[UEFI Practice] Introduction to SlimBootloader

原创 jiangwei0512 Posted on 2018-10-20 11:03:02 Read 6.2k Collection 14 Likes 1

Category Column: UEFI Development Basics Article Tags: uefi slimbootloader

UEFI Development ... This column includes this content 136 articles Subscribe to our column

摘要 SlimBootloader is an open source bootloader developed by Intel. It is similar to the Intel version of coreboot and is designed for the x86 platform, emphasizing security and scalability. It can initialize hardware and load the operating system, supports a variety of common OS, and can be obtained from GitHub. This article introduces its compilation process, including the tools and steps required in the Windows environment, and how to test it on QEMU.

The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

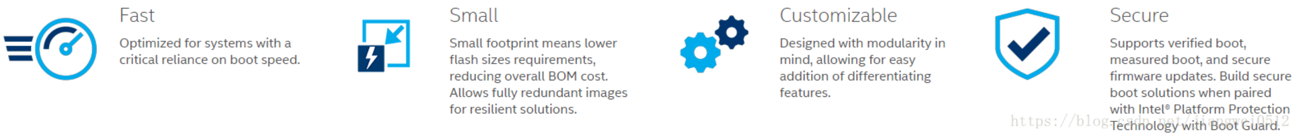
What is Slim Bootloader

Slim Bootloader is a Bootloader launched by Intel. A complete introduction to it can be found on the following website:

Overview: Intel® Slim Bootloader

It can be considered as the Intel version of coreboot, which is more inclined towards the x86 platform in terms of platform support, security, and scalability.

Slim Bootloader is the same as other Bootloaders, its main function is to initialize the hardware and load the OS. However, compared with other Bootloaders, Slim Bootloader has the following advantages:



Fast
Optimized for systems with a critical reliance on boot speed.

Small
Small footprint means lower flash sizes requirements, reducing overall BOM cost. Allows fully redundant images for resilient solutions.

Customizable
Designed with modularity in mind, allowing for easy addition of differentiating features.

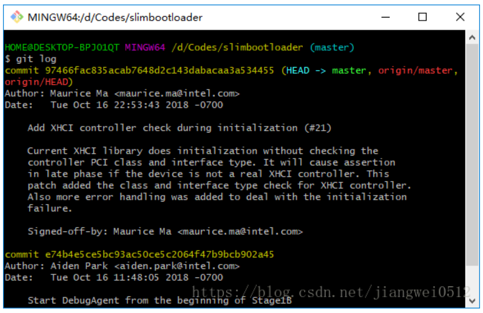
Secure
Supports verified boot, measured boot, and secure firmware updates. Build secure boot solutions when paired with Intel® Platform Protection Technology with Boot Guard.

It supports various common operating systems and is open source (using BSD license).

The source code for Slim Bootloader can be downloaded from GitHub - slimbootloader/slimbootloader: Visit http://slimbootloader.github.io for documentation .

Since Slim Bootloader is still in the early development stage, the code may differ due to updates. Please refer to the actual downloaded code.

The compilation and running mentioned in this article are based on the version downloaded on 20181020. The actual version is as follows:

