**UEFI is implemented on the Qualcomm platform.**

The Unified Extensible firmware interface, unified by Unified extensible firmware interface, is a standard that describes the type interface in detail.

The Extensfirmware Interface(Extensible FirmwareInterface,EFI)isIntel's recommended standard for the architecture, interface, and service of PC firmware. The main purpose is to provide a consistent, properly specified startup service on all platforms prior to OS loading (before startup) and is seen as a successor to the BIOS that has been around for nearly 20 years.

Since UEFI is meant to replace the BIOS, it naturally has its own advantages.

BioS is a compilation implementation, using a 16bit real-mode addressing method, with a maximum support of only 1Mof memory,limited code readability and implemented functionality, and inconvenient porting. The maximum disk space supported by the BIOS is no more than 2TB.

UEFI overcomes all of these disadvantages with a C-language implementation, layered, modular design, CPU and drive-independentness. UEFI can be understood as a complete system that contains power-up timing, drive implementation,os environment establishment (this os can be understood as UEFI running unique os,non-linux, windows),applications. Where the application supports network configuration, class shell environment,fastboot,linuxloader, etc.

The noun abbreviations involved in UEFI are : .

Unified extensible firmware interface (UEFI)

SEC:security .

PEI:pre EFI initialization .

DXE:driver execution Environment .

BDS:Boot Dev Select.

TSL:Transient System Load.

RT: runtime .

AL:after life .

GUID:Globally Unique Identifier .

CSM: The Compatibility Support Module.

TCG:Trust Ed Computing Group.

PE:portable executable .

COFF: Common object file format.

FV: Firmware Volume.

The implementation of Qualcomm's platform UEFI is divided into two parts: .

1. XBL contains chip-related protocols and key applications such as charging.)

2. ABL contains chip-independent applications such as fastboot and linuxloader.

The protocol concept is heavily used in UEFI code, which actually refers to the driver and contains the driver function pointer and data. Take the rampatition as an example: .

boot\_images/QcomPkg/Include/Protocol/EFIRamPartition.h declares the rampatition protocol: .

typedef.

EFI\_STATUS.

(EFIAPI isEFI\_RAMPARTITION\_GETRAMPARTITIONS.))(

INEFI\_RAMPARTITION\_PROTOCOL this,

OUTRamPartitionEntry sRamPartitions,

IN OUT UINT32 snumPartition.

struct\_EFI\_RAMPARTITION\_PROTOCOL . . .

   UINT64                                  Revision;

  EFI\_RAMPARTITION\_GETRAMPARTITIONVERSION GetRamPartitionVersion;

EFI\_RAMPARTITION\_GETHIGHESTBANKBIT Get Highest BankBit;

   EFI\_RAMPARTITION\_GETRAMPARTITIONS GetRamPartitions.;

};

This protocol was implemented in boot\_images/QcomPkg/Drivers/EnvDxe/EnvDxe.c.

STATIC EFI\_RAMPARTITION\_PROTOCOL RampartitionProtocol.

{

  EFI\_RAMPARTITION\_PROTOCOL\_REVISION,

EFI\_GetRamPartitionVersion.

EFI\_GetHighestBankBit.

EFI\_GetRamPartitions.

};

Protocol, i.e. drive, is implemented in XBL and can be used directly in ABL.

UEFI start-up process:

For the Qualcomm platform start-up process in turn, PBL-XBL-gt;ABL.

General customization is concentrated in the ABL, which is part of the code tree as follows:

Andrioid code path bootable/bootloader/edk2/QcomModulePkg.

Application.

- LinuxLoader.

LinuxLoader.c.

Linuxloader.inf.

-- Include.

- Library.

BoardCustom.h.

- Board.h.

S-BootImage.h.

S.--BootLinux.h.

S.-BootStats.h.

S---Decompress.h.

DeviceInfo.h.

S-DrawUI.h.

FastbootMenu.h.

Sfonts.h.

KeyPad.h.

S. -- LinuxLoaderLib.h.

- list.h.

S-LocateDeviceTree.h.

S.- MenuKeysDetection.h.

S-PartitionTableUpdate.h.

S. -- Recovery.h.

Reg.h.

- ShutdownServices.h.

S--StackCanary.h.

- UnlockMenu.h.

UpdateCmdLine.h.

UpdateDeviceTree.h.

VerifiedBootMenu.h.

- Protocol.

EFICardInfo.h.

EFI ChargerEx.h.

EFIChipInfo.h.

EFIChipInfoType.h.

EFIEraseBlock.h.

EFILimits.h.

EFIMdtp.h.

EFIPlatformInfo.h.

EFIPlatformInfoType.h.

EFIPmicPon.h.

EFIPmicVersion.h.

EFIQseecom.h.

EFIRamPartition.h.

EFIReasonReset.h.

EFIRng.h.

EFIScmModeSwitch.h.

EFIUsbDevice.h.

EFIUsbEx.h.

EFIVerifiedBoot.h.

UsbEx.h.

- Library.

- BootLib.

- Board.c.

S-BootLib.inf.

S.--BootLinux.c.

S.--BootStats.c.

- Decompress.c.

DeviceInfo.c.

S.S.-- DrawUI.c.

FastbootMenu.c.

KeyPad.c.

S. -- LinuxLoaderLib.c.

S-LocateDeviceTree.c.

S.- MenuKeysDetection.c.

- PartitionTableUpdate.c.

S. -- Recovery.c.

- ShutdownServices.c.

- UnlockMenu.c.

UpdateCmdLine.c.

UpdateDeviceTree.c.

- VerifiedBootMenu.c.

FastbootLib.

- FastbootCmds.c.

FastbootCmds.h.

FastbootLib.inf.

FastbootMain.c.

FastbootMain.h.

MetaFormat.h.

- SparseFormat.h.

S.-UsbDescriptors.c.

S.-UsbDescriptors.h.

StackCanary.

- StackCanary.c.

- StackCanary.inf.

- zlib.

- Adler32.c.

- inffast.c.

- inffast.h.

- inffixed.h.

- inflate.c.

- inflate.h.

inftrees.c.

- inftrees.h.

zconf.h.

Zlib.h.

Zlib.inf.

- zutil.c.

- zutil.h.

QcomModulePkg.dec.

QcomModulePkg.dsc.

QcomModulePkg.fdf.

Tools.

-- app\_path\_set.cmm.

-- check\_paths.cmm.

-- debug\_app.cmm.

-- elf\_tools.py.

-- image\_header.py.

-- load\_uefi\_dump.cmm.

-- log\_save.cmm.

-- symbol\_Load.cmm.

uefi\_core\_path\_set.cmm.

This section mainly supports the linux loader used to load linux, as well as fastboot. User modifications are mainly concentrated in these two parts, the portal function LinuxEntryLoader.

File descriptions in UEFI that use several suffix formats:

fdf:flash definitionfile, describing the range of flash partition addresses.

dec:package file, which defines THE GUID information for different modules.

dsc:description file, which mainly contains all the inf files that need to be used.

inf: Compilation information for a single module, similar to makefile.

efi: The resulting UEFI executable is finally compiled.