[UEFI Practice] Build Script BuildLoader.py in SlimBootloader

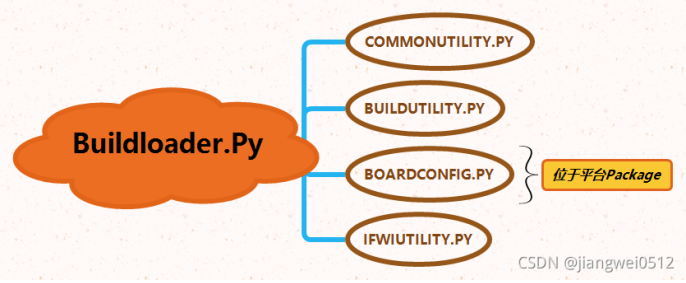
原创 jiangwei0512 Posted on 2021-10-24 22:30:57

Category Column: UEFI Development Basics Article Tags: sbl

摘要 This article introduces the key steps in the SlimBootloader build process, including the Python script BuildLoader.py, how it handles the build of BaseTools, configures environment variables, and creates the Conf directory. It also shows in detail how to handle BoardConfig files, set environment variables, and build component lists.

Overview

The SlimBootloader build depends on the Python script BuildLaoder.py, which in turn depends on additional scripts located in BootloaderCorePkg\Tools, as shown below:



Code Description

The source code is listed directly here, and the description is added:

Python AI generated projects 登录复制 run

```python
#!/usr/bin/env python
## @ BuildLoader.py
# Build bootloader main script
#
# Copyright (c) 2016 - 2021, Intel Corporation. All rights reserved.<BR>
#  SPDX-License-Identifier: BSD-2-Clause-Patent

##
# Import Modules
#
import os
import sys

# BootloaderCorePkg\Tools的绝对路径，里面包含SBL需要使用的Python脚本
tool_dir = os.path.join(os.path.dirname (os.path.realpath(__file__)), 'BootloaderCorePkg', 'Tools')
# 为了不生成.pyc文件，跟功能无关
sys.dont_write_bytecode = True
# 添加到环境变量，这样SBL需要使用的Python脚本就可以直接访问了BuildUtility.py
sys.path.append (tool_dir)

import re
import errno
import shutil
import argparse
import subprocess
import multiprocessing
from   ctypes import *
# 这里的BuildUtility就来自BootloaderCorePkg\Tools目录下的Python文件BuildUtility.py
from   BuildUtility import *

import glob


# 检查是否有BaseTools，如果没有就构建
def rebuild_basetools ():
    exe_list = 'GenFfs  GenFv  GenFw  GenSec  Lz4Compress  LzmaCompress'.split()
    ret = 0
    sblsource = os.environ['SBL_SOURCE'] # 在main()中设置，SBL根目录

    if os.name == 'posix': # Linux
        if not check_files_exist (exe_list, os.path.join(sblsource, 'BaseTools', 'Source', 'C', 'bin')):
            ret = run_process (['make', '-C', 'BaseTools'])

    elif os.name == 'nt': # Windows
        # 如果没有构建工具（就是exe_list定义的工具），就创建
        if not check_files_exist (exe_list, os.path.join(sblsource, 'BaseTools', 'Bin', 'Win32'), '.exe'):
            print ("Could not find pre-built BaseTools binaries, try to rebuild BaseTools ...")
            ret = run_process (['BaseTools\\toolsetup.bat', 'forcerebuild'])

    if ret:
        print ("Build BaseTools failed, please check required build environment and utilities !")
        sys.exit(1)


# 创建Conf目录及其下文件
def create_conf (workspace, sbl_source):
    # create conf and build folder if not exist
    workspace = os.environ['WORKSPACE'] # 也在main()下设置，值就是跟'SBL_SOURCE'一样，都是SBL根目录
    if not os.path.exists(os.path.join(workspace, 'Conf')):
        os.makedirs(os.path.join(workspace, 'Conf')) # 创建Conf目录
    for name in ['target', 'tools_def', 'build_rule']: # 拷贝基本配置文件
        txt_file = os.path.join(workspace, 'Conf/%s.txt' % name)
        if not os.path.exists(txt_file):
            shutil.copy (
                os.path.join(sbl_source, 'BaseTools/Conf/%s.template' % name),
                os.path.join(workspace, 'Conf/%s.txt' % name))


# 设置环境变量
```

```python
#   PATH
#   PYTHONPATH
#   EDK_TOOLS_PATH
#   BASE_TOOLS_PATH
#   CONF_PATH
#   SBL_KEY_DIR
def prep_env (toolchain_preferred = ''):
    sblsource = os.environ['SBL_SOURCE']
    os.chdir(sblsource)

    # Verify toolchains first
    # 检测各类工具的版本是否满足要求
    # 大致检测的内容:
    # 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:\Wasm\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]
    verify_toolchains(toolchain_preferred)

    # Update Environment vars
    if os.name == 'nt':
        os.environ['PATH'] = os.environ['PATH'] + ';' + os.path.join(sblsource, 'BaseTools', 'Bin', 'Win32')
        os.environ['PATH'] = os.environ['PATH'] + ';' + os.path.join(sblsource, 'BaseTools', 'BinWrappers', 'WindowsLike')
        os.environ['PYTHONPATH'] = os.path.join(sblsource, 'BaseTools', 'Source', 'Python')
    else:
        os.environ['PATH'] = os.environ['PATH'] + ':' + os.path.join(sblsource, 'BaseTools', 'BinWrappers', 'PosixLike')

    os.environ['EDK_TOOLS_PATH'] = os.path.join(sblsource, 'BaseTools')
    os.environ['BASE_TOOLS_PATH'] = os.path.join(sblsource, 'BaseTools')
    os.environ['CONF_PATH'] = os.path.join(os.environ['WORKSPACE'], 'Conf')

    if 'SBL_KEY_DIR' not in os.environ:
        os.environ['SBL_KEY_DIR'] = os.path.join(sblsource, '..', 'SblKeys')

    create_conf (os.environ['WORKSPACE'], sblsource) # 创建Conf目录

    # Check if BaseTools has been compiled
    rebuild_basetools () # 生成构建工具


# 返回所有的BoardConfig*.py文件(比如BoardConfig.py和BoardConfigOverride.py), 并放到board_cfgs中
# 注意返回的是绝对路径
# 下面是一种结果:
# [
# 'F:\\Gitee\\slimbootloader\\Platform\\ApollolakeBoardPkg\\BoardConfig.py',
# 'F:\\Gitee\\slimbootloader\\Platform\\CoffeelakeBoardPkg\\BoardConfig.py',
# 'F:\\Gitee\\slimbootloader\\Platform\\CometlakeBoardPkg\\BoardConfig.py',
# 'F:\\Gitee\\slimbootloader\\Platform\\CometlakevBoardPkg\\BoardConfig.py',
# 'F:\\Gitee\\slimbootloader\\Platform\\ElkhartlakeBoardPkg\\BoardConfig.py',
# 'F:\\Gitee\\slimbootloader\\Platform\\QemuBoardPkg\\BoardConfig.py',
# 'F:\\Gitee\\slimbootloader\\Platform\\QemuBoardPkg\\BoardConfigOverride.py',
# 'F:\\Gitee\\slimbootloader\\Platform\\TigerlakeBoardPkg\\BoardConfig.py'
# ]
def get_board_config_file (check_dir, board_cfgs):
    # 指定platform_dir为根目录下的Platform目录, 正常情况下, SBL根目录下就有一个Platform目录
    platform_dir = os.path.join (check_dir, 'Platform')
    if not os.path.isdir (platform_dir):
        if os.path.basename(check_dir) == 'Platform':
            platform_dir = check_dir
        else:
            return
    # 指定Platform目录下支持的所有平台
    board_pkgs = os.listdir(platform_dir)
    for pkg in board_pkgs:
        # Allow files starting with 'BoardConfig' only
        for cfgfile in glob.glob(os.path.join(platform_dir, pkg, 'BoardConfig*.py')):
            # board_cfgs指定所有平台下的BoardConfig.py文件
            board_cfgs.append(cfgfile)


# 单板基本配置项, 会写到Platform.dsc, 并最终在dsc中使用
class BaseBoard(object):
    def __init__(self, *args, **kwargs):

        # NOTE: Variables starting with '_' will not be exported to Platform.dsc


        self.LOGO_FILE              = 'Platform/CommonBoardPkg/Logo/Logo.bmp'

        self._RSA_SIGN_TYPE          = 'RSA2048'
        self._SIGN_HASH              = 'SHA2_256'
        self.SIGN_HASH_TYPE          = HASH_TYPE_VALUE[self._SIGN_HASH]
        self._SIGNING_SCHEME         = 'RSA_PSS'

