



XR871 Memory Layout Developer Guide

Revision 1.0

Aug 9, 2017

Declaration

THIS DOCUMENTATION IS THE ORIGINAL WORK AND COPYRIGHTED PROPERTY OF XRADIO TECHNOLOGY ("XRADIO"). REPRODUCTION IN WHOLE OR IN PART MUST OBTAIN THE WRITTEN APPROVAL OF XRADIO AND GIVE CLEAR ACKNOWLEDGEMENT TO THE COPYRIGHT OWNER.

THE INFORMATION FURNISHED BY XRADIO IS BELIEVED TO BE ACCURATE AND RELIABLE. XRADIO RESERVES THE RIGHT TO MAKE CHANGES IN CIRCUIT DESIGN AND/OR SPECIFICATIONS AT ANY TIME WITHOUT NOTICE. XRADIO DOES NOT ASSUME ANY RESPONSIBILITY AND LIABILITY FOR ITS USE. NOR FOR ANY INFRINGEMENTS OF PATENTS OR OTHER RIGHTS OF THE THIRD PARTIES WHICH MAY RESULT FROM ITS USE. NO LICENSE IS GRANTED BY IMPLICATION OR OTHERWISE UNDER ANY PATENT OR PATENT RIGHTS OF XRADIO. THIS DATASHEET NEITHER STATES NOR IMPLIES WARRANTY OF ANY KIND, INCLUDING FITNESS FOR ANY PARTICULAR APPLICATION.

THIRD PARTY LICENCES MAY BE REQUIRED TO IMPLEMENT THE SOLUTION/PRODUCT. CUSTOMERS SHALL BE SOLELY RESPONSIBLE TO OBTAIN ALL APPROPRIATELY REQUIRED THIRD PARTY LICENCES. XRADIO SHALL NOT BE LIABLE FOR ANY LICENCE FEE OR ROYALTY DUE IN RESPECT OF ANY REQUIRED THIRD PARTY LICENCE. XRADIO SHALL HAVE NO WARRANTY, INDEMNITY OR OTHER OBLIGATIONS WITH RESPECT TO MATTERS COVERED UNDER ANY REQUIRED THIRD PARTY LICENCE.

Revision History

Version	Date	Summary of Changes
1.0	2017-8-9	Initial Version

Table 1-1 Revision History

Contents

Declaration.....	2
Revision History	3
Contents.....	4
Tables.....	5
Figures	6
1 概述	7
1.1 固件配置文件 image.cfg.....	7
1.2 编译配置文件 appos.ld.....	8
1.3 配置文件与 Memory 的关系.....	9
1.4 IMAGE SECTION 定义	10
2 使用指南	11
2.1 image.cfg 的使用	11
2.2 appos.ld 的使用.....	11
2.3 固件 OEM 签名	12

Tables

Table 1-1 Revision History	3
----------------------------------	---

Figures

图 1-1 image.cfg 内容示例.....	7
图 1-2 image.cfg 中 ATTR 字段定义	8
图 1-3 配置文件与 Memory 的关系	9

