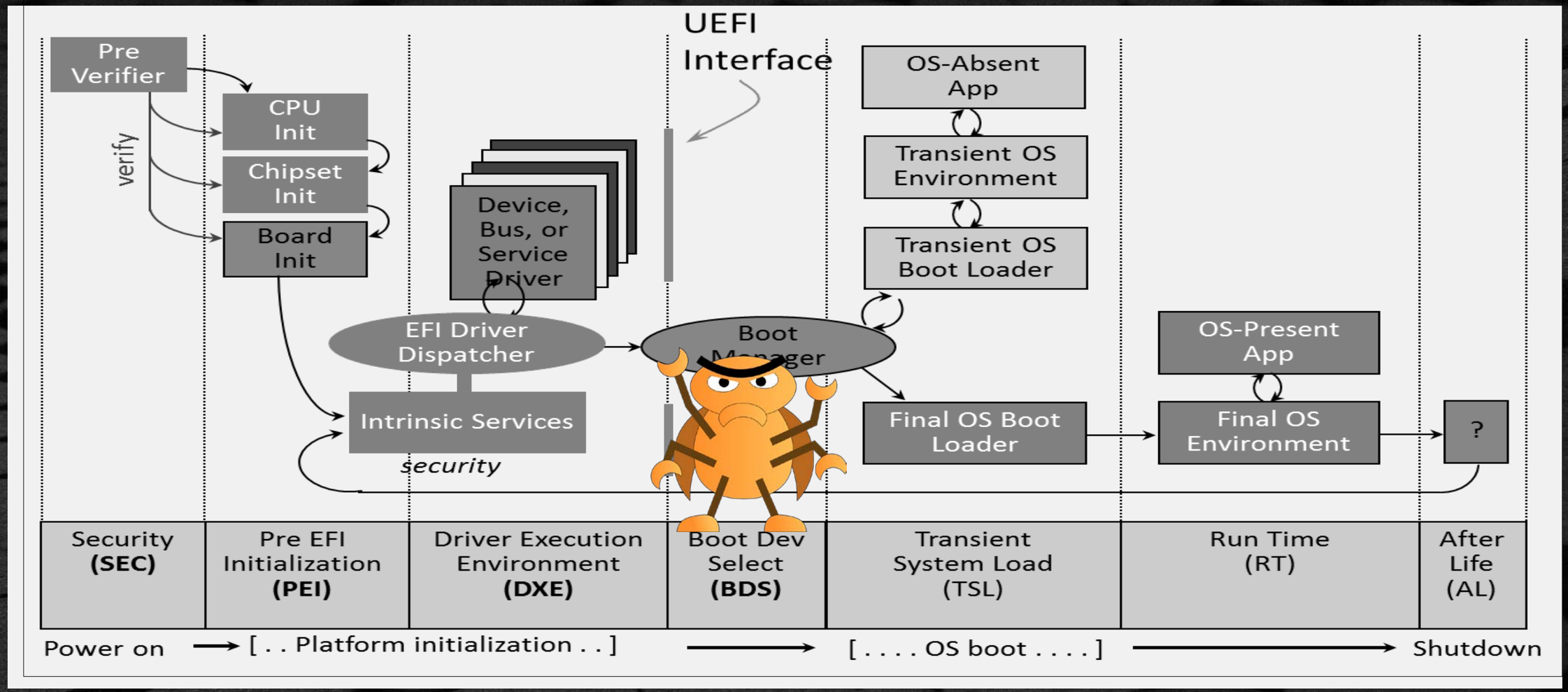
Debugging the Boot Phases



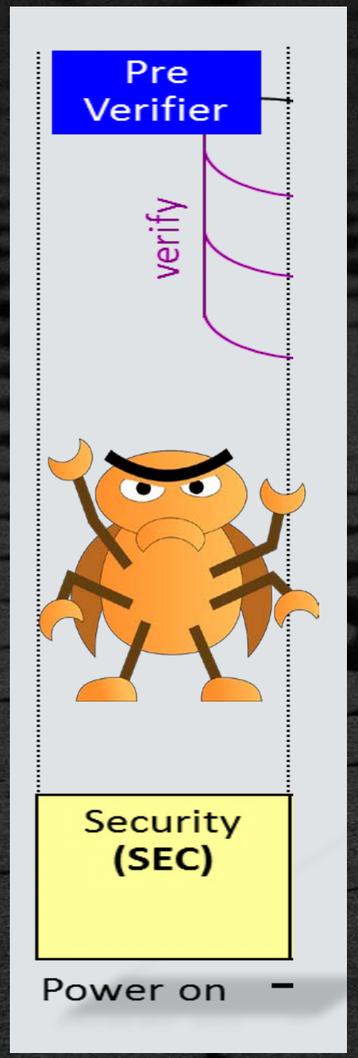
Debugging the Boot Phases



Debugging the Boot Phases

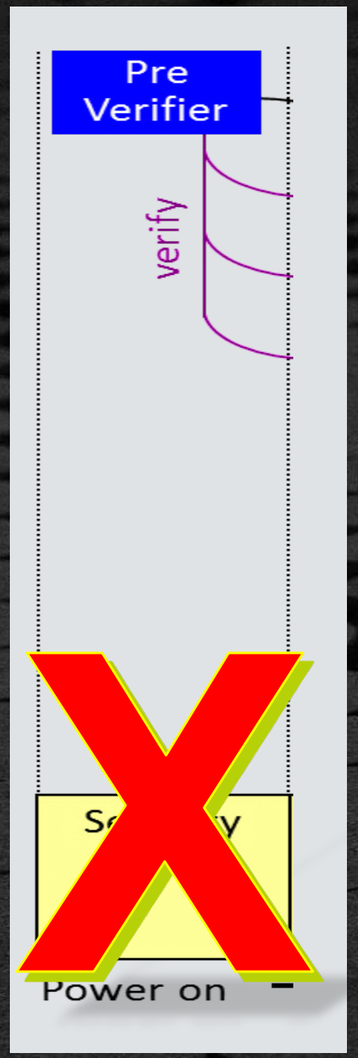


Debugging the Boot Phases - SEC



Debugging Sec Phase

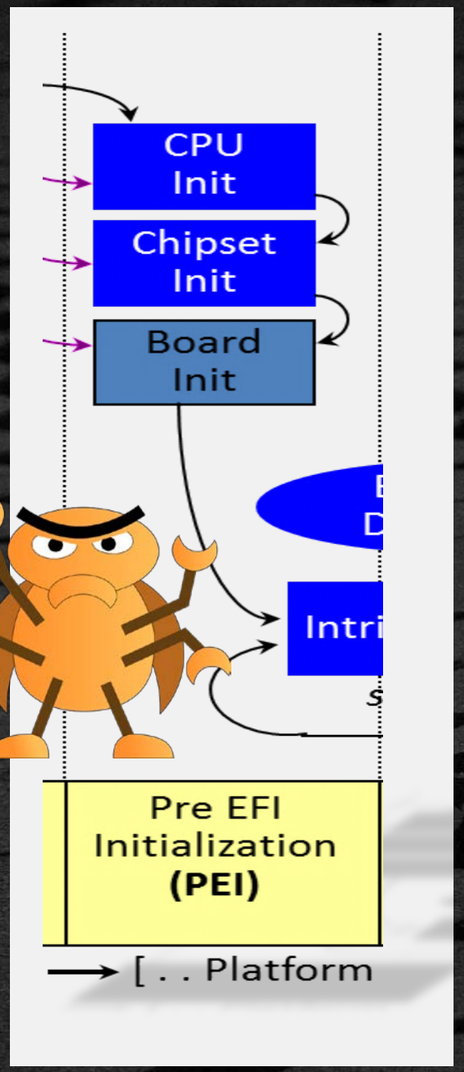
Debugging the Boot Phases - SEC



Debugging Sec Phase

SORRY – Requires a hardware debugger

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 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 1004
        CpuBreakpoint();
        PeimEntryPoint(PeimFileHandle, (const EFI_PEI_SERVICES **) &Private->Ps);
```

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 ((EFI_D_INFO, "DXE IPL Entry\n"));
// Add a call to CpuBreakpoint();  approx. line 468
  CpuBreakpoint();
  Status = TempPtr.DxeIpl->Entry (
                             TempPtr.DxeIpl,
                             &PrivateData.Ps,
                             PrivateData.HobList
```