        # Default key dir is set by SBL_KEY_DIR. _KEY_DIR is set to NULL.
        self._KEY_DIR = ''
        self._MASTER_PRIVATE_KEY    = 'KEY_ID_MASTER' + '_' + self._RSA_SIGN_TYPE
        self._CFGDATA_PRIVATE_KEY   = 'KEY_ID_CFGDATA' + '_' + self._RSA_SIGN_TYPE
        self._CONTAINER_PRIVATE_KEY = 'KEY_ID_CONTAINER' + '_' + self._RSA_SIGN_TYPE

        self.KEY_GEN                 = 0

        self.VERINFO_IMAGE_ID       = 'SB_???? '
        self.VERINFO_PROJ_ID        = 1
        self.VERINFO_CORE_MAJOR_VER = 1
        self.VERINFO_CORE_MINOR_VER = 0

        self.VERINFO_PROJ_MAJOR_VER = 0
        self.VERINFO_PROJ_MINOR_VER = 1
        self.VERINFO_SVN            = 1
        self.VERINFO_BUILD_DATE     = '01/01/2018'

        self.LOWEST_SUPPORTED_FW_VER = 1

        self.FLASH_BLOCK_SIZE       = 0x1000
        self.FLASH_LAYOUT_START     = 0x100000000
        self.FLASH_BASE             = 0
        self.FLASH_SIZE             = 0

        self.PCI_EXPRESS_BASE       = 0xE0000000
        self.ACPI_PM_TIMER_BASE     = 0x0408
        self.USB_KB_POLLING_TIMEOUT = 1

```

```
        self.VERIFIED_BOOT_STAGE_1B   = 0x0
        self.BOOT_MEDIA_SUPPORT_MASK  = 0xFFFFFFFF
        self.FILE_SYSTEM_SUPPORT_MASK = 0x00000003
        self.DEBUG_OUTPUT_DEVICE_MASK = 0x00000003
        self.DEBUG_PORT_NUMBER        = 0x00000002
        self.CONSOLE_IN_DEVICE_MASK   = 0x00000001
        self.CONSOLE_OUT_DEVICE_MASK  = 0x00000001

        self.HAVE_VBT_BIN          = 0
        self.HAVE_FIT_TABLE        = 0
        self.HAVE_VERIFIED_BOOT    = 0
        self.HAVE_MEASURED_BOOT    = 0
        self.HAVE_FSP_BIN          = 1
        self.HAVE_ACPI_TABLE       = 1
        self.HAVE_PSD_TABLE        = 0
        self.HAVE_SEED_LIST        = 0

        self.FIT_ENTRY_MAX_NUM     = 10

        self.ENABLE_PCI_ENUM       = 1
        self.ENABLE_SMP_INIT       = 1
        self.ENABLE_FSP_LOAD_IMAGE = 0
        self.ENABLE_SPLASH         = 0
        self.ENABLE_FRAMEBUFFER_INIT = 0
        self.ENABLE_PRE_OS_CHECKER = 0
        self.ENABLE_CRYPTO_SHA_OPT  = IPP_CRYPTO_OPTIMIZATION_MASK['SHA256_V8']
        self.ENABLE_FWU            = 0
        self.ENABLE_SOURCE_DEBUG   = 0
        self.ENABLE_GRUB_CONFIG    = 0
        self.ENABLE_SMBIOS         = 0
        self.ENABLE_LINUX_PAYLOAD  = 0
        self.ENABLE_CONTAINER_BOOT = 1
        self.ENABLE_CSME_UPDATE    = 0
        self.ENABLE_EMMC_HS400     = 1
        self.ENABLE_DMA_PROTECTION = 0
        self.ENABLE_MULTI_USB_BOOT_DEV = 0
        self.ENABLE_SBL_SETUP      = 0
        self.ENABLE_PAYLOD_MODULE  = 0
        self.ENABLE_FAST_BOOT      = 0
        self.ENABLE_LEGACY_EF_SEG  = 1
        # 0: Disable  1: Enable  2: Auto (disable for UEFI payload, enable for others)
        self.ENABLE_SMM_REBASE     = 0

        self.SUPPORT_ARI           = 0
        self.SUPPORT_SR_IOV        = 0
        self.SUPPORT_X2APIC        = 0

        self.BUILD_CSME_UPDATE_DRIVER    = 0

        self.CPU_MAX_LOGICAL_PROCESSOR_NUMBER = 16

        self.ACM_SIZE              = 0
        self.DIAGNOSTICACM_SIZE    = 0
        self.UCODE_SIZE            = 0
        self.CFGDATA_SIZE          = 0
        self.MRCDATA_SIZE          = 0
        self.VARIABLE_SIZE         = 0
        self.UEFI_VARIABLE_SIZE    = 0
        self.FWUPDATE_SIZE         = 0

        self.SPI_IAS1_SIZE         = 0
        self.SPI_IAS2_SIZE         = 0

        self.KM_SIZE               = 0x1000 # valid only if ACM_SIZE > 0
        self.BPM_SIZE              = 0x1000 # valid only if ACM_SIZE > 0
        self.CFG_DATABASE_SIZE     = 0

        self.FSP_M_STACK_TOP       = 0
        self.STAGE1A_XIP           = 1
        self.STAGE1B_XIP           = 1
        self.STAGE1_STACK_BASE_OFFSET = 0
        self.STAGE2_XIP            = 0
        self.STAGE2_LOAD_HIGH      = 1
        self.PAYLOAD_LOAD_HIGH     = 1
        self.PAYLOAD_EXE_BASE      = 0x00800000

        #     0: Direct access from flash
        # other: Load image into memory address
        self.PAYLOAD_LOAD_BASE     = 0
        self.FWUPDATE_LOAD_BASE    = 0

        # OS Loader FD/FV sizes
        self.OS_LOADER_FD_SIZE     = 0x0004E000

        self.OS_LOADER_FD_NUMBLK   = self.OS_LOADER_FD_SIZE // self.FLASH_BLOCK_SIZE

        self.PLD_HEAP_SIZE         = 0x02000000
        self.PLD_STACK_SIZE        = 0x00010000
        self.PLD_RSVD_MEM_SIZE     = 0x00004000

        # These memory sizes need to be page aligned
        self.LOADER_RSVD_MEM_SIZE         = 0x0038C000
        self.LOADER_ACPI_NVS_MEM_SIZE     = 0x00008000
        self.LOADER_ACPI_RECLAIM_MEM_SIZE = 0x00068000

        self.CFGDATA_REGION_TYPE   = FLASH_REGION_TYPE.BIOS

        self.RELEASE_MODE          = 0
        self.NO_OPT_MODE           = 0
        self.FSPDEBUG_MODE         = 0
        self.MIN_FSP_REVISION      = 0
        self.FSP_IMAGE_ID          = ''

        self.TOP_SWAP_SIZE         = 0
        self.REDUNDANT_SIZE        = 0

        self._PAYLOAD_NAME         = ''
        self._FSP_PATH_NAME        = ''
        self._EXTRA_INC_PATH       = []

        self._PLATFORM_ID          = None
        self._MULTI_VBT_FILE       = {}
        self._CFGDATA_INT_FILE     = []
        self._CFGDATA_EXT_FILE     = []

        self.IPP_HASH_LIB_SUPPORTED_MASK   = IPP_CRYPTO_ALG_MASK[self._SIGN_HASH]
```

```
        self.HASH_STORE_SIZE       = 0x400  #Hash store size to be allocated in bootloader

        self.PCI_MEM64_BASE        = 0
        self.BUILD_ARCH            = 'IA32'
        self.KEYH_SVN              = 0
        self.CFGDATA_SVN           = 0

        for key, value in list(kwargs.items()):
            setattr(self, '%s' % key, value)


# 将构建操作作为一个类来处理
class Build(object):
    # 构造函数，初始化使用Build(board)，board的类型是BaseBoard及其子类
    def __init__(self, board):
        self._toolchain                     = os.environ['TOOL_CHAIN'] # 在BuildUtility.py中设置
        self._workspace                     = os.environ['WORKSPACE']
        self._board                         = board
        self._image                         = "SlimBootloader.bin"
        self._arch                          = board.BUILD_ARCH
        self._target                        = 'RELEASE' if board.RELEASE_MODE  else 'NOOPT' if board.NO_OPT_MODE else 'DEBUG'
        self._fsp_basename                  = 'FspDbg'  if board.FSPDEBUG_MODE else 'FspRel'
        self._fv_dir                        = os.path.join(self._workspace, 'Build', 'BootloaderCorePkg', '%s_%s' % (self._target, self._toolchain), 'FV')
        self._key_dir                       = self._board._KEY_DIR
        self._img_list                      = board.GetImageLayout()
        # pld就是Payload，如果没有指定payload参数，则默认的值是OsLoader.efi
        self._pld_list                      = get_payload_list (board._PAYLOAD_NAME.split(';'))
        self._comp_list                     = []
        self._region_list                   = []

        # enforce feature configs rules
        if self._board.ENABLE_SBL_SETUP:
            self._board.ENABLE_PAYLOD_MODULE = 1
        # Python可以增加新的成员在类实例中
        if not hasattr(self._board, 'MICROCODE_INF_FILE'):
            self._board.MICROCODE_INF_FILE  = 'Silicon/%s/Microcode/Microcode.inf' % self._board.SILICON_PKG_NAME
        if not hasattr(self._board, 'ACPI_TABLE_INF_FILE'):
            self._board.ACPI_TABLE_INF_FILE = 'Platform/%s/AcpiTables/AcpiTables.inf' % self._board.BOARD_PKG_NAME

        for stage in ['1A', '1B', '2']:
            soc_inf = 'SOC_INIT_STAGE%s_LIB_INF_FILE' % stage
            if not hasattr(self._board, soc_inf):
                soc_init_lib = 'Silicon/%s/Library/Stage%sSocInitLib/Stage%sSocInitLib.inf' % (self._board.SILICON_PKG_NAME, stage, stage)
                setattr(self._board, 'SOC_INIT_STAGE%s_LIB_INF_FILE' % stage, soc_init_lib)
            brd_inf = 'BRD_INIT_STAGE%s_LIB_INF_FILE' % stage
            if not hasattr(self._board, brd_inf):
                brd_init_lib = 'Platform/%s/Library/Stage%sBoardInitLib/Stage%sBoardInitLib.inf' % (self._board.BOARD_PKG_NAME, stage, stage)
                setattr(self._board, 'BRD_INIT_STAGE%s_LIB_INF_FILE' % stage, brd_init_lib)

        if not hasattr(self._board, 'SOC_FWU_LIB_INF_FILE'):
            self._board.SOC_FWU_LIB_INF_FILE = 'Silicon/%s/Library/FirmwareUpdateLib/FirmwareUpdateLib.inf' % self._board.SILICON_PKG_NAME

    # BoardConfig.py中如果有定义PlatformBuildHook函数，这会执行
    def board_build_hook (self, phase):
        if getattr(self._board, "PlatformBuildHook", None):
            self._board.PlatformBuildHook (self, phase)

    def update_fit_table (self):

        if not self._board.HAVE_FIT_TABLE:
            return

        print('Updating FIT')
        # self._fv_dir就是Build\BootloaderCorePkg\DEBUG_VS2019\FV（根据编译器可能会有不同）
        # self._image就是SlimBootloader.bin
        img_file = os.path.join (self._fv_dir, self._image)
        fi = open(img_file,'rb')
        rom = bytearray(fi.read()) # 得到字节数组
        fi.close()

        # Find FIT pointer @ 0xFFFFFFC0
        # FIT_ENTRY.FIT_OFFSET来自BootloaderCorePkg\Tools\IfwiUtility.py，值是-40
        fit_address = c_uint32.from_buffer(rom, len(rom) + FIT_ENTRY.FIT_OFFSET)
        print('  FIT Address: 0x%08X' % fit_address.value)

        # 如果支持ACM，则地址必须是64字节对齐的
        if self._board.ACM_SIZE > 0:
            # Check FIT address alignment for 64 bytes if ACM is used
            # because BIOS IBB segments base/size require 64 bytes alignment.
            if fit_address.value & ~0x3F != fit_address.value:
                raise Exception ('  FIT address (0x%08X) is not 64-byte aligned' % fit_address.value)

        # Check FIT address range
        # 4G以下的BIOS基址。FIT地址从基址到4G之间
        base = 0x100000000 - len(rom);
        if (fit_address.value < base) or (fit_address.value > (base + len(rom))):
            raise Exception('  FIT address (0x%08X) out of range' % fit_address.value)

        # Check FIT signature
        fit_offset = fit_address.value - base
        fit_header = FIT_ENTRY.from_buffer(rom, fit_offset)
        if fit_header.address != bytes_to_value (bytearray(FIT_ENTRY.FIT_SIGNATURE)):
            raise Exception('  FIT signature not found')

        num_fit_entries = 0
        if self._board.UCODE_SIZE > 0:
            ucode_base = self._board.UCODE_BASE
            ucode_offset = ucode_base - base;
            if (ucode_offset < 0):
                raise Exception ('  UCODE %x\n  UCODE address (0x%08X) out of range' % (base, ucode_base))

            # Collect all CPU uCode images
            u_code_images = []
            while ucode_offset < len(rom):
                ucode_hdr = UCODE_HEADER.from_buffer(rom, ucode_offset)
                if ucode_hdr.header_version == 1:
                    if ucode_hdr.total_size:
                        ucode_size = ucode_hdr.total_size
                    else:
                        ucode_size = 0x0800
                    u_code_images.append((ucode_offset, ucode_size))
                    ucode_offset += ucode_size
                    num_fit_entries += 1
                else:
                    break
```

```
            # Patch FIT with addresses of uCode images
            for i in range(0, num_fit_entries):
                fit_entry = FIT_ENTRY.from_buffer(rom, fit_offset + (i+1)*16)
                # uCode Update
                if len(u_code_images) > 0:
                    offset, size = u_code_images.pop(0)
                    fit_entry.set_values(base + offset, 0, 0x100, 0x1, 0)
                    print ('  Patching entry %d with 0x%08X - uCode' % (i, fit_entry.address))
                else:
                    print ('  Nullifying unused uCode patch entry %d' % i)
                    fit_entry.type      = 0x7f

            if len(u_code_images) > 0:
                raise Exception('  Insufficient uCode entries in FIT. Need %d more.' % len(u_code_images))

            # ACM
        if self._board.ACM_SIZE > 0:
            fit_entry = FIT_ENTRY.from_buffer(rom, fit_offset + (num_fit_entries+1)*16)
            fit_entry.set_values(self._board.ACM_BASE, 0, 0x100, 0x2, 0)
            print ('  Patching entry %d with 0x%08X:0x%08X - ACM' % (num_fit_entries, fit_entry.address, fit_entry.size))
            num_fit_entries     += 1

            # Diagnostic ACM Fit entry should be in sequential order with/without BTG enabled
            # Save the next FIT entry for Diagnostic ACM here and set it later below
            if self._board.DIAGNOSTICACM_SIZE > 0:
                diagnosticacm_index      = num_fit_entries
                num_fit_entries          += 1

            # BIOS Module (IBB segment 0): from FIT table end to 4GB
            # Record it now and update later since the FIT size is unknown yet
            patch_entry = num_fit_entries
            num_fit_entries     += 1

            # BIOS Module (IBB segment 1): from Stage1A base to FIT table start
            addr = self._board.STAGE1A_BASE
            module_size =  (fit_address.value - addr) >> 4
            fit_entry = FIT_ENTRY.from_buffer(rom, fit_offset + (num_fit_entries+1)*16)
            fit_entry.set_values(addr, module_size, 0x100, 0x7, 0)
            print ('  Patching entry %d with 0x%08X:0x%08X - BIOS Module(Stage1A base to FIT table start)' % (num_fit_entries, fit_entry.address, fit_entry.size))
            num_fit_entries     += 1

            # BIOS Module (IBB segment 2): full Stage1B
            addr = self._board.STAGE1B_BASE
            module_size = self._board.STAGE1B_SIZE >> 4
            fit_entry = FIT_ENTRY.from_buffer(rom, fit_offset + (num_fit_entries+1)*16)
            fit_entry.set_values(addr, module_size, 0x100, 0x7, 0)
            print ('  Patching entry %d with 0x%08X:0x%08X - BIOS Module(Stage1B)' % (num_fit_entries, fit_entry.address, fit_entry.size))
            num_fit_entries     += 1

            # KM
            addr = self._board.ACM_BASE + self._board.ACM_SIZE - (self._board.KM_SIZE + self._board.BPM_SIZE)
            fit_entry = FIT_ENTRY.from_buffer(rom, fit_offset + (num_fit_entries+1)*16)
            fit_entry.set_values(addr, self._board.KM_SIZE, 0x100, 0xb, 0)
            print ('  Patching entry %d with 0x%08X:0x%08X - KM' % (num_fit_entries, fit_entry.address, fit_entry.size))
            num_fit_entries     += 1

            # BPM
            addr = self._board.ACM_BASE + self._board.ACM_SIZE - self._board.BPM_SIZE
            fit_entry = FIT_ENTRY.from_buffer(rom, fit_offset + (num_fit_entries+1)*16)
            fit_entry.set_values(addr, self._board.BPM_SIZE, 0x100, 0xc, 0)
            print ('  Patching entry %d with 0x%08X:0x%08X - BPM' % (num_fit_entries, fit_entry.address, fit_entry.size))
            num_fit_entries     += 1

            # Patch the entry 'FIT table end to 4GB' since FIT table size is known now
            # The size of the FIT table end address needs to be adjusted to align with 64
            # bytes so that IBB segment start address is 64 byte aligned as per required.
            addr = fit_address.value + (num_fit_entries + 1) * 16
            addr = (addr + 0x3F) & 0xFFFFFFC0
            module_size = (0x100000000 - addr) >> 4
            fit_entry = FIT_ENTRY.from_buffer(rom, fit_offset + (patch_entry+1)*16)
            fit_entry.set_values(addr, module_size, 0x100, 0x7, 0)
            print ('  Patching entry %d with 0x%08X:0x%08X - BIOS Module(FIT table end to 4GB)' % (patch_entry, fit_entry.address, fit_entry.size))

        else :
            if self._board.DIAGNOSTICACM_SIZE > 0:
                diagnosticacm_index       = num_fit_entries
                num_fit_entries           += 1

            addr = fit_address.value + (num_fit_entries + 1) * 16

        # Add Diagnostic ACM with the reserved fit entry saved
        if self._board.DIAGNOSTICACM_SIZE > 0:
            fit_entry = FIT_ENTRY.from_buffer(rom, fit_offset + (diagnosticacm_index+1)*16)
            fit_entry.set_values(self._board.DIAGNOSTICACM_BASE, self._board.DIAGNOSTICACM_SIZE, 0x100, 0x3, 0)
            print('  Patching entry %d with 0x%08X:0x%08X - Diagnostic ACM' % (diagnosticacm_index, fit_entry.address, fit_entry.size))

        # Check FIT length
        spaceleft = addr - (fit_address.value + fit_header.size)
        if spaceleft > 0:
                raise Exception('  Insufficient FIT entries in FIT table, need %d more entries !' %  ((spaceleft + 15) // 16))

        print ('  FIT %d entries added' % num_fit_entries)

        # Update FIT checksum
        print('  Updating Checksum')
        fit_header.size      = num_fit_entries + 1
        fit_header.type      = 0x80 # Valid checksum
        fit_header.version   = 0x0100
        fit_header.checksum  = 0
        fit_sum = sum(rom[fit_offset:fit_offset+fit_header.size*16])
        fit_header.checksum = (0 - fit_sum) & 0xff
        fit_data = rom[fit_offset:fit_offset+fit_header.size*16]

        fo  = open(img_file,'r+b')

        fo.seek(fit_offset)
        fo.write(fit_data)

        if self._board.REDUNDANT_SIZE != 0:
            # Update FIT table in STAGE1A_B
            print('Updating FIT in STAGE1A_B')
            fit_offset -= self._board.TOP_SWAP_SIZE
            rom[fit_offset:fit_offset+fit_header.size*16] = fit_data

            # Update components base in Fit table.
```

```
            fit_data = rom[fit_offset:fit_offset+fit_header.size*16]
            for i in range(0, num_fit_entries):
                fit_entry = FIT_ENTRY.from_buffer(fit_data, (i+1)*16)
                if (0x100000000 - fit_entry.address) > self._board.TOP_SWAP_SIZE * 2:
                    fit_entry.address  -= self._board.REDUNDANT_SIZE
                    print('  Patching entry %d from 0x%08X with 0x%08X size:0x%08X ' %
                        (i, fit_entry.address + self._board.REDUNDANT_SIZE, fit_entry.address, fit_entry.size))
            fit_header = FIT_ENTRY.from_buffer(fit_data)
            fit_header.checksum  = 0
            fit_sum = sum(fit_data)
            fit_header.checksum = (0 - fit_sum) & 0xff
            fo.seek(fit_offset)
            fo.write(fit_data)

        fo.close()


    def update_hash_table (self, img_file):

        if not self._board.HAVE_VERIFIED_BOOT:
            return

        print('Updating HashStore %s' % os.path.basename (img_file))

        fi = open(img_file,'rb')
        stage1_bins = bytearray(fi.read())
        fi.close()

        hs_offset = stage1_bins.find (HashStoreTable.HASH_STORE_SIGNATURE)
        if hs_offset < 0:
            raise Exceptoin ("HashStoreTable not found in '%s'!" % os.path.basename(img_file))

        comp_name, part_name = get_redundant_info (img_file)
        if part_name:
            part_name = '_' + part_name

        hash_file_list = [
           ('STAGE1B%s.hash' % part_name, HASH_USAGE['STAGE_1B']),
           ('STAGE2.hash',                HASH_USAGE['STAGE_2']),
           ('PAYLOAD.hash',               HASH_USAGE['PAYLOAD'])
        ]
        if self._board.ENABLE_FWU:
            hash_file_list.append (('FWUPDATE.hash', HASH_USAGE['PAYLOAD_FWU']))

        hash_file_list.append (('MSTKEY.hash', HASH_USAGE['PUBKEY_MASTER']))

        if len(hash_file_list) > HashStoreTable.HASH_STORE_MAX_IDX_NUM:
            raise Exception ('Insufficant hash entries !')