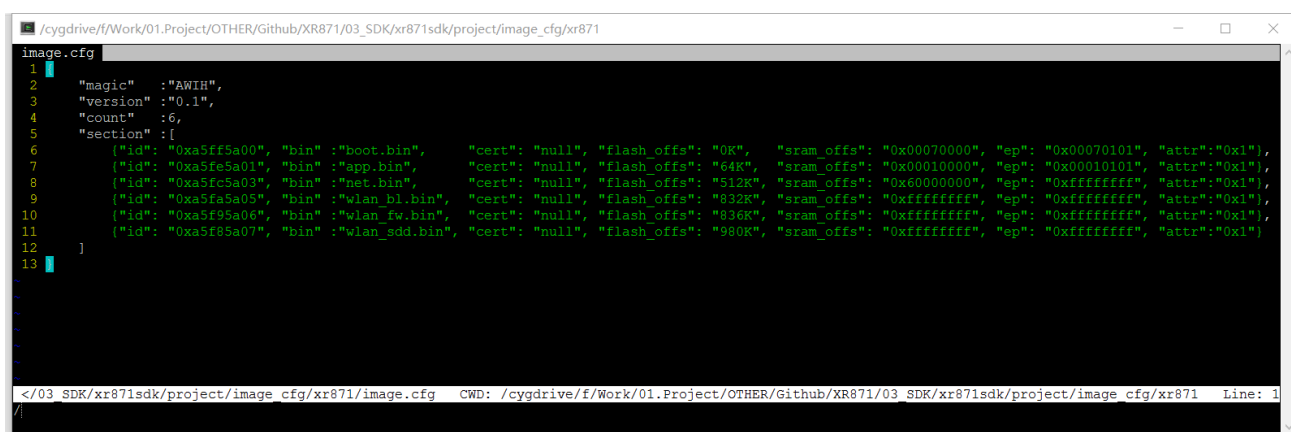
1 概述

存储空间布局（Memory Layout）设计基于该芯片上支持的空间存储，可分为三种类型：片上 ROM，SPI Nor Flash 和片上 SRAM。此文档旨在指引开发者了解布局细节，并用以指导开发。此内容涉及 2 个文档和多个工具，此处一一说明。文件包括：image.cfg 和 appos.ld。

1.1 固件配置文件 image.cfg

为了有效的组织各固件和预置文件的相互关系，并将各固件打包给工具使用，此处提供一固件配置文件 image_XXX.cfg，image.cfg 中列出了系统中存在的文件及配置内容项，在各子系统 FW 编译完成之后，打包工具 mkimage 根据配置依次处理各子系统固件，并根据配置文件将其打包在一起。打包配置文件采用 JSON 格式，字段如下，打包程序根据该配置打包每一个镜像，并建立索引表，最终将所有固件生成一个完整的 xxx_system.img 文件。

image.cfg 可参考 sdk-code/project/image_cfg/xr871/xxx.cfg 等文件，其内容表现形式如下：



```

1  image.cfg
2  [
3    "magic" : "AWIH",
4    "version" : "0.1",
5    "count" : 6,
6    "section" : [
7      {
8        "id": "0xa5ff5a00", "bin": "boot.bin", "cert": "null", "flash_offs": "0K", "sram_offs": "0x00070000", "ep": "0x00070101", "attr": "0x1"},
9        "id": "0xa5fe5a01", "bin": "app.bin", "cert": "null", "flash_offs": "64K", "sram_offs": "0x00010000", "ep": "0x00010101", "attr": "0x1"},
10       "id": "0xa5fc5a03", "bin": "net.bin", "cert": "null", "flash_offs": "512K", "sram_offs": "0x60000000", "ep": "0xffffffff", "attr": "0x1"},
11       "id": "0xa5fa5a05", "bin": "wlan_bl.bin", "cert": "null", "flash_offs": "832K", "sram_offs": "0xffffffff", "ep": "0xffffffff", "attr": "0x1"},
12       "id": "0xa5f95a06", "bin": "wlan_fw.bin", "cert": "null", "flash_offs": "836K", "sram_offs": "0xffffffff", "ep": "0xffffffff", "attr": "0x1"},
13       "id": "0xa5f85a07", "bin": "wlan_sdd.bin", "cert": "null", "flash_offs": "980K", "sram_offs": "0xffffffff", "ep": "0xffffffff", "attr": "0x1"}
14    ]
15  ]

```

图 1-1 image.cfg 内容示例

各字段含义如下：

- magic* - 固件魔数，表头标识，固定为AWIH
- version* - 版本号
- count* - 打包文件列表中文件的数目
- section* - 文件列表
 - id* - 该段文件 id，id 定义在 sdk-code/include/sys/image.h 中定义
 - bin* - 该段固件名称，表示当前 bin 文件所代表的固件，该字段仅做打包参考使用

- cert** - 该固件 OEM 签名证书，如果没有填 null，该字段仅做打包参考使用
- flash_offs** - 该段固件存放在 FLASH 中的位置偏移，以 KB 为单位，**注意各 bin 文件的空间位置不要重叠（起始+大小）**
- sram_offs** - 该段固件加载到 SRAM 中的地址偏移，0xFFFFFFFF 为无效值，使用时忽略
- ep** - 该段固件的 ENTRY POINT，0xFFFFFFFF 为无效值，使用时忽略
- attr** - 表示该段固件属性，定义如下，**属性必须严格定义**

