tianocore

UEFI & EDK II Training

EDK II Debugging with Linux Lab

tianocore.org

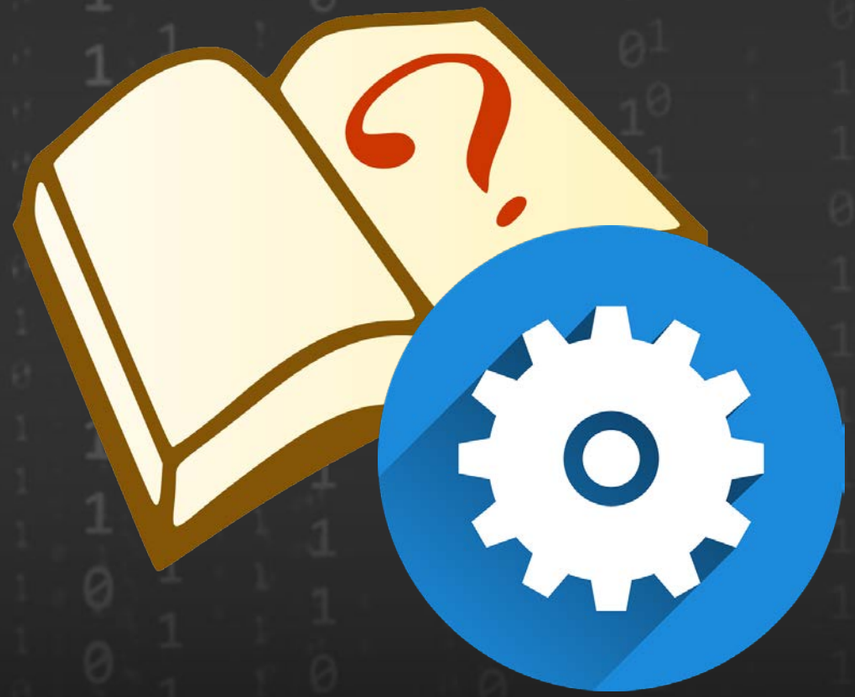
Copy and Paste LabGuide.md

Lesson Objective

- Using PCDs to Configure `DebugLib` - LAB
- Change the `DebugLib` instance to modify the debug output - LAB
- Debug EDK II using GDB - LAB

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 to next slide if Writing UEFI App Lab completed (Lab Guide)

- Perform Lab Setup from previous Labs (Lab Guide)
- 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

```
bash$ 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: (LabGuide.md for copy and paste)

```
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))  );
```

Lab 1: Update the Qemu Script

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
```

Open | RunQemu.sh ~/run-ovmf | Save

```
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
```

See contents of `~/FW/edk2Linux/SRunQemu.sh`
(copy and paste to `~/run-ovmf/RunQemu.sh`)

Save and Exit

Lab 1: Build and Test Application

Build SampleApp – Cd to ~/src/edk2-ws/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

