Application Note

ZHCAEU7 – 12. 2024

xWRLx432 内存分配和使用指南

Allen Yin *Central FAE*

摘要

目前毫米波雷达在自动驾驶，智慧家庭，智能家居等领域得到了广泛的应用，在很多用户场景下，用户希望使用低功耗的毫米波雷达芯片来满足降低整体功耗或电池供电的需求。TI 推出了 AWRL6432/1432 （汽车级）与 IWRL6432/1432（工业级）系列低功耗毫米波雷达芯片，可以满足用户降低系统功耗的需求。如何有效使用片内内存对于产品性能和最终产品功耗非常重要，这篇文章介绍当使用 RBL(Rom Bootloader)和 SBL(Secondary Bootloader)时 xWRL6432/1432 毫米波雷达芯片的内存分配和使用的注意事项。

修改记录

Version	Date	Author	Notes
1.0	Sep 19th 2024	Allen Yin	First version

目录

图

表

1. xWRL6432/1432 介绍

随着汽车，工业及家庭智能化程度的提高，毫米波雷达传感器逐渐发展成一种成熟的传感方式，毫米波雷达技术有助于实现需要具备远距离、适应多种环境和更高分辨率的产品。为了满足成本和功率受限型汽车和工业应用的需求，当前的 60/77GHz 雷达传感器需要采用全新设计架构。TI 推出了 xWRL1432（76-81GHz）和 xWRL6432（57 – 64GHz） 系列芯片，其内置高级电源管理功能，可自动快速切换不同的工作状态，将平均功耗从高于 1W 的典型值降低至低于毫瓦级别，能够让高性能雷达传感应用于电池供电的应用和供电功率有限的应用。

xWRLx432 系列芯片作为高集成度的毫米波雷达处理器，由以下几个模块组成，

- 毫米波射频部分主要为射频收发和采样，包含接收低噪声放大器（LNA）,发射放大器（PA），混频器，模数转换（ADC）, 本振发生器(Synthesizer), 锁相环（APLL）等；
- 射频前端控制器，射频控制器的固件运行于内置的 M3 处理器，用于控制射频前端的参数配置，校准及监控, 并对采样信号进行抽取滤波；
- 运行于 160Mhz 的 Cortex M4F （APPSS）是由用户编程，配合使用 80Mhz 的硬件加速器（HWASS）共同完成雷达数字信号的处理及应用层软件集成；
- 片内集成了 1MB 的内部 RAM 用于代码执行和数据交互。
- 时钟与电源管理模块（PRCM），管理芯片各部分的电源和时钟输入以达到功耗控制需求。

