*tianocore

UEFI & EDK II Training

EDK II Debugging with Linux Lab

tianocore.org

LESSON OBJECTIVE

Define `DebugLib` and its attributes

List the ways to debug

Using PCDs to Configure `DebugLib` - LAB

Change Compiler & Linker Flags for debugging

Change the `DebugLib` instance to modify the debug output - LAB

Debug EDK II using GDB - LAB

DEBUGGING OVERVIEW

DEBUG METHODS

DEBUG and ASSERT macros
in EDK II code

DEBUG instead of Print
functions

Software/hardware debuggers

Shell commands to test
capabilities for simple
debugging

EDK II Debugging

EDK II DebugLib Library

Debug and **Assert** macros in code

Enable/disable when compiled (`target.txt`)

Connects a Host to capture debug messages

tianocore

DEBUGGING WITH PCDS

Using PCDs to Configure DebugLib

MdePkg Debug Library Class

```
[PcdsFixedAtBuild. PcdsPatchableInModule]
    •  •  •

   gEfiMdePkgTokenSpaceGuid.PcdDebugPropertyMask|0x1f
   gEfiMdePkgTokenSpaceGuid.PcdDebugPrintErrorLevel|0x80000040
```

PCDs Set which drivers report errors and change what messages get printed

PcdDebugPropertyMask Values

Debugging *Features* Enabled

```
#define DEBUG_PROPERTY_DEBUG_ASSERT_ENABLED       0x01
#define DEBUG_PROPERTY_DEBUG_PRINT_ENABLED        0x02
#define DEBUG_PROPERTY_DEBUG_CODE_ENABLED         0x04
#define DEBUG_PROPERTY_CLEAR_MEMORY_ENABLED       0x08
#define DEBUG_PROPERTY_ASSERT_BREAKPOINT_ENABLED  0x10
#define DEBUG_PROPERTY_ASSERT_DEADLOOP_ENABLED    0x20
```

Default value in OvmfPkg is 0x2f

Default value in EmulatorPkg is 0x1f

Determines which debugging features are enabled

Debug *Messages* Displayed

```
#define DEBUG_INIT       0x00000001   // Initialization
#define DEBUG_WARN       0x00000002   // Warnings
#define DEBUG_LOAD       0x00000004   // Load events
#define DEBUG_FS         0x00000008   // EFI File system
#define DEBUG_POOL       0x00000010   // Alloc & Free's  Pool
#define DEBUG_PAGE       0x00000020   // Alloc & Free's  Page
#define DEBUG_INFO       0x00000040   // Verbose
#define DEBUG_DISPATCH   0x00000080   // PEI/DXE Dispatchers
#define DEBUG_VARIABLE   0x00000100   // Variable
#define DEBUG_BM         0x00000400   // Boot Manager
#define DEBUG_BLKIO      0x00001000   // BlkIo Driver
#define DEBUG_NET        0x00004000   // SNP / Network Io Driver
#define DEBUG_UNDI       0x00010000   // UNDI Driver
#define DEBUG_LOADFILE   0x00020000   // Load File
#define DEBUG_EVENT      0x00080000   // Event messages
#define DEBUG_GCD        0x00100000   // Global Coherency Database changes
#define DEBUG_CACHE      0x00200000   // Memory range cache-ability changes
#define DEBUG_VERBOSE    0x00400000   // Detailed debug messages that may
                                      // significantly impact boot performance
#define DEBUG_ERROR      0x80000000   // Error
```

Aliases EFI_D_INIT == DEBUG_INIT etc.

Determines which messages we want to print

Change all instances of a PCD in platform DSC

```
[PcdsFixedAtBuild.IA32]
gEfiMdePkgTokenSpaceGuid.PcdDebugPrintErrorLevel|0x00000000
```

Change a single module's PCD values in DSC

```
MyPath/MyModule.inf {
<PcdsFixedAtBuild>
gEfiMdePkgTokenSpaceGuid.PcdDebugPrintErrorLevel|0x80000000
}
```