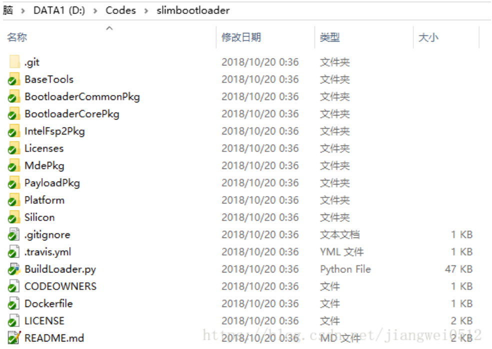


MINGW64:/d/Codes/slimbootloader

```
HOME@DESKTOP-BPJ01QT MINGW64 /d/Codes/slimbootloader (master)
$ git log
commit 97466fac835acab7648d2c143dabacaa3a534455 (HEAD -> master, origin/master,
origin/HEAD)
Author: Maurice Ma <maurice.ma@intel.com>
Date:   Tue Oct 16 22:53:43 2018 -0700

    Add XHCI controller check during initialization (#21)

    Current XHCI library does initialization without checking the
    controller PCI class and interface type. It will cause assertion
    in late phase if the device is not a real XHCI controller. This
    patch added the class and interface type check for XHCI controller.
    Also more error handling was added to deal with the initialization
    failure.

    Signed-off-by: Maurice Ma <maurice.ma@intel.com>

commit e74b4e5ce5bc93ac50ce5c2064f47b9bcb902a45
Author: Aiden Park <aiden.park@intel.com>
Date:   Tue Oct 16 11:48:05 2018 -0700

    Start DebugAgent from the beginning of Stage1B
```

Compilation of Slim Bootloader

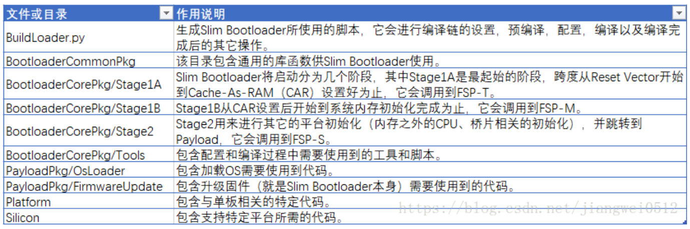
The code structure downloaded from the above github is as follows:



脑 › DATA1 (D:) › Codes › slimbootloader

名称	修改日期	类型	大小
.git	2018/10/20 0:36	文件夹	
BaseTools	2018/10/20 0:36	文件夹	
BootloaderCommonPkg	2018/10/20 0:36	文件夹	
BootloaderCorePkg	2018/10/20 0:36	文件夹	
IntelFsp2Pkg	2018/10/20 0:36	文件夹	
Licenses	2018/10/20 0:36	文件夹	
MdePkg	2018/10/20 0:36	文件夹	
PayloadPkg	2018/10/20 0:36	文件夹	
Platform	2018/10/20 0:36	文件夹	
Silicon	2018/10/20 0:36	文件夹	
.gitignore	2018/10/20 0:36	文本文档	1 KB
.travis.yml	2018/10/20 0:36	YML 文件	1 KB
BuildLoader.py	2018/10/20 0:36	Python File	47 KB
CODEOWNERS	2018/10/20 0:36	文件	1 KB
Dockerfile	2018/10/20 0:36	文件	1 KB
LICENSE	2018/10/20 0:36	文件	2 KB
README.md	2018/10/20 0:36	MD 文件	2 KB

You can see that it is actually very similar to the structure of EDK. The following are the more important parts:



文件或目录	作用说明
BuildLoader.py	生成Slim Bootloader所使用的脚本，它会进行编译链的设置，预编译，配置，编译以及编译完成后的其它操作。
BootloaderCommonPkg	该目录包含通用的库函数供Slim Bootloader使用。
BootloaderCorePkg/Stage1A	Slim Bootloader将启动分为几个阶段，其中Stage1A是最起始的阶段，跨度从Reset Vector开始到Cache-As-RAM（CAR）设置好为止，它会调用到FSP-T。
BootloaderCorePkg/Stage1B	Stage1B从CAR设置后开始到系统内存初始化完成为止，它会调用到FSP-M。
BootloaderCorePkg/Stage2	Stage2用来进行其它的平台初始化（内存之外的CPU、桥片相关的初始化），并跳转到Payload，它会调用到FSP-S。
BootloaderCorePkg/Tools	包含配置和编译过程中需要使用到的工具和脚本。
PayloadPkg/OsLoader	包含加载OS需要使用到代码。
PayloadPkg/FirmwareUpdate	包含升级固件（就是Slim Bootloader本身）需要使用到的代码。
Platform	包含与单板相关的特定代码。
Silicon	包含支持特定平台所需的代码。

