UEFI 执行流程

及

BSP Performance Debug

haobo.gao

Foxconn ZZDC

September 28, 2018

Contents

Chapter 1

Bsp Performance Debug

前段时间,我尝试帮助分析了 PL2P-2393 关于

```
("Power On to home screen (for 2nd boot or later)"is worse than PL2O.(PL2P: 37.52sec, PL2O
      : 31.26sec))
```

Performance 方面的一个 issue. 接下来和大家分享下过程中遇到的问题和得到的经验。

1.1 思路

1.1.1 控制硬件及其参数配置不变

首先用最好使用同一台手机,进行问题复现。这样能确定硬件的一致。

在硬件一致的前提下,我们首先要确定软件方面对于硬件方面的配置是否有差异。根据我的认知,以下三方面的硬件差异会对 Perf ormance 造成影响:

- CPU 的频率
- memory 的大小及频率
- flash 的 Throughput(吞吐量)

由于是同一台手机,所以硬件因素是可以保证的。关于参数的配置通过 uefi log 的打印我们可以看到 log 中存在:

```
S - Core 0 Frequency, 3952 MHz
S - Flash Throughput, 80000 KB/s  (4694664 Bytes,  58271 us)
DDR Frequency, 1296 MHz
Core 0 Freq: 1670400 MHz
```

要确认以上参数的一致,不过发现 Flash Throughput 会有小范围浮动,不足以造成很大的差异。

1.1.2 kernel 配置

在确定软件对于硬件的配置参数都一样的情况下,我们也要确认一下使用的 kernel 编译配置。kernel 有两种编译配置,一种用在开发阶段是 sdm660_defconfig。这种配置编译出来的

kernel 中有一些 debug 信息，高通文档描述说是 has bad performance。还有一种用于产品阶段的配置，sdm660-perf_defconfig 这个配置摘掉了 debug info 为 good performance。

编译一个 userdebug 版本 bad performance：

```
1 TARGET_BUILD_VARIANT=userdebug
```

编译一个 user 版本 good performance：

```
1 TARGET_BUILD_VARIANT=user
```

1.1.3 版本升级因素

考虑可能是版本升级代码变动造成的影响。我们需要抓取 log 锁定问题的范围。

1.2 本文必备的 uefi 基础

为了能看懂 log，需要对 uefi 启动流程做一个大致的介绍，由于我还没有完全把我看到的文档解释和 uefi 的源码对应起来，很多地方还存在疑惑，所以并不能详细介绍 uefi 的启动流程。但是现阶段可以确定的部分足以用来处理这个问题。

1.2.1 uefi 组件

xbl 的源代码目录

位于 BOOT.XF.1.4/boot_images/目录下的代码编译可以生成 xbl.elf 和 pmic.elf 等 1.1:

abl 的源代码目录

位于 LINUX/android/bootable/bootloader/edk2/ 目录下的代码编译成为 abl.elf 1.2:

