[UEFI Practice] SlimBootloader Usage

原创 jiangwei0512 Posted on 2021-10-01 22:51:03 Read 2.9k Collection 5 Likes 2

Category Column: UEFI Development Basics Article Tags: slimbootloader

摘要 This article details the latest developments of Intel SlimBootloader, including its features as an open source platform boot loader, source code acquisition, build process, and practical application of compiling using Python scripts and QEMU platform in Windows environment.

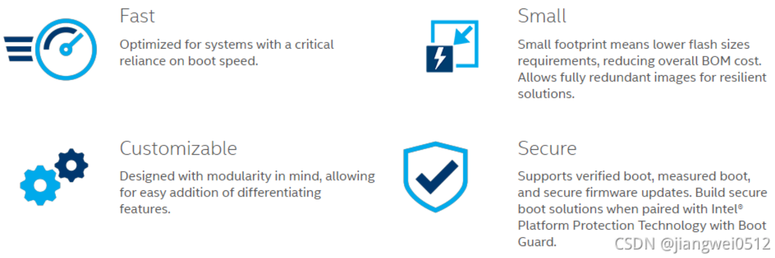
The summary is generated in C Know , supported by DeepSeek-R1 full version, go to experience>

illustrate

The basic contents of Slim Bootloader have been introduced in [UEFI Practice] Slim Bootloader Introduction, but because it is a long time ago, many contents have expired and cannot be used, so here is an updated version .

Brief description

Intel® Slim Bootloader, referred to as Intel® SBL (hereinafter referred to as SBL), is a bootloader launched by Intel. A complete introduction to it can be found at the following website: https://www.intel.com/content/www/us/en/design/products-and-solutions/technologies/slim-bootloader/overview.html . It can be considered as the Intel version of coreboot, which is more inclined to the x86 platform in terms of platform support, security, and scalability. Its features are as follows:



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

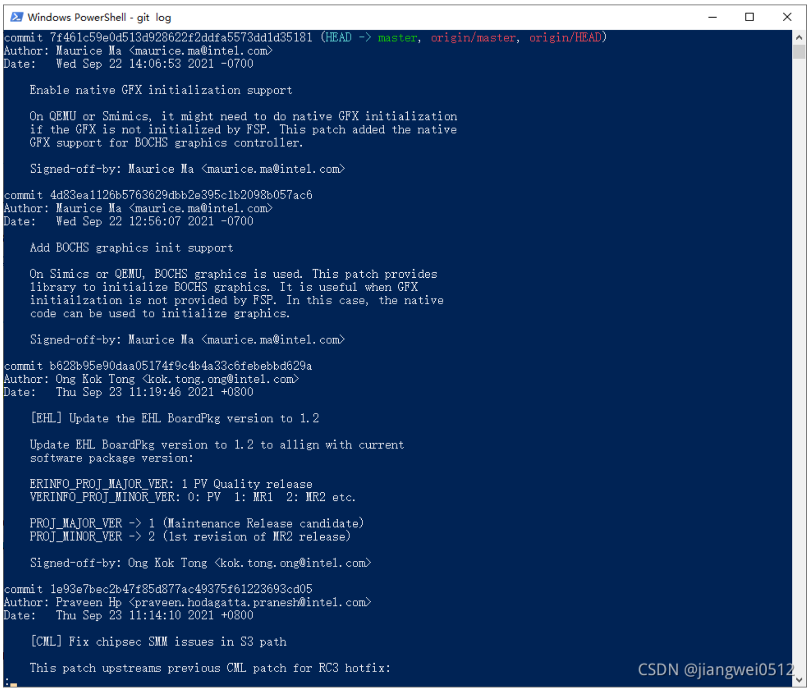
Supports verified boot, measured boot, and secure firmware updates. Build secure boot solutions when paired with Intel® Platform Protection Technology with Boot Guard.

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

Source code download

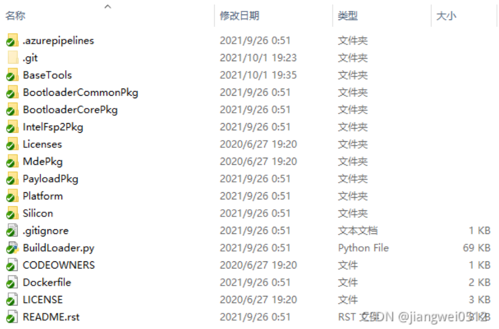
The source code of SBL can be downloaded from https://github.com/slimbootloader/slimbootloader . There is also a mirror on Gitee at https://gitee.com/jiangwei0512/slimbootloader .

Since the SBL version is constantly evolving, the code may differ due to updates. Please refer to the actual downloaded code. The current version is:



```
commit 7f461c59e0d513d928622f2ddfa5573dd1d35181 (HEAD -> master, origin/master, origin/HEAD)
Author: Maurice Ma <maurice.ma@intel.com>
Date:   Wed Sep 22 14:06:53 2021 -0700

    Enable native GFX initialization support

    On QEMU or Smimics, it might need to do native GFX initialization
    if the GFX is not initialized by FSP. This patch added the native
    GFX support for BOCHS graphics controller.

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

commit 4d83ea1126b5763629dbb2e395c1b2098b057ac6
Author: Maurice Ma <maurice.ma@intel.com>
Date:   Wed Sep 22 12:56:07 2021 -0700

    Add BOCHS graphics init support

    On Simics or QEMU, BOCHS graphics is used. This patch provides
    library to initialize BOCHS graphics. It is useful when GFX
    initiailzation is not provided by FSP. In this case, the native
    code can be used to initialize graphics.

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

commit b628b95e90daa05174f9c4b4a33c6febebbd629a
Author: Ong Kok Tong <kok.tong.ong@intel.com>
Date:   Thu Sep 23 11:19:46 2021 +0800

    [EHL] Update the EHL BoardPkg version to 1.2

    Update EHL BoardPkg version to 1.2 to allign with current
    software package version:

    ERINFO_PROJ_MAJOR_VER: 1 PV Quality release
    VERINFO_PROJ_MINOR_VER: 0: PV  1: MR1  2: MR2 etc.

    PROJ_MAJOR_VER -> 1 (Maintenance Release candidate)
    PROJ_MINOR_VER -> 2 (1st revision of MR2 release)

    Signed-off-by: Ong Kok Tong <kok.tong.ong@intel.com>

commit 1e93e7bec2b47f85d877ac49375f61223693cd05
Author: Praveen Hp <praveen.hodagatta.pranesh@intel.com>
Date:   Thu Sep 23 11:14:10 2021 +0800

    [CML] Fix chipsec SMM issues in S3 path

    This patch upstreams previous CML patch for RC3 hotfix:
:
```

The downloaded source code directory structure is as follows:



名称	修改日期	类型	大小
.azurepipelines	2021/9/26 0:51	文件夹	
.git	2021/10/1 19:23	文件夹	
BaseTools	2021/10/1 19:35	文件夹	
BootloaderCommonPkg	2021/9/26 0:51	文件夹	
BootloaderCorePkg	2021/9/26 0:51	文件夹	
IntelFsp2Pkg	2021/9/26 0:51	文件夹	
Licenses	2020/6/27 19:20	文件夹	
MdePkg	2020/6/27 19:20	文件夹	
PayloadPkg	2021/9/26 0:51	文件夹	
Platform	2021/9/26 0:51	文件夹	
Silicon	2021/9/26 0:51	文件夹	
.gitignore	2021/9/26 0:51	文本文档	1 KB
BuildLoader.py	2021/9/26 0:51	Python File	69 KB
CODEOWNERS	2020/6/27 19:20	文件	1 KB
Dockerfile	2021/9/26 0:51	文件	2 KB
LICENSE	2020/6/27 19:20	文件	3 KB
README.rst	2021/9/26 0:51	RST 文件	3 KB