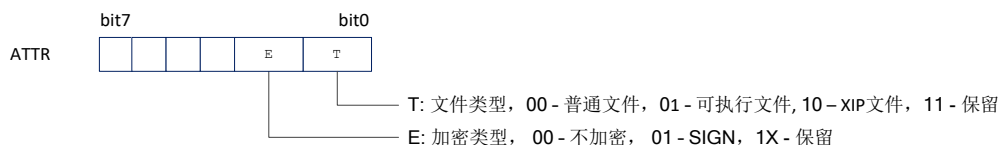


图 1-2 image.cfg 中 ATTR 字段定义

注：此文件是组织各固件文件的唯一参考，不需要其他位置设定，因此可以根据各自的需求和实际情况进行调整，如修改 FLASH 的偏移地址或增加一个文件打包烧录，但如果涉及到新加文件，则需要在 image.h 中定义对应 ID。

1.2 编译配置文件 appos.ld

image.cfg 给出了固件组织是条件参考，那么固件在内存中的排列则有 gcc 链接脚本来指定，在工程中该文件是指名为 appos_xxx.ld 的文件定义，可参考 sdk-code/project/linker_script/gcc/xr871/xxx.ld。在链接脚本中定义了 MEMORY 的组成，系统的 ENTRY POINT 以及 SECTIONS 的组成，其中 SECTIONS 对 .text, .data, .bss 等段进行了约束定义，以满足开发者将不同的执行代码和数据排布在不同的区间位置上。其内容表现如下所示：

```
MEMORY
{
    RAM (rwx) : ORIGIN = 0x00010000, LENGTH = 448K
}

__RAM_BASE = ORIGIN(RAM);
ENTRY(Reset_Handler)

SECTIONS
{
    .text :
    {
        __text_start__ = .;
        KEEP(*(.isr_vector))
        ...
    } > RAM

    .data :
    {
        ...
    } > RAM

    .bss :
    {
        ...
    } > RAM

    .heap (COPY) :
    {
```



```

...
} > RAM

.stack_dummy (COPY):
{
    *(.stack*)
} > RAM
}

```

如果系统使用了 XIP (eXecute In Place) 功能机制, 则需要使用 `appos_xip.ld` 的定义, 在其中对 XIP 使用的 MEMORY 地址空间和 SECTION 进行详细定义, 后续章节将详细说明。