效果图

以下是效果图1.3。

1.2.2 xbl region

xbl 的编译过程

下图1.4简单展示了 XBL 是如何被编译成为 elf 文件的。

xbl region

xbl.elf 中有 4 个域1.5。

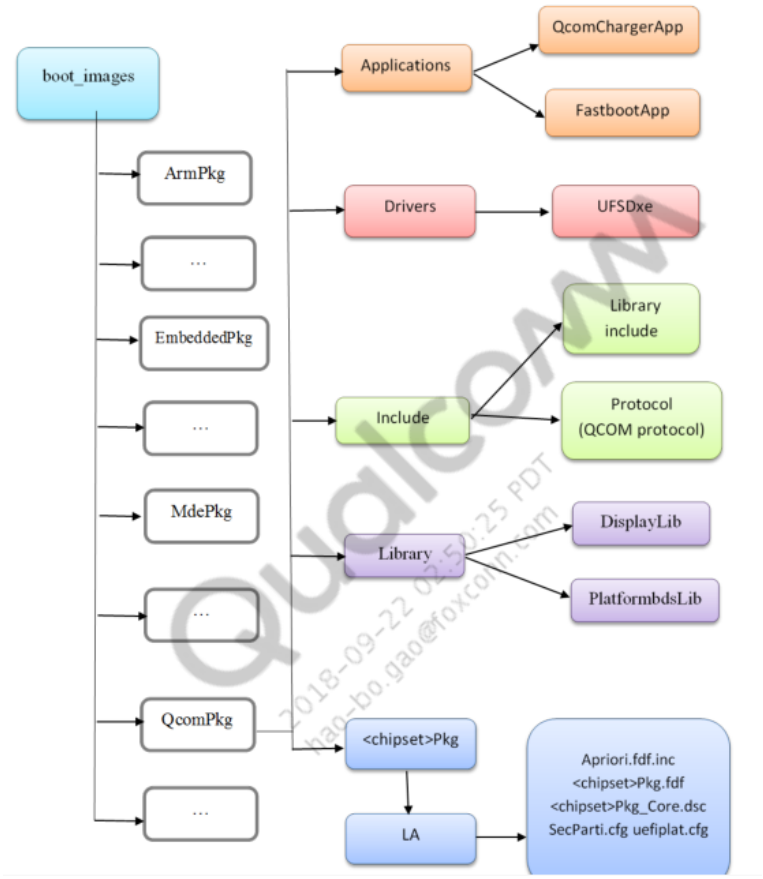

Figure 1.1: xbl 目录

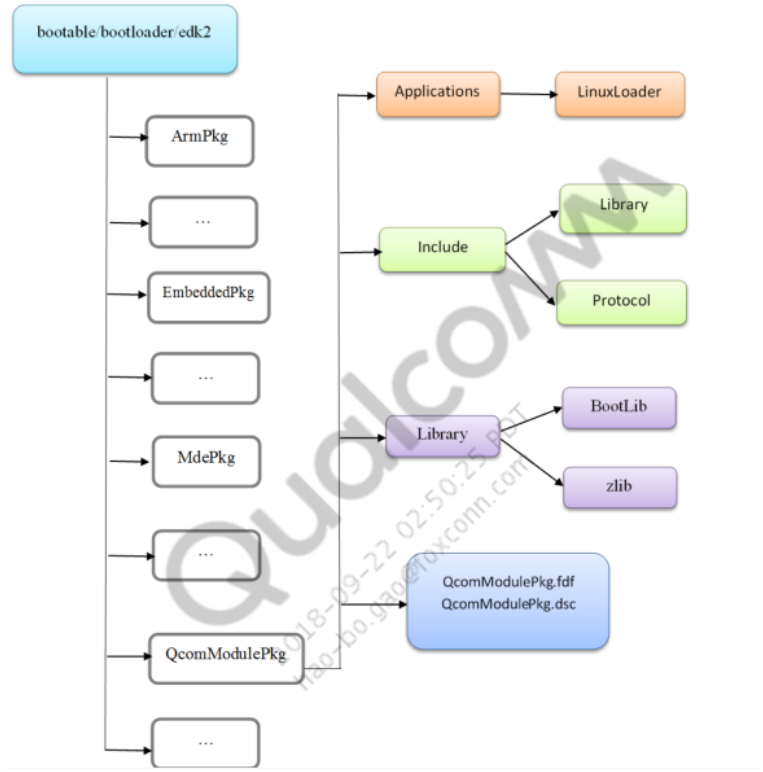

Figure 1.2: abl 目录

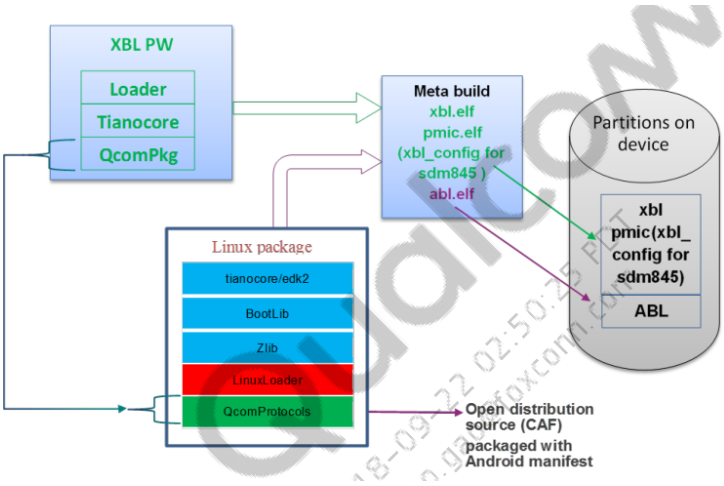

Figure 1.3: uefi build chat

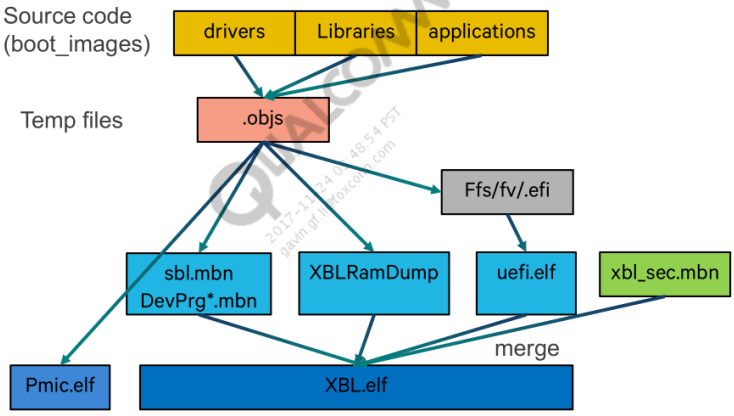

Figure 1.4: xbl made for

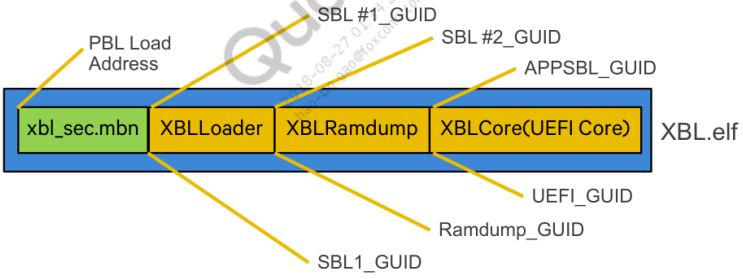

Figure 1.5: xbl region

1.2.3 uefi 执行流程

图1.6表示 uefi 执行的阶段。sec 相关的 region 最先被执行。接下来会去加载 DXE , 也就是驱动。到 BDS（boot device select）时, 会检测 hot key, 并执行与 hot key 对应的动作。本次我们讨论的是加载 abl。接着, 由 Android 源码目录下的 edk2 编译生成的 abl.elf 中的 linux_loader 完成 HLOS 的加载和执行。uefi boot serveice 结束。

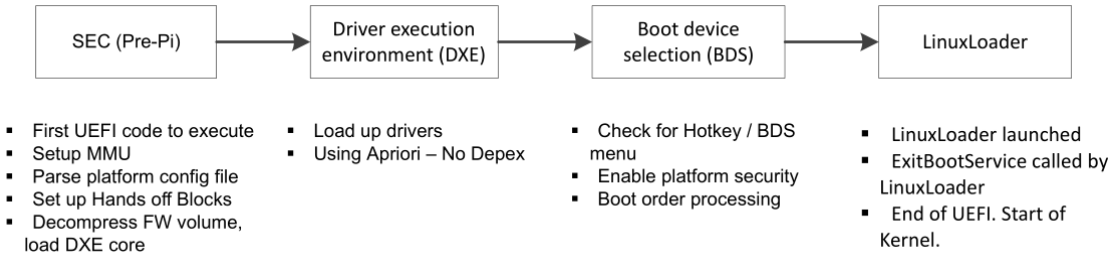

Figure 1.6: uefi 阶段

1.3 代码位置以及 log 片段介绍

这里介绍 uefi 一些阶段对应的 log 以及其对应的代码。

1.3.1 UEFI Start 之前的 log

如果你使用串口抓取 log , 是不会看到有2.1, 所列举的 log. 这是因为在 xbl loader 编译的时候没有打开 FEATURE_BOOT_LOGGER_UART 编译选项, 所以对于早期的串口打印函数比如 boot_log_message_raw. 在没有打开这个选项的时候, 用于串口打印的 inline 函数被声明为一个空的函数体。所以同样的 log 仅仅通过 boot_log_message_ram 记录在 log buffer 中随后某个时间被插入到 blog 的开始部分。

如果十分想要看这部分的串口 log, 又没有更好的方法, 请尝试在

```
1 boot_images/QcomPkg/Sdm660Pkg/Library/XBLLoaderLib/XBLLoaderDevProgLib.inf
```

文件中, 找到如下文本片段:

```
1 [BuildOptions.AARCH64]
2 GCC:*_*_*_CC_FLAGS = -Werror -DBOOT_LOADER -DBOOT_WATCHDOG_DISABLED -DBOOT_PBL_H=\"
      pbl_sbl_shared.h\" -DBUILD_BOOT_CHAIN -DRAM_PARTITION_TABLE_H=\"ram_partition.h\" -
      DBOOT_INTERNAL_HEAP_SIZE=0x01800 -DBOOT_EXTERNAL_HEAP_SIZE=0x10000 -
      DFEATURE_BOOT_SDCC_BOOT -DFEATURE_BOOT_LOAD_ELF -
      DFEATURE_BOOT_SKIP_ELF_HASH_VERIFICATION -DFEATURE_BOOT_VERSION_ROLL_BACK -
      DUSE_GNU_LD -DUSE_LOADER_UTILS -DFEATURE_BOOT_LOGGER_RAM -
      DFEATURE_BOOT_LOGGER_TIMER -DFEATURE_BOOT_LOGGER_JTAG -DFEATURE_BOOT_LOGGER_UART -
      DFEATURE_BOOT_EXTERN_SECIMG_AUTH_COMPLETED -DFEATURE_DEVICEPROGRAMMER_IMAGE
3 MSFT:*_*_*_CC_FLAGS = -DBOOT_LOADER -DBOOT_WATCHDOG_DISABLED -DBOOT_PBL_H=\"pbl_sbl_shared
      .h\" -DBUILD_BOOT_CHAIN -DRAM_PARTITION_TABLE_H=\"ram_partition.h\" -
      DBOOT_INTERNAL_HEAP_SIZE=0x01800 -DBOOT_EXTERNAL_HEAP_SIZE=0x10000 -
      DFEATURE_BOOT_SDCC_BOOT -DFEATURE_BOOT_LOAD_ELF -
      DFEATURE_BOOT_SKIP_ELF_HASH_VERIFICATION -DFEATURE_BOOT_VERSION_ROLL_BACK -
      DUSE_GNU_LD -DUSE_LOADER_UTILS -DFEATURE_BOOT_LOGGER_RAM -
      DFEATURE_BOOT_LOGGER_TIMER -DFEATURE_BOOT_LOGGER_JTAG -DFEATURE_BOOT_LOGGER_UART -
      DFEATURE_BOOT_EXTERN_SECIMG_AUTH_COMPLETED -DFEATURE_DEVICEPROGRAMMER_IMAGE
```

并在对应的位置添加 -DFEATURE_BOOT_LOGGER_UART 编译选项。

```
#ifdef FEATURE_BOOT_LOGGER_UART
void boot_log_message_uart(char *, uint32, char, char *);
#else
static inline void boot_log_message_uart(char * m, uint32 t, char l, char * c)
{
}
#endif

void boot_log_message_raw(char * message,uint32 timestamp,char log_type,char *
      optional_data)
{
  /*Logs message with time stamp in ram, must be initialized first.*/
  boot_log_message_ram(message,timestamp,log_type,optional_data);
  /* Transmit the message with time stamp */
  boot_log_message_uart(message,timestamp,log_type,optional_data);
}
```