The following is a brief description of the main files or directories:

File or Directory	illustrate
BaseTools	The actual tool used to generate the SBL binary is a part of the tools located in BootloaderCorePkg\Tools.
BootloaderCommonPkg	Contains common library functions for use by SBL.
BootloaderCorePkg	SBL divides the startup into several different stages, and their main codes are included in this directory: Stage1A: The initial stage, spanning from the Reset Vector to the setting of Case As RAM (CAR), it will call FSP-T; Stage1B: From the setting of CAR to the completion of memory initialization, it will call FSP-M; Stage2: Perform other platform initialization (CPU and Chipset-related initialization outside of memory), and jump to Payload, it will call FSP-S.
IntelFsp2Pkg	FSP support package.
MdePkg	EDKII universal code.
PayloadPkg	Contains code for loading OS and upgrading firmware.
Platform	The code supported by the platform.
Silicon	SoC support code.
BuildLoader.py	Generates the script used by the Slim Bootloader binary, which will set up the build chain, precompile, compile, and other operations after the build is completed.

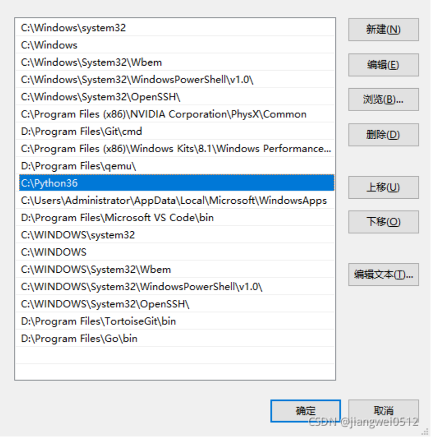
The code organization logic of SBL is to maintain the independence and scalability of the modules. The following requirements must be guaranteed:

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.

Building the SBL binary

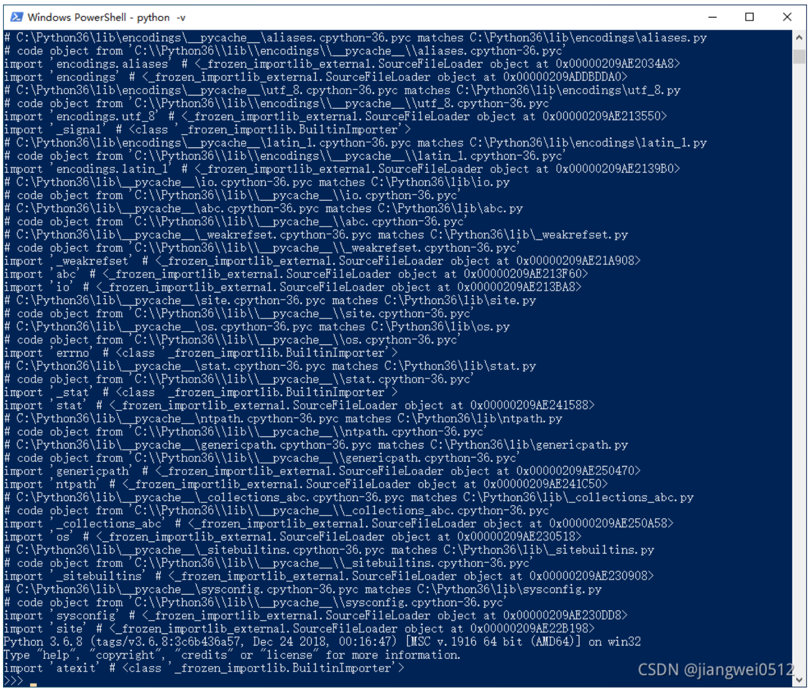
Windows 10 is used here to build SBL. Usually, the binary generated by compiling under Windows is smaller than that using GCC. For Intel platforms, UEFI development under Windows is more common, which is why Windows is used.

SBL uses Python scripts `BuildLoader.py` to complete the entire build process. The current SBL version uses Python3, so you need to install Python3 and put the execution path in the environment variable:



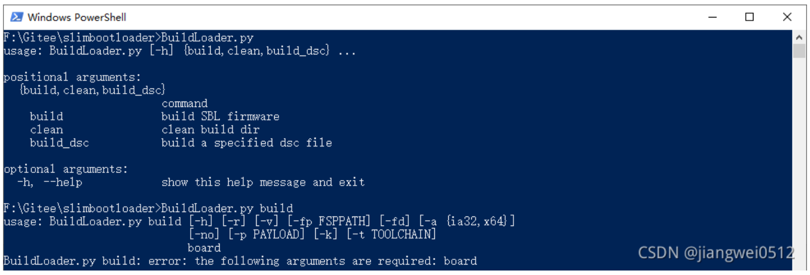
```
C:\Windows\system32
C:\Windows
C:\Windows\System32\Wbem
C:\Windows\System32\WindowsPowerShell\v1.0\
C:\Windows\System32\OpenSSH\
C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common
D:\Program Files\Git\cmd
C:\Program Files (x86)\Windows Kits\8.1\Windows Performance...
D:\Program Files\qemu\
C:\Python36
C:\Users\Administrator\AppData\Local\Microsoft\WindowsApps
D:\Program Files\Microsoft VS Code\bin
C:\WINDOWS\system32
C:\WINDOWS
C:\WINDOWS\System32\Wbem
C:\WINDOWS\System32\WindowsPowerShell\v1.0\
C:\WINDOWS\System32\OpenSSH\
D:\Program Files\TortoiseGit\bin
D:\Program Files\Go\bin
```