1.3 配置文件与 Memory 的关系

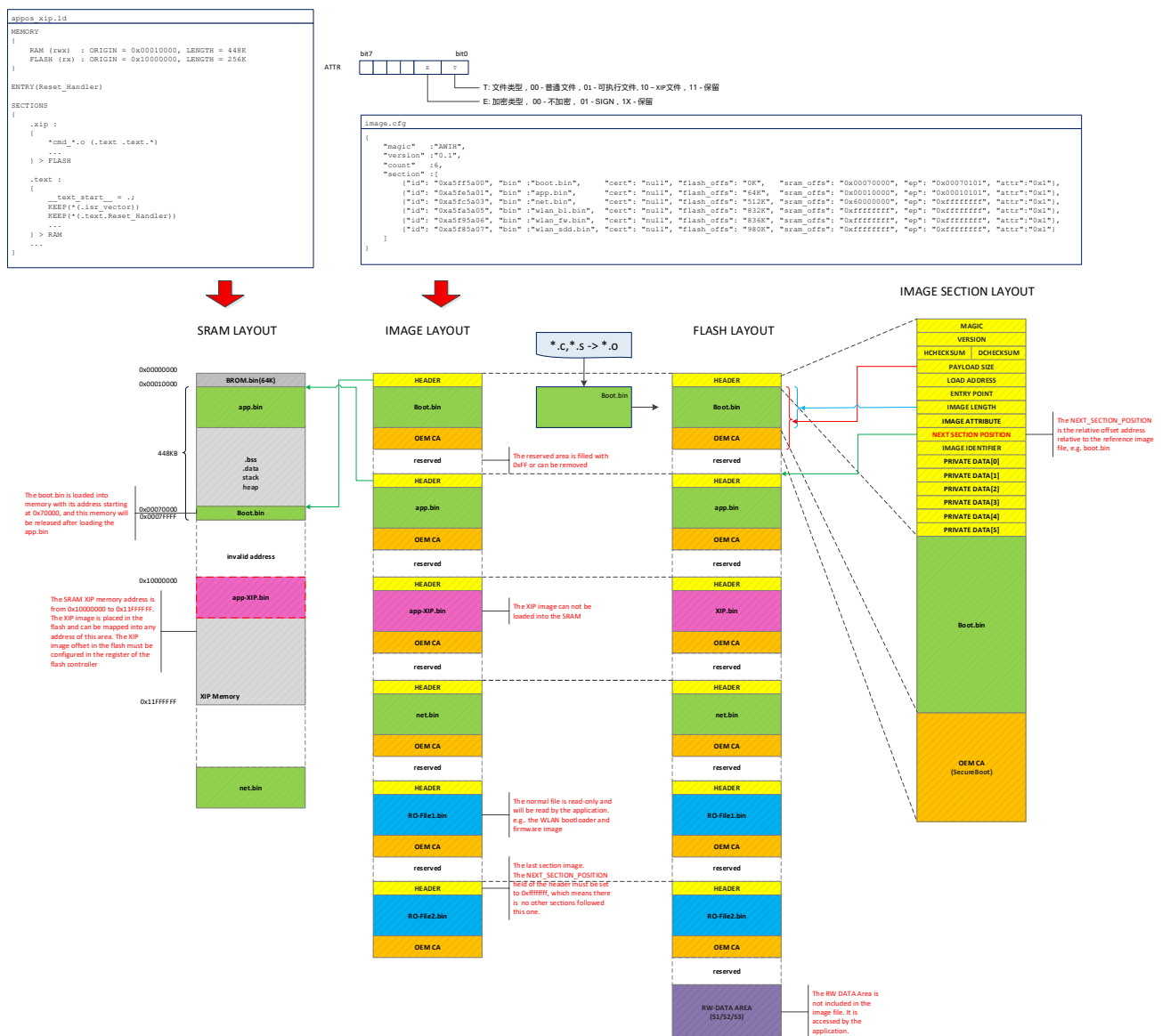


图 1-3 配置文件与 Memory 的关系

1.4 IMAGE SECTION 定义

当编译完成后，工具链生成应用对应的程序二进制文件，如 **app.bin**，之后通过打包工具 **mkimage** 将 **app.bin** 与其他 **bin** 文件一起进行打包，为了完成固件包内部索引，打包时为每一个文件增加一个头部信息，从而生成 **IMAGE SECTION** 段内容，该头部长为 64 字节，其内容如关系图中的 **IMAGE SECTION LAYOUT** 定义，内容如下：

```
typedef struct section_header {
    uint32_t magic_number;      /* magic number          */
    uint32_t version;          /* version: 0.0.0.0      */
    uint16_t header_chksum;    /* header checksum       */
    uint16_t data_chksum;      /* data checksum         */
    uint32_t data_size;        /* data size             */
    uint32_t load_addr;        /* load address          */
    uint32_t entry;            /* entry point           */
    uint32_t body_len;         /* body length           */
    uint32_t attribute;        /* attribute              */
    uint32_t next_addr;        /* next section address  */
    uint32_t id;               /* section ID            */
    uint32_t priv[6];          /* private data          */
} section_header_t;
```

各字段含义如下：

- magic** - 固件魔数，表头标识，固定为 **AWIH**，与 **image.cfg** 一致
- version** - 版本号，与 **image.cfg** 一致
- header_chksum** - 头部的累加和，用于简单校验
- data_chksum** - 数据体部分（不包含头部，含证书）的累加和，用于简单校验
- data_size** - 数据体部分的总长度（不包含头部，含证书）
- load_addr** - 该文件被 **load** 到内存的位置信息，对应到 **image.cfg** 中的 **sram_offs** 字段
- entry** - 如果该文件为某系统的可执行文件，该字段指明其 **ENTRY POINT** 的地址
- body_len** - 该段 **bin** 文件正体的长度，即 **app.bin** 或其他文件本身的长度
- attribute** - 该段文件属性，与 **image.cfg** 一致
- next_addr** - 下一个有效段的偏移地址，分为两种情况：
 - 非 OTA 类固件 - 指的是下一个段相对于 **boot.bin** 起始的偏移，即 0 地址的偏移，等于绝对地址
 - OTA 类固件 - 第一备份区和非 OTA 类固件相同，第二备份区指的是相对于 **boot.cfg** 区域的偏移
- id** - 该段文件的编号标识，在 **image.h** 中定义