PBL LOG

关于这部分 log 中出现的:

```
PBL, Start
PBL, End
```

容易让人怀疑 PBL 在执行的时候是否打印出了如下的 log。并且在代码中搜索相关字符串,居然能搜索到。根据高通文档描述我们这里应该是没有 PBL 部分的源码的,PBL 是固化到 Rom 中的一小段程序。

经过追查发现,如下代码解释了这个问题:

```
/*@brief
*   This funcion will parse the PBL timestamp milestones passed to SBL
*   and insert them into the boot log.  Currently PBL's unit of measure is
*   clock ticks.  PBL does not pass the clock frequency yet so it is hard
*   wired to 19.2 Mhz.  Also PBL does not pass the names of each field so for
*   now reference a structure of our own to get the names which will have
*   logic ready for the day PBL starts passing them.
*/
void boot_pbl_log_milestones(boot_pbl_shared_data_type * pbl_shared_data)
{
    ...
  for (count = 0;count < (sizeof(pbl_shared_data->timestamps) / sizeof(uint32));count++,
       pbl_timestamp++)
  {
    pbl_us_value = ( ( (uint64)(*pbl_timestamp) ) * PS_PER_PBL_TIMESTAMP_TICK ) / 1000000;
    /*Logs message and time. */
    boot_log_message_raw(boot_log_pbl_milestone_names[count],(uint32)pbl_us_value,
          LOG_MSG_TYPE_BOOT,NULL);
  }
    ...
}
```

这部分代码打印了那段 log , 注释上大意是,这个函数会去把 PBL 传递过来 (SBL) 的时间戳数据加到 bootlog 中。PBL 时期的时钟是连接到 19.2MHZ, 并且 PBL 并没有把每个时间戳对应的名字传递过来,所以在 SBL 中需要自己定义一组这样的名字。

高通启动流程

参见1.7.boot 详细流程感兴趣的可以去看高通的相关文档[1] 我在这里仅仅简要介绍。

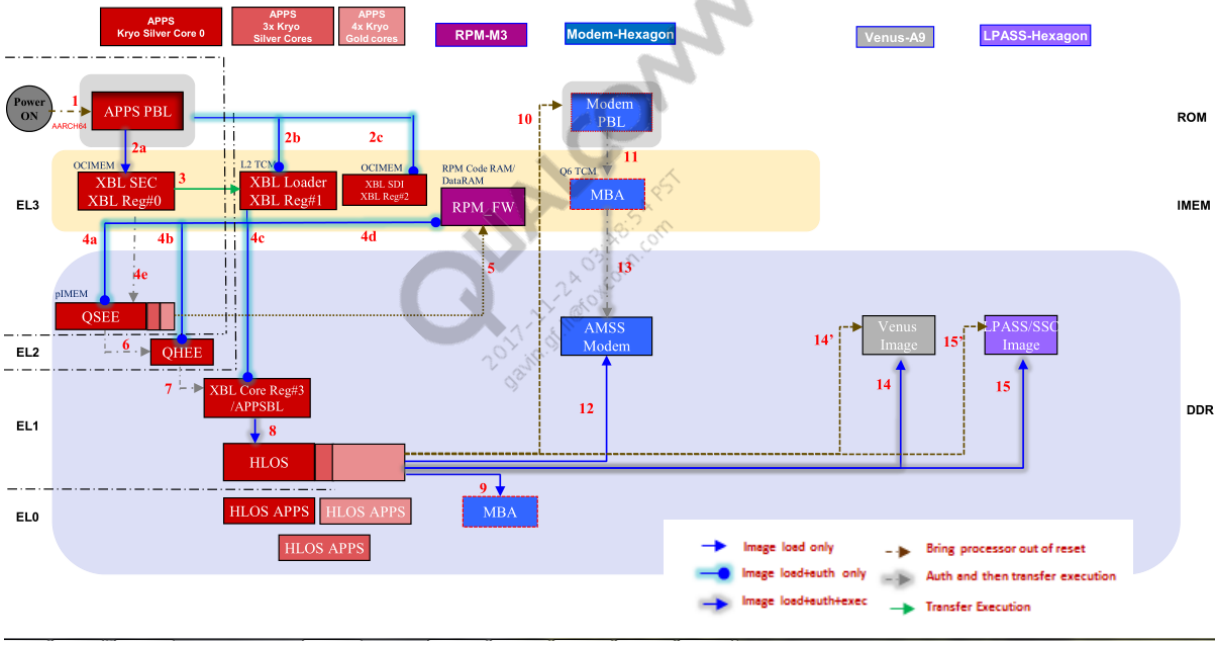

Figure 1.7: SDM660 boot flow

1.3.2 xbl loader

从有 log 打印开始已经在 xbl loader 中了。

boot_images/QcomPkg/Sdm660Pkg/Library/XBLLoaderLib/sbl1_mc.c

```
1 sbl1_main_ctl
```

log2.1 是由 sbl1_main_ctl 的执行产生的，sbl1_main_ctl 是 xbl_loader 的主体。主要完成了 ram 的配置和初始化，初始化了堆栈，内存映射，qsee 接口。设置 hash 运算检测是否使能 secboot。然后通过 boot_config_process_bl2.2.1 跳去执行 sbl1_config_table 中定义的动作，见代码2.2.2。在 boot_config_process_bl 中在 for 循环中调用了 boot_config_process_entry，处理每一个在 sbl1_config_table 上的表项。具体做了：

- 执行每一个表项中的 pre_procs。这个函数会在加载镜像前执行。
- 检查 boot_load_cancel 是否被注册（指向一个函数），有时候注册表中可能会注册这样一个类型的函数检查镜像的加载是否该被取消。
- 执行 Load/Authenticate, 根据 image type 我们加载认证镜像。
- Execute, 如果 configuration table 中的 excute 是 TURE ，将在 load 和 Authenticate 之后立刻执行 exec_func。这个动作是和下文提到的 jump 是互斥的。
- Execute post_procs , 在被加载认证执行之后，执行 post_procs.

[1] 80_P8754_2_D_SDM660_SDM630_BOOT_AND_COREBSP_ARCHIT.pdf

- 如果 configuration table 中的 jump 是 TURE，会在 post_procs 之后执行 jump_func. 这个动作是和上文提到的 execute 相互斥。

以上这个步骤对应了图1.7中的:

- 4a. Loads and authenticates the QSEE image from the boot device to pIMEM.
- 4b. Loads and authenticates the QHEE image from the boot device to DDR.
- 4c. Loads and authenticates the XBL Core (region #3) and ABL image from the boot device to DDR.
- 4d. Loads and authenticates the RPM firmware image from the boot device to RPM code RAM.
- 4e. XBL SEC transfers execution to QSEE.

