Tiano Power-on Debug Guide

Revision 0.1

Intel reference number OR-

Revision 0.10

March 18, 2003

Enterprise Platforms and Services Division

Revision History

Date	Revision Number	Modifications
10/14/02	0.1	Initial draft

Disclaimers

Table of Contents

1. Introduction

1.1 Purpose of this Document

This document describes some useful techniques for tackling the challenges of power-on while using Tiano EPG Core.

2. Poweron Debugging Techniques

2.1 Conventions

While listing the specific components, this document uses “Westville” and “E7500” and “ICH3” components. These need to be translated to the appropriate Platform Name, North Bridge name and South Bridge Name respectively.

2.2 General Poweron strategy

The primary concern during the power-on debugging is that the hardware is untested and may not work as expected.

There are two poweron BIOS can be built with all the components that are necessary to boot an OS (at least EFI shell/DOS). If the system fails to boot, it is necessary to

1. Isolate the failure to a software/hardware component - Use port 80 codes or something equivalent to locate the point of failure. If serial port is working, the debug outputs to serial port work much better. Once the approximate point of failure is detected, appropriate drivers/PEIMs can be instruments or modified to narrow down to the exact failure and work around it.

2. If it is an optional component/feature (e.g. SMIs are not needed to boot DOS or EFI shell), it can be removed from the build.

3. If the component is required to boot (e.g. PCI) and the failure is due to hardware, the low level hardware layer needs to be “patched” to work around the issues. This document describes the various components that directly touch hardware and how they can be controlled.

In general, it is easier to boot to EFI shell than DOS because (a) EFI shell is in the flash (b) We can booting to EFI shell while maintaining the driver stacking. Driver stacking makes it easier to debug issues and add hardware “workarounds”.

2.2.1 General Model

DXE drivers and PEIMs access the hardware in a controlled way. It is easy to control the behavior because of an orderly PEIM/driver stack.

The stack is very simple in PEI phase because the IO subsystem is not enumerated in the PEI phase.

In the DXE phase, the IO subsystem is enumerated and the stack can get fairly deep. As far as hardware drivers are concerned, the stack mimics the actual hardware layout (Device Path concept) with a parent-child relationship. For example, the USB keyboard driver depends upon the UHCI controller driver. UHCI controller being a PCI device depends upon another hardware driver, the PCI Host Bridge/Root Bridge driver. The PCI Host bridge/Root bridge driver depends upon the CPU to generate IO instructions.

If a driver is turned off (or its behavior is altered), all its children devices (those devices that appear below in the hardware hierarchy) are turned off as well. In the above example, changes to UCHI driver will affect all USB devices below the UHCI controller. Turning off UHCI controller in the ICH will prevent UHCI driver from running, which automatically blocks all USB drivers including KBD, mouse, mass storage.

2.2.2 Notable Exceptions

These rules do not apply to Legacy Option ROMs and CSM will access PCI config space and PCI MMIO/IO space directly. Remove these from the build when debugging potentially serious hardware issues.

Standalone PEIMs that do not utilize the PPIs (such as the one we are planning to write for TNB memory sizing) do not follow these rules.

There are PEIMs and drivers with special status. These do not follow the stack rules. These are Status code PEIM/driver and debug driver,

Component = Platform\IntelEpg\Common\StatusCode\Pei\MonoStatusCode.inf is the status code PEIM.

2.2.3 General Model for exporting setup options/platform hooks

A number of platform abstractions (PPIs and Protocols) are defined. Various core drivers rely on these interfaces to obtain the platform specific configuration information. The platform is free to derive this information from any source including setup variables. Any changes to setup questions are usually saved in setup variables (e.g. IDE mode Legacy/Native). The platform is free to choose an appropriate layout of these variables and the platform drivers are linked against this map. The core drivers maintain binary compatibility by calling the platform abstractions to get the settings instead of accessing the setup variables directly.

The platform abstractions are also used to communicate platform specific information to the core drivers. For example, the core LegacyBios driver calls the LegacyBiosPlatform protocol to obtain PIRQ routing information. The generic CPU driver calls the Platform MP protocol to find out mapping between processor's physical position and its APIC ID at reset. All the platform specific drivers/PEIMs are under the respective platform tip.