Regarding the above directories, there are some additional requirements to make the code easy to expand and compatible

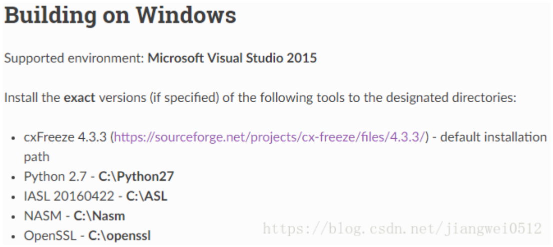
1. BootloaderCorePkg and PayloadPkg should be independent and unrelated to each other.

2. PayloadPkg code should not depend on Platform or Silicon.

3. PayloadPkg should only depend on BootloaderCommonPkg.

Compilation under Windows

Compilation under Windows requires the following preparation:



Building on Windows

Supported environment: **Microsoft Visual Studio 2015**

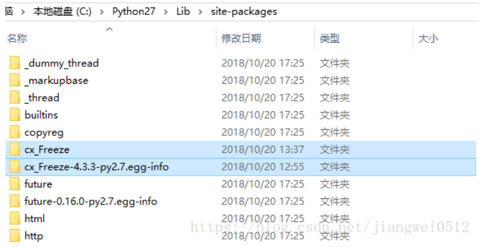
Install the **exact** versions (if specified) of the following tools to the designated directories:

- cxFreeze 4.3.3 (https://sourceforge.net/projects/cx-freeze/files/4.3.3/) - default installation path
- Python 2.7 - **C:\Python27**
- IASL 20160422 - **C:\ASL**
- NASM - **C:\Nasm**
- OpenSSL - **C:\openssl**

What needs to be explained here are the first and the last ones, while the ones in the middle have been explained in other articles and will not be introduced here.

cxFreeze is used to convert Python scripts into executable files under Windows. These executable files are located in the BaseTools\Bin\Win32 directory. There is no Win32 directory in the default source code.

The location of cxFreeze after installation is as follows:

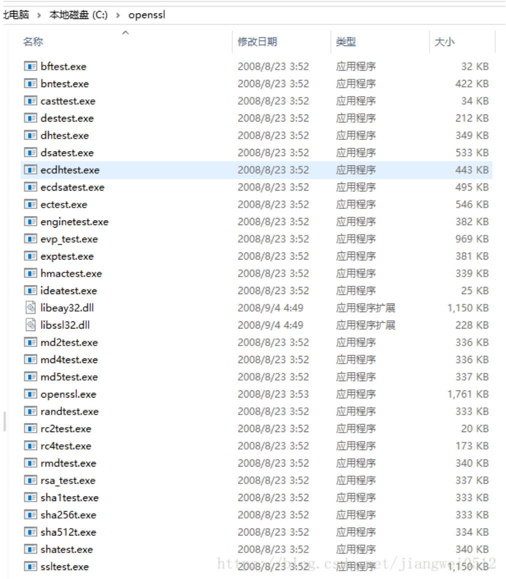


脑 › 本地磁盘 (C:) › Python27 › Lib › site-packages

名称	修改日期	类型	大小
_dummy_thread	2018/10/20 17:25	文件夹	
_markupbase	2018/10/20 17:25	文件夹	
_thread	2018/10/20 17:25	文件夹	
builtins	2018/10/20 17:25	文件夹	
copyreg	2018/10/20 17:25	文件夹	
cx_Freeze	2018/10/20 13:37	文件夹	
cx_Freeze-4.3.3-py2.7.egg-info	2018/10/20 12:55	文件夹	
future	2018/10/20 17:25	文件夹	
future-0.16.0-py2.7.egg-info	2018/10/20 17:25	文件夹	
html	2018/10/20 17:25	文件夹	
http	2018/10/20 17:25	文件夹	

Another one is openssl. The Windows version of openssl can be downloaded from OpenSSL for Windows .

