Security of BIOS/UEFI System Firmware from Attacker and Defender Perspectives

Section 5. Hands-On Learning of EFI Environment

Yuriy Bulygin *
Alex Bazhaniuk *
Andrew Furtak *
John Loucaides **

* Advanced Threat Research, McAfee
** Intel

License

Training materials are shared under Creative Commons “Attribution” license CC BY 4.0

Provide the following attribution:

Derived from “Security of BIOS/UEFI System Firmware from Attacker and Defender Perspective” training by Yuriy Bulygin, Alex Bazhaniuk, Andrew Furtak and John Loucaides available at https://github.com/advanced-threat-research/firmware-security-training

Section 5. Hands-On Learning of EFI Environment

5.1 UEFI Shell

Exercise 5.1

Getting Familiar with UEFI Shell

Booting in UEFI Shell and Using Built-in Shell Commands

- Replace bootloader with uefi shell:

```
$cp /boot/efi/EFI/boot/bootx64.efi /boot/efi/EFI/boot/bootx64.efi.bak

$cp /boot/efi/EFI/boot/shell_bootx64.efi /boot/efi/EFI/boot/bootx64.efi
```

- Reboot system: `$shutdown –r now`
- Test built-in shell commands from UEFI shell (list in next slide)
- Recover original bootloader in UEFI shell:

```
shell> fs0:

shell> rm EFI\boot\bootx64.efi

shell> cp EFI\boot\bootx64.efi.bak EFI\boot\bootx64.efi
```

Full UEFI Shell Commands/Apps

Tool	Description
help -b	Displays all UEFI shell internal commands
mode	Displays or changes the console output device mode.
memmap	Displays the memory map maintained by the EFI environment.
dmem	Displays the contents of system or device memory.
mm	Displays or modifies MEM/MMIO/IO/PCI/PCIE address space.
pci	Displays PCI device list or PCI function configuration space.
drivers	Displays the EFI driver list.
dmpstore	Displays all EFI NVRAM variables.
dh	Displays EFI handle information.
openinfo	Displays the protocols and agents associated with a handle.
dblk	Displays the contents of one or more blocks from a block device.
eficompress	Compress a file
efidecompress	Decompress a file
smbiosview	Displays SMBIOS information
loadpcirom	Loads a PCI Option ROM from the specified file.
edit/hexedit	Editor, hex editor
map	Defines a mapping between a user-defined name and a device handle.
vol	Displays the volume information for the file system that is specified by fs.

5.2 Building UEFI Firmware with EDK II

Exercise 5.2

Building EDK2 and flashing SPI image

Exercise Outline

Pre-requirement: Boot your system from USB stick. Connect your system to minnowboard through Ethernet cable and change IP address of your system to 192.168.1.1/24

1. Build open source EDK2 BIOS image for MinnowBoard on your system
2. Copy newly build Flash image to MinnowBoard (use `scp` command for it)
3. Flash it onto SPI using CHIPSEC
4. Read SPI image using CHIPSEC
5. Read SPI image using Dediprog HW SPI Flash programmer (optional)

MinnowBoard MAX Build Resources

Documents, Release Nodes, Pre-built Firmware Binary images, Buildable Development Tree, Flash Update Utilities:

http://firmware.intel.com/projects/minnowboard-max

Using EDK II with Native GCC:

http://tianocore.sourceforge.net/wiki/Using_EDK_II_with_Native_GCC

Create a Full Source Tree

1. Create a new folder (directory) on the root of your local hard drive (development machine) for use as your work space (In USB stick work space: "`/home/user/Desktop/bios`").
2. Checkout packages from: https://svn.code.sf.net/p/edk2/code/branches/UDK2014.SP1/
3. Download: `MinnowBoard_MAX-{version}-Binary.Objects.zip`
4. Download and patch `openssl`
5. Download `edksetup.sh`
6. Patch `Vlv2TbltDevicePkg/bld_vlv.sh` depends on GCC version

Or use script: `bios_download_and_build.sh`

Apply debug patch

- Apply debug patch to `Vlv2TbltDevicePkg` directory:

```
$cd Vlv2TbltDevicePkg

$patch –p 0 < ~/Desktop/patches/debug.patch

$cd ..
```

Build MinnowBoard UEFI Firmware

1. *Install iASL compiler*
2. *Install python, gcc, build-essential, subversion, uuid-dev*
3. Run: `$source edksetup.sh`
4. Run:

   ```
   $cd Vlv2TbltDevicePkg
   $chmod +x bld_vlv.sh Build_IFWI.sh GenBiosId
   ```

5. Build EDKII Firmware:

   `$./Build_IFWI.sh MNW2 Debug` # also use this to rebuild BIOS

6. Firmware binary `MNW2MAX_X64_D_0079_01_GCC.bin` should now be in directory `Stitch/`

   ```
   $ ls -lah Stitch/
   -rw-r--r--  1 user user 8.0M Jun  4 18:06 MNW2MAX_X64_D_0079_01_GCC.bin
   ```

7. Copy SPI image to MinnowBoard system (use `scp` command for coping).

Read SPI Image Using CHIPSEC

Check SPI flash before/after erase and write operations.

CHIPSEC SPI commands:

```
    $ chipsec_util spi dump rom.bin
```

Flash Image Onto SPI Using CHIPSEC

1. Disable BIOS Write Protection in UEFI Setup (DONE)
2. Enable writes to BIOS region of SPI flash memory

```
$python chipsec_util.py spi disable-wp
```

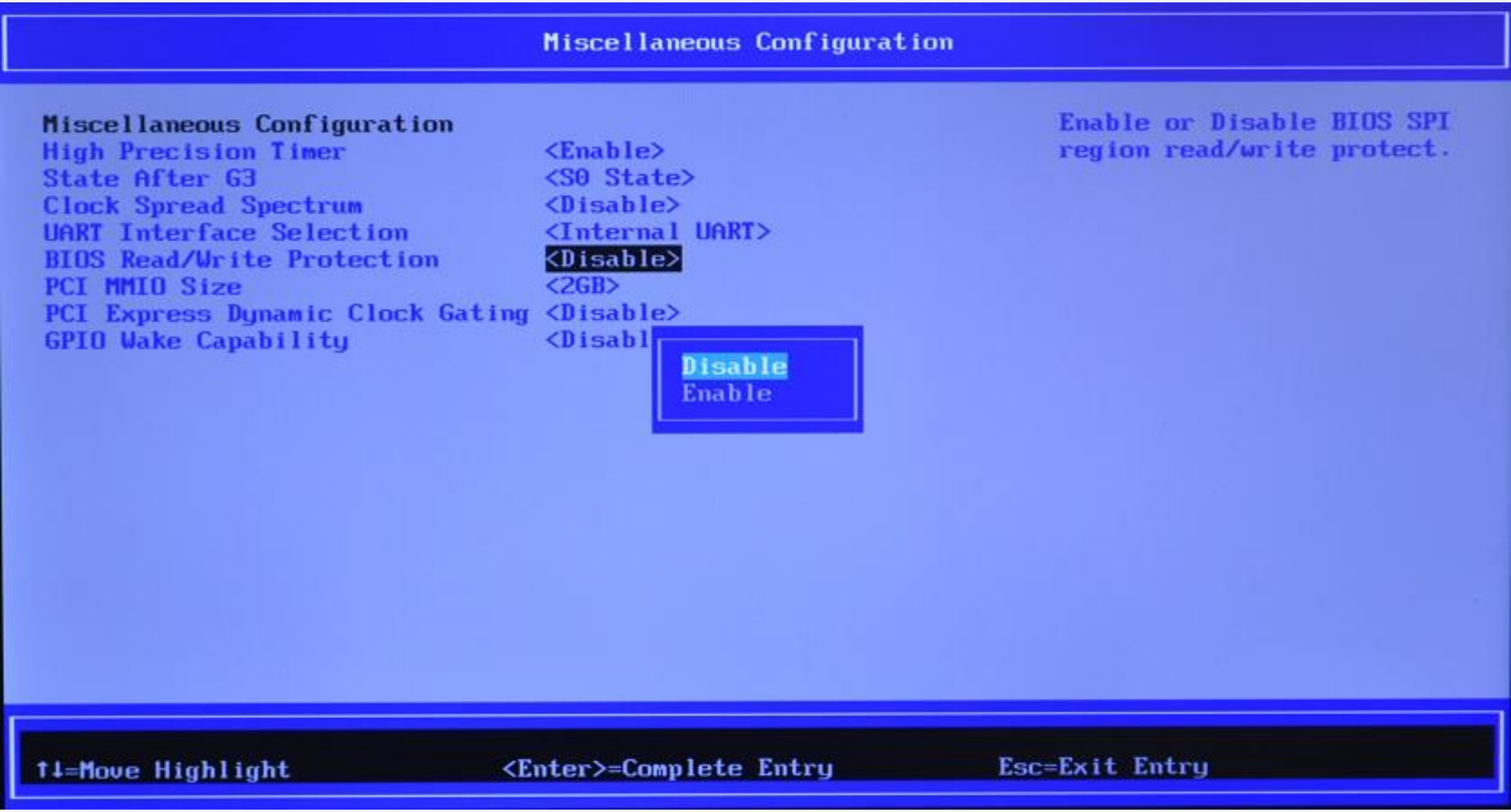
3. Erase full SPI flash memory chip

```
for(( i=0; i<2048; i++ ))
do
   R=$(echo "obase=16; $i*4096" | bc)
   python chipsec_util.py spi erase $R;
done
```