之后控制权 qsee qsee 再去执行 Qhee 然后控制权交给 XBL core。这部分没有 log 也没有在代码中找到相关代码。

1.3.3 XBL core

xbl core 接管控制权后,程序的执行流已经来到了,boot_images/QcomPkg/XBLCore/Sec.c 中的 Main 函数。紧跟这的一些打印:

```
1 UEFI Start     [  757] SEC
```

其中的 DisplayEarlyInfo

```
1 UEFI Ver    : 4.2.180915.BOOT.XF.1.4-00258-S660LZB-1
2 Build Info  : 64b Sep 15 2018 16:39:17
3 Boot Device : eMMC
```

在 main 结尾会有这样一个调用

```
1 LoadDxeCoreFromFv (NULL, 0);
```

这个函数会去加载 DxeCore.efi , 然后去执行它。

现在进入了 DXE 阶段 DXE 阶段的入口为 DxeMain, 其中起了一些服务配置了一些表。然后通过:

```
1 Status = FwVolBlockDriverInit (gDxeCoreImageHandle, gDxeCoreST);
```

加载并执行了其他的一些.efi 文件(DRIVER),这是 dxe 的重要部分(加载驱动并执行),其中包含:

```
1  - 0x09B8D3000 [  916] FIHDxe.efi
```

在 DxeMain 的最后有:

```
1 gBds->Entry (gBds);
```

通过前面加载的 QcomBds.efi, 已经被配置为如下 [2]:

[2]代码在 BOOT.XF.1.4/boot_images/QcomPkg/Drivers/BdsDxe/BdsEntry.c

```
EFI_BDS_ARCH_PROTOCOL  gBds = {
  BdsEntry
};
```

1.3.4 BDS

现在程序来到了 BDS(boot device select)阶段。在 BdsEntry[3]2.2.3 中
在 BDS 的最后通过 LaunchDefaultBDSApps 加载执行:

- QcomChargerApp.efi
- FIHHWIDApp.efi
- LinuxLoader.efi

1.3.5 ABL/linuxloader

最后来到了 abl 中的 LinuxLoaderEntry

```
ENTRY_POINT                     = LinuxLoaderEntry
LinuxLoaderEntry
```

这里面有 fih 添加的很多标有 fih_ 的函数。判断 BootReason 属于

```
  NORMAL_MODE = 0x0,
  RECOVERY_MODE = 0x1,
  FASTBOOT_MODE = 0x2,
  ALARM_BOOT = 0x3,
  DM_VERITY_LOGGING = 0x4,
  DM_VERITY_ENFORCING = 0x5,
  DM_VERITY_KEYSCLEAR = 0x6,
  FACTORY_MODE         = 0x7,
  OEM_RESET_MIN = 0x20,
  OEM_RESET_MAX = 0x3f,
  EMERGENCY_DLOAD = 0xFF,
```

中哪一类。

最后:

```
BootLinux (&Info);
```

1.3.6 dmesg log 中挂接根文件系统开始 init 进程的标志

在 dmesg 中 kernel 挂接到根文件系统, 开始 init 进程的标志是:

```
[    1.345851] VFS: Mounted root (ext4 filesystem) readonly on device 252:0.
[    1.347865] Freeing unused kernel memory: 8192K
[    1.396266] init: init first stage started!
```

[3]boot_images/QcomPkg/Drivers/QcomBds

1.4 几种 log 的分析

1.4.1 dmesg

抓取方法:

```
adb shell dmesg > dmesg.txt
```

KPI(Key Performance Indicator)的计算

我们可以通过 dmesg 中的 KPI 信息计算出 boot loader 的时间, dmesg 中会有一些类似于这样的 log:

```
[    0.231817] KPI: Bootloader start count = 63197          \\A begin at edk
[    0.231822] KPI: Bootloader end count = 313686           \\B end of edk
[    0.231828] KPI: Bootloader display count = 3689772022
[    0.231833] KPI: Bootloader load kernel count = 7456
[    0.231840] KPI: Kernel MPM timestamp = 326766           \\C end of boot loader
[    0.231845] KPI: Kernel MPM Clock frequency = 32768      \\D clock
```

根据高通的文档[4]的关于 KPI 计算方面的描述, 我们可以计算出如下信息:

- NHLOS time : A/D = 1.93S
- abl time: (B-A)/D = 7.64S 可以从下文的 9558-710 = 7641 得到验证。
- Boot loader: C/D - kmsg(C)[5] = 9.97 - 0.23 = 9.74s

1.4.2 Other Kernel part

必要的时候我们也想知道内核每个模块的加载时间。从而判断是不是模块加载浪费了时间。

为了达成这个目的我们需要在

```
/LINUX/android/kernel/init/main.c

int __init_or_module do_one_initcall(initcall_t fn)
```

中的:

```
    if (initcall_debug)
        ret = do_one_initcall_debug(fn);
```

把 initcall_debug 替换为 1.

即:

```
    if (1)
        ret = do_one_initcall_debug(fn);
```

initcall_debug 本身是一个 bool 类型的变量, 在这里强制使执行流来到 do_one_initcall_debug 这样就可以在 dmesg 中看到一些各个模块加载的时间信息了:

[4] kba-160919232945_3_how_to_debug_boot_time_performance_issue

[5] 标记 C 的打印时刻, 即 0.231840

```
initcall msm_serial_hsl_init+0x0/0xac returned 0 after 262555 usecs
initcall fts_driver_init+0x0/0x20 returned 0 after 171317 usecs
initcall ufs_qcom_phy_qmp_20nm_driver_init+0x0/0x20 returned 0 after 2572 usecs
initcall ufs_qcom_phy_qmp_14nm_driver_init+0x0/0x24 returned 0 after 1727 usecs
initcall ufs_qcom_phy_qmp_v3_driver_init+0x0/0x24 returned 0 after 1010 usecs
initcall ufs_qcom_phy_qrbtc_v2_driver_init+0x0/0x24 returned 0 after 838 usecs
```

1.4.3 uefi log

抓取方法:

```
adb shell cat /proc/blog>E:\uefi.txt
```

亦可以使用串口线抓取。

一般情况下我们直接取到的 uefi 的 log 中已经包含了一些时间戳信息。

```
UEFI Start      [  710] SEC
Render Splash   [ 1729]
Platform Init   [ 1889] BDS
UEFI Total : 1207 ms
POST Time       [ 1917] OS Loader
Loading Image Start : 2996 ms
Loading Image Done : 2996 ms
Total Image Read size : 512 Bytes
Loading Image Start : 2996 ms
Loading Image Done : 3224 ms
Decompressing kernel image start: 8264 ms
Decompressing kernel image done: 8612 ms
Shutting Down UEFI Boot Services: 8762 ms
Exit BS         [ 9558] UEFI End
```

Build A Debug version

通过修改 boot_images/scripts/build_boot_images.sh 中的编译条件,可以得到打印信息更为丰富的 uefi log.

```
  -r --release
      Provide a release mode, one of "DEBUG" or "RELEASE". Default is to build both.
```

在 boot_images/scripts/build_boot_images.sh 中找到如下文本片段,把其中的 RELEASE 改变为 DEBUG.

```
# build xbl ------------------------------------------------------

cd $root

cd ../boot_images/QcomPkg/Sdm660Pkg
python ../buildit.py --variant LA -r DEBUG -t Sdm660Pkg,QcomToolsPkg --build_flags=
       cleanall 2>&1 | tee -a $root/out/$PROJ_NAME-0-$MAJOR_VER$MINOR_VER-build_xbl1.log
python ../buildit.py --variant LA -r DEBUG -t Sdm660Pkg,QcomToolsPkg 2>&1 | tee -a $root/
       out/$PROJ_NAME-0-$MAJOR_VER$MINOR_VER-build_xbl1.log

# copy image ------------------------------------------------------

src=$root"/QcomPkg/Sdm660Pkg/Bin/660/LA/DEBUG"
```

更改之后,编译出的 DEBUG 版本,log 会多出来一些信息,比如:

```
APC1 Total 700
- 0x09B445000 [ 1836] MdtpDxe.efi
- 0x09AC4F000 [ 1837] HashDxe.efi
- 0x09B475000 [ 1837] RngDxe.efi
- 0x09B453000 [ 1838] MpPowerDxe.efi
- 0x09AC6B000 [ 1838] ChargerExDxe.efi
- 0x09AC34000 [ 1839] UsbMsdDxe.efi
- 0x09AC28000 [ 1839] UsbDeviceDxe.efi
-------------------------------
Platform Init  [ 1891] BDS
UEFI Ver    : 4.2.180915.BOOT.XF.1.4-00258-S660LZB-1
Platform    : MTP
Chip Name   : SDM630
Chip Ver    : 1.0
Core 0 Freq: 1670400 MHz
Mounting FAT Volume: logfs
-------------------------------
```