Move the bin directory of the downloaded compressed package to the root directory of disk C, and rename bin to openssl, as shown in the following figure:

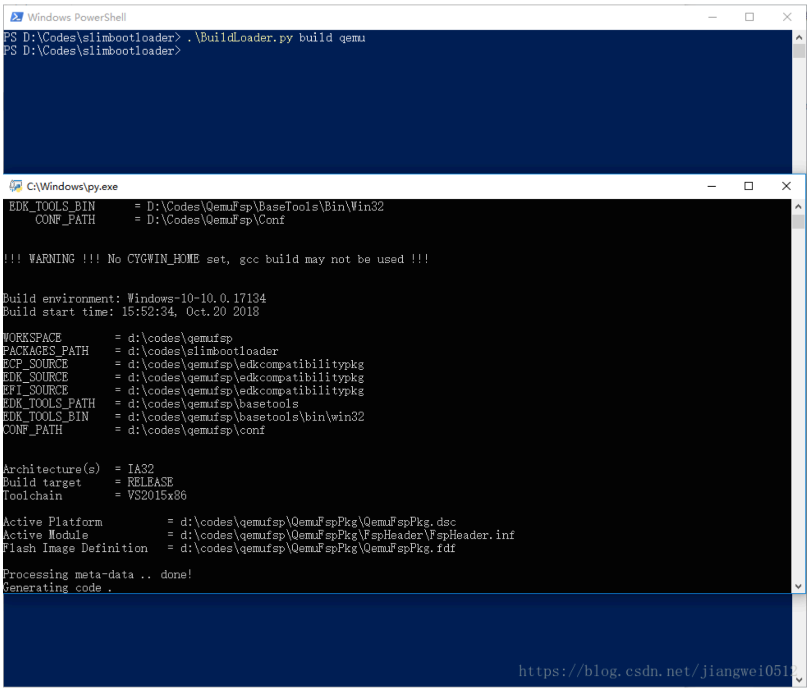


电脑 › 本地磁盘 (C:) › openssl

名称	修改日期	类型	大小
bftest.exe	2008/8/23 3:52	应用程序	32 KB
bntest.exe	2008/8/23 3:52	应用程序	422 KB
casttest.exe	2008/8/23 3:52	应用程序	34 KB
destest.exe	2008/8/23 3:52	应用程序	212 KB
dhtest.exe	2008/8/23 3:52	应用程序	349 KB
dsatest.exe	2008/8/23 3:52	应用程序	533 KB
ecdhtest.exe	2008/8/23 3:52	应用程序	443 KB
ecdsatest.exe	2008/8/23 3:52	应用程序	495 KB
ectest.exe	2008/8/23 3:52	应用程序	546 KB
enginetest.exe	2008/8/23 3:52	应用程序	382 KB
evp_test.exe	2008/8/23 3:52	应用程序	969 KB
exptest.exe	2008/8/23 3:52	应用程序	381 KB
hmactest.exe	2008/8/23 3:52	应用程序	339 KB
ideatest.exe	2008/8/23 3:52	应用程序	25 KB
libeay32.dll	2008/9/4 4:49	应用程序扩展	1,150 KB
libssl32.dll	2008/9/4 4:49	应用程序扩展	228 KB
md2test.exe	2008/8/23 3:52	应用程序	336 KB
md4test.exe	2008/8/23 3:52	应用程序	336 KB
md5test.exe	2008/8/23 3:52	应用程序	337 KB
openssl.exe	2008/8/23 3:53	应用程序	1,761 KB
randtest.exe	2008/8/23 3:52	应用程序	333 KB
rc2test.exe	2008/8/23 3:52	应用程序	20 KB
rc4test.exe	2008/8/23 3:52	应用程序	173 KB
rmdtest.exe	2008/8/23 3:52	应用程序	340 KB
rsa_test.exe	2008/8/23 3:52	应用程序	337 KB
sha1test.exe	2008/8/23 3:52	应用程序	333 KB
sha256t.exe	2008/8/23 3:52	应用程序	333 KB
sha512t.exe	2008/8/23 3:52	应用程序	334 KB
shatest.exe	2008/8/23 3:52	应用程序	340 KB
ssltest.exe	2008/8/23 3:52	应用程序	1,150 KB