The version information is as follows (`Ctrl+z` + `Enter` exit Python interface):



```
# C:\Python36\lib\encodings\__pycache__\aliases.cpython-36.pyc matches C:\Python36\lib\encodings\aliases.py
# code object from 'C:\\Python36\\lib\\encodings\\__pycache__\\aliases.cpython-36.pyc'
import 'encodings.aliases' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE2034A8>
import 'encodings' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209ADDBDDA0>
# C:\Python36\lib\encodings\__pycache__\utf_8.cpython-36.pyc matches C:\Python36\lib\encodings\utf_8.py
# code object from 'C:\\Python36\\lib\\encodings\\__pycache__\\utf_8.cpython-36.pyc'
import 'encodings.utf_8' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE213550>
import '_signal' # <class '_frozen_importlib.BuiltinImporter'>
# C:\Python36\lib\encodings\__pycache__\latin_1.cpython-36.pyc matches C:\Python36\lib\encodings\latin_1.py
# code object from 'C:\\Python36\\lib\\encodings\\__pycache__\\latin_1.cpython-36.pyc'
import 'encodings.latin_1' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE2139B0>
# C:\Python36\lib\__pycache__\io.cpython-36.pyc matches C:\Python36\lib\io.py
# code object from 'C:\\Python36\\lib\\__pycache__\\io.cpython-36.pyc'
# C:\Python36\lib\__pycache__\abc.cpython-36.pyc matches C:\Python36\lib\abc.py
# code object from 'C:\\Python36\\lib\\__pycache__\\abc.cpython-36.pyc'
# C:\Python36\lib\__pycache__\_weakrefset.cpython-36.pyc matches C:\Python36\lib\_weakrefset.py
# code object from 'C:\\Python36\\lib\\__pycache__\\_weakrefset.cpython-36.pyc'
import '_weakrefset' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE21A908>
import 'abc' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE213F60>
import 'io' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE213BA8>
# C:\Python36\lib\__pycache__\site.cpython-36.pyc matches C:\Python36\lib\site.py
# code object from 'C:\\Python36\\lib\\__pycache__\\site.cpython-36.pyc'
# C:\Python36\lib\__pycache__\os.cpython-36.pyc matches C:\Python36\lib\os.py
# code object from 'C:\\Python36\\lib\\__pycache__\\os.cpython-36.pyc'
import 'errno' # <class '_frozen_importlib.BuiltinImporter'>
# C:\Python36\lib\__pycache__\stat.cpython-36.pyc matches C:\Python36\lib\stat.py
# code object from 'C:\\Python36\\lib\\__pycache__\\stat.cpython-36.pyc'
import '_stat' # <class '_frozen_importlib.BuiltinImporter'>
import 'stat' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE241588>
# C:\Python36\lib\__pycache__\ntpath.cpython-36.pyc matches C:\Python36\lib\ntpath.py
# code object from 'C:\\Python36\\lib\\__pycache__\\ntpath.cpython-36.pyc'
# C:\Python36\lib\__pycache__\genericpath.cpython-36.pyc matches C:\Python36\lib\genericpath.py
# code object from 'C:\\Python36\\lib\\__pycache__\\genericpath.cpython-36.pyc'
import 'genericpath' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE250470>
import 'ntpath' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE241C50>
# C:\Python36\lib\__pycache__\_collections_abc.cpython-36.pyc matches C:\Python36\lib\_collections_abc.py
# code object from 'C:\\Python36\\lib\\__pycache__\\_collections_abc.cpython-36.pyc'
import '_collections_abc' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE250A58>
import 'os' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE230518>
# C:\Python36\lib\__pycache__\_sitebuiltins.cpython-36.pyc matches C:\Python36\lib\_sitebuiltins.py
# code object from 'C:\\Python36\\lib\\__pycache__\\_sitebuiltins.cpython-36.pyc'
import '_sitebuiltins' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE230908>
# C:\Python36\lib\__pycache__\sysconfig.cpython-36.pyc matches C:\Python36\lib\sysconfig.py
# code object from 'C:\\Python36\\lib\\__pycache__\\sysconfig.cpython-36.pyc'
import 'sysconfig' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE230DD8>
import 'site' # <_frozen_importlib_external.SourceFileLoader object at 0x00000209AE22B198>
Python 3.6.8 (tags/v3.6.8:3c6b436a57, Dec 24 2018, 00:16:47) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
import 'atexit' # <class '_frozen_importlib.BuiltinImporter'>
>>>
```

Then you can see how to use it by executing `BuildLoader.py` the script:



```
F:\Gitee\slimbootloader>BuildLoader.py
usage: BuildLoader.py [-h] {build,clean,build_dsc} ...

positional arguments:
  {build,clean,build_dsc}
                        command
    build               build SBL firmware
    clean               clean build dir
    build_dsc           build a specified dsc file

optional arguments:
  -h, --help            show this help message and exit

F:\Gitee\slimbootloader>BuildLoader.py build
usage: BuildLoader.py build [-h] [-r] [-v] [-fp FSPPATH] [-fd] [-a {ia32,x64}]
                            [-no] [-p PAYLOAD] [-k] [-t TOOLCHAIN]
                            board
BuildLoader.py build: error: the following arguments are required: board
```

The command used to build the SBL binary is (no XXX is specified in the above figure, so an error is reported):

bash AI generated projects 登录复制

```bash
BuildLoader.py build XXX
```

`XXX` Refers to the target platform, which can be determined by looking `Platform\xxxBoardPkg\BoardConfig.py` at `self.BOARD_NAME` . For example, in this example, the SBL binary used to build QEMU corresponds to `Platform\QemuBoardPkg\BoardConfig.py` , where `self.BOARD_NAME` the value is as follows:

bash AI generated projects 登录复制

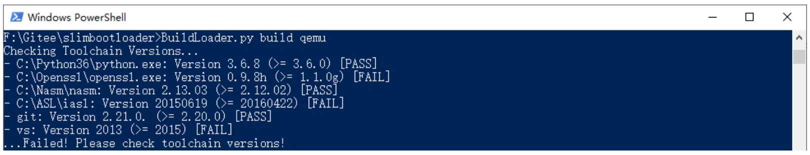
```bash
self.BOARD_NAME           = 'qemu'
```

So the actual build instructions are:

bash AI generated projects 登录复制

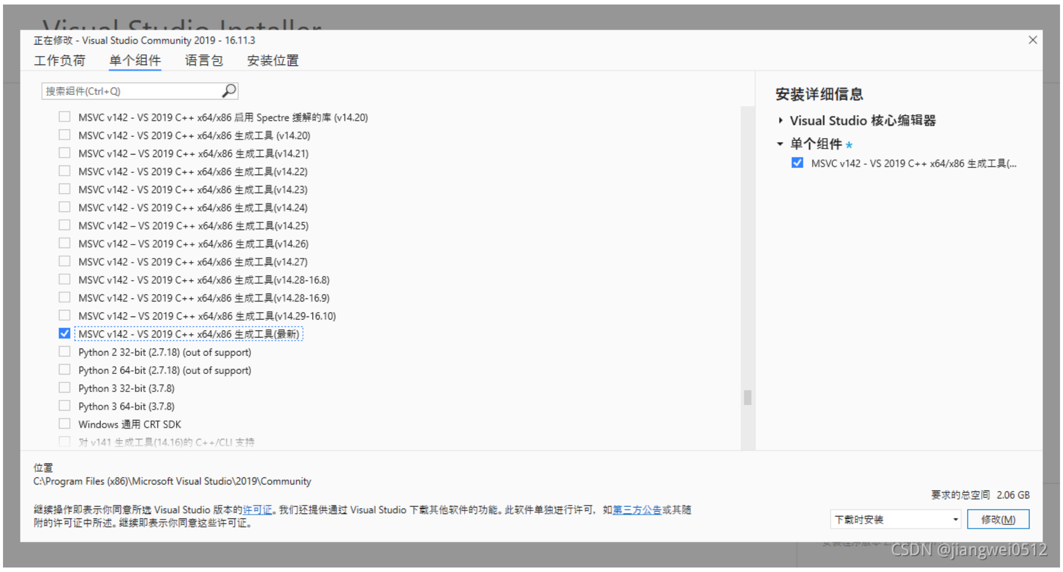
```bash
BuildLoader.py build qemu
```

The execution results are as follows:



```
F:\Gitee\slimbootloader>BuildLoader.py build qemu
Checking Toolchain Versions...
- C:\Python36\python.exe: Version 3.6.8 (>= 3.6.0) [PASS]
- C:\Openssl\openssl.exe: Version 0.9.8h (>= 1.1.0g) [FAIL]
- C:\Nasm\nasm: Version 2.13.03 (>= 2.12.02) [PASS]
- C:\ASL\iasl: Version 20150619 (>= 20160422) [FAIL]
- git: Version 2.21.0. (>= 2.20.0) [PASS]
- vs: Version 2013 (>= 2015) [FAIL]
...Failed! Please check toolchain versions!
```