        hash_idx   = 0
        hash_store = HashStoreTable.from_buffer(stage1_bins, hs_offset)
        hash_len   = HASH_DIGEST_SIZE[HASH_VAL_STRING[self._board.SIGN_HASH_TYPE]]
        hash_store_data_buf = bytearray()
        hash_store.UsedLength = sizeof(HashStoreTable())
        for hash_file, usage in hash_file_list:
            # If the hash verification is not required for certain stage, skip it
            if hash_file == 'PLDDYN':
                hash_data = bytearray(b'\x00' * hash_len)
            else:
                src_path = os.path.join(self._fv_dir, hash_file)
                if not os.path.exists(src_path):
                    raise Exception ("Hash data file '%s' not found !" % hash_file )
                fh = open(src_path,'rb')
                hash_data = bytearray(fh.read())
                fh.close()
                if hash_len != len (hash_data):
                    raise Exception ("Hash data file '%s' length is incorrect !" % hash_file )

            # update hash data
            hashstoredata = HashStoreData()
            hashstoredata.Usage   = usage
            hashstoredata.HashAlg = self._board.SIGN_HASH_TYPE
            hashstoredata.DigestLen = hash_len

            hash_store.UsedLength += hash_len + sizeof(HashStoreData())

            #Append hash store data entries
            hash_store_data_buf = hash_store_data_buf + bytearray(hashstoredata) + hash_data
            print(' Update HashStore entry %d with file %s' % (hash_idx, hash_file))
            hash_idx += 1

        #Update Hash store Table
        fo = open(img_file,'r+b')
        fo.seek(hs_offset)
        fo.write(hash_store)
        #Update Hash store data
        fo.seek(hs_offset + sizeof(hash_store))
        fo.write(hash_store_data_buf)
        fo.close()


    def update_component_list (self):

        def process_image_list (idx, offset):

            region_name, part_name = get_redundant_info (img_list[idx][0])
            redundant   = True if part_name == 'B' else False

            flags  = 0
            if redundant:
                flags |= FLASH_MAP.FLASH_MAP_DESC_FLAGS['BACKUP']
            if region_name in ['TOP_SWAP', 'REDUNDANT', 'NON_REDUNDANT', 'NON_VOLATILE']:
                flags |= FLASH_MAP.FLASH_MAP_DESC_FLAGS[region_name]

            oldidx         = len (comp_list)
            parent_size    = getattr(self._board, '%s_SIZE' % region_name, 0)
            remaining_size = parent_size
            for comp in img_list[idx][1]:
                if comp[3] & STITCH_OPS.MODE_FILE_IGNOR:
                    continue
                compress = FLASH_MAP.FLASH_MAP_DESC_FLAGS['COMPRESSED'] if comp[1] else 0
                if comp[0] in region_name_list:
                    idx = region_name_list.index (comp[0])
                    region_size = process_image_list (idx, offset)
                    region_list.append ({'name':comp[0], 'offset':offset, 'size':region_size})
                    offset += region_size
```

```
                else:
                    comp_list.append ({'name':comp[0], 'size':comp[2], 'flag':flags | compress})
                remaining_size -= comp[2]

            if remaining_size > 0:
                comp_node = find_component_in_image_list (img_list[idx][0], img_list)
                pos  = STITCH_OPS.MODE_POS_HEAD if comp_node is None else comp_node[4]
                comp = {'name':'EMPTY.bin', 'size':remaining_size, 'flag':flags}
                if pos == STITCH_OPS.MODE_POS_HEAD:
                    comp_list.insert (oldidx, comp)
                else:
                    comp_list.append (comp)
            elif remaining_size < 0:
                if parent_size == 0:
                    parent_size = -remaining_size
                else:
                    raise Exception ('Insufficant space, please adjust %s_SIZE (0x%X more is requried) !' % (region_name, -remaining_size))

            return parent_size

        # Create compoent list and update base and offset
        img_list        = self._img_list
        region_name_list = [img[0] for img in img_list]
        comp_list        = []
        region_list      = []

        try:
            master_name = self._image
            master_idx  = region_name_list.index(master_name)
            process_image_list (master_idx, 0)
            image_size = sum (comp['size'] for comp in comp_list)
            image_base = self._board.FLASH_LAYOUT_START - image_size
            image_offs = 0
            for comp in comp_list:
                comp['bname']  = get_redundant_info (comp['name'])[0]
                comp['offset'] = image_offs
                comp['base']   = image_base + image_offs
                image_offs    += comp['size']

            for rgn in region_list:
                rgn['base'] = image_base + rgn['offset']

        except ValueError:
            print("Warning: No '%s' component in image list !" % master_name)

        #print_component_list (comp_list)

        self._comp_list   = comp_list
        self._region_list = region_list


    def patch_stages (self):

        print('Patching STAGE1A')
        extra_cmd = [
            "STAGE1A:STAGE1A",
            "0xFFFFFFFC,            _BASE_STAGE1A_,                    @Patch BFV",
            "_OFFS_STAGE1A_,        Stage1A:__ModuleEntryPoint,        @Patch Stage1A Entry",
            "_OFFS_STAGE1A_+4,      Stage1A:BASE,                      @Patch Module Base",
            "<Stage1A:__gPcd_BinaryPatch_PcdVerInfoBase>,  {3473A022-C3C2-4964-B309-22B3DFB0B6CA:0x1C}, @Patch VerInfo",
            "<Stage1A:__gPcd_BinaryPatch_PcdFileDataBase>, {EFAC3859-B680-4232-A159-F886F2AE0B83:0x1C}, @Patch PcdBase"
        ]

        if self._arch == 'X64':
            # Find signature at top 4KB
            vtf_patch_data_base = get_vtf_patch_base (os.path.join(self._fv_dir, 'STAGE1A.fd'))
            page_table_len = 0x8000
            if self._board.STAGE1_DATA_SIZE < page_table_len:
                raise Exception ("STAGE1_DATA_SIZE is too small to build x64 page table, "
                                 "it requires at least 0x%X !" % page_table_len)
            page_tbl_off = self._board.STAGE1_STACK_BASE_OFFSET + self._board.STAGE1_STACK_SIZE + \
                           self._board.STAGE1_DATA_SIZE - page_table_len
            extra_cmd.extend ([
                "0x%08X, 0x%08X,                                        @Page Table Offset" % (vtf_patch_data_base + 0x00, page_tbl_off),
                "0x%08X, _BASE_STAGE1A_ - _OFFS_STAGE1A_,                    @FSP-T Base" % (vtf_patch_data_base + 0x04),
                "0x%08X, Stage1A:_TempRamInitParams,                         @FSP-T UPD"  % (vtf_patch_data_base + 0x0C),
            ])

        extra_cmd.append (
            "0xFFFFFFF8, {3CEA8EF3-95FC-476F-ABA5-7EC5DFA1D77B:0x1C}, @Patch FlashMap",
        )

        if self._board.HAVE_FIT_TABLE:
            if self._board.ACM_SIZE > 0:
                extra_cmd.append (
                    "0xFFFFFFC0,        ({CD17FF5E-7731-4D16-8441-FC7A113C392F:0x1C} + 0x3F) & ~0x3F,  @FIT table"
                )
            else:
                extra_cmd.append (
                    "0xFFFFFFC0,        {CD17FF5E-7731-4D16-8441-FC7A113C392F:0x1C},                   @FIT table"
                )
            extra_cmd.extend ([
                    "<[0xFFFFFFC0]>+0,  0x5449465F,                                                    @FIT Signature Low" ,
                    "<[0xFFFFFFC0]>+4,  0x2020205F,                                                    @FIT Signature High",
                    "<[0xFFFFFFC0]>+8,  [CD17FF5E-7731-4D16-8441-FC7A113C392F:0x18] & 0xFFFFFF ,       @FIT FFS section length",
                    "<[0xFFFFFFC0]>+8,  {CD17FF5E-7731-4D16-8441-FC7A113C392F:0x1C} + [[0xFFFFFFC0] + 8] - [0xFFFFFFC0],  @FIT table max length",
                ])
        if self._board.HAVE_VERIFIED_BOOT:
            extra_cmd.append (
                "<Stage1A:__gPcd_BinaryPatch_PcdHashStoreBase>, {18EDB1DF-1DBE-4EC5-8E26-C44808B546E1:0x1C}, @Patch HashStore",
            )
        patch_fv(self._fv_dir, *extra_cmd)

        print('Patching STAGE1B')
        patch_fv(
                self._fv_dir,
                "STAGE1B:STAGE1B",
                "_OFFS_STAGE1B_,        Stage1B:__ModuleEntryPoint,        @Patch Stage1B Entry",
                "_OFFS_STAGE1B_+4,      Stage1B:BASE,                      @Patch Stage1B Base",
                "<Stage1B:__gPcd_BinaryPatch_PcdCfgDataIntBase>, {016E6CD0-4834-4C7E-BCFE-41DFB88A6A6D:0x1C}, @Patch Internal CfgDataBase"
                )

        print('Patching STAGE2')
        extra_cmd = []
        if self._board.HAVE_VBT_BIN:
            extra_cmd.append (
```

```
                "<Stage2:__gPcd_BinaryPatch_PcdGraphicsVbtAddress>, {E08CA6D5-8D02-43AE-ABB1-952CC787C933:0x1C}, @Patch VBT"
            )
        if self._board.HAVE_ACPI_TABLE:
            extra_cmd.append (
                "<Stage2:__gPcd_BinaryPatch_PcdAcpiTablesAddress>, {7E374E25-8E01-4FEE-87F2-390C23C606CD:0x1C}, @Patch ACPI",
            )
        if self._board.ENABLE_SPLASH:
            extra_cmd.append (