Since there is no actual board to test, only the Slim Bootloader for QEMU is compiled for testing. The specific method is as follows:

1. Enter the source code root directory, open a command line window, and run the following command:



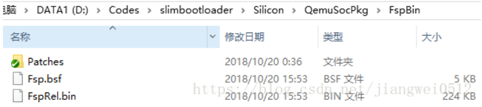
```
PS D:\Codes\slimbootloader> .\BuildLoader.py build qemu
PS D:\Codes\slimbootloader>
```

```
 EDK_TOOLS_BIN      = D:\Codes\QemuFsp\BaseTools\Bin\Win32
     CONF_PATH      = D:\Codes\QemuFsp\Conf


!!! WARNING !!! No CYGWIN_HOME set, gcc build may not be used !!!


Build environment: Windows-10-10.0.17134
Build start time: 15:52:34, Oct.20 2018

WORKSPACE          = d:\codes\qemufsp
PACKAGES_PATH      = d:\codes\slimbootloader
ECP_SOURCE         = d:\codes\qemufsp\edkcompatibilitypkg
EDK_SOURCE         = d:\codes\qemufsp\edkcompatibilitypkg
EFI_SOURCE         = d:\codes\qemufsp\edkcompatibilitypkg
EDK_TOOLS_PATH     = d:\codes\qemufsp\basetools
EDK_TOOLS_BIN      = d:\codes\qemufsp\basetools\bin\win32
CONF_PATH          = d:\codes\qemufsp\conf


Architecture(s)    = IA32
Build target       = RELEASE
Toolchain          = VS2015x86

Active Platform          = d:\codes\qemufsp\QemuFspPkg\QemuFspPkg.dsc
Active Module            = d:\codes\qemufsp\QemuFspPkg\FspHeader\FspHeader.inf
Flash Image Definition   = d:\codes\qemufsp\QemuFspPkg\QemuFspPkg.fdf

Processing meta-data .. done!
Generating code .
```

FSP will be compiled here first (if not prepared in advance, it will be downloaded here).

For an introduction to FSP, please refer to [UEFI Practice] FSP .

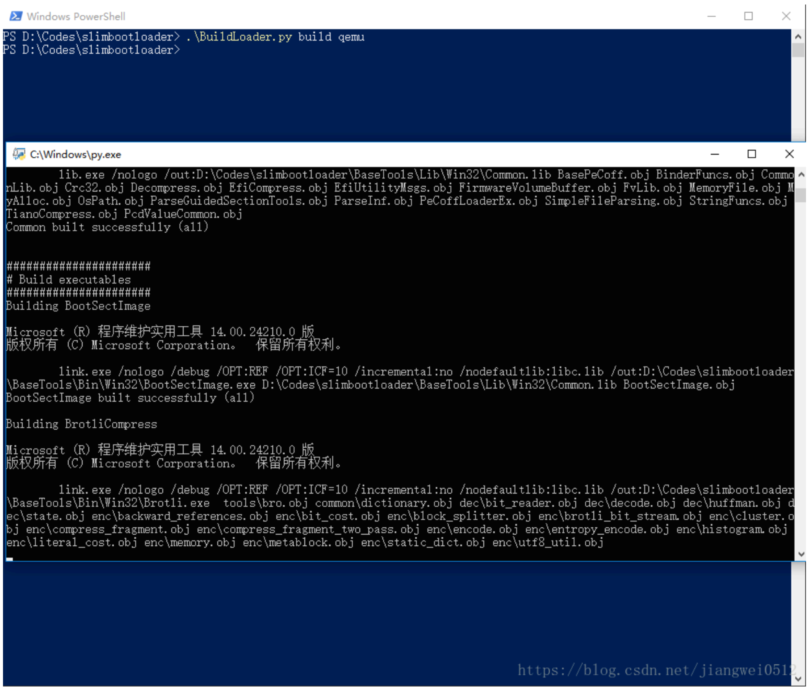
After compilation is complete, the generated files will be placed in the following location:



脑 › DATA1 (D:) › Codes › slimbootloader › Silicon › QemuSocPkg › FspBin

名称	修改日期	类型	大小
Patches	2018/10/20 0:36	文件夹	
Fsp.bsf	2018/10/20 15:53	BSF 文件	5 KB
FspRel.bin	2018/10/20 15:53	BIN 文件	224 KB

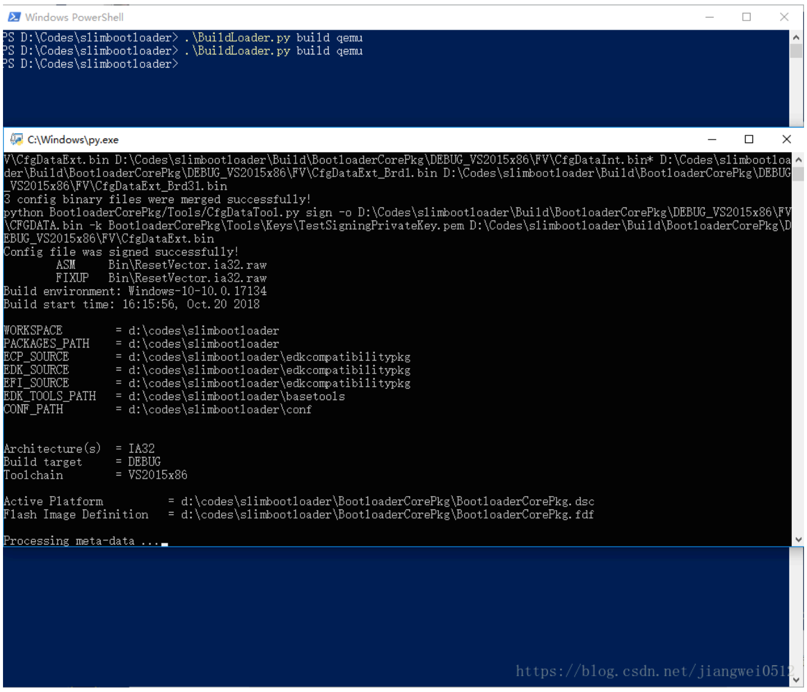
After that, we will start compiling Slim Bootloader. However, we need to compile Win32 tools first (the generated tools are located in BaseTools\Bin\Win32):



```
PS D:\Codes\slimbootloader> .\BuildLoader.py build qemu
PS D:\Codes\slimbootloader>
```

```
        lib.exe /nologo /out:D:\Codes\slimbootloader\BaseTools\Lib\Win32\Common.lib BasePeCoff.obj BinderFuncs.obj Commo
nLib.obj Crc32.obj Decompress.obj EfiCompress.obj EfiUtilityMsgs.obj FirmwareVolumeBuffer.obj FvLib.obj MemoryFile.obj M
yAlloc.obj OsPath.obj ParseGuidedSectionTools.obj ParseInf.obj PeCoffLoaderEx.obj SimpleFileParsing.obj StringFuncs.obj
TianoCompress.obj PcdValueCommon.obj
Common built successfully (all)


######################
# Build executables
######################
Building BootSectImage

Microsoft (R) 程序维护实用工具 14.00.24210.0 版
版权所有 (C) Microsoft Corporation。  保留所有权利。

        link.exe /nologo /debug /OPT:REF /OPT:ICF=10 /incremental:no /nodefaultlib:libc.lib /out:D:\Codes\slimbootloader
\BaseTools\Bin\Win32\BootSectImage.exe D:\Codes\slimbootloader\BaseTools\Lib\Win32\Common.lib BootSectImage.obj
BootSectImage built successfully (all)

Building BrotliCompress

Microsoft (R) 程序维护实用工具 14.00.24210.0 版
版权所有 (C) Microsoft Corporation。  保留所有权利。

        link.exe /nologo /debug /OPT:REF /OPT:ICF=10 /incremental:no /nodefaultlib:libc.lib /out:D:\Codes\slimbootloader
\BaseTools\Bin\Win32\Brotli.exe  tools\bro.obj common\dictionary.obj dec\bit_reader.obj dec\decode.obj dec\huffman.obj d
ec\state.obj enc\backward_references.obj enc\bit_cost.obj enc\block_splitter.obj enc\brotli_bit_stream.obj enc\cluster.o
bj enc\compress_fragment.obj enc\compress_fragment_two_pass.obj enc\encode.obj enc\entropy_encode.obj enc\histogram.obj
enc\literal_cost.obj enc\memory.obj enc\metablock.obj enc\static_dict.obj enc\utf8_util.obj
```