`BuildLoader.py` The dependent tools and their version information will be checked, so the whole picture is very important. The above tools need to be installed in the current Windows environment. Because this machine was used for UEFI development before, most of the tools are available, but the versions of openssl, iasl and Visual Studio do not meet the requirements. The latest version of openssl can be downloaded from https://sourceforge.net/projects/openssl-for-windows/ , and the latest version of iasl can be downloaded from https://www.acpica.org/downloads/binary-tools . Visual Studio needs to use version 2015 or above. Here, the 2019 version is selected. You can download it from the official website and install it online. It should be noted that you need to select the following content during installation:



正在修改 - Visual Studio Community 2019 - 16.11.3

- ☐ MSVC v142 - VS 2019 C++ x64/x86 启用 Spectre 缓解的库 (v14.20)
- ☐ MSVC v142 - VS 2019 C++ x64/x86 生成工具 (v14.20)
- ☐ MSVC v142 – VS 2019 C++ x64/x86 生成工具(v14.21)
- ☐ MSVC v142 - VS 2019 C++ x64/x86 生成工具(v14.22)
- ☐ MSVC v142 - VS 2019 C++ x64/x86 生成工具(v14.23)
- ☐ MSVC v142 - VS 2019 C++ x64/x86 生成工具(v14.24)
- ☐ MSVC v142 – VS 2019 C++ x64/x86 生成工具(v14.25)
- ☐ MSVC v142 – VS 2019 C++ x64/x86 生成工具(v14.26)
- ☐ MSVC v142 - VS 2019 C++ x64/x86 生成工具(v14.27)
- ☐ MSVC v142 - VS 2019 C++ x64/x86 生成工具(v14.28-16.8)
- ☐ MSVC v142 - VS 2019 C++ x64/x86 生成工具(v14.28-16.9)
- ☐ MSVC v142 – VS 2019 C++ x64/x86 生成工具(v14.29-16.10)
- ☑ MSVC v142 - VS 2019 C++ x64/x86 生成工具(最新)
- ☐ Python 2 32-bit (2.7.18) (out of support)
- ☐ Python 2 64-bit (2.7.18) (out of support)
- ☐ Python 3 32-bit (3.7.8)
- ☐ Python 3 64-bit (3.7.8)
- ☐ Windows 通用 CRT SDK
- ☐ 对 v141 生成工具(14.16)的 C++/CLI 支持

安装详细信息

- Visual Studio 核心编辑器
- 单个组件 *
  - ☑ MSVC v142 - VS 2019 C++ x64/x86 生成工具(...

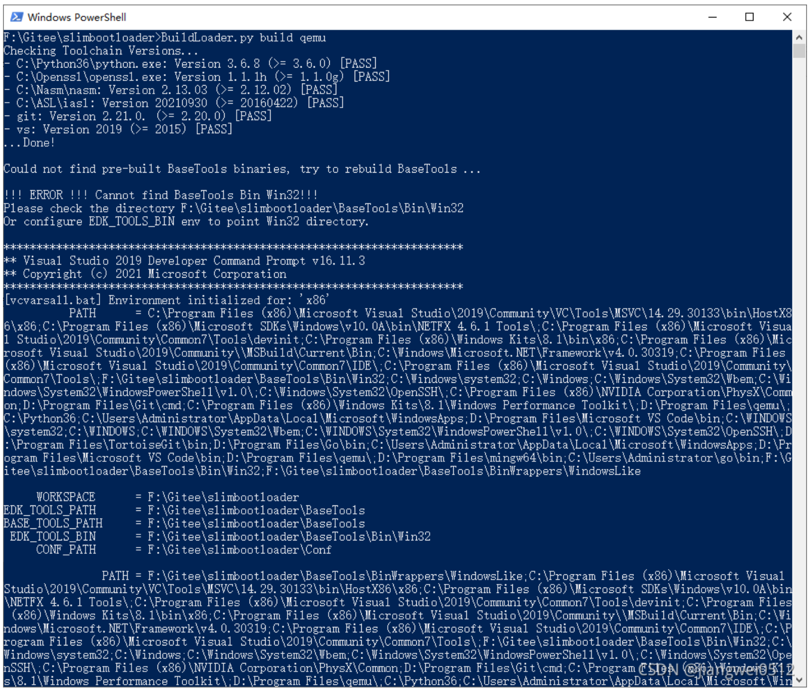
位置
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community

继续操作即表示你同意所选 Visual Studio 版本的许可证。我们还提供通过 Visual Studio 下载其他软件的功能。此软件单独进行许可，如第三方公告或其随附的许可证中所述。继续即表示你同意这些许可证。