1.4.4 Event log

从 Event log 中,我们可以找到在 boot 阶段,各阶段的用时。

通过如下方法可以获取一份 event log.

```
adb logcat -b events -d > logcat_events.txt
```

下面是 evet log 中的一些对这个问题有用的信息。我们一一列举讨论,这些技巧来自高通的一个 KBA 档[6]。

- user space 开始,在 kernel 起来后用了 7.9s。

  ```
  12-06 09:59:16.224   645   645 I boot_progress_start: 7989
  ```

  这个时间点我认为是 init.rc 脚本执行到一定程度,就像 init 脚本执行完进入命令行,足够可以开始第二阶段 Zygote。我对 init 具体行为并不了解,以下是博客上看到的:

  生成文件系统,启动 vold、media、SurfaceFlinger 等 Nativie 服务在这个阶段你可以看到带“Android”文字静态 logo 和带“android”文字的开机动画。[7]

- Zygote start.

  ```
  01-01 12:02:23.129   645   645 I boot_progress_preload_start: 10342
  ```

  源码在:frameworks/base/cmds/app_process/app_main.cpp 等

  zygote 是一个在 init.rc 中被指定启动的服务,该服务对应的命令是/system/bin/app_process [8]

  - 建立 java Runtime,建立虚拟机。
  - 建立 Socket 接收 ActivityManangerService 的请求,用于 Fork 应用程序。

[6] kba-160919232945_3_how_to_debug_boot_time_performance_issue

[7] 本句来自 qianxuedegushi 的 CSDN 博客,全文地址请点击:博客原文

[8] 参考 wangzefengw 的 CSDN 博客,全文地址请点击:博客原文

- 启动 System Server。

- Zygote end

```
01-01 12:02:25.665   645   645 I boot_progress_preload_end: 12878
```

- System ready

```
01-01 12:02:25.971  1709  1709 I boot_progress_system_run: 13184
```

SystemServer ready, 开始启动 Android 系统服务，如 PMS，APMS 等。

- package scan begin

```
01-01 12:02:26.943  1709  1709 I boot_progress_pms_start: 14156
```

PMS(PackageManagerService)开始扫描安装的应用。

PMS 用来管理所有的 package 信息，包括安装、卸载、更新以及解析 AndroidManifest.xml 以组织相应的数据结构，这些数据结构将会被 PMS、ActivityMangerService 等等 service 和 application 使用到。

- scan system folder

```
01-01 12:02:27.565  1709  1709 I boot_progress_pms_system_scan_start: 14778
```

PMS 先行扫描/system 目录下的安装包。

- scan data folder

```
01-01 12:02:32.829  1709  1709 I boot_progress_pms_data_scan_start: 20042
```

PMS 扫描/data 目录下的安装包

- scan end

```
01-01 12:02:32.842  1709  1709 I boot_progress_pms_scan_end: 20055
```

PMS 扫描结束

- PMS ready

```
01-01 12:02:33.132  1709  1709 I boot_progress_pms_ready: 20345
```

PMS 就绪

- AMS ready

```
01-01 12:02:39.605  1709  1709 I boot_progress_ams_ready: 26818
```

ActivityManagerService AMS 是系统的引导服务，应用进程的启动、切换和调度、四大组件的启动和管理都需要 AMS 的支持。[9]

[9] 本文来自刘望舒的 CSDN 博客，全文地址请点击：原文

- 系统启动完成 Home activiy already start finish and it is idle .then will trigger this event。

```
01-01 12:02:41.243  1709  1761 I boot_progress_enable_screen: 28456
```

1.5 在 uefi 中打印时间戳

通过对以上的 log 进行采集与分析比较，基本可以确定软件部分在 boot 过程中的各阶段的用时，对比哪里慢了我们就重点分析那个地方。

在这次情形下发现 uefi P 版本比 O 慢了 1.2s 左右，我们简单的对比一下时间戳有：

point	差异	PL2O	PL2P	NOTE
UEFI Start		747	757	
	458	512	970	
Render Splash		1259	1727	
	9	155	164	
Platform Init		1414	1891	
	9	33	42	
OS Loader		1445	1933	
	58	6819	6877	
UEFI Boot Services		8264	8810	
	757	65	822	
UEFI End		8329	9632	

从上面大致的对比可缩小排查的范围。我们需要按照在 uefi 执行流程，添加更为详细的时间来锁定问题。

通过添加类似的语句来不断的缩小范围，锁定问题

```
DEBUG ((EFI_D_ERROR, "\n->check_multi_header S: %lu ms\n", GetTimerCountms ()));
```