Minimize message output and minimize size increase

Other Debug Related Libraries

ReportStatusCodeLib – Progress codes

`gEfiMdePkgTokenSpaceGuid.PcdReportStatusCodePropertyMask`

PostCodeLib – Enable Post codes

`gEfiMdePkgTokenSpaceGuid.PcdPostCodePropertyMask`

PerformanceLib – Enable Measurement

`gEfiMdePkgTokenSpaceGuid.PcdPerformanceLibraryPropertyMask`

Lab 1 – Adding Debug Statements

In this lab, you'll add debug statements to the previous lab's SampleApp UEFI Shell application
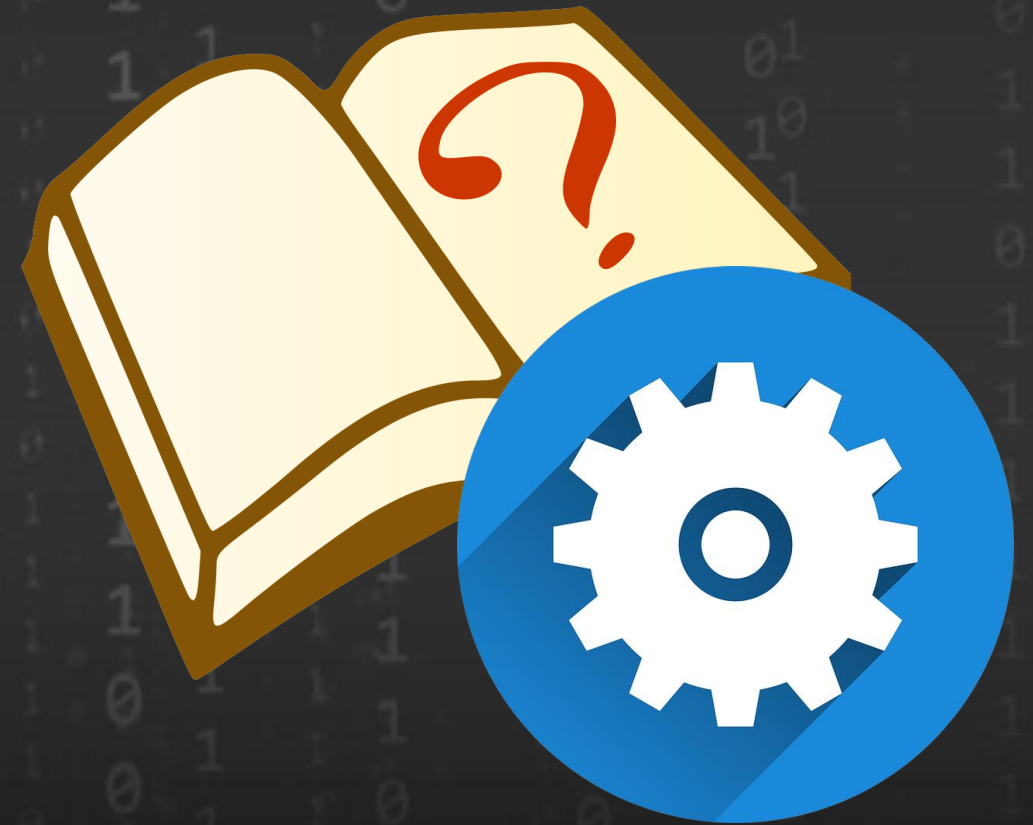
Lab 1: Catch up from previous lab

Skip if Lab <u>Writing UEFI App Lab</u> completed

- Perform <u>Lab Setup</u> from previous Labs
- Create a Directory under the workspace `~/src/edk2-ws/edk2 SampleApp`
- Copy contents of `~/FW/LabSampleCode/SampleAppDebug` to `~/src/edk2-ws/edk2/SampleApp`
- Open `~src/edk2-ws/edk2/OvmfPkg/OvmfPkgX64.dsc`
- Add the following to the [Components] section:

```
# Add new modules here
SampleApp/SampleApp.inf
```

- **Save** and **close** the file `OvmfPkgX64.dsc`

Lab 1: Add debug statements to SampleApp

- Open a Terminal Command Prompt and type `cd ~/src/edk2-ws`

```
bash$ export WORKSPACE=$PWD
bash$ export PACKAGES_PATH=$WORKSPACE/edk2:$WORKSPACE/edk2-libc
bash$ cd edk2
bash$ . edksetup.sh
```

- Open ~/src/edk2/SampleApp/SampleApp.c

- Add the following to the include statements at the top of the file after below the last "include" statement:

```
#include <Library/DebugLib.h>
```

Lab 1: Add debug statements to SampleApp

Locate the UefiMain function. Then copy and paste the following code after the "`EFI_INPUT_KEY  KEY;`" statement: and before the first `Print()` statement as shown in the screen shot below:
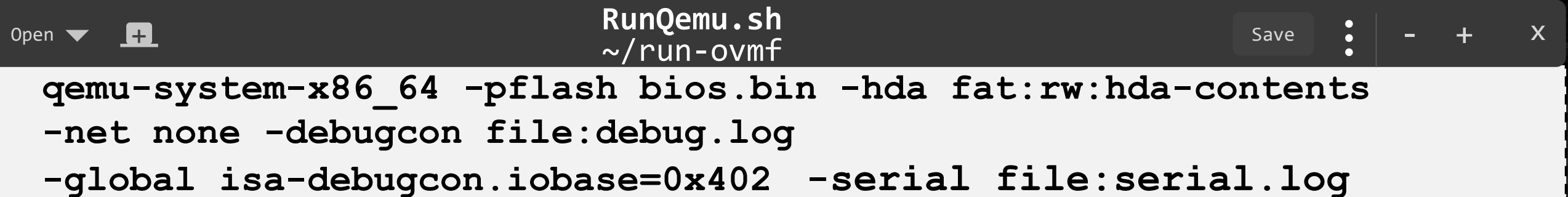
```
DEBUG ((0xffffffff, "\n\nUEFI Base Training DEBUG DEMO\n") );
DEBUG ((0xffffffff, "0xffffffff USING DEBUG ALL Mask Bits Set\n") );

DEBUG ((DEBUG_INIT,     " 0x%08x USING DEBUG DEBUG_INIT\n" , (UINTN)(DEBUG_INIT))  );
DEBUG ((DEBUG_WARN,     " 0x%08x USING DEBUG DEBUG_WARN\n", (UINTN)(DEBUG_WARN))   );
DEBUG ((DEBUG_LOAD,     " 0x%08x USING DEBUG DEBUG_LOAD\n", (UINTN)(DEBUG_LOAD))   );
DEBUG ((DEBUG_FS,       " 0x%08x USING DEBUG DEBUG_FS\n", (UINTN)(DEBUG_FS))   );
DEBUG ((DEBUG_POOL,     " 0x%08x USING DEBUG DEBUG_POOL\n", (UINTN)(DEBUG_POOL))   );
DEBUG ((DEBUG_PAGE,     " 0x%08x USING DEBUG DEBUG_PAGE\n", (UINTN)(DEBUG_PAGE))   );
DEBUG ((DEBUG_INFO,     " 0x%08x USING DEBUG DEBUG_INFO\n", (UINTN)(DEBUG_INFO))   );
DEBUG ((DEBUG_DISPATCH, " 0x%08x USING DEBUG DEBUG_DISPATCH\n",(UINTN)(DEBUG_DISPATCH)));
DEBUG ((DEBUG_VARIABLE, " 0x%08x USING DEBUG DEBUG_VARIABLE\n",(UINTN)(DEBUG_VARIABLE)));
DEBUG ((DEBUG_BM,       " 0x%08x USING DEBUG DEBUG_BM\n", (UINTN)(DEBUG_BM))   );
DEBUG ((DEBUG_BLKIO,    " 0x%08x USING DEBUG DEBUG_BLKIO\n", (UINTN)(DEBUG_BLKIO))  );
DEBUG ((DEBUG_NET,      " 0x%08x USING DEBUG DEBUG_NET\n", (UINTN)(DEBUG_NET))   );
DEBUG ((DEBUG_UNDI,     " 0x%08x USING DEBUG DEBUG_UNDI\n", (UINTN)(DEBUG_UNDI))   );
DEBUG ((DEBUG_LOADFILE, " 0x%08x USING DEBUG DEBUG_LOADFILE\n",(UINTN)(DEBUG_LOADFILE)));
DEBUG ((DEBUG_EVENT,    " 0x%08x USING DEBUG DEBUG_EVENT\n", (UINTN)(DEBUG_EVENT))   );
DEBUG ((DEBUG_GCD,      " 0x%08x USING DEBUG DEBUG_GCD\n", (UINTN)(DEBUG_EVENT)) );
DEBUG ((DEBUG_CACHE,    " 0x%08x USING DEBUG DEBUG_CACHE\n", (UINTN)(DEBUG_CACHE)) );
DEBUG ((DEBUG_VERBOSE,  " 0x%08x USING DEBUG DEBUG_VERBOSE\n", (UINTN)(DEBUG_VERBOSE)) );
DEBUG ((DEBUG_ERROR,    " 0x%08x USING DEBUG DEBUG_ERROR\n", (UINTN)(DEBUG_ERROR))  );
```