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
```

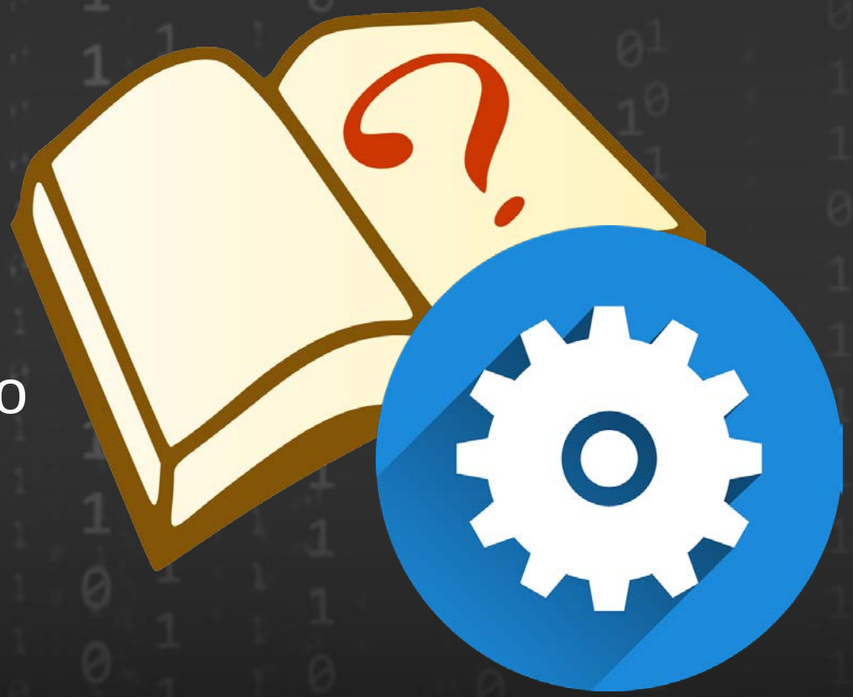
Exit QEMU

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

Test by Invoking Qemu

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

Run the application from the shell