The same interfaces are used to implement general purpose platform hooks. For example, the generic CPU driver calls the platform MP driver after it has enumerated all the CPUs so that the platform can override the BSP selection or make other adjustments. These interfaces are defined in the respective specifications.

2.3 Driver/PEIMs divided based on hardware

2.3.1 CpuIo

This is the root driver.

Component = Cpu\Pentium\CpuIo\Dxe\CpuIo.inf – This driver is the only one that executes in/out instructions or accesses to memory mapped IO. You can change this driver to fake read/write to an IO port or redirect writes to one port to another. These changes will affect all DXE drivers by changing this driver.

The equivalent PEIM is
Component = Cpu\Pentium\CpuIo\Pei\CpuIo.inf
Most IO accesses occur in DXE phase, but change both PEIM and DXE driver the same way.

Watch out for:

1. The Status code driver and debug driver do not go through CpuIO because of their special status. These two must be changed/removed for this blocking to work.
2. SMM drivers use the cpu/Pentium/dispatcher/smm/io.c.

2.3.2 PCI Accesses

Component =
Platform\IntelEpg\Manhattan\MchPciHostBridge\Dxe\WestvillePciHostBridge.inf
Must be ported to your platform. This driver sets up the PCI Root bridge and can be changed to chipset generated PCI busses disappear. All PCI related drivers will call this driver to access PCI configuration space and memory/IO allocated to PCI devices. You can prevent most of the DXE stack from accessing a PCI device/function or fake a value on config read to a specific PCI config register. This driver also needs to access PCI config space of the chipset. It has been written such that it uses the same low level registers no matter whether called by other drivers or internally. Note that this driver sits above CpuIO and will use CpuIo to perform in/out instructions.

The equivalent PEIM is
Component = Chipset\PcatCompatible\SingleSegmentPciCfg\Pei\PciCfg.inf

Exceptions:

1. The runtime drivers, such as Flash driver cannot use services of this driver and use the library function in library\RuntimeDxe\EfiRuntimeLib\Ia32\PlatformIoLib.c. There are very few of these drivers. SMM drivers use the same mechanism.

The PCI bus is enumerated by the PCI bus driver, but the PCI bus driver itself does not perform low level accesses to hardware. It will depend upon the PCI Host Bridge driver to do that. PCI bus driver will simply follow various PCI SIG Specifications.

The following driver allows the platform to declare incompatible PCI devices. If a device is declared as incompatible PCI device, the PCI bus driver will use the BAR size information provided by this driver instead of from the actual BARs. This driver can also be used to enforce any alignment beyond the natural alignment that is required by the PCI spec.
Component = Platform\Generic\Dxe\IncompatiblePciDevice\IncompatiblePciDevice.inf

2.3.3 PC-AT compatible hardware:

These drivers access the PC-AT compatible hardware in the chipset and are the only drivers that do so. All of these use CPU IO to access hardware.

1. Component = Chipset\PcatCompatible\8259InterruptController\Dxe\Legacy8259.inf is the only component that accesses 8259 PIC.
2. Component = Chipset\PcatCompatible\Metronome\Dxe\LegacyMetronome.inf – This driver uses the 8254 and port 61 to produce fixed delays. The PEI equivalent is Component = Chipset\PcatCompatible\8254Timer\Pei\8254Stall.inf. This PEIM/Driver pair can be changed to scale delays or replace hardware delays by software delays (loops).
3. Component = Chipset\PcatCompatible\8254Timer\Dxe\Timer.inf – This driver programs the 8254 to generate timer tick.
4. Component = Chipset\PcatCompatible\RealTimeClock\RuntimeDxe\PcRtc.inf – This driver accesses the date/time registers in the RTC to provide date/time services.
5. Component = Chipset\PcatCompatible\Cf9Reset\RuntimeDxe\Cf9Reset.inf – This driver provides reset functionality by using port cf9. All DXE drivers that want to perform reset will go thru' this driver.
6. Component = Chipset\PcatCompatible\Speaker\Dxe\LegacySpeaker.inf and Component = Chipset\PcatCompatible\Speaker\Pei\LegacySpeaker.inf– Uses the standard speaker to provide beep services.
7. Component = Chipset\PcatCompatible\8237DmaController\Pei\8237DmaController.inf – Used during recovery to program the DMA controller.