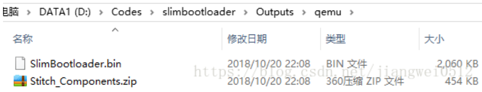
Then continue compiling:



```
PS D:\Codes\slimbootloader> .\BuildLoader.py build qemu
PS D:\Codes\slimbootloader> .\BuildLoader.py build qemu
PS D:\Codes\slimbootloader>
```

```
V\CfgDataExt.bin D:\Codes\slimbootloader\Build\BootloaderCorePkg\DEBUG_VS2015x86\FV\CfgDataInt.bin* D:\Codes\slimbootloa
der\Build\BootloaderCorePkg\DEBUG_VS2015x86\FV\CfgDataExt_Brd1.bin D:\Codes\slimbootloader\Build\BootloaderCorePkg\DEBUG
_VS2015x86\FV\CfgDataExt_Brd31.bin
3 config binary files were merged successfully!
python BootloaderCorePkg/Tools/CfgDataTool.py sign -o D:\Codes\slimbootloader\Build\BootloaderCorePkg\DEBUG_VS2015x86\FV
\CFGDATA.bin -k BootloaderCorePkg\Tools\Keys\TestSigningPrivateKey.pem D:\Codes\slimbootloader\Build\BootloaderCorePkg\D
EBUG_VS2015x86\FV\CfgDataExt.bin
Config file was signed successfully!
        ASM     Bin\ResetVector.ia32.raw
        FIXUP   Bin\ResetVector.ia32.raw
Build environment: Windows-10-10.0.17134
Build start time: 16:15:56, Oct.20 2018

WORKSPACE        = d:\codes\slimbootloader
PACKAGES_PATH    = d:\codes\slimbootloader
ECP_SOURCE       = d:\codes\slimbootloader\edkcompatibilitypkg
EDK_SOURCE       = d:\codes\slimbootloader\edkcompatibilitypkg
EFI_SOURCE       = d:\codes\slimbootloader\edkcompatibilitypkg
EDK_TOOLS_PATH   = d:\codes\slimbootloader\basetools
CONF_PATH        = d:\codes\slimbootloader\conf


Architecture(s)  = IA32
Build target     = DEBUG
Toolchain        = VS2015x86

Active Platform          = d:\codes\slimbootloader\BootloaderCorePkg\BootloaderCorePkg.dsc
Flash Image Definition   = d:\codes\slimbootloader\BootloaderCorePkg\BootloaderCorePkg.fdf

Processing meta-data ...
```

The binary generated by the final compilation is located in the Outputs directory under the root directory:



电脑 › DATA1 (D:) › Codes › slimbootloader › Outputs › qemu ›

名称	修改日期	类型	大小
SlimBootloader.bin	2018/10/20 22:08	BIN 文件	2,060 KB
Stitch_Components.zip	2018/10/20 22:08	360压缩 ZIP 文件	454 KB

Compilation under Linux

Not available at this time.