除了头部之外，整个 **IMAGE SECTION** 还有文件本身和证书组成，说明如下：

- 文件正体** - 即 **app.bin**，**net.bin** 等文件本身
- 证书** - 通过 **openssl** 工具对 **app.bin** 等文件正体根据指定的 **openssl.conf** 配置内容和私钥信息进行签名得到的证书文件

2 使用指南

2.1 image.cfg 的使用

- 使用默认的 image.cfg

在 `sdk-code/project/xxx/gcc/Makefile` 中如果未明确指定 `IMAGE_CFG` 则使用默认的文件，默认文件由 `sdk-code/project/project.mk` 中指定，如下：

```
IMAGE_CFG_PATH ?= ../$(ROOT_PATH)/project/image_cfg/$(CONFIG_CHIP_NAME)
IMAGE_CFG ?= $(IMAGE_CFG_PATH)/image$(SUFFIX_WLAN)$(SUFFIX_XIP)$(SUFFIX_OTA).cfg
```

其中关联 3 个变量定义，相关内容如下：

```
ifeq ($(_PRJ_CONFIG_WLAN_STA_AP), y)
    SUFFIX_WLAN := _sta_ap
endif
ifeq ($(_PRJ_CONFIG_XIP), y)
    SUFFIX_XIP := _xip
endif
ifeq ($(_PRJ_CONFIG_OTA), y)
    SUFFIX_OTA := _ota
endif
```

其中 `_PRJ_CONFIG_WLAN_STA_AP` 等配置选项在 `sdk-code/project/xxx/gcc/localconfig.mk` 中进行设定，由此可见，根据不同的配置将指定不同的 `cfg` 文件，在 SDK 中给出了几个参考，文件如下。

```
image.cfg
image_ota.cfg
image_sta_ap.cfg
image_xip.cfg
image_xip_ota.cfg
```

- 使用指定的 image.cfg

使用指定的 `image.cfg` 只需要在 `sdk-code/project/xxx/gcc/Makefile` 中将 `IMAGE_CFG` 指定到配置文件的绝对路径即可，并按照 `image.cfg` 的说明描述进行对应的修改即可，如下：

```
IMAGE_CFG := ./image_xxx.cfg
```

2.2 appos.ld 的使用

- 不使用 FLASH XIP

在此系统上有 448KB 的 SRAM 空间，如果应用程序 Code + Data 不超过此范围则可以直接在 SRAM 中运行系统，不需要运行 FLASH XIP 机制，此时只需要在对应的 `localconfig.mk` 中关闭 XIP 选项即可（不配置或者配置成 n），在这种情况下，系统将使用默认的 `appos.ld` 作为链接的配置文件，如下：

```
_PRJ_CONFIG_XIP := n
```

- 使用 FLASH XIP

当应用系统的 Code + Data 的大小超过 448KB 的时候，SRAM 无法装载所有程序，此时需要把部分 Code 放到 FLASH 并开启 XIP 模式实现 SRAM 的 Code 段的扩展，在此情况下需要在对应的 `localconfig.mk` 中开启 XIP 配置，如下：

```
__PRJ_CONFIG_XIP := y
```

如果在 **Makefile** 中未明确指定所使用的 **ld** 文件，那么编译系统将使用默认的文件 **sdk-code/project/linker_script/gcc/xr871/appos_xip.ld**，同时，开发者需要将想要放入到 **XIP** 映射区域的代码明确的在 **appos_xip.ld** 中指明，示例如下：

```
SECTIONS
{
    .xip :
    {
        *libairkiss.a: (.text .text.*) /* 将 airkiss 库的代码段放在 XIP 区域 */
        *cmd_*.o (.text .text.*)      /* 将以 cmd_ 为前缀的 c 目标文件的代码段放在 XIP 区域 */
        *main.o (.text .text.*)       /* 将 main 函数的目标文件的代码段放在 XIP 区域 */
    } > FLASH
    ...
}
```

注意：*XIP 中只能放置与硬件无直接关系的行为代码，通常是上层应用程序。*

2.3 固件 OEM 签名

（待定）