2.3.4 Legacy ISA hardware drivers:

ISA devices are not enumerable the way PCI devices are. The ISA bus driver uses the device list provided by the SIO driver for enumerating ISA devices.
The SIO driver can be something like
Component = Chipset\NationalLpcPc7417\Dxe\LpcPc87417.inf

This driver enables various logical devices in the SIO and reports their presence to the ISA bus driver. The SIO driver depends upon a platform driver provide platform settings including setup options. E.g.
Component = Platform\IntelEpg\Westville\LpcPlatform\Dxe\LpcPlatform.inf

The platform driver can be changed to hide devices/change the base addresses. Other option is to update the SIO driver to not use the platform driver and make changes to the SIO driver.

ISA bus driver produces ISA_IO for use by ISA peripheral drivers. ISA_IO will call PCI Root bridge driver since ISA is a child of PCI. ISA bus driver is
Component = Bus\Isa\IsaBus\Dxe\IsaBus.inf

These are the drivers for the typical legacy ISA peripherals.

1. Component = Platform\IntelEpg\Bus\Isa\IsaSerial\Dxe\IsaSerial.inf – Driver that controls serial port.
2. Component = Bus\Isa\IsaFloppy\Dxe\IsaFloppy.inf – Driver that manages legacy floppy and allows read/writes to floppy drives.
3. Component = Bus\Isa\IsaFloppy\Pei\FloppyPeim.inf - Driver that manages legacy floppy and allows reads from the floppy drives during recovery process.
4. Component = Bus\Isa\Ps2Keyboard\Dxe\Ps2Keyboard.inf – Driver for the PS2 keyboard.
5. Component = Bus\Isa\Ps2Mouse\Dxe\Ps2Mouse.inf – Driver for PS/2 mouse. Has to co-operate with the PS2 KBD driver while initializing the KBC.

2.3.5 IDE hardware drivers:

The key hardware driver is the IDE controller driver.

Component = Chipset\IntelIch\Ich2\IdeController\Dxe\IdeController.inf

The IDE bus driver uses this hardware driver to configure IDE devices behind ICHx. If the IDE function in ICH is turned off, neither the IDE controller driver nor the IDE bus driver will run.

2.3.6 CPU drivers

These are generic drivers/PEIMs-
Component = Cpu\Pentium\Cpu\Dxe\MpCpu.inf
Component = Cpu\Pentium\Cpu\Pei\ComplexCpuPeim.inf

Component = Cpu\Pentium\BiosUpdate\Prestonia\M02F2734\M02F2734 – Microcode Update, add your own.

Component = Platform\IntelEpg\Westville\PlatformMp\Dxe\PlatformMp.inf – This driver abstracts platform from the generic MP/CPU driver.

2.3.7 Memory

These are generic memory test driver/PEIM pair. In general, the PEI phase tests minimum amount of memory and the DXE phase tests the remaining. The DXE driver uses the standard memory HOBs to find out how much memory is untested. If the HOBs report that all the memory is tested, the DXE driver will not retest it.

Component = Universal\GenericMemoryTest\Dxe\Above4GSupport.inf
Component = Universal\GenericMemoryTest\Pei\Ia32\BaseMemoryTest.inf

Component = Cpu\Pentium\VirtualMemory\Dxe\VirtualMemory.inf – Provides API to access memory above 4 GB (for IA-32 processors).

The following driver abstracts platform specific details and provides error detection/logging capabilities.

Component = Platform\IntelEpg\Westville\PlatformMemoryTest\Dxe\PlatformMemTest.inf

The following PEIMs are used during memory sizing in Westville Tiano. These can be replaced by a monolithic PEIM in TC.

Component = Chipset\IntelE7500\MemoryController\Pei\MemoryController.inf
Component = Platform\Generic\Pei\SdrDdrMemoryInit\SdrDdrMemoryInit.inf
Component = Chipset\IntelIch\Ich2\Smbus\Pei\Ich2Smbus.inf

2.3.8 SMM drivers

Component = Chipset\IntelIch\Ich2\SmmControl\Dxe\SmmControl.inf – This driver provides the ability to generate software SMI and is extensively used by the SMM infrastructure.

Component = Chipset\IntelE7500\SmmAccess\Dxe\SmmAccess.inf – This driver abstracts read/write/open/close SMRAM functions and is required for SMM to work.

Disabling either of these drivers will prevent all SMM activity.