4. Write newly built firmware image to SPI flash memory

```
$python chipsec_util.py spi write 0x0 <NEW_BUILT_BIOS_FILE>
```

Changing BIOS WP in UEFI Setup

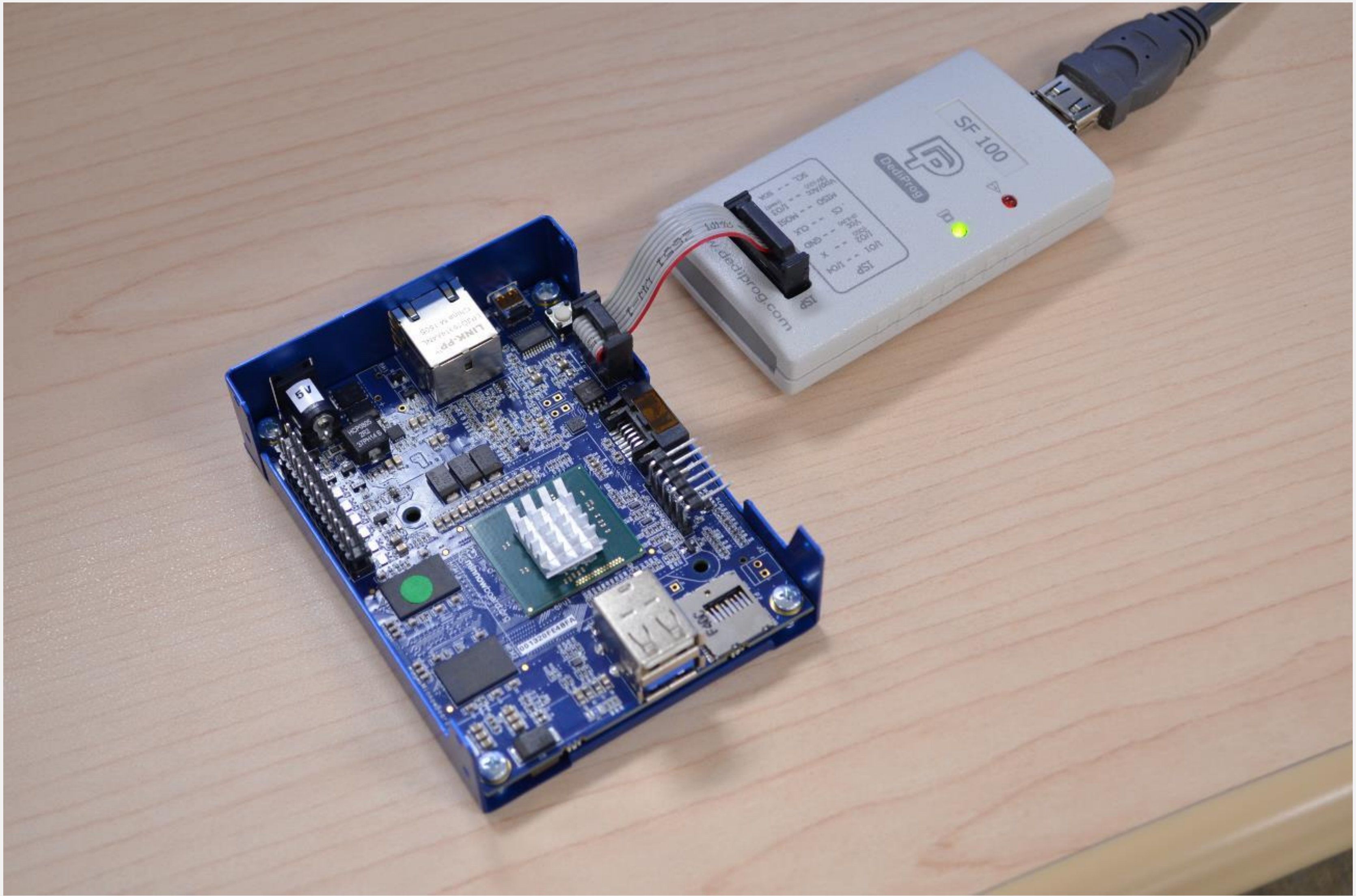


Miscellaneous Configuration

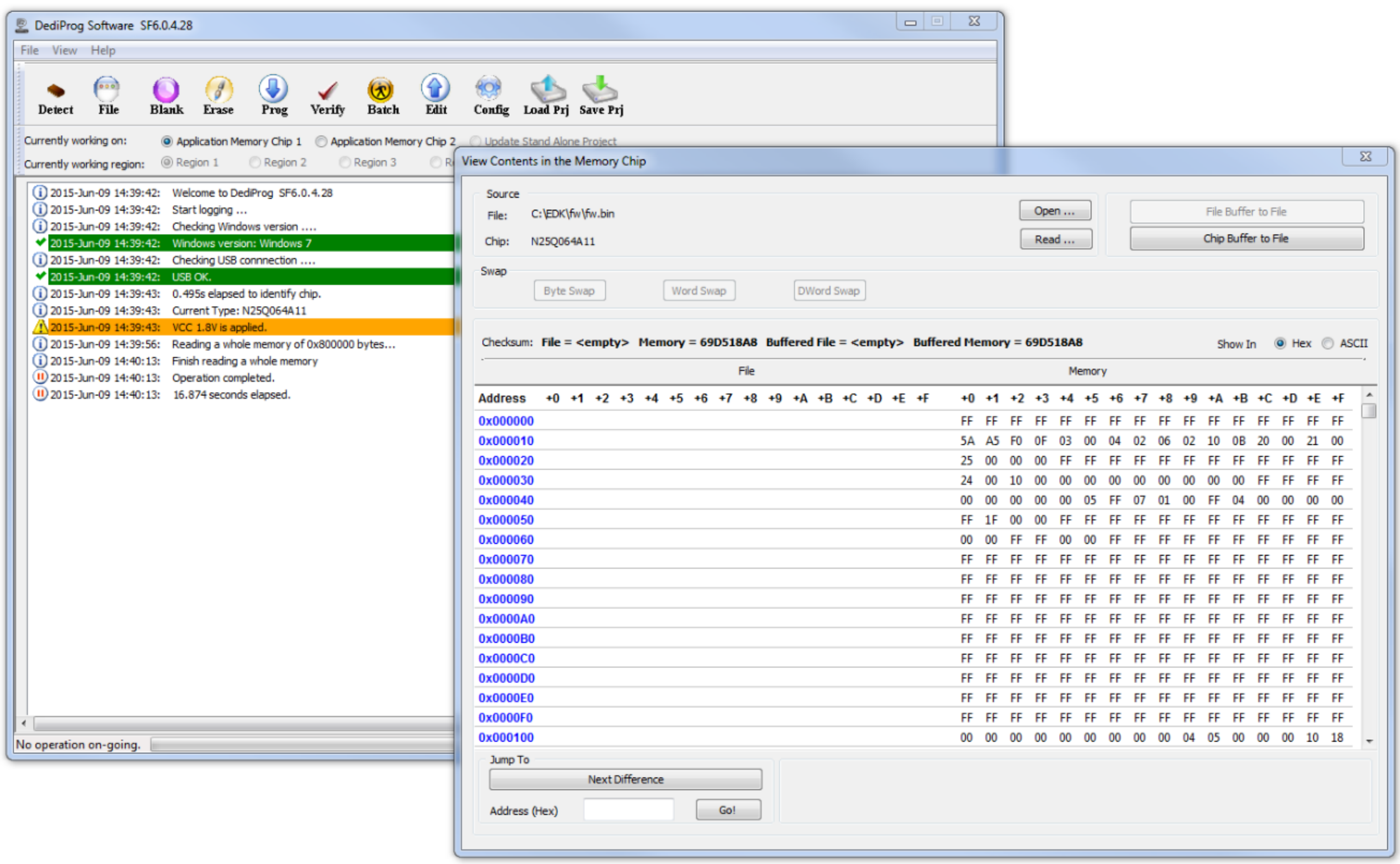
```
Miscellaneous Configuration
High Precision Timer               <Enable>
State After G3                     <S0 State>
Clock Spread Spectrum              <Disable>
UART Interface Selection           <Internal UART>
BIOS Read/Write Protection         <Disable>
PCI MMIO Size                      <2GB>
PCI Express Dynamic Clock Gating   <Disable>
GPIO Wake Capability               <Disabl
```

Enable or Disable BIOS SPI region read/write protect.

Manipulating SPI Image Using Dediprog Hardware SPI Flash Programmer



DediProg Software



Currently working on: ● Application Memory Chip 1 ○ Application Memory Chip 2 ○ Update Stand Alone Project