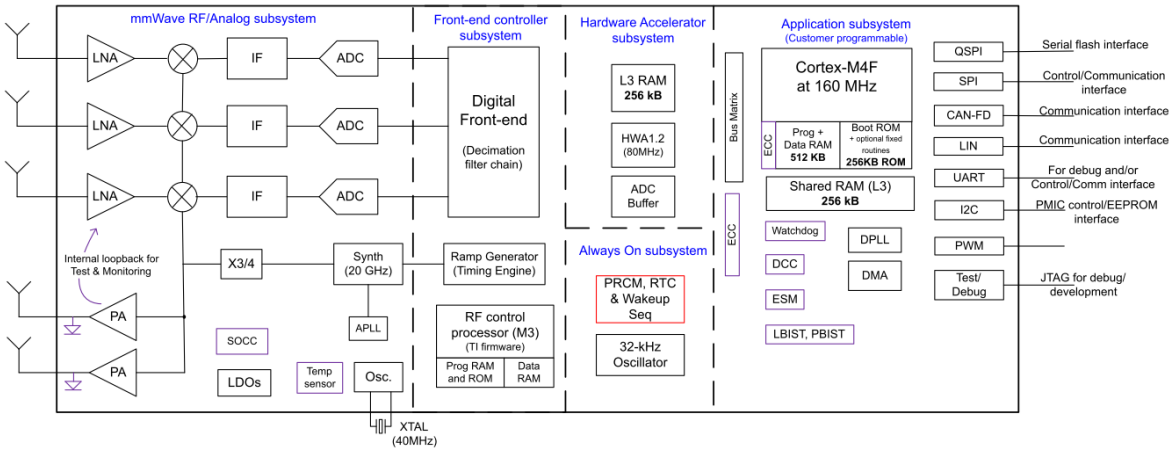


图 1. xWRLx432 芯片系统框图

2. 配置 xWRLx432 内存空间

xWRLx432 的片内 1MB 内存分成三个部分，

- APPSS RAM （512KB）主要被 M4F 使用，由三个物理 Bank 组成（256KB, 128KB, 128KB），用于用户程序段和数据段的存放，当 M4F 访问时起始地址为 0x400000，HWASS 访问起始地址为 0x22400000；
- Radar cube RAM （256 KB）供 HWASS 使用，用于加速器存放及处理雷达数据，APPSS 也可以读写这块地址空间，访问的起始地址是 0x60000000；
- Shared RAM （256KB）可以灵活的分配给 HWASS 或 M4F 访问，它由两个物理 Bank 组成（128KB, 128KB），分配给 HWA 或 M4F 的最小单位是 Bank，在芯片上电启动后，用户可以通过配置 TOP RCM 控制器的寄存器 HWA_PD_MEM_SHARE_REG ，以及 APP CTRL 控制器的寄存器 APPSS_SHARED_MEM_CLK_GATE 进行控制，寄存器说明见参考文献 4 xWRLx432 Technical Reference Manual， 用户可以根据需要随时切换 Shared RAM 的分配，但应注意 Shared RAM 数据在切换时会失效。

综上所述，xWRLx432 片内内存的地址划分可以使用如下三种方式

使用模式	APPSS 空间	HWASS 空间
模式 1	0x0040 0000-0x004B FFFF (768KB)	0x6000 0000-0x6003 FFFF (256KB)
模式 2	0x0040 0000-0x0049 FFFF (640KB)	0x6000 0000-0x6005 FFFF (384KB)
模式 3	0x0040 0000-0x0047 FFFF (512KB)	0x6000 0000-0x6007 FFFF (512KB)

表 1. xWRLx432 内存使用模式

同时，为了支持低功耗使用，xWRLx432 将片内的内存划分成不同的 Cluster，对这些 Cluster 可以通过 TOP RCM 中对应寄存器设置在对应的 APPSS 或 HWASS 电源域激活时内存簇是否需要上电提供给用户访问，或在休眠时是否需要保存该内存簇里的数据，内存簇的划分如下表所示。 通过内存簇的激活和休眠机制，合理的使用内存，减少内存的激活，可以有效的控制功耗。

RAM_1			RAM_2		RAM_3	Shared	HWA
256KB			128KB		128KB	256KB	256KB
BANK #1(1)			BANK #2		BANK #3		
Cluster #1	Cluster #3	Cluster #4	Cluster #2	Cluster #5		Cluster #6	
64kB	64KB	128KB	16KB	112KB	128KB	256KB	256KB

表 2. 低功耗内存簇划分

2.1 分配共享内存

在芯片上电启动后，片内的 RBL (Rom Bootloader) 将会自动从 Flash 加载用户程序 Boot Image 初始化芯片，此时 Shared RAM 可以通过读取启动时载入的 Boot Image 的 Image Header 设置分配到 APPSS 或者 HWASS。 Boot Image 通过 TI 提供的编译工具在工程编译时自动生成，其格式 为

Multicore Image Header
Alignment Bytes
APPSS Image in RPRC Format
Alignment Bytes
FECSS Image in RPRC Format
Alignment Bytes

表 3. 启动表头信息

在 Image header 的表项中存在 Shared Memory Allocation 字段（详见参考文献 4 的 4.7 章）决定是否需要分配 Shared RAM 到 APPSS。在项目工程里，通过修改 makefile 中的 SH_MEM_CONFIG 参数配置即可，0x0 对应上述模式 3，0x1 对应模式 2，0x3 对应模式 1，不建议使用 0x2 进行配置。

需要注意当 Shared RAM 分配到 HWASS 时，APPSS 可以读写 0x60000000 的 Radar cube RAM 地址空间（包括 Shared RAM 空间），但此时访问 HWASS 空间内的 Shared RAM 的延迟和速率要比直接分配给 APPSS 要慢很多，因此，用户应该视使用情况来分配 Shared RAM，APPSS 需大量访问 HWASS 内存空间时，应考虑使用 EDMA 加速数据的拷贝。

2.2 链接文件配置说明

通过配置用户工程里的链接文件(.cmd)配置，可以将用户的数据段和代码段指定到对应的内存区间，方法如下。

首先设定不同的内存区间，用户可以根据自己的需要配置不同的区间，注意内存区间之间不能有重叠，否则链接的时候会报错。

```
MEMORY
{
    M4F_VECS :      ORIGIN = 0x00400000 , LENGTH = 0x00000200
    M4F_RAM  :      ORIGIN = 0x00400200 , LENGTH = 0x00040000 - 0x00000200
}
```