                "<Stage2:__gPcd_BinaryPatch_PcdSplashLogoAddress>, {5E2D3BE9-AD72-4D1D-AAD5-6B08AF921590:0x1C}, @Patch Logo",
            )
        patch_fv(
            self._fv_dir,
            "STAGE2:STAGE2",
            "_OFFS_STAGE2_,     Stage2:__ModuleEntryPoint, @Patch Stage2 Entry",
            "_OFFS_STAGE2_+4,   Stage2:BASE,               @Patch Stage2 Base",
            *extra_cmd
            )


    # 自动生成dsc文件，生成的主要是Platform.dsc文件
    def create_dsc_inc_file (self, file):
        lines = []

        lines.append('%s\n' % AUTO_GEN_DSC_HDR) # 一些注释的头部，用来说明不要自定生成的文件
        lines.append('# Platform specific macro definitions\n')
        lines.append('[Defines]\n')
        # vars()函数返回BaseBoard对象的属性及其值，是一个字典
        for attr in sorted(vars(self._board)):
            # _开头的属性不管
            if attr.startswith('_'):
                continue
            value = getattr(self._board, attr)
            if type(value) is not str:
                if value == 0 or value == 1:
                    value = '0x%x' % value
                else:
                    value = '0x%08X' % value
            lines.append('  DEFINE %-24s = %s\n'   % (attr, value))

        if getattr(self._board, "GetPlatformDsc", None):
            dsc_dict = self._board.GetPlatformDsc()

            # add extra include searching path
            if len(self._board._EXTRA_INC_PATH) > 0:
                inc_dir = []
                for each in self._board._EXTRA_INC_PATH:
                    inc_dir.append(('-I$(WORKSPACE)/%s' % each).replace('/', os.sep))
                dsc_dict['BuildOptions.Common.EDKII'] = ['*_*_*_CC_FLAGS = ' + ' '.join(inc_dir)]

            for sect in dsc_dict:
                lines.append('\n# Platform specific sections\n')
                lines.append('[%s]\n' % sect)
                for line in dsc_dict[sect]:
                    lines.append('  %s\n' % line)
                lines.append('\n')

        elif getattr(self._board, "GetDscLibrarys", None):
            # Deprecated, please use GetPlatformDsc instead
            libsdict = self._board.GetDscLibrarys()
            for arch in libsdict:
                lines.append('\n# Platform specific libraries\n')
                lines.append('[LibraryClasses.%s]\n' % arch)
                for lib in libsdict[arch]:
                    lines.append('  %s\n' % lib)
                lines.append('\n')

        update = True
        text = ''.join(lines)
        # 如果原本就存在，可以不更新
        if os.path.exists(file):
                old_text = get_file_data (file, 'r')
                if text == old_text:
                        update = False

        if update:
                open (file, 'w').write(text)


    def create_platform_vars (self):
        for comp in self._comp_list: # FLASH_MAP是定义在IfwiUtility.py中的类
            if comp['flag'] & FLASH_MAP.FLASH_MAP_DESC_FLAGS['BACKUP'] or comp['bname'] == 'EMPTY':
                continue
            setattr(self._board, '%s_BASE' % comp['bname'], comp['base'])

        image_base = self._board.FLASH_LAYOUT_START
        for idx, comp_name in enumerate(['STAGE1A', 'STAGE1B', 'STAGE2']):
            if not hasattr(self._board, '%s_BASE' % comp_name):
                image_base -= getattr(self._board, '%s_SIZE' % comp_name)
                if idx > 0:
                    image_base &= ~0xFFFFF
                setattr(self._board, '%s_BASE' % comp_name, image_base)

            if getattr(self._board, '%s_XIP' % comp_name) or comp_name == 'STAGE1A':
                setattr(self._board, '%s_FD_SIZE' % comp_name, getattr(self._board, '%s_SIZE' % comp_name))
                setattr(self._board, '%s_FD_BASE' % comp_name, getattr(self._board, '%s_BASE' % comp_name))

            if getattr(self._board, '%s_XIP' % comp_name):
                setattr(self._board, '%s_LOAD_BASE' % comp_name, getattr(self._board, '%s_BASE' % comp_name))
            else:
                var_name = '%s_LOAD_BASE' % comp_name
                if not hasattr(self._board, var_name):
                    setattr(self._board, var_name, getattr(self._board, '%s_BASE' % comp_name))
                for var in ['%s_FD_SIZE', '%s_FD_BASE', '%s_LOAD_BASE']:
                    var_name = var % comp_name
                    if not hasattr(self._board, var_name):
                        raise Exception ('%s needs to be defined' % var_name)

            fd_size = getattr(self._board, '%s_FD_SIZE' % comp_name)
            setattr(self._board, '%s_FD_NUMBLK' % comp_name, fd_size // self._board.FLASH_BLOCK_SIZE)

        pld_list = ['PAYLOAD']
        if self._board.ENABLE_FWU:
            pld_list.append ('FWUPDATE')
        for pld in pld_list:
            if not hasattr(self._board, '%s_LOAD_BASE' % pld):
                if not hasattr(self._board, '%s_BASE' % pld):
```

```
                raise Exception ('%s_BASE or %s_LOAD_BASE needs to be defined !' % (pld, pld))
            setattr(self._board, '%s_LOAD_BASE' % pld, getattr(self._board, '%s_BASE' % pld))

    setattr(self._board, 'FSP_T_OFFSET'       , 0)
    setattr(self._board, 'STAGE1A_FV_OFFSET'  , getattr(self._board, 'FSP_T_OFFSET')      + getattr(self._board, 'FSP_T_SIZE'))
    setattr(self._board, 'STAGE1A_FV_SIZE'    , getattr(self._board, 'STAGE1A_FD_SIZE')   - getattr(self._board, 'FSP_T_SIZE'))
    setattr(self._board, 'STAGE1B_FV_OFFSET'  , 0)
    setattr(self._board, 'STAGE1B_FV_SIZE'    , getattr(self._board, 'STAGE1B_FD_SIZE')   - getattr(self._board, 'FSP_M_SIZE'))
    setattr(self._board, 'STAGE2_FV_OFFSET'   , 0)
    setattr(self._board, 'STAGE2_FV_SIZE'     , getattr(self._board, 'STAGE2_FD_SIZE')    - getattr(self._board, 'FSP_S_SIZE'))
    setattr(self._board, 'FSP_S_OFFSET'       , getattr(self._board, 'STAGE2_FV_OFFSET')  + getattr(self._board, 'STAGE2_FV_SIZE'))
    setattr(self._board, 'FSP_M_OFFSET'       , getattr(self._board, 'STAGE1B_FV_OFFSET') + getattr(self._board, 'STAGE1B_FV_SIZE'))
    setattr(self._board, 'FSP_T_BASE'         , getattr(self._board, 'STAGE1A_FD_BASE')   + getattr(self._board, 'FSP_T_OFFSET'))
    setattr(self._board, 'FSP_M_BASE'         , getattr(self._board, 'STAGE1B_FD_BASE')   + getattr(self._board, 'FSP_M_OFFSET'))
    setattr(self._board, 'FSP_S_BASE'         , getattr(self._board, 'STAGE2_FD_BASE')    + getattr(self._board, 'FSP_S_OFFSET'))

    for stage_c, fsp_c in [('1A', 'T'), ('1B', 'M'), ('2', 'S')]:
      fv_size = getattr(self._board, 'STAGE%s_FV_SIZE' % stage_c)
      if fv_size < 0:
        raise Exception ('STAGE%s_FD_SIZE is too small, please adjust it to be at least 0x%x in BoardConfig.py !' %
                         (stage_c, getattr(self._board, 'FSP_%s_SIZE' % fsp_c)))

    if getattr(self._board, 'FLASH_SIZE') == 0:
        if not hasattr (self._board, 'SLIMBOOTLOADER_SIZE'):
            raise Exception ('FLASH_SIZE needs to be defined !')
        else:
            setattr(self._board, 'FLASH_SIZE' , getattr(self._board, 'SLIMBOOTLOADER_SIZE'))
    setattr(self._board, 'FLASH_BASE' , 0x100000000 - getattr(self._board, 'FLASH_SIZE'))

    if getattr(self._board, 'ACM_SIZE') > 0:
        acm_base = getattr(self._board, 'ACM_BASE')
        if acm_base & 0x7FFF:
            raise Exception ('ACM base[FSP-T+CAR:0x%x] must be 32KB aligned!' % acm_base)

    if getattr(self._board, 'DIAGNOSTICACM_SIZE') > 0:
        diagnosticacm_base = getattr(self._board, 'DIAGNOSTICACM_BASE')
        if diagnosticacm_base & 0x0FFF:
            raise Exception ('Diagnostic ACM base[FSP-T+CAR:0x%x] must be 4KB aligned!' % diagnosticacm_base)

    # Generate Pci Enum Policy Info
    pci_enum_policy_list = [
        'DOWNGRADE_IO32',
        'DOWNGRADE_MEM64',
        'DOWNGRADE_PMEM64',
        'DOWNGRADE_BUS0',
        'FLAG_ALLOC_PMEM_FIRST',
        'BUS_SCAN_TYPE',
        'BUS_SCAN_ITEMS'
    ]
    pci_enum_policy_dict = {}
    for policy_list in pci_enum_policy_list:
        policy_name = '_PCI_ENUM_%s' % policy_list
        policy_value = None
        if not hasattr(self._board, policy_name):
            if policy_list == 'BUS_SCAN_ITEMS':
                policy_value = '0'
            elif 'DOWNGRADE' in policy_list:
                policy_value = 1
            else:
                policy_value = 0
        else:
            policy_value = getattr(self._board, policy_name)
        pci_enum_policy_dict[policy_list] = policy_value
    self._board.PCI_ENUM_POLICY_INFO = gen_pci_enum_policy_info(pci_enum_policy_dict)

def create_redundant_components (self):
    if self._board.REDUNDANT_SIZE == 0:
        return

    print("Generating redundant components")

    shutil.copy(
        os.path.join(self._fv_dir, 'STAGE1A.fd'),
        os.path.join(self._fv_dir, 'STAGE1A_A.fd'))

    shutil.copy(
        os.path.join(self._fv_dir, 'STAGE1A.fd'),
        os.path.join(self._fv_dir, 'STAGE1A_B.fd'))

    # Patch flashmap to indicate boot parititon
    fo = open(os.path.join(self._fv_dir, 'STAGE1A_B.fd'), 'r+b')
    bins = bytearray(fo.read())
    fmapoff = bytes_to_value(bins[-8:-4]) - bytes_to_value(bins[-4:]) + self._board.STAGE1A_FV_OFFSET
    fmaphdr = FLASH_MAP.from_buffer (bins, fmapoff)
    if fmaphdr.sig != FLASH_MAP.FLASH_MAP_SIGNATURE:
        raise Exception ('Failed to locate flash map in STAGE1A_B.fd !')