Currently working region: ● Region 1 ○ Region 2 ○ Region 3

```
2015-Jun-09 14:39:42:   Welcome to DediProg SF6.0.4.28
2015-Jun-09 14:39:42:   Start logging ...
2015-Jun-09 14:39:42:   Checking Windows version ....
2015-Jun-09 14:39:42:   Windows version: Windows 7
2015-Jun-09 14:39:42:   Checking USB connection ....
2015-Jun-09 14:39:42:   USB OK.
2015-Jun-09 14:39:43:   0.495s elapsed to identify chip.
2015-Jun-09 14:39:43:   Current Type: N25Q064A11
2015-Jun-09 14:39:43:   VCC 1.8V is applied.
2015-Jun-09 14:39:56:   Reading a whole memory of 0x800000 bytes...
2015-Jun-09 14:40:13:   Finish reading a whole memory
2015-Jun-09 14:40:13:   Operation completed.
2015-Jun-09 14:40:13:   16.874 seconds elapsed.
```

No operation on-going.

Source

File: C:\EDK\fw\fw.bin

Chip: N25Q064A11

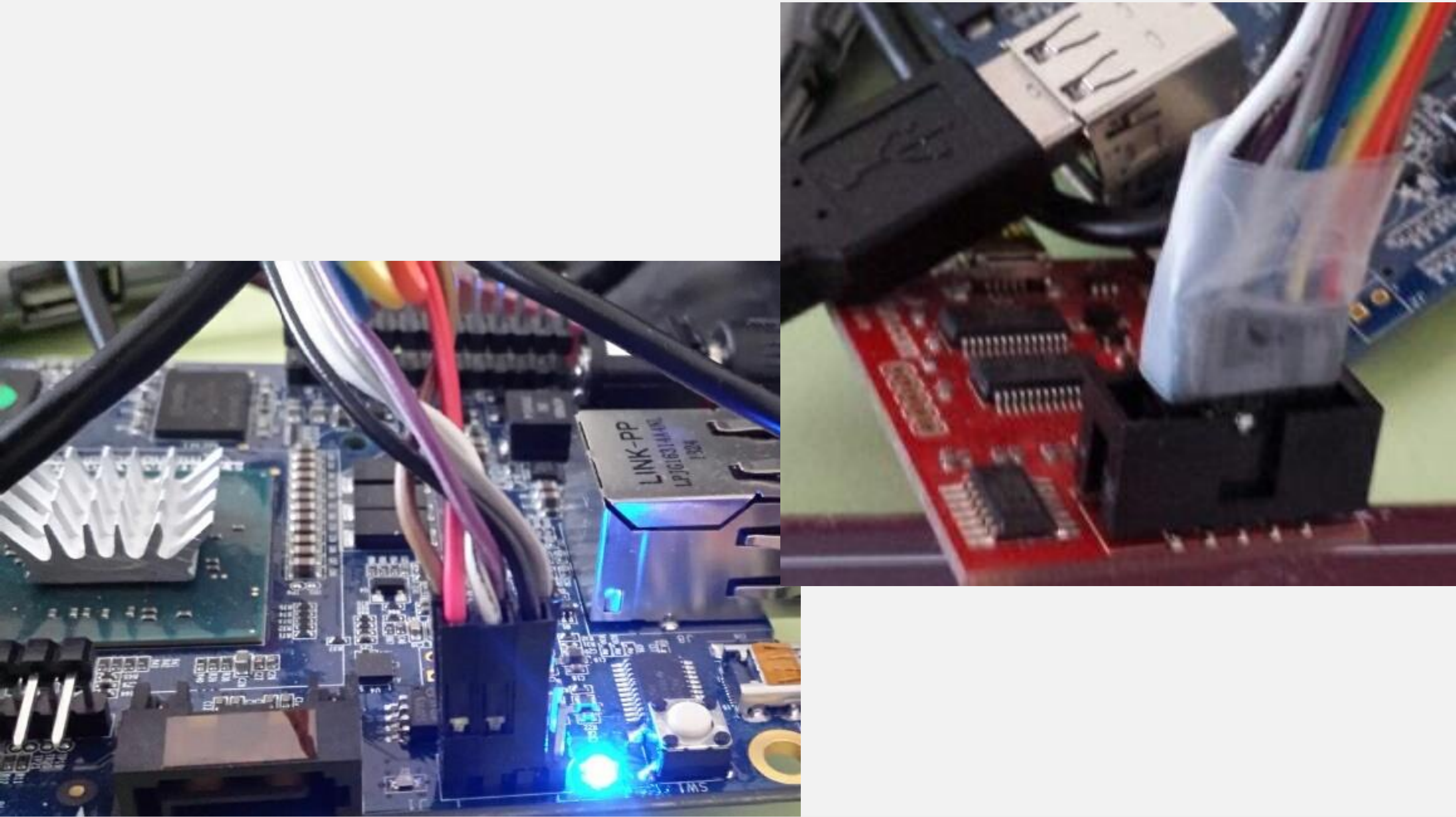
Checksum: **File = <empty> Memory = 69D518A8 Buffered File = <empty> Buffered Memory = 69D518A8**