Edit the Linux shell script to run the QEMU from the run-ovmf directory and add the option for a serial log

```
bash$ gedit RunQemu.sh
```

RunQemu.sh
~/run-ovmf

Open ▼   +   X   Save ⋮ − + X

```
qemu-system-x86_64 -pflash bios.bin -hda fat:rw:hda-contents
-net none -debugcon file:debug.log
-global isa-debugcon.iobase=0x402  -serial file:serial.log
```

Save and Exit

Build SampleApp – Cd to ~/src/edk2 dir

```
bash$ build
```

Copy the OVMF.fd to the run-ovmf directory naming it bios.bin

```
bash$ cd ~/run-ovmf
bash$ cp ~/src/edk2-ws/Build/OvmfX64/DEBUG_GCC5/FV/OVMF.fd bios.bin
```

Copy SampleApp.efi to hda-contents

```
bash$ cd ~/run-ovmf/hda-contents
bash$ cp ~/src/edk2-ws/Build/OvmfX64/DEBUG_GCC5/X64/SampleApp.efi .
```

Lab 1: Run the Qemu Script

tianocore

Test by Invoking Qemu

```
bash$ cd ~/run-ovmf
bash$ . RunQemu.sh
```

Run the application from the shell

```
Shell> SampleApp
```

Check the contents of the debug.log file

```
bash$ cat debug.log
```

Exit QEMU

debug.log file

```
Loading driver at 0x00006803000 EntryPoint=0x000068045B4 SampleApp.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 6B1B518
ProtectUefiImageCommon - 0x68170C0
  - 0x0000000006803000 - 0x0000000000002C80
InstallProtocolInterface: 752F3136-4E16-4FDC-A22A-E5F46812F4CA 7EA26F8

UEFI Base Training DEBUG DEMO
0xffffffff USING DEBUG ALL Mask Bits Set
 0x00000001 USING DEBUG EFI_D_INIT
 0x00000002 USING DEBUG EFI_D_WARN
 0x00000004 USING DEBUG EFI_D_LOAD
 0x00000008 USING DEBUG EFI_D_FS
 0x00000040 USING DEBUG EFI_D_INFO
 0x80000000 USING DEBUG EFI_D_ERROR
u-uefi@uuefi-TPad:~/run-ovmf$
```

Lab 2 – Changing PCD Value

In this lab, you'll learn how to use PCD values to change debugging capabilities.
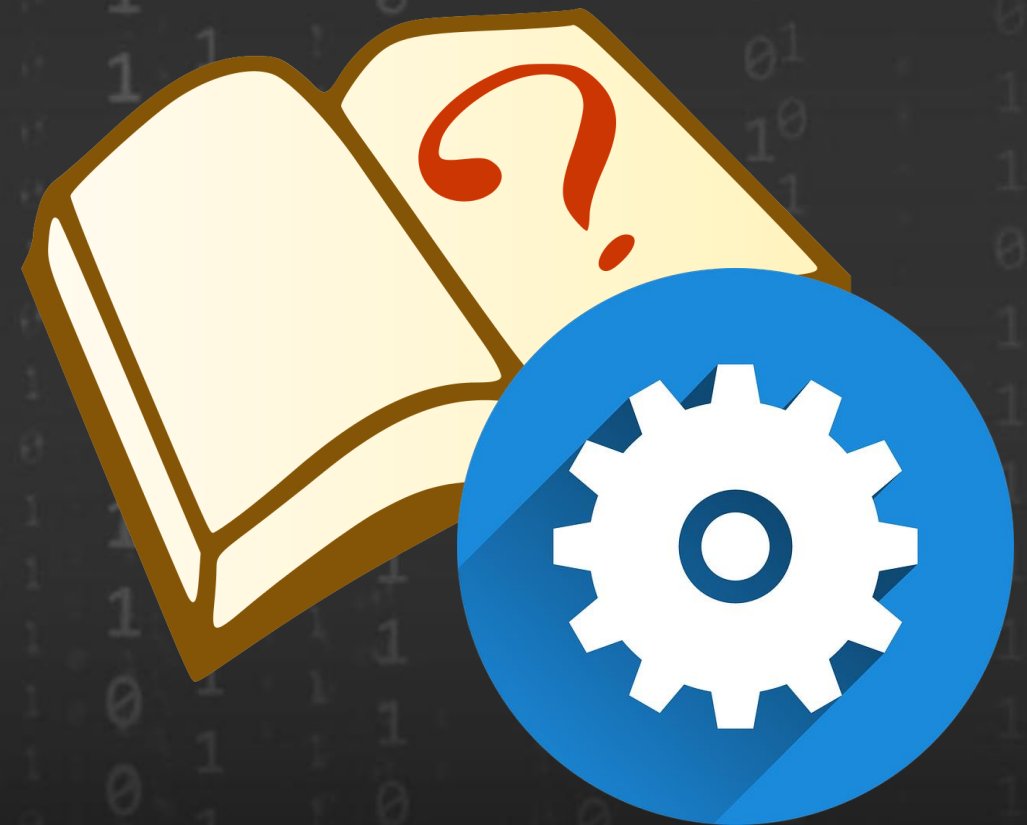
Lab 2: Change PCDs for SampleApp