如果报编译错误，可以尝试检查是否添加：

```
QcomBaseLib.h
```

的 header file。并确定在相应的.inf 中添加：

```
DebugLib
QcomBaseLib
```

Chapter 2

附录

2.1 附录 xbl loader info

```
re Boot: Off
S - Boot Config @ 0x00786070 = 0x000001c1
S - JTAG ID @ 0x00786130 = 0x000ac0e1
S - OEM ID @ 0x00786138 = 0x00000000
S - Serial Number @ 0x00784138 = 0xcd2bcd5b
S - OEM Config Row 0 @ 0x00784188 = 0x0000000000000000
S - OEM Config Row 1 @ 0x00784190 = 0x0000000000000000
S - Feature Config Row 0 @ 0x007841a0 = 0x0070302012400300
S - Feature Config Row 1 @ 0x007841a8 = 0x0000000000000820
S - Core 0 Frequency, 3952 MHz
S - PBL Patch Ver: 4
S - I-cache: On
S - D-cache: On



B -         0 - PBL, Start
B -      7114 - bootable_media_detect_entry, Start
B -    135758 - bootable_media_detect_success, Start
B -    135764 - elf_loader_entry, Start
B -    137445 - auth_hash_seg_entry, Start
B -    137780 - auth_hash_seg_exit, Start
B -    189261 - elf_segs_hash_verify_entry, Start
B -    238916 - elf_segs_hash_verify_exit, Start
B -    238929 - auth_xbl_sec_hash_seg_entry, Start
B -    268084 - auth_xbl_sec_hash_seg_exit, Start
B -    268086 - xbl_sec_segs_hash_verify_entry, Start
B -    274847 - xbl_sec_segs_hash_verify_exit, Start
B -    274894 - PBL, End




sbl1_boot_logger_init

B -    301340 - SBL1, Start
B -    304664 - usb: hs_phy_nondrive_start
B -    305000 - usb: hs_phy_nondrive_finish
B -    305030 - boot_flash_init, Start
D -        30 - boot_flash_init, Delta
B -    305061 - sbl1_ddr_set_default_params, Start
D -       152 - sbl1_ddr_set_default_params, Delta
```

```
B -    305213 - boot_config_data_table_init, Start
B -    324825 - Using default CDT
D -     19611 - boot_config_data_table_init, Delta - (0 Bytes)
B -    324855 - CDT Version:3,Platform ID:8,Major ID:1,Minor ID:0,Subtype:0

boot_config_process_entry
B -    324855 - Image Load, Start

boot_auth_image_hashtable-> boot_sec_img_auth_verify_metadata
D -       671 - Auth Metadata
D -       458 - Segments hash check

D -      4972 - PMIC Image Loaded, Delta - (34272 Bytes)

sbl1_config_table 中有load_apdp_image_pre_procs
加载时要预先执行的函数列表
boot_procedure_func_type load_apdp_image_pre_procs[] =
{
  /* Initialize PMIC and railway driver */
  sbl1_hw_pre_ddr_init,

B -    329827 - pm_device_init, Start
B -    331596 - PM: 0x1210: 00
B -    331596 - PM: battery present
B -    332114 - PM: PON REASON: PM0=0x8000028000000001:0x0 PM1=0x8000088000000020:0x0
B -    372984 - PM: SET_VAL:Skip
D -     44072 - pm_device_init, Delta
B -    373899 - pm_driver_init, Start
B -    377620 - PM: OCP Clearing for L4A is Skipped :PM660 is not supported the LDO4
D -      4056 - pm_driver_init, Delta
B -    377956 - pm_sbl_chg_init, Start
B -    380182 - PM: Trigger FG IMA Reset
B -    380396 - PM: Trigger FG IMA Reset.Completed
B -    381677 - PM: EntryVbat: 4313; EntrySOC: -1
B -    381768 - PM: BATT TEMP: 28 DegC
D -      3995 - pm_sbl_chg_init, Delta
B -    381982 - vsense_init, Start
D -         0 - vsense_init, Delta


APDP Image Loaded
    sbl1_ddr_set_params,

B -    429623 - Pre_DDR_clock_init, Start
D -       396 - Pre_DDR_clock_init, Delta
D -         0 - sbl1_ddr_set_params, Delta

(boot_procedure_func_type)sbl1_ddr_init,
B -    435265 - DSF version = 35.0, DSF RPM version = 22.0
B -    435265 - Max Frequency = 1296 MHz,  platform_id = 0x3007
B -    435387 - do_ddr_training, Start
B -    442341 - Bootup frequency set to 1296000
D -      7045 - do_ddr_training, Delta
B -    457347 - ddr_mfr = 0xff
B -    457347 - ddr_type = 0x7
B -    457378 - ddr_size = 0x1000 MB


sbl1_hw_init_secondary,
B -    469395 - clock_init, Start
D -       305 - clock_init, Delta
}
```

```
105
106 boot_config_process_entry
107 sbl1_load_sec_partition
108
109 B -    470127 - Image Load, Start
110 D -         0 - APDP Image Loaded, Delta - (0 Bytes)
111 B -    470188 - usb: EMMC Serial - af7320
112 B -    470249 - usb: fedl, vbus_low
113 B -    472963 - PM: 0: PON=0x1:HARD_RESET: ON=0x80:PON_SEQ: POFF=0x2:PS_HOLD: OFF=0x80:
        POFF_SEQ
114 B -    473055 - PM: 1: PON=0x20:PON1: ON=0x80:PON_SEQ: POFF=0x8:GP1: OFF=0x80:POFF_SEQ
115 B -    473085 - PM: SMEM Chgr Info Write Success
116 B -    473177 - sbl1_efs_handle_cookies, Start
117 D -       579 - sbl1_efs_handle_cookies, Delta
118 B -    473848 - Image Load, Start
119 D -      2775 - Auth Metadata
120 D -      1068 - Segments hash check
121 D -      6527 - QSEE Dev Config Image Loaded, Delta - (42024 Bytes)
122 B -    480405 - Image Load, Start
123 D -       518 - SEC Image Loaded, Delta - (4096 Bytes)
124 B -    480924 - Image Load, Start
125 D -     21472 - Auth Metadata
126 D -     18270 - Segments hash check
127 D -     61762 - QSEE Image Loaded, Delta - (1945112 Bytes)
128 B -    542717 - Image Load, Start
129 D -      2776 - Auth Metadata
130 D -      3050 - Segments hash check
131 D -     10675 - QHEE Image Loaded, Delta - (273136 Bytes)
132 B -    553453 - Image Load, Start
133 D -      2867 - Auth Metadata
134 D -      2196 - Segments hash check
135 D -     11407 - RPM Image Loaded, Delta - (219340 Bytes)
136 B -    565348 - Image Load, Start
137 D -        30 - STI Image Loaded, Delta - (0 Bytes)
138 B -    565836 - Image Load, Start
139 D -      2806 - Auth Metadata
140 D -      3324 - Segments hash check
141 D -     11559 - ABL Image Loaded, Delta - (379472 Bytes)
142 B -    577456 - Image Load, Start
143 D -      3386 - Auth Metadata
144 D -     13939 - Segments hash check
145 D -     29280 - APPSBL Image Loaded, Delta - (1792000 Bytes)
146 B -    607133 - SBL1, End
147 D -    305823 - SBL1, Delta
148 S - Flash Throughput, 80000 KB/s  (4694664 Bytes,  58271 us)
149 S - DDR Frequency, 1296 MHz
```

2.2 附录关键代码

2.2.1 boot_config_process_bl

```
1 void boot_config_process_bl
2 (
3   bl_shared_data_type *bl_shared_data,
4   image_type host_img,
5   boot_configuration_table_entry * boot_config_table
6 )
7 {
```

```
  boot_configuration_table_entry *curr_entry = NULL;

  BL_VERIFY( bl_shared_data != NULL && boot_config_table != NULL,
             BL_ERR_NULL_PTR_PASSED|BL_ERROR_GROUP_BOOT);

  /* For every entry in the boot configuration table */
  for(curr_entry = boot_config_table;
      curr_entry->host_img_id != NONE_IMG;
      curr_entry++)
  {
    /* Process entries sequentially only for the specific host_img */
    if(curr_entry->host_img_id == host_img)
    {
      boot_config_process_entry(bl_shared_data,
                                curr_entry);
    }
  }

  return;
}
```

2.2.2 boot_configuration_table_entry

```
typedef struct
{
  image_type host_img_id;                /**< Image ID of the host */
  config_image_type host_img_type;       /**< Image type of the host */
  image_type target_img_id;              /**< Image ID of the target */
  config_image_type target_img_type;     /**< Image type of the target */
  secboot_sw_type target_img_sec_type;   /**< Secure Software ID of the target */
  boot_boolean load;                     /**< Target will be loaded */
  boot_boolean auth;                     /**< Target will be authenticated */
  boot_boolean exec;                     /**< Target will execute immediately after
                                              loading/authentication */
  boot_boolean jump;                     /**< Target will be jumped to after
                                              post procedures are complete */
  boot_procedure_func_type exec_func;    /**< Function pointer to execute function.
                                              Must be defined if exec is TRUE */
  boot_procedure_func_type jump_func;    /**< Function pointer to jump function.
                                              Must be defined if jump is TRUE */
  boot_procedure_func_type *pre_procs;   /**< Function pointer array to pre-procedures */
  boot_procedure_func_type *post_procs;  /**< Function pointer array to post-procedures
       */
  boot_logical_func_type boot_load_cancel;/**< Function pointer to cancel loading */
  uint8 * target_img_partition_id;       /**< Target image partition ID information */
  uint8 * target_img_str;                 /**< Target image name information  */
  whitelist_region_type * whitelist_table; /**< Whitelist table */
  boot_boolean boot_ssa_enabled;          /**< Subsystem self authentication (ssa)*/
  boot_boolean enable_rollback_protection; /**< Enable Roll back protection feature or not
       */
  boot_boolean enable_xpu;                /**< Enable XPU configuration for the image */
  uint32 xpu_proc_id;                     /**< XPU proc id to be passed to TZ */
  uint32 sbl_qsee_interface_index;       /**< Index of this image's entry in sbl qsee
       interface struct */
  uint64 seg_elf_entry_point;            /**< Entry point for segmented ELF */
  uint32 RESERVED_3;                     /**< RESERVED */
  uint32 RESERVED_4;                     /**< RESERVED */
}boot_configuration_table_entry;