Other SMM drivers that rely on the CPU’s ability to process SMIs correctly are –

Component = Cpu\Pentium\Start\Smm\SmmStart.inf
Component = Cpu\Pentium\Base\Smm\SmmBase.inf

Component = Cpu\Pentium\Core\Smm\SmmCore.inf

2.3.9 Flash Drivers

Component = Platform\Generic\RuntimeDxe\FvbServices\WvFvb.inf
This driver provides low level flash access to the higher level stack (such as the variable driver etc.). It must be ported to your platform.

Component = Platform\Generic\Pei\FlashMap\WestvilleFlashMap.inf
Should be ported to match the platform flash map. (TBD, wrong location)

2.3.10 Misc Platform/chipset components

These components will still use CpuIo to access MMIO/IO space and PCI Root Bridge IO to access their configuration space.

Component = Platform\IntelEpg\Westville\PlatformPeim\Pei\PlatformPeim.inf

Component = Chipset\IntelIch\Ich3\Ich3Init\Dxe\Ich3Init.inf
Component = Chipset\IntelIch\Ich2\Smbus\Dxe\IchSmbus.inf
Component = Chipset\IntelIch\Ich3\Gpio\Dxe\Ich3Gpio.inf
Component = Chipset\IntelIch\Ich2\PowerButtonReset\Dxe\Ich2PowerButtonReset.inf
Component = Chipset\IntelIch\NewIchSmm\Smm\Ich3SmmDispatcher.inf

Component = Platform\IntelEpg\Westville\Ich3Platform\Dxe\Ich3Platform.inf – Abstracts platform specific details from ICH3 drivers.

2.3.11 CSM components

Component = Csm\LegacyBios\Dxe\LegacyBios.inf – required for CSM to work. CSM can be disabled by removing this driver.

Component = Platform\IntelEpg\Westville\LegacyBiosPlatform\Dxe\LegacyBiosPlatform.inf – Abstracts the platform from the core CSM driver.

The following components touch hardware directly but do not follow Tiano rules (because they are legacy components).
and other option ROMs.
Component = Platform\IntelEpg\Westville\ScsiRom\WestvilleScsiRom.inf
Component = Platform\IntelEpg\Westville\VideoRom\WestvilleVideoRom.inf
Component = Csm\Ami\AmiLegacy16.inf

2.4 Software Only Components

BY definition, anything outside Cpu or Chipset cannot touch hardware directly. The only hardware they use directly is system memory. Many drivers in Universal are software only drivers (like MBR partition driver). These are not likely to cause problems during power-on unless the memory/cache system is unstable and running these drivers stresses the memory/cache subsystem. These generic components have been verified to work on one or more platforms. That does not make these components bug-free, but there is a reasonable assurance that most of the critical bugs have been worked our.

2.4.1 SEC Core

Although this is named core, there are platform specific components in it.

Component = Platform\IntelEpg\Common\CpuCar\seccore.inf

To complete this component, you will need to supply an include file in your platform directory e.g. Platform/IntelEpg/Manhattan/CpuCar/CpuCar.inc. This file should be ported to match your platform.
Use
Component = Cpu\Pentium\SoftSdv\Sec\SecCore.inf
for SoftSdv platforms.

2.4.2 PEI Core

Component = Core\Pei\PeiMain.inf
This is truly a core component and does not need any porting. You will need this, otherwise your PEIMs won't be dispatched.
Make sure PACKAGE is set to PeiCore.

2.4.3 DXE Core

Component = Core\Dxe\DxeMain.inf
This is truly a core component and does not need any porting. You will need this, otherwise you will hang after completing PEI phase or go to recovery.
Make sure PACKAGE is set to DxeMain

2.4.4 PEIMs

Component = Universal\DxeIpl\Pei\DxeIpl.inf

Component = Universal\BiosId\Pei\BiosId.inf - Support for BIOS ID.
Component = Universal\Disk\FileSystem\Fat\Pei\PeiFat.inf – during recovery