Open `~src/edk2-ws/OvmfPkg/OvmfPkgX64.dsc`
Replace `SampleApp/SampleApp.inf` with the following:

```
SampleApp/SampleApp.inf {
    <PcdsFixedAtBuild>
    gEfiMdePkgTokenSpaceGuid.PcdDebugPropertyMask|0xff
    gEfiMdePkgTokenSpaceGuid.PcdDebugPrintErrorLevel|0xffffffff
 }
```

Save and close `~src/edk2/OvmfPkg/OvmfPkgX64.dsc`
Build SampleApp : `bash$ build`
Copy SampleApp.efi to hda-contents

```
bash$ cd ~/run-ovmf/hda-contents
bash$ cp ~/src/edk2/Build/OvmfX64/DEBUG_GCC5/X64/SampleApp.efi .
```

Lab 2: Run the Qemu Script

tianocore

Test by Invoking Qemu

```
bash$ cd ~/run-ovmf
bash$ . RunQemu.sh
```

Run the application from the shell

```
Shell> SampleApp
```

Check the contents of the debug.log file

```
bash$ cat debug.log
```

Exit QEMU

debug.log file

```
- 0x0000000006803000 - 0x0000000000002C80
InstallProtocolInterface: 752F3136-4E16-4FDC-A22A-E5F46812F4CA 7EA2

UEFI Base Training DEBUG DEMO
0xffffffff USING DEBUG ALL Mask Bits Set
0x00000001 USING DEBUG EFI_D_INIT
0x00000002 USING DEBUG EFI_D_WARN
0x00000004 USING DEBUG EFI_D_LOAD
0x00000008 USING DEBUG EFI_D_FS
0x00000010 USING DEBUG EFI_D_POOL
0x00000020 USING DEBUG EFI_D_PAGE
0x00000040 USING DEBUG EFI_D_INFO
0x00000080 USING DEBUG EFI_D_DISPATCH
0x00000100 USING DEBUG EFI_D_VARIABLE
0x00000400 USING DEBUG EFI_D_BM
0x00001000 USING DEBUG EFI_D_BLKIO
0x00004000 USING DEBUG EFI_D_NET
0x00010000 USING DEBUG EFI_D_UNDI
0x00020000 USING DEBUG EFI_D_LOADFILE
0x00080000 USING DEBUG EFI_D_EVENT
0x80000000 USING DEBUG EFI_D_ERROR
u-uefi@uuefi-TPad:~/run-ovmf$
```

CHANGING FLAGS

Changing Compiler & Linker Flags

Precedence for Debug Flags Hierarchy

DSC [BuildOptions] section
...orm

INF [BuildOptions] section

DSC <BuildOptions> under a specific module

1. Tools_def.txt
2. DSC [BuildOptions] section (platform scope)
3. INF [BuildOptions] section (module scope)
4. DSC <BuildOptions> under a specific module

Example from Microsoft* compiler to turn off optimization

"/O2" to "/O1" requires "/Od /O1" flags

Change common flags in platform DSC

```
[BuildOptions]
    DEBUG_*_IA32_CC_FLAGS = /Od /Oy-
```

Change a single module's flags in DSC

```
MyPath/MyModule.inf {
<BuildOptions>
    DEBUG_*_IA32_CC_FLAGS = /Od /Oy-
}
```

DebugLib USAGE

The DebugLib Class

Interface

MdePkg\Include\Library\DebugLib.h

Macros
(where PCDs are checked)

```
    ASSERT (Expression)

    DEBUG (Expression)

    ASSERT_EFI_ERROR (StatusParameter)

    ASSERT_PROTOCOL_ALREADY_INSTALLED(…)
```

Advanced Macros

```
    DEBUG_CODE (Expression)

    DEBUG_CODE_BEGIN() & DEBUG_CODE_END()

    DEBUG_CLEAR_MEMORY(…)
```

Implementation

DebugLib Instances (1)

BaseDebugLibSerialPort

- Instance of `DebugLib`
- Uses `SerialPortLib` class to send debug output to serial port
- Default for many platforms: `BaseDebugLibNull`
- OVMF uses it with Switch `DEBUG_ON_SERIAL_PORT`

1

Implementation

DebugLib Instances (2)

`UefiDebugLibConOut` `UefiDebugLibStdErr`

- Instances of `DebugLib` (for apps and drivers)

- Send all debug output to console/debug console

2

28

Implementation

DebugLib Instances (3)

PeiDxeDebugLibReportStatusCode

- Sends ASCII String specified by Description Value to the `ReportStatusCode()`
- May also use the `SerialPortLib` class to send debug output to serial port
- `BaseDebugLibNull` - Resolves references