Check for transition from DxeIpl to DXE

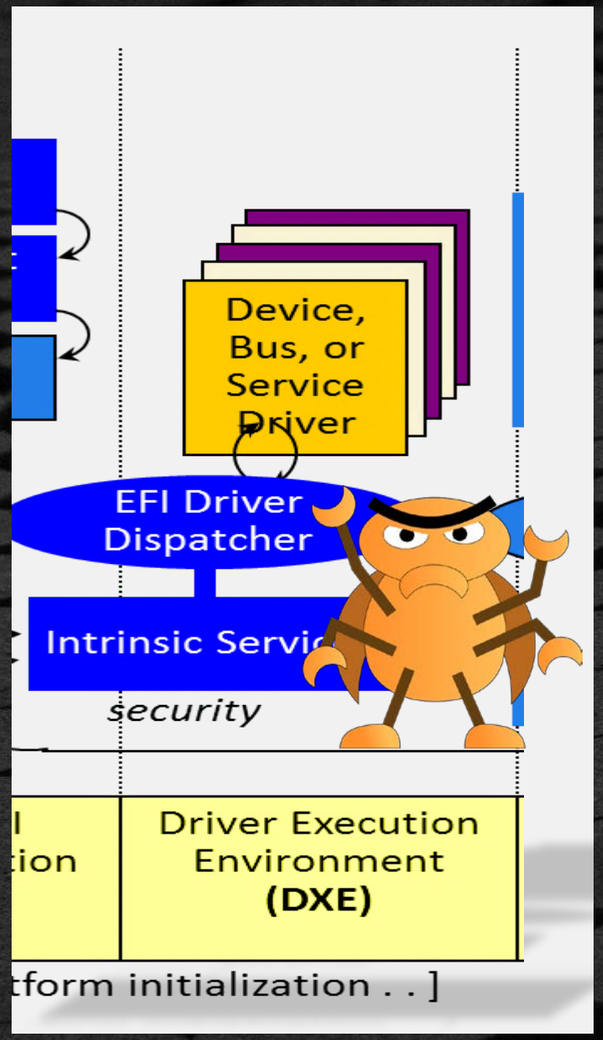
Critical point before calling DXE Core in:

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 790
  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: Trace Each Driver Load

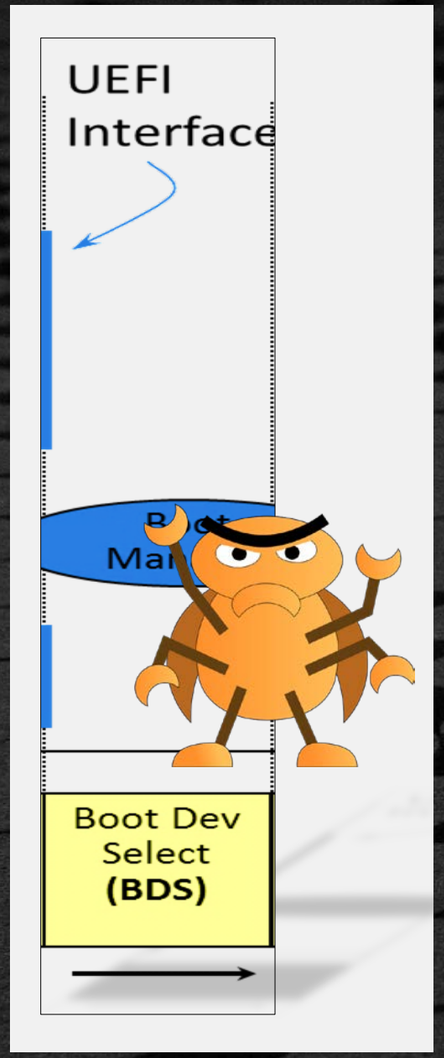
DXE Dispatcher calls to each driver's entry point in:

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 1673
    CpuBreakpoint();
    Image->Status = Image->EntryPoint (ImageHandle, Image->Info.SystemTable);
```

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

BDS Phase – Entry Point

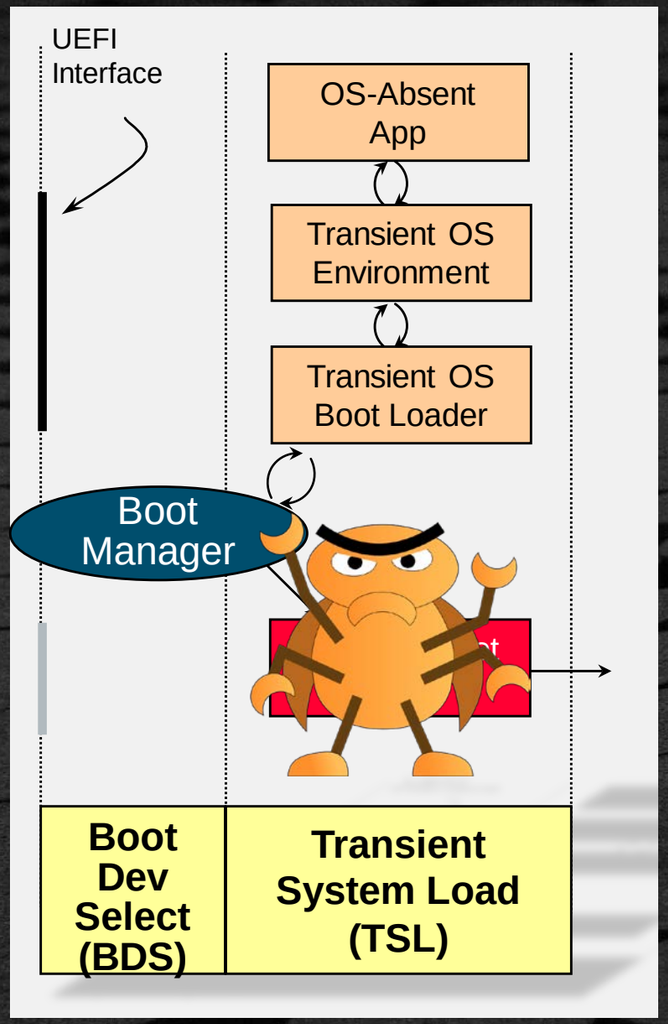
DXE call to BDS entry point in:

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 554
  CpuBreakpoint();
  gBds->Entry (gBds);

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

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/Max/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();
```