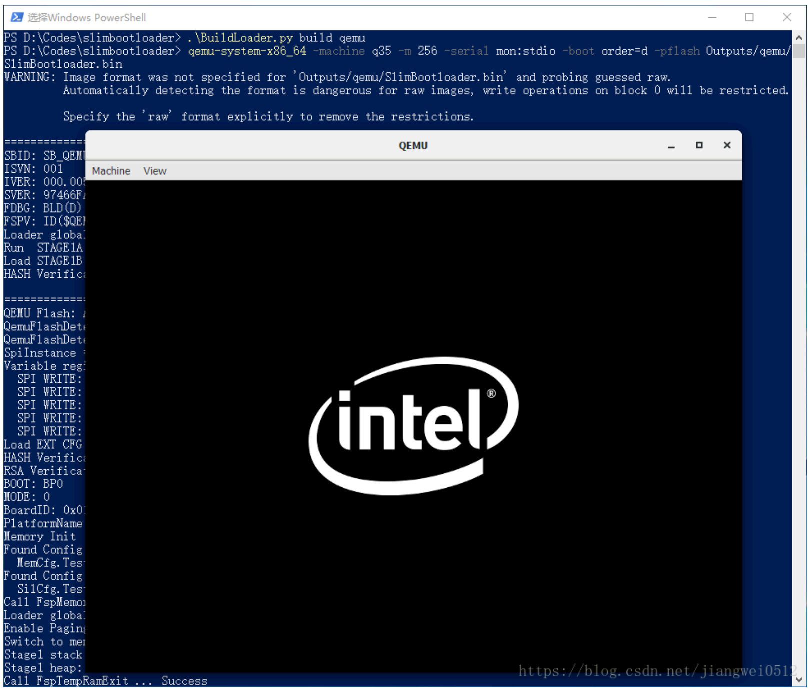
Use of Slim Bootloader

After the compilation is complete, execute the following command:

AI generated projects 登录复制

```
qemu-system-x86_64 -machine q35 -m 256 -serial mon:stdio -boot order=d -pflash Outputs/qemu/SlimBootloader.bin
```

The results are as follows:



```
PS D:\Codes\slimbootloader> .\BuildLoader.py build qemu
PS D:\Codes\slimbootloader> qemu-system-x86_64 -machine q35 -m 256 -serial mon:stdio -boot order=d -pflash Outputs/qemu/
SlimBootloader.bin
WARNING: Image format was not specified for 'Outputs/qemu/SlimBootloader.bin' and probing guessed raw.
         Automatically detecting the format is dangerous for raw images, write operations on block 0 will be restricted.

         Specify the 'raw' format explicitly to remove the restrictions.

============
SBID: SB_QEMU
ISVN: 001
IVER: 000.005
SVER: 97466F
FDBG: BLD(D)
FSPV: ID($QEM
Loader global
Run  STAGE1A
Load STAGE1B
HASH Verifica

============
QEMU Flash: 
QemuFlashDete
QemuFlashDete
SpiInstance =
Variable regi
  SPI WRITE:
  SPI WRITE:
  SPI WRITE:
  SPI WRITE:
  SPI WRITE:
Load EXT CFG
HASH Verifica
RSA Verifica
BOOT: BP0
MODE: 0
BoardID: 0x0
PlatformName
Memory Init
Found Config
  MemCfg.Tes
Found Config
  SilCfg.Tes
Call FspMemor
Loader global
Enable Paging
Switch to mem
Stage1 stack
Stage1 heap:
Call FspTempRamExit ... Success
```

Note that QEMU must have been installed before this.

Since there is no actual system to boot, you can only see the following interface, but the Slim Bootloader is booted normally.