Default for most platforms

3

Change common library instances in the platform DSC by module type

```
[LibraryClasses.common.IA32]
DebugLib|MdePkg/Library/BaseDebugLibNull/BaseDebugLibNull.inf
```

Change a single module's library instance in the platform DSC

```
MyPath/MyModule.inf {
<LibraryClasses>
DebugLib|MdePkg/Library/BaseDebugLibSerialPort.inf
}
```

Lab 2 – Library Instances for Debugging

In this lab, you'll learn how to add specific debug library instances.
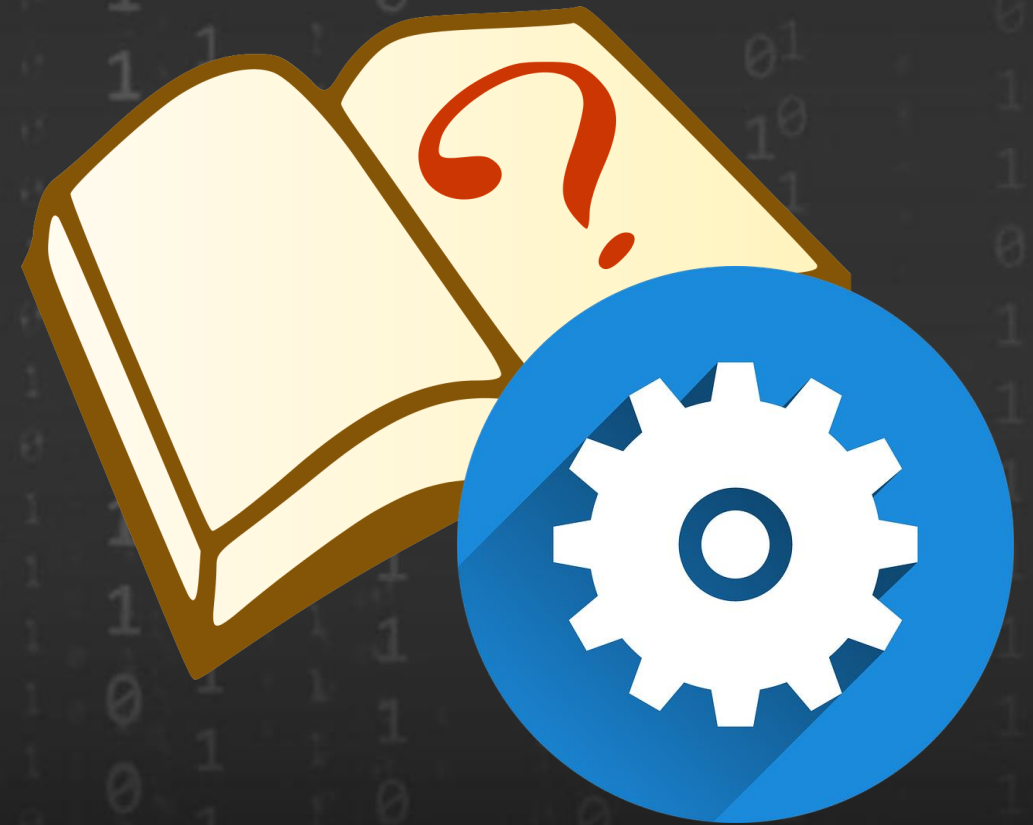
Lab 3: Using Library Instances for Debugging

Open `~src/edk2-ws/edk2/OvmfPkg/OvmfPkgX64.dsc`
Replace `SampleApp/SampleApp.inf { . . .}` with the following:

```
SampleApp/SampleApp.inf {
    <LibraryClasses>
     DebugLib|MdePkg/Library/UefiDebugLibConOut/UefiDebugLibConOut.inf
 }
```

Save and close `~src/edk2-ws/edk2/OvmfPkg/OvmfPkgX64.dsc`
Build SampleApp – `Cd to ~/src/edk2-ws/edk2`   `bash$ build`

Copy SampleApp.efi to hda-contents
```
bash$ cd ~/run-ovmf/hda-contents
bash$ cp ~/src/edk2-ws/Build/OvmfX64/DEBUG_GCC5/X64/SampleApp.efi .
```

Test by Invoking Qemu

```
bash$ cd ~/run-ovmf
bash$ . RunQemu.sh
```

Run the application from the shell

```
Shell> SampleApp
```

See that the output from the Debug
statements now goes to the QEMU
console

Exit QEMU

Debug output to console

```
Shell>
Shell> sampleapp

UEFI Base Training DEBUG DEMO
0xffffffff USING DEBUG ALL Mask Bits Set
 0x00000001 USING DEBUG EFI_D_INIT
 0x00000002 USING DEBUG EFI_D_WARN
 0x00000004 USING DEBUG EFI_D_LOAD
 0x00000008 USING DEBUG EFI_D_FS
 0x00000040 USING DEBUG EFI_D_INFO
 0x80000000 USING DEBUG EFI_D_ERROR
System Table: 0x07E33018

Press any Key to continue :

Enter text. Include a dot ('.') in a sentence then <Enter> to exit

.
Shell> _
```