```

```
35
36
37  boot_configuration_table_entry sbl1_config_table[] =
38  {
39
40    /* SBL1 -> PMIC */
41    {
42      SBL1_IMG,                    /* host_img_id */
43      CONFIG_IMG_QC,               /* host_img_type */
44      GEN_IMG,                     /* target_img_id */
45      CONFIG_IMG_ELF,              /* target_img_type */
46      SECBOOT_PMIC_SW_TYPE,        /* target_img_sec_type */
47      TRUE,                        /* load */
48      TRUE,                        /* auth */
49      FALSE,                       /* exec */
50      FALSE,                       /* jump */
51      NULL,                        /* exec_func */
52      NULL,                        /* jump_func */
53      load_pmic_pre_procs,         /* pre_procs */
54      load_pmic_post_procs,        /* post_procs */
55      pmic_load_cancel,            /* load_cancel */
56      pmic_partition_id,           /* target_img_partition_id */
57      (uint8*)PMIC_BOOT_LOG_STR,   /* target_img_str */
58      NULL,                        /* whitelist table */
59      FALSE,                       /* boot_ssa_enabled */
60      TRUE,                        /* enable_rollback_protection*/
61      FALSE,                       /* enable_xpu */
62      0x0,                         /* xpu_proc_id*/
63      0x0,                         /* sbl_qsee_interface_index */
64      0x0                          /* seg_elf_entry_point*/
65    },
66
67    /* SBL1 -> APDP */
68    {
69      SBL1_IMG,                    /* host_img_id */
70      CONFIG_IMG_QC,               /* host_img_type */
71      GEN_IMG,                     /* target_img_id */
72      CONFIG_IMG_ELF,              /* target_img_type */
73      SECBOOT_APDP_SW_TYPE,        /* target_img_sec_type */
74      TRUE,                        /* load */
75      TRUE,                        /* auth */
76      FALSE,                       /* exec */
77      FALSE,                       /* jump */
78      NULL,                        /* exec_func */
79      NULL,                        /* jump_func */
80      load_apdp_image_pre_procs,     /* pre_procs */
81      load_apdp_image_post_procs,    /* post_procs */
82      apdp_load_cancel,            /* load_cancel */
83      apdp_partition_id,           /* target_img_partition_id */
84      APDP_BOOT_LOG_STR,           /* target_img_str */
85      NULL,                        /* whitelist table */
86      FALSE,                       /* boot_ssa_enabled */
87      TRUE,                        /* enable_rollback_protection*/
88      FALSE,                       /* enable_xpu */
89      0x0,                         /* xpu_proc_id*/
90      0x0,                         /* sbl_qsee_interface_index */
91      0x0                          /* seg_elf_entry_point*/
92    },
93
94    /* SBL1 -> Qsee Dev config image  */
95    {
96      SBL1_IMG,                          /* host_img_id */
```

```
    CONFIG_IMG_QC,                      /* host_img_type */
    GEN_IMG,                            /* target_img_id */
    CONFIG_IMG_ELF,                     /* target_img_type */
    SECBOOT_QSEE_DEVCFG_SW_TYPE,        /* target_img_sec_type */
    TRUE,                               /* load */
    TRUE,                               /* auth */
    FALSE,                              /* exec */
    FALSE,                              /* jump */
    NULL,                               /* exec_func */
    NULL,                               /* jump_func */
    NULL,                               /* pre_procs */
    load_qsee_devcfg_image_post_procs,  /* post_procs */
    qsee_devcfg_image_load_cancel,      /* load_cancel */
    qsee_devcfg_image_partition_id,     /* target_img_partition_id */
    QSEE_DEVCFG_BOOT_LOG_STR,           /* target_img_str */
    NULL,                               /* whitelist table */
    FALSE,                              /* boot_ssa_enabled */
    FALSE,                              /* enable_rollback_protection*/
    FALSE,                              /* enable_xpu */
    0x0,                                /* xpu_proc_id*/
    0x0,                                /* sbl_qsee_interface_index */
    0x0                                 /* seg_elf_entry_point*/
  },

 /* SBL1 -> QSEE */
  {
    SBL1_IMG,                   /* host_img_id */
    CONFIG_IMG_QC,              /* host_img_type */
    GEN_IMG,                    /* target_img_id */
    CONFIG_IMG_ELF,             /* target_img_type */
    SECBOOT_QSEE_SW_TYPE,       /* target_img_sec_type */
    TRUE,                       /* load */
    TRUE,                       /* auth */
    FALSE,                      /* exec */
    FALSE,                      /* jump */
    NULL,                       /* exec_func */
    NULL,                       /* jump_func */
    NULL,                       /* pre_procs */
    load_qsee_image_post_procs, /* post_procs */
    NULL,                       /* load_cancel */
    qsee_partition_id,          /* target_img_partition_id */
    QSEE_BOOT_LOG_STR,          /* target_img_str */
    NULL,                       /* whitelist table */
    FALSE,                      /* boot_ssa_enabled */
    TRUE,                       /* enable_rollback_protection*/
    FALSE,                      /* enable_xpu */
    0x0,                        /* xpu_proc_id*/
    0x0,                        /* sbl_qsee_interface_index */
    0x0                         /* seg_elf_entry_point*/
  },


  /* SBL1 -> QHEE */
  {
    SBL1_IMG,                   /* host_img_id */
    CONFIG_IMG_QC,              /* host_img_type */
    GEN_IMG,                    /* target_img_id */
    CONFIG_IMG_ELF,             /* target_img_type */
    SECBOOT_QHEE_SW_TYPE,       /* target_img_sec_type */
    TRUE,                       /* load */
    TRUE,                       /* auth */
    FALSE,                      /* exec */
```

```
    FALSE,                      /* jump */
    NULL,                       /* exec_func */
    NULL,                       /* jump_func */
    NULL,                       /* pre_procs */
    NULL,                       /* post_procs */
    NULL,                       /* load_cancel */
    qhee_partition_id,          /* target_img_partition_id */
    QHEE_BOOT_LOG_STR,          /* target_img_str */
    NULL,                       /* whitelist table */
    FALSE,                      /* boot_ssa_enabled */
    TRUE,                       /* enable_rollback_protection*/
    FALSE,                      /* enable_xpu */
    0x0,                        /* xpu_proc_id*/
    0x0,                        /* sbl_qsee_interface_index */
    0x0                         /* seg_elf_entry_point*/
  },


  /* SBL1 -> RPM */
  {
    SBL1_IMG,                   /* host_img_id */
    CONFIG_IMG_QC,              /* host_img_type */
    GEN_IMG,                    /* target_img_id */
    CONFIG_IMG_ELF,             /* target_img_type */
    SECBOOT_RPM_FW_SW_TYPE,     /* target_img_sec_type */
    TRUE,                       /* load */
    TRUE,                       /* auth */
    FALSE,                      /* exec */
    FALSE,                      /* jump */
    NULL,                       /* exec_func */
    NULL,                       /* jump_func */
    NULL,                       /* pre_procs */
    NULL,                       /* post_procs */
    rpm_load_cancel,            /* load_cancel */
    rpm_partition_id,           /* target_img_partition_id */
    RPM_BOOT_LOG_STR,           /* target_img_str */
    NULL,                       /* whitelist table */
    FALSE,                      /* boot_ssa_enabled  */
    TRUE,                       /* enable_rollback_protection */
    FALSE,                      /* enable_xpu */
    0x0,                        /* xpu_proc_id*/
    0x0,                        /* sbl_qsee_interface_index */
    0x0                         /* seg_elf_entry_point*/
  },


  /* SBL1 -> STI */
  {
    SBL1_IMG,                     /* host_img_id */
    CONFIG_IMG_QC,                /* host_img_type */
    GEN_IMG,                      /* target_img_id */
    CONFIG_IMG_ELF,               /* target_img_type */
    SECBOOT_STI_SW_TYPE,          /* target_img_sec_type */
    TRUE,                         /* load */
    TRUE,                         /* auth */
    FALSE,                        /* exec */
    TRUE,                         /* jump */
    NULL,                         /* exec_func */
    sti_jump_func,                /* jump_func */
    NULL,                         /* pre_procs */
    NULL,                         /* post_procs */
    sti_load_cancel,              /* load_cancel */
```

```
    sti_partition_id,              /* target_img_partition_id */
    STI_BOOT_LOG_STR,              /* target_img_str */
    NULL,                          /* whitelist table */
    FALSE,                         /* boot_ssa_enabled  */
    TRUE,                          /* enable_rollback_protection */
    FALSE,                         /* enable_xpu */
    0x0,                           /* xpu_proc_id*/
    0x0,                           /* sbl_qsee_interface_index */
    0x0                            /* seg_elf_entry_point*/
  },


  /* SBL1 -> ABL */
  {
    SBL1_IMG,                    /* host_img_id */
    CONFIG_IMG_QC,               /* host_img_type */
    GEN_IMG,                     /* target_img_id */
    CONFIG_IMG_ELF,              /* target_img_type */
    SECBOOT_ABL_SW_TYPE,         /* target_img_sec_type */
    TRUE,                        /* load */
    TRUE,                        /* auth */
    FALSE,                       /* exec */
    FALSE,                        /* jump */
    NULL,                        /* exec_func */
    NULL,                        /* jump_func */
    NULL,                        /* pre_procs */
    NULL,                        /* post_procs */
    abl_load_cancel,             /* load_cancel */
    abl_partition_id,            /* target_img_partition_id */
    ABL_BOOT_LOG_STR,            /* target_img_str */
    NULL,                        /* whitelist table */
    FALSE,                       /* boot_ssa_enabled  */
    TRUE,                        /* enable_rollback_protection */
    FALSE,                       /* enable_xpu */
    0x0,                         /* xpu_proc_id*/
    0x0,                         /* sbl_qsee_interface_index */
    SCL_XBL_CORE_CODE_BASE       /* seg_elf_entry_point*/
  },


  /* SBL1 -> APPSBL */
  {
    SBL1_IMG,                    /* host_img_id */
    CONFIG_IMG_QC,               /* host_img_type */
    GEN_IMG,                     /* target_img_id */
    CONFIG_IMG_ELF,              /* target_img_type */
    SECBOOT_APPSBL_SW_TYPE,      /* target_img_sec_type */
    TRUE,                        /* load */
    TRUE,                        /* auth */
    FALSE,                       /* exec */
    TRUE,                        /* jump */
    NULL,                        /* exec_func */
    qsee_jump_func,              /* jump_func */
    NULL,                        /* pre_procs */
    sbl1_post_appsbl_procs,      /* post_procs */
    appsbl_load_cancel,          /* load_cancel */
    appsbl_partition_id,         /* target_img_partition_id */
    APPSBL_BOOT_LOG_STR,         /* target_img_str */
    NULL,                        /* whitelist table */
    FALSE,                       /* boot_ssa_enabled  */
    TRUE,                        /* enable_rollback_protection */
    FALSE,                       /* enable_xpu */
```

```
    0x0,                        /* xpu_proc_id*/
    0x0,                        /* sbl_qsee_interface_index */
    SCL_XBL_CORE_CODE_BASE      /* seg_elf_entry_point*/
  },


  /* LAST ENTRY */
  {
    NONE_IMG,
  }

};
```