2.4.5 DXE Drivers

Component = Universal\FirmwareVolume\GuidedSectionExtraction\Crc32SectionExtract\Dxe\Crc32SectionExtract.inf
Component = Universal\UserInterface\HiiDataBase\Dxe\HiiDatabase.inf
Component = Universal\DataHub\DataHub\Dxe\DataHub.inf
Component = Universal\WatchdogTimer\Dxe\WatchDogTimer.inf
Component = Universal\Runtime\Dxe\Runtime.inf
Component = Universal\Variable\Variable\RuntimeDxe\Variable.inf
Component = Universal\MonotonicCounter\RuntimeDxe\MonotonicCounter.inf
Component = Universal\Bds\Dxe\Bds.inf
Component= Universal\Security\Authorization\Dxe\Authorization.inf
Component = Universal\Security\SecurityStub\Dxe\SecurityStub.inf
Component = Csm\BiosThunk\BlockIo\Dxe\BiosBlockIo.inf
Component = Csm\BiosThunk\VgaMiniPort\Dxe\BiosVgaMiniPort.inf
Component = Csm\BiosThunk\Video\Dxe\BiosVideo.inf
Component = Platform\IntelEpg\Westville\ConPlatform\Dxe\ConPlatform.inf
Component = Universal\Console\ConSplitter\Dxe\ConSplitter.inf
Component = Universal\DeviceIo\Dxe\DeviceIo.inf
Component = Universal\Disk\DiskIo\Dxe\DiskIo.inf
Component = Universal\Disk\FileSystem\Fat\Dxe\Fat.inf
Component = Universal\Console\GraphicsConsole\Dxe\GraphicsConsole.inf
Component = Universal\Console\Terminal\Dxe\Terminal.inf
Component = Universal\Console\VgaClass\Dxe\VgaClass.inf
Component = Universal\DataHub\DataHubStdErr\Dxe\DataHubStdErr.inf
Component = Universal\Ebc\Dxe\Ebc.inf
Component = Universal\UserInterface\SetupBrowser\Dxe\SetupBrowser.inf
Component = Application\Shell\Shell.inf
Component= Universal\PasswordSecurity\Dxe\PasswordSecurity.inf

Component = Universal\Disk\Partition\Dxe\Partition.inf

Component = Platform\IntelEpg\Westville\AcpiTables\AcpiTables.inf
Component = Universal\Acpi\AcpiSupport\Dxe\AcpiSupport.inf

Component = Csm\BiosThunk\Keyboard\Dxe\BiosKeyboard.inf
Component = Csm\BiosThunk\Snp16\Dxe\BiosSnp16.inf
Component = Universal\ErrorManager\Dxe\ErrorManager.inf

Component = Universal\Disk\UnicodeCollation\English\Dxe\English.inf
Component = Universal\SmBios\Dxe\Smbios.inf

Component =
Platform\IntelEpg\Westville\SmbiosMiscSubclass\Dxe\MiscSubclassDriver.inf
Component = Platform\IntelEpg\Westville\Setup\Dxe\SetupDriver.inf

Component = Bus\Usb\UsbBot\Dxe\UsbBot.inf
Component = Bus\Usb\UsbCbi\Dxe\Cbi0\UsbCbi0.inf
Component = Bus\Usb\UsbKb\Dxe\UsbKb.inf
Component = Bus\Usb\UsbMassStorage\Dxe\UsbMassStorage.inf
Component = Bus\Usb\UsbMouse\Dxe\UsbMouse.inf

Component = Platform\IntelEpg\Westville\AcpiPlatform\Dxe\AcpiPlatform.inf
Component = Platform\IntelEpg\Westville\AcpiTable\AcpiTable.inf

Component = Universal\SmmRuntime\RuntimeDxe\SmmRuntime.inf

2.4.6 Optional PEIMs including Server Management drivers

Do not include in the ESS BIOSs.
Component = Bus\ServerManagement\PhysicalLayer\Ipmi\Pei\PeiIpmiBmc.inf
Component = Universal\Frb\Pei\PeiFrb.inf

2.4.7 Optional Drivers including Server Management drivers

Do not include in the ESS BIOSs.