要求的总空间 2.06 GB

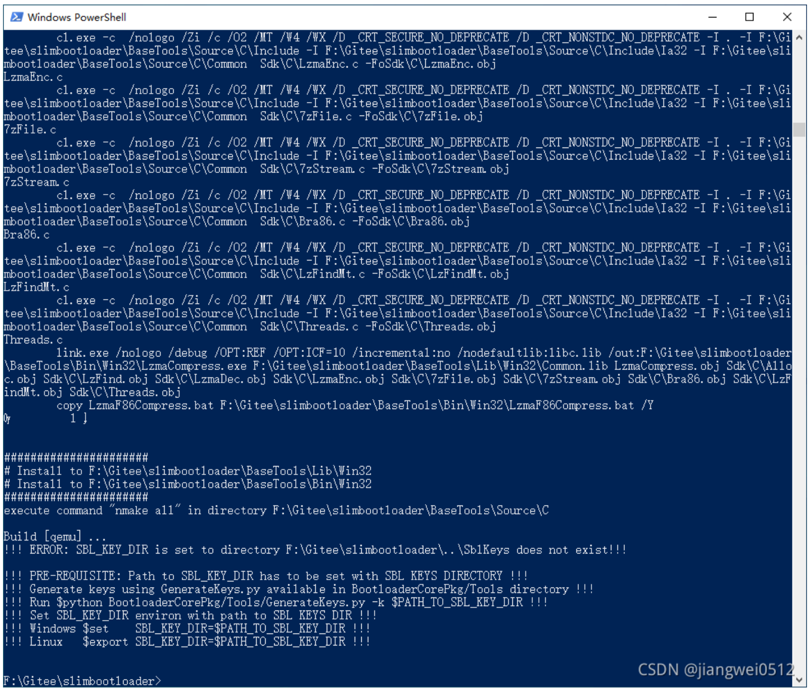
下载时安装

Here we can ensure that the tools required for SBL compilation (such as cl.exe, etc.) can be installed correctly. Execute the compilation command again, and the results are as follows:



```
F:\Gitee\slimbootloader>BuildLoader.py build qemu
Checking Toolchain Versions...
- C:\Python36\python.exe: Version 3.6.8 (>= 3.6.0) [PASS]
- C:\Openssl\openssl.exe: Version 1.1.1h (>= 1.1.0g) [PASS]
- C:\Nasm\nasm: Version 2.13.03 (>= 2.12.02) [PASS]
- C:\ASL\iasl: Version 20210930 (>= 20160422) [PASS]
- git: Version 2.21.0. (>= 2.20.0) [PASS]
- vs: Version 2019 (>= 2015) [PASS]
...Done!

Could not find pre-built BaseTools binaries, try to rebuild BaseTools ...

!!! ERROR !!! Cannot find BaseTools Bin Win32!!!
Please check the directory F:\Gitee\slimbootloader\BaseTools\Bin\Win32
Or configure EDK_TOOLS_BIN env to point Win32 directory.

**********************************************************************
** Visual Studio 2019 Developer Command Prompt v16.11.3
** Copyright (c) 2021 Microsoft Corporation
**********************************************************************
[vcvarsall.bat] Environment initialized for: 'x86'
          PATH      = C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\bin\HostX8
6\x86;C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin\NETFX 4.6.1 Tools\;C:\Program Files (x86)\Microsoft Visua
l Studio\2019\Community\Common7\Tools\devinit;C:\Program Files (x86)\Windows Kits\8.1\bin\x86;C:\Program Files (x86)\Mic
rosoft Visual Studio\2019\Community\\MSBuild\Current\Bin;C:\Windows\Microsoft.NET\Framework\v4.0.30319;C:\Program Files
(x86)\Microsoft Visual Studio\2019\Community\Common7\IDE\;C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\
Common7\Tools\;F:\Gitee\slimbootloader\BaseTools\Bin\Win32;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Wi
ndows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\OpenSSH\;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Comm
on;D:\Program Files\Git\cmd;C:\Program Files (x86)\Windows Kits\8.1\Windows Performance Toolkit\;D:\Program Files\qemu\;
C:\Python36;C:\Users\Administrator\AppData\Local\Microsoft\WindowsApps;D:\Program Files\Microsoft VS Code\bin;C:\WINDOWS
\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\System32\OpenSSH\;D
:\Program Files\TortoiseGit\bin;D:\Program Files\Go\bin;C:\Users\Administrator\AppData\Local\Microsoft\WindowsApps;D:\Pr
ogram Files\Microsoft VS Code\bin;D:\Program Files\qemu\;D:\Program Files\mingw64\bin;C:\Users\Administrator\go\bin;F:\G
itee\slimbootloader\BaseTools\Bin\Win32;F:\Gitee\slimbootloader\BaseTools\BinWrappers\WindowsLike

     WORKSPACE      = F:\Gitee\slimbootloader
EDK_TOOLS_PATH      = F:\Gitee\slimbootloader\BaseTools
BASE_TOOLS_PATH     = F:\Gitee\slimbootloader\BaseTools
 EDK_TOOLS_BIN      = F:\Gitee\slimbootloader\BaseTools\Bin\Win32
     CONF_PATH      = F:\Gitee\slimbootloader\Conf

               PATH = F:\Gitee\slimbootloader\BaseTools\BinWrappers\WindowsLike;C:\Program Files (x86)\Microsoft Visual
Studio\2019\Community\VC\Tools\MSVC\14.29.30133\bin\HostX86\x86;C:\Program Files (x86)\Microsoft SDKs\Windows\v10.0A\bin
\NETFX 4.6.1 Tools\;C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\Tools\devinit;C:\Program Files
 (x86)\Windows Kits\8.1\bin\x86;C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\\MSBuild\Current\Bin;C:\Wi
ndows\Microsoft.NET\Framework\v4.0.30319;C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\Common7\IDE\;C:\P
rogram Files (x86)\Microsoft Visual Studio\2019\Community\Common7\Tools\;F:\Gitee\slimbootloader\BaseTools\Bin\Win32;C:\
Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Windows\System32\Ope
nSSH\;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;D:\Program Files\Git\cmd;C:\Program Files (x86)\Windows Kit
s\8.1\Windows Performance Toolkit\;D:\Program Files\qemu\;C:\Python36;C:\Users\Administrator\AppData\Local\Microsoft\Win
```

You can see that the tool detection has passed, but it will still report an error in the end:

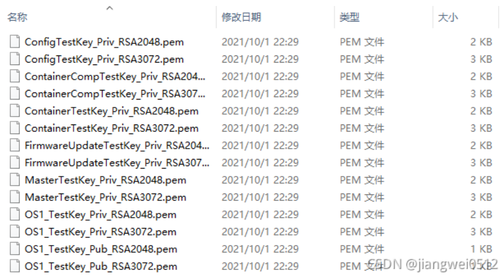


```
        cl.exe -c  /nologo /Zi /c /O2 /MT /W4 /WX /D _CRT_SECURE_NO_DEPRECATE /D _CRT_NONSTDC_NO_DEPRECATE -I . -I F:\Gi
tee\slimbootloader\BaseTools\Source\C\Include -I F:\Gitee\slimbootloader\BaseTools\Source\C\Include\Ia32 -I F:\Gitee\sli
mbootloader\BaseTools\Source\C\Common  Sdk\C\LzmaEnc.c -FoSdk\C\LzmaEnc.obj
LzmaEnc.c
        cl.exe -c  /nologo /Zi /c /O2 /MT /W4 /WX /D _CRT_SECURE_NO_DEPRECATE /D _CRT_NONSTDC_NO_DEPRECATE -I . -I F:\Gi
tee\slimbootloader\BaseTools\Source\C\Include -I F:\Gitee\slimbootloader\BaseTools\Source\C\Include\Ia32 -I F:\Gitee\sli
mbootloader\BaseTools\Source\C\Common  Sdk\C\7zFile.c -FoSdk\C\7zFile.obj
7zFile.c
        cl.exe -c  /nologo /Zi /c /O2 /MT /W4 /WX /D _CRT_SECURE_NO_DEPRECATE /D _CRT_NONSTDC_NO_DEPRECATE -I . -I F:\Gi
tee\slimbootloader\BaseTools\Source\C\Include -I F:\Gitee\slimbootloader\BaseTools\Source\C\Include\Ia32 -I F:\Gitee\sli
mbootloader\BaseTools\Source\C\Common  Sdk\C\7zStream.c -FoSdk\C\7zStream.obj
7zStream.c
        cl.exe -c  /nologo /Zi /c /O2 /MT /W4 /WX /D _CRT_SECURE_NO_DEPRECATE /D _CRT_NONSTDC_NO_DEPRECATE -I . -I F:\Gi
tee\slimbootloader\BaseTools\Source\C\Include -I F:\Gitee\slimbootloader\BaseTools\Source\C\Include\Ia32 -I F:\Gitee\sli
mbootloader\BaseTools\Source\C\Common  Sdk\C\Bra86.c -FoSdk\C\Bra86.obj
Bra86.c
        cl.exe -c  /nologo /Zi /c /O2 /MT /W4 /WX /D _CRT_SECURE_NO_DEPRECATE /D _CRT_NONSTDC_NO_DEPRECATE -I . -I F:\Gi
tee\slimbootloader\BaseTools\Source\C\Include -I F:\Gitee\slimbootloader\BaseTools\Source\C\Include\Ia32 -I F:\Gitee\sli
mbootloader\BaseTools\Source\C\Common  Sdk\C\LzFindMt.c -FoSdk\C\LzFindMt.obj
LzFindMt.c
        cl.exe -c  /nologo /Zi /c /O2 /MT /W4 /WX /D _CRT_SECURE_NO_DEPRECATE /D _CRT_NONSTDC_NO_DEPRECATE -I . -I F:\Gi
tee\slimbootloader\BaseTools\Source\C\Include -I F:\Gitee\slimbootloader\BaseTools\Source\C\Include\Ia32 -I F:\Gitee\sli
mbootloader\BaseTools\Source\C\Common  Sdk\C\Threads.c -FoSdk\C\Threads.obj
Threads.c
        link.exe /nologo /debug /OPT:REF /OPT:ICF=10 /incremental:no /nodefaultlib:libc.lib /out:F:\Gitee\slimbootloader
\BaseTools\Bin\Win32\LzmaCompress.exe F:\Gitee\slimbootloader\BaseTools\Lib\Win32\Common.lib LzmaCompress.obj Sdk\C\Allo
c.obj Sdk\C\LzFind.obj Sdk\C\LzmaDec.obj Sdk\C\LzmaEnc.obj Sdk\C\7zFile.obj Sdk\C\7zStream.obj Sdk\C\Bra86.obj Sdk\C\LzF
indMt.obj Sdk\C\Threads.obj
        copy LzmaF86Compress.bat F:\Gitee\slimbootloader\BaseTools\Bin\Win32\LzmaF86Compress.bat /Y
0         1 ]


######################
# Install to F:\Gitee\slimbootloader\BaseTools\Lib\Win32
# Install to F:\Gitee\slimbootloader\BaseTools\Bin\Win32
######################
execute command "nmake all" in directory F:\Gitee\slimbootloader\BaseTools\Source\C

Build [qemu] ...
!!! ERROR: SBL_KEY_DIR is set to directory F:\Gitee\slimbootloader\..\SblKeys does not exist!!!

!!! PRE-REQUISITE: Path to SBL_KEY_DIR has to be set with SBL KEYS DIRECTORY !!!
!!! Generate keys using GenerateKeys.py available in BootloaderCorePkg/Tools directory !!!
!!! Run $python BootloaderCorePkg/Tools/GenerateKeys.py -k $PATH_TO_SBL_KEY_DIR !!!
!!! Set SBL_KEY_DIR environ with path to SBL KEYS DIR !!!
!!! Windows $set    SBL_KEY_DIR=$PATH_TO_SBL_KEY_DIR !!!
!!! Linux   $export SBL_KEY_DIR=$PATH_TO_SBL_KEY_DIR !!!


F:\Gitee\slimbootloader>
```