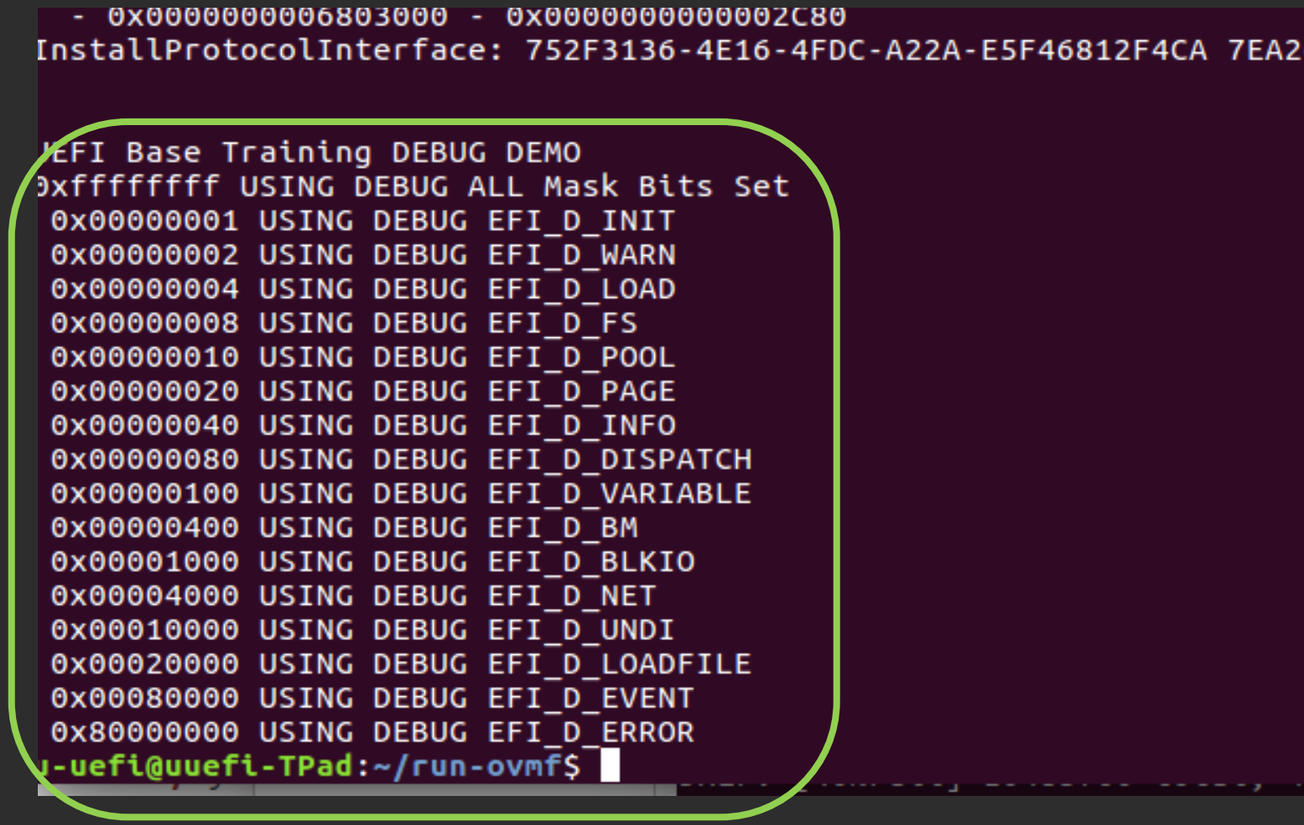
```
Shell> SampleApp
```

Check the contents of the debug.log file

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

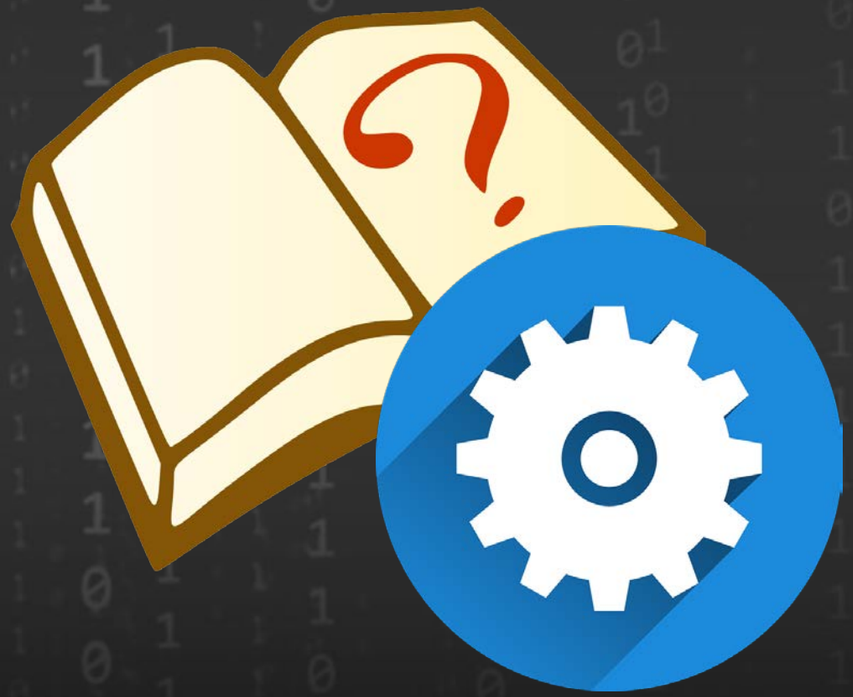
Exit QEMU

debug.log file



Lab 3 – 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 .
```