Component = Universal\Frb\Dxe\FrbDriver.inf
Component = Universal\EmpDriver\Dxe\EmpDriver.inf
Component = Universal\Network\Bis\Dxe\BaseCode\Bis.inf
Component = Universal\BmcAcpi\Smm\BmcAcpiSwChild.inf
Component = Universal\ServerBmcVariable\Dxe\BmcVariableDriver.inf
Component = Universal\IpmiNetworkConfig\Dxe\IpmiNetworkConfig.inf
Component = Universal\LegacyConsole\Dxe\LegacyConsole.inf
Component = Cpu\Pentium\MpInit\Dxe\MpInit.inf
Component =
Universal\SmBios\Dxe\MemorySubClassDriver\DualChannelSdram\MemorySubClass.in
Component = Universal\Network\PxeDhcp4\Dxe\Dhcp4.inf

Component = Universal\Network\PxeBc\Dxe\Bc.inf
Component = Universal\SmDiagBoot\Dxe\SmDiagBootDriver.inf
Component = Universal\SmRemoteBootControl\Dxe\SmRemoteBootControl.inf
Component = Universal\Network\Snp32_64\Dxe\Snp.inf
Component = Universal\ServerManagement\ComLayer\Comlayer.inf
Component = Universal\ServerManagement\GenericElog\GenericElog.inf
Component = Universal\ServerManagement\GenericFru\GenericFru.inf
Component = Universal\ServerManagement\GenericSensor\GenericSensor.inf
Component = Bus\ServerManagement\PhysicalLayer\Ipmi\Dxe\IpmiBmc.inf
Component = Universal\ServerManagement\RedirElog\Bmc\BmcElog.inf
Component = Universal\ServerManagement\RedirElog\Fv\FvElog.inf
Component = Universal\ServerManagement\RedirSensor\IpmiRedirSensor.inf
Component = Universal\ServerManagement\RedirFru\IpmiRedirFru.inf
Component = Universal\ServerManagement\SmmErrorLog\SmmErrorLog.inf
Component = Universal\ServerManagement\Smtp\Smtp.inf
Component = Cpu\Logo\Xeon\Logo.inf – Do not include logo to always force diagnostic screen.

2.5 Libraries:

In general, leaving libraries in the CLF does not hurt. They are not included in any driver/PEIM unless the driver/PEIM requests it. In rare circumstances, building a library may give you a build error. In that case, leaving the library out will help. The following section divides the libraries into the necessary ones vs. the optional ones.

2.5.1 Mandatory Libraries

NoCode= Library
Library = Library\Dxe\Hob\HobLib.inf
Library = Library\Dxe\EfiUiLib\EfiUiLib.inf

2.5.2 Status code Libraries useful for debugging

The first one is always required.
Library = Platform\Generic\MonoStatusCode\Library\Pei\SimpleCpuIo\SimpleCpuIo.inf

Link the appropriate libraries to match the progress/error code display devices.

Library =
Platform\Generic\MonoStatusCode\Library\Pei\SerialStatusCode\SerialStatusCode.inf
and Library =
Platform\Generic\RuntimeDxe\StatusCode\Lib\BsSerialStatusCode\BsSerialStatusCode.inf - Sends debug prints to serial port

Library =
Platform\Generic\MonoStatusCode\Library\Pei\BeepStatusCode\BeepStatusCode.inf - Beeps on critical errors.

Library =
Platform\Generic\MonoStatusCode\Library\Pei\LedStatusCode\LedStatusCode.inf
Library =
Platform\Generic\RuntimeDxe\StatusCode\Lib\RtLedStatusCode\RtLedStatusCode.inf –
Logs status codes in memory for later retrieval

Library =
Platform\Generic\RuntimeDxe\StatusCode\Lib\BsDataHubStatusCode\BsDataHubStatusCode.inf – Logs the status code to data hub.

2.5.3 Optional Libraries

2.5.3.1 BIS libraries

These libraries add support for BIS (Boot Integrity Service) and will never be needed for poweron.
Library = Universal\Network\BIS\Dxe\oasis\src\addins\intel\cssmcl\CssmclLib.inf
Library = Universal\Network\BIS\Dxe\oasis\src\addins\intel\cssmcsp\CssmcspLib.inf
Library = Universal\Network\BIS\Dxe\oasis\src\addins\intel\cssmvl\CssmvlLib.inf
Library = Universal\Network\BIS\Dxe\oasis\src\fwk\cssm\CssmLib.inf
Library = Universal\Network\BIS\Dxe\oasis\src\fwk\port\PortLib.inf
Library = Universal\Network\BIS\Dxe\oasis\src\fwk\util\ber_der\r1_1\R1_1Lib.inf
Library = Universal\Network\BIS\Dxe\oasis\src\icl\IclLib.inf
Library = Universal\Network\BIS\Dxe\oasis\src\Integrity\IntegrityLib.inf