    fmaphdr.attributes |=  fmaphdr.FLASH_MAP_ATTRIBUTES['BACKUP_REGION']
    fo.seek(fmapoff)
    fo.write(fmaphdr)

    # Patch microcode base in FSP-T UPD
    if self._board.HAVE_FSP_BIN and self._board.TOP_SWAP_SIZE > 0:
        fspt_bin = os.path.join(self._fv_dir, 'FSP_T.bin')
        upd_sig  = get_fsp_upd_signature (fspt_bin)
        upd_off  = bins.find (upd_sig, self._board.STAGE1A_FV_OFFSET)
        if upd_off < 0:
            raise Exception ('Could not find FSP-T UPD signatures in STAGE1A_B.fd !')
        if bins.find (upd_sig, upd_off + 1) > 0:
            raise Exception ('Found multiple FSP-T UPD signatures in STAGE1A_B.fd !')
        ucode_upd_off = upd_off + 0x20
        ucode_base = bytes_to_value (bins[ucode_upd_off + 0 : ucode_upd_off + 4])
        if ucode_base == 1:
            # APL/QEMU FSP-T UPD has revision 1 format
            ucode_upd_off = upd_off + 0x24
            ucode_base = bytes_to_value (bins[ucode_upd_off + 0 : ucode_upd_off + 4])
        ucode_size = bytes_to_value (bins[ucode_upd_off + 4 : ucode_upd_off + 8])
        if ucode_size > 0 and ucode_base > 0:
            if ucode_base != self._board.UCODE_BASE:
                raise Exception ('Incorrect microcode region base in FSP-T UPD parameter !')
            if ucode_base < 0x100000000 - self._board.TOP_SWAP_SIZE * 2:
                # Microcode is located outside of top swap region, patch it
                ucode_base -= self._board.REDUNDANT_SIZE
                print ('Patching UCODE base in FSP-T UPD parameter to 0x%08X for STAGE1A_B.fd' % ucode_base)
                fo.seek (ucode_upd_off)
                fo.write (value_to_bytes (ucode_base, 4))
```

```
        fo.close()

        # Stage 1B_B will be created during rebasing
        shutil.copy(
            os.path.join(self._fv_dir, 'STAGE1B.fd'),
            os.path.join(self._fv_dir, 'STAGE1B_A.fd'))

        stage1b_path   = os.path.join(self._fv_dir, 'STAGE1B.fd')
        stage1b_b_path = os.path.join(self._fv_dir, 'STAGE1B_B.fd')

        if self._board.STAGE1B_XIP:
            # Rebase stage1b.fd
            print("Rebasing STAGE1B_B")
            rebase_stage (stage1b_path, stage1b_b_path, -self._board.REDUNDANT_SIZE)

            # rebase FSPM in Stage1B and update stage1B hash in key store
            if self._board.HAVE_FSP_BIN:
                fsp_path = os.path.join(self._fv_dir, 'Fsp.bin')
                rebase_fsp (fsp_path, self._fv_dir, self._board.FSP_T_BASE, self._board.FSP_M_BASE - self._board.REDUNDANT_SIZE, self._board.FSP_S_BASE)
                split_fsp (fsp_path, self._fv_dir)

                # write rebased fspm to second firmware
                di = open(os.path.join(self._fv_dir, 'FSP_M.bin'), 'rb').read()
                fo = open(stage1b_b_path,'r+b')
                fo.seek(self._board.FSP_M_OFFSET)
                fo.write(di)
                fo.close()
        else:
            shutil.copy(stage1b_path, stage1b_b_path)


    def create_bootloader_image (self, layout_name):

        layout_file = open(os.path.join(self._fv_dir, layout_name), 'w')
        layout_file.write("BOARD_INFO = ['%s']\n" % self._board.BOARD_NAME)

        rgn_name_list = [rgn['name'] for rgn in self._region_list]

        for idx, (comp_name, file_list)  in enumerate(self._img_list):
            if (self._board.ENABLE_FWU == 0) and (comp_name == 'Stitch_FWU.bin') :
                print("No firmware update payload specified, skip firmware update.")
                continue
            out_file = comp_name
            out_path = os.path.join(self._fv_dir, out_file)
            bins = bytearray()
            new_list = []
            for src, algo, val, mode, pos in file_list:
                if mode & STITCH_OPS.MODE_FILE_IGNOR:
                    continue
                new_list.append((src, algo, val, mode, pos))
                if src == 'EMPTY':
                    bins.extend (b'\xff' * val)
                    continue
                src_path = os.path.join(self._fv_dir, src)
                bas_path = os.path.splitext(src_path)[0]
                if not os.path.exists(src_path):
                    raise Exception ("Component '%s' could not be found !" % src)

                if algo:
                    compress(src_path, algo)
                    src_path = bas_path + '.lz'
                else:
                    if src == 'STAGE2.fd':
                        raise Exception ("STAGE2.fd must be compressed, please change BoardConfig.py file !")
                if src not in rgn_name_list:
                    gen_hash_file (src_path, HASH_VAL_STRING[self._board.SIGN_HASH_TYPE], '', False)

                if mode != STITCH_OPS.MODE_FILE_NOP:
                    dst_path = bas_path + '.pad'
                    align_pad_file(src_path, dst_path, val, mode, pos)
                    src_path = dst_path
                else:
                    if val and (val != os.path.getsize(src_path)):
                        raise Exception ("Size of file '%s' does not match expected 0x%X !" % (src_path, val))
                if 'STAGE1A' in src :
                    self.update_hash_table (src_path)

                fi = open(src_path, 'rb')
                bins.extend(bytearray(fi.read()))
                fi.close()

            if comp_name == self._image:
                bins = b'\xff' * (self._board.SLIMBOOTLOADER_SIZE - len(bins)) + bins

            fo = open(out_path,'wb')
            fo.write(bins)
            fo.close()

            comp_file = out_file
            comp_node = find_component_in_image_list (comp_name, self._img_list)
            if comp_node:
                space = comp_node[2] - len(bins)
                if space > 0:
                    empty = ('EMPTY', '', space, STITCH_OPS.MODE_FILE_NOP, STITCH_OPS.MODE_POS_HEAD)
                    if comp_node[4] == STITCH_OPS.MODE_POS_HEAD:
                        new_list.insert (0, empty)
                    else:
                        new_list.append (empty)
                    comp_file = os.path.splitext(comp_name)[0] + '.pad'

            image_base = self._board.FLASH_LAYOUT_START
            comp_name  = comp_name.replace ('TOP_SWAP_B.', 'TOP_SWAP_A.')

            if comp_name in rgn_name_list:
                idx = rgn_name_list.index(comp_name)
                image_base = self._region_list[idx]['base'] + self._region_list[idx]['size']

            layout_file.write("IMAGE_INFO = ['%s', 0x%X, %d]\n" % (comp_file, image_base, True))
            layout_file.write("IMAGE_LIST = %s\n" % new_list)

        layout_file.close()

        self.update_fit_table ()

        # generate flash layout file
        layout_file = os.path.splitext(self._image)[0] + '.txt'
```

```
        report_image_layout (self._fv_dir, layout_name, layout_file)

        # copy files to staging directory for stitching
        if getattr(self._board, "GetOutputImages", None):
            extra_list = self._board.GetOutputImages()
        else:
            extra_list = []
        out_file = os.path.join("Outputs", self._board.BOARD_NAME, 'Stitch_Components.zip')
        copy_images_to_output (self._fv_dir, out_file, self._img_list, rgn_name_list, extra_list)


    def pre_build(self):
        # Update search path
        sbl_dir = os.environ['SBL_SOURCE']
        plt_dir = os.environ['PLT_SOURCE']
        os.environ['PACKAGES_PATH'] = plt_dir
        if plt_dir != sbl_dir:
            os.environ['PACKAGES_PATH'] += os.pathsep + sbl_dir

        # Generate SblKeys
        if self._board.KEY_GEN:
            if not os.path.exists(os.environ.get('SBL_KEY_DIR')):
                print ("Generating default Sbl Keys to %s !!" % os.environ.get('SBL_KEY_DIR'))
                key_gen_tool = os.path.join(sbl_dir, 'BootloaderCorePkg', 'Tools', 'GenerateKeys.py')
                args_list = ['python', key_gen_tool, '-k', os.environ['SBL_KEY_DIR']]
                ret = subprocess.call(args_list)
                if ret:
                    raise Exception  ('Failed to generate keys !!')
            else:
                print ("WARNING: Key generation option is set but directory with Sbl Keys exists at %s!!" % os.environ.get('SBL_KEY_DIR'))
                print ("Skip keys generation!!")

        # Check for SBL Keys directory
        check_for_slimbootkeydir()

        # create build folder if not exist
        if not os.path.exists(self._fv_dir):
            os.makedirs(self._fv_dir)

        # Validate HASH_TYPE_VALUE config
        if (HASH_TYPE_VALUE[self._board._SIGN_HASH] != self._board.SIGN_HASH_TYPE):
            raise Exception  ('SIGN_HASH_TYPE is not set correctly!!')

        # Validate IPP_HASH_LIB_SUPPORTED_MASK config
        if(IPP_CRYPTO_ALG_MASK[self._board._SIGN_HASH] & self._board.IPP_HASH_LIB_SUPPORTED_MASK) == 0:
            raise Exception  ('IPP_HASH_LIB_SUPPORTED_MASK is not set correctly!!')

        # check if FSP binary exists
        if self._board._FSP_PATH_NAME != '':
            fsp_path_name = self._board._FSP_PATH_NAME
        else:
            fsp_path_name = os.path.join('Silicon', self._board.SILICON_PKG_NAME, "FspBin")

        fsp_dir  = os.path.join(plt_dir,  fsp_path_name)
        work_dir = plt_dir
        if not os.path.exists(fsp_dir):
            fsp_dir  = os.path.join(sbl_dir, fsp_path_name)
            work_dir = sbl_dir
        fsp_path = os.path.join(fsp_dir, self._fsp_basename + '.bin')

        if self._board.HAVE_FSP_BIN:
            check_build_component_bin = os.path.join(tool_dir, 'PrepareBuildComponentBin.py')
            if os.path.exists(check_build_component_bin):
                ret = subprocess.call([sys.executable, check_build_component_bin, work_dir, self._board.SILICON_PKG_NAME, '/d' if self._board.FSPDEBUG_MODE else '/r'])
                if ret:
                    raise Exception  ('Failed to prepare build component binaries !')