Lab 3: Run the Qemu Script

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

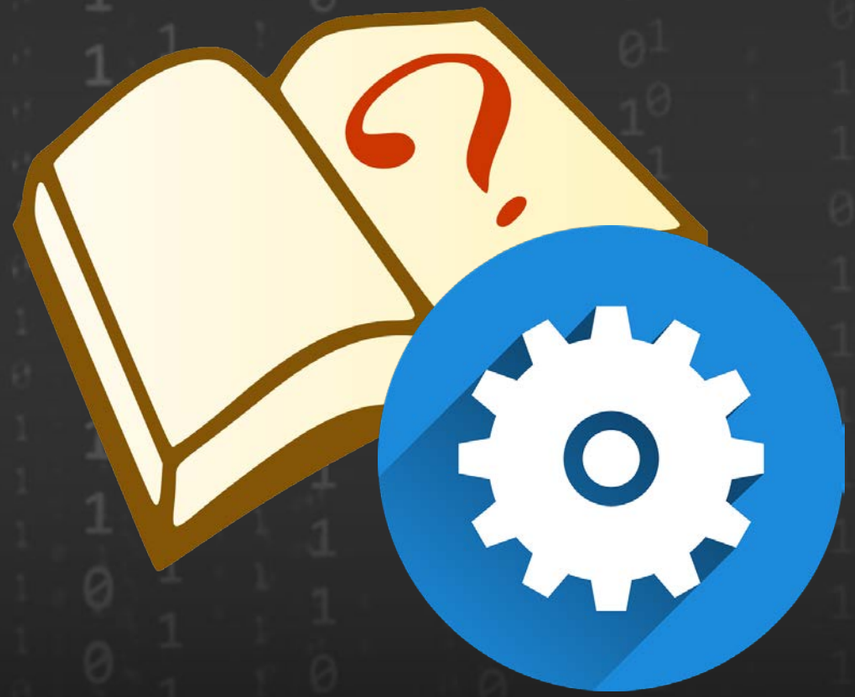
Debug output to console

```
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> _
```

tianocore

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

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 .
```

Lab 4: Run the Qemu Script

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 serial.log
```

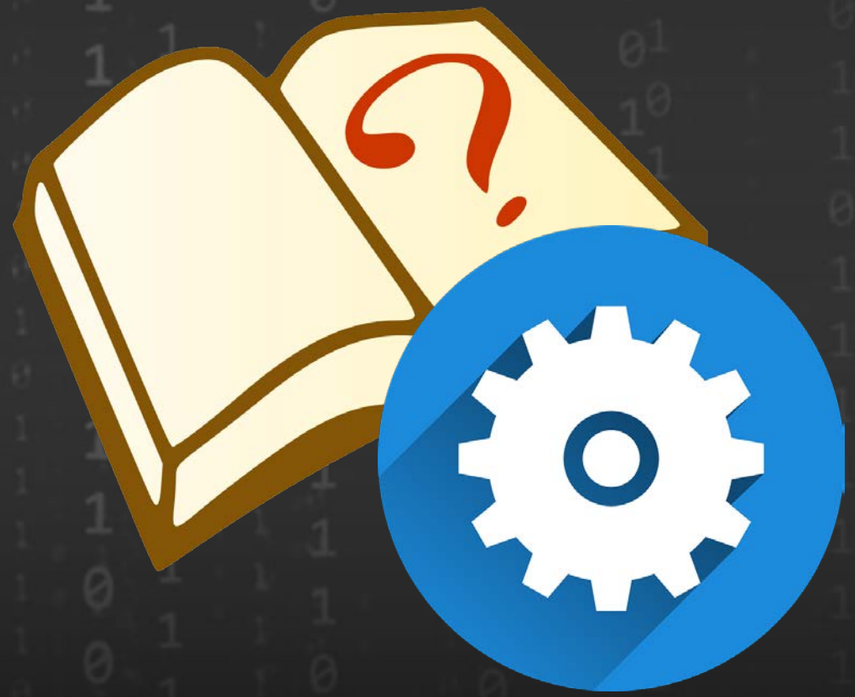
Exit QEMU

serial.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$
```

tianocore

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.

Lab 5.1: Update the Qemu Script

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 “-s” to the following to RunQemu.sh (Note, this is for the IA32 and the –serial is not there.)

```
  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

Lab 5.2: Build Ovmf for IA32

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
```

Lab 5.4: Check debug.log

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
```

Lab 5.5: Add a Debug Print

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
```

Lab 5.6: Invoking GDB

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

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.
```

Lab 5.8: Load the Symbols for SampleApp

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.
```

Lab 5.9: Attach GDB to QEMU

Attach the GDB debugger to QEMU

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

Continue in GDB