The previous content is to build the compilation tool in BaseTools. You can see that it is completed. There is another step to generate the key before building the SBL binary. The command used is:

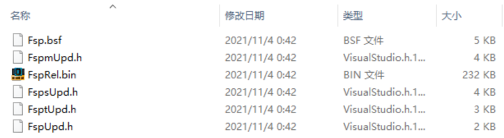
```bash
F:\Gitee\slimbootloader>BootloaderCorePkg\Tools\GenerateKeys.py -k ..\SblKeys
```

The main reason is that the directory where the command is executed is specifically stated here. The reason is that SBL will place the key at the same level of the directory where it is located by default. The corresponding directory name is SblKeys by default. After executing the command, the following key is generated:



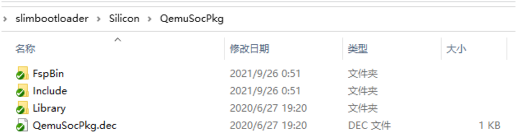
名称	修改日期	类型	大小
ConfigTestKey_Priv_RSA2048.pem	2021/10/1 22:29	PEM 文件	2 KB
ConfigTestKey_Priv_RSA3072.pem	2021/10/1 22:29	PEM 文件	3 KB
ContainerCompTestKey_Priv_RSA204...	2021/10/1 22:29	PEM 文件	2 KB
ContainerCompTestKey_Priv_RSA307...	2021/10/1 22:29	PEM 文件	3 KB
ContainerTestKey_Priv_RSA2048.pem	2021/10/1 22:29	PEM 文件	2 KB
ContainerTestKey_Priv_RSA3072.pem	2021/10/1 22:29	PEM 文件	3 KB
FirmwareUpdateTestKey_Priv_RSA204...	2021/10/1 22:29	PEM 文件	2 KB
FirmwareUpdateTestKey_Priv_RSA307...	2021/10/1 22:29	PEM 文件	3 KB
MasterTestKey_Priv_RSA2048.pem	2021/10/1 22:29	PEM 文件	2 KB
MasterTestKey_Priv_RSA3072.pem	2021/10/1 22:29	PEM 文件	3 KB
OS1_TestKey_Priv_RSA2048.pem	2021/10/1 22:29	PEM 文件	2 KB
OS1_TestKey_Priv_RSA3072.pem	2021/10/1 22:29	PEM 文件	3 KB
OS1_TestKey_Pub_RSA2048.pem	2021/10/1 22:29	PEM 文件	1 KB
OS1_TestKey_Pub_RSA3072.pem	2021/10/1 22:29	PEM 文件	1 KB

After the key is generated, you also need to generate FSP. Since the SBL binary for QEMU is built here, you need to generate FSP for QEMU. The details of FSP will not be explained here. For how to generate FSP, you can download https://gitee.com/jiangwei0512/edk2-beni , and then generate FSP through the BuildFsp.py script. The generated content is as follows:



名称	修改日期	类型	大小
Fsp.bsf	2021/11/4 0:42	BSF 文件	5 KB
FspmUpd.h	2021/11/4 0:42	VisualStudio.h.1...	4 KB
FspRel.bin	2021/11/4 0:42	BIN 文件	232 KB
FspsUpd.h	2021/11/4 0:42	VisualStudio.h.1...	4 KB
FsptUpd.h	2021/11/4 0:42	VisualStudio.h.1...	3 KB
FspUpd.h	2021/11/4 0:42	VisualStudio.h.1...	2 KB

After that, you need to put the generated content into the SBL directory. The specific directory is as follows:

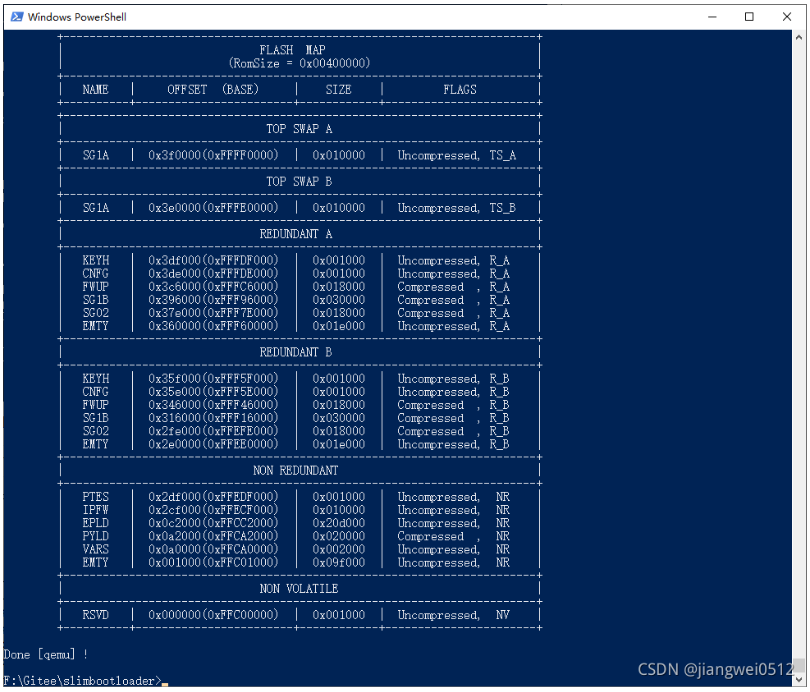


› slimbootloader › Silicon › QemuSocPkg

名称	修改日期	类型	大小
FspBin	2021/9/26 0:51	文件夹	
Include	2021/9/26 0:51	文件夹	
Library	2020/6/27 19:20	文件夹	
QemuSocPkg.dec	2020/6/27 19:20	DEC 文件	1 KB

Put the header file in the Include directory (if there is already a file with the same name, just overwrite it), and put Fsp.bsf and FspRel.bin in the FspBin directory.