        # create FSP size and UPD size can be known
        fsp_list = ['FSP_T', 'FSP_M', 'FSP_S']
        if self._board.HAVE_FSP_BIN:
            split_fsp (fsp_path, self._fv_dir)
        else:
            # create dummy FSP files
            for each in fsp_list:
                open(os.path.join(self._fv_dir, each + '.bin'),'wb').close()
        # generate size variables
        for each in fsp_list:
            fsp_bin  = os.path.join(self._fv_dir, "%s.bin" % each)
            if self._board.HAVE_FSP_BIN:
                if self._board.FSP_IMAGE_ID:
                    imageid = get_fsp_image_id (fsp_bin)
                    if self._board.FSP_IMAGE_ID != imageid:
                        raise Exception ('Expecting FSP ImageId: %s, but got %s !' % (self._board.FSP_IMAGE_ID, imageid))
                revision = get_fsp_revision (fsp_bin)
                if revision < self._board.MIN_FSP_REVISION:
                    raise Exception ('Required minimum FSP revision is 0x%08X, but current revision is 0x%08X !' %
                                     (self._board.MIN_FSP_REVISION, revision))
            setattr(self._board, '%s_SIZE' % each, get_fsp_size(fsp_bin) if self._board.HAVE_FSP_BIN else 0)
            setattr(self._board, '%s_UPD_SIZE' % each, get_fsp_upd_size(fsp_bin) if self._board.HAVE_FSP_BIN else 1)

        if self._board.BUILD_CSME_UPDATE_DRIVER:
            if os.name != 'nt':
                raise Exception  ('BUILD_CSME_UPDATE_DRIVER is enabled, build only works in WINDOWS !')

        # create component base/size variables
        self.update_component_list ()
        self.create_platform_vars ()

        # generate a padding file
        gen_file_with_size (os.path.join(self._fv_dir, 'PADDING.bin'),  0)

        # create flashmap file
        comp_list = self._comp_list
        if len(self._comp_list) == 0 and getattr(self._board, "GetFlashMapList", None):
            comp_list = self._board.GetFlashMapList()
        gen_flash_map_bin (os.path.join(self._fv_dir, 'FlashMap.bin'), comp_list)

        if not self._board.HAVE_VERIFIED_BOOT and self._board.HAVE_MEASURED_BOOT:
            raise Exception ('Verified Boot must also enabled to enable Measured Boot!')

        # create hashstore file
        key_hash_list = []
        mst_key = self._board._MASTER_PRIVATE_KEY
        if self._board.HAVE_VERIFIED_BOOT:
            hash_store_size = sizeof(HashStoreTable) + (sizeof(HashStoreData) + HASH_DIGEST_SIZE[HASH_VAL_STRING[self._board.SIGN_HASH_TYPE]]) * HashStoreTable().HASH_STORE_MAX_IDX_NUM
            gen_file_with_size (os.path.join(self._fv_dir, 'HashStore.bin'), hash_store_size)
```

```
1231
1232            #Intialize with HashStoreTable
1233            fo = open(os.path.join(self._fv_dir, 'HashStore.bin'),'r+b')
1234            hash_store_table = HashStoreTable()
1235            hash_store_table.TotalLength = hash_store_size
1236            fo.write(hash_store_table)
1237            fo.close()
1238
1239            # create key hash file
1240            if getattr(self._board, "GetKeyHashList", None):
1241                key_hash_list = self._board.GetKeyHashList ()
1242
1243        svn = self._board.KEYH_SVN
1244        gen_pub_key_hash_store (mst_key, key_hash_list, HASH_VAL_STRING[self._board.SIGN_HASH_TYPE],
1245                                self._board._SIGNING_SCHEME, svn, self._key_dir, os.path.join(self._fv_dir, 'KEYHASH.bin'))
1246
1247        # create fit table
1248        if self._board.HAVE_FIT_TABLE:
1249            FitSize = sizeof(FIT_ENTRY) * (self._board.FIT_ENTRY_MAX_NUM - 1)
1250            if self._board.ACM_SIZE > 0:
1251                # Make sure FIT table start address and end address are 64 bytes aligned.
1252                FitSize = ((FitSize + 0x3F) & ~0x3F) + 0x3F
1253            gen_file_with_size (os.path.join(self._fv_dir, 'FitTable.bin'),  FitSize)
1254
1255
1256        # create bootloader version info file
1257        ver_info_name = 'VerInfo'
1258        ver_bin_file = os.path.join(self._fv_dir, ver_info_name + '.bin')
1259        ver_txt_file = os.path.join(os.environ['PLT_SOURCE'], 'Platform', self._board.BOARD_PKG_NAME, ver_info_name + '.txt')
1260
1261        keys = ['VERINFO_IMAGE_ID', 'VERINFO_BUILD_DATE', 'VERINFO_PROJ_MINOR_VER',
1262                'VERINFO_PROJ_MAJOR_VER', 'VERINFO_CORE_MINOR_VER', 'VERINFO_CORE_MAJOR_VER',
1263                'VERINFO_SVN', 'FSPDEBUG_MODE', 'RELEASE_MODE', 'BUILD_ARCH']
1264        ver_dict = {}
1265        for key in keys:
1266            ver_dict[key] = getattr (self._board, key)
1267        if self._board.USE_VERSION:
1268            ver_info = get_verinfo_via_file (ver_dict, ver_txt_file)
1269        else:
1270            ver_info = get_verinfo_via_git  (ver_dict, os.environ['PLT_SOURCE'])
1271            gen_ver_info_txt (ver_txt_file, ver_info)
1272        gen_file_from_object (ver_bin_file, ver_info)
1273
1274        # create VBT file
1275        if self._board.HAVE_VBT_BIN:
1276            gen_vbt_file (self._board.BOARD_PKG_NAME, self._board._MULTI_VBT_FILE, os.path.join(self._fv_dir, 'Vbt.bin'))
1277
1278        # create platform include dsc file
1279        # 生成Platform.dsc文件
1280        platform_dsc_path = os.path.join(sbl_dir, 'BootloaderCorePkg', 'Platform.dsc')
1281        self.create_dsc_inc_file (platform_dsc_path)
1282
1283        # rebase FSP accordingly
1284        if self._board.HAVE_FSP_BIN:
1285            rebase_fsp(fsp_path, self._fv_dir, self._board.FSP_T_BASE, self._board.FSP_M_BASE, self._board.FSP_S_BASE)
1286            split_fsp(os.path.join(self._fv_dir, 'Fsp.bin'), self._fv_dir)
1287
1288        # create master key hash
1289        if self._board.HAVE_VERIFIED_BOOT:
1290            mst_priv_key     = self._board._MASTER_PRIVATE_KEY
1291            mst_pub_key_file = os.path.join(self._fv_dir, "MSTKEY.bin")
1292            gen_pub_key (mst_priv_key, mst_pub_key_file)
1293            gen_hash_file (mst_pub_key_file, HASH_VAL_STRING[self._board.SIGN_HASH_TYPE], '', True)
1294
1295        # create configuration data
1296        if self._board.CFGDATA_SIZE > 0:
1297            svn = self._board.CFGDATA_SVN
1298            # create config data files
1299            gen_config_file (self._fv_dir, self._board.BOARD_PKG_NAME, self._board._PLATFORM_ID,
1300                             self._board._CFGDATA_PRIVATE_KEY, self._board.CFG_DATABASE_SIZE, self._board.CFGDATA_SIZE,
1301                             self._board._CFGDATA_INT_FILE, self._board._CFGDATA_EXT_FILE,
1302                             self._board._SIGNING_SCHEME, HASH_VAL_STRING[self._board.SIGN_HASH_TYPE], svn)
1303
1304        # rebuild reset vector
1305        vtf_dir = os.path.join('BootloaderCorePkg', 'Stage1A', 'Ia32', 'Vtf0')
1306        x = subprocess.call([sys.executable, 'Build.py', self._arch.lower()],  cwd=vtf_dir)
1307        if x: raise Exception ('Failed to build reset vector !')