Lab 4: Serial port Instance of DebugLib

In this lab, you'll change the
`DebugLib` to the Serial port instance.
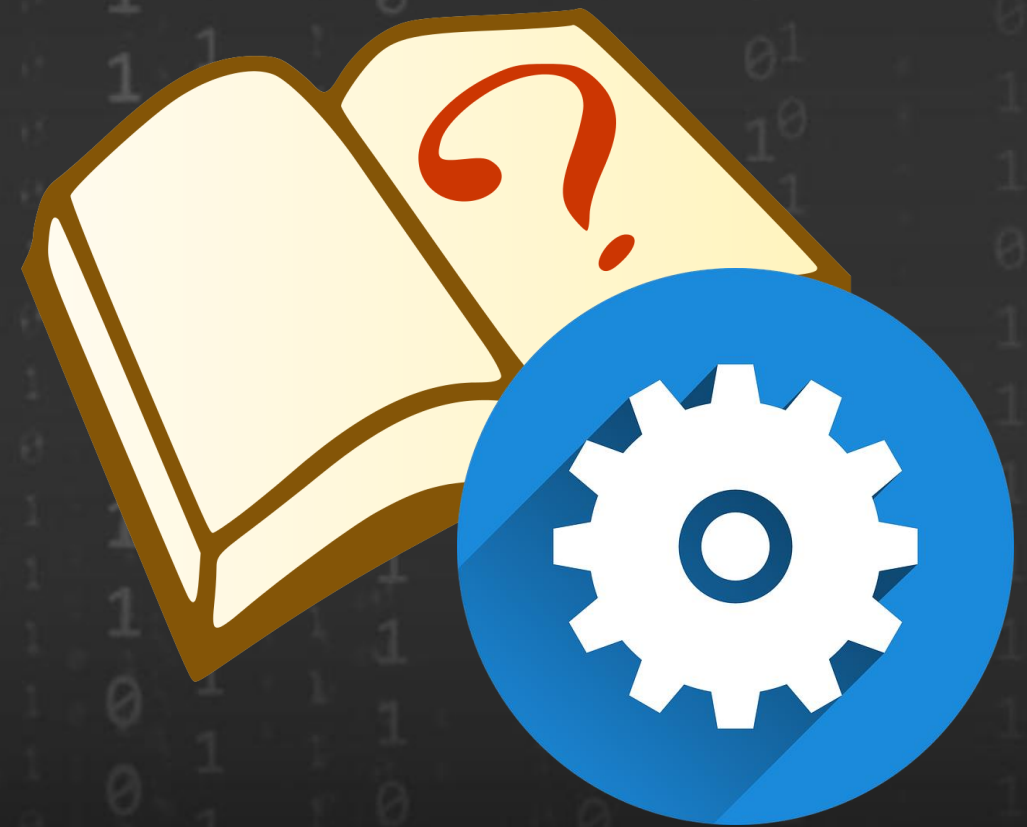
Lab 4: Using Serial port Library Instances

tianocore

Open `~src/edk2-ws/edk2/OvmfPkg/OvmfPkgX64.dsc`
Replace `SampleApp/SampleApp.inf { . . .}` with the following:

```
SampleApp/SampleApp.inf {
    <LibraryClasses>
      DebugLib|MdePkg/Library/BaseDebugLibSerialPort/BaseDebugLibSerialPort.inf
 }
```

Save and close `~src/edk2-ws/edk2/OvmfPkg/OvmfPkgX64.dsc`
Build SampleApp – `Cd to ~/src/edk2-ws/edk2` `bash$ build`

Copy SampleApp.efi to hda-contents

```
bash$ cd ~/run-ovmf/hda-contents
bash$ cp ~/src/edk2-ws/Build/OvmfX64/DEBUG_GCC5/X64/SampleApp.efi .
```

tianocore

Test by Invoking Qemu

```
bash$ cd ~/run-ovmf
bash$ .  RunQemu.sh
```

serial.log file

Run the application from the shell

```
Shell> SampleApp
```

Check the contents of the
debug.log file

```
bash$ cat serial.log
```

Exit QEMU

```
Loading driver at 0x00006803000 EntryPoint=0x000068045B4 SampleApp.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 6B1B518
ProtectUefiImageCommon - 0x68170C0
  - 0x0000000006803000 - 0x0000000000002C80
InstallProtocolInterface: 752F3136-4E16-4FDC-A22A-E5F46812F4CA 7EA26F8

UEFI Base Training DEBUG DEMO
0xffffffff USING DEBUG ALL Mask Bits Set
 0x00000001 USING DEBUG EFI_D_INIT
 0x00000002 USING DEBUG EFI_D_WARN
 0x00000004 USING DEBUG EFI_D_LOAD
 0x00000008 USING DEBUG EFI_D_FS
 0x00000040 USING DEBUG EFI_D_INFO
 0x80000000 USING DEBUG EFI_D_ERROR
u-uefi@uuefi-TPad:~/run-ovmf$
```

Lab 5: Debugging EDK II with GDB

In this lab, you'll learn how setup the Linux GDB to use with EDK II and Qemu
See also the tianocore.org wiki page:
How to use GDB with QEMU.
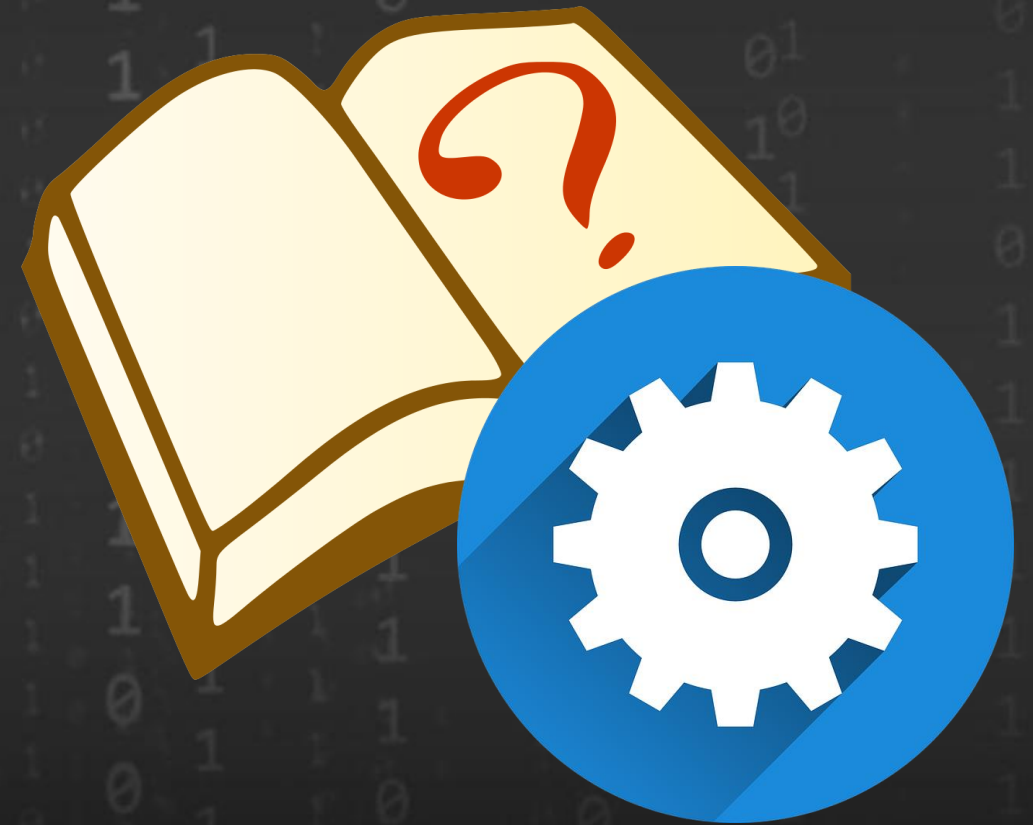
tianocore