This completes the setup of the FSP binary, and then executes the previous build command to successfully complete the build:



```
       +------------------------------------------------------------------+
       |                           FLASH  MAP                             |
       |                     (RomSize = 0x00400000)                       |
       +------------------------------------------------------------------+
       |   NAME   |       OFFSET  (BASE)       |    SIZE    |      FLAGS   |
       +----------+----------------------------+------------+--------------+
       +------------------------------------------------------------------+
       |                           TOP SWAP A                             |
       +------------------------------------------------------------------+
       |   SG1A   |   0x3f0000(0xFFFF0000)     |  0x010000  |  Uncompressed, TS_A |
       +------------------------------------------------------------------+
       |                           TOP SWAP B                             |
       +------------------------------------------------------------------+
       |   SG1A   |   0x3e0000(0xFFFE0000)     |  0x010000  |  Uncompressed, TS_B |
       +------------------------------------------------------------------+
       |                          REDUNDANT A                             |
       +------------------------------------------------------------------+
       |   KEYH   |   0x3df000(0xFFFDF000)     |  0x001000  |  Uncompressed, R_A |
       |   CNFG   |   0x3de000(0xFFFDE000)     |  0x001000  |  Uncompressed, R_A |
       |   FWUP   |   0x3c6000(0xFFFC6000)     |  0x018000  |  Compressed  , R_A |
       |   SG1B   |   0x396000(0xFFF96000)     |  0x030000  |  Compressed  , R_A |
       |   SG02   |   0x37e000(0xFFF7E000)     |  0x018000  |  Compressed  , R_A |
       |   EMTY   |   0x360000(0xFFF60000)     |  0x01e000  |  Uncompressed, R_A |
       +------------------------------------------------------------------+
       |                          REDUNDANT B                             |
       +------------------------------------------------------------------+
       |   KEYH   |   0x35f000(0xFFF5F000)     |  0x001000  |  Uncompressed, R_B |
       |   CNFG   |   0x35e000(0xFFF5E000)     |  0x001000  |  Uncompressed, R_B |
       |   FWUP   |   0x346000(0xFFF46000)     |  0x018000  |  Compressed  , R_B |
       |   SG1B   |   0x316000(0xFFF16000)     |  0x030000  |  Compressed  , R_B |
       |   SG02   |   0x2fe000(0xFFEFE000)     |  0x018000  |  Compressed  , R_B |
       |   EMTY   |   0x2e0000(0xFFEE0000)     |  0x01e000  |  Uncompressed, R_B |
       +------------------------------------------------------------------+
       |                         NON REDUNDANT                            |
       +------------------------------------------------------------------+
       |   PTES   |   0x2df000(0xFFEDF000)     |  0x001000  |  Uncompressed,  NR |
       |   IPFW   |   0x2cf000(0xFFECF000)     |  0x010000  |  Uncompressed,  NR |
       |   EPLD   |   0x0c2000(0xFFCC2000)     |  0x20d000  |  Uncompressed,  NR |
       |   PYLD   |   0x0a2000(0xFFCA2000)     |  0x020000  |  Compressed  ,  NR |
       |   VARS   |   0x0a0000(0xFFCA0000)     |  0x002000  |  Uncompressed,  NR |
       |   EMTY   |   0x001000(0xFFC01000)     |  0x09f000  |  Uncompressed,  NR |
       +------------------------------------------------------------------+
       |                          NON VOLATILE                            |
       +------------------------------------------------------------------+
       |   RSVD   |   0x000000(0xFFC00000)     |  0x001000  |  Uncompressed,  NV |
       +----------+----------------------------+------------+--------------+

Done [qemu] !

F:\Gitee\slimbootloader>
```

The generated binary is `Build\BootloaderCorePkg\DEBUG_VS2019\FV` located at `SlimBootloader.bin` .

use

Because the SBL binary is built for QEMU, if you want to use it, you can directly start it through the QEMU command. Of course, QEMU needs to be installed separately, which can be downloaded from https://www.qemu.org/download/ . The command to use is as follows:

bash　　AI generated projects　　登录复制

```bash
qemu-system-x86_64 -machine q35 -nographic -serial mon:stdio -pflash Build\BootloaderCorePkg\DEBUG_VS2019\FV\SlimBootloader.bin
```

Here are the execution results:

bash　　AI generated projects　　登录复制

```bash
F:\Gitee\slimbootloader>qemu-system-x86_64 -machine q35 -nographic -serial mon:stdio -pflash Build\BootloaderCorePkg\DEBUG_VS2019\FV\SlimBootloader.bin
WARNING: Image format was not specified for 'Build\BootloaderCorePkg\DEBUG_VS2019\FV\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.

============= Intel Slim Bootloader STAGE1A =============
SBID: SB_QEMU
ISVN: 001
IVER: 001.000.001.001.03646
SVER: 7F461C59E0D513D9
FDBG: BLD(D IA32) FSP(R)
FSPV: ID($QEMFSP$) REV(00001000)
Loader global data @ 0x00001C38
Run  STAGE1A @ 0x00070000
Load STAGE1B @ 0x00040000
HASH verification for usage (0x00000001) with Hash Alg (0x2): Success

============= Intel Slim Bootloader STAGE1B =============
Host Bridge Device ID:0x29C0
Board ID:0x1 - Loading QEMU!
QEMU Flash: Attempting flash detection at FFC00000
QemuFlashDetected => FD behaves as FLASH
QemuFlashDetected => Yes
SpiInstance = 0000E470
Variable region: 0xFFCA0000:0x2000
  SPI WRITE: FFCA0010  00000004
  SPI WRITE: FFCA0011  00000001
  SPI WRITE: FFCA0014  00000008
  SPI WRITE: FFCA001C  00000004
  SPI WRITE: FFCA0011  00000001
Loading Component KEYH:_HS_
Registering container KEYH
HASH verification for usage (0x00000100) with Hash Alg (0x2): Success
SignType (0x2) SignSize (0x180)  SignHashAlg (0x2)
RSA verification for usage (0x00000100): Success
HASH verification for usage (0x00000000) with Hash Alg (0x2): Success
Append public key hash into store: Success
Load EXT CFG Data @ 0x0000EB54:0x0158 ... Success
HASH verification for usage (0x00000200) with Hash Alg (0x2): Success
SignType (0x2) SignSize (0x180)  SignHashAlg (0x2)
RSA verification for usage (0x00000200): Success
BOOT: BP0
MODE: 0
BoardID: 0x01
PlatformName: QEMU_01
Memory Init
Found Config TAG: 0x180 @ 0x0000ECCC
  MemCfg.Test1=11223344, MemCfg.Test2=11223346
Found Config TAG: 0x200 @ 0x0000ECDC
```

```
  SilCfg.Test1=11223347, SilCfg.Test2=04030201
Call FspMemoryInit ... Success
Loader global data @ 0x06EBFD70
Load page table from memory @ 0x06B74000
Remapping Stage to 0x06B77000
    FSP Resource HOB Range        Type       Owner
================================= ==== ====================================
0000000000000000-00000000000A0000  00  00000000-0000-0000-0000-000000000000
00000000000A0000-0000000000100000  05  00000000-0000-0000-0000-000000000000
0000000000100000-0000000006F00000  00  00000000-0000-0000-0000-000000000000
0000000006F00000-0000000007000000  05  69A79759-1373-4367-A6C4-C7F59EFD986E
0000000007000000-0000000008000000  05  D038747C-D00C-4980-B319-490199A47D55

Switch to memory stack @ 0x06EFFF00
Stage1 stack: 0x2000 (0xF28 used)
Stage1 heap: 0xE000 (0x1F14 used)
Call FspTempRamExit ... Success
Memory FSP @ 0x06F00000
Memory TOP @ 0x06B74000
Loading Component FLMP:SG02
HASH verification for usage (0x00000002) with Hash Alg (0x2): Success
Loaded STAGE2 @ 0x06E55000

============= Intel Slim Bootloader STAGE2 =============
Unmapping Stage
Board GPIO Init
Get base platform GPIO table from board ID 0
Programming 7 GPIO entries
GPIO GPP_A00 DATA: 0x00000000 0x00000010
GPIO GPP_A02 DATA: 0x80000002 0x00000012
GPIO GPP_A03 DATA: 0xC0000003 0x00000013
GPIO GPP_A04 DATA: 0x01000004 0x00000014
GPIO GPP_A05 DATA: 0x41000005 0x00000015
GPIO GPP_A06 DATA: 0x81000006 0x00000016
GPIO GPP_A07 DATA: 0xC1000007 0x00000017
Test variable services
  SPI WRITE: FFCA0020  00000004
  SPI WRITE: FFCA0021  00000001
  SPI WRITE: FFCA0024  00000008
  SPI WRITE: FFCA002C  00000004
  SPI WRITE: FFCA0011  00000001
  SPI WRITE: FFCA0021  00000001
  SPI WRITE: FFCA0011  00000001
Loading Component IPFW:TST3
Registering container IPFW
HASH verification for usage (0x00001000) with Hash Alg (0x2): Success
SignType (0x2) SignSize (0x180)  SignHashAlg (0x2)
RSA verification for usage (0x00001000): Success
HASH verification for usage (0x00000000) with Hash Alg (0x2): Success
SignType (0x2) SignSize (0x180)  SignHashAlg (0x2)
RSA verification for usage (0x00000000): Success
Load IP firmware @ 0:0x0 - Bad Buffer Size
Silicon Init
Select VBT ImageId 0x00000001
Call FspSiliconInit ...
Success
Graphics Info: 800 x 600 x 32 @ 0x80000000
MEM: 0000000000000000 00000000000A0000 00 1
MEM: 00000000000A0000 0000000000060000 00 2
MEM: 0000000000100000 0000000006A00000 00 1
MEM: 0000000006B00000 0000000000004000 01 2
MEM: 0000000006B04000 0000000000068000 00 3
MEM: 0000000006B6C000 0000000000008000 00 4
MEM: 0000000006B74000 000000000038C000 00 2
MEM: 0000000006F00000 0000000000100000 00 2
MEM: 0000000007000000 0000000001000000 00 2
MEM: 00000000FFC00000 0000000000400000 00 2
MP Init (Wakeup)
MP Init (Run)
Detected 1 CPU threads
 CPU  0 APIC ID: 0
SMM rebase done on 1 CPUs
PCI Enum
Call FspNotifyPhase(20) ... Success
ACPI Init
Publish ACPI table: FACP
Publish ACPI table: HPET
Publish ACPI table: APIC
Publish ACPI table: MCFG
Publish ACPI table: FPDT
Publish ACPI table: BGRT
Publish ACPI table: TEST
ACPI Ret: Success
Enable SMRR
Loading Payload ID PYLD
Loading Component FLMP:PYLD
HASH verification for usage (0x00000004) with Hash Alg (0x2): Success
Load Payload ID 0x00000000 @ 0x06CB7000
PE32 Format Payload
HOB @ 0x06EC0000
Created 4 OS boot options (Current: 0)
Stage2 stack: 0x40000 (stack used 0x9CC, HOB used 0xFB0, 0x3E684 free)
Stage2 heap: 0x34C000 (0x209000 used, 0x143000 free)
Payload entry: 0x06CB7260
Jump to payload


Payload startup
ACPI PmTimer Base: 0x408
PCI Express  Base: 0xE0000000


====================Os Loader====================


Press any key within 1 second(s) to enter the command shell
Boot options (in HEX):

Idx|ImgType|DevType|DevNum|Flags|HwPart|FsType|SwPart|File/Lbaoffset
  0|      0|    SD |    0 |   0 |    0 | AUTO |    0 | iasimage.bin
  1|      0|  SATA |    0 |   0 |   FF | AUTO |    0 | iasimage.bin
  2|      0|  NVME |    0 |   0 |    0 | AUTO |    0 | iasimage.bin
  3|      0|   USB |    0 |   0 |    0 | AUTO |    0 | iasimage.bin


======== Try Booting with Boot Option 0 ========
BootMediumPciBase(0x300)
Getting boot image from SD
MMC global data init
Use SDMA instead of ADMA2
MMC Phase 1 init
SdMmcHcReset: reset done with Time out
SdMmcHcReset Fail Status = 0x80000012
Failed to init media - Time out
Failed to Initialize Boot Device - Type 1, Instance 0
Payload normal heap: 0x2000000 (0xB000 used)
Payload reserved heap: 0x4000 (0x0 used)
Payload stack: 0x10000 (0x4D4 used)


======== Try Booting with Boot Option 1 ========
BootMediumPciBase(0x1F02)
Getting boot image from SATA
Init AHCI controller E00FA000
AHCI port [2] has a [cdrom]
Try to find boot partition
AhciDeviceRead Status = Device Error
Partition type: UNKNOWN  (1 logical partitions)
Find partition success
Init File system
AhciDeviceRead Status = Device Error
AhciDeviceRead Status = Device Error
No partitions found, Status = Device Error
Failed to Initialize Boot File System - SwPart 0
Payload normal heap: 0x2000000 (0x10F000 used)
Payload reserved heap: 0x4000 (0x0 used)
Payload stack: 0x10000 (0x664 used)

Deinit AHCI controller 0
Not a AHCI controller or Disabled

======== Try Booting with Boot Option 2 ========
BootMediumPciBase(0x300)
Getting boot image from NVME
NvmExpressDriverBindingStart: start
NvmeControllerInit: the controller doesn't support NVMe command set
NvmExpressDriverBindingStart: end with Unsupported
Failed to init media - Unsupported
Failed to Initialize Boot Device - Type 6, Instance 0
Payload normal heap: 0x2000000 (0x10F000 used)
Payload reserved heap: 0x4000 (0x0 used)
Payload stack: 0x10000 (0x664 used)


======== Try Booting with Boot Option 3 ========
BootMediumPciBase(0x400)
Getting boot image from USB
Failed to initialize USB bus !
Failed to init media - Unsupported
Failed to Initialize Boot Device - Type 5, Instance 0
Payload normal heap: 0x2000000 (0x10F000 used)
Payload reserved heap: 0x4000 (0x0 used)
Payload stack: 0x10000 (0x664 used)


Shell>
```

收起 ^

Finally entered the UEFI Shell.