1308
1309    def build(self):
1310        print("Build [%s] ..." % self._board.BOARD_NAME)
1311
1312        # Run pre-build
1313        # self.board_build_hook都是在特定单板定义的
1314        self.board_build_hook ('pre-build:before')
1315        self.pre_build()
1316        self.board_build_hook ('pre-build:after')
1317
1318        # Run build
1319        cmd_args = [
1320            "build" if os.name == 'posix' else "build.bat",
1321            "--platform", os.path.join('BootloaderCorePkg', 'BootloaderCorePkg.dsc'),
1322            "-b",         self._target,
1323            "--arch",     self._arch,
1324            "--tagname",  self._toolchain,
1325            "-n",         str(multiprocessing.cpu_count()),
1326            "-y",         "Report.log",
1327            "-Y",         "PCD",
1328            "-Y",         "FLASH",
1329            "-Y",         "LIBRARY"]
1330        run_process (cmd_args)
1331
1332        # Run post-build
1333        self.board_build_hook ('post-build:before')
1334        self.post_build()
1335        self.board_build_hook ('post-build:after')
1336
1337        print("Done [%s] !" % self._board.BOARD_NAME)
1338
1339
1340    def post_build(self):
1341
1342        # create bootloader reserved binary of 4K size
1343        # 这里创建的文件都是全FF，而没有实际内容
1344        gen_file_with_size (os.path.join(self._fv_dir, 'SBLRSVD.bin'), 0x1000)
1345
1346        # create variable region for UEFI payload
```

```
        # 上面似乎写错了，应该是不Payload，而是UEFI变量
        if self._board.UEFI_VARIABLE_SIZE > 0:
            gen_file_with_size (os.path.join(self._fv_dir, 'UEFIVARIABLE.bin'), self._board.UEFI_VARIABLE_SIZE)

        # create ACM binary
        if self._board.ACM_SIZE > 0:
            gen_file_with_size (os.path.join(self._fv_dir, 'ACM.bin'), self._board.ACM_SIZE)

        # create Diagnostic ACM binary
        if self._board.DIAGNOSTICACM_SIZE > 0:
            gen_file_with_size (os.path.join(self._fv_dir, 'DIAGNOSTICACM.bin'), self._board.DIAGNOSTICACM_SIZE)

        # create MRC data
        if self._board.MRCDATA_SIZE:
            gen_file_with_size (os.path.join(self._fv_dir, 'MRCDATA.bin'), self._board.MRCDATA_SIZE)


        # create variable binary
        if self._board.VARIABLE_SIZE:
            # VariableRegionHeader类在BuildUtility.py定义，继承自Structure，可以使用from_buffer得到结构体数据
            varhdr = VariableRegionHeader.from_buffer(bytearray(b'\xFF' * sizeof(VariableRegionHeader)))
            varhdr.Signature = b'VARS'
            varhdr.Size      = self._board.VARIABLE_SIZE >> 1
            varhdr.State     = 0xFE
            varfile = open (os.path.join(self._fv_dir, "VARIABLE.bin"), "wb")
            varfile.write(varhdr)
            varfile.write(b'\xFF' * (self._board.VARIABLE_SIZE - sizeof(varhdr)));
            varfile.close()


        # create microcode binary
        if self._board.UCODE_SIZE > 0:
            shutil.copy (
                os.path.join(self._fv_dir, '../%s/Microcode.bin' % self._arch),
                os.path.join(self._fv_dir, "UCODE.bin"))

        # generate payload
        gen_payload_bin (self._fv_dir, self._arch, self._pld_list,
                         os.path.join(self._fv_dir, "PAYLOAD.bin"),
                         self._board._CONTAINER_PRIVATE_KEY, HASH_VAL_STRING[self._board.SIGN_HASH_TYPE],
                         self._board._SIGNING_SCHEME, self._board.BOARD_PKG_NAME)

        # create firmware update key
        if self._board.ENABLE_FWU:
            srcfile = "../%s/PayloadPkg/FirmwareUpdate/FirmwareUpdate/OUTPUT/FirmwareUpdate.efi" % self._arch
            shutil.copyfile(
                os.path.join(self._fv_dir, srcfile),
                os.path.join(self._fv_dir, "FWUPDATE.bin"))

        # create SPI IAS image if required
        if self._board.SPI_IAS1_SIZE > 0 or self._board.SPI_IAS2_SIZE > 0:
            for idx in range (1, 3):
                file_path  = os.path.join('Platform', self._board.BOARD_PKG_NAME, 'SpiIasBin', 'iasimage%d.bin' % idx)
                file_space = getattr(self._board, 'SPI_IAS%d_SIZE' % idx)
                gen_ias_file (file_path, file_space, os.path.join(self._fv_dir, "SPI_IAS%d.bin" % idx))

        # generate container images
        if getattr(self._board, "GetContainerList", None):
            container_list = self._board.GetContainerList ()
            component_dir = os.path.join(os.environ['PLT_SOURCE'], 'Platform', self._board.BOARD_PKG_NAME, 'Binaries')
            gen_container_bin (container_list, self._fv_dir, component_dir, self._key_dir , '')

        # patch stages
        self.patch_stages ()

        # create redundant components
        self.create_redundant_components ()


        # stitch all components
        layout_name = 'ImgStitch.txt'
        self.create_bootloader_image (layout_name)

        # print flash map
        if len(self._comp_list) > 0:
            print_addr = False if getattr(self._board, "GetFlashMapList", None) else True
            flash_map_text = decode_flash_map (os.path.join(self._fv_dir, 'FlashMap.bin'), print_addr)
            print('%s' % flash_map_text)
            fd = open (os.path.join(self._fv_dir, 'FlashMap.txt'), 'w')
            fd.write (flash_map_text)
            fd.close ()


def main():

    # Set SBL_SOURCE and WORKSPACE Environment variable at first
    # SBL_SOURCE和WORKSPACE的值都是当前文件所在的目录，且是绝对路径
    os.environ['SBL_SOURCE'] = os.path.dirname(os.path.abspath(__file__))
    if 'WORKSPACE' not in os.environ:
        os.environ['WORKSPACE'] = os.environ['SBL_SOURCE']

    board_cfgs   = []
    board_names  = []
    module_names = []

    # Find all boards
    # search_dir也是当前文件所在路径
    search_dir = os.environ['SBL_SOURCE']
    # PLT_SOURCE默认应该是没有的，所以不会进入if分支
    if 'PLT_SOURCE' in os.environ:
        search_dir = os.path.abspath(os.path.join(search_dir, os.path.pardir))
    # 就是根目录下所有的目录和文件，注意listdir()还会返回文件
    board_pkgs = os.listdir (search_dir)
    for pkg in board_pkgs:
        get_board_config_file (os.path.join (search_dir, pkg), board_cfgs)

    board_cfgs.sort()
    for cfgfile in board_cfgs:
        # 就是Board
        module_name = os.path.basename(os.path.dirname(cfgfile))[:-8] + os.path.basename(cfgfile)[:-3]
        # 获得的就是平台Package中的BoardConfig.py导入的模块，里面有Board类，它是BaseBoard的子类
        brdcfg = load_source(module_name, cfgfile)
        # 在BoardConfig.py中定义的BOARD_NAME，比如'apl'、'qemu'等
        board_names.append(brdcfg.Board().BOARD_NAME)
        module_names.append(brdcfg)

```

```python
# 开始处理命令参数，参数有对应的执行函数，比如build参数就对应这里的cmd_build函数，以此类推
ap = argparse.ArgumentParser()
sp = ap.add_subparsers(help='command')

# 参数build对应执行的函数
def cmd_build(args):
    prep_env (args.toolchain)

    for index, name in enumerate(board_names):
        if args.board == name:
            brdcfg = module_names[index]
            # 来自BoardConfig.py中的Board类，USE_VERSION这个参数没有用到，不过Python可以增加新的成员在类实例中，只是最终似乎也没有用到
            board  = brdcfg.Board( # =前面的是BaseBoard类的成员，后面的是参数值
                                    BUILD_ARCH        = args.arch.upper(), \
                                    RELEASE_MODE      = args.release,     \
                                    NO_OPT_MODE       = args.noopt,       \
                                    FSPDEBUG_MODE     = args.fspdebug,    \
                                    USE_VERSION       = args.usever,      \
                                    _PAYLOAD_NAME     = args.payload,     \
                                    _FSP_PATH_NAME    = args.fsppath,     \
                                    KEY_GEN           = args.keygen
                                    );
            os.environ['PLT_SOURCE']  = os.path.abspath (os.path.join (os.path.dirname (board_cfgs[index]), '../..'))
            Build(board).build()
            break

buildp = sp.add_parser('build', help='build SBL firmware')
buildp.add_argument('-r',  '--release', action='store_true', help='Release build')
buildp.add_argument('-v',  '--usever',  action='store_true', help='Use board version file')
buildp.add_argument('-fp', dest='fsppath', type=str, help='FSP binary path relative to FspBin in Silicon folder', default='')
buildp.add_argument('-fd', '--fspdebug', action='store_true', help='Use debug FSP binary')
buildp.add_argument('-a',  '--arch', choices=['ia32', 'x64'], help='Specify the ARCH for build. Default is to build IA32 image.', default ='ia32')
buildp.add_argument('-no', '--noopt', action='store_true', help='No compile/link optimization for debugging purpose. Not enabled in Release build.')
buildp.add_argument('-p',  '--payload' , dest='payload', type=str, help='Payload file name', default ='OsLoader.efi')
buildp.add_argument('board', metavar='board', choices=board_names, help='Board Name (%s)' % ', '.join(board_names))
buildp.add_argument('-k', '--keygen', action='store_true', help='Generate default keys for signing')
buildp.add_argument('-t', '--toolchain', dest='toolchain', type=str, default='', help='Perferred toolchain name')
buildp.set_defaults(func=cmd_build)

# 参数clean对应执行的函数
def cmd_clean(args):
    workspace = os.environ['WORKSPACE']
    sbl_dir   = os.environ['SBL_SOURCE']
    dirs  = ['Build', 'Conf']
    files = [
        os.path.join (sbl_dir, 'BootloaderCorePkg/Stage1A/Ia32/Vtf0/Bin/ResetVector.ia32.raw'),
        os.path.join (sbl_dir, 'BootloaderCorePkg/Platform.dsc'),
        os.path.join (workspace, 'Report.log')
    ]

    if args.distclean:
        dirs.extend ([
            'Outputs',
        ])

        files.extend ([
        ])

    for dir in dirs:
        dirpath = os.path.join (workspace, dir)
        print('Removing %s' % dirpath)
        shutil.rmtree(dirpath, ignore_errors=True)

    for file in files:
        if os.path.exists(file):
            print('Removing %s' % file)
            os.remove(file)

    if os.path.exists(os.path.join (sbl_dir, '.git')):
        cmd = 'git clean -xdf BaseTools'
        x = subprocess.call(cmd.split(' '), cwd=sbl_dir)
        if x: raise Exception ('Failed to run clean-up commands !')

    print('Clean Done !')

cleanp = sp.add_parser('clean', help='clean build dir')
cleanp.add_argument('-d',  '--distclean', action='store_true', help='Distribution clean')
cleanp.set_defaults(func=cmd_clean)

# 参数build_dsc执行的主体
def cmd_build_dsc(args):
    prep_env (args.toolchain)

    # Build a specified DSC file
    def_list = []
    if args.define is not None:
        for each in args.define:
            def_list.extend (['-D', '%s' % each])

    cmd_args = [
        "build" if os.name == 'posix' else "build.bat",
        "--platform", args.dsc,
        "-b",         'RELEASE' if args.release else 'DEBUG',
        "--arch",     args.arch.upper(),
        "--tagname",  os.environ['TOOL_CHAIN'],
        "-n",         str(multiprocessing.cpu_count()),
        ] + def_list

    run_process (cmd_args)

build_dscp = sp.add_parser('build_dsc', help='build a specified dsc file')
build_dscp.add_argument('-r',  '--release', action='store_true', help='Release build')
build_dscp.add_argument('-a',  '--arch', choices=['ia32', 'x64'], help='Specify the ARCH for build. Default is to build IA32 image.', default ='ia32')
build_dscp.add_argument('-d',  '--define', action='append', help='Specify macros to be passed into DSC build')
build_dscp.add_argument('-p',  '--dsc', type=str, required=True, help='Specify a DSC file path to build')
build_dscp.add_argument('-t',  '--toolchain', dest='toolchain', type=str, default='', help='Perferred toolchain name')
build_dscp.set_defaults(func=cmd_build_dsc)

args = ap.parse_args()
if len(args.__dict__) <= 1:
    # No arguments or subcommands were given.
    ap.print_help()
    ap.exit()

args.func(args)
```

```
if __name__ == '__main__':
    main()
```