Edit the Linux shell script to run the QEMU from the run-ovmf directory and add the option for GDB "-s" to generate a symbol file and also use IA32 instead of x86_64

```
bash$ cd ~/run-ovmf
bash$ gedit RunQemu.sh
```

add the following to RunQemu.sh

```
qemu-system-i386  -s  -pflash bios.bin -hda fat:rw:hda-contents -net
none -debugcon file:debug.log -global isa-debugcon.iobase=0x402
```

Save and Exit

Open `~src/edk2-ws/edk2/OvmfPkg/OvmfPkgIa32.dsc` and add the application to the end of the [Components] section:

```
[Components]
# add at the end of the components section OvmfPkgIa32.dsc
  SampleApp/SampleApp.inf
```

Save and close `~src/edk2-ws/edk2/OvmfPkg/OvmfPkgIa32.dsc`

Build OVMF for IA32

```
bash$ build -a IA32 -p OvmfPkg/OvmfPkgIa32.dsc
```

Copy the the OVMF.fd to the run-ovmf directory renaming it bios.bin:

```
bash$ cd ~/run-ovmf/
bash$ cp ~/src/edk2-ws/Build/OvmfIa32/DEBUG_GCC5/FV/OVMF.fd bios.bin
```

Lab 5.3: Build Ovmf for IA32

Copy the output of SampleApp to the hda-contents directory:

```
bash$ cd ~/run-ovmf/hda-contents
bash$ cp ~/src/edk2-ws/Build/OvmfIa32/DEBUG_GCC5/IA32/SampleApp  .
```

The following will be in the ~/run-ovmf/hda-contents/

```
SampleApp.efi
SampleApp.debug
SampleApp (Directory)
```

Open a Terminal(1) Prompt and Invoke Qemu

```
bash$ cd ~/run-ovmf
bash$ . RunQemu.sh
```

Run the application from the shell

```
Shell> SampleApp
```

tianocore

Open another Terminal(2) Prompt in the run-ovmf directory and check the debug.log file.

```
bash$ cd ~/run-ovmf
bash$ cat debug.log
```

See the line: `Loading driver at 0x00006AEE000` is the memory location where your UEFI Application is loaded.

```
 InstallProtocolInterface: 5B1B31A1-9562-11D2-8E3F-00A0C969723B 6F0F028
Loading driver at 0x00006AEE000 EntryPoint=0x00006AEE756 SampleApp.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 6F0FF10
```

tianocore

Add a DEBUG statement to your SampleApp.c application to get the entry point of your code.

Add the following DEBUG line just before the DEBUG statements from the previous lab:

```
UefiMain (
// . . .
    EFI_INPUT_KEY        Key;
    // ADD the following line
    DEBUG ((EFI_D_INFO, "My Entry point: 0x%p\r\n", (CHAR16*)UefiMain )  );
```

When you print out the debug.log again, the exact entry point for your code will show.

This is useful to double check symbols are fixed up to the correct line numbers in the source file.

```
Loading driver at 0x00006AEE000 EntryPoint=0x00006AEE756 SampleApp.efi
InstallProtocolInterface: BC62157E-3E33-4FEC-9920-2D3B36D750DF 6F0FF10
ProtectUefiImageCommon - 0x6F0F028
  - 0x0000000006AEE000 - 0x0000000000002B00
InstallProtocolInterface: 752F3136-4E16-4FDC-A22A-E5F46812F4CA 7EA4B00
My Entry point: 0x06AEE496
```

In the terminal(2) prompt Invoke GDB (note - at first there will be nothing in the source window)

```
bash$ cd ~/run-ovmf/hda-contents
bash$ gdb --tui
```

Load your UEFI Application SampleApp.efi with the "file" command.

```
(gdb) file SampleApp.efi
Reading symbols from SampleApp.efi...(no debugging symbols found)...done.
```

Check where GDB has for ".text" and ".data" offsets with "info files" command.

```
(gdb) info files
Symbols from "/home/u-mypc/run-ovmf/hda-contents/SampleApp.efi".
Local exec file:
        `/home/u-mypc/run-ovmf/hda-contents/SampleApp.efi',
        file type pei-i386.
        Entry point: 0x756
        0x00000240 - 0x000028c0 is .text
        0x000028c0 - 0x00002980 is .data
        0x00002980 - 0x00002b00 is .reloc
```