```
(gdb) c
Continuing.
```

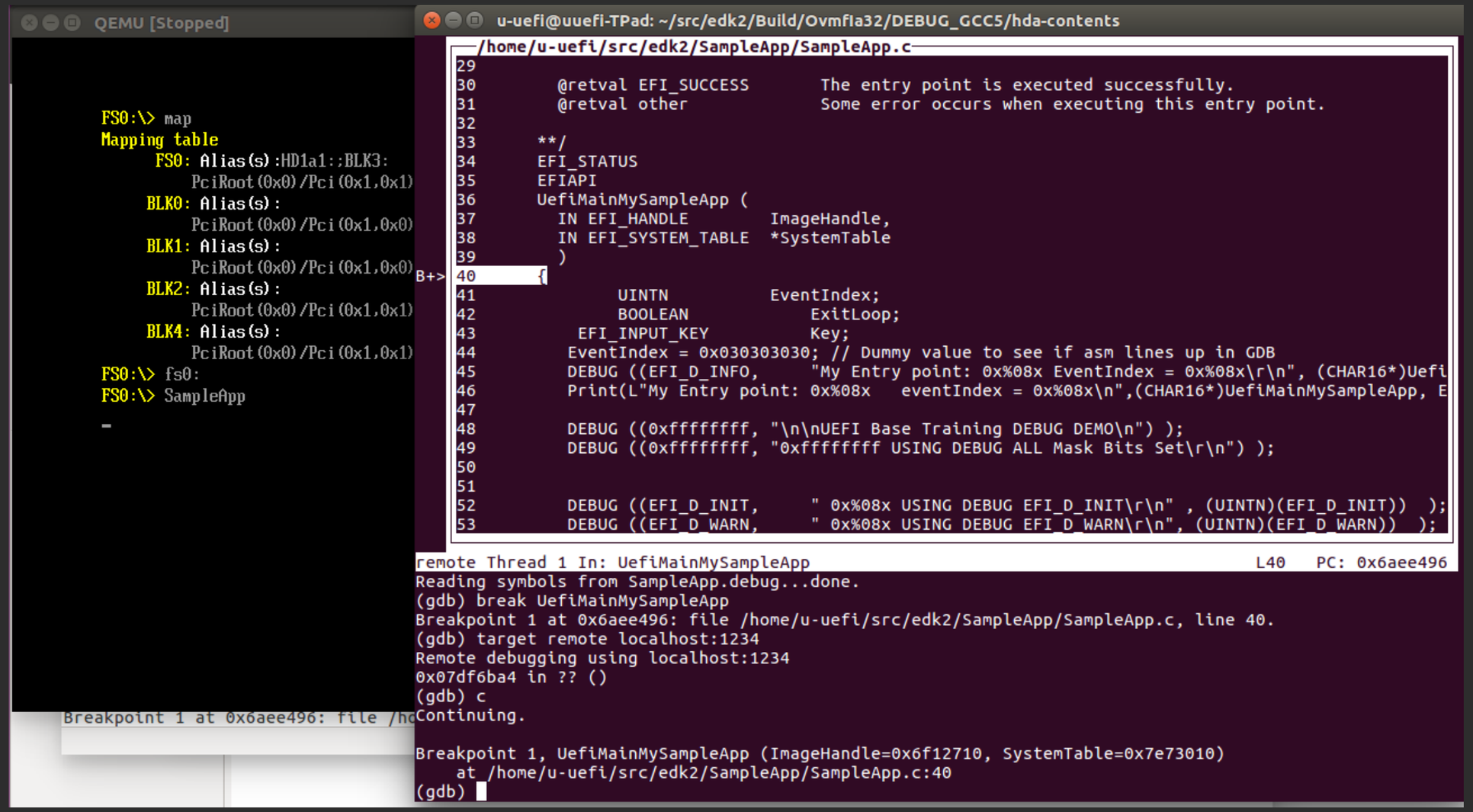
In the QEMU Window Invoke your application again

```
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

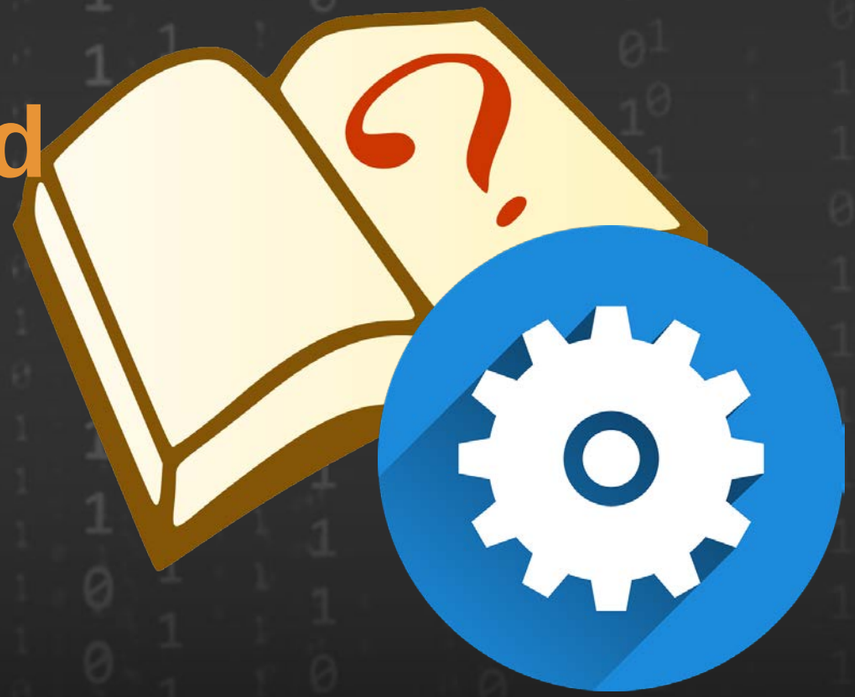
Lab 5: GBD and QEMU Windows

The GDB window will look similar to this



Lab 6: Debugging EDK II add Debug to Boot Flow

In this lab, you'll learn how add Debug statements to the EDK II Boot flow and check the debug log output



Lab 6: Debug Boot Flow

Edit the MdeModulePkg/Core/Pei/PeiMain/PeiMain.c and add a “DEBUG” print ~line 489 before the call to the PeiDispatcher:

```
DEBUG((DEBUG_INFO, "**********Before call to Pei Dispatcher ********\n"));
```

Save PeiMain.c

```
486     //
487     // Call PEIM dispatcher
488     //
489     DEBUG((DEBUG_INFO, "**********Before call to Pei Dispatcher ********\n"));
490     PeiDispatcher (SecCoreData, &PrivateData);
491
```

Lab 6: Build and Test Application

Build – Cd to ~/src/edk2-ws/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
```

Lab 6: Run the Qemu Script

Test by Invoking Qemu

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

Check the contents of the debug.log file

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

Exit QEMU

debug.log file

```
Loading PEIM at 0x227A1DC0000 EntryPoint=0x227A1DC1078
PeiCore.efi
Reinstall PPI: 8C8CE578-8A3D-4F1C-9935-896185C32DD3
Reinstall PPI: 5473C07A-3DCB-4DCA-BD6F-1E9689E7349A
Reinstall PPI: B9E0ABFE-5979-4914-977F-6DEE78C278A6
Install PPI: F894643D-C449-42D1-8EA8-85BDD8C65BDE
**********Before call to Pei Dispatcher ********
Loading PEIM 9B3ADA4F-AE56-4C24-8DEA-F03B7558AE50
```

tianocore

Summary

- Using PCDs to Configure `DebugLib` - LAB
- Change the `DebugLib` instance to modify the debug output - LAB
- Debug EDK II using GDB - LAB

tianocore

Questions?

Return to Main Training Page

Return to Training Table of contents for next presentation link

tianocore

ACKNOWLEDGEMENTS