然后将不同的段（section）指定到 RAM 区间，通常使用 8 字节的对齐。

```
SECTIONS
{
    .vectors:{} palign(8) > M4F_VECS
    .bss:   {} palign(8) > M4F_RAM
    .text:  {} palign(8) > M4F_SHA_RAM
    .data:  {} palign(8) > M4F_RAM
    .rodata: {} palign(8) > M4F_SHA_RAM
    .sysmem: {} palign(8) > M4F_RAM
    .stack:  {} palign(8) > M4F_RAM
    .jumptoApp: {} palign(8) > M4F_RAM
    .jumptoAppdata: {} palign(8) > M4F_RAM
}
```

常用段名称的说明如下表，

vectors	中断向量表地址，改变中断向量表地址以后，必须修改工程的 makefile 文件内的 BOOT_VECTOR_ADDRESS 到修改后的地址
bss	未初始化的全局变量
text	用户代码段
data	已初始化的全局变量以及静态变量
rodata	常数段 （const）
sysmem	堆空间 （malloc 开辟的内存）
stack	系统栈空间
jumptoApp jumptoAppdata	SBL 配置 Shared RAM 的代码及参数，必须放在 APPSS RAM 内

表 4. 内存段名称

在分配空间时，用户可以使用如下方式将段指定到不连续的内存空间上，

```
.text:  {} align(8) >> M4F_RAM12 | M4F_RAM3
```

用户亦可以使用自己的段名将代码或者数据放置到指定的内存地址上，流程如下，

1. 设定内存区间
   M4F_USER_RAM : ORIGIN = 0x00460000 , LENGTH = 0x00010000
2. 指定用户内存段到该区间
   .usercode: {} palign(8) > M4F_USER_RAM
   .userdata: {} palign(8) > M4F_USER_RAM

3. 在 C 代码里，使用 __attribute__((section(""))) 将代码或者数据放在特定的段中，如

```
__attribute__((section(".userdata"))) uint8_t  test[ARRAY_SIZE];
__attribute__((section(".usercode"),  myfunction))
```

3. 使用 RBL 和 SBL 启动

在一些场景下，用户需要能够支持 OTA 升级功能或者在启动后选择不同的程序运行，则需要通过芯片内置 RBL （ROM Boot Loader） 加载用户 SBL （Secondary Boot Loader ），SBL 管理和加载对应的用户程序。SBL 也可以通过串口或其它接口接收 Boot Image 来更新和运行 Flash 中的用户程序。

TI SDK 提供的 SBL Demo 程序默认 Flash 的划分如下，P1 存放 SBL，P2 存放用户升级后的应用程序，在 P4 里可以写入出厂默认程序，SBL 从 P2 读取应用程序进行启动，如果 P2 读取失败或者校验不正确，则加载 P4 的出厂应用程序进行启动。用户也可以修改 SBL 代码满足不同的启动运行需求。

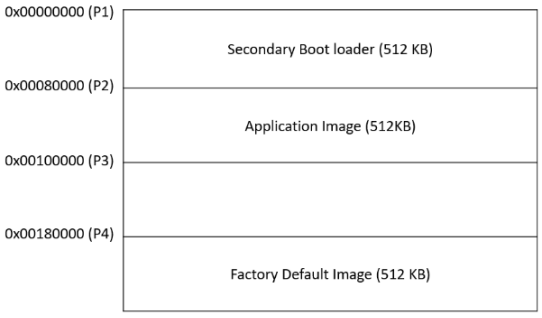


表 5. SBL Flash 分区

仅使用 RBL 直接启动用户程序时，需注意 APPSS RAM 的 0x00458000 到 0x00460000 的 32KB 地址将会被 RBL 使用作为数据段地址，因此用户程序的代码段和数据段等在加载需要写入内存里的段不能用这段 32KB 地址，这段 32KB 地址可以分配给 stack 或 heap 等运行用户程序后才会使用到的动态数据段；使用 RBL 启动 SBL，SBL **不能使用此** 32KB 地址，而用户程序也不能使用这段内存作为程序段或者静态数据段。