Lab 5.7: Calculate Addresses

tianocore

We need to calculate our addresses for ".text" and ".data" section.
The application is loaded under `0x00006AEE000` (loading driver point - NOT Entrypoint) and we know text and data offsets.

```
text = 0x00006AEE000   +   0x00000240 = 0x06AEE240
data = 0x00006AEE000   +   0x00000240 + 0x000028c0 = 0x06AF0B00
```

Unload the .efi file

```
(gdb) file
No executable file now.
No symbol file now.
```

tianocore

Load the symbols with the fixed up address using SampleApp output .debug file using the "**add-symbol-file**" command:

```
(gdb) add-symbol-file SampleApp.debug 0x06AEE240 -s .data 0x06AF0B00
add symbol table from file "SampleApp.debug" at
        .text_addr = 0x6aee240
        .data_addr = 0x6af0b00
(y or n) y
Reading symbols from SampleApp.debug...done.
```

Set a break point at UefiMain

```
(gdb) break UefiMain
Breakpoint 1 at 0x6aee496: file /home/u-uefi/src/edk2/SampleApp/SampleApp.c, line 40.
```

Attach the GDB debugger to QEMU

```
(gdb) target remote localhost:1234
Remote debugging using localhost:1234
0x07df6ba4 in ?? ()
```

Continue in GDB

```
(gdb) c
Continuing.
```
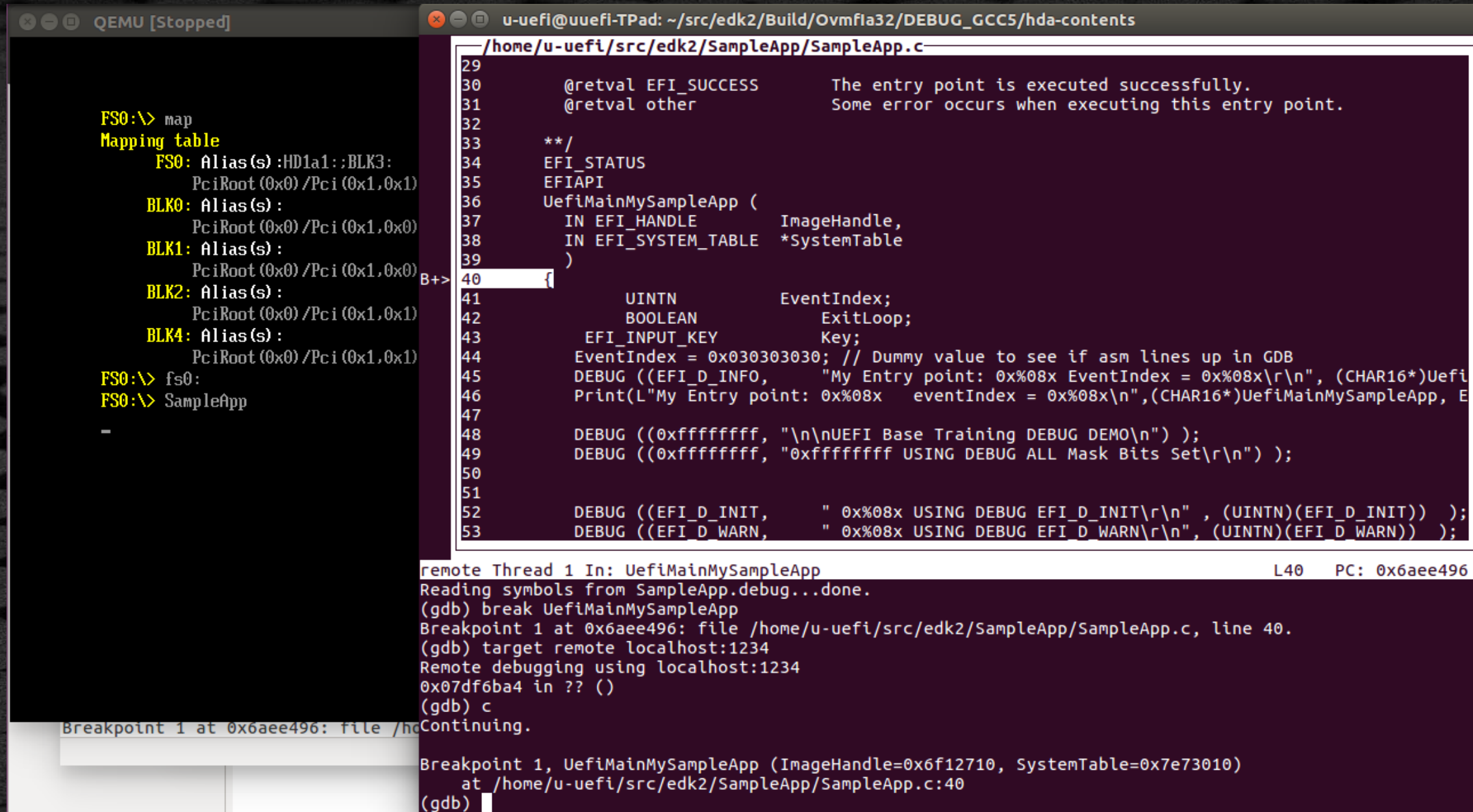
In the QEMU Window Invoke your application again

```
Fs0:\> SampleApp.efi
```

The GDB will hit your break point in your UEFI application's entry point and you can begin to debug with source code debugging

tianocore

The GDB window will look similar to this

SUMMARY

- Define `DebugLib` and its attributes
- List the ways to debug
- Using PCDs to Configure `DebugLib` - LAB
- Change Compiler & Linker Flags for debugging
- Change the `DebugLib` instance to modify the debug output - LAB
- Debug EDK II using GDB - LAB

Questions?

Return to Main Training Page

Return to Training Table of contents for next presentation link