Debug in Pre-Boot – UEFI Shell Application

Use UDK Debugger and GDB to debug SampleApp

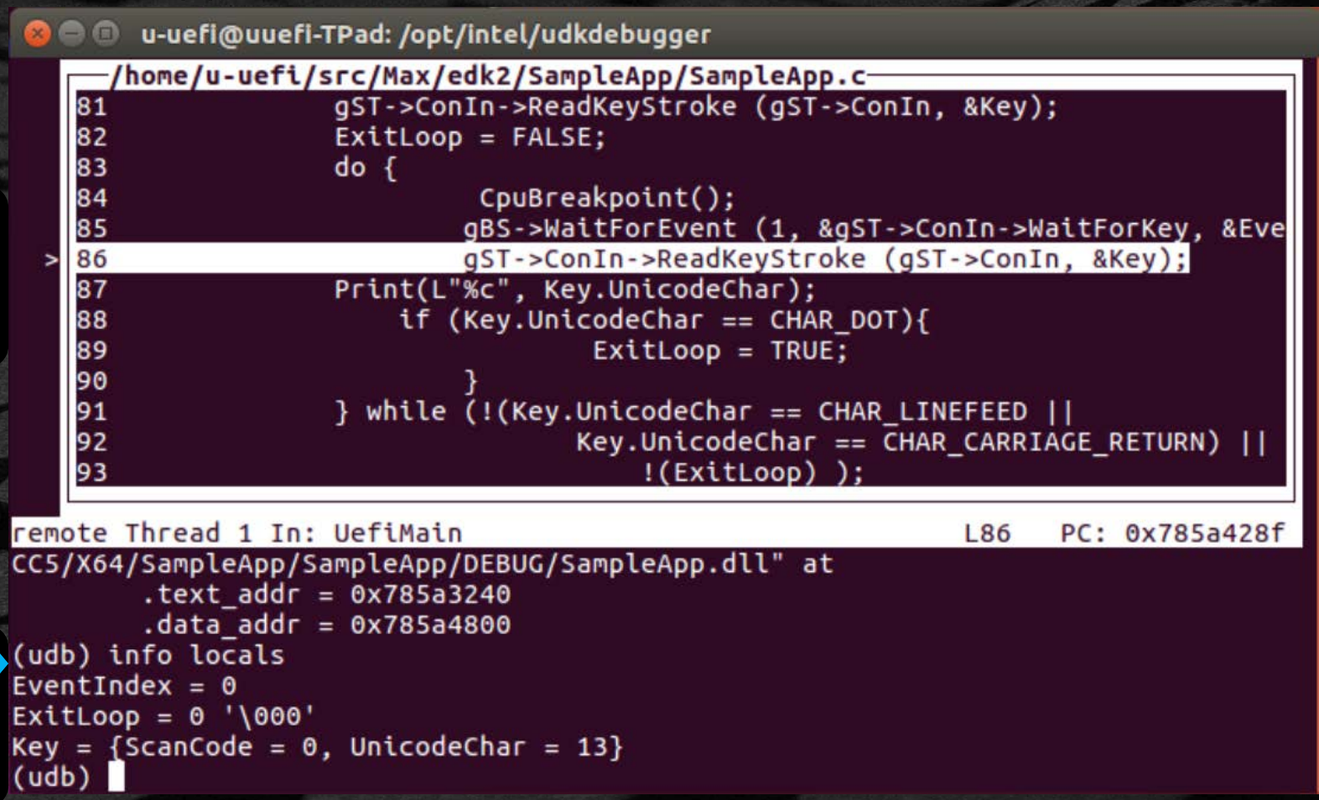
At the shell prompt on the target invoke SampleApp

```
Shell> Fs0:
FS0:/> SampleApp
```

GDB will break at the CpuBreakpoint
Begin debugging SampleApp

```
(udb) layout src
(udb) info locals
(udb) next
```

Terminal (2)



tianocore

SUMMARY

- Identify the Intel® UEFI Development Kit Debugger host and target basic configuration and components
- Access the debugger tools
- Make changes to the target firmware
- Launch the debug application
- Use debug commands
- Debugging PI’s phases

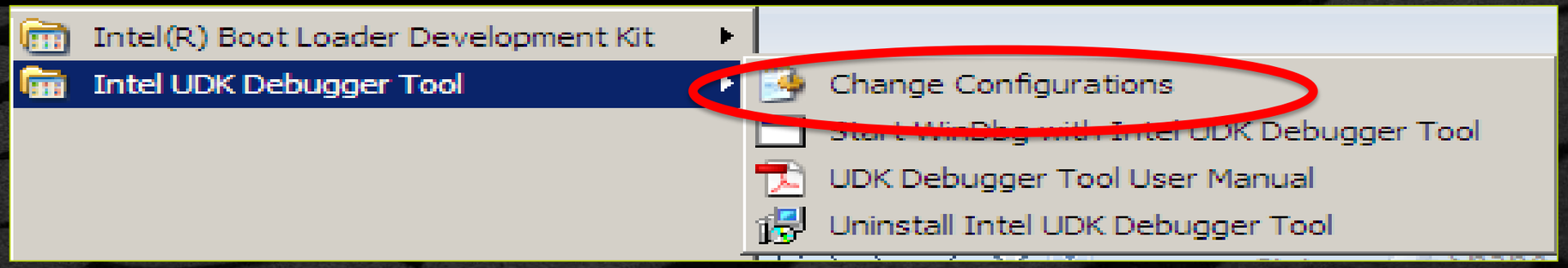

Questions?


tianocore

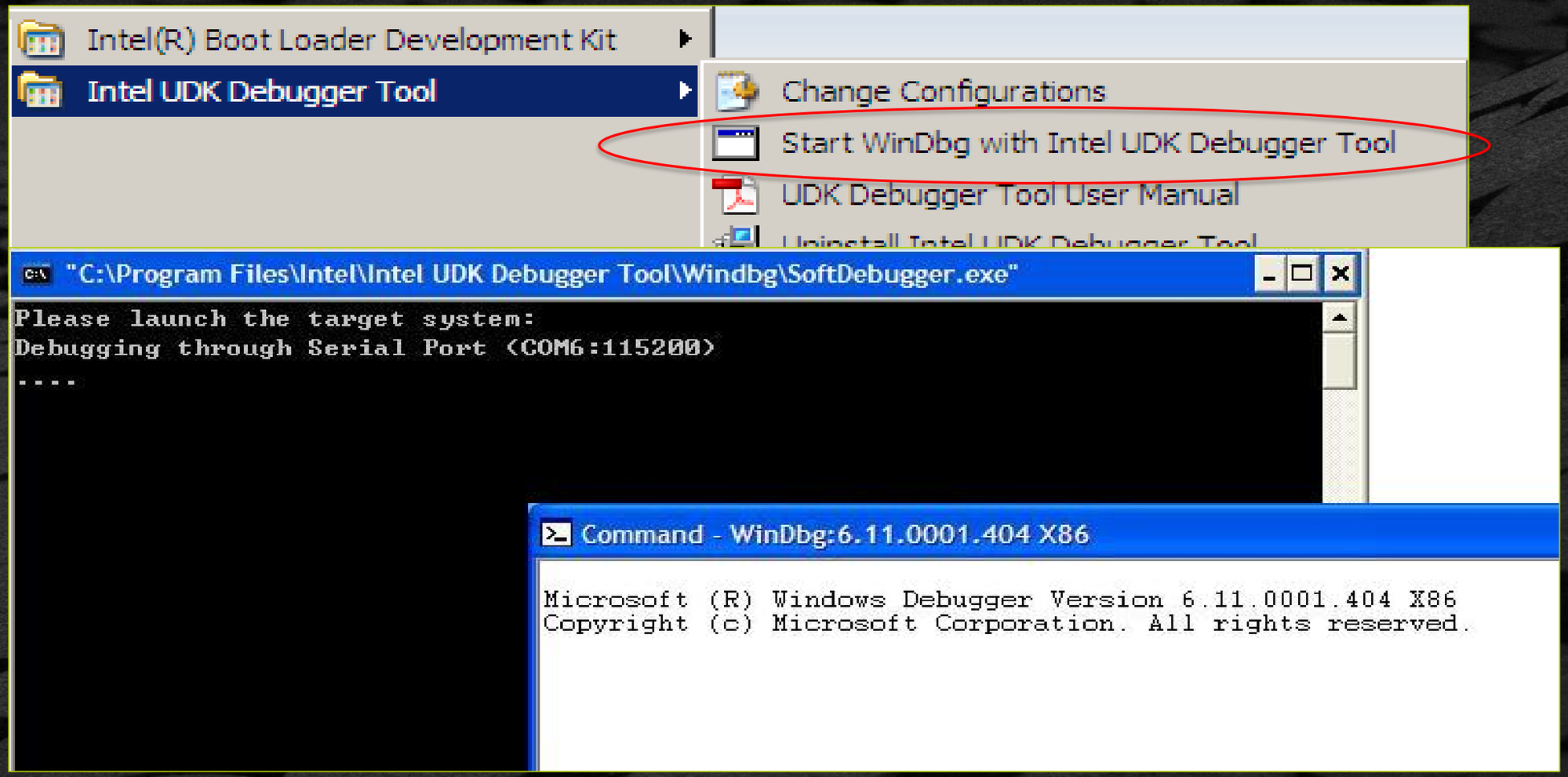
DEBUGGING USING WINDBG

Configure SoftDebugger.ini (Host)

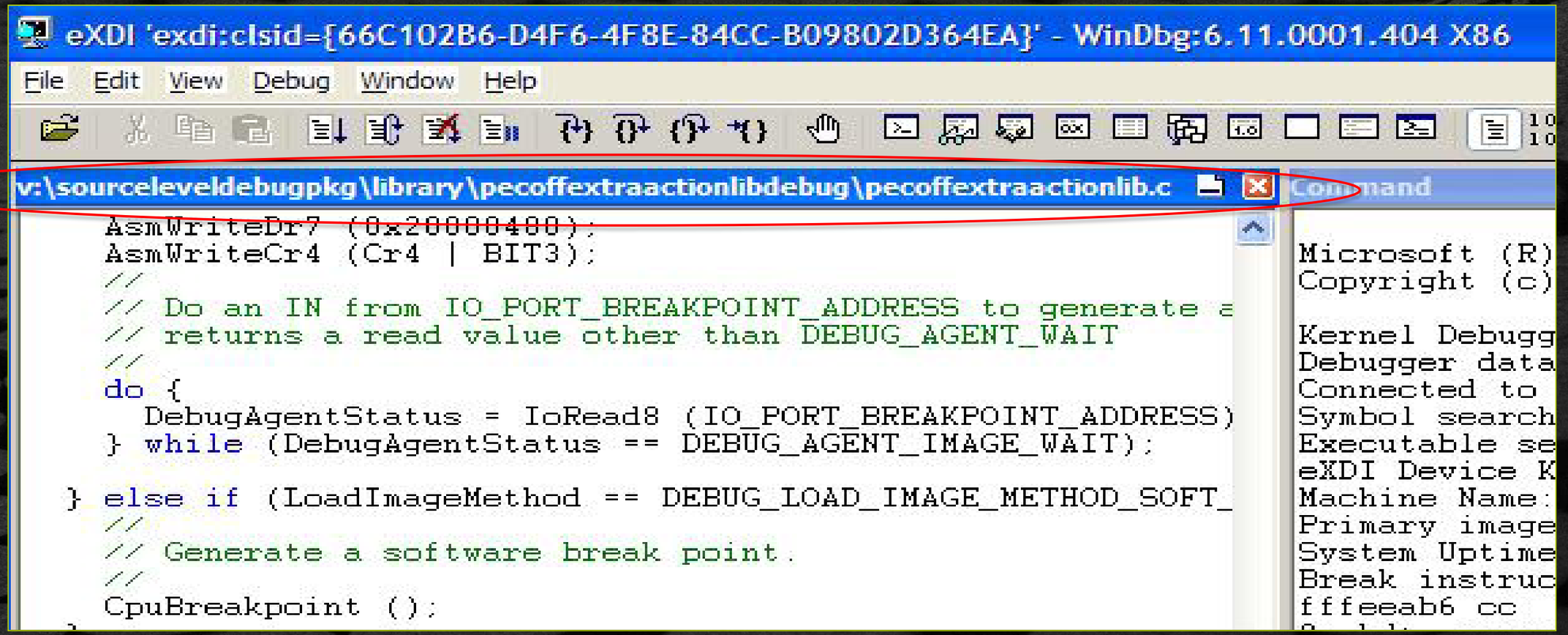
```
Debug Port]
; Channel = Usb
Channel = Serial

;The following settings only apply when Channel=Serial
Port = COM1
FlowControl = 1
BaudRate = 115200

[Target System]

[Debugger]

[Features]
LoadModuleSymbol = 1
TerminalRedirectionPort = 20715
[Maintenance]
Debug=1
```



Launch the UDK Debugger Tool Application

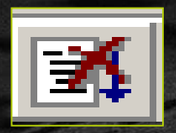


Initial Breakpoint for Debugger



Initial Breakpoint for Debugger

Code can be viewed after a “Control Break”



eXDI 'exdi:clsid={66C102B6-D4F6-4F8E-84CC-B09802D364EA}' - WinDbg:6.11.0001.404 X86