在使用 SBL 进行二级启动时，要特别注意 SBL 和用户程序的内存分配并合理规划，防止 SBL 和用户程序使用的内存段冲突，否则可能会造成 SBL 运行错误或者用户程序无法正常启动。同时，考虑 xWRLx432 的 Shared RAM 不能使用 8bit 或者 16 bit 的写指令的特性，将用户程序的代码段和只读数据段放入 Shared RAM 中是更好的选择，那么 SBL 的代码和数据段都不能放在 Shared RAM 防止在 SBL 加载用户代码时被异常改写而不能正常启动。

3.1 使用 SBL 启动示例

当前 SDK 的示例程序都是默认从 RBL 启动，下面以自带的 mmw demo 和 SBL 为例解释如何修改内存分配以达到 SBL 启动应用程序，并将应用程序代码段及只读数据段指定到 Shared RAM 的操作。

首先需参考 SDK 的用户手册通过 CCS 运行 mmw demo 得到当前运行的射频配置时使用 L3 内存的大小，注意在配置文件内关闭低功耗模式 CCS 内才能有信息输出。

```
[Cortex_M4_0] ========== Memory Stats ==========
                        Size       Used       Free
              L3       425984     122000     303984
              Local     28672       1472      27200
```

在不同的波形配置下 L3 使用大小不同， 在示例配置下，L3 仅使用了 122KB，因此可以使用表 1 中的模式 1 进行配置，释放更多的内存给 APPSS 使用。

因为 SDK 使用的 SBL 程序仅为 50KB 左右，可以将其全部放在 APPSS RAM 的 BANK3 (BANK1 或者 BANK2 亦可)，同时在用户程序中指定这段内存作为程序启动后使用段 （如 stack, heap…等等），具体的使用如下表，用户可根据自己的需求进行更细致的划分。

内存区间	SBL	Application
0x400000 - 0x458000	Not used	Code and data
0x458000 – 0x460000	Not used	Stack, heap and scratch buffer
0x460000 - 0x480000	Code and data	Stack, heap and scratch buffer
0x480000 - 0x4C0000	Not used	Code and data (read only)

表 6. 内存使用示例

对于 Shared RAM 的配置，SBL 和用户程序的片内内存划分遵循相同的划分方式可以避免后期不必要的内存访问冲突，即在 SBL 和用户程序的 makefile 里设置相同的 SH_MEM_CONFIG 参数。

具体的修改如下，

- SBL 工程

makefile_ccs_bootimage_gen 文件，指定中断向量表到 APPSS RAM 的 BANK 3 中。

```
BOOT_VECTOR_ADDRESS=0x460000
SH_MEM_CONFIG=0x3
```

linker.cmd 文件，把所有的 SBL 代码和数据指定到 APPSS RAM 的 BANK 3 中。

```
SECTIONS
{
   /* This has the M4F entry point and vector table, this MUST be same as BOOT_VECTOR_ADDRESS */
   .vectors:{} palign(8) > M4F_VECS
   .bss:   {} palign(8) > M4F_RAM3
   RUN_START(__BSS_START)
   RUN_END(__BSS_END)
   .text:  {} palign(8) > M4F_RAM3
   .data:  {} palign(8) > M4F_RAM3
   .rodata: {} palign(8) > M4F_RAM3
   .sysmem: {} palign(8) > M4F_RAM3
   .stack:  {} palign(8) > M4F_RAM3

   .jumptoApp: {} palign(8) > M4F_RAM3
   .jumptoAppdata: {} palign(8) > M4F_RAM3
}

MEMORY
{
   M4F_VECS :     ORIGIN = 0x00460000 , LENGTH = 0x00000200
   M4F_RAM12  :  ORIGIN = 0x00400000 , LENGTH = (0x00058000)
   M4F_RBL   :    ORIGIN = 0x00458000 , LENGTH = 0x8000              /* 32KB of RAM2 is being used by RBL */
   M4F_RAM3  :   ORIGIN = 0x00460200 , LENGTH = 0x00020000-0x200
}
```

- mmw demo 工程

makefile_ccs_bootimage_gen 文件， 指定中断向量表到 Shared RAM 中。

```
BOOT_VECTOR_ADDRESS=0x480000
SH_MEM_CONFIG=0x3
```