2.2.3 BdsEntry

```
VOID
EFIAPI
BdsEntry (
  IN EFI_BDS_ARCH_PROTOCOL  *This
  )
{
  EFI_STATUS           Status;
  LIST_ENTRY           BootOrderList;
  BDS_INIT_OPTION      InitOption;
  UINT8                USBFirst = 0;
  UINTN                VarSize = sizeof(USBFirst);

  ConnectControllersDone = FALSE;

  ProcessFvLoading ();

  PERF_END   (NULL, "DXE", NULL, 0);

  Status = UpdateFWVendorInST();
  ASSERT_EFI_ERROR(Status);

  PlatformBdsInitEx(&InitOption);

  Status = gRT->GetVariable(L"AttemptUSBFirst", &gOSAVendorGuid,
                            NULL, &VarSize, &USBFirst);
  if (EFI_ERROR(Status))
    USBFirst = 0;

  if ((InitOption == BootFromRemovableMedia) || USBFirst)
    AttemptBootFromRemovable();

  AttemptBootNext();

  //Blindly copy the BootOrder variable's associated BootOption's into the List structure
  Status = BuildBootOptionListFromBootOrder(&BootOrderList);
  if (EFI_ERROR(Status))
  {
    //BootOrder variable does not exist, or GetVariable otherwise failed
    DEBUG((EFI_D_WARN, "[QcomBds] BootOrder not found\n"));
    //Enumerate all other options and try booting them
    EnumerateAndBoot(AllMediaTypes, BOOT_REMOVABLE_FIRST, HALT_ON_FAIL);
  }
  else
  {
    //BootOrder has been parsed into list, try booting these options
```

```
46      DEBUG((EFI_D_INFO, "[QcomBds] Attempting BootOrder entries\n"));
47      Status = BootFromBootOptionList(&BootOrderList);
48      if (EFI_ERROR(Status))
49      {
50        //None of the boot order variables were valid, enumerate all other options
51          DEBUG((EFI_D_WARN, "[QcomBds] BootOrder failed"));
52        EnumerateAndBoot(AllMediaTypes, BOOT_REMOVABLE_FIRST, HALT_ON_FAIL);
53      }
54    }
55    //Should halt before getting here
56    ASSERT(FALSE);
57  }
```

2.2.4 PlatformBdsInitEx

```
1   VOID
2       EFIAPI
3   PlatformBdsInitEx (BDS_INIT_OPTION *InitOption)
4   {
5       EFI_STATUS                      Status;
6   #ifndef PRE_SIL
7       EFI_QCOM_TOOLSUPPORT_PROTOCOL *ToolSupportProtocol = NULL;
8       EFI_ENCRYPTION_PROTOCOL       *Encryption = NULL;
9   #endif
10
11      DEBUG ((DEBUG_ERROR, "--------------------------------\n"));
12      DisplayBootTime("Platform Init ", "BDS", TRUE);
13
14
15      /* Handle all variable storage related things */
16      PerformVariableActions();
17
18      /* Make consoles available to Apps via ST */
19      BdsLibConnectAllConsoles ();
20
21      /* Sample and display system information (if required on LCD display screen) */
22      InitLcdDebugFlag ();
23      DisplayVersion ();
24      DisplayPlatformInfo ();
25
26    /* Mount LogFS partition if enabled in the uefi platform config */
27    MountFatPartition(L"logfs");
28
29      DEBUG ((DEBUG_ERROR, "--------------------------------\n"));
30
31      /*****************************************************************************
32       * WARNING:    START
33       *
34       * NOTE: Security Alert..!!
35       *
36       *   The code until the function SetupPlatformSecurity is called, is running
37       *   WITHOUT UEFI security enabled yet. So, in production image any Menu/Shell
38       *   or app running, before security enabled, will be a security hole.
39       *
40       *   So do NOT modify any code in this WARNING context.
41       *
42       */
43      if (!RETAIL)
44      {
45          /* Detect hotkey for development purposes
```

```
46          * If enabled in PROD image this would be a SECURITY HOLE
47          * So do not enable this */
48         PlatformBdsDetectHotKey ();
49     }
50
51     /* Load default debug application specified in config file */
52     LaunchDefaultBDSApps ();
53
54     /* Validate and take any action for the HW platform configuration */
55     ValidateHWConfig ();
56
57     /*  This is the place where UEFI Security is enabled, including UEFI Image
           Authentication.
58      *  So any image executed before this call, has to be strictly from part of an already
59      *  authenticated image/package and the app should not be another app launching
           platform
60      *  like shell (unless it also make sure the app launched is from authenticated
           package).
61      */
62     SetupPlatformSecurity ();
63     ....
```