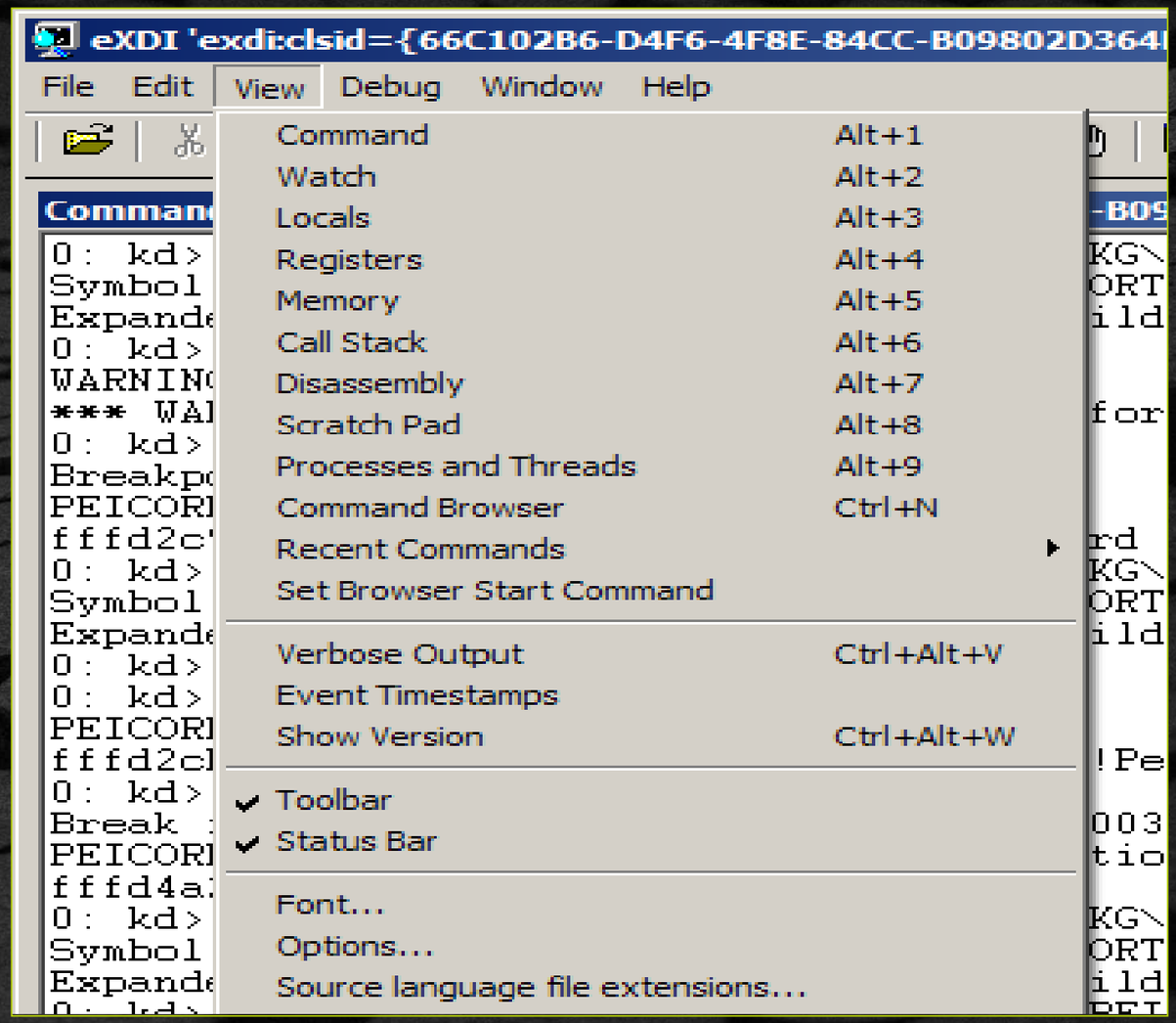
File Edit View Debug Window Help

f:\r9\l\mdepkg\library\peihoblib\hoblib.c

```
  EFI_PEI_HOB_POINTERS  Hob;

  ASSERT (HobStart != NULL);

  Hob.Raw = (UINT8 *) HobStart;
  //
  // Parse the HOB list until end of list or matching type is found.
  //
  while (!END_OF_HOB_LIST (Hob)) {
    if (Hob.Header->HobType == Type) {
      return Hob.Raw;
    }
    Hob.Raw = GET_NEXT_HOB (Hob);
  }
  return NULL;
}

/**
  Returns the first instance of a HOB type among the whole HOB list.

  This function searches the first instance of a HOB type among the whole HOB list.
  If there does not exist such HOB type in the HOB list, it will return NULL.
```

View Source from Call Stack

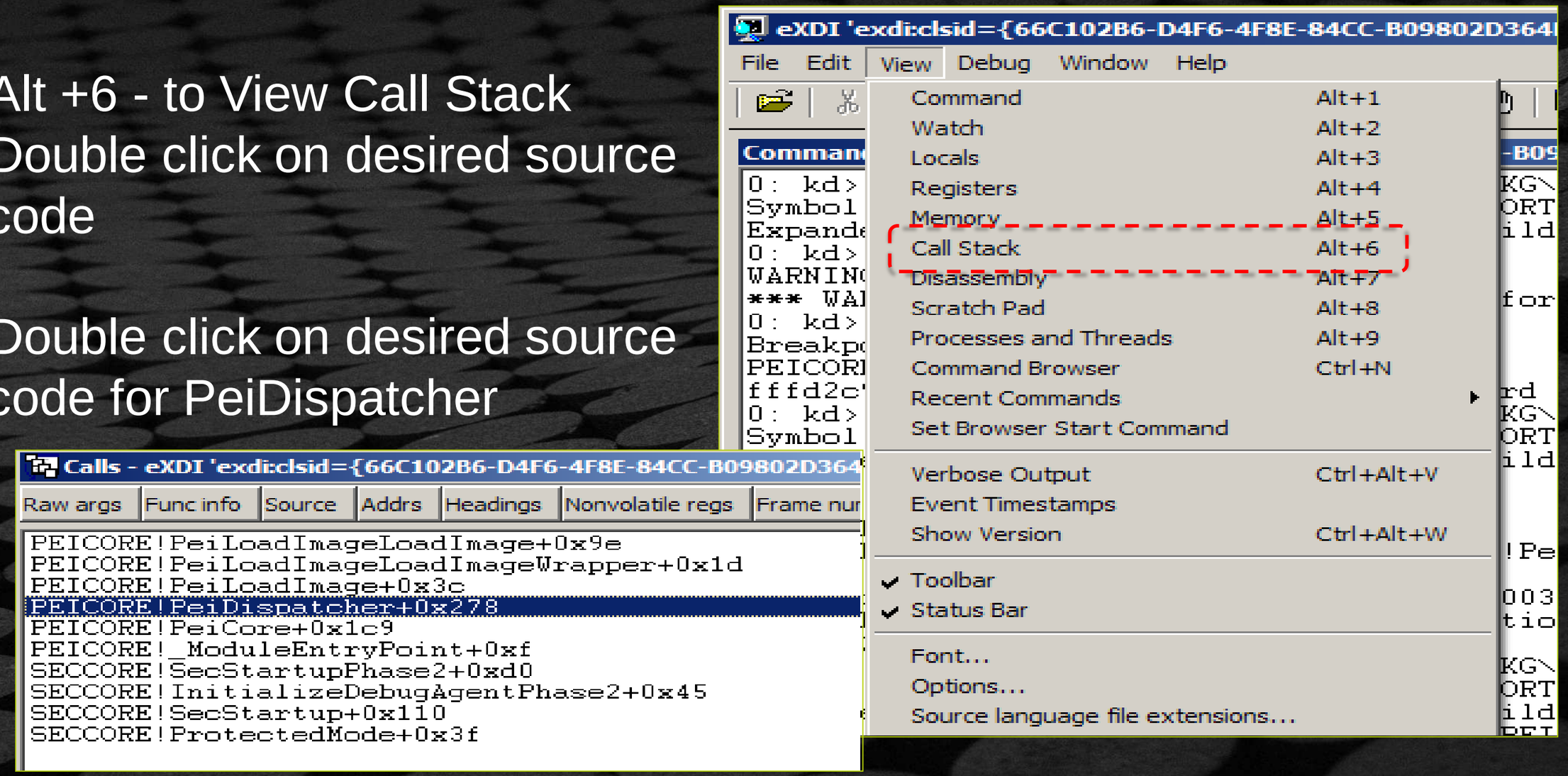
- Alt +6 - to View Call Stack
- Double click on desired source code



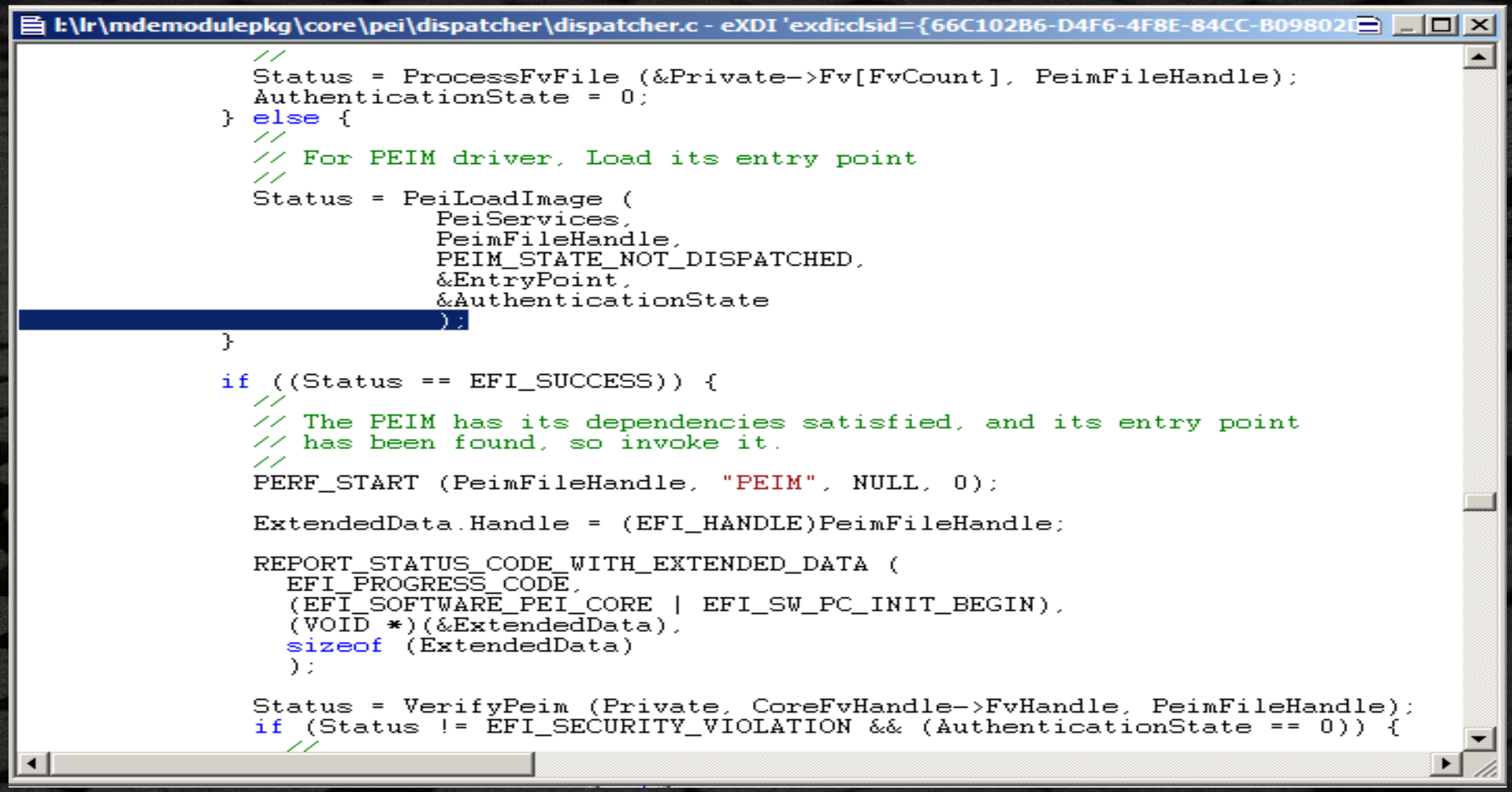
View Source from Call Stack

- Alt +6 - to View Call Stack
- Double click on desired source code

- Double click on desired source code for PeiDispatcher

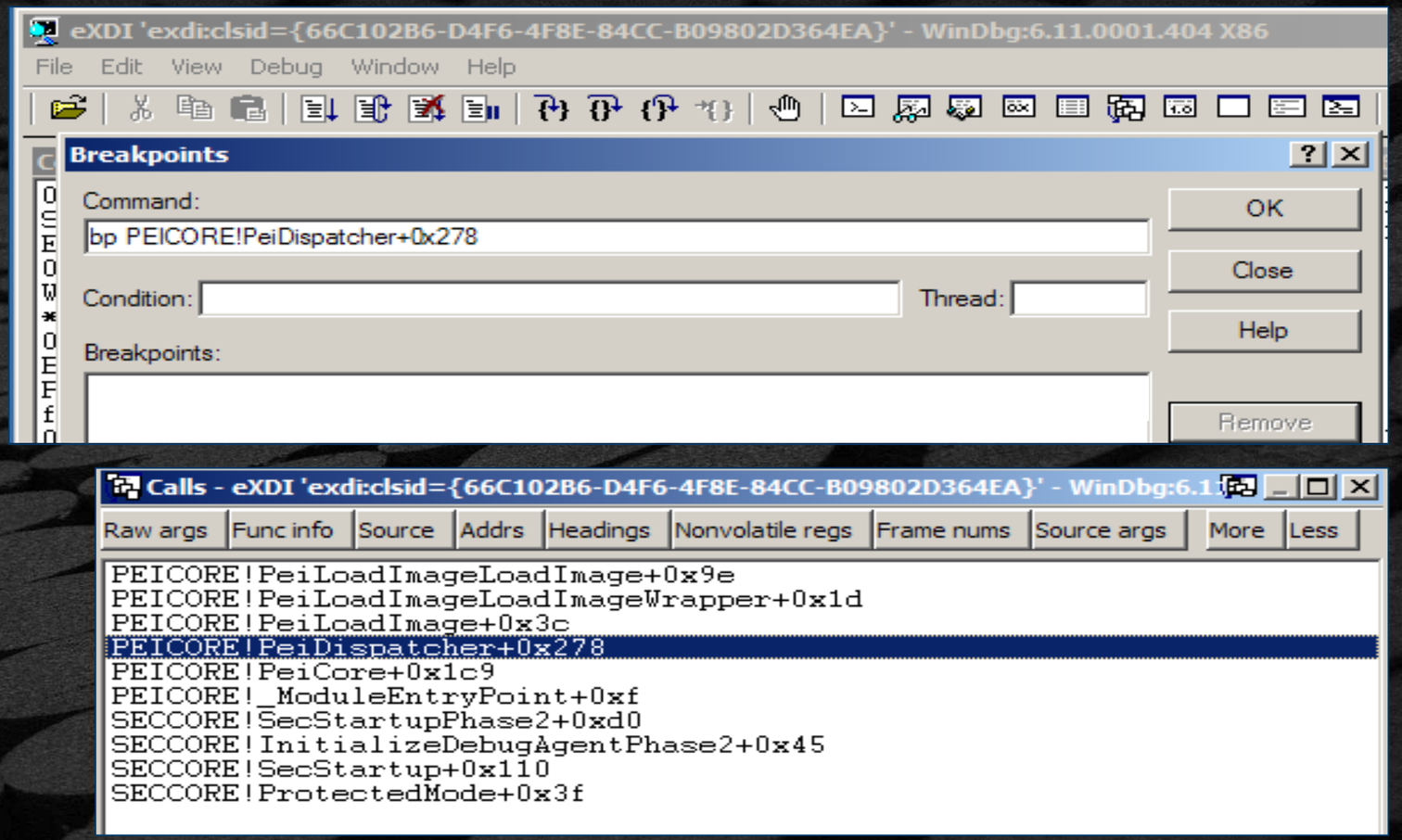


PeiDispatcher.c Opened from Call Stack



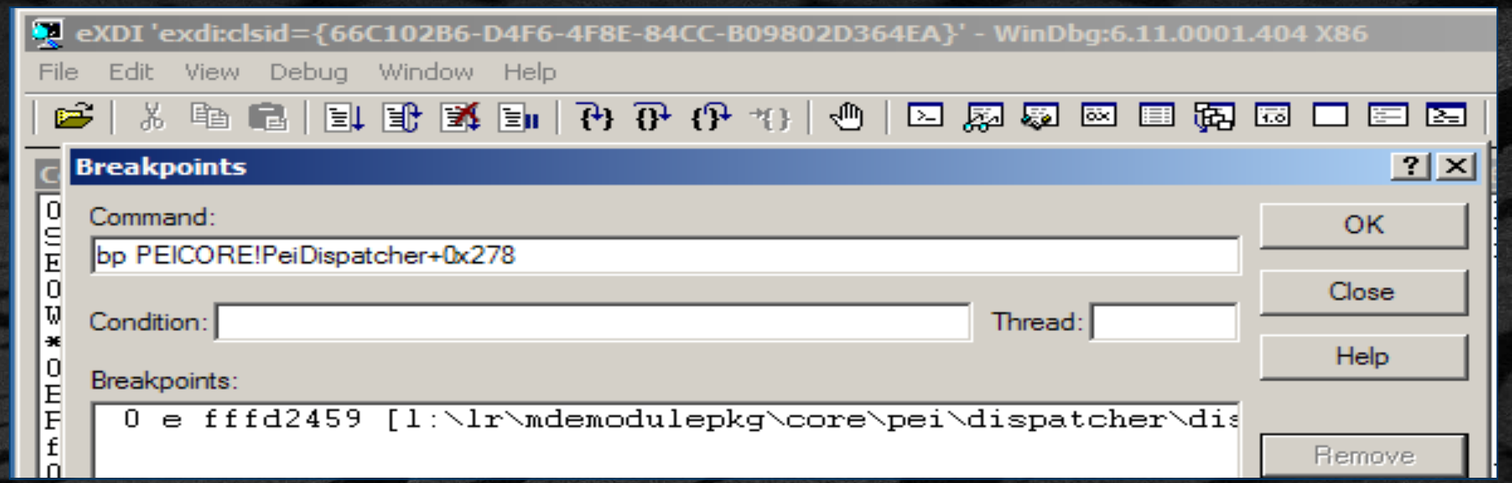
Set a Break Point from the Call Stack

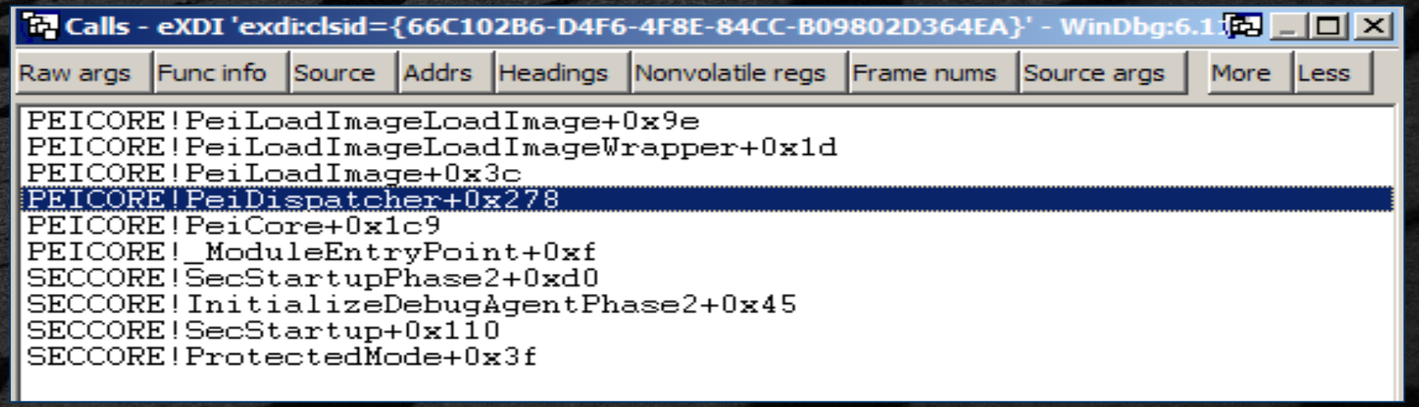
- Click on desired location in the Call Stack
- Select w/ Cntl-C (copy)
- Alt+F9 – Breakpoints menu
- Add “bp” command
- And Cntl-v to (paste) from Call Stack reference
- Click “OK”



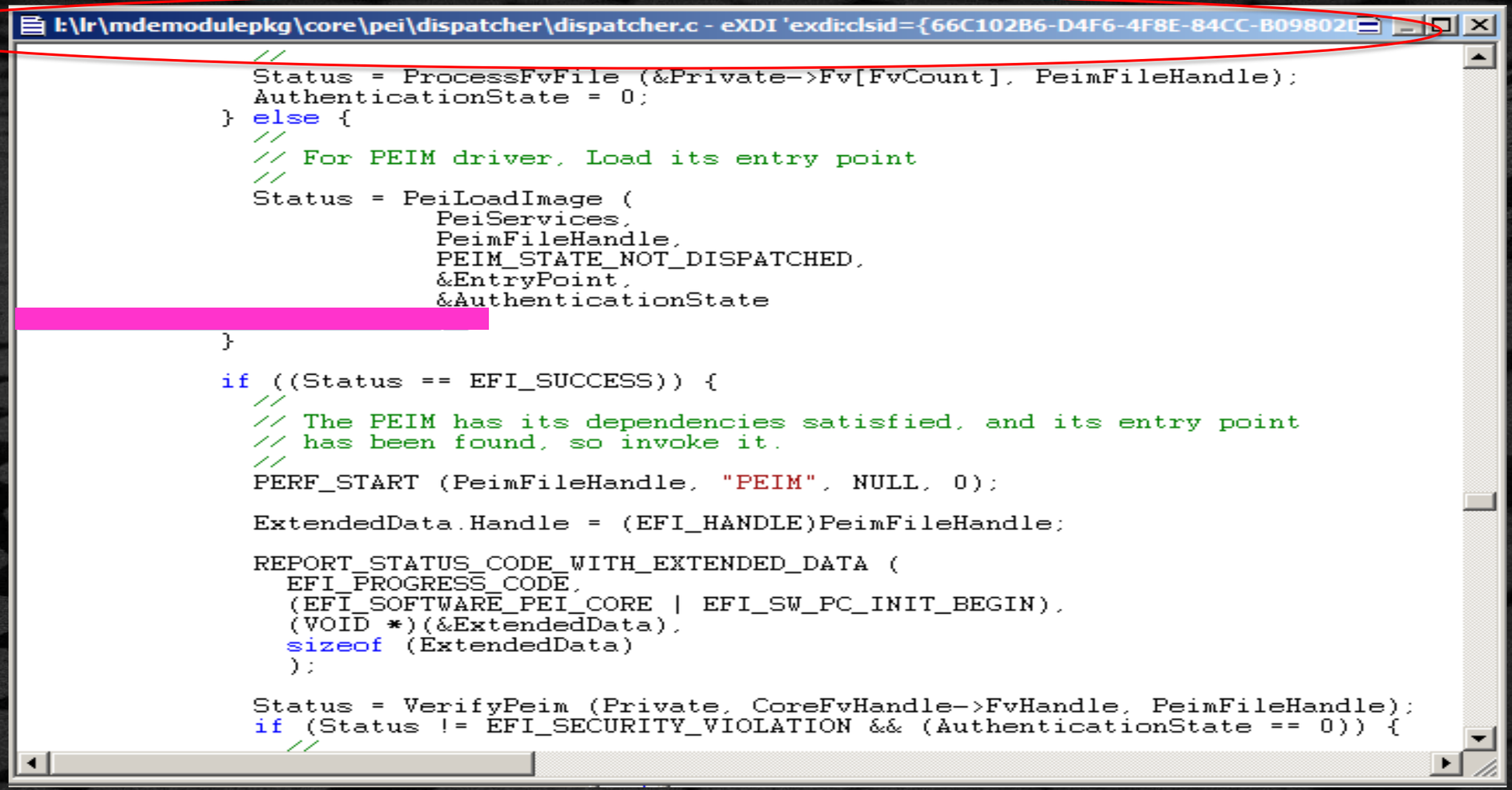
Set a Break Point from the Call Stack

- Click on desired location in the Call Stack
- Select w/ Cntl-C (copy)
- Alt+F9 – Breakpoints menu
- Add “bp” command
- And Cntl-v to (paste) from Call Stack reference
- Click “OK”
- Press “F5” to go





Next “Go” will break in Pei Dispatcher.c



tianocore

Setting a Break Point

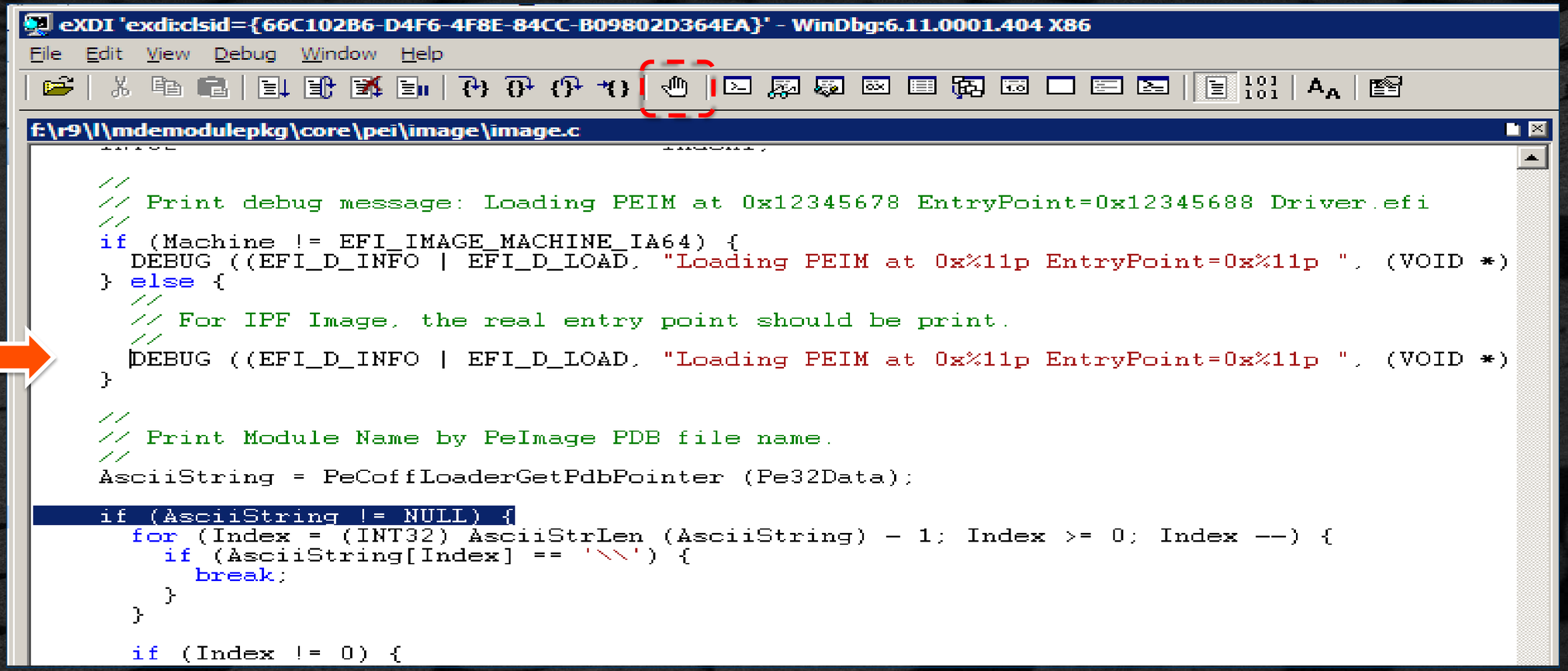
eXDI 'exdi:clsid={66C102B6-D4F6-4F8E-84CC-B09802D364EA}' - WinDbg:6.11.0001.404 X86

File Edit View Debug Window Help

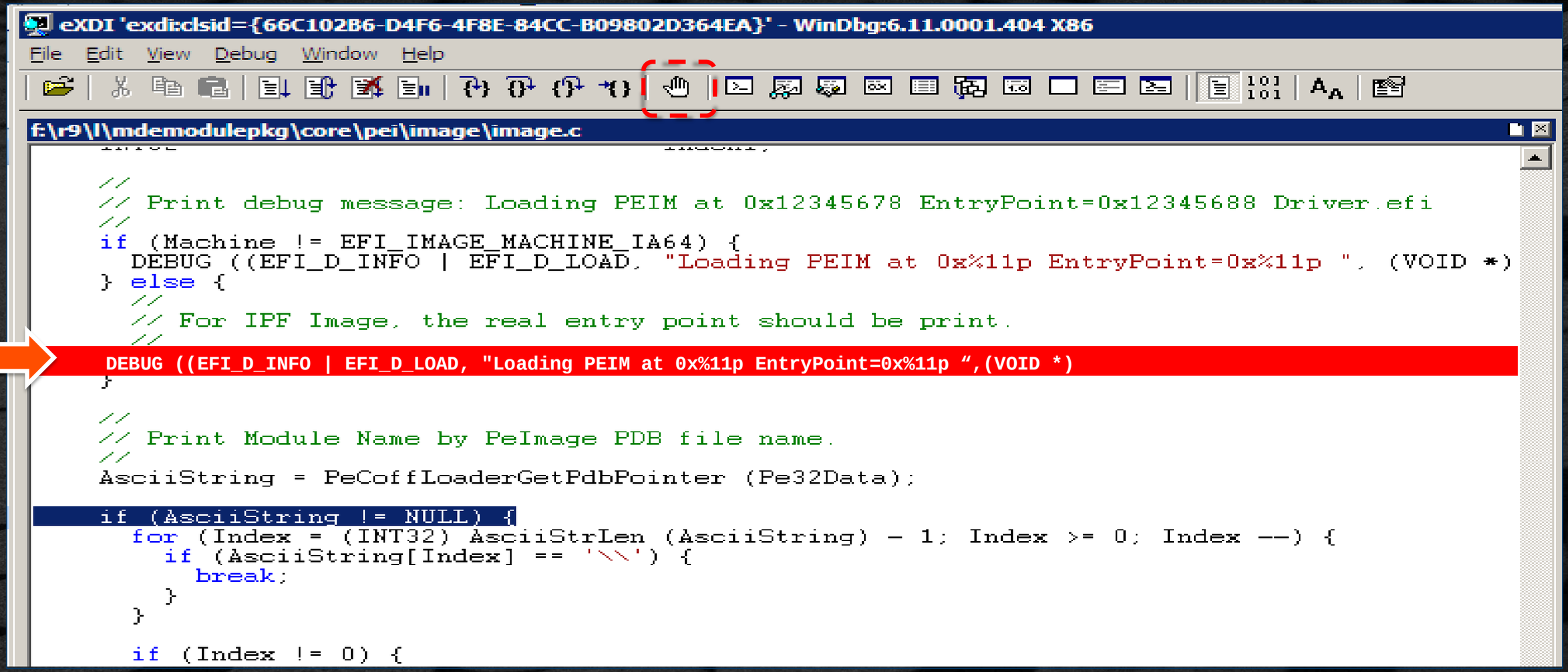
f:\r9\l\mdemodulepkg\core\pei\image\image.c

```
  //
  // Print debug message: Loading PEIM at 0x12345678 EntryPoint=0x12345688 Driver.efi
  //
  if (Machine != EFI_IMAGE_MACHINE_IA64) {
    DEBUG ((EFI_D_INFO | EFI_D_LOAD, "Loading PEIM at 0x%11p EntryPoint=0x%11p ", (VOID *)
  } else {
    //
    // For IPF Image, the real entry point should be print.
    //
    DEBUG ((EFI_D_INFO | EFI_D_LOAD, "Loading PEIM at 0x%11p EntryPoint=0x%11p ", (VOID *)
  }

  //
  // Print Module Name by PeImage PDB file name.
  //
  AsciiString = PeCoffLoaderGetPdbPointer (Pe32Data);

  if (AsciiString != NULL) {
    for (Index = (INT32) AsciiStrLen (AsciiString) - 1; Index >= 0; Index --) {
      if (AsciiString[Index] == '\\') {
        break;
      }
    }

    if (Index != 0) {
```

Setting a Break Point



Setting a Break Point



Backup

Back to Through Boot Flow