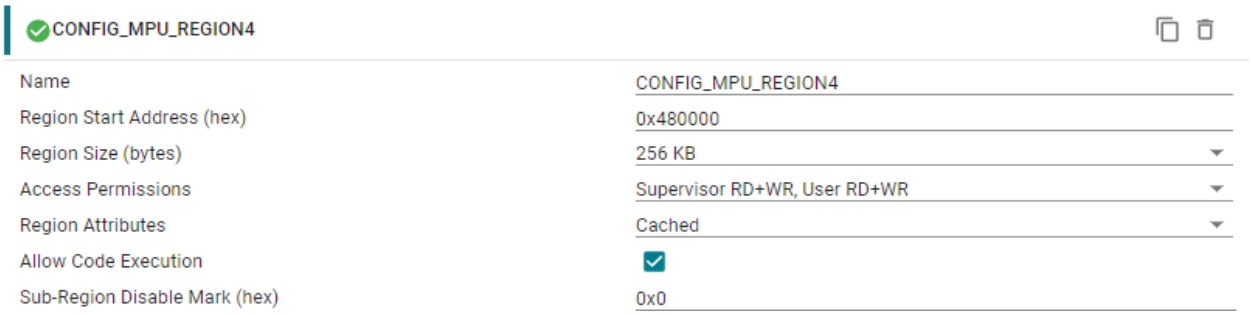
linker.cmd 文件，将代码及只读的数据段放在 Shared RAM 中，可读写的数据段指定到未被 SBL 使用的 APPSS RAM，stack 及 heap 等运行时使用的段可以覆盖 SBL 使用的内存段，对于 SBL 启动场景，RBL 使用的 32KB 内存段可以被应用程序作为动态的数据段使用。

```
SECTIONS
{
   /* This has the M4F entry point and vector table*/
   .vectors:{} palign(8) > M4F_VECS
   .bss:   {} palign(8) > M4F_RAM12
   RUN_START(__BSS_START)
   RUN_END(__BSS_END)
   .text:  {} align(8) >> M4F_SHARE_MEM| M4F_RAM12
   .data:  {} align(8) >> M4F_RAM12
   .rodata: {} align(8) >> M4F_SHARE_MEM | M4F_RAM12
   .sysmem: {} palign(8) > M4F_RBL | M4F_RAM3
   .stack:  {} palign(8) > M4F_RBL | M4F_RAM3
   .l3:    {} palign(8) > HWASS_SHM_MEM
}

MEMORY
{
   M4F_VECS :          ORIGIN = 0x00480000 , LENGTH = 0x00000200
   M4F_RAM12 :         ORIGIN = 0x00400000 , LENGTH = (0x00058000)
   M4F_RBL   :         ORIGIN = 0x00458000 , LENGTH = 0x8000        /* 32KB of RAM2 is being used by RBL */
   M4F_RAM3 :          ORIGIN = 0x00460000 , LENGTH = 0x00020000
   M4F_SHARE_MEM: ORIGIN = 0x00480200 , LENGTH = 0x00040000-0x200

   HWASS_SHM_MEM : ORIGIN = 0x60000000, LENGTH = 0x00040000 /*96KB in FECSS PD and 160KB in HWA PD */
}
```

修改 mmw_demo.c 及 dpc.c 里 L3 buffer 的大小与 linker.cmd 里的定义匹配。

```
#define L3_MEM_SIZE (0x40000)
```

example.syscfg 里增加 Shared RAM 空间的定义



mmwave_demo.c 文件里去除对 Shared RAM 段的初始化，因为其已经在 SBL 初始化，此时重复初始化会改写其中已经写入的代码段与数据段；

```
SOC_memoryInit(SOC_RCM_MEMINIT_HWA_SHRAM_INIT|SOC_RCM_MEMINIT_TPCCA_INIT|SOC_RCM_MEMINIT_TPCCB_INIT|SOC_RCM_MEMINIT_FECSS_SHRAM_INIT);
```

至此全部修改完毕，重新编译 SBL 和 mmw demo 工程即可。

4. 总结

本文详细介绍了在基于 xWRLx432 开发的低功耗毫米波雷达的产品开发中，如何根据产品的需求灵活的分配和使用片内内存，以达到充分利用芯片性能的目标，并通过示例给用户演示了这些方法。

参考文献

1. AWRL1432 Single-Chip 76- to 81-GHz Automotive Radar Sensor datasheet (Rev. B)
2. IWRL6432 Single-Chip 57- to 64GHz Industrial Radar Sensor datasheet (Rev. A)
3. IWRL6432 Device Errata (Rev. A)
4. AWRL6432, IWRL6432, AWRL1432, IWRL1432 Technical Reference Manual (Rev. A)
5. MMWAVE-L-SDK

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265