Show In ● Hex ○ ASCII

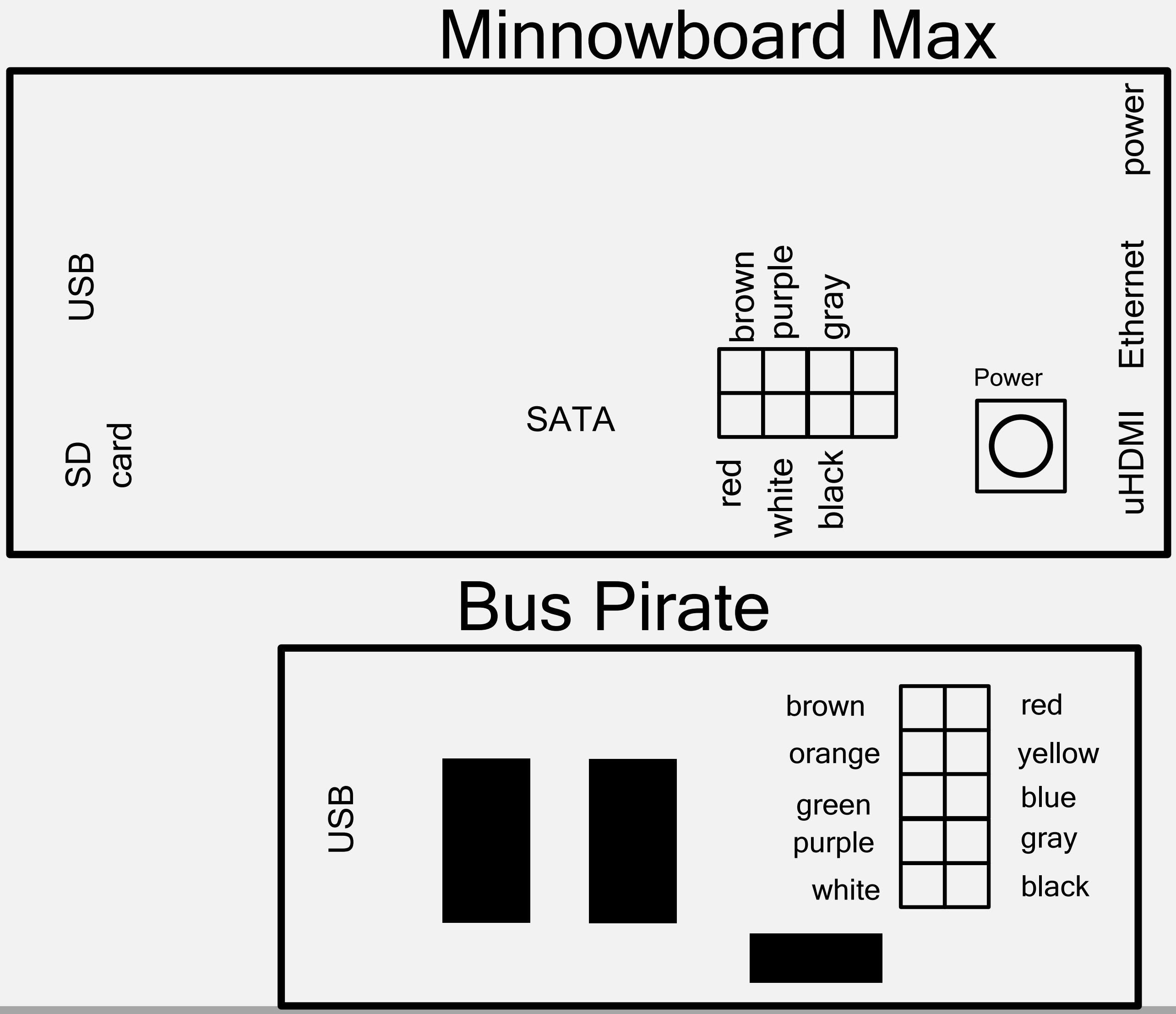
```
                                   File                                                          Memory
Address   +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F    +0 +1 +2 +3 +4 +5 +6 +7 +8 +9 +A +B +C +D +E +F
0x000000                                                     FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x000010                                                     5A A5 F0 0F 03 00 04 02 06 02 10 0B 20 00 21 00
0x000020                                                     25 00 00 00 FF FF FF FF FF FF FF FF FF FF FF FF
0x000030                                                     24 00 10 00 00 00 00 00 00 00 00 00 FF FF FF FF
0x000040                                                     00 00 00 00 00 05 FF 07 01 00 FF 04 00 00 00 00
0x000050                                                     FF 1F 00 00 FF FF FF FF FF FF FF FF FF FF FF FF
0x000060                                                     00 00 FF FF 00 00 FF FF FF FF FF FF FF FF FF FF
0x000070                                                     FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x000080                                                     FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x000090                                                     FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0000A0                                                     FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0000B0                                                     FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0000C0                                                     FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0000D0                                                     FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0000E0                                                     FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x0000F0                                                     FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF FF
0x000100                                                     00 00 00 00 00 00 00 00 00 04 05 00 00 00 10 18
```

Address (Hex):

Manipulating SPI image using Bus Pirate as HW SPI Flash programmer



Manipulating SPI Image Using Bus Pirate as HW SPI Flash Programmer



Minnowboard Max

power

Ethernet

uHDMI

USB

SD card

SATA

brown purple gray

red white black

Power

Bus Pirate

USB

brown

orange

green

purple

white

red

yellow

blue

gray

black

5.3 EDK II Debug

Exercise 5.3

EFI Debug

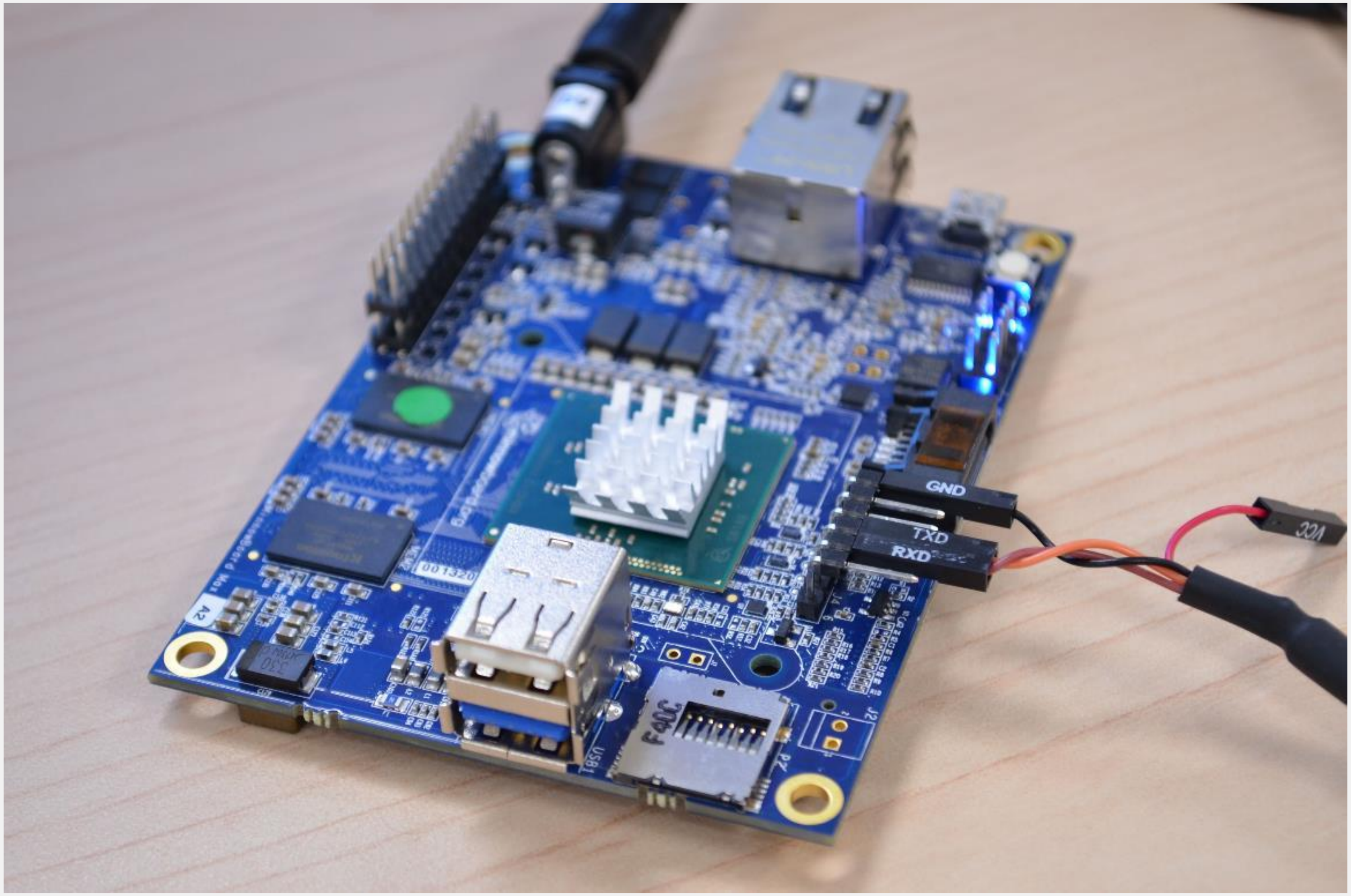
Exercise Outline

1. Serial connection setup
2. Configure host system for debugging
3. Debug example with GDB

Debug & Release Differences

- DEBUG has a slower boot than RELEASE because of time it takes to display debug info
- DEBUG has a larger image than RELEASE because the embedded debug info
- DEBUG uses the serial port for debug string output
- DEBUG contains the debug strings
- DEBUG contains detailed debug strings that show the boot process and various ASSERT/TRACE errors

Connect to UART port



To read UART output run minicom:

```
$minicom -D /dev/ttyUSB0
```

Configuring the Debug Host (Done)

UDK debugger configuration:

- Download `2013-WW52-UDK.Debugger.Tool-1.4-Linux.zip` from:

http://firmware.intel.com/develop/intel-uefi-tools-and-utilities/intel-uefi-development-kit-debugger-tool

- Install UDK debugger tool:

```
$./UDK_Debugger_Tool_v1_4_x86_64.bin
```

- Check/change configuration file: /etc/udkdebugger.conf

```
    $ cat /etc/udkdebugger.conf
[Debug Port]
Channel = Serial
Port = /dev/ttyUSB0
FlowControl = 0
BaudRate = 115200
```

GDB on Debug Host System (Done)

Rebuild GDB on HOST:

- Download gdb source code:

```
$ apt-get install gdb-source
$ cp /usr/src/gdb.tar.bz2 ~/Desktop/udk-debugger/
$ cd ~/Desktop/udk-debugger/
$ bzip2 -dc gdb.tar.bz2| tar -xf -
$ cd gdb
```

- Download (from http://expat.sourceforge.net) and install expat
- Configure and build gdb with expat:

```
$./configure --with-expat --with-python
$ make && make install
```

UEFI Firmware Debugging

- Run UDK-GDB-SERVER

```
$/opt/intel/udkdebugger/bin/udk-gdb-server
```

- Reboot Debug Target

- Run GDB on Debug Host

```
$/home/user/Desktop/udk-debugger/gdb/gdb
(gdb) target remote :1234
Remote debugging using :1234
```

Source Level Debug

```
(gdb) source /opt/intel/udkdebugger/script/udk-gdb-script
##############################################################
# This gdb configuration file contains settings and scripts
# for debugging UDK firmware.
# WARNING: Setting pending breakpoints is NOT supported!
# Additional commands for source level debugging will be
added!
##############################################################
Loading symbol for address: 0x78e1472c
add symbol table from file
"/home/user/Desktop/bios/Build/Vlv2TbltDevicePkg/DEBUG_GCC48/X
64/MdeModulePkg/Core/Dxe/DxeMain/DEBUG/DxeCore.debug" at
        .text_addr = 0x78dde260
        .data_addr = 0x78e180e0
(udb) where
```

Example: Setting Breakpoints

Set breakpoint on **CoreLoadPeImage**/**CoreLoadImage** functions

```
(udb) b CoreLoadPeImage
Breakpoint 1 at 0x78de2b22: file
/home/user/Desktop/bios/MdeModulePkg/Core/Dxe/Image/Image.c,
line 462.

(udb) b CoreLoadImage
Breakpoint 2 at 0x78de477c: file
/home/user/Desktop/bios/MdeModulePkg/Core/Dxe/Image/Image.c,
line 1409.

(udb) c
Continuing.
Loading symbol for address: 0x78de477c

Breakpoint 2, CoreLoadImage (BootPolicy=0 '\000',
ParentImageHandle=0x78d78f18, FilePath=0x78d5f018,
SourceBuffer=0x0, SourceSize=0, ImageHandle=0x78d5ee98)
    at
/home/user/Desktop/bios/MdeModulePkg/Core/Dxe/Image/Image.c:
1409
1409      Tick = 0;
```

Example: Setting Breakpoints

Set breakpoint to **`CoreExitBootServices`**:

```
(udb) break CoreExitBootServices
Breakpoint 1 at 0x78ddfdac: file
/home/user/Desktop/bios/MdeModulePkg/Core/Dxe/DxeMain/DxeMai
n.c, line 731.

(udb) c
Continuing.
Loading symbol for address: 0x78ddfdac

Breakpoint 1, CoreExitBootServices (ImageHandle=0x768d8798,
MapKey=3615) at
/home/user/Desktop/bios/MdeModulePkg/Core/Dxe/DxeMain/DxeMai
n.c:731
731       gTimer->SetTimerPeriod (gTimer, 0);
```

Example: Setting Breakpoints

Set breakpoint to `DxeImageVerificationHandler` which contains *PE/TE header confusion* vulnerability in Secure Boot implementation

```
(udb) break DxeImageVerificationHandler
(udb) c
Continuing.
```

Debug function `DxeImageVerificationHandler` using step:

```
(udb) step
```

Useful GDB commands

Disassembly:

```
(gdb) display/i $pc
(gdb) set  disassemble-next-line on
(gdb) show disassemble-next-line
(gdb) layout asm(udb)
```

Other:

```
(gdb) info breakpoints
(gdb) info args       # Print args to the function of the current stack frame
(gdb) list            #  Shows the current or given source context.
(gdb) info registers  #  Show registers
(gdb) info frame      #  Show the stack frame info
```

5.4 EDK II Overview

EDK II

- Most of the source code written in C
- Provides Flash Mapping Tool generating Firmware Volumes and the resulting SPI flash image
- Build Existing EDK Modules
- EDKII projects are made up of packages (DEC files)
- Compiles to .EFI files: UEFI/DXE Driver, PEIM, UEFI Application, DXE Library

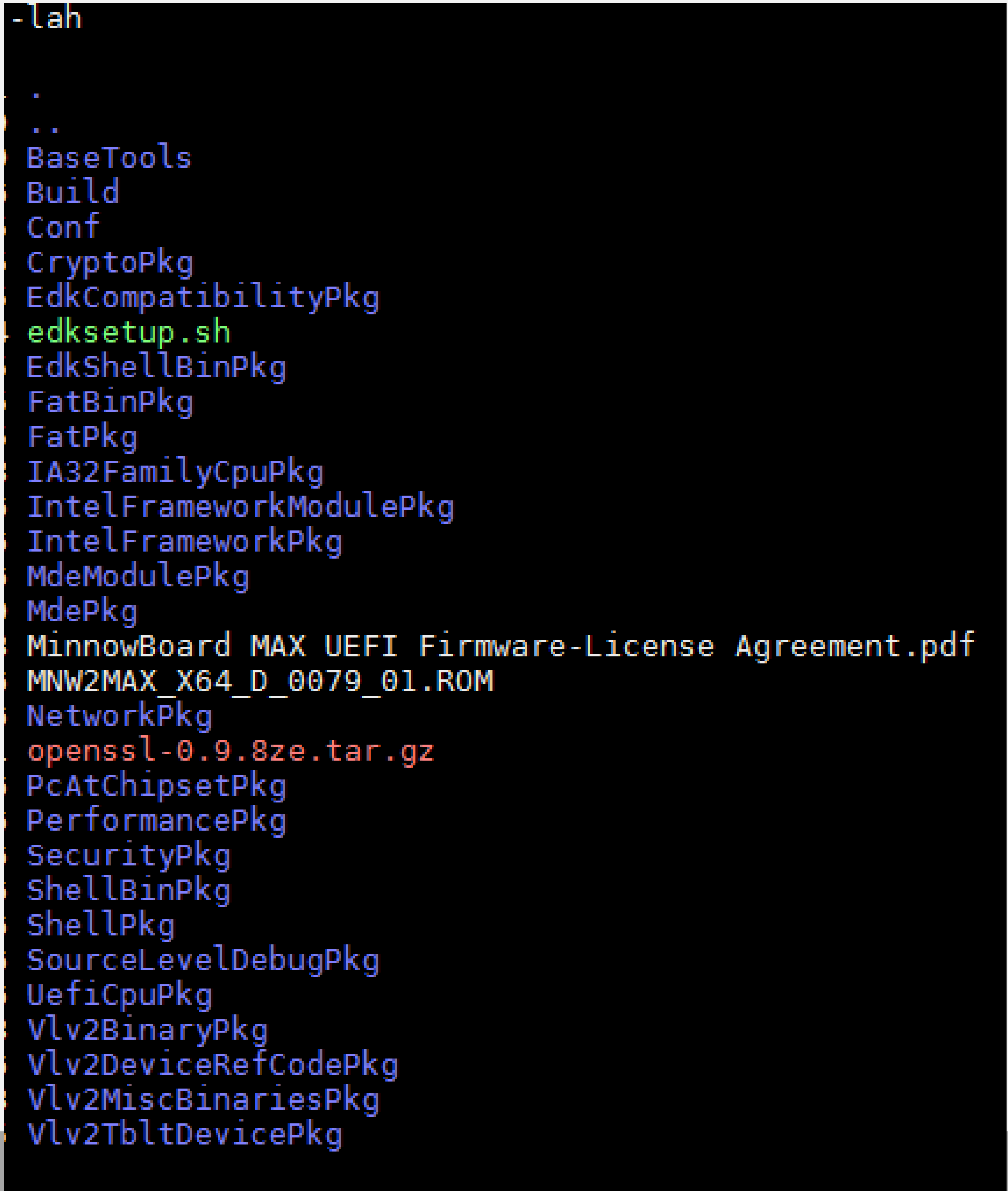
MinnowBoard Max EDKII Source Tree

Package concept for each EDK II sub-directory

Platform specific packages (**`Vlv2..Pkg`**) are also there

EDK II build process reflects the package

```
# edksetup
# build -p
Nt32Pkg\Nt32Pkg.dsc
-a IA32
```



```
-lah


 .
 ..
 BaseTools
 Build
 Conf
 CryptoPkg
 EdkCompatibilityPkg
 edksetup.sh
 EdkShellBinPkg
 FatBinPkg
 FatPkg
 IA32FamilyCpuPkg
 IntelFrameworkModulePkg
 IntelFrameworkPkg
 MdeModulePkg
 MdePkg
 MinnowBoard MAX UEFI Firmware-License Agreement.pdf
 MNW2MAX_X64_D_0079_01.ROM
 NetworkPkg
 openssl-0.9.8ze.tar.gz
 PcAtChipsetPkg
 PerformancePkg
 SecurityPkg
 ShellBinPkg
 ShellPkg
 SourceLevelDebugPkg
 UefiCpuPkg
 Vlv2BinaryPkg
 Vlv2DeviceRefCodePkg
 Vlv2MiscBinariesPkg
 Vlv2TbltDevicePkg
```

EDK II Packages

`MdePkg` - Include files and libraries for Industry Standard Specifications

`MdeModulePkg` - Modules only definitions from the Industry Standard Specification are defined in the **`MdePkg`**

`SecurityPkg` – Implements security related functionality (Secure Boot, Authenticated Variables, etc.)

`CryptoPkg` – Provides crypto functionality

`ShellPkg & NetworkPkg` - Functionality of shell & network stack

`IA32FamilyCpuPkg` – Package supporting IA32 family processors

`IntelFrameworkPkg` - Include files and libraries for those parts of the Intel Platform Innovation Framework for EFI specifications not adopted “as is” by the UEFI or PI specifications

`Nt32Pkg` – Windows UEFI emulator

EDKII File Extensions

.DSC - Platform Description file (recipe for creating a package, contains definitions to build the package)

.DEC - Package Declaration file

.INF - Module Definition (defines a component)

.FDF - Flash Description File (describes information about flash parts)

.FV - Firmware volume (FV) binary file

Platform Configuration Database (PCD)

- PCD options define parameters which allow modules to define firmware configuration without recompile/rebuild or source code change
- There’s an API to access to PCD options
- PCD options can store platform or feature configuration settings

PCD Example

- PCD options are defined in the DEC files in any package

```
./SecurityPkg/SecurityPkg.dec:

gEfiSecurityPkgTokenSpaceGuid.PcdOptionRomImageVerificationPolicy
|0x04|UINT32|0x00000001
```

- Values of PCD options are set in DSC files

```
[PcdsFixedAtBuild.IA32]
...
gEfiIchTokenSpaceGuid.PcdIchAcpiIoPortBaseAddress|0x400
```

5.5 Building UEFI Applications/Drivers

Building UEFI Application

UDKII User Manual describes a module building process in Chapter 3.4. You need to create a new package containing the module or add the module to existing package

http://tianocore.sourceforge.net/wiki/EDK_II_User_Documentation

UEFI Driver Wizard is of great help for module creation

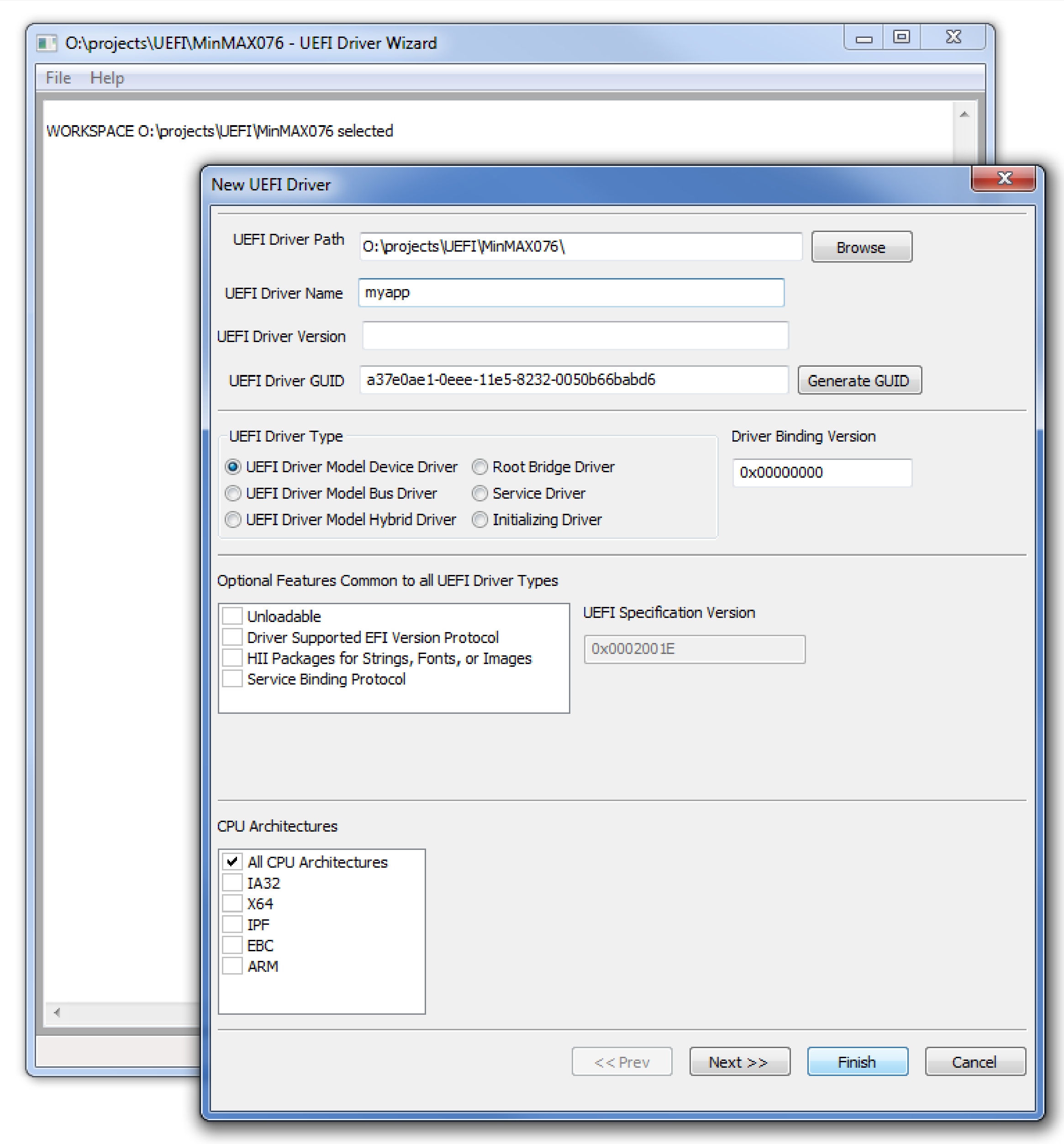
http://tianocore.sourceforge.net/wiki/UEFI_Driver_Wizard

Building UEFI Application

For this exercise, there’s a `myapp` package and a module generated with UEFI Driver Wizard and located in `myapp` folder in the UEFI source tree

```
myapp/
   myapp.h
   myapp.c
   myapp.dec – package declaration file
   myapp.dsc – platform build description file
   myapp.inf – module information file
   myapp.uni – unicode string file
```

UEFI Driver Wizard



O:\projects\UEFI\MinMAX076 - UEFI Driver Wizard

WORKSPACE O:\projects\UEFI\MinMAX076 selected

New UEFI Driver

UEFI Driver Path: O:\projects\UEFI\MinMAX076\

UEFI Driver Name: myapp

UEFI Driver Version:

UEFI Driver GUID: a37e0ae1-0eee-11e5-8232-0050b66babd6

UEFI Driver Type

- ● UEFI Driver Model Device Driver
- ○ UEFI Driver Model Bus Driver
- ○ UEFI Driver Model Hybrid Driver
- ○ Root Bridge Driver
- ○ Service Driver
- ○ Initializing Driver

Driver Binding Version: 0x00000000

Optional Features Common to all UEFI Driver Types

- □ Unloadable
- □ Driver Supported EFI Version Protocol
- □ HII Packages for Strings, Fonts, or Images
- □ Service Binding Protocol

UEFI Specification Version: 0x0002001E

CPU Architectures

- ☑ All CPU Architectures
- □ IA32
- □ X64
- □ IPF
- □ EBC
- □ ARM

Building a UEFI application

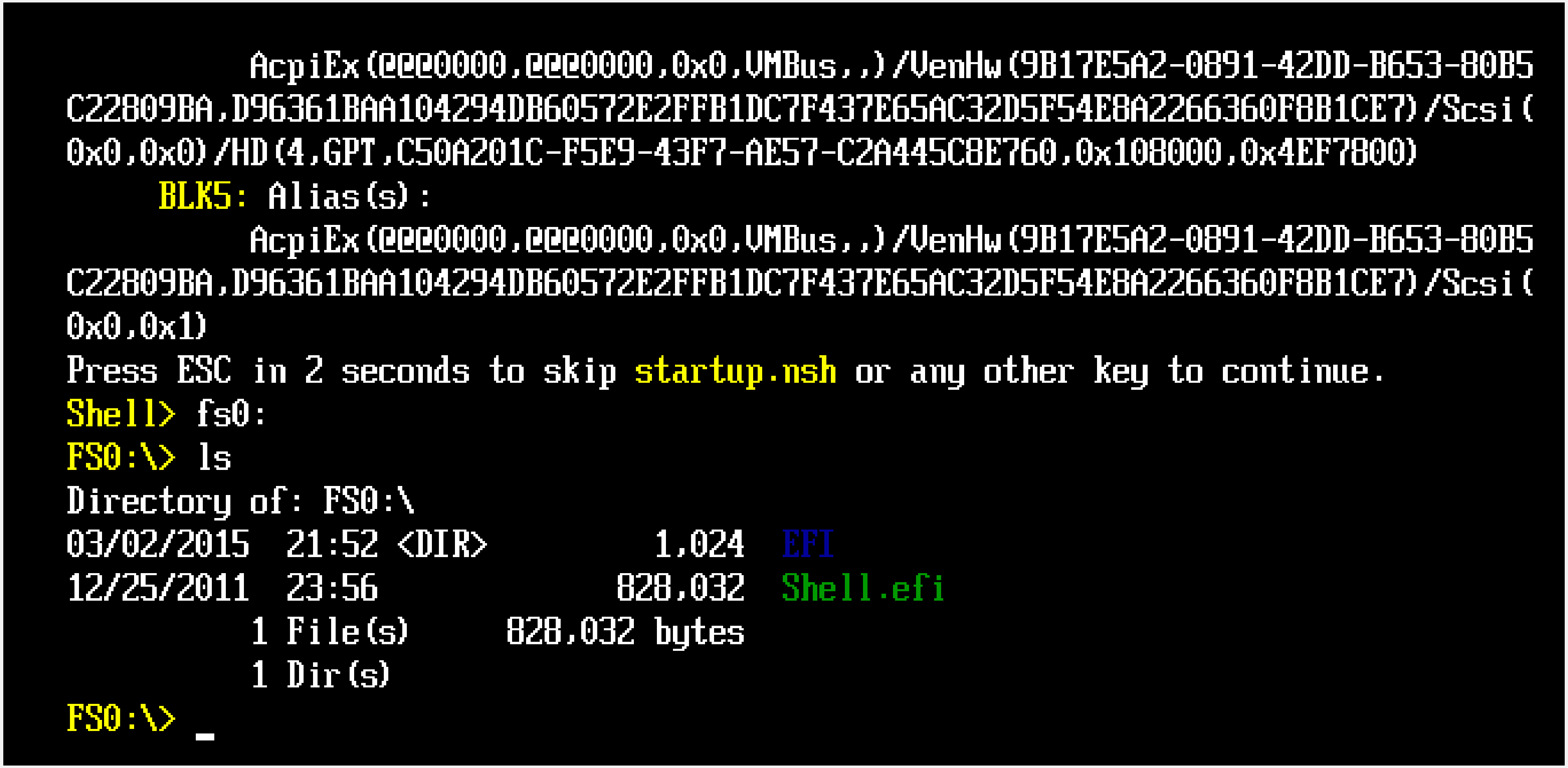
Linux:

```
#!/usr/bin/env bash
source edksetup.sh
build -a X64 -p myapp/myapp.dsc -m myapp/myapp.inf
```

Windows:

```
call edksetup.bat
build -a X64 -p myapp\myapp.dsc -m myapp\myapp.inf
```

UEFI Shell



```
          AcpiEx(@@@0000,@@@0000,0x0,VMBus,,)/VenHw(9B17E5A2-0891-42DD-B653-80B5
C22809BA,D96361BAA104294DB60572E2FFB1DC7F437E65AC32D5F54E8A2266360F8B1CE7)/Scsi(
0x0,0x0)/HD(4,GPT,C50A201C-F5E9-43F7-AE57-C2A445C8E760,0x108000,0x4EF7800)
     BLK5: Alias(s):
          AcpiEx(@@@0000,@@@0000,0x0,VMBus,,)/VenHw(9B17E5A2-0891-42DD-B653-80B5
C22809BA,D96361BAA104294DB60572E2FFB1DC7F437E65AC32D5F54E8A2266360F8B1CE7)/Scsi(
0x0,0x1)
Press ESC in 2 seconds to skip startup.nsh or any other key to continue.
Shell> fs0:
FS0:\> ls
Directory of: FS0:\
03/02/2015  21:52 <DIR>         1,024  EFI
12/25/2011  23:56             828,032  Shell.efi
          1 File(s)     828,032 bytes
          1 Dir(s)
FS0:\> _
```

Reading Command-Line Arguments

Some information can be found here:

Creating a Shell Application

http://tianocore.sourceforge.net/wiki/Creating_a_Shell_Application

From ShellLib we'll use following structure

```
SHELL_PARAM_ITEM,
```

and functions

```
ShellCommandLineParseEx
ShellCommandLineGetFlag
ShellCommandLineGetValue
```

Reading Command-Line Arguments

```
EFI_STATUS EFIAPI ShellCommandLineParseEx ( IN CONST SHELL_PARAM_ITEM *  CheckList,
                                            OUT LIST_ENTRY **  CheckPackage,
                                            OUT CHAR16 **ProblemParam  OPTIONAL,
                                            IN BOOLEAN  AutoPageBreak,
                                            IN BOOLEAN  AlwaysAllowNumbers  )
```

Checks the command line arguments passed against the list of valid ones. Optionally removes NULL values first. If no initialization is required, then return RETURN_SUCCESS.

Parameters:

[in] CheckListThe pointer to list of parameters to check.
[out] CheckPackageThe package of checked values.
[out] ProblemParamOptional pointer to pointer to unicode string for the paramater that caused failure.
[in] AutoPageBreakWill automatically set PageBreakEnabled.
[in] AlwaysAllowNumbersWill never fail for number based flags.

Return values:

EFI_SUCCESS	The operation completed sucessfully.
EFI_OUT_OF_RESOURCES	A memory allocation failed.
EFI_INVALID_PARAMETER	A parameter was invalid.
EFI_VOLUME_CORRUPTED	The command line was corrupt.
EFI_DEVICE_ERROR	The commands contained 2 opposing arguments. One of the command line arguments was returned in ProblemParam if provided.
EFI_NOT_FOUND	A argument required a value that was missing. The invalid command line argument was returned in ProblemParam if provided.

Reading Command-Line Arguments

```
BOOLEAN EFIAPI ShellCommandLineGetFlag ( IN CONST LIST_ENTRY
*CONST  CheckPackage,
                                        IN CONST CHAR16 *CONST  KeyString  )
```

Checks for presence of a flag parameter.
Flag arguments are in the form of "-<Key>" or "/<Key>", but do not have a value following the key.

If CheckPackage is NULL then return FALSE. If KeyString is NULL then ASSERT().

Parameters:

[in] CheckPackage The package of parsed command line arguments.
[in] KeyStringThe Key of the command line argument to check for.

Return values:

TRUE The flag is on the command line.
FALSE The flag is not on the command line.

Reading Command-Line Arguments

```
CONST CHAR16* EFIAPI ShellCommandLineGetValue ( IN CONST LIST_ENTRY
*CONST  CheckPackage,
                                                                  IN CONST CHAR16 *CONST  KeyString  )
```

Checks for presence of a flag parameter.
Value parameters are in the form of "-<Key> value" or "/<Key> value".

If CheckPackage is NULL then return NULL.

Parameters:

[in] CheckPackage   The package of parsed command line arguments.
[in] KeyStringThe   Key of the command line argument to check for.

Return values:

NULL   The flag is not on the command line.
!=NULL   The pointer to unicode string of the value.

Reading Command-Line Arguments

```
#define NAME_OPTION  (L”-n”)
#define GUID_OPTION  (L”-g”)
#define HELP_OPTION  (L”-?”)

SHELL_PARAM_ITEM    ParamList[] = {
  { NAME_OPTION,   TypeValue },
  { GUID_OPTION,   TypeValue },
  { HELP_OPTION,   TypeFlag },
  { NULL,          TypeMax },
};

LIST_ENTRY  *ParamPackage;
ShellCommandLineParseEx(ParamList, &ParamPackage,
NULL, TRUE, FALSE);
```

Reading Command-Line Arguments

```
CONST CHAR16        *name_str = 0;
CONST CHAR16        *guid_str = 0;

if (ShellCommandLineGetFlag(ParamPackage,NAME_OPTION)){
    name_str = ShellCommandLineGetValue (ParamPackage,
NAME_OPTION);
}

if (ShellCommandLineGetFlag(ParamPackage, GUID_OPTION )) {
    guid_str = ShellCommandLineGetValue (ParamPackage,
GUID_OPTION );
}
```

Using UEFI Runtime Services

Good reference:
http://wiki.phoenix.com/wiki/index.php/EFI_RUNTIME_SERVICES

Global variable gRT, points to runtime service table, declared in

Library/UefiRuntimeServicesTableLib.h

Calling the service function:

```
gRT->GetVariable((CHAR16*)name_str, &guid,
                    &attributes, &size, buffer);
```

Dependencies

Add include files to myapp.h:

```
#include <Library/ShellLib.h>
#include <Protocol/EfiShell.h>
#include <Protocol/EfiShellInterface.h>
#include <Protocol/EfiShellParameters.h>
```

Dependencies

ShellPkg is used. Add ShellPkg and its dependencies to package files.

Add to myapp.inf [Packages] section

```
    ShellPkg/ShellPkg.dec
```

Add [LibraryClasses] section

```
    ShellLib to myapp.inf
```

Specify the location of ShellLib module INF file and dependecies in myapp.dsc [LibraryClasses]:

```
ShellLib|ShellPkg/Library/UefiShellLib/UefiShellLib.inf
FileHandleLib|ShellPkg/Library/UefiFileHandleLib/UefiFileHandleLib.inf
HiiLib|MdeModulePkg/Library/UefiHiiLib/UefiHiiLib.inf
SortLib|ShellPkg/Library/UefiSortLib/UefiSortLib.inf
UefiHiiServicesLib|MdeModulePkg/Library/UefiHiiServicesLib/UefiHiiServicesLib.inf
```

Exercise 5.4

Building UEFI Application

References

TianoCore.org site documentation

http://sourceforge.net/projects/edk2/files/

EDK II INF File Specification, Version 1.2, Intel, 2009.

EDK II DSC File Specification, Version 1.2, Intel, 2009.

EDK II DEC File Specification, Version 1.2, Intel, 2009.

EDK II FDF (Flash Description File) File Specification, Version 1,2, Intel, 2009.

EDK II Build Specification, Version 1.2, Intel, 2009.

Training materials are available on Github

https://github.com/advanced-threat-research/firmware-security-training

Yuriy Bulygin	@c7zero
Alex Bazhaniuk	@ABazhaniuk
Andrew Furtak	@a_furtak
John Loucaides